Э.Майника

АЛГОРИТМЫ ОПТИМИЗАЦИИ НА СЕТЯХ И ГРАФАХ

М.: Мир, 1981, 324 стр.

Предметный указатель

Предисловие редактора перевода

Сетевые и графовые модели охватывают довольно широкий класс задач, встречающихся при проектировании систем, планировании работ, распределении продукции, организации транспортных перевозок, размещении различных центров обслуживания населения и т. п. Во многих практически интересных случаях эти задачи характеризуются линейной целевой функцией и линейными ограничениями, так что для их решения, вообще говоря, могли бы успешно применяться известные методы линейного программирования. Однако характерной особенностью таких задач (если только они правильно отображают реальную ситуацию) является большая размерность, обусловливающая необходимость поиска более эффективных алгоритмов оптимизации, которые позволяли бы экономить вычислительные ресурсы конкретных систем и обеспечивать их гибкость по отношению к изменениям исходных данных. Плодотворной основой для построения таких алгоритмов могут служить их представления на сетях и графах.

С этой точки зрения предлагаемая вниманию советского читателя книга Майники интересна прежде всего тем, что содержит инженерное изложение основных вопросов теории графов. При этом уровень формализации задач выбран таким, чтобы книга была доступна специалистам с самой различной математической подготовкой: для ее чтения не требуется обращаться к каким-либо фундаментальным работам по теории графов, так как все основные понятия и определения вводятся по ходу изложения. Называя это изложение «интуитивным», автор подчеркивает прикладной характер книги, заключающийся в постепенном продвижении от физического смысла задачи к алгоритмическим построениям.

Математики-теоретики, вероятно, не найдут для себя в книге ничего нового, а при желании даже обнаружат в ряде мест недостаточную строгость доказательств, отличную от принятой терминологию, отсутствие теорем существования и т. п. Однако и для них книга будет весьма полезна, поскольку в ней по существу впервые дано систематическое рассмотрение актуальных задач на *ориентированных* графах, сочетающее в себе несомненные достоинства теоретической монографии Ф. Харрари «Теория графов» (М.: Мир, 1973) и блестящей прикладной работы Л. Форда и Д. Фалкерсона «Потоки в сетях» (М.: Мир, 1966).

В переводе мы в основном следовали сложившейся терминологии, однако сочли необходимым предложить вместо таких малоинформативных терминов, как аугментальная (augmenting) цепь, альтернирующая (alternating) цепь и экспонированная (exposed) вершина, более содержательные, на наш взгляд, термины — увеличивающая цепь, чередующая цепь и открытая вершина. Такой перевод, по нашему мнению, в большей степени соответствует прикладному характеру книги.

Автор этой книги Эдвард Майника — профессор факультета количественных методов анализа Иллинойского университета, член Американского общества исследования операций — опубликовал множество статей по оптимизационным алгоритмам на сетях и графах в специальных журналах. Алгоритмы, описываемые в книге, охватывают распределительные задачи, задачи выбора маршрута, задачи сетевого планирования, транспортные задачи, задачи размещения центров массового обслуживания.

Уже одно это перечисление говорит о несомненной ценности книги для специалистов, занятых разработкой автоматизированных систем управления, а также для студентов соответствующих специальностей, в рамках которых предусматривается изучение методов и моделей исследования операций.

Перевод выполнили М. И. Рубинштейн (предисловие, гл. 1—4) и М. Б. Кацнельсон (гл. 5—9).

Е. Масловский

Еве и Стенли, ожиданий которых я, надеюсь, не обманул

Предисловие

Эта книга не является очередным трудом по теории графов. В ней рассматриваются только алгоритмы поиска оптимального решения ряда задач, которые могут быть сформулированы в терминах сетей или графов.

Изучая эти алгоритмы еще аспирантом, я стал ощущать их многообразие, элегантность, внутреннюю взаимосвязь, более того, я почувствовал настоятельную необходимость собрать и объединить такие алгоритмы, рассеянные по различным журналам и монографиям. Я надеюсь, что мне удалось в рамках этой книги внести определенный вклад в соответствующий раздел в виде достаточно полного, взаимосвязанного и ясного изложения некоторых важных алгоритмов.

Данная книга может служить самостоятельным учебным пособием для студентов предпоследнего и последнего года обучения по многим специальностям. Знание основ математики и исследования операций, конечно, является нелишним, однако вряд ли играет существенную роль.

В книге принят интуитивный подход к анализу внутренних механизмов рассматриваемых алгоритмов, познанию их взаимосвязей и практическому использованию. При этом вопросы, связанные с принадлежностью описываемых алгоритмов к определенным разделам современной математики, а также вопросы программирования этих алгоритмов для ЭВМ не рассматриваются.

Глава 1 содержит основные понятия и определения. В книге сделана попытка добиться максимально возможной взаимной независимости отдельных глав, за исключением первой. Этот принцип нарушается лишь тогда, когда какой-то алгоритм использует другой в качестве вспомогательного. Однако и в этих случаях соответствующий материал можно изучать независимо, если вспомогательный алгоритм принимается читателем «на веру». Материал книги может быть исчерпывающе изучен за один семестр, а при изъятии некоторых незначительных подробностей этот же материал можно освоить, правда с меньшей глубиной, и за полсеместра.

Автору потребовалось немало чернил, чтобы пройти весь путь от замысла книги до ее издания. Я хочу выразить благодарность Рэнди Брауну из Кентского университета, Эллису Джонсону из фирмы IBM, Джорджу Немхаузеру из Корнеллского университета и Дугласу Ширу из Национального бюро стандартов за то, что

они внимательно прочитали рукопись и высказали ряд полезных замечаний. Я также искренне признателен моему коллеге Леонарду Кенту за его веру в реальность замысла данной книги и за его поддержку, которую я ощущал на протяжении нескольких лет работы над книгой. Наконец, хочу поблагодарить своих студентов, которые в течение трех лет безропотно выслушивали на занятиях материал отдельных глав данной книги. Но прежде всего это книга для вас, читатели, и ваших последователей, в какой бы области вы ни работали.

Эдвард Майника

ГЛАВА 1

Введение в теорию графов и сетей

1.1. Вводные замечания

Теория графов представляет собой раздел математики, имеющий широкие практические приложения. Многие проблемы, возникающие в таких весьма различных областях знания, как психология, химия, электротехника, планирование перевозок, управление, торговля и образование, могут быть сформулированы как задачи теории графов. Ввиду этого теория графов интересна не только сама по себе, но также и тем, что представляет общую основу, на которой результаты, полученные в различных областях знания, могут быть собраны, классифицированы, обобщены и распространены.

В отличие от других научных дисциплин теория графов имеет вполне определенную дату рождения. Первая работа по теории графов, написанная швейцарским математиком Леонардом Эйлером (1707—1783 гг.), была опубликована в 1736 г. в Трудах Академии наук в Санкт-Петербурге. Исследование Эйлера было проведено в связи с так называемой задачей о Кенигсбергских мостах. Город Кенигсберг (нынешний Калининград), располагавшийся тогда в Восточной Пруссии, был построен в месте слияния двух рек на их берегах и на двух островах (рис. 1.1). В городе было семь мостов, которые соединяли острова между собой и с береговыми частями города. Мог ли любой житель Кенигсберга, выйдя из дома, пройти по всем семи мостам города в точности по одному разу и вернуться домой? Ответ на этот вопрос должен быть отрицательным. В дальнейшем, когда в гл. 6 будет рассмотрен обобщенный вариант данной задачи, названной задачей почтальона, мы поймем, с чем это связано.

Развитие теории графов в конце XIX и начале XX вв. было связано с распространением представлений о молекулярном строении вещества и становлением теории электрических цепей. К 50-м годам нашего века в теории графов сложились два существенно различных направления: *алгеб-*

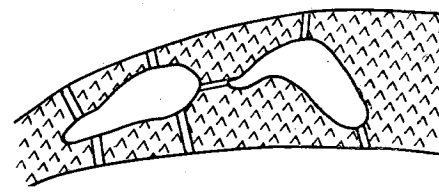


Рис. 1.1. Кенигсберг в 1736 г.

раическое и *оптимизационное*. Последнее получило широкое развитие благодаря появлению электронных вычислительных машин (ЭВМ) и в связи с разработкой методов линейного программирования. Мы будем почти везде рассматривать лишь оптимизационное направление теории графов.

Что же представляет собой граф? Любой граф состоит из двух групп элементов: точек и стрелок, соединяющих эти точки. Точки могут изображаться на плоскости, хотя могут и не иметь такой определенной «физической» привязки. Стрелки могут изображаться линиями (как прямыми, так и не прямыми), соединяющими пары точек. Например, для графа, изображенного на рис. 1.2, точки помечены буквами a, b, c, d, а стрелки — буквами α, β, γ, δ, ε, φ. Отметим, что имеются две стрелки α и β, которые идут из точки a в точку b, т. е. имеют началом точку a и концом точку b. Тот же самый граф можно было бы задать не рисунком, а просто перечнем его точек — a, b, c, d — и перечислением стрелок — $\alpha = (a, b)$, $\beta = (a, b)$, $\gamma = (c, a)$, $\delta = (b, c)$, $\varepsilon = (b, d)$, $\varphi = (d, c)$ — представленных упорядоченными парами точек, где первая точка пары определяет начало соответствующей стрелки, а вторая — ее конец. Придерживаясь стандартной терминологии, мы будем точки графа называть *вершинами*, а стрелки графа — *дугами*. Используя эти термины и представления, мы можем теперь дать формальное определение графа.

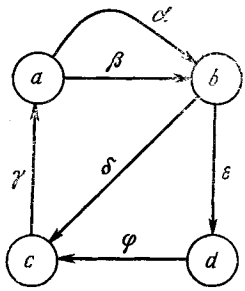


Рис. 1.2.

Граф — это совокупность множества X, элементы которого называются *вершинами*, и множества A упорядоченных пар вершин, элементы которого называются *дугами*. Граф обозначается как (X, A).

Предполагается, что как множество X, так и множество A содержат конечное число элементов.

Как правило, вершины будут обозначаться строчными латинскими буквами, а дуги — строчными греческими буквами. Например, на рис. 1.2 дуга γ может быть обозначена через (c, a), где вершина c определяет начало, а вершина a — конец дуги γ. Если две вершины соединяются (в одном направлении) не одной, а несколькими дугами, то последние могут быть представлены упорядоченной парой соответствующих вершин с различными индексами. Например, для графа, изображенного на рис. 1.2, дугу α можно было бы обозначить через $(a, b)_1$, а дугу β — через $(a, b)_2$. В тех случаях, когда это не приводит к недоразумениям, такая дополнительная индексация использоваться не будет.

ПРИМЕР 1. Обозначим через X множество всех аэропортов в шт. Иллинойс (США). Пусть A представляет собой множество всех пар аэропортов (x, y), таких, что существует беспосадочная гражданская авиалиния между аэропортом x и аэропортом y. Очевидно, (X, A) представляет собой граф.

ПРИМЕР 2. Пусть X обозначает множество всех пассажиров, находящихся на борту самолета, совершающего трансатлантический рейс. Обозначим через A множество всех пар пассажиров (x, y), таких, что пассажир x старше пассажира y и оба они говорят на одном и том же языке. Очевидно, (X, A) является графом с множеством вершин X и множеством дуг A. Могут ли в рассматриваемом графе одновременно присутствовать дуга (x, y) и дуга (y, x) ?

Определенные задачи теории графов (например, простейшая задача о паросочетании, с которой мы встретимся позже) требуют наличия информации только о концевых точках дуг. В таких задачах нет необходимости различать начало и конец дуги, т. е. не нужно приписывать дугам определенные направления. Граф, в котором направления дуг не задаются, называется *неориентированным*. Неориентированные дуги называются *ребрами*. Например, если в графе, изображенном на рис. 1.2, заменить стрелки ненаправленными линиями, то он превратится в неориентированный граф, в котором по сравнению с исходным дуги заменены ребрами.

На протяжении всей книги мы будем использовать обозначение (X, E) для неориентированного графа с множеством вершин X и множеством ребер E и обозначение (X, A) для графа с множеством вершин X и множеством дуг A.

Сеть — это не что иное, как граф, каждой дуге которого поставлены в соответствие одно или несколько чисел. В частности, если в примере 1 каждой дуге графа приписать конкретные значения расстояний, то исходный граф превратится в сеть.

Все последующие главы книги посвящены тем или иным прикладным оптимизационным задачам на графах или на сетях. Процедуры, которые даются для численного определения оптимальных решений рассматриваемых задач, называются *алгоритмами*, что и обусловливает название данной книги. Поскольку некоторые оптимизационные алгоритмы включают как составную часть и другие алгоритмы, на последовательность их рассмотрения в книге накладываются естественные ограничения, определившие порядок изложения материала в книге.

1.2. Некоторые понятия и определения

Ниже приводится ряд основных понятий и определений, которые используются на протяжении всей книги. Это делается для того, чтобы уменьшить зависимость между отдельными главами. Мотивировка введения тех или иных понятий будет дана в последующих главах, где достаточно подробно рассматриваются различные

прикладные задачи. Итак, рассмотрим некоторые понятия и определения.

Дуга, начальная и конечная вершина которой совпадают, называется *петлей*. На рис. 1,3 дуга β является петлей.

Будем говорить, что вершина и дуга *инцидентны* друг другу, если вершина является для этой дуги концевой или начальной точкой. На рис. 1.3 дуга α и вершина b инцидентны друг другу.

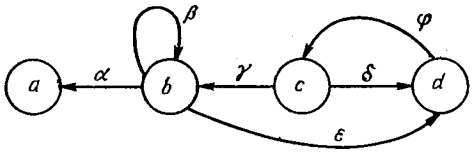


Рис. 1.3.

Будем говорить, что две дуги *инцидентны* друг другу, если обе они инцидентны одной и той же вершине. На рис. 1.3 дуги α и γ инцидентны друг другу, поскольку обе они инцидентны вершине b.

Будем называть две вершины *соседними*, если есть дуга, их соединяющая. На рис. 1.3 вершины b и c соседние, поскольку существует дуга γ, которая соединяет эти вершины.

Рассмотрим произвольную последовательность вершин x_1, x_2, ..., x_n, x_{n+1}. *Цепью* называется любая последовательность дуг $\alpha_1, \alpha_2, \ldots, \alpha_n$, такая, что концевыми точками дуги α_i являются вершины x_i и x_{i+1}, т. е. $\alpha_i = (x_i, x_{i+1})$ или $\alpha_i = (x_{i+1}, x_i)$ для $i = 1, 2, \ldots, n$. Вершина x_1 называется *начальной вершиной* цепи. Вершина x_{n+1} называется *конечной вершиной* цепи. Будем говорить, что цепь соединяет начальную вершину с конечной. *Длина* цепи совпадает с числом входящих в нее дуг. На рис. 1.3 последовательность дуг α, β, γ, δ, φ образует цепь длины 5, которая соединяет вершину a с вершиной c.

Цепь, для которой $\alpha_i = (x_i, x_{i+1})$ при всех $i = 1, 2, \ldots, n$, представляет собой *путь*. Длина пути, его начальная и конечная вершины могут быть определены точно так же, как для цепи. Например, на рис. 1.3 дуги β, ε и φ образуют путь длины 3, соединяющий вершину b с вершиной c.

Циклом называется цепь, у которой начальная и конечная вершины совпадают. *Контуром* называется путь, у которого начальная и конечная вершины совпадают. Длина цикла или контура определяется так же, как длина соответствующей цепи. Например, на рис. 1.3 дуги γ, ε, δ образуют цикл длины 3, а дуги γ, ε, φ — контур длины 3.

Цепь, путь, цикл или контур называются *простыми*, если ни одна вершина не инцидентна более чем двум входящим в нее дугам (т. е. если цепь, путь, цикл или контур не содержат внутри себя циклов). На рис. 1.3 цепь (α, γ) — простая, в то время как цепь (α, β, γ) таковой не является; цикл (γ, ε, δ) является простым, а цикл (α, β, ε, δ) простым назвать нельзя.

Граф называется *связным*, если в нем для каждой пары вершин найдется соединяющая их цепь. Например, графы, изображенные на рис. 1.2 и 1.3, являются связными, а граф, изображенный на рис. 1.4, является несвязным, поскольку в нем нет цепи, соединяющей вершины d и e. Любой граф можно рассматривать как неко-

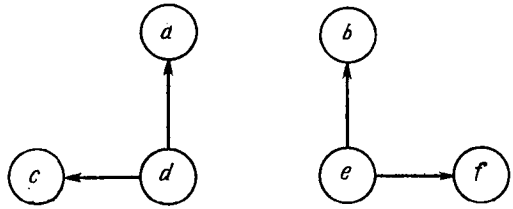


Рис. 1.4.

торую совокупность связных графов. Каждый из этих графов называется *компонентом* исходного графа. Граф, изображенный на рис. 1.4, имеет два компонента (назовите, какие).

Пусть X' представляет собой некоторое подмножество множества X, содержащее вершины графа $G = (X, A)$. Граф, множество вершин которого совпадает с X', а множество дуг включает все дуги множества A с концевыми вершинами в X', называется *подграфом графа G, порожденным X'*.

Пусть A' представляет собой некоторое подмножество множества A, содержащее дуги графа $G = (X, A)$. Граф, для которого

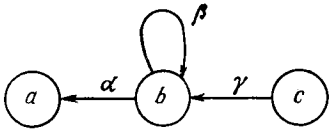


Рис. 1.5. Подграф, порожденный подмножеством вершин $\{a, b, c\}$.

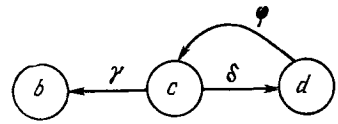


Рис. 1.6. Подграф, порожденный подмножеством дуг $\{\gamma, \delta, \psi\}$.

множество дуг совпадает с A', а множество вершин включает вершины, инцидентные дугам из A', называется *подграфом графа G, порожденным A'*. Например, для графа, изображенного на рис. 1.3, подграф, порожденный подмножеством вершин $\{a, b, c\}$, изображен на рис. 1.5. Подграф того же графа, порожденный подмножеством дуг $\{\gamma, \delta, \varphi\}$, изображен на рис. 1.6.

Совокупность дуг называется *деревом*, если она удовлетворяет следующим двум условиям: 1) порождает связный подграф; 2) не содержит циклов.

В графе, изображенном на рис. 1.3, следующие совокупности дуг образуют дерево:

$\{\alpha, \gamma, \varepsilon\}$, $\{\alpha, \gamma, \varphi\}$, $\{\alpha, \varepsilon, \delta\}$, $\{\varphi, \gamma\}$, $\{\alpha, \gamma\}$, $\{\varepsilon\}$, $\{\gamma\}$.

Совокупность дуг того же графа $\{\varphi, \gamma, \varepsilon\}$ не образует дерева, поскольку она содержит цикл.

Лесом называется любая совокупность дуг, не содержащая циклов. Таким образом, лес состоит из одного или большего числа деревьев. Совокупность дуг, представленных на рис. 1.4, образует лес, состоящий из двух деревьев.

Покрывающим деревом графа называется любое дерево, образованное совокупностью его дуг, включающих все вершины графа. В графе, изображенном на рис. 1.3, совокупность дуг $\{\alpha, \varepsilon, \varphi\}$ образует покрывающее дерево, поскольку она включает все вершины данного графа a, b, c и d. Ясно, что граф, состоящий более чем из одного компонента, не имеет покрывающего дерева. Любой же связный граф содержит некоторое покрывающее дерево.

Дерево, состоящее из одной дуги, включает две вершины; дерево, состоящее из двух дуг, включает три вершины; дерево, состоящее из трех дуг, включает четыре вершины; вообще дерево, состоящее из $(n-1)$ дуги, должно включать n вершин. Следовательно, любое покрывающее дерево связного графа, имеющего n вершин, состоит из $(n-1)$ дуги.

Множество дуг, исключение которых из графа увеличивает число его компонентов, называется *разрезом*. Разрез, который не содержит в качестве собственного подмножества никакого другого разреза, называется *простым разрезом*. Для графа, изображенного на рис. 1.3, множество дуг $\{\delta, \varepsilon, \gamma, \varphi\}$ образует разрез, так как исключение их приводит к графу, состоящему из 3 компонентов. Этот разрез не является простым, поскольку содержит в себе другой разрез $\{\delta, \varepsilon, \varphi\}$, который, кстати, является простым.

Пусть G — произвольный граф без петель, состоящий из m вершин и n дуг. Пусть $\underline{G}$ — матрица, состоящая из m строк, каждая из которых соответствует определенной вершине, и n столбцов, каждый из которых соответствует определенной дуге. Пусть через $\underline{G}_{ij}$ обозначен элемент (i, j) матрицы $\underline{G}$, который определяется следующим образом:

$$\underline{G}_{ij} = \begin{cases} +1, & \text{если вершина, которой соответствует } i\text{-я строка, является начальной для дуги, соответствующей } j\text{-му столбцу;} \\ -1, & \text{если вершина, которой соответствует } i\text{-я строка, является конечной для дуги, соответствующей } j\text{-му столбцу;} \\ 0, & \text{во всех других случаях.} \end{cases}$$

Очевидно, в каждом столбце матрицы $\underline{G}$ все элементы, за исключением двух, будут нулевыми. Ненулевые элементы каждого столбца совпадают с $+1$ и -1. Матрица $\underline{G}$ называется *матрицей графа* G[1].

Матрица графа, изображенного на рис. 1.2, имеет следующий вид:

	α	β	γ	δ	ε	φ
a	-1	-1	$+1$	0	0	0
b	$+1$	$+1$	0	-1	-1	0
c	0	0	-1	$+1$	0	$+1$
d	0	0	0	0	1	-1

Очевидно, некоторая матрица может быть матрицей графа в том и только в том случае, если каждый ее столбец содержит только два отличных от нуля элемента: $+1$ и -1.

Матричное представление дает удобный способ описания графа, не связанный с перечислением вершин и дуг графа или построением диаграмм. Машинные программы оптимизационных алгоритмов, описываемых в данной книге, неизменно используют матричное описание графа. Однако с целью облегчения представления излагаемого материала далее для графов даются интуитивно более понятные описания в виде соответствующих диаграмм.

1.3. Линейное программирование

Многие из задач, рассматриваемых в данной книге, могут быть сформулированы как задачи линейного программирования. Данный раздел содержит краткий обзор определений, относящихся к линейному программированию, которые будут использованы в последующих главах. Хотя приводимых здесь определений вполне достаточно для формального понимания последующего материала, при их описании мы отнюдь не стремились к тому, чтобы обеспечить глубокое или интуитивное понимание самих методов линейного программирования. Для этого читателю рекомендуется обратиться к любой книге, специально посвященной линейному программированию.

Начнем рассмотрение некоторых аспектов линейного программирования с формулировки общей задачи. Пусть $x_1, x_2, \ldots, x_n$ — некоторые переменные, которые могут принимать любые неотрицательные действительные значения. Пусть $c_1, c_2, \ldots, c_n, b_1, b_2,$

[1] Чаще $\underline{G}$ называют матрицей инциденций графа G. Поскольку в данной книге другие матричные представления графа не используются, такое упрощение терминологии допустимо. — *Прим. перев.*

..., b_m, a_{ij} — действительные числа при $i = 1, 2, \ldots, m$ и $j = 2, \ldots, n$.

Задачей линейного программирования называется любая задача, которая может быть представлена в следующем виде:

максимизировать

$$\sum_{j=1}^{j=n} c_j x_j \tag{1}$$

при условии, что

$$\sum_{j=1}^{j=n} a_{1j} x_j \leqslant b_1,$$

$$\sum_{j=1}^{j=n} a_{2j} x_j \leqslant b_2 \tag{2}$$

. . . . . .

$$\sum_{j=1}^{j=n} a_{mj} x_j \leqslant b_m,$$

$$x_1 \geqslant 0, \quad x_2 \geqslant 0, \ldots, x_n \geqslant 0. \tag{3}$$

В любом из соотношений (2) знак $\leqslant$ может быть заменен на знак $\geqslant$ или знак равенства, а в любом из соотношений (3) знак $\geqslant$ может быть заменен знаком $\leqslant$. Кроме того, любое из соотношений (3) может быть просто опущено. В этом случае соответствующая переменная может иметь любой знак. Наконец, сумма (1) может не максимизироваться, а минимизироваться. Приведем пример задачи линейного программирования:

максимизировать

$$2x_1 + 7x_2 - 3x_3$$

при условии, что

$$2x_1 - 2x_2 + 1x_3 \leqslant 6,$$
$$4x_1 - 6x_2 - 3x_3 \geqslant 7{,}325,$$
$$-8{,}25x_1 + 1x_2 - 0{,}3x_3 = 8,$$
$$x_1 \geqslant 0, \; x_2 \leqslant 0, \; x_2 \text{ не имеет ограничений по знаку.}$$

Выражение (1), значение которого максимизируется или минимизируется, называется *целевой функцией*. Целевая функция представляет собой линейную комбинацию переменных x_1, x_2, ..., x_n. Соотношения системы (2), число которых равно m и которым должны удовлетворять n исходных переменных, называются *ограни-*

чениями задачи линейного программирования. Заметим, что левая часть каждого ограничения представляет собой линейную комбинацию переменных. Соотношения системы (3) называются *условиями неотрицательности* в задаче линейного программирования.

Из задачи линейного программирования с n переменными и m ограничениями, представленной в виде соотношений (1), (2) и (3), можно получить некоторую задачу линейного программирования, называемую *двойственной* по отношению к исходной. Двойственная задача имеет n ограничений, каждое из которых соответствует определенной переменной исходной задачи, и m переменных y_1, y_2, ..., y_m, каждая из которых соответствует определенному ограничению исходной задачи. Двойственная задача линейного программирования для задачи, представленной соотношениями (1), (2) и (3), формулируется следующим образом:

минимизировать

$$\sum_{i=1}^{i=m} b_i y_i \tag{1'}$$

при условии, что

$$\sum_{i=1}^{i=m} a_{i1} y_i \geqslant c_1,$$

$$\sum_{i=1}^{i=m} a_{i2} y_i \geqslant c_2 \tag{2'}$$

. . . . . .

$$\sum_{i=1}^{i=m} a_{in} y_i \geqslant c_n,$$

$$y_1 \geqslant 0, \quad y_2 \geqslant 0, \ldots, \quad y_m \geqslant 0. \tag{3'}$$

Если бы в i-м ограничении системы (2) исходной задачи вместо знака $\leqslant$ стоял бы знак $\geqslant$, то i-е условие неотрицательности в системе соотношений (3′) двойственной задачи следовало бы заменить соотношением $y_i \leqslant 0$. Если бы i-е ограничение системы (2) исходной задачи имело вид равенства, то i-е ограничение неотрицательности в системе (3′) двойственной задачи следовало бы опустить и считать, что переменная y_i не имеет ограничений по знаку.

Если j-е условие неотрицательности в исходной задаче имеет вид $x_j \leqslant 0$, то соответствующее j-е ограничение в системе (2′) двойственной задачи имеет вид неравенства со знаком $\leqslant$, т. е. вид $\Sigma a_{ij} y_i \leqslant c_j$. Если переменная x_j не ограничена по знаку, тогда j-е ограничение двойственной задачи имеет вид равенства, т. е. вид $\Sigma a_{ij} y_i = c_j$.

Наконец, если исходная задача линейного программирования является задачей максимизации, то двойственная ей задача является задачей минимизации, и наоборот.

Исходная задача линейного программирования обычно называется *прямой* задачей. Поскольку двойственная задача также является задачей линейного программирования, то в линейном программировании можно рассматривать пары задач в составе прямой и двойственной. При этом, как может легко показать сам читатель, задача, двойственная по отношению к двойственной задаче, совпадает с исходной прямой задачей.

Двойственной по отношению к задаче линейного программирования, рассмотренной в приведенном выше примере, является следующая задача:

минимизировать

$$-6y_1 + 7{,}325y_2 + 8y_3$$

при условии, что

$$2y_1 + 4y_2 - 8{,}25y_3 \geqslant 2,$$
$$-2y_1 - 6y_2 + 1y_3 \leqslant 7,$$
$$1y_1 - 3y_2 - 0{,}3y_3 = -3,$$
$$y_1 \geqslant 0,\ y_2 \leqslant 0,\ y_3 \text{ не имеет ограничений по знаку.}$$

Уместно поставить следующие вопросы. Как установить, что некоторое решение $x_1, x_2, .., x_n$, удовлетворяющее всем соотношениям систем (2) и (3), обеспечивает оптимум целевой функции (1)? Как установить, что некоторое решение двойственной задачи $y_1, y_2, \ldots, y_m$, удовлетворяющее всем соотношениям систем (2′) и (3′), дает оптимальное значение целевой функции (1′)? Другими словами, как установить, что некоторое допустимое решение задачи линейного программирования является ее оптимальным решением?

Ответы на эти вопросы могут быть получены на основе так нызываемых условий *дополняющей нежесткости*.

Пусть $(x_1, x_2, \ldots, x_n)$ — множество допустимых значений переменных прямой задачи, а $(y_1, y_2, \ldots, y_m)$ — множество допустимых значений переменных двойственной задачи. Тогда, используя (2) и (2′), получим для целевой функции прямой задачи следующую оценку:

$$\sum_{i=1}^{i=n} c_i x_i \leqslant \sum_{i=1}^{i=n} x_i \sum_{j=1}^{j=m} a_{ji} y_j = \sum_{j=1}^{j=m} y_j \sum_{i=1}^{i=n} a_{ji} x_i \leqslant \sum_{j=1}^{j=m} y_j b_j. \qquad (4)$$

Последняя сумма в выражении соотношений (4) совпадает с выражением для целевой функции двойственной задачи. Следовательно, значение целевой функции прямой задачи всегда меньше

или равно значению целевой функции двойственной задачи. Если бы можно было найти такие решения прямой и двойственной задач, при которых оба неравенства в соотношении (4) выполняются как строгие равенства, то соответствующие значения переменных давали бы оптимальные решения прямой и двойственной задач. Следовательно, необходимо найти условия, при которых в соотношении (4) имеют место только строгие равенства.

Левое неравенство в соотношении (4) переходит в равенство тогда и только тогда, когда

$$\left(\sum_{j=1}^{j=m} a_{ji}y_j - c_i\right) x_i = 0 \quad (i = 1, 2, \ldots, n). \tag{5}$$

Правое неравенство в отношении (4) переходит в равенство тогда и только тогда, когда

$$\left(b_j - \sum_{i=1}^{i=n} a_{ji}x_i\right) y_i = 0 \quad (j = 1, 2, \ldots, m). \tag{6}$$

Таким образом, если множества допустимых значений переменных прямой и двойственной задач удовлетворяют соотношениям (5) и (6), то эти множества определяют и оптимальные решения соответственно прямой и двойственной задач. Уравнения (5) и (6) дают возможность определить, являются ли некоторые допустимые решения пары двойственных задач линейного программирования их оптимальными решениями. Из этих уравнений ясно видно, почему соответствующие условия оптимальности называются условиями *дополняющей нежесткости.*

Разность $(\sum_{j=1}^{j=m} a_{ji}y_i - c_i)$, фигурирующая в уравнении (5), называется величиной *нежесткости i-го ограничения двойственной задачи.* Аналогично разность $(b_j - \sum_{i=1}^{i=n} a_{ji}x_i)$, фигурирующая в уравнении (6), называется величиной нежесткости *j-го ограничения прямой задачи.*

Задача линейного программирования, в которой некоторые ограничения являются неравенствами, может быть следующим образом приведена к эквивалентной задаче линейного программирования, в которой все ограничения представлены равенствами.

Если i-е ограничение исходной задачи имеет вид $\Sigma a_{ij}x_j \leqslant b_i$, добавим к правой части соответствующего неравенства новую переменную $s_i \geqslant 0$, формируя тем самым новое ограничение $\Sigma a_{ij}x_j + s_i = b_i$. Пусть при этом коэффициент при s_i в целевой функции равен 0. Тогда введение новой переменной не меняет целевой функции, а лишь преобразует i-е ограничение задачи в равенство.

Если i-е ограничение исходной задачи имеет вид $\Sigma a_{ij}x_j \geqslant b_i$, вычтем из правой части соответствующего неравенства новую переменную $s_i \geqslant 0$, формируя тем самым новое ограничение $\Sigma a_{ij}x_j - s_i = b_i$. Пусть при этом коэффициент при s_i в целевой функции равен 0. Тогда введение новой переменной не меняет целевой функции, а лишь преобразует i-е ограничение задачи в равенство.

В частности, рассматриваемая в данном разделе в качестве примера задача линейного программирования может быть преобразована в эквивалентную задачу линейного программирования с ограничениями в виде равенств. Соответствующая эквивалентная формулировка имеет следующий вид:

максимизировать

$$2x_1 + 7x_2 - 3x_3 + 0s_1 + 0s_2$$

при условии, что

$$2x_1 - 2x_2 + 1x_3 + 1s_1 + 0s_2 = 6,$$
$$4x_1 - 6x_2 - 3x_3 + 0s_1 - 1s_2 = 7{,}325,$$
$$-8{,}25x_1 + 1x_2 - 0{,}3x_3 + 0s_1 + 0s_2 = 8,$$

$x_1 \geqslant 0$, $x_2 \leqslant 0$, $s_1 \geqslant 0$, $s_2 \geqslant 0$, x_3 не имеет ограничений по знаку.

Вообще в задаче линейного программирования с ограничениями в виде равенств число переменных должно превышать число ограничений, т. е. должно быть выполнено соотношение $n > m$. Таким образом, в рассматриваемом случае число переменных превышает число ограничений на величину $(n - m)$. Предположим, что мы произвольно выбрали $(n - m)$ переменных и положили их равными 0. Тогда соответствующие переменные могли бы быть исключены из состава ограничений. Полученная таким образом система состояла бы из m уравнений с m неизвестными.

Данная система из m линейных уравнений могла бы быть решена любым стандартным методом, например методом, использующим правило Крамера, или методом исключения Гаусса — Жордана. Если оказывается, что рассматриваемая система имеет единственное решение, при котором значения всех переменных неотрицательны, то оно называется *базисным*. Поскольку каждый выбор $(n - m)$ дополнительных переменных, которые первоначально считаются равными нулю, может приводить не более чем к одному базисному решению, в задаче может присутствовать лишь конечное множество различных базисных решений.

Один из самых важных результатов теории линейного программирования состоит в следующем: если в задаче линейного программирования существует по крайней мере одно оптимальное решение, то существует и такое оптимальное решение задачи, которое является базисным.

Следовательно, чтобы найти оптимальное решение задачи линейного программирования, необходимо исследовать лишь конечное множество базисных решений.

Практически задачи линейного программирования решаются с помощью процедуры, названной *симплекс-алгоритмом*. Этот алгоритм исходя из некоторого базисного решения определенным образом порождает другое базисное решение, имеющее по сравнению с исходным лучшее значение целевой функции. Подобная процедура повторяется вплоть до получения базисного решения, которое удовлетворяет условиям оптимальности. Число таких повторений конечно, так как ни одно из базисных решений не встречается в них более одного раза (каждое новое решение «лучше» предыдущих), а общее число базисных решений конечно.

Некоторые из задач теории графов, представленных в последующих главах, будут решаться аналогичным образом. А именно будет отыскиваться некоторое базисное решение и на его основе — другое базисное решение, которое по значению целевой функции лучше исходного. При этом подобная процедура будет повторяться конечное число раз до получения оптимального решения.

УПРАЖНЕНИЯ

1. Постройте граф, множество вершин которого соответствует множеству курсов обучения, которые вам необходимо пройти для получения определенной ученой степени. При этом дугу от вершины x к вершине y включайте в граф только в том случае, если курс x предшествует курсу y. Интерпретируйте следующие элементы построенного графа: а) путь, б) цепь, в) цикл, г) контур, д) связную компоненту.

2. *Степень захода* $d^-(x)$ вершины x определяется как число дуг, «заходящих» в вершину x. *Степень исхода* $d^+(x)$ вершины x определяется как число дуг, «исходящих» из вершины x. Степень $d(x)$ вершины x определяется как сумма степеней захода и исхода для данной вершины.

Покажите, что в любом графе G количество вершин с нечетными степенями четно. Покажите, что для любого графа $G = (X, A)$ имеет место соотношение.

$$\sum_{x \in X} d^+(x) = \sum_{x \in X} d^-(x).$$

3. Возможно ли, чтобы разрез и цикл содержали в точности одну общую дугу? Если невозможно, то почему?

4. Пусть T — покрывающее дерево графа G. Покажите, что в G для любых двух вершин существует единственная соединяющая их цепь, состоящая только из ребер дерева T.

5. Рассмотрите следующую задачу линейного программирования:

Максимизировать

$$3x_1 + 2x_2 + 1x_3 - 4x_4$$

при условии, что

$$2x_1 + 1x_2 + 3x_3 + 7x_4 \leqslant 10,$$
$$x_1 \geqslant 0, \quad x_2 \geqslant 0, \quad x_3 \geqslant 0, \quad x_4 \geqslant 0.$$

а) Найдите оптимальное решение задачи. (*Указание.* Рассматривайте только базисные решения.)

б) Сформулируйте для исходной задачи двойственную задачу линейного программирования.

в) Используйте условия дополняющей нежесткости для получения оптимального решения двойственной задачи.

6. Любая ли часть дерева сама является деревом? Любая ли часть леса сама является лесом?

7. Образует ли последовательность дуг α, γ, ε графа, изображенного на рис. 1.3, цепь, путь? Ответьте на тот же вопрос для последовательности дуг α, γ, δ, ε.

8. Пусть лес F состоит из t деревьев и содержит v вершин. Сколько дуг содержится в F?

9. Многие страны Общего рынка имеют общие границы. Постройте граф G, вершины которого соответствуют различным странам Общего рынка. Две вершины этого графа соединены дугой, если соответствующие страны имеют общую границу. Является ли граф G связным? Найдите в графе G разрез, включающий наименьшее число дуг. Имеется ли в Общем рынке страна, исключение которой из сообщества разорвало бы все наземные коммуникации между оставшимися странами?

ЛИТЕРАТУРА

Теория графов

1. Berge C., Graphs and Hypergraphs (translated by E. Minieka), North-Holland, Amsterdam, 1973.
2. Busacker R., Saaty T., Finite Graphs and Networks, McGraw-Hill, New York, 1965.
3. Ford L. R., Fulkerson D. R., Flows in Networks, Princeton Press, Princeton, 1962. [Имеется перевод: Форд Л. Р., Фалкерсон Д. Р. Потоки в сетях. — М.: Мир, 1966.]
4. Frank H., Frisch I., Communication, Transmission, and Transportation Networks, Addison-Wesley, Reading, 1971.
5. Harary F., Garph Theory, Addison-Wesley, Reading, 1969. [Имеется перевод: Харари Ф. Теория графов. — М.: Мир, 1973.]
6. Hu T. C., Integer Programming and Network Flows, Addison-Wesley, Reading, 1969. [Имеется перевод: Ху Т. Целочисленное программирование и потоки в сетях. — М.: Мир, 1974.]
7. Ore O., Graphs and Their Uses, Random House new Mathematical Library, Random House, New York, 1963. [Имеется перевод: Оре О. Графы и их применение. — М.: Мир, 1965.]
8. Potts R. B., Oliver R. M., Flows in Transportation Networks, Academic Press, New York, 1972.
9. Wilson R., Introduction to Graph Theory, Academic Press, New York, 1972.

Линейное программирование

10. Dantzig G. B., Linear Programming and Extensions. Princeton Press, Princeton, 1963. [Имеется перевод: Данциг Дж. Линейное программирование и его обобщения. — М.: Мир, 1966.]
11. Hadley G., Linear Programming Addison-Wesley, Reading, 1962.
12. Simonnard M., Linear Programming, (translated by W. B. Jewell), Prentice-Hall, Englewood, 1968.

ГЛАВА 2

Алгоритмы построения деревьев

Граф может содержать много различных деревьев. В данной главе рассматривается ряд алгоритмов, используемых для построения деревьев, удовлетворяющих некоторым свойствам оптимальности.

2.1. Алгоритмы построения покрывающих деревьев

Рассмотрим граф $G = (X, E)$, в котором направления дуг не задаются. Предположим, что каждому ребру (x, y) этого графа приписан вес $a(x, y)$. Определим *вес дерева* как сумму весов ребер, его составляющих.

В данном разделе принят следующий порядок изложения. Сначала рассматривается алгоритм построения какого-нибудь покрывающего дерева графа G. Затем рассматривается алгоритм построения покрывающего дерева графа G минимального веса, т. е. покрывающего дерева, вес которого не больше веса любого другого покрывающего дерева графа G.

ПРИМЕР 1. (*Распространение слухов.*) Рассмотрим небольшую деревушку, в которой некоторые из жителей имеют каждодневные встречи друг с другом. Может ли в этой деревне распространиться какой-либо слух?

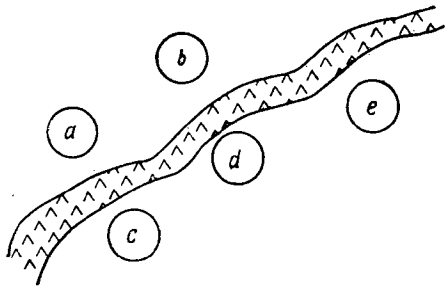
Чтобы ответить на этот вопрос, поставим в соответствие каждому жителю деревни вершину графа. Соединим две вершины ребром, если соответствующие жители ежедневно общаются друг с другом. При условии связности полученного таким образом графа на поставленный выше вопрос можно ответить положительно. Чтобы определить, является ли данный граф связным, можно было бы, например, попытаться построить для него покрывающее дерево. Если граф не содержит ни одного покрывающего дерева, то он не может быть связным и слух не может распространиться по всей деревне.

ПРИМЕР 2. В управлении шоссейных дорог рассматривается проект строительства новых дорог, которые должны связать пять городов некоторого района (причем не обязательно непосредственно каждую пару городов). Стоимость прокладки дороги между каждой парой городов известна (рис. 2.1). Построим граф, вершины которого соответствуют городам, а ребра — дорогам, которые могут быть проложены между определенными городами. Припишем каждому ребру вес, который равен стоимости строительства соответствующей дороги.

Составление проекта строительства дороги теперь можно свести к задаче построения для соответствующего графа покрывающего дерева минимальной стоимости. Это возможно в силу того, что, во-первых, ребра любого покрывающего дерева соединяют каждую вершину (город) графа с любой другой вершиной (городом) и, во-вторых, покрывающее дерево минималь-

ного веса представляет собой совокупность новых дорог минимальной общей стоимости.

Заметим, что если допускается возможность построения сети дорог, которой дороги соединяются в пунктах, отличных от мест расположения рассматриваемой группы городов, то соответствующая задача по сравнению с задачей о минимальном покрывающем дереве становится более сложной



От \ К	*a*	*b*	*c*	*d*	*e*
a	*x*	5	50	80	90
b	5	*x*	70	60	50
c	50	70	*x*	8	20
d	80	60	8	*x*	10
e	90	50	20	10	*x*

Рис. 2.1. К построению сети шоссейных дорог.

Перейдем теперь к обсуждению алгоритма построения покрывающего дерева, который является одним из наиболее изящных среди алгоритмов, рассматриваемых в дальнейшем. Суть алгоритма состоит в просмотре в произвольном порядке ребер исходного графа. При каждом просмотре в отношении соответствующего ребра принимается решение о том, будет или не будет оно включено в покрывающее дерево. Алгоритм можно представить как процесс *окрашивания* ребер. При этом голубой цвет используется для окраски ребер включаемых в покрывающее дерево, а оранжевый — для окраски ребер, не включаемых в это дерево.

В алгоритме при рассмотрении ребра осуществляется лишь проверка того, не образует ли данное ребро в совокупности с ребрами, уже включенными в дерево (голубыми ребрами), цикл. Если результат проверки является положительным, то рассматриваемое ребро не включается в дерево (окрашивается в оранжевый цвет). В противном случае это ребро включается в дерево (окрашивается в голубой цвет).

Как же в алгоритме осуществляется проверка того, образует ли просматриваемое ребро цикл в совокупности с ребрами, уже включенными в дерево (голубыми ребрами)? Ребра, включенные в дерево, составляют граф, имеющий один или несколько связных компонентов. Вершины, принадлежащие отдельному связному компоненту, объединяются в совокупность, которая в рамках рас-

сматриваемого алгоритма будет называться «букетом». Некоторое ребро образует цикл с ребрами, уже включенными в дерево, если обе его концевые вершины принадлежат одному и тому же связному компоненту (одному и тому же букету).

Работа алгоритма по построению покрывающего дерева заканчивается, когда количество ребер, окрашенных в голубой цвет, становится равным числу, на единицу меньшему числу вершин рассматриваемого графа, или, что то же самое, когда все вершины графа оказываются в одном букете. Указанные условия завершения алгоритма действительно эквивалентны, поскольку любое покрывающее дерево должно содержать число ребер, на единицу меньше числа вершин. Причем эти условия относятся к случаю, когда исходный граф содержит покрывающее дерево. Если же граф такого дерева не содержит, т. е. является несвязным, алгоритм заканчивается после раскраски всех ребер графа, при этом число ребер, окрашенных в голубой цвет, оказывается недостаточным для формирования покрывающего дерева.

Алгоритм построения покрывающего дерева

Перед началом работы алгоритма все ребра исходного графа являются не окрашенными и ни один из букетов не сформирован.

Шаг 1. Выбрать любое ребро, не являющееся петлей. Окрасить это ребро в голубой цвет и сформировать букет, включив в него концевые вершины окрашенного ребра.

Шаг 2. Выбрать любое неокрашенное ребро, которое не является петлей. (Если в графе такого ребра не найдется, закончить процедуру: исходный граф не содержит покрывающего дерева.) После указанного выбора возможны четыре различных случая.

а) Обе концевые вершины выбранного ребра принадлежат одному и тому же букету.

б) Одна из концевых вершин выбранного ребра принадлежит некоторому букету, а другая концевая вершина не принадлежит ни одному из уже сформированных букетов.

в) Ни одна из концевых вершин не принадлежит ни одному из сформированных букетов.

г) Концевые вершины выбранного ребра принадлежат различным букетам.

Если имеет место случай (а), окрасить выбранное ребро в оранжевый цвет (оно не включается в дерево) и вернуться к началу шага 2. Если имеет место случай (б), окрасить выбранное ребро в голубой цвет (оно включается в дерево) и включить его концевую вершину, не принадлежавшую ранее ни одному букету, в тот же букет, которому принадлежит другая концевая вершина рассматриваемого ребра. Если имеет место случай (в), окрасить рассматриваемое ребро в голубой цвет и сформировать новый букет из

его концевых вершин. Наконец, если имеет место случай (Г) окрасить рассматриваемое ребро в голубой цвет, а оба букета которым принадлежат его концевые вершины, слить в один новый букет.

По завершении шага 2 перейти к шагу 3.

Шаг 3. Если все вершины графа вошли в один букет, закончить процедуру, так как при этом условии голубые ребра образуют покрывающее дерево. В противном случае вернуться к началу шага 2.

Заметим, что описанный алгоритм состоит из многократного повторения шага 2, причем при каждом выполнении этого шага некоторое ребро окрашивается в определенный цвет и остается окрашенным на протяжении всей процедуры. Поэтому, если исходный граф состоит из конечного числа ребер, алгоритм должен закончиться через конечное число шагов.

Если алгоритм не заканчивается построением покрывающего дерева, то исходный граф вообще не содержит ни одного покрывающего дерева. Это обусловливается следующим обстоятельством. По окончании работы алгоритма будут сформированы по крайней мере два букета, для которых не найдется ни одного ребра, соединяющего какую-либо вершину одного из них с какой-либо вершиной другого. Действительно, если бы такое ребро нашлось, оно в процессе работы алгоритма было бы окрашено в голубой цвет, а соответствующие букеты слились бы в один букет. Таким образом, алгоритм делает именно то, для чего он предназначен: строит для графа покрывающее дерево (при условии что граф содержит такое дерево).

Описанный алгоритм обладает свойством, состоящим в том, что каждое ребро просматривается не более одного раза. Если в ходе алгоритма некоторое ребро окрашивается, то это ребро в дальнейшем не может быть просмотрено вновь. Алгоритмы типа алгоритма построения покрывающего дерева, в которых каждый из имеющихся элементов просматривается не более одного раза и при просмотре отдельных элементов «судьба» последних решается раз и навсегда, называются «поедающими» алгоритмами. Достоинство поедающих алгоритмов заключается в том, что, во-первых, в процессе их работы не приходится расходовать время на повторный просмотр соответствующих элементов и, во-вторых, для них достаточно просто определить максимальное количество выполняемых операций. В частности, для алгоритма построения покрывающего дерева число выполненных шагов не должно превысить числа ребер в соответствующем графе.

ПРИМЕР 3. Построим покрывающее дерево для графа, изображенного на рис. 2.2. Будем рассматривать ребра графа в следующем произвольно выбранном порядке: (a, b), (d, e), (a, d), (b, e), (e, c), (c, b), (a, c) и (c, d).

Ребро	Цвет	Букет № 1	Букет № 2
		Сначала пуст	Сначала пуст
(a, b)	Голубой	a, b	Пуст
(d, e)	»	a, b	d, e
(a, d)	»	a, b, d, e	Пуст
(b, e)	Оранжевый	a, b, d, e	»
(e, c)	Голубой	a, b, d, e, c	»

Результаты выполнения каждого шага алгоритма приводятся ниже. Алгоритм заканчивается просмотром ребра (e, c), поскольку после этого вершины рассматриваемого графа попадают в один букет. Итак, четыре голубых ребра (a, b), (d, e), (a, d) и (e, c) образуют для данного графа покрывающее дерево.

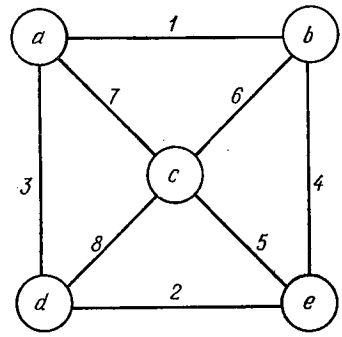


Рис. 2.2.

Конечно, вид построенного с помощью рассматриваемого алгоритма покрывающего дерева зависит от того, в каком порядке в нем просматривались ребра исходного графа. В частности, если бы в графе из примера 3 ребра просматривались в обратном порядке, то алгоритм построил бы покрывающее дерево, состоящее из ребер (c, d), (a, c), (c, b) и (e, d).

Теперь рассмотрим задачу поиска покрывающего дерева с минимально или максимально возможным весом, которая называется соответственно задачей о минимальном или максимальном покрывающем дереве. Очевидно, если граф содержит некоторое покрывающее дерево, то в нем можно выделить как минимальное, так и максимальное покрывающее дерево. Эти покрывающие деревья могут быть легко построены с помощью рассмотренного выше алгоритма при выборе определенного порядка просмотра ребер.

Алгоритм построения минимального покрывающего дерева

Данный алгоритм представляет собой алгоритм построения покрывающего дерева, в котором ребра просматриваются в порядке возрастания их весов (первое ребро — с минимальным весом, пос-

леднее ребро — с максимальным). Если два или более ребер имеют одинаковые веса, то они упорядочиваются произвольным образом.

Алгоритм построения максимального покрывающего дерева

Данный алгоритм представляет собой алгоритм построения покрывающего дерева, в котором ребра просматриваются в порядке убывания их весов (первое ребро — с максимальным весом, последнее ребро — с минимальным). Если два или более ребер имеют одинаковые веса, они упорядочиваются произвольным образом.

Обоснование алгоритма построения минимального дерева. Обоснование данного алгоритма будет проведено по схеме доказательства от противного. Пусть алгоритм строит дерево T, но на самом деле минимальным покрывающим деревом является некоторое дерево S. Так как деревья S и T не идентичны, они отличаются по крайней мере одним ребром. Пусть ребро $e_1 = (x, y)$ — первое из просмотренных в алгоритме ребер, которое содержится в T, но отсутствует в S. Поскольку S является покрывающим деревом, в нем найдется единственная цепь, соединяющая вершины x и y. Обозначим эту цепь через $c(x, y)$. Если ребро $e_1 = (x, y)$ добавить к дереву S, то оно вместе с цепью $c(x, y)$ образует цикл. Так как дерево T не содержит циклов, то цикл $(e_1, S(x, y))$ должен содержать по крайней мере одно ребро, скажем ребро e_2, которое не содержится в T.

Исключим из дерева S ребро e_2 и включим в него ребро e_1. Полученную таким образом совокупность ребер обозначим через S'. Очевидно, S', так же как S и T, является покрывающим деревом. Поскольку дерево S по предположению является минимальным, вес дерева S' должен быть не меньше веса дерева S, откуда следует, что $a(e_1) \geqslant a(e_2)$.

Предположим, что ребро e_2 в алгоритме построения дерева T было просмотрено раньше, чем ребро e_1. Поскольку ребро e_2 не было включено в дерево T, оно должно составлять цикл с ребрами, просмотренными до e_2 и включенными в T. Однако это противоречит тому, что ребра, с которыми e_2 составляет цикл в T, должны входить и в дерево S, так как e_1 — первое ребро в последовательности просматриваемых в алгоритме построения дерева T ребер, которое содержится в T, но не содержится в S. Таким образом, налицо противоречие, ибо в дереве S существование цикла невозможно. Отсюда следует заключение, что ребро e_1 в соответствующем алгоритме должно просматриваться раньше, чем ребро e_2, и потому $a(e_2) \geqslant a(e_1)$. Окончательно получаем, что $a(e_1) = a(e_2)$, и потому деревья S и S' имеют одинаковый суммарный вес. При этом дерево S' имеет на единицу большее число общих ребер с деревом T, чем дерево S.

Далее приведенное рассуждение может быть повторено для случая, когда в качестве минимального покрывающего дерева берется не дерево S, а дерево S'. В результате будет построено новое минимальное покрывающее дерево S'', которое по сравнению с S' содержит на единицу большее число ребер, общих с деревом T. После конечного числа соответствующих преобразований минимального покрывающего дерева будет получено минимальное покрывающее дерево, полностью совпадающее с T. Это завершает обоснование алгоритма построения минимального покрывающего дерева.

Что касается алгоритма построения максимального покрывающего дерева, то его обоснование совпадает с приведенным выше обоснованием алгоритма построения минимального покрывающего дерева после повсеместной замены в последнем слова «минимальный» на слово «максимальный» и всех знаков неравенств на противоположные.

Построим минимальное покрывающее дерево в задаче из примера 2. Для данных, приведенных на рис. 2.1, упорядочение ребер по возрастанию соответствующих им весов приводит к последовательности: (a, b), (c, d), (d, e), (c, e), (a, c), (b, e), (b, d), (b, c), (a, d) и (a, e). Далее работа алгоритма построения минимального покрывающего дерева может быть представлена следующим образом.

Ребро	Цвет	Букет № 1	Букет № 2
		Сначала пуст	Сначала пуст
(a, b)	Голубой	a, b	Пуст
(c, d)	»	a, b	c, d
(d, e)	»	a, b	c, d, e
(c, e)	Оранжевый	a, b	c, d, e
(a, c)	Голубой	a, b, c, d, e	Пуст

Здесь алгоритм заканчивается, поскольку все вершины исходного графа попадают в один букет. При этом дуги (a, b), (c, d), (d, e) и (a, c) оказываются окрашенными в голубой цвет и, следовательно, образуют минимальное покрывающее дерево. (Найдите суммарный вес этого дерева.) Заметим, что дерево, образованное дугами (a, b), (c, d), (d, e) и (b, e), также является минимальным покрывающим деревом.

2.2. Алгоритм построения максимального ориентированного леса

В разд. 2.1 были рассмотрены алгоритмы построения деревьев, в которых не учитывалась ориентация дуг. В данном разделе будет изложен алгоритм Эдмондса [1], предназначенный для построения максимального ориентированного леса, т. е. леса, в котором учтена ориентация дуг.

Предположим, что управляющему сбытом некоторой межнациональной компании требуется передать какое-либо сообщение своим агентам на местах. Как это осуществить наилучшим образом? Один из возможных способов передачи сообщения мог бы сводиться просто к тому, что управляющий связывается с каждым из агентов по телефону. Однако такой способ мог бы оказаться слишком дорогостоящим. Дело в том, что управляющий может находиться в Чикаго, а агенты, например, в Лондоне и Париже. При этом для управляющего было бы разумнее с точки зрения затрат не звонить каждому из агентов, а позвонить одному из них, например в Лондон, с тем чтобы лондонский агент затем позвонил в Париж.

По-видимому, наилучшим решением в подобных ситуациях было бы указание каждому агенту определенного абонента, которому он должен передать сообщение, после того как сам его получит, т. е. создание определенной системы передачи сообщений.

Каким условиям должна удовлетворять такая система? Во-первых, любое лицо, входящее в систему, должно получать передаваемое сообщение ровно один раз (дублирование бессмысленно). Во-вторых, система должна обеспечивать наименьшие затраты при выполнении своих функциональных задач.

Построим граф, каждая вершина которого соответствует лицу, входящему в систему передачи сообщений, а каждая дуга представляет собой возможность передачи сообщения от одного лица другому.

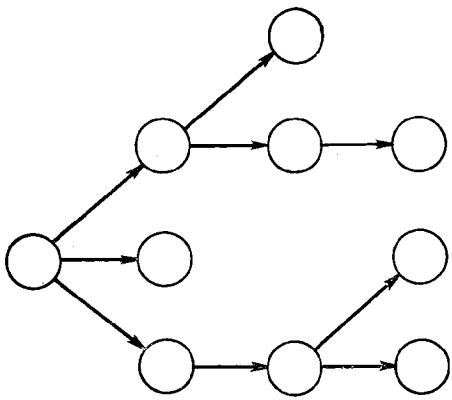


Рис. 2.3. Ориентированное дерево.

Ориентированное дерево определяется как дерево, в котором никакие две дуги не заходят в одну и ту же вершину (рис. 2.3). Заметим, что в ориентированном дереве из одной вершины могут исходить несколько дуг. Очевидно, ориентированное дерево

можно использовать для описания рассмотренной выше системы передачи сообщений. *Корнем* ориентированного дерева называется единственная его вершина, которая не имеет ни одной заходящей дуги.

Ориентированный лес определяется как обычный лес, но состоящий не из простых деревьев, а из деревьев ориентированных.

Покрывающим ориентированным деревом называется ориентированное дерево, являющееся одновременно и покрывающим деревом. *Покрывающим ориентированным лесом* называется ориентированный лес, который включает все вершины соответствующего графа.

Поставим в соответствие каждой дуге (x, y) рассматриваемого графа вес $a(x, y)$. Вес ориентированного леса (или ориентированного дерева) определяется как сумма весов дуг, входящих в этот лес (или дерево).

Максимальным ориентированным лесом графа G называется ориентированный лес графа G с максимально возможным весом. *Максимальным ориентированным деревом* графа G называется ориентированное дерево графа G с максимально возможным весом. *Минимальный ориентированный лес* и *минимальное ориентированное дерево* определяются аналогичным образом.

В данном разделе представлен принадлежащий Эдмондсу [1] алгоритм, называемый *алгоритмом построения максимального ориентированного леса*. Этот алгоритм строит соответствующий лес для любого графа G.

Как показывается ниже, алгоритм построения максимального ориентированного леса также может быть использован для поиска 1) минимального ориентированного леса, 2) максимального покрывающего дерева (при условии его существования для соответствующего графа), 3) минимального покрывающего ориентированного дерева (при условии его существования), 4) максимального покрывающего ориентированного дерева с корнем в заданной вершине (при условии его существования) и (5) минимального покрывающего ориентированного дерева с корнем в заданной вершине (при условии его существования).

Указанные выше задачи решаются с помощью алгоритма построения максимального ориентированного леса (далее этот алгоритм называется основным) следующим образом.

1. *Минимальный ориентированный лес* находится после изменения знака величины веса каждой дуги.

Максимальный ориентированный лес для новых весов дуг соответствует минимальному ориентированному лесу для исходных весов дуг.

2. *Максимальное покрывающее дерево* находится с учетом следующего свойства. Пусть положительная константа M прибавляется к весу каждой дуги. Очевидно, при увеличении M число дуг

в максимальном ориентированном лесе будет увеличиваться. Поскольку ни один ориентированный лес не может содержать дуг больше, чем покрывающее ориентированное дерево, то генерируемый основным алгоритмом максимальный ориентированный лес при достаточно большом M является покрывающим ориентированным деревом (если, конечно, хотя бы одно такое дерево содержится в соответствующем графе).

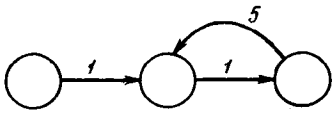


Рис. 2.4. Максимальный ориентированный лес имеет вес, равный 5; покрывающее ориентированное дерево имеет вес, равный 2.

Заметим, что даже если веса всех дуг положительны и граф содержит покрывающее ориентированное дерево, максимальный ориентированный лес не обязательно является покрывающим ориентированным деревом (см., например, рис. 2.4).

3. *Минимальное покрывающее дерево* (если оно существует) также может быть найдено с помощью основного алгоритма. Это осуществляется после изменения знака величины веса каждой дуги на отрицательный. После этого к весу каждой дуги следует добавить достаточно большую по величине положительную константу M. Как уже отмечалось в п. 2, в результате такого преобразования весов дуг основной алгоритм должен построить лес с максимально возможным числом дуг. Следовательно, алгоритм построит покрывающее ориентированное дерево (если, конечно, такое дерево существует). Более того, это будет минимальное покрывающее ориентированное дерево.

4 и 5. Основной алгоритм может быть также использован для поиска максимального (или минимального) *покрывающего ориентированного дерева с корнем в заданной вершине*, скажем в вершине a. (В примере с управляющим сбытом, приведенном в начале раздела, ищется минимальное покрывающее ориентированное дерево с корнем в вершине, соответствующей городу Чикаго). При этом к исходному графу добавляется дополнительная вершина a' и дуга (a', a) с произвольным весом. Если дополненный граф содержит покрывающее ориентированное дерево, то это дерево должно иметь корень в вершине a', поскольку она не имеет заходящих дуг. Любому покрывающему ориентированному дереву в дополненном графе соответствует покрывающее ориентированное дерево с корнем в вершине a в исходном графе. Применяя к дополненному графу основной алгоритм и изменяя веса дуг так же как в пп. 2 и 3, можно построить максимальное (или минимальное) покрывающее дерево, если, конечно, исходный граф содержит хотя бы одно покрывающее дерево.

В основном алгоритме используются два букета — букет вершин и букет дуг. Первый содержит только те вершины, которые

были просмотрены в процессе выполнения алгоритма, а второй — дуги, условно включенные в максимальный ориентированный лес[1]) в ходе процедуры просмотра вершин. Кстати, на протяжении всей процедуры дуги второго букета (точнее его текущего варианта)

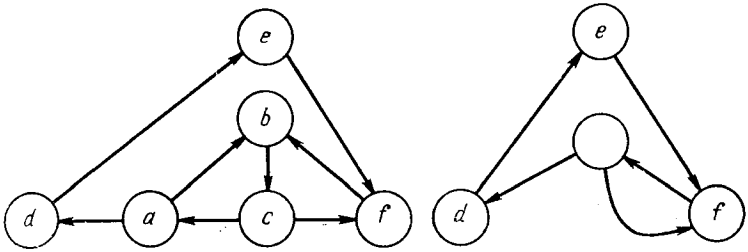


Рис. 2.5. Исходный граф и граф, полученный из исходного путем стягивания контура (a, b), (b, c), (c, a).

образуют лес для соответствующего графа. Перед началом процедуры оба букета пусты.

В основном алгоритме последовательно в произвольном порядке просматриваются вершины графа. Просмотр вершины состоит в том, что из дуг, заходящих в данную вершину, выбирается дуга с максимальным весом[2]). Если добавление этой дуги к дугам, уже вошедшим в букет, не нарушает свойства букета образовывать лес, то выбранная дуга вводится в букет. В противном случае (когда выбранная дуга образует *контур* с некоторыми дугами, ранее вошедшими в букет) формируется новый уменьшенный граф путем «стягивания» дуг и вершин выявленного контура в одну вершину (рис.2.5). В новом графе веса некоторых дуг соответствующим образом корректируются. Кроме того, для нового графа корректируется состав букета вершин и букета дуг: в них остаются только те элементы (вершины или дуги), которые присутствуют в новом графе. После проведенной корректировки процедура просмотра вершин продолжается аналогичным образом и заканчивается тогда, когда просмотрены все вершины исходного графа.

По окончании процедуры просмотра вершин дуги последнего сформированного букета образуют ориентированный лес в соответствующем графе. После этого осуществляется переход ко второму этапу алгоритма, состоящему в следующем. Граф, полученный по окончании первого этапа, расширяется путем замены фиктивной вершины контуром, который в нее стягивался при соответ-

[1]) Некоторые из условно включенных в искомый лес дуг на последнем этапе алгоритма могут быть исключены (см. далее). — *Прим. перев.*

[2]) Вершина, не имеющая заходящих дуг, может быть вообще исключена из рассмотрения. — *Прим. перев.*

ствующем преобразовании графов. В новый расширенный граф, а также в соответствующий ему букет включаются все дуги указанного контура, за исключением одной. Исключенная дуга выбирается так, чтобы дуги, составляющие новый букет, образовывали лес в соответствующем графе. Этот процесс последовательного расширения графа продолжается до тех пор, пока не будет восстановлен исходный граф. При этом оказывается, что дуги соответствующего букета образуют для исходного графа максимальный ориентированный лес.

Обозначим исходный граф, для которого отыскивается максимальный ориентированный лес, через G_0, а через G_1, G_2, ... и т. д. графы, которые последовательно получаются из графа G_0 в ходе процедуры стягивания выявляемых контуров. Букеты вершин и букеты дуг для соответствующих графов обозначим через V_0, V_1, V_2, ... и A_0, A_1, A_2, ... Теперь можно формально описать основной алгоритм.

Алгоритм построения максимального ориентированного леса

Первоначально все букеты V_0, V_1, ... и A_0, A_1 ... пустые. Положить $i = 0$.

Шаг 1. Если все вершины графа G_i входят в букет V_i, перейти к шагу 3. В противном случае выбрать в G_i любую вершину v, не входящую в букет V_i, и включить ее в V_i. Далее среди дуг, заходящих в вершину v, выбрать дугу γ с максимальным положительным весом. Если вершина v не имеет соответствующей дуги, вернуться к началу данного шага. В противном случае включить найденную дугу γ в букет A_i. Если после включения в A_i дуги γ букет A_i продолжает оставаться лесом в графе G_i, то снова вернуться к началу данного шага. В противном случае перейти к шагу 2.

Шаг 2. Поскольку добавление к букету A_i дуги γ приводит к тому, что A_i перестает быть лесом в G_i, дуга γ в совокупности с некоторыми другими дугами из A_i образует контур. Этот контур будет обозначаться через C_i.

Выявить контур C_i. Стянуть все дуги и вершины контура C_i в одну вершину, обозначаемую далее v_i. Полученный в результате такого стягивания граф считать графом G_{i+1}. Таким образом, любая дуга в графе G_i, инцидентная в точности одной вершине контура C_i, будет инцидентна вершине v_i в графе G_{i+1}, а вершинами графа G_{i+1}, помимо вершины v_i, являются все вершины графа G_i, не входящие в контур C_i.

Для всех дуг графа G_{i+1}, за исключением тех, которые являются заходящими в вершину v_i, оставить веса теми же, какими они были у этих дуг в графе G_i. Для каждой дуги (x, y) графа G_i, переходящей в графе G_{i+1} в дугу (x, v_i), положить

$$a(x, v_i) = a(x, y) + a(r, s) - a(t, y), \qquad (1)$$

где (r, s) — дуга, имеющая в контуре C_i минимальный вес, а (t, y) — дуга контура C_i, заходящая в вершину y. [Заметим здесь, что $a(r, s) \geqslant 0$, $a(t, y) \geqslant a(r, s)$ и $a(t, y) \geqslant a(x, y)$, поскольку в алгоритме при просмотре вершины y была выбрана дуга (t, y).]

Сформировать множество V_{i+1}, включив в него все вершины графа G_{i+1}, которые содержались в V_i (таким образом, $v_i \in V_i$). Сформировать множество A_{i+1}, включив в него все дуги графа G_{i+1}, которые содержались в A_i (таким образом, A_{i+1} содержит все дуги A_i, не входящие в контур C_i).

Увеличить i на единицу и перейти к шагу 1.

Шаг 3. Данный шаг выполняется только тогда, когда все вершины графа G_i попадают в букет V_i, а дуги букета A_i образуют в графе G_i ориентированный лес.

Если $i = 0$, закончить процедуру алгоритма: дуги букета A_0 образуют максимальный ориентированный лес для исходного графа G_0. Если $i \neq 0$, исследовать два возможных случая.

а) Вершина v_{i-1} является корнем некоторого ориентированного дерева в ориентированном лесе, образуемом дугами из A_i.

б) Вершина v_{i-1} не является корнем некоторого ориентированного дерева в лесе, образованном дугами из A_i.

Для случая (а) рассмотреть дуги из букета A_i в совокупности с дугами контура C_{i-1}. (Эти дуги содержат в точности один контур, а именно контур C_{i-1}.) Удалить из рассматриваемого множества дуг такую дугу контура C_{i-1}, которая имеет наименьший вес. Из полученного таким образом множества дуг составить новый букет A_{i-1}. Очевидно, дуги, нового букета A_{i-1}, так же как и дуги старого букета A_{i-1}, образуют в графе G_{i-1} ориентированный лес.

Для случая (б) в A_i имеется единственная дуга (x, v_{i-1}), заходящая в вершину v_{i-1}. Эта дуга соответствует некоторой дуге (x, y) в графе G_{i-1}, где y — некоторая вершина контура C_{i-1}, стянутого в вершину v_{i-1}.

Сформировать множество дуг, состоящее из дуг букета A_i и дуг контура C_{i-1}. Это множество дуг содержит в точности один контур, а именно контур C_{i-1} и ровно две дуги, заходящие в вершину y, а именно дугу (x, y) и соответствующую дугу в контуре C_{i-1}. Удалить из сформированного множества дуг дугу контура C_{i-1}, заходящую в вершину y. Из полученного таким образом множества дуг составить новый букет A_{i-1}. Очевидно, дуги нового букета образуют в графе G_{i-1} ориентированный лес.

Уменьшить i на единицу и повторить шаг 3.

Обоснование алгоритма построения максимального ориентированного леса. Будем рассматривать графы G_i, возникающие в ходе исполнения алгоритма, а также ориентированные леса этих графов, определяемые букетами A_i, которые формируются алгорит-

мом на шаге 3. Прежде всего покажем, что если ориентированный лес в графе G_t, определяемый букетом A_t, является максимальным, то то же самое относится к лесу в графе G_{t-1}, определяемому букетом A_{t-1}.

Для доказательства введем некоторые определения. Пусть G' представляет собой подграф графа G_{t-1}, включающий все дуги последнего, за исключением дуг, заходящих в вершины контура C_{t-1}. Пусть G'' представляет собой подграф графа G_{t-1}, включающий все дуги последнего, за исключением дуг, вошедших в G'. Обозначим через A'_{t-1} дуги из A_{t-1}, принадлежащие G', а через A''_{t-1} — дуги из A_{t-1}, принадлежащие G''. Очевидно, дуги, входящие в A'_{t-1} и A''_{t-1}, образуют ориентированные деревья соответственно в графах G' и G''.

Если ориентированный лес, определяемый A_{t-1}, не является максимальным в графе G_{t-1}, то в нем найдется некоторый ориентированный лес B с бо́льшим суммарным весом. Обозначим через B' совокупность дуг из B, принадлежащих G', а через B'' —совокупность дуг из B, принадлежащих G''. Поскольку суммарный вес дуг из B больше суммарного веса дуг букета A_{t-1}, то либо суммарный вес дуг из B' больше суммарного веса дуг из A'_{t-1}, либо суммарный вес дуг из B'' больше суммарного веса дуг из A''_{t-1}.

Сформулируем два следующих утверждения:

Утверждение 1. Ориентированный лес, определяемый букетом A''_{t-1}, является в графе G'' максимальным.

Утверждение 2. Суммарный вес дуг букета A'_{t-1} равен суммарному весу дуг из B'.

Если оба эти утверждения справедливы, то ориентированный лес, определяемый A_{t-1}, должен быть максимальным в графе G_{t-1}.

Отметим, что ориентированный лес, формируемый в алгоритме для графа G_i с наибольшим i (конечный граф), является в этом графе максимальным. Это определяется тем, что указанный лес включает дуги, каждая из которых имеет максимальный положительный вес среди дуг, заходящих в соответствующую вершину. Итак, если G_i — конечный граф, то лес, определяемый A_i, является в этом графе максимальным. Если утверждения 1 и 2 справедливы, формируемый на шаге 3 алгоритма букет A_{i-1} также определяет максимальный ориентированный лес в графе G_{i-1}. Повторяя это рассуждение, можно прийти к заключению, что при справедливости утверждений 1 и 2 лес, определяемый букетом A_0, является максимальным ориентированным лесом в исходном графе G_0.

Таким образом, для обоснования рассматриваемого алгоритма остается доказать справедливость утверждений 1 и 2.

Доказательство утверждения 1. Пусть контур C_{t-1} включает n вершин. В графе G'' для каждой из этих вершин найдется одна

заходящая дуга с положительным весом. (В противном случае в ходе выполнения алгоритма в графе G_{t-1} не должен был бы образоваться контур C_{t-1}.) Так как граф G'' состоит всего лишь из n вершин, имеющих заходящие дуги, то максимальный ориентированный лес для этого графа не может содержать более чем n дуг. Более того, вес любого ориентированного леса в графе G'' не может превысить суммарного веса дуг, составляющих контур C_{t-1}. Это обусловливается тем, что каждая из дуг контура C_{t-1} является для соответствующей вершины заходящей дугой с максимальным положительным весом. Но никакой лес не может содержать контур C_{t-1} целиком. Поэтому по крайней мере одна из n вершин контура C_{t-1}, скажем вершина y, в любом ориентированном лесе графа G'' либо вообще не имеет заходящей дуги, либо имеет заходящую дугу вида (x, y), где $x \notin C_{t-1}$.

Построим в графе G'' для каждой вершины z контура C_{t-1} ориентированный лес B_z. Построение осуществим следующим образом:

а) Включим в лес B_z все дуги контура C_{t-1}, за исключением дуги, заходящей в вершину z.

б) Включим в лес B_z любую из дуг (x, z), где $x \notin C_{t-1}$, с максимальным положительным весом.

Выберем в построенном множестве $\{B_z\}$ лес $B^*{}_z$ с максимальным весом. Из соотношения (1) следует, что $B^*{}_z$ совпадает с ориентированным лесом, определяемым букетом A''_{t-1}.

Рассмотрим любой ориентированный лес B_1 в графе G'', который не совпадает с лесом B_z. Если в B_1 отсутствует только одна дуга контура C_{t-1}, то B_1 не может быть максимальным ориентированным лесом в G'', так как, совпадая с одним из B_z, он не совпадает с $B^*{}_z$. Если в B_1 отсутствуют две или более дуг контура C_{t-1}, то каждая из этих дуг в B_1 либо заменена дугой, заходящей в ту же вершину и имеющей меньший вес, либо вообще не заменена какой-либо дугой. В обоих случаях имеет место уменьшение веса ориентированного леса по сравнению с $B^*{}_z$. Следовательно, B_1 не может быть максимальным ориентированным лесом в графе G''.

Таким образом, ориентированный лес, определяемый A''_{t-1}, является в графе G'' максимальным, и можно считать, что A''_{t-1} совпадает с B''. Это завершает доказательство утверждения 1.

Доказательство утверждения 2. Возможны два случая:

а) Ориентированный лес, определяемый букетом A_t, включает дугу (x, v_{t-1}), т. е. дугу, заходящую в вершину v_{t-1}.

б) Ориентированный лес, определяемый букетом A_t, не включает дуги, заходящей в вершину v_{t-1}.

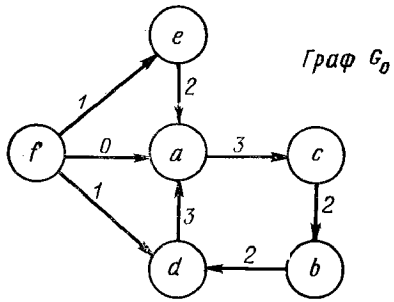
Рассмотрим каждый из этих случаев в отдельности.

Случай (а). По соответствующему предположению лес, определяемый букетом A_t, содержит дугу (x, v_{t-1}). Кроме того, в силу

утверждения 1 B'' совпадает с A''_{t-1}. Поэтому B'' содержит некоторую дугу (x, y), соответствующую дуге (x, v_{t-1}), где $x \notin C_{t-1}$, $y \in C_{t-1}$. Поскольку B является в графе G_{t-1} ориентированным лесом, то лес B' не может содержать путь, начинающийся в некоторой вершине контура C_{t-1} и заканчивающийся в вершине x. Причем среди ориентированных лесов графа G', не содержащих указанного пути, лес B' является максимальным.

Каждая дуга графа G' соответствует дуге графа G_t с таким же весом. Более того, каждому ориентированному лесу в графе G'' соответствует ориентированный лес в графе G_{t-1} такого же веса. Поскольку лес A_t содержит дугу (x, v_{t-1}), лес A'_{t-1} не должен содержать пути, соединяющего контур C_{t-1} с вершиной x. При этом если вес леса, определяемого A'_{t-1}, меньше веса леса B', то лес графа G_t, определяемый A_t, не может быть максимальным. Это противоречит основному предположению. Следовательно, лес, определяемый A'_{t-1}, имеет тот же самый вес, что и лес B'. Это доказывает утверждение для рассматриваемого случая.

Случай (б). Каждой дуге графа G' соответствует дуга графа G_t с таким же весом. Более того, любой ориентированный лес в графе G' соответствует ориентированному лесу в графе G_t с тем же весом. Поскольку в A_t нет дуги, заходящей в вершину v_{t-1}, каждой дуге



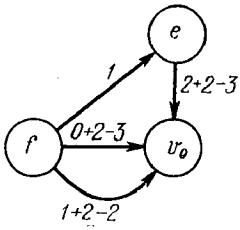
Просмотренные вершины	V_0	A_0
a	a	(d, a)
b	a, b	$(d,a), (c, b)$
c	a, b, c	$(d, a), (c, b), (a, c)$
d	a, b, c, d	$(d, a), (c, b), (a, c), (b, d)$

Рис. 2.6. Применение алгоритма построения максимального ориентированного леса.

в A_t соответствует дуга в A'_{t-1}. Следовательно, если бы ориентированный лес, определяемый A_{t-1}, не был в графе G' максимальным, то же самое относилось бы к ориентированному лесу в графе G_t, определяемому букетом A_t. Но это противоречит основному предположению.

Таким образом, ориентированный лес, определяемый A'_{t-1}, должен быть в графе G' максимальным, а значит, вес его должен совпадать с весом леса B'. Это завершает доказательство утверждения 2.

Пример применения основного алгоритма. Используем алгоритм для получения максимального ориентированного леса в графе, изображенном на рис. 2.6. Вес каждой дуги этого графа указан на рисунке рядом с соответ-



Граф G_1

Просмотренные вершины	V_1	A_1
e	e	(f, e)
f	e, f	(f, e)
v_0	e, f, v_0	(f, e), (e, v_0)

Рис. 2.7. Применение алгоритма построения максимального ориентированного леса (продолжение).

ствующей дугой. В процессе выполнения алгоритма вершины просматриваются для определенности в алфавитном порядке. Результаты просмотра первых четырех вершин a, b, c и d приведены на рис. 2.6.

После просмотра вершины d дуги букета A_0 уже не определяют ориентированный лес, поскольку они образуют контур (a, c), (c, b), (b, d), (d, a). На данном этапе выполнения алгоритма осуществляется стягивание этого контура в одну вершину v_0. Получаемый в результате такого стягивания граф G_1 изображен на рис. 2.7. Здесь же рядом с каждой дугой приведены их веса с указанием в соответствующих случаях формул пересчета. Граф G_1 состоит только из трех вершин e, f и v_0. Результаты просмотра этих вершин приведены на рис. 2.7.

После завершения просмотра всех трех вершин графа G_1 алгоритм формирует в графе G_1 максимальный ориентированный лес, состоящий из дуг (f, e) и (e, v_0). В ходе выполнения шага 3 алгоритма, исходным для которого является построенный в графе G_1 лес, вершина v_0 заменяется соответствующим контуром, при этом к дугам (f, e) и (e, v_0) добавляются дуги (a, c), (c, b), (b, d) и (d, a). Далее дуга (d, a) удаляется из рассматриваемой совокупности дуг, так что в ней остается лишь одна дуга, заходящая в вершину a, — дуга (e, a). Полученная совокупность дуг (f, e), (e, a), (a, c), (c, b) и (b, d) определяет максимальный ориентированный лес в графе G_0. Действительно, вес этого дерева равен $1 + 2 + 3 + 2 + 2 = 10$ и является максимально возможным.

Заметим, что построенный лес оказался в графе G_0 покрывающим ориентированным деревом с корнем в вершине f.

УПРАЖНЕНИЯ

1. Выполняя условия примера 2, постройте систему магистральных шоссейных дорог, имеющую минимальную стоимость.

2. Постройте минимальное покрывающее дерево для графа, изображенного на рис. 2.8.

3. Постройте для графа, изображенного на рис. 2.8, минимальное покрывающее дерево, включающее дуги (a, b) и (c, d).

4. Постройте максимальный ориентированный лес в графе, изображенном на рис. 2.8.

5. Для графа, изображенного на рис. 2.8, постройте покрывающее ориентированное дерево с корнем в вершине a.

6. Предположим, что после применения алгоритма построения максимального ориентированного дерева вы узнаете, что используемые в алгоритме веса всех дуг были занижены на 5 единиц. Можно ли в такой ситуации использовать уже полученные результаты или необходимо повторить процедуру алгоритма? Дайте ответ на этот вопрос и поясните его.

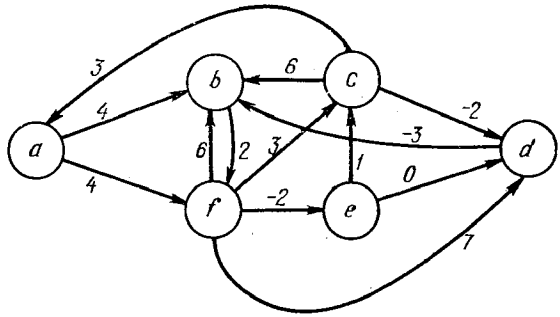


Рис. 2.8.

7. Рассмотрите следующий «поедающий» алгоритм: «Упорядочить дуги по их весам так, чтобы первой в этой упорядоченности была дуга с наибольшим весом. В качестве первой дуги для формируемого ориентированного леса выбрать первую дугу в полученной выше упорядоченности. Последовательно просматривать и далее дуги этой упорядоченности, оставляя те из них, которые вместе с уже выбранными дугами составляют ориентированный лес в рассматриваемом графе».

Покажите, что описанный алгоритм не всегда формирует максимальный ориентированный лес.

8. Уточните алгоритм построения максимального ориентированного леса для частного случая, в котором все дуги имеют одинаковые веса.

9. Пусть необходимо построить систему нефтепроводов, которые должны соединять семь нефтеочистительных заводов (Н1, Н2, Н3, Н4, Н5, Н6, Н7), принадлежащих некоторой компании, с портом (П), куда поступает импортируемая сырая нефть. Стоимость прокладки нефтепровода между любыми пунктами составляет 1000 долл. в расчете на одну милю плюс затраты в размере 4000 долл. на монтаж каждого участка нефтепровода длиной в одну милю. Расстояния между всеми парами пунктов задаются следующей таблицей:

	П	Н1	Н2	Н3	Н4	Н5	Н6	Н7
П	0	5	6	8	2	6	9	10
Н1		0	4	10	5	8	6	10
Н2			0	11	8	4	9	10
Н3				0	10	3	6	7
Н4					0	2	5	9
Н5						0	10	5
Н6							0	8
Н7								0

Найдите систему нефтепроводов минимальной суммарной стоимости.

10. Предположим, что данные упражнения 9 представляют собой ежегодные затраты на эксплуатацию нефтепровода между соответствующими пунктами. Найдите систему нефтепроводов, позволяющую осуществлять переброску нефти от порта ко всем нефтеперерабатывающим заводам и требующую минимальных годовых эксплуатационных затрат.

11. Рассмотрим следующую ситуацию. Пусть в некоторой научной лаборатории каждый исследователь получает ежедневно задание от сотрудника, занимающего в лаборатории более высокое служебное положение. Сотрудники лаборатории делятся на четыре категории: руководитель лаборатории, научный сотрудник, ассистент, лаборант. В каждой категории имеется соответственно 1, 4, 6, 5 и 8 человек. В силу различного опыта и образовательного уровня сотрудников лаборатории время, необходимое для объяснения задания одними сотрудниками другим, различно. Зададим соответствующие времена следующей таблицей:

От \ К	Научному сотруднику	Ассистенту	Лаборанту
Руководителя	5	9	11
Научного сотрудника		4	8
Ассистента			4

Найдите наилучший способ повседневного распределения заданий.

ЛИТЕРАТУРА

1. Edmonds J., Optimum Branchings, Mathematics of the Decision Sciences, Lectures in Applied Mathematics, Vol. 2, *AMS* (G. Dantzig and A. Veinott, eds.), pp. 346—361, 1968.
2. Kruskal J. B., On the Shortest Spanning Subtree of a Graph and the Traveling Selesman Problem, *Proc. AMS*, **7**, pp. 48—50, 1956.
3. Rosentiehl P., L'Arbre Minimum d'un Graphe, International Seminar on Graph Theory, Rome, July 1966.

ГЛАВА 3

Алгоритмы поиска путей

В данной главе описывается несколько алгоритмов поиска путей оптимальных в том или ином смысле. В разд. 3.1 описан алгоритм поиска кратчайшего пути, соединяющего две выделенные вершины исходного графа. В разд. 3.2 рассматриваются алгоритмы поиска кратчайшего пути между каждой парой вершин исходного графа. В разд. 3.3 обсуждаются алгоритмы поиска путей, являющихся первыми, вторыми, третьими и т. д. по степени близости к кратчайшим путям, определяемым алгоритмами предыдущих разделов. Наконец, в разд. 3.4 все рассматриваемые алгоритмы трактуются на более общем уровне.

3.1. Алгоритм поиска кратчайшего пути

Каждой дуге (x, y) исходного графа G поставим в соответствие число $a(x, y)$. Если в графе G отсутствует некоторая дуга (x, y), положим $a(x, y) = \infty$. Будем называть число $a(x\ y)$ длиной дуги (x, y), хотя $a(x, y)$ можно также интерпретировать как соответствующие затраты или соответствующий весовой коэффициент. Определим длину пути как сумму длин отдельных дуг, составляющих этот путь.

Для любых двух вершин s и t графа G могут существовать несколько путей, соединяющих вершину s с вершиной t. В данном разделе будет рассмотрен алгоритм, который определяет такой путь, ведущий из вершины s в вершину t, который имеет минимально возможную длину. Этот путь называется *кратчайшим путем* между вершинами c и t.

ПРИМЕР 1. Предположим, что вы хотите проехать из Бостона в Лос-Анджелес, используя магистральные шоссейные дороги, соединяющие различные штаты. Как выбрать кратчайший маршрут?

Постройте граф, вершины которого соответствуют точкам соединения рассматриваемых дорог. Пусть каждая дуга графа соответствует шоссейной дороге, соединяющей пункты, представленные соответствующими вершинами. Пусть длина дуги определяется длиной (в километрах) соответствующего участка дороги. Теперь задача поиска оптимального маршрута движения между Бостоном и Лос-Анджелесом может быть сведена к задаче отыскания в построенном графе кратчайшего пути между вершинами, соответствующими Бостону, откуда вы начинаете путешествие, и Лос-Анджелесу, где ваше путешествие заканчивается.

ПРИМЕР 2. Пассажир обращается за услугами к некоторой авиакомпании. Он желает попасть воздушным путем из Спрингфилда (шт. Иллинойс) в столицу Турции Анкару, проведя в воздухе как можно меньше времени, поскольку полет на самолете вызывает у него чувство страха. Какой маршрут в данном случае должна предложить авиакомпания пассажиру?

Чтобы решить этот вопрос, необходимо построить некоторый граф. Вершины этого графа представляют аэропорты, через которые может проходить маршрут полета от Спрингфилда до Анкары, а дуги соответствуют полетам между определенными аэропортами. Длиной каждой дуги следует считать время соответствующего полета. Теперь авиакомпании необходимо ознакомиться с последующим материалом данного раздела. После изучения этого материала станет понятно, как найти в построенном графе кратчайший путь, соединяющий вершины, соответствующие Спрингфилду и Анкаре.

ПРИМЕР 3. Предположим, что вам необходимо иметь в своем распоряжении автомобиль на протяжении ближайших пяти лет до выхода на пенсию. В настоящий момент имеется большой выбор автомобилей, которые вы можете либо купить, либо взять напрокат. Автомобили имеют различные сроки эксплуатации и разную стоимость. Каким образом вы должны сделать выбор в такой ситуации?

Представим моменты времени возможных сделок на ближайший пятилетний период, связанных с покупкой или использованием напрокат автомобиля, вершинами некоторого графа. (Для упрощения в качестве моментов времени сделок можно рассматривать лишь первые дни каждого месяца.) Изобразим в данном графе факт приобретения автомобиля дугой, соединяющей вершину, соответствующую моменту покупки или началу использования напрокат, с вершиной, соответствующей моменту окончания срока службы или использования напрокат автомобиля. Пусть длина каждой дуги построенного графа совпадает со стоимостью соответствующей сделки. Наилучшее сочетание решений, принятых в течение рассматриваемого пятилетнего периода, должно соответствовать кратчайшему пути в построенном графе между вершиной, представляющей начальный момент, и вершиной, отображающей момент ухода лица, принимающего решения, на пенсию.

ПРИМЕР 4. Представим себе следующую ситуацию. Коммивояжер планирует поездку из Бостона в Лос Анджелес. Цель поездки — посещение одного из основных клиентов. Коммивояжер собирается воспользоваться той же системой магистральных шоссейных дорог, которая фигурировала в примере 1. Помимо основной цели поездки коммивояжер планирует посещение и других пунктов, расположенных на пути из Бостона в Лос-Анджелес, где размещаются другие его клиенты. При этом коммивояжер приблизительно знает, какую сумму комиссионных он заработает после возможной встречи с каждым из клиентов. Какой маршрут поездки из Бостона в Лос-Анджелес следует выбрать коммивояжеру?

В рассматриваемом случае длиной каждой дуги в графе, представляющем рассматриваемую систему магистральных шоссейных дорог, следует считать ожидаемую «чистую стоимость» проезда (транспортные затраты за вычетом ожидаемой суммы комиссионных) по определенному участку данной системы дорог. Теперь можно сказать, что коммивояжер должен ехать по маршруту, соответствующему кратчайшему пути в построенном графе между вершинами «Бостон» и «Лос-Анджелес».

Обратите внимание на то, что в рассматриваемом примере стоимости, характеризующие дуги, будут отрицательными для тех участков маршрута, на которых коммивояжер ожидает получить прибыль, и положительными для тех участков, где он ожидает столкнуться с материальными потерями. (Заметим, что в примере 1 затраты были неотрицательными.) Далее мы увидим, что в каждой из этих двух ситуаций необходимо использовать различные алгоритмы решения задачи о кратчайшем пути.

ПРИМЕР 5. Мелкий вкладчик решает вопрос о целесообразном размещении своего капитала в следующем году. Он располагает некоторым набором вариантов возможного размещения (может положить деньги на сберегательную книжку, вложить средства в депозитный сертификат, облигации и т. п.). Для упрощения предположим, что вклады могут производиться или изыматься в начале каждого месяца. Поставим в соответствие началу каждого месяца вершину графа. В этом графе каждому вкладу поставлена в соответствие дуга (x, y) в том случае, если вклад был сделан в месяце x, а срок его истекает в месяце y. Заметим, что указанный граф не содержит контуров. Пусть длина каждой дуги равна взятому с обратным знаком доходу от соответствующего вклада. Наилучшая стратегия вложений капитала соответствует кратчайшему пути (в данном случае пути, имеющему максимальное отрицательное значение длины), соединяющему вершину начала и конца рассматриваемого года. Данный пример почти полностью аналогичен примеру 3, за исключением того, что в нем значения длин дуг графа являются не положительными, а отрицательными числами.

Описываемый в данном разделе алгоритм позволяет находить в графе кратчайший путь между двумя выделенными вершинами s и t при положительных длинах дуг. Этот алгоритм, предложенный в 1959 г. Дейкстрой [2], считается одним из наиболее эффективных алгоритмов решения задачи.

Главная идея, лежащая в основе алгоритма Дейкстры, предельно проста. Предположим, что нам известны m вершин, ближайших к вершине s (близость любой вершины x к вершине s определяется длиной кратчайшего пути, ведущего из s в x). Пусть также известны сами кратчайшие пути, соединяющие вершину s с выделенными m вершинами[1]). Покажем теперь, как может быть определена $(m + 1)$-я ближайшая к s вершина.

Окрасим вершину s и m ближайших к ней вершин[2]). Построим для каждой неокрашенной вершины y пути, непосредственно соединяющие с помощью дуг (x, y) каждую окрашенную вершину x с y. Выберем из этих путей кратчайший и будем считать его условно кратчайшим путем из вершины s в вершину y.

Какая же из неокрашенных вершин является $(m + 1)$-й ближайшей к s вершиной? Та, для которой условно кратчайший путь имеет наименьшую длину. Это обусловливается тем, что кратчайший путь из вершины s в $(m + 1)$-ю ближайшую вершину при положительном значении длин всех дуг должен содержать в качестве промежуточных лишь окрашенные вершины, т. е. вершины, входящие в число m вершин, ближайших к вершине s.

Итак, если известны m ближайших к s вершин, то $(m + 1)$-я ближайшая к s вершина может быть найдена так, как это описано выше. Начиная с $m = 0$, описанная процедура может повторяться

---

[1]) Кратчайшим путем из s в x считается нулевой путь, не содержащий дуг. Длина этого пути, естественно, принимается равной нулю.

[2]) В алгоритме, описанном в данном разделе, необходимо выделять вершины и дуги только одной категории, поэтому автор не указывает цвета окраски. — *Прим. перев.*

до тех пор, пока не будет получен кратчайший путь, ведущий из вершины s к вершине t.

Имея в виду приведенные соображения, мы можем теперь формально описать алгоритм Дейкстры.

Алгоритм Дейкстры поиска кратчайшего пути

Шаг 1. Перед началом выполнения алгоритма все вершины и дуги не окрашены. Каждой вершине в ходе выполнения алгоритма присваивается число $d(x)$, равное длине кратчайшего пути из s в x, включающего только окрашенные вершины.

Положить $d(s) = 0$ и $d(x) = \infty$ для всех x, отличных от s. Окрасить вершину s и положить $y = s$ (y — последняя из окрашенных вершин).

Шаг 2. Для каждой неокрашенной вершины x следующим образом пересчитать величину $d(x)$:

$$d(x) = \min\{d(x),\ d(y) + a(y, x)\}. \qquad (1)$$

Если $d(x) = \infty$ для всех неокрашенных вершин x, закончить процедуру алгоритма: в исходном графе отсутствуют пути из вершины s в неокрашенные вершины. В противном случае окрасить ту из вершин x, для которой величина $d(x)$ является наименьшей. Кроме того, окрасить дугу, ведущую в выбранную на данном шаге вершину x (для этой дуги достигался минимум в соответствии с выражением (1)). Положить $y = x$.

Шаг 3. Если $y = t$, закончить процедуру: кратчайший путь из вершины s в вершину t найден (это единственный путь из s в t, составленный из окрашенных дуг). В противном случае перейти к шагу 2.

Отметим, что каждый раз, когда окрашивается некоторая вершина (не считая вершины s), окрашивается и некоторая дуга, заходящая в данную вершину. Таким образом, на любом этапе алгоритма в каждую вершину заходит не более чем одна окрашенная дуга. Кроме того, окрашенные дуги не могут образовать в исходном графе цикл, так как в алгоритме не может окрашиваться дуга, концевые вершины которой уже окрашены. Следовательно, можно сделать вывод о том, что окрашенные дуги образуют в исходном графе ориентированное дерево с корнем в вершине s. Это дерево называется ориентированным деревом кратчайших путей. Единственный путь от вершины s до любой вершины x, принадлежащей дереву кратчайших путей, является кратчайшим путем между указанными вершинами.

Если кратчайшему пути из вершины s в вершину x в дереве кратчайших путей принадлежит вершина y, то часть этого пути, заключенная между x и y, является кратчайшим путем между этими вершинами. Действительно, если бы между x и y существовал

более короткий путь, то упомянутый выше путь между вершинами s и x не мог бы быть кратчайшим.

Поскольку на всех этапах алгоритма Дейкстры окрашенные дуги образуют в исходном графе ориентированное дерево, алгоритм можно рассматривать как процедуру наращивания ориентированного дерева с корнем в вершине s. Когда в этой процедуре наращивания достигается вершина t, процедура может быть остановлена.

Если бы мы хотели определить кратчайшие пути из вершины s во все вершины исходного графа, то процедуру наращивания дерева следовало бы продолжить до тех пор, пока все вершины графа не были бы включены в ориентированное дерево кратчайших путей. При этом для исходного графа было бы получено покрывающее ориентированное дерево (при условии, что в этом графе содержится хотя бы одно такое дерево). Итак, для того, чтобы описанный выше алгоритм позволял получать дерево кратчайших путей от вершины s до всех остальных вершин, его третий шаг должен быть скорректирован следующим образом: если все вершины оказываются окрашенными, закончить процедуру (для любой вершины x имеется единственный путь из s в x, состоящий из окрашенных дуг, и этот путь является кратчайшим путем между соответствующими вершинами); в противном случае перейти к шагу 2.

ПРИМЕР 6. Применим алгоритм Дейкстры к графу, изображенному на рис. 3.1, для нахождения в нем кратчайшего пути между вершинами s и t.

Перед первым выполнением шага 2 алгоритма окрашена только вершина s. Кроме того, $d(s) = 0$ и $d(x) = \infty$ для всех вершин x, не совпадающих с s.

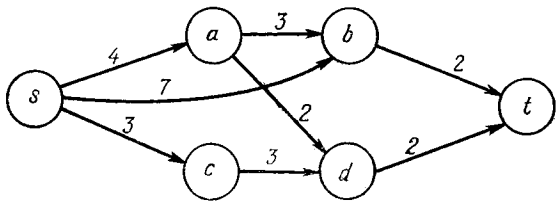


Рис. 3.1.

Шаг 2 ($y = s$).

$d(a) = \min\{d(a),\ d(s) + a(s,\ a)\} = \min\{\infty,\ 0 + 4\} = 4$

$d(b) = \min\{d(b),\ d(s) + a(s,\ b)\} = \min\{\infty,\ 0 + 7\} = 7$

$d(c) = \min\{d(c),\ d(s) + a(s,\ c)\} = \min\{\infty,\ 0 + 3\} = 3$

$d(d) = \min\{d(d),\ d(s) + a(s,\ d)\} = \min\{\infty,\ 0 + \infty\} = \infty$

$d(t) = \min\{d(t),\ d(s) + a(s,\ t)\} = \min\{\infty,\ 0 + \infty\} = \infty$

Поскольку величина $d(c) = 3$ является минимальной из величин $d(a)$, $d(b)$, $d(d)$, $d(t)$ и $d(t)$, то вершина c окрашивается. Так же окрашивается и дуга (s, c), которая и определяет величину $d(c)$. Текущее дерево кратчайших путей состоит из дуги s, c (рис. 3.2, $а$).

Шаг 3. Поскольку вершина t остается неокрашенной, осуществляется переход к шагу 2.

Шаг 2 ($y = c$).

$$d(a) = \min\{d(a),\ d(c) + a(c,\ a)\} = \min\{4,\ 3 + \infty\} = 4$$
$$d(b) = \min\{d(b),\ d(c) + a(c,\ b)\} = \min\{7,\ 3 + \infty\} = 7$$
$$d(d) = \min\{d(d),\ d(c) + a(c,\ d)\} = \min\{\infty,\ 3 + 3\} = 6$$
$$d(t) = \min\{d(t),\ d(c) + a(c,\ t)\} = \min\{\infty,\ 3 + \infty\} = \infty$$

Поскольку величина $d(a) = 4$ является минимальной из величин $d(a)$, $a(b)$, $d(d)$ и $d(t)$, то вершина a окрашивается. Так же окрашивается и дуга (s, a), которая определяет величину $d(a)$. Текущее дерево кратчайших путей теперь состоит из дуг (s, c) и (s, a) (рис. 3.2, $б$).

Шаг 3. Поскольку вершина t остается неокрашенной, осуществляется переход к шагу 2.

Шаг 2 ($y = a$).

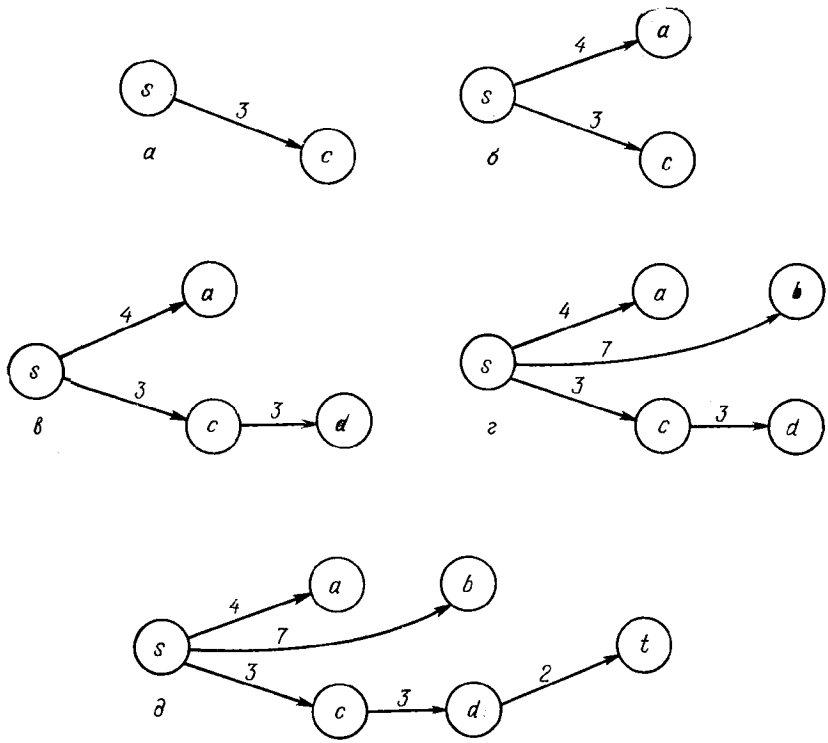


Рис. 3.2. Растущее ориентированное дерево кратчайших путей.

$d(b) = \min\{d(b),\ d(a) + a(a,\ b)\} = \min\{7,\ 4 + 3\} = 7$
$d(d) = \min\{d(d),\ d(a) + a(a,\ d)\} = \min\{6,\ 4 + 2\} = 6$
$d(t) = \min\{d(t),\ d(a) + a(a,\ t)\} = \min\{\infty,\ 4 + \infty\} = \infty$

Поскольку величина $d(d) = 6$ является минимальной из величин $d(b)$, $d(d)$ и $d(t)$, то вершина d окрашивается. Можно считать, что величину $d(d)$ определяют как дуга (c, d), так и дуга (a, d). Поэтому можно окрасить любую из этих дуг. Окрасим, например, дугу (c, d). Текущее дерево кратчайших путей состоит теперь из дуг (s, c), (s, a) и (s, d) (рис. 3.2, *в*).

Шаг 3. Поскольку вершина t остается неокрашенной, осуществляется переход к шагу 2.

Шаг 2 ($y = d$).

$d(b) = \min\{d(b),\ d(d) + a(d,\ b)\} = \min\{7,\ 6 + \infty\} = 7$
$d(t) = \min\{d(t),\ d(d) + a(d,\ t)\} = \min\{\infty,\ 6 + 2\} = 8$

Поскольку величина $d(b) = 7 = \min\{d(b), d(t)\}$ меньше величины $d(t)$ то вершина b окрашивается. Так же окрашивается и дуга (s, b), которая определяет величину $d(b)$. Текущее дерево кратчайших путей теперь состоит из дуг (s, c), (s, a), (c, d) и (s, b) (рис. 3.2, *г*).

Шаг 3. Поскольку вершина t остается неокрашенной, осуществляется переход к шагу *2*.

Шаг 2 ($y = b$).

$d(t) = \min\{d(t),\ d(b) + a(b,\ t)\} = \min\{8,\ 7 + 2\} = 8$

Итак, вершина t наконец-то окрашивается. Вместе с нею окрашивается дуга (d, t), определяющая величину $d(t)$. Окончательно построенное дерево кратчайших путей состоит из дуг (s, c), (s, a), (c, d), (s, b) и (d, t) (рис. 3.2, *д*).

Кратчайший путь, соединяющий вершину s с вершиной t, состоит из дуг (s, c), (c, d) и (d, t) и имеет длину $3 + 3 + 2 = 8$. Это не единственный кратчайший путь между вершинами s и t. Путь, состоящий из дуг (s, a), (a, d) и (d, t), имеет длину $4 + 2 + 2 = 8$ и также является кратчайшим путем между вершинами s и t. Кратчайший путь будет единственным в том случае, если в процедуре алгоритма ни разу не возникает неоднозначность в выборе окрашиваемой дуги.

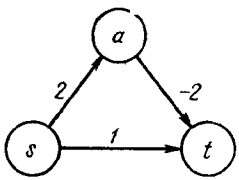


Рис. 3.3.

Исходное предположение в алгоритме Дейкстры состояло в неотрицательности длин дуг исходного графа. К каким результатам приводил бы алгоритм Дейкстры, если некоторые из длин дуг были бы отрицательными? Для примера рассмотрим граф, изображенный на рис. 3.3. В этом графе кратчайшим путем, соединяющим вершины s и t, является путь, состоящий из дуг (s, a) и (a, t). Длина этого пути равна $2 - 2 = 0$. Читатель может легко убедиться в том, что алгоритм Дейкстры, будучи применен к графу, изображенному на рис. 3.3, ошибочно укажет в качестве кратчайшего путь, состоящий из дуги (s, t). Таким образом, нет никакой гарантии, что алгоритм Дейкстры будет находить кратчайший путь в случаях, когда длины дуг могут быть отрицательными (как это, кстати, имеет место в примерах 4 и 5).

К счастью, алгоритм Дейкстры может быть обобщен на случай, когда некоторые из дуг имеют отрицательные длины. Идея соответствующего обобщения принадлежат Форду [5]. Необходимая модификация алгоритма Дейкстры состоит в следующем:

1. На шаге 2 алгоритма пересчет величин $d(x)$ с помощью соотношения (1) производится для всех вершин, а не только для неокрашенных. Следовательно, числа $d(x)$ могут уменьшаться как для неокрашенных, так и для окрашенных вершин.

2. Если для некоторой окрашенной вершины x происходит уменьшение величины $d(x)$, то с этой вершины и инцидентной ей окрашенной дуги окраска снимается.

3. Процедура алгоритма заканчивается только тогда, когда все вершины окрашены и когда после выполнения шага 2 ни одно из чисел $d(x)$ не меняется.

Обоснование модифицированного алгоритма Дейкстры (алгоритма Форда). Обоснование алгоритма Форда проведем по схеме доказательства от противного. При этом будем иметь в виду, что процедура алгоритма Форда может быть окончена только после того, как для всех x и y начинает выполняться соотношение

$$d(x) + a(x, y) \geqslant d(y). \qquad (2)$$

(Иначе с вершины y была бы обязательно снята окраска на итерации, непосредственно следующей за итерацией, на которой была окрашена вершина x.)

Итак, предположим, что после окончания процедуры алгоритма Форда величина $d(y)$ для некоторой вершины y не совпадает с длиной кратчайшего пути до этой вершины из вершины s (если в графе несколько таких вершин y, то выбирается та из них, кратчайший путь до которой из вершины s содержит наименьшее число дуг). Отметим, что если величина $d(z)$ для произвольной вершины z конечна, то она представляет собой длину некоторого пути, соединяющего вершины s и z. Поэтому по окончании процедуры алгоритма величина $d(y)$ должна совпадать с длиной некоторого пути из вершины s в вершину y, и в силу сделанного предположения эта величина должна превышать длину кратчайшего пути между указанными вершинами. Пусть вершина x является предпоследней вершиной кратчайшего пути между s и y. Если таких путей несколько, то в качестве x берется та вершина, кратчайший путь до которой от вершины s включает наименьшее число дуг. В силу выбора вершин y и x величина $d(x)$ должна совпадать с длиной кратчайшего пути из s в x, откуда $d(y) > d(x) + a(x, y)$. Однако последнее соотношение противоречит соотношению (2). Полученное противоречие завершает обоснование алгоритма Форда.

ПРИМЕР 7. Применим алгоритм Форда к графу, изображенному на рис. 3.3.

Шаг 1. Окрашивается вершина s. Полагается $d(s) = 0$, $d(a) = \infty$ и $d(t) = \infty$.

Шаг 2 ($y = s$).

$$d(a) = \min\{d(a),\ d(s) + a(s,\ a)\} = \min\{\infty,\ 0 + 2\} = 2$$
$$d(t) = \min\{d(t),\ d(s) + a(s,\ t)\} = \min\{\infty,\ 0 + 1\} = 1$$

Поскольку $d(t)$ меньше, чем $d(a)$, вершина t окрашивается. Вместе с ней окрашивается дуга $(s,\ t)$, которая и составляет текущее дерево кратчайших путей.

Шаг 3. Поскольку окрашены не все вершины, делается переход к шагу 2.

Шаг 2 ($y = t$).

Поскольку из вершины t не исходит ни одной дуги, все числа $d(x)$ остаются неизменными. Поэтому окрашивается вершина a и вместе с ней дуга $(s,\ a)$. Теперь дерево кратчайших путей состоит из дуг $(s,\ t)$ и $(s,\ a)$.

Шаг 3. Осуществляется возврат к шагу 2 с тем, чтобы попытаться уменьшить числа $d(x)$.

Шаг 2 ($y = a$). Определяются:

$$d(t) = \min\{d(t),\ d(a) + a(a,\ t)\} = \min\{1,\ 2 - 2\} = 0$$
$$d(s) = \min\{d(s),\ d(a) + a(a,\ s)\} = \min\{0,\ 2 + \infty\} = 0$$

Поскольку величина $d(t)$ уменьшается от 1 до 0, с вершины t и дуги $(s,\ t)$ снимается окраска. Теперь дерево кратчайших путей состоит только из дуги $(s,\ a)$.

В исходном графе остается неокрашенной только одна вершина — вершина t. Эта вершина окрашивается вместе с дугой $(a,\ t)$, поскольку в качестве y на данном шаге фигурирует a. Дерево кратчайших путей теперь включает дуги $(s,\ a)$ и $(a,\ t)$.

Шаг 3. Осуществляется очередной возврат к шагу 2.

Шаг 2 ($y = t$).

Поскольку из вершины t не исходит ни одной дуги, величины $d(x)$ не меняются. Неокрашенных вершин в исходном графе нет.

Шаг 3. Поскольку все вершины графа оказываются окрашенными и на предыдущем шаге алгоритма ни одну из величин $d(x)$ не удалось уменьшить, процедура алгоритма заканчивается. Кратчайший путь из вершины s в вершину t состоит из дуг $(s,\ a)$, $(a,\ t)$ и имеет длину $2 - 2 = 0$.

Во всех ли случаях алгоритм Форда решает задачу нахождения кратчайшего пути? Оказывается, не во всех. Алгоритм Форда не решает указанной задачи при наличии в исходном графе контура, имеющего отрицательную длину. Действительно, в этом случае, неограниченное число раз «повторяя» этот контур, можно получить путь со сколь угодно малой по величине длиной. Для коммивояжера из примера 4 контур отрицательной длины соответствует «прибыльному» замкнутому маршруту, повторяя который неограниченное число раз, коммивояжер может получить сколь угодно большую по величине прибыль[1]). В примере 5 соответствующий граф вообще не имеет контуров, поэтому в этом случае алгоритм Форда может применяться вне всяких сомнений.

А что делать в ситуации, когда нет сведений об отсутствии или наличии в рассматриваемом графе контуров отрицательной длины?

---

[1]) Следует, конечно, учитывать, условность данного примера. — *Прим. перев.*

В такой ситуации можно обычным образом применять алгоритм Форда, но в процессе его работы необходимо учитывать, сколько раз окрашиваются отдельные вершины. Как только число окрашиваний какой-либо вершины достигает величины N, где N — число вершин графа, процедуру алгоритма можно остановить: исходный граф содержит контур отрицательной длины. Если же этого не происходит, то процедура алгоритма Форда завершается за конечное число шагов, правильно решая исходную задачу.

Для обоснования приведенного правила предположим, что исходный граф не содержит контуров отрицательной длины. Тогда если у некоторой вершины x величина $d(x)$ приобретает окончательное значение (т. е. определяется длина кратчайшего пути из s в x), то в худшем случае каждая из оставшихся вершин может быть окрашена вновь (или впервые) только один раз, прежде чем будет окончательно окрашена одна из них. Таким образом, ни одна вершина не может быть окрашена более $(N-1)$ раза.

3.2. Алгоритмы поиска всех кратчайших путей

В предыдущем разделе была рассмотрена задача нахождения на графе кратчайшего пути из некоторой выделенной вершины до любой другой вершины. В данном разделе будет рассмотрена задача поиска на графе кратчайшего пути между каждой парой вершин. Конечно, эта более общая задача могла бы быть решена путем многократного применения алгоритма Дейкстры с последовательным выбором каждой вершины графа в качестве вершины s. Однако реализация соответствующей процедуры потребовала бы сравнительно больших вычислительных затрат. К счастью, существуют алгоритмы более эффективные, чем процедура многократного повторения алгоритма Дейкстры. Далее рассматриваются два весьма схожих алгоритма поиска на графе кратчайших путей между всеми парами вершин. Эти алгоритмы принадлежат Флойду [4] и Данцигу [1]. В обоих алгоритмах для длин дуг допускаются отрицательные значения, однако не допускается наличие контуров отрицательной длины.

ПРИМЕР 1. Предположим, что авиакомпания, фигурировавшая в примере 2 разд. 3.1, должна для многочисленных пассажиров ежедневно разрабатывать маршруты полетов между различными городами. Эта авиакомпания в целях экономии своих затрат стремится предоставлять пассажирам наиболее короткие маршруты. Поэтому ей хотелось бы заранее знать кратчайшие маршруты между каждой парой городов США, если, например, речь идет о полетах в пределах США.

Прежде чем представлять алгоритмы, необходимо ввести некоторые обозначения. Перенумеруем вершины исходного графа целыми числами от 1 до N. Обозначим через d_{ij}^{m} длину кратчайшего пути из вершины i в вершину j, который в качестве промежуточ-

ных может содержать только первые m вершин графа. (Напомним, что промежуточной вершиной пути является любая принадлежащая ему вершина, не совпадающая с его начальной или конечной вершинами.) Если между вершинами i и j не существует ни одного пути указанного типа, то условно будем считать, что $d_{ij}^m = \infty$. Из данного определения величин d_{ij}^m следует, что величина d_{ij}^0 представляет длину кратчайшего пути из вершины i в вершину j, не имеющего промежуточных вершин, т. е. длину кратчайшей дуги, соединяющей i с j (если такие дуги присутствуют в графе). Для любой вершины i положим $d_{ii}^0 = 0$. Отметим далее, что величина d_{ij}^n представляет длину кратчайшего пути между вершинами i и j.

Обозначим через D^m матрицу размера $N \times N$, элемент (i, j) которой совпадает с d_{ij}^m. Если в исходном графе нам известна длина каждой дуги, то мы можем сформировать матрицу D^0. Наша цель состоит в определении матрицы D^N, представляющей кратчайшие пути между всеми вершинами рассматриваемого графа.

В алгоритме Флойда в качестве исходной выступает матрица D^0. Вначале из этой матрицы вычисляется матрица D^1. Затем по матрице D^1 вычисляется матрица D^2 и т. д. Процесс повторяется до тех пор, пока по матрице D^{N-1} не будет вычислена матрица D^N.

Рассмотрим основную идею, лежащую в основе алгоритма Флойда. Предположим, что нам известны:

а) кратчайший путь из вершины i в вершину m, в котором в качестве промежуточных допускается использование только первых $(m-1)$ вершин;

б) кратчайший путь из вершины m в вершину j, в котором в качестве промежуточных допускается использование только первых $(m-1)$ вершин;

в) кратчайший путь из вершины i в вершину j, в котором в качестве промежуточных допускается использование только первых $(m-1)$ вершин.

Поскольку по предположению исходный граф не может содержать контуров отрицательной длины, один из двух путей — путь, совпадающий с представленным в п. «в», или путь, являющийся объединением путей из пп. «а» и «б», — должен быть кратчайшим путем из вершины i в вершину j, в котором в качестве промежуточных допускается использование только первых m вершин. Таким образом,

$$d_{ij}^m = \min \{d_{im}^{m-1} + d_{mj}^{m-1},\ d_{ij}^{m-1}\}. \qquad (3)$$

Из соотношения (3) видно, что для вычисления элементов матрицы D^m необходимо располагать лишь элементами матрицы D^{m-1}. Более того, соответствующие вычисления могут быть проведены без обращения к исходному гр афу. Теперь мы в состоянии дать

формальное описание алгоритма Флойда для нахождения на графе кратчайших путей между всеми парами вершин.

Алгоритм Флойда

Шаг 1. Перенумеровать вершины исходного графа целыми числами от 1 до N. Определить матрицу D^0, задав величину каждого ее элемента (i, j) равной длине кратчайшей дуги, соединяющей вершину i с вершиной j. Если в исходном графе указанные вершины не соединяются дугами, положить $d_{ij}^0 = \infty$. Кроме того, для всех i положить $d_{ii}^0 = 0$.

Шаг 2. Для целого m, последовательно принимающего значения 1, 2, ..., N, определить по величинам элементов матрицы D^{m-1} величины элементов матрицы D^m, используя рекурсивное соотношение (3), т. е. соотношение $d_{ij}^m = \min\{d_{im}^{m-1} + d_{mj}^{m-1}, d_{ij}^{m-1}\}$. При определении величины каждого элемента матрицы D^m фиксировать соответствующий кратчайший путь.

По окончании данной процедуры величина элемента (i, j) матрицы D^N определяет длину кратчайшего пути, ведущего из вершины i в вершину j.

То, что описанный алгоритм действительно находит кратчайшие пути, может быть индуктивно доказано на основе следующего факта. Длина кратчайшего пути из вершины i в вершину j, допускающего использование в качестве промежуточных первых m вершин, должна быть не больше длины кратчайшего пути из i в j, допускающего использование в качестве промежуточных первых $(m-1)$ вершин, и не больше длины кратчайшего пути из i в j, допускающего использование в качестве промежуточных первых $(m-1)$ вершин и обязательно — вершины m.

Отметим, что для всех i и m должно быть $d_{ii}^m = 0$. Поэтому нет необходимости в вычислении диагональных элементов матриц $D^1, D^2, \ldots, D^N$. Кроме того, для всех $i = 1, 2, \ldots, N$ имеют место соотношения $d_{im}^{m-1} = d_{im}^m$ и $d_{m,i}^{m-1} = d_{mi}^m$. Эти соотношения обусловливаются тем, что вершина m в отсутствие контуров отрицательной длины не может выступать в качестве промежуточной в любых кратчайших путях, которые начинаются или заканчиваются в самой вершине m. Следовательно, при определении матрицы D^m нет необходимости в пересчете элементов m-й строки и m-го столбца матрицы D^{m-1}. Таким образом, в матрице D^m по формуле (3) необходимо считать лишь величины $(N-1)(N-2)$ элементов, в число которых не входят диагональные элементы, а также элементы из m-й строки и m-го столбца.

ПРИМЕР 2. (*Применение алгоритма Флойда*). Для графа, изображенного на рис. 3.4, матрица D^0, составленная из длин дуг графа, такова:

$$D^0 = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 7 & \infty \\ 6 & 5 & 0 & 2 \\ 1 & \infty & 4 & 0 \end{bmatrix}$$

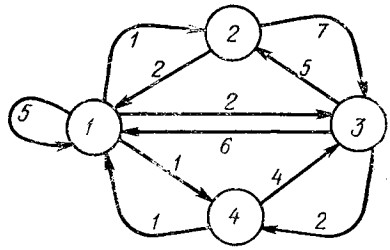


Рис. 3.4.

Величины элементов матрицы D^1 и соответствующие им кратчайшие пути определяются следующим образом:

$d^1_{ij} = \min \{ d^0_{i1} + d^0_{1j},\ d^0_{1j} \}$	Соответствующие пути
$d^1_{11} = d^0_{11} = 0$	
$d^1_{12} = d^0_{12} = 1$	(1,2)
$d^1_{13} = d^0_{13} = 2$	(1,3)
$d^1_{14} = d^0_{14} = 1$	(1,4)
$d^1_{21} = d^0_{21} = 2$	(2,1)
$d_{22} = 0$	
$d^1_{23} = \min \{ d^0_{21} + d^0_{13},\ d^0_{23} \} = \min \{2 + 2, 7\} = 4$	(2,1), (1,3)
$d^1_{24} = \min \{ d^0_{21} + d^0_{14},\ d^0_{24} \} = \min \{2 + 1, \infty\} = 3$	(2,1), (1,4)
$d^1_{31} = d^0_{31} = 6$	(3,1)
$d^1_{32} = \min \{ d^0_{31} + d^0_{12},\ d^0_{32} \} = \min \{6 + 1, 5\} = 5$	(3,2)
$d^1_{33} = 0$	
$d^1_{34} = \min \{ d^0_{31} + d^0_{14},\ d^0_{34} \} = \min \{6 + 1, 2\} = 2$	(3,4)
$d^1_{41} = d^0_{41} = 1$	(4,1)
$d^1_{42} = \min \{ d^0_{41} + d^0_{12},\ d^0_{42} \} = \min \{1 + 1, \infty\} = 2$	(4,1), (1,2)
$d^1_{43} = \min \{ d^0_{41} + d^0_{13},\ d^0_{43} \} = \min \{1 + 2, 4\} = 3$	(4,1), (1,3)
$d^1_{44} = 0$	

Аналогичным образом могут быть определены величины элементов матриц D^2, D^3 и D^4 и соответствующие им кратчайшие пути. Полученные результаты приводятся ниже.

Матрица D^2:

$$D^2 = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 6 & 5 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{bmatrix}$$

Кратчайшие пути для элементов матрицы D^2:

	(1,2)	(1,3)	(1,4)
(2,1)		(2,1), (1,3)	(2,1), (1,4)
(3,1)	(3,2)		(3,4)
(4,1)	(4,1), (1,2)	(4,1), (1,3)	

Матрица D^3:

$$D^3 = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 6 & 5 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{bmatrix}$$

Кратчайшие пути для элементов матрицы D^3:

	(1,2)	(1,3)	(1,4)
(2,1)		(2,1) (1,3)	(2,1), (1,4)
(3,1)	(3,2)		(3,4)
(4,1)	(4,1), (1,2)	(4,1), (1,3)	

Матрица D^4:

$$D^4 = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 3 & 4 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{bmatrix}$$

Кратчайшие пути для элементов матрицы D^4

	(1,2)	(1,3)	(1,4)
(2,1)		(2,1), (1,3)	(2,1), (1,4)
(3,4), (4,1)	(3,4), (4,1), (1,2)		(3,4)
(4,1)	(4,1), (1,2)	(4,1), (1,3)	

Заметим, что при произвольной нумерации вершин исходного графа в процессе выполнения алгоритма кратчайшие пути будут отыскиваться на более ранних стадиях для тех вершин, которые, имея близкие номера, являются «близкими» и по длинам соответствующих кратчайших путей.

В представленном выше численном примере кратчайшие пути между соответствующими вершинами формировались по мере выполнения процедуры алгоритма. Очевидно, для задач реальных размерностей такой способ формирования кратчайших путей яв-

ляется практически малопригодным. Следовательно, необходимо разработать более эффективный способ определения дуг, составляющих кратчайший путь.

Введем в рассмотрение предпоследние вершины путей. Пусть p_{ij} обозначает предпоследнюю вершину кратчайшего пути, соединяющего вершину i с вершиной j. (Если между указанными вершинами имеется несколько соединяющих их кратчайших путей, то для пары (i, j) может существовать несколько предпоследних вершин. В этом случае через p_{ij} следовало бы обозначать множество всех этих вершин. Однако если нас интересует только один кратчайший путь, то можно ограничить рассмотрение лишь одной из вершин, составляющих множество p_{ij}.) Если p_{ij} известно для каждой пары вершин i и j, то все промежуточные вершины кратчайшего пути из i в j могут быть определены следующим образом. Пусть предпоследней вершиной искомого пути является вершина k, т. е. $p_{ij} = k$. Тогда вторая от конца вершина на этом пути является предпоследней вершиной кратчайшего пути из i в k, т. е. совпадает с p_{ik}. Данную процедуру можно повторять и далее до тех пор, пока не будет пройден в обратном направлении весь кратчайший путь из вершины i в вершину j. Таким образом, для того, чтобы определить кратчайшие пути между всеми парами вершин, необходимо располагать лишь элементами p_{ij} для всех пар вершин (i, j).

Существуют два способа определения p_{ij} — *встроенный* (tentative) и *внешний* (terminal). Опишем каждый из них в отдельности.

1. *Встроенный способ.* Перед началом работы алгоритма Флойда (точнее, перед шагом 2) в качестве p_{ij} принять элемент i (это проделать для каждого j). Далее в процессе выполнения алгоритма при использовании соотношения (3) каждый раз фиксировать, какая из величин $(d_{i,m}^{m-1} + d_{m,j}^{m-1})$ или $d_{i,j}^{m-1}$ меньше. Как только меньшей оказывается первая из этих величин, положить p_{ij} равным m. В противном случае оставить p_{ij} неизменным. (Если оказывается, что величины, входящие в правую часть соотношения (3), одинаковы, то можно либо не менять p_{ij}, либо принять его равным m.)

После окончания выполнения алгоритма элементы p_{ij}, сформированные данным способом, действительно определяют предпоследние вершины кратчайших путей, ведущих из вершины i в вершину j.

2. *Внешний способ.* Считая работу алгоритма Флойда законченной, а значит, и зная матрицу D^N, определить каждое p_{ij}, положив его равным любому k, для которого $d_{ik}^N + d_{kj}^0 = d_{ij}^N$; при этом для формирования p_{ij} требуются только матрицы D^0 и D^N.

Отметим, что в ситуации, когда алгоритм Флойда[1]) уже выпол-

[1]) Имеется в виду та основная модификация алгоритма, в которой вычисляются только длины кратчайших путей. — *Прим. перев.*

нен, единственное, что остается сделать для определения кратчайших путей, — это воспользоваться внешним способом формирования p_{ij}. Если же заранее известно, что необходимо определить не только длины кратчайших путей, но и сами пути, то, безусловно, лучше применять встроенный способ, поскольку реализация его в рамках основного алгоритма лишь незначительно увеличивает объем вычислений.

Еще один алгоритм поиска на графе кратчайших путей между всеми парами вершин был предложен Данцигом. Алгоритм Данцига весьма близок к алгоритму Флойда и отличается от последнего лишь иным порядком выполнения тех же самых операций.

Итак, рассмотрим алгоритм Данцига. Снова перенумеруем вершины исходного графа целыми числами от 1 до N и обозначим через d_{ij}^m длину кратчайшего пути из вершины i в вершину j, в котором допускается использование в качестве промежуточных m первых вершин графа. Пусть теперь в отличие от алгоритма Флойда матрица D^m, состоящая из величин d_{ij}^m, при каждом $m = 1, 2, \ldots, N$ имеет размерность не $N \times N$, а $m \times m$. Так же, как и раньше, требуется определить матрицу D^N, элемент (i, j) которой определяет длину кратчайшего пути из вершины i в вершину j. Как и в алгоритме Флойда, в алгоритме Данцига матрица D^1 определяется из матрицы D^0, матрица D^2 из матрицы D^1 и т. д. Наконец, матрица D^N определяется[1] из матрицы D^{N-1}.

В чем же идея алгоритма Данцига? Во-первых, отметим, что каждая новая вычисляемая матрица D^m содержит на одну строку и на один столбец больше, чем ее предшественница, матрица D^{m-1}. Элементы матрицы D^m, не входящие в последние строку и столбец (число таких элементов равно $(m-1)^2$, определяются точно так же, как в алгоритме Флойда. Что же касается остальных элементов d_{ij}^m, где $i = m$ или $j = m$, то они определяются с учетом приводимых ниже соображений. Кратчайший путь из вершины i в вершину m (или наоборот), в котором допускается использование в качестве промежуточных только первых m вершин графа, не может иметь среди промежуточных вершину m, поскольку любой контур в исходном графе имеет неотрицательную длину. В силу данного обстоятельства такой кратчайший путь из вершины i в вершину m должен иметь своей первой частью кратчайший путь из вершины i в некоторую вершину j ($j < m$), который допускает использование в качестве промежуточных только $(m-1)$ первых вершин графа, а второй частью — кратчайшую дугу, ведущую из вершины j в вершину m (конечно, следует рассматривать только

[1] На самом деле при определении любой из матриц D^k при $k > 1$ в обоих алгоритмах используются не только элементы матрицы D^{k-1}, но и элементы матрицы D^0, определяемой так же, как в алгоритме Флойда. — *Прим. перев.*

такие вершины j, для которых имеется хотя бы одна дуга, ведущая из j в m). Аналогично кратчайший путь из вершины m в вершину i, в котором допускается использование в качестве промежуточных только m первых вершин графа, должен иметь своей первой частью кратчайшую дугу, ведущую из вершины m в некоторую вершину $j (j < m)$, а второй частью — кратчайший путь из вершины j в вершину i, который допускает использование в качестве промежуточных только $(m-1)$ первых вершин (конечно, следует рассматривать только такие вершины j, для которых имеется хотя бы одна дуга, ведущая из m в j). Наконец, отметим, что величины d^m_{mm} должны полагаться равными нулю.

С учетом приведенных соображений мы можем теперь формально описать алгоритм Данцига.

Алгоритм Данцига поиска всех кратчайших путей

Шаг 1. Перенумеровать вершины исходного графа целыми числами от 1 до N. Сформировать матрицу D^0 (размерностью $N \times N$), каждый элемент i, j которой $d_{ij}{}^0$ определяет длину кратчайшей дуги, ведущей из вершины i в вершину j. В отсутствие такой дуги положить $d^0{}_{ij} = \infty$.

Шаг 2. Здесь через D^m обозначается матрица размерностью $m \times m$ с элементами d^m_{ij}, $m = 1, 2, \ldots, N$. Последовательно определить элементы матрицы D^m из элементов матрицы D^{m-1} для m, принимающего значения 1, 2, .., N:

$$d^m_{mj} = \min_{i=1, 2, \ldots, m-1} \{d^0_{mi} + d^{m-1}_{ij}\} \ (j = 1, 2, \ldots, m-1), \tag{4}$$

$$d^m_{im} = \min_{j=1, 2, \ldots, m-1} \{d^{m-1}_{ij} + d^0_{jm}\} \ (i = 1, 2, \ldots, m-1), \tag{5}$$

$$d^m_{ij} = \min \{d^m_{im} + d^m_{mj}, \ d^{m-1}_{ij}\} \ (i, j = 1, 2, \ldots, m-1). \tag{6}$$

Кроме того, для всех i и m положить

$$d^m_{ii} = 0. \tag{7}$$

Кратчайшие пути, длины которых определяются величинами d^N_{ij} элементов матрицы D^N, могут быть определены аналогично тому, как это делалось в алгоритме Флойда.

Сколько операций выполняется в алгоритме Данцига? Чтобы ответить на этот вопрос, заметим, что в алгоритме Данцига выполняются по существу те же самые операции, что и в алгоритме Флойда, только в другом порядке. Действительно, уравнение (3), используемое в алгоритме Флойда, просто совпадает с уравнением 6) алгоритма Данцига. Уравнения (4) и (5), используемые в алго-

ритме Данцига, просто $(m-1)$ раз «повторяют» уравнение (3) из алгоритма Флойда. Таким образом, оба алгоритма включают одно и то же число операций.

ПРИМЕР 3. (*Применение алгоритма Данцига*). Применим алгоритм Данцига для поиска кратчайших путей между всеми парами вершин графа, изображенного на рис. 3.4. Матрица D^0, составленная из длин дуг этого графа, такова:

$$D^0 = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 7 & \infty \\ 6 & 5 & 0 & 2 \\ 1 & \infty & 4 & 0 \end{bmatrix}$$

Очевидно, $D^1 = [d_{11}^1] = [0]$. Величины элементов матрицы D^2 находятся следующим образом:

Элементы	Соответствующий путь
$d_{11}^2 = 0$	
$d_{12}^2 = \min\{d_{11}^1 + d_{12}^0\} = 0 + 1 = 1$	(1,2)
$d_{22}^2 = 0$	
$d_{21}^2 = \min\{d_{21}^0 + d_{11}^1\} = 2 + 0 = 2$	(2,1)

Заметим, что приведенные результаты идентичны данным, расположенным в левой верхней подматрице размерностью 2×2 матрицы D^2, которая была получена в примере 2 с помощью алгоритма Флойда. Величины элементов матрицы D^3 находятся следующим образом:

Элементы	Соответствующий путь
$d_{13}^3 = \min\{d_{11}^2 + d_{13}^0,\ d_{12}^2 + d_{23}^0\} = \min\{0+2,\ 1+7\} = 2$	(1,3)
$d_{23}^3 = \min\{d_{21}^2 + d_{13}^0,\ d_{22}^2 + d_{23}^0\} = \min\{2+2,\ 0+7\} = 4$	(2,1), (1, 3)
$d_{31}^3 = \min\{d_{31}^0 + d_{11}^2,\ d_{32}^0 + d_{21}^2\} = \min\{6+0,\ 5+2\} = 6$	(3,1)
$d_{32}^3 = \min\{d_{32}^0 + d_{22}^0,\ d_{31}^0 + d_{12}^2\} = \min\{5+0,\ 6+1\} = 5$	(3,2)
$d_{12}^3 = \min\{d_{12}^2,\ d_{13}^3 + d_{32}^3\} = \min\{1,\ 2+5\} = 1$	(1,2)
$d_{21}^3 = \min\{d_{21}^2,\ d_{23}^3 + d_{31}^3\} = \min\{2,\ 4+6\} = 2$	(2,1)
$d_{11}^3 = d_{22}^3 = d_{33}^3 = 0$	

Таким образом,

$$D^3 = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 4 \\ 6 & 5 & 0 \end{bmatrix}$$

Соответствующая матрица кратчайших путей имеет вид

$$\begin{bmatrix} & (1,2) & (1,3) \\ (2,1) & & (2,1),\ (1,3) \\ (3,1) & (3,2) & \end{bmatrix}$$

Заметим, что вычисленная выше матрица D^3 совпадает с левой верхней подматрицей размерностью 3×3 матрицы D^3, полученной в примере 2 с помощью алгоритма Флойда.

Для завершения расчетов заинтересованному читателю остается повторить процедуру поиска D^m для $m = 4$. При этом он сможет убедиться, что полученная им матрица D^4 полностью совпадает с матрицей D^4, полученной в примере 2 при использовании алгоритма Флойда.

Поскольку внешний способ формирования кратчайших путей использует матрицы D^0 и D^N, он одинаково применим как для алгоритма Флойда, так и для алгоритма Данцига. Что же касается встроенного способа, то его реализация в алгоритме Данцига несколько отлична от соответствующей реализации для алгоритма Флойда. В частности, при использовании соотношения (4) p_{mj} определяется тем, какая из сравниваемых в соотношении (4) величин меньше[1]). Аналогично определяется p_{im} при использовании соотношения (5)[2]). При использовании в алгоритме Данцига соотношения (6) p_{ij} определяются так же, как и при использовании в алгоритме Флойда соотношения (3). Точнее, полагается $p_{ij} = m$, если из двух величин $(d^m_{im} + d^m_{mj})$ и d^{m-1}_{ij} меньшей оказывается первая и p_{ij} остается неизменным в противном случае.

Заканчивая на этом рассмотрение задачи поиска на графе кратчайших путей, следует с удовлетворением отметить, что имеется определенная возможность выбора среди алгоритмов решения этой задачи. Действительно, в данном случае можно было бы применить алгоритм Флойда (или эквивалентный ему алгоритм Данцига), а можно было воспользоваться алгоритмом Дейкстры с многократным повтороением последнего при выборе каждой вершины графа в качестве начальной. Таким образом, возникает необходимость в сравнении объемов вычислений по каждому из алгоритмов. Осуществляемая при этом процедура оценки числа операций, выполняемых в том или ином алгоритме, получила название *анализа вычислительной сложности.*

---

[1]) Точнее, p_{mj} совпадает с p_{ij}, где i — индекс, для которого в соотношении (4) достигается минимум. — *Прим. перев.*

[2]) Точнее, p_{im} совпадает с индексом j, для которого в соотношении (5) достигается минимум. — *Прим. перев.*

Анализ вычислительной сложности достаточно просто проводить для таких алгоритмов, в которых число выполняемых операций практически неизменно. Именно такими алгоритмами при фиксированном исходном графе являются алгоритмы Дейкстры, Флойда и Данцига. Однако существуют алгоритмы, точное число операций в которых не может быть определено заранее. Например, нельзя заранее определить число операций алгоритма Форда, поскольку нельзя до выполнения алгоритма точно указать, сколько раз будет окрашиваться каждая вершина графа. Для алгоритмов типа алгоритма Форда обычно при анализе вычислительной сложности определяют верхнюю границу возможного числа операций.

Как видно из описания алгоритмов поиска кратчайших путей, в основном они состоят из операций двух типов: операции сложения и операции сравнения по минимуму[1]). При анализе вычислительной сложности любого из этих алгоритмов необходимо каким-либо образом выявить соотношение между вычислительными затратами на выполнение операции сложения и сравнения. Конечно, это соотношение определяется используемыми вычислительными средствами (автоматическими или ручными). Однако для большего удобства мы будем предполагать, что для выполнения обеих операций требуется одинаковое время.

Перейдем теперь к определению числа операций в алгоритмах Флойда (Данцига), Дейкстры и Форда. В алгоритме Флойда вычисляются N матриц D^1, D^2, ..., D^N, каждая из которых состоит из N^2 элементов. Следовательно, в алгоритме Флойда необходимо вычислять N^3 элементов. Каждое такое вычисление осуществляется с помощью соотношения (3) и требует выполнения одной операции сложения и одной операции сравнения[2]). Следовательно, в алгоритме Флойда выполняется N^3 сложений и N^3 сравнений. (Строго говоря, указанные числа несколько превышают действительные показатели, поскольку некоторые элементы матрицы D^i могут быть непосредственно приравнены соответствующим элементам из матрицы D^{i-1} без проведения вычислений с помощью соотношения (3). Это относится к элементам i-й строки и i-го столбца матриц D^i и D^{i-1}, являющихся в этих матрицах одинаковыми.) Итак, общее количество операций, выполняемых в алгоритме Флойда, пропорционально $2N^3$. Используя более строгую терминологию, этот результат можно сформулировать следующим обра-

---

[1]) Имеется в виду операция, в которой из двух величин определяется меньшая. Автор в дальнейшем отличает такую операцию сравнения от операции, в которой из двух величин определяется бо́льшая. — *Прим. перев.*

[2]) Для краткости здесь и далее операция сравнения по минимуму называется просто операцией сравнения. Там, где это понадобится, будет использоваться точное название соответствующей операции. — *Прим. перев.*

зом: алгоритм Флойда требует для своего выполнения времени счета порядка $O(2N^3)$.

Далее проанализируем вычислительную сложность алгоритма Дейкстры. На первой итерации этого алгоритма должны быть просмотрены $(N-1)$ неокрашенных вершин. Поскольку просмотры вершин осуществляются с помощью уравнения (1), то на первой итерации выполняется $(N-1)$ операций сложения, $(N-1)$ операций сравнения, а также производится выбор наименьшего из $(N-1)$ чисел (т. е. выполняется еще $(N-1)$ операций сравнения). Итак, первая итерация включает $3(N-1)$ операций. Аналогично можно показать, что вторая итерация включает $3(N-2)$ операций, третья $3(N-3)$ операций и т. д. Общее число операций в алгоритме Дейкстры определяется соотношением

$$\sum_{i=1}^{i=N} 3\,(N-i) = 3N\,(N-1)/2.$$

Помимо указанных операций на каждой итерации алгоритма необходимо также выполнять операции по выявлению окрашенных и неокрашенных вершин. Это, вообще говоря, приводит к дополнительным вычислительным затратам, которые, однако, могут быть сведены к минимуму при использовании соответствующих приемов реализации алгоритма при составлении программы. Рассмотрение этих приемов выходит за рамки настоящей книги. Соответствующий материал можно найти в работах [13, 15]. Итак, алгоритм Дейкстры имеет время счета порядка $O(1{,}5\,N^2)$. Отсюда можно заключить, что алгоритм Форда в наихудшем случае имеет время счета порядка $O(1{,}5N^3)$. Действительно, в алгоритме Дейкстры каждая вершина окрашивается ровно один раз, а в алгоритме Форда она может окрашиваться до $(N-1)$ раза.

Таким образом, нами получены следующие результаты: в алгоритме Дейкстры выполняется $1{,}5N^2$ операций[1]); в алгоритме Форда в наихудшем случае выполняется $1{,}5N^3$ операций; в алгоритме Флойда выполняется $2N^2$ операций; в алгоритме Данцига выполняется $2N^3$ операций.

Как уже отмечалось, алгоритм Флойда мог бы быть заменен алгоритмом Дейкстры, многократно повторяемым при выборе в качестве начальной каждой вершины исходного графа. Соответствующая процедура связана с затратами времени порядка $O(1{,}5N^3)$, что меньше времени счета для алгоритма Флойда, имеющего порядок $O(2N^3)$. Надо, однако, иметь в виду, что при наличии в исходном графе дуг отрицательной длины (при этом, конечно, исключается наличие контуров отрицательной длины), алгоритм Дейкстры должен быть заменен алгоритмом Форда, что приводит к времени счета порядка $O(1{,}5N^4)$. Данная оценка при достаточно

[1]) Имеются в виду операции сложения и сравнения. — *Прим. перев.*

большом N превышает величину $O(2N^3)$ затрат времени для алгоритма Флойда. Однако не следует забывать, что действительное количество операций в алгоритме Форда, по всей вероятности, будет значительно меньше используемой для него оценки.

В заключение отметим, что анализ вычислительной сложности становится все более притягательной областью для многих исследователей. Читателя, который интересуется данной проблематикой, мы отсылаем к обзорным статьям Карпа [6] и Лоулера [8]. Тем же читателям, которые интересуются эффективными приемами программирования алгоритмов, рассмотренных выше, мы рекомендуем полезную работу Кнута [7].

3.3. Алгоритм поиска k кратчайших путей

В двух предыдущих разделах данной главы была рассмотрена задача поиска кратчайших путей. Однако во многих случаях полезно знать не только кратчайшие пути, но и вторые, третьи и т. д. по длине пути. В частности, в примере 2 из разд. 3.1 авиакомпания могла бы обратиться ко второму кратчайшему маршруту между Спрингфилдом и Анкарой, если клиенты этой авиакомпании не смогут воспользоваться кратчайшим маршрутом в связи с трудностями получения въездной визы или из-за отмены каких-то полетов, или, наконец, вследствие большой загрузки авиалиний, принадлежащих кратчайшему маршруту.

В данном разделе вначале описывается алгоритм, называемый *алгоритмом двойного поиска* (double — sweep), который находит k первых кратчайших путей из некоторой фиксированной вершины ко всем остальным вершинам исходного графа. Вслед за этим описываются два алгоритма, называемые обобщенным алгоритмом Флойда и обобщенным алгоритмом Данцига, которые находят k первых кратчайших путей между каждой парой вершин исходного графа.

Алгоритмы Дейкстры, Флойда и Данцига, представленные в разд. 3.1 и 3.2, позволяют находить только самые короткие пути между вершинами. Эти алгоритмы включают в основном операции сложения и сравнения, которые выполняются над обычными числами, представляющими длины соответствующих дуг или путей. В частности, использование в алгоритме Дейкстры соотношения (1) было связано с выполнением лишь операций сложения и сравнения. То же самое относится к использованию в алгоритме Флойда соотношения (3) и в алгоритме Данцига — соотношений (4) —(7).

Алгоритмы, которые рассматриваются в данном разделе (алгоритм двойного поиска, обобщенный алгоритм Флойда и обобщенный алгоритм Данцига), также состоят исключительно из операций сложения и сравнения. Однако эти операции выполняются не над числами (как в алгоритмах Дейкстры, Флойда и Данцига), а

над наборами из k чисел, представляющих длины соответствующих дуг или путей. Учитывая это, введем множество R^k векторов $(d_1, d_2, \ldots, d_k)$, удовлетворяющих условию $d_1 < d_2 < \ldots < d_k$[1]. Таким образом, компоненты векторов из R^k различны и упорядочены в порядке возрастания их величин. Например, вектор $(-3, -1, 0, 4, 27)$ принадлежит R^5.

Пусть $A = (a_1, a_2, \ldots, a_k)$ и $B = (b_1, b_2, \ldots, b_k)$ — два вектора из множества R^k. *Обобщенная операция сравнения* для A и B, обозначаемая знаком $+$, определяется следующим образом:

$$A + B = \min_k \{a_i, b_i : i = 1, 2, \ldots, k\}, \tag{8}$$

где через $\min_k \{X\}$ обозначен вектор, составленный из k различных наименьших по величине элементов множества X.

Обобщенная операция сложения для векторов A и B, обозначаемая знаком $\times$, определяется следующим образом:

$$A \times B = \min_k \{a_i + b_j : i, j = 1, 2, \ldots, k\}. \tag{9}$$

Например, если $A = (1, 3, 4, 8)$ и $B = (3, 5, 7, 16)$, то $A + B = \min_4 (1, 3, 4, 8, 3, 5, 7, 17) = (1, 3, 4, 5)$, а $A \times B = \min_4(1 + 3, 1 + 5, 1 + 7, 1 + 16, 3 + 3, 3 + 5, 3 + 7, 3 + 16, 4 + 3, \ldots) = (4, 6, 7, 8)$. Заметим, что компоненты векторов $A + B$ и $A \times B$ можно упорядочить таким образом, чтобы выполнялись соотношения $A + B \in R^k$ и $A \times B \in R^k$. Кроме того, поскольку компоненты векторов из R^k упорядочены в порядке возрастания их величин, обобщенная операция сравнения требует выполнения не более чем k обычных операций сравнения, а обобщенная операция сложения требует выполнения не более чем $k(k-1)$ обычных операций сложения и не более чем $k(k-1)$ обычных операций сравнения.

Обобщая обозначения, которые использовались в предыдущих разделах, определим на множестве R^k вектор $d^0{}_{ij} = (d^0{}_{ij1}, d^0{}_{ij2}, \ldots, d^0{}_{ijk})$. Компоненты этого вектора задают длины k кратчайших дуг, ведущих из вершины i в вершину j. Если какие-либо две дуги, ведущие из i в j, имеют одинаковые длины, то соответствующее число определяет лишь один из компонентов вектора $d^0{}_{ij}$. Если среди рассматриваемых дуг нельзя найти k дуг, имеющих различные длины, то неопределенные компоненты вектора $d^0{}_{ij}$ условно полагаются равными ∞. Например, если вершина i соединена с вершиной j тремя дугами, имеющими длины соответственно 9, 13 и 9, то для $k = 4$ $d^0{}_{53} = (9, 13, \infty, \infty)$. При $i = j$ будем считать, что имеется некоторая дуга нулевой длины, являющаяся петлей в вершине i.

---

[1]) В дальнейшем будет ясно, что в R^k включаются и такие векторы, в которых некоторые из компонентов условно приравнены к ∞. — *Прим. перев.*

Обозначим через D^0 матрицу, элемент (i, j) которой совпадает с вектором $d^0{}_{ij}$[1]).

Далее введем вектор $d^m_{ij} = (d^m_{ij1}, d^m_{ij2}, \ldots, d^m_{ijk}) \in R^k$, компоненты которого задают длины k кратчайших путей, ведущих из вершины i в вершину j и использующих в качестве промежуточных вершины 1, 2, .., m. (Здесь так же, как и в разд. 3.1, вершины исходного графа перенумерованы целыми числами от 1 до N.) Обозначим через D^m матрицу, каждый элемент (i,j) которой совпадает с d^m_{ij}.

Обозначим, наконец, через $d^*{}_{ij} = (d^*{}_{ij1}, d^*{}_{ij2}, \ldots, d^*{}_{ijk}) \in R^k$ вектор, компоненты которого совпадают с длинами k кратчайших путей, ведущих из вершины i в вершину j. Эти длины будем называть *оптимальными длинами соответствующих путей*. Обозначим через D^* матрицу, каждый элемент (i, j) которой совпадает с $d^*{}_{ij}$.

Пусть матрица L образована из матрицы D^0 путем замены всех компонентов каждого вектора $d^0{}_{ij}$ при $i \leqslant j$ «величиной» ∞. Пусть матрица U образована из матрицы D^0 путем замены всех компонентов каждого вектора $d^0{}_{ij}$ при $i \leqslant j$ «величиной» ∞. Матрицы L и U называются соответственно верхней треугольной и нижней треугольной подматрицами матрицы D^0. Например, рассмотрим случай, в котором $k = 2$ и

$$D^0 = \begin{bmatrix} (0,1) & (6,10) & (2,9) \\ (1,8) & (0,\infty) & (3,\infty) \\ (\infty,\infty) & (2,4) & (0,\infty) \end{bmatrix}$$

Тогда матрицы L и U имеют следующий вид:

$$L = \begin{bmatrix} (\infty,\infty) & (\infty,\infty) & (\infty,\infty) \\ (1,8) & (\infty,\infty) & (\infty,\infty) \\ (\infty,\infty) & (2,4) & (\infty,\infty) \end{bmatrix}$$

$$U = \begin{bmatrix} (\infty,\infty) & (6,10) & (2,9) \\ (\infty,\infty) & (\infty,\infty) & (3,\infty) \\ (\infty,\infty) & (\infty,\infty) & (\infty,\infty) \end{bmatrix}$$

Если мы хотим определить длины k кратчайших путей, ведущих из некоторой фиксированной вершины (скажем, вершины 1) в каждую вершину исходного графа, то нам необходимо определить только первую строку $(d^*{}_{11}, d^*{}_{12}, \ldots, d^*{}_{1N})$ матрицы D^*. Заметим, что эта строка состоит из N векторов, принадлежащих R^k. Следовательно, необходимо определить Nk величин и Nk путей, имеющих соответствующие длины.

[1]) То есть в данном случае D^0 является не матрицей, а тензором третьего ранга. — *Прим. перев.*

Алгоритм двойного поиска [11, 12] позволяет эффективно находить указанные Nk путей. При этом, конечно, требуется, чтобы в исходном графе отсутствовали контуры отрицательной длины. Работа алгоритма двойного поиска начинается с формирования некоторой строки $(d_{11}^{(0)}, d_{12}^{(0)}, \ldots, d_{1N}^{(0)})$, покомпонентно оценивающей сверху оптимальную строку $(d^*_{11}, d^*_{12}, \ldots, d^*_{1N})$. Это означает, что каждый компонент любого вектора $d^{(0)}$ не меньше соответствующего компонента вектора d^*_{1j}. При этом в начальной точке должно быть $d_{111}^{(0)} = 0$. В частности, все компоненты оценочной строки (за исключением компонента $d_{111}^{(0)}$, равного нулю) могли бы быть приняты равными ∞. Затем в алгоритме вычисляются новые (меньшие) оценки для каждого вектора d^*_{1i}. Это осуществляется путем многократной корректировки исходного оценочного вектора $d_{1i}^{(0)}$. Корректировка состоит в переходе к новому вектору $d_{1i}^{(0)}$, получаемому в результате выполнения обобщенной операции сравнения над «старым» вектором $d_{1i}^{(0)}$ и вектором $d_{1j}^{(0)} \times d_{ji}^{(0)}$. Корректировка проводится для всех целых j от 1 до N. По окончании процесса корректировки получается новая оценка вектора d^*_{1i} — вектор d^1_{1i}. Так может быть получена вся новая оценочная строка $(d_{11}^{(1)}, d_{11}^{(1)}, \ldots, d_{1N}^{(1)})$. Далее процесс продолжается аналогичным образом. Работа алгоритма заканчивается при совпадении двух последовательно получаемых оценочных строк — $(d_{11}^{(i)}, d_{12}^{(i)}, \ldots, d_{1N}^{(i)})$ и $(d_{11}^{(i+1)}, d_{12}^{(i+1)}, \ldots, d_{1N}^{(i+1)})$ для $i \geqslant 1$. При этом можно показать, что последняя из полученных оценочных строк совпадает с оптимальной строкой $(d_{11}^*, d_{12}^*, \ldots, d_{1N}^*)$ (см. обоснование алгоритма двойного поиска).

Для алгоритма двойного поиска существует дополнительная возможность повышения эффективности, связанная с использованием в процессе пересчета оценочной строки $d_1^{(i)} = (d_{11}^{(i)}, d_{12}^{(i)}, \ldots, d_{1N}^{(i)})$ в новую оценочную строку $d_1^{(i+1)} = (d_{11}^{(i+1)}, d_{12}^{(i+1)}, \ldots, d_{1N}^{(i+1)})$, некоторых уже вычисленных к текущему моменту векторов d_{1i}. Такое немедленное использование пересчитанных векторов d_{1i} ускоряет сходимость алгоритма.

С учетом всех высказанных соображений мы можем теперь формально описать алгоритм двойного поиска.

Алгоритм двойного поиска

Шаг 1. (*Начальная установка*). Сформировать начальную оценочную строку $d_1^{(0)} = (d_{11}^{(0)}, d_{12}^{(0)}, \ldots, d_{1N}^{(0)})$ для оптимальной строки d_1^* из компонент, не меньших по величине соответствующих ком-

[1]) Выбор начальных значений компонент может быть осуществлен, например, так, как это предлагалось делать выше при описании общей схемы алгоритма. — *Прим. перев.*

понент строки d_1^* [1]. При этом положить $d_{111}^0 = 0$ (по предположению в вершине 1 имеется петля нулевой длины).

Шаг 2. При условии что для строки d_1^* определена оценочная строка d_1^{2r}, определить оценочные строки $d_1^{(2r+1)}$ и $d_1^{(2r+2)}$ следующим образом:

Обратный поиск:

$$d_1^{(2r+1)} = d_1^{(2r+1)} L + d_1^{(2r)}. \tag{10}$$

Прямой поиск:

$$d_1^{(2r+2)} = d_1^{(2r+2)} U + d_1^{(2r+1)}. \tag{11}$$

Шаг 2, являющийся основным шагом алгоритма, проводится последовательно для $r = 0, 1, 2, \ldots$ и т.д.

Отметим, что умножение и сложение в соотношениях (10) и (11) следует понимать в обобщенном смысле согласно определениям, данным соответственно в (8) и (9).

Выполнение алгоритма двойного поиска заканчивается при совпадении двух последовательных оценок $d_1^{(t-1)}$ и $d_1^{(t)}$ для некоторого $t > 1$.

Вполне возможно, что на данном этапе изложения читатель захотел бы познакомиться с деталями проведения расчетов по соотношениям (10) и (11), в которых определяемая оценочная строка фигурирует как в правой, так и в левой части. Что касается соотношения (10), то здесь сначала в строке $d_1^{(2r+1)}$ определяется вектор $d_{1N}^{(2r+1)}$. (Это возможно в силу того, что N-й столбец матрицы L состоит из векторов, все компоненты которых совпадают с ∞, и потому правая часть (10) фактически от $d_1^{(2r+1)}$ не зависит.) Затем по соотношению (10) определяется «второй от конца» в строке $d_1^{(2r+1)}$ вектор $d_{1,N-1}^{(2r+1)}$. (Это возможно в силу того, что $(N-1)$-й столбец матрицы L состоит из векторов, среди которых только у вектора последней строки матрицы L некоторые из компонентов могут быть не равны ∞. Поэтому правая часть соотношения (10) фактически зависит лишь от вектора $d_{1N}^{(2r+1)}$, который уже был вычислен на предыдущем шаге.) Далее по соотношению (10) определяются векторы $d_{1,N-2}^{(2r+1)}$, $d_{1,N-3}^{(2r+1)}$ и т. д. до определения всех векторов строки $d_1^{(2r+1)}$. Таким образом, с помощью соотношения (10) можно определить все векторы строки $d_1^{(2r+1)}$, если начинать с последнего и кончать первым. Это позволяет назвать вычисления с помощью соотношения (10) процедурой *обратного поиска*. Аналогично с помощью соотношения (11) можно определить все векторы строки $d_1^{(2r+2)}$, если начинать с первого и кончать последним. Соответственно проведение вычислений по соотношению (11) можно назвать процедурой *прямого поиска*.

С помощью алгоритма двойного поиска в исходном графе определяются длины некоторых путей. Но как определить сами эти пути? Поскольку данный алгоритм начинает работу с произвольного набора величин, лишь оценивающих сверху соответствующие компоненты оптимальной строки d_1^*, то отдельные компоненты оценочных строк, генерируемых алгоритмом, не обязательно определяют длины каких-то путей исходного графа. Поэтому в рамках алгоритма нет возможности строить в каждой итерации пути, соответствующие компонентам новой оценочной строки, как это делалось в алгоритмах Флойда и Данцига[1]).

После применения алгоритма двойного поиска путь, соответствующий каждому компоненту оптимальной строки d_1^*, может быть восстановлен следующим образом. Предположим, что мы хотим найти m-й кратчайший путь из вершины 1 в вершину j. Этот путь должен включать l-й кратчайший путь ($l \leqslant m$), ведущий из вершины 1 в некоторую вершину i, и дугу, ведущую из вершины i в вершину j. Суммарная длина указанных пути и дуги должна быть равна величине d_{1jm}^*. Вершина i, для которой это имеет место, называется предпоследней вершиной m-го кратчайшего пути из вершины 1 в вершину j. Итак, предпоследняя вершина определяется в результате просмотра[2]) компонентов, составляющих матрицу D^0 и строку d_1^*. Как только вершина i определена, описанная процедура может быть повторена для поиска предпоследней вершины в l-м кратчайшем пути, ведущем из вершины 1 в вершину i. Повторяя процедуру и далее, можно в обратном направлении пройти весь m-й кратчайший путь из вершины 1 в вершину j.

Вполне возможно, что на каких-то этапах описанной процедуры предпоследняя вершина, скажем m-го кратчайшего пути из вершины 1 в вершину i, окажется не единственной. В этом случае можно зафиксировать все вершины, претендующие на роль предпоследней в данном пути, и при необходимости пройти в обратном направлении все пути, имеющие одну и ту же длину и являющиеся m-ми кратчайшими путями из вершины 1 в вершину i.

Если необходимо найти все m-е кратчайшие пути, то наиболее разумной организацией вычислительной процедуры является такая, при которой вначале определяются все наикратчайшие пути, затем все вторые кратчайшие пути и т. д. Это связано с тем, что при определении m-х крачайших путей необходимо располагать всеми l-ми кратчайшими путями для $l \leqslant m$.

---

[1]) По-видимому, здесь подразумевается не непосредственное построение путей, а лишь определение с помощью встроенного способа (см. предыдущий раздел) элементов (или множеств) p_{ij}. — *Прим. перев.*

[2]) При указанном просмотре определяется не только соответствующая вершина i, но и параметр l для соответствующего пути из вершины 1 в вершину i. — *Прим. перев.*

Обоснование алгоритма двойного поиска. В соотношении (10) осуществляется обобщенная операция ставнения соответствующих векторов строки $d_1^{(2r+1)}L$ и строки $d_1^{(2r)}$. Аналогично в соотношении (11) осуществляется обобщенная операция сравнения соответствующих векторов строки $d_1^{(2r+2)}U$ и строки $d_1^{(2r+1)}$. Поэтому в последовательно генерируемых алгоритмом строках $d_1^{(t)}$ соответствующие компоненты не могут увеличиваться. Другими словами, последовательность строк $d_1^{(0)}, d_1^{(1)}, \ldots, d_1^{(t)}$ состоит из Nk невозрастающих числовых последовательностей. При этом ни одно из входящих в них чисел не может стать меньше величины соответствующего компонента оптимальной строки d_1^* (по предположению указанные величины могут быть как конечными, так и бесконечными). Это связано со следующим обстоятельством. Предположим, что *первая* оценка длины какого-то кратчайшего пути, оказавшаяся меньше ее действительного значения, относится к m-му кратчайшему пути из вершины 1 в вершину j. С учетом вида соотношений (10) и (11) истинное значение указанного пути могло бы быть получено как сумма оценки длины какого-либо кратчайшего пути из вершины 1 в некоторую вершину i и длины дуги, ведущей из вершины i в вершину j. Отсюда следует, что оценка длины указанного пути из вершины 1 в вершину i должна быть меньше истинного значения длины данного пути. Но это противоречит предположению о том, что единственная оценка длины, меньшая действительного ее значения, была получена лишь для m-го кратчайшего пути из вершины 1 в вершину j.

Итак, мы доказали, что последовательность оценочных строк $d_1^{(0)}, d_1^{(1)}, \ldots, d_1^{(t)}$ состоит из Nk невозрастающих последовательностей оценок длин соответствующих кратчайших путей, причем эти оценки либо больше, либо равны действительным значениям указанных длин. Теперь нам остается показать, что каждая из этих Nk последовательностей за конечное число шагов сходится к истинному значению длины соответствующего пути. Мы сделаем это, показав, что в генерируемых алгоритмом оценочных строках число компонентов, точно определяющих длины каких-то кратчайших путей, увеличивается после каждой итерации двойного поиска по крайней мере на единицу.

Нам понадобятся два вспомогательных результата.

ЛЕММА 3.1. Если вершина i является предпоследней вершиной m-го кратчайшего пути P, ведущего из вершины 1 в вершину j, то часть этого пути P', заключенная между вершинами 1 и i, является некоторым l-м кратчайшим путем из вершины 1 в вершину i, где $l \leqslant m$.

Доказательство. Если бы путь P' не являлся одним из l-х кратчайших путей, где $l \leqslant m$, то нашлось бы m кратчайших путей из вершины 1 в вершину i, длины которых были бы меньше длины

пути P'. Каждый из этих путей мог бы быть «продлен» до вершины j путем добавления к нему соответствующей дуги. Это привело бы к построению m кратчайших путей, длины которых меньше длины пути P. Полученное противоречие завершает доказательство.

ЛЕММА 3.2. Если в процессе выполнения рассматриваемого алгоритма определены длины m кратчайших путей из вершины 1 в каждую другую вершину, то длина $(m+1)$-го кратчайшего пути, начинающегося и заканчивающегося в вершине 1, будет определена в ходе следующей итерации двойного поиска.

Доказательство. Рассмотрим любой $(m+1)$-й кратчайший путь P, начинающийся и заканчивающийся в вершине 1. Пусть i — предпоследняя вершина пути P. В силу леммы 3.1 часть этого пути P', заключенная между вершинами 1 и i, представляет собой какой-либо l-й кратчайший путь из вершины 1 в вершину i, где $l \leqslant m+1$. При этом путь P' не может быть $(m+1)$-м кратчайшим путем, иначе наряду с путем P нашлось бы еще m путей, начинающихся и заканчивающихся в вершине 1, которые так же, как и путь P, проходят через вершину i и имеют меньшую, чем у P, длину. Но последнее противоречит тому, что среди $(m+1)$-х кратчайших путей, начинающихся и заканчивающихся в вершине 1, есть «путь», не содержащий дуг, для которого $d^*_{11}=0$.

Итак, путь P' — один из первых m кратчайших путей, ведущих из вершины 1 в вершину i. С учетом этого, а также с учетом того, что известны длины k кратчайших дуг, ведущих из вершины i в вершину j, можно заключить, что длина пути P будет определена в алгоритме на следующей итерации двойного поиска.

Теперь, используя две приведенные выше леммы, покажем, что на каждой новой итерации рассматриваемого алгоритма получается хотя бы одно новое значение длины какого-то из кратчайший путей. Предположим, что перед началом r-й итерации двойного поиска определены длины m кратчайших путей из вершины 1 в каждую из остальных вершин исходного графа при некотором $m \geqslant 0$. Пусть j — любая вершина, для которой величина $d^*_{1j,m+1}$ (т. е. длина $(m+1)$-го кратчайшего пути из вершины 1 в вершину j) еще не была определена. Рассмотрим соответствующий путь.

Пусть i —предпоследняя вершина этого пути. Если длина $(m+1)$-го кратчайшего пути из вершины 1 в вершину i уже была определена на предыдущих итерациях алгоритма, то на новой итерации будет определена и длина рассматриваемого пути, поскольку этот путь состоит из какого-то l-го кратчайшего пути из вершины 1 в вершину i, где $l \leqslant m+1$, и из дуги, ведущей из вершины i в вершину j.

Если же длина $(m+1)$-го кратчайшего пути из вершины 1 в вершину i не была определена на предыдущих итерациях алгоритма, то нельзя утверждать, что на новой итерации будет получена длина $(m+1)$-го кратчайшего пути из вершины 1 в вершину j.

Предположим, что действительно в новой итерации не удается получить длину рассматриваемого пути. Тогда в силу леммы 3.1 $(m+1)$-й кратчайший путь из вершины 1 в вершину j должен включать $(m+1)$-й кратчайший путь из вершины 1 в вершину i и дугу, ведущую из вершины i в вершину j.

Повторим проведенное рассуждение, заменив вершину j вершиной i. Далее при необходимости будем последовательно повторять это рассуждение для новых вершин типа вершины i. В результате придем к ситуации, в которой либо новая итерация алгоритма дает не определявшуюся ранее длину соответствующего кратчайшего пути, либо весь $(m+1)$-й кратчайший путь из вершины 1 в вершину j оказывается пройденным в обратном направлении до вершины 1. В последнем случае в силу леммы 3.2 на новой итерации алгоритма будет получена длина $(m+1)$-го кратчайшего пути, начинающегося и заканчивающегося в вершине 1. Это завершает доказательство сходимости алгоритма двойного поиска за конечное число шагов.

Из приведенного доказательства вытекает важное следствие. Пусть исходный граф не содержит контуров отрицательной длины, достижимых из вершины 1. Тогда после $r \geqslant N$ итераций рассматриваемого алгоритма первые компоненты всех векторов оценочной строки $(d_{11}^{(2r)}, d_{12}^{(2r)}, \ldots, d_{1N}^{(2r)})$ в дальнейшем будут неизменными. Аналогично после $r \geqslant 2N$ итераций первые два компонента всех векторов оценочной строки $(d_{11}^{(2r)}, d_{12}^{(2r)}, \ldots, d_{1N}^{2r})$ также в дальнейшем будут неизменными. И вообще после $r \geqslant cN$ итераций алгоритма двойного поиска первые c компонентов всех векторов оценочной строки $(d_{11}^{(2r)}, d_{12}^{(2r)}, \ldots, d_{1N}^{(2r)})$ далее не изменяются. (Заметим, что здесь используется индекс $2r$, так как на каждой итерации определяются две оценки каждого компонента оптимальной строки d_1^*.)

Таким образом, мы можем выявить наличие контура отрицательной длины, достижимого из вершины 1, по уменьшению (или просто по изменению) любой из первых c компонентов в векторах оценочной строки $d_{11}^{(2r)}, d_{12}^{(2r)}, \ldots, d_{1N}^{(2r)}$ после r итераций алгоритма, где $r \geqslant c$ при любом $c = 1, 2, \ldots, N$.

Поскольку на каждой итерации алгоритма двойного поиска вычисляется длина по крайней мере одного нового кратчайшего пути, то до завершения его работы потребуется выполнить самое большее Nk итераций. Однако практически требуется значительно меньшее число итераций. Это связано с тем, что в ходе выполнения одной итерации, как правило, вычисляются длины не одного, а большего числа кратчайших путей.

Какое же количество обобщенных операций сложения и сравнения выполняется в рамках одной итерации алгоритма двойного поиска? Матрица D^0 содержит $(N^2 - N)$ элементов, не считая диагональных. Следовательно, каждая из матриц L и U включает

только $^1/_2(N^2 - N)$ элементов (векторов), компоненты которых могут принимать конечные значения. На итерациях алгоритма с каждым элементом матриц L и U связано выполнение одной обобщенной операции сложения и одной обобщенной операции сравнения. Таким образом, каждая итерация требует, грубо говоря, выполнения N^2 обобщенных операций сложения и N^2 обобщенных операций сравнения, т. е. в алгоритме двойного поиска может быть выполнено самое большее $^1/_2kN^3$ обобщенных операций сложения и такое же количество обобщенных операций сравнения.

При $k = 1$ в рассматриваемом алгоритме выполняется не более чем $^1/_2N^3$ операций сложения и $^1/_2N^3$ операций сравнения. То есть при $k = 1$ время счета по алгоритму двойного поиска имеет порядок $O(N^3)$, что меньше времени счета по алгоритму Форда (в наихудшем случае), имеющего порядок $O(1{,}5\ N^3)$, но больше времени счета по алгоритму Дейкстры, имеющему порядок $O(1{,}5\ N^2)$. При рассмотрении результатов проведенного сравнения, конечно, следует учитывать, что для алгоритма двойного поиска и алгоритма Форда были использованы наихудшие оценки времени счета.

ПРИМЕР 1. (*Применение алгоритма двойного поиска.*) Используем алгоритм двойного поиска для нахождения первых трех кратчайших путей из

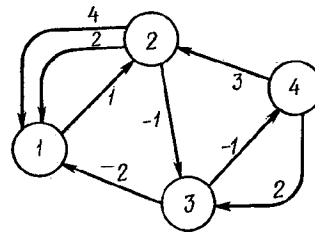


Рис. 3.5. Алгоритм двойного поиска

вершины 1 во все вершины графа, изображенного на рис. 3.5. Матрица D^0, составленная из длин дуг этого графа, имеет вид

$$\begin{bmatrix} (0,\infty,\infty) & (1,\infty,\infty) & (\infty,\infty,\infty) & (\infty,\infty,\infty) \\ (2,4,\infty) & (0,\infty,\infty) & (-1,\infty,\infty) & (\infty,\infty,\infty) \\ (2,\infty,\infty) & (\infty,\infty,\infty) & (0,\infty,\infty) & (-1,\infty,\infty) \\ (\infty,\infty,\infty) & (3,\infty,\infty) & (2,\infty,\infty) & (0,\infty,\infty) \end{bmatrix}$$

Матрица L сводится к следующей:

$$\begin{bmatrix} (\infty,\infty,\infty) & (\infty,\infty,\infty) & (\infty,\infty,\infty) & (\infty,\infty,\infty) \\ (2,4,\infty) & (\infty,\infty,\infty) & (\infty,\infty,\infty) & (\infty,\infty,\infty) \\ (2,\infty,\infty) & (\infty,\infty,\infty) & (\infty,\infty,\infty) & (\infty,\infty,\infty) \\ (\infty,\infty,\infty) & (3,\infty,\infty) & (2,\infty,\infty) & (\infty,\infty,\infty) \end{bmatrix}$$

Матрица U сводится к следующей:

$$\begin{bmatrix} (\infty, \infty, \infty) & (1, \infty, \infty) & (\infty, \infty, \infty) & (\infty, \infty, \infty) \\ (\infty, \infty, \infty) & (\infty, \infty, \infty) & (-1, \infty, \infty) & (\infty, \infty, \infty) \\ (\infty, \infty, \infty) & (\infty, \infty, \infty) & (\infty, \infty, \infty) & (-1, \infty, \infty) \\ (\infty, \infty, \infty) & (\infty, \infty, \infty) & (\infty, \infty, \infty) & (\infty, \infty, \infty) \end{bmatrix}$$

Обратный поиск с использованием соотношения (10) связан в данном случае с последовательными вычислениями по формулам:

$$d_{14}^{(2r+1)} = d_{14}^{(2r)}$$

$$d_{13}^{(2r+1)} = (2, \infty, \infty) \times d_{14}^{(2r+1)} + d_{13}^{(2r)}$$

$$d_{12}^{(2r+1)} = (3, \infty, \infty) \times d_{14}^{(2r+1)} + d_{12}^{(2r)}$$

$$d_{11}^{(2r+1)} = (2, 4, \infty) \times d_{12}^{(2r+1)} + (2, \infty, \infty) \times d_{13}^{(2r+1)} + d_{11}^{(2r)}$$

Прямой поиск с использованием соотношения (11) в данном случае связан с последовательными вычислениями по формулам:

$$d_{11}^{(2r+2)} = d_{11}^{(2r+1)}$$

$$d_{12}^{(2r+2)} = (1, \infty, \infty) \times d_{11}^{(2r+2)} + d_{12}^{(2r+1)}$$

$$d_{13}^{(2r+2)} = (-1, \infty, \infty) \times d_{12}^{(2r+2)} + d_{13}^{(2r+1)}$$

$$d_{14}^{(2r+2)} = (-1, \infty, \infty) \times d_{13}^{(2r+2)} + d_{14}^{(2r+1)}$$

В приведенной ниже таблице даны результаты проведения первых пяти простых итераций[1]). Результаты, полученные в четвертой и пятой простых итерациях, совпадают и, следовательно, представляют собой длины соответствующих кратчайших путей.

	$d_{11}^{(r)}$	$d_{12}^{(r)}$	$d_{13}^{(r)}$	$d_{14}^{(r)}$	Поиск
$r = 0$	(0, ∞, ∞)	(∞, ∞, ∞)	(∞, ∞, ∞)	(∞, ∞, ∞)	
$r = 1$	(0, ∞, ∞)	(∞, ∞, ∞)	(∞, ∞, ∞)	(∞, ∞, ∞)	Обратный
$r = 2$	(0, ∞, ∞)	(1, ∞, ∞)	(0, ∞, ∞)	(−1, ∞, ∞)	Прямой
$r = 3$	(0, 2, 3)	(1, 2, ∞)	(0, 1, ∞)	(−1, ∞, ∞)	Обратный
$r = 4$	(0, 2, 3)	(1, 2, 3)	(0, 1, 2,)	(−1, 0, 1)	Прямой
$r = 5$	(0, 2, 3)	(1, 2, 3)	(0, 1, 2)	(−1, 0, 1)	Обратный

Отметим, что в данном случае алгоритм двойного поиска приводит к конечному результату после пяти простых итераций (или после двух с половиной полных итераций), что близко к теоретической оценке числа итераций, составляющей $^1/_2Nk = 6$.

[1]) Под простой итерацией понимается только прямой или только обратный поиск. — *Прим. перев.*

Перейдем теперь к рассмотрению более общей задачи, в кото рой на графе ищутся k первых кратчайших путей, ведущих из лю бой вершины в любую другую вершину. Алгоритмы Флойда и Данцига, рассмотренные в разд. 3.2, решают эту задачу для слу чая $k = 1$. Теперь мы обобщим эти алгоритмы на случае $k > 1$.

Конечно, рассматриваемая задача могла бы быть решена путем применения к исходному графу алгоритма двойного поиска с вы бором в качестве начальной любой другой вершины графа. Однако в вычислительном отношении такой подход к решению задачи не слишком эффективен, поскольку часть имеющейся в соответствую щей процедуре информации никак не используется.

Прежде чем переходить к описанию обобщенного алгоритма Флойда и обобщенного алгоритма Данцига, предназначенных для нахождения первых k кратчайших путей между каждой парой вер шин, необходимо ввести ряд новых обозначений.

Пусть компоненты вектора d_{ii} из R^k определяют длины k пу тей, начинающихся и заканчивающихся в вершине i. Первый ком понент вектора d_{ii} совпадает с нулем и соответствует длине пути не содержащего дуг. Все пути, длины которых определяются ком понентами d_{ii}, являются контурами, содержащими вершину i. Определим для вектора d_{ii} *свертку* d_{ii}^C как вектор из R^k, компонен ты которого определяют длины k кратчайших контуров, содер жащих вершину i; указанные контуры составлены из контуров представленных длинами в d_{ii}. Точнее,

$$d_{ii}^C = d_{ii} \times d_{ii} \times d_{ii} \times \ldots \times d_{ii} \in R^k \qquad (12$$

(Напомним, что знаком $\times$ обозначается обобщенная операция сло жения.)

В основе построения обобщенного алгоритма Флойда и обоб щенного алгоритма Данцига лежат те же самые идеи, которые ис пользовались при построении исходного алгоритма Флойда и исходного алгоритма Данцига. Отличие состоит лишь в том, что в обобщенных алгоритмах обычные операции сложения и сравне ния заменяются обобщенными операциями сложения и сравнения. Кроме того, в обобщенных алгоритмах Флойда и Данцига необ ходимо учитывать возможность возникновения кратчайших пу тей, включающих контуры, привязанные к начальной и конечной вершинам рассматриваемых путей.

Обобщенный алгоритм Флойда фактически совпадает с исход ным алгоритмом Флойда, за исключением того, что в нем каждое d_{ij}^m представляет собой вектор из R^k, составленный из длин k первых кратчайших путей из вершины i в вершину j. При этом, как и в исходном алгоритме, в указанных путях в качестве про межуточных допускается использование лишь первых m вершин рассматриваемого графа. В обобщенном алгоритме Флойда соот-

ношение (3) заменяется следующей системой соотношений:

$$d^m_{mm} = \left(d^{m-1}_{mm}\right)^c \tag{13}$$

$$d^m_{im} = d^{m-1}_{im}\, d^m_{mm} \text{ (для всех } i \neq m) \tag{14}$$

$$d^m_{mi} = d^m_{mm} d^{m-1}_{mi} \text{ (для всех } i \neq m) \tag{15}$$

$$d^m_{ij} = d^m_{im}\, d^m_{mj} + d^{m-1}_{ij} \text{ (для всех } i, j \neq m) \tag{16}$$

Обобщенный алгоритм Данцига фактически совпадает с исходным алгоритмом Данцига, за исключением того, что в нем каждое d^m_{ij} представляет собой вектор из R^k, составленный из длин k первых кратчайших путей из вершины i в вершину j. При этом, как и в исходном алгоритме, в указанных путях допускается использование в качестве промежуточных лишь m первых вершин исходного графа. В обобщенном алгоритме Данцига соотношения (7), (4), (5) и (6) переходят соответственно в следующие:

$$d^m_{mm} = \left(d^0_{mm} + \sum_{i=1}^{m-1}\sum_{j=1}^{m-1} d^0_{mi} d^{m-1}_{ij} d^0_{jm}\right)^c \tag{17}$$

$$d^m_{mi} = \sum_{j=1}^{j=m-1} d^m_{mm} d^0_{mj} d^{m-1}_{ji} \quad (i = 1, 2, \ldots, m-1) \tag{18}$$

$$d^m_{im} = \sum_{j=1}^{j=m-1} d^{m-1}_{ij} d^0_{jm} d^m_{mm} \quad (i = 1, 2, \ldots, m-1) \tag{19}$$

$$d^m_{ij} = d^m_{im} d^m_{mj} + d^{m-1}_{ij} \quad (i, j = 1, 2, \ldots, m-1) \tag{20}$$

Отметим, что как в обобщенном алгоритме Флойда, так и в обобщенном алгоритме Данцига при вычислении матрицы D^m первым определяется вектор d^m_{mm}. Затем с использованием d^m_{mm} вычисляются векторы d^m_{im} и d^m_{mi} для всех $i \neq m$. Наконец, после этого определяются все векторы d^m_{ij}, где $i \neq m$, $j \neq m$.

В обобщенном алгоритме Флойда векторы d^m_{mm} вычисляются по соотношению (13), которым устанавливаются длины k кратчайших комбинаций из простых контуров в вершине m, использующих в качестве промежуточных лишь первые m вершин исходного графа.

Далее вычисляются векторы d^m_{im} с использованием соотношения (14), которым определяется длина k кратчайших комбинаций из путей, представленных длинами в d^{m-1}_{im}, и из контуров, представленных длинами в d^m_{mm}. Аналогичным образом с использованием соотношения (15) вычисляются векторы d^m_{mi}. После этого с по-

мощью соотношения (13) вычисляются векторы d_{ij}^m ($i \neq m$, $j \neq m$). Соотношение (16) отличается от своего аналога — соотношения (3) — только тем, что обычные операции сложения и сравнения заменяются в соотношении (16) соответственно обобщенными операциями сложения и сравнения.

В обобщенном алгоритме Данцига элементы d_{mm}^m вычисляются с использованием соотношения (17), которым определяются длины k кратчайших комбинаций из петель в вершине m и из контуров, образованных путями, соединяющими некоторую вершину i с некоторой вершиной j ($i < m$, $j < m$ и, возможно, $j = 1$), а также дугами, ведущими из вершины m в вершину i и из вершины j в вершину m (т. е. имеются в виду контуры, длины которых представлены в d_{mi}^0, d_{ij}^{m-1}, d_{jm}^0). Соотношение (18) отличается от своего аналога — соотношения (4) — тем, что в первом обычные операции сложения и сравнения заменены обобщенными операциями сложения и сравнения, и тем, что в длины искомых кратчайших путей допускается вклад от контуров, представленных длинами в d_{mm}^m. Такое же соответствие можно проследить между соотношением (19) и его аналогом — соотношением (5). Наконец, соотношение (20) есть просто соотношение (6), переформулированное в терминах обобщенных операций сложения и сравнения.

Так же как и оптимальность исходных алгоритмов, оптимальность обобщенного алгоритма Флойда и обобщенного алгоритма Данцига может быть доказана методом индукции по N, т. е. по числу вершин в графе.

Определим теперь, какое количество обобщенных операций выполняется в алгоритме Флойда. В алгоритме должны вычисляться N матриц D^1, ..., D^N. При вычислении каждой матрицы необходимо один раз выполнить расчеты по соотношению (13), $(N - 1)$ раз выполнить расчеты по соотношению (14) и столько же раз по соотношению (15). Наконец, необходимо $(N - 1)^2$ раз провести расчеты по соотношению (16). Расчеты по соотношению (13) связаны с выполнением операции свертки, которая в силу соотношения (12) включает выполнение k обобщенных операций сложения.

Расчеты по соотношениям (14) и (15) в каждом случае связаны с выполнением одной обобщенной операцией сложения. Наконец, расчеты по соотношению (16) связаны с выполнением одной обобщенной операции сложения и одной обобщенной операции сравнения.

Таким образом, вычисление каждой из матриц D^1, D^2, ..., D^N требует выполнения $(N^2 + k - 2)$ обобщенных операций сложения и $(N - 1)^2$ обобщенных операций сравнения, или приблизительно N^2 обобщенных операций сложения и N^2 обобщенных опера-

ций сравнения. Итак, обобщенный алгоритм Флойда требует выполнения приблизительно N^3 обобщенных операций сложения и N^3 обобщенных операций сравнения. Поскольку в обобщенном алгоритме Данцига по существу выполняются в несколько ином порядке те же самые операции, что и в обобщенном алгоритме Флойда, то оба алгоритма требуют одного и того же количества операций. Таким образом, время счета для обобщенного алгоритма Флойда и для обобщенного алгоритма Данцига составляет величину порядка $O(2N^3)$.

Как соотносятся оценки объема вычислений для обобщенных алгоритмов Флойда и Данцига с аналогичной оценкой для алгоритма двойного поиска, приспособленного к решению той же самой задачи? Как было показано, время счета для алгоритма двойного поиска составляет величину порядка $O(kN^3)$, которая определяется выполнением $\frac{1}{2}N^3k$ обобщенных операций сложения и $\frac{1}{2}N^3k$ обобщенных операций сравнения. При N-кратном выполнении алгоритма двойного поиска (по разу при выборе каждой вершины графа в качестве начальной) время счета составит величину $O(kN^4)$. Однако практически время счета оказывается значительно меньше указанной оценки, поскольку число итераций в алгоритме двойного поиска, как правило, не достигает максимально возможной величины.

Таким образом, выбор между алгоритмом Флойда (алгоритмом Данцига) и N раз повторяемым алгоритмом двойного поиска определяется (1) соотношением величин kN^4 и $2N^3$, (2) оценкой того, насколько раньше, чем в наихудшем случае, закончит свою работу алгоритм двойного поиска, и (3) учетом того обстоятельства, что вычислительные затраты на выполнение обобщенных операций сложения и сравнения в алгоритме двойного поиска несколько меньше, чем в альтернативных алгоритмах. (Последнее объясняется тем, что обычно векторы, с которыми оперирует алгоритм двойного поиска, имеют большое число компонентов, равных ∞.) Заметим, что результаты вычислительных экспериментов с алгоритмом двойного поиска можно найти в работе [12].

3.4. Поиск других оптимальных путей

До сих пор нами исследовались такие алгоритмы поиска кратчайших путей, которые могли быть представлены как вполне определенные последовательности операций сложения и сравнения на минимум. В данном разделе будут рассмотрены модификации этих алгоритмов, позволяющие решать задачи поиска не кратчайших путей, а путей, оптимальных в некотором другом смысле.

Первой мы рассмотрим задачу об «узких» местах. *Узким местом*

пути называется кратчайшая из входящих в него дуг. *Задача об узких местах* — это задача поиска такого пути между двумя вершинами, в котором длина кратчайшей дуги максимальна.

ПРИМЕР 1. Рассмотрим транспортную сеть, включающую мосты. Будем предполагать, что при превышении грузоподъемности некоторого моста последний разрушается. Допустим, что мы хотим определить максимальный вес груза, который может быть транспортирован в рассматриваемой сети из пункта a в пункт b без превышения грузоподъемности находящихся на маршруте движения транспорта мостов. Если поставить в соответствие мостам дуги определенного графа и задать длины дуг равными соответствующим ограничениям на нагрузку моста, то задача определения максимально допустимого веса груза сводится к задаче поиска такого пути между вершинами a и b, в котором длина кратчайшей дуги максимальна.

В задаче о кратчайших путях мы связывали с каждым из них число, называемое длиной пути. При этом длина пути определяется как сумма длин входящих в него дуг. В задаче об узких местах мы связываем с каждым путем число, соответствующее его узкому месту и совпадающее с длиной кратчайшей дуги. Указанное число определяется выбором минимального из чисел, представляющих длину дуг рассматриваемого пути. В задаче о кратчайших путях из двух путей более предпочтительным является тот, у которого длина меньше. Для осуществления соответствующего выбора требуется выполнить одну операцию сравнения на минимум. В задаче об узких местах из двух путей более предпочтительным считается тот, для которого длина кратчайшей дуги больше. При осуществлении соответствующего выбора также выполняется одна операция сравнения, но не на минимум, а на максимум. Таким образом, задача об узких местах является весьма сходной с задачей о кратчайших путях. Для решения обеих задач можно использовать одни и те же алгоритмы. Различие в них будет состоять лишь в том, что операции сложения и сравнения на минимум, выполняемые при решении второй задачи, при решении первой замещаются соответственно операциями сравнения на минимум и на максимум.

Итак, если алгоритмы Флойда и Данцига, а также их обобщенные варианты и алгоритмы двойного поиска выполнять с заменой операций сложения операциями сравнения на минимум и с заменой операций сравнения на минимум операциями сравнения на максимум, то полученные модификации алгоритмов будут порождать пути с максимальной длиной кратчайшей дуги. Более того, если в исходных алгоритмах провести указанную замену только для операций сложения, не затрагивая операции сравнения на минимум, то соответствующие модификации алгоритмов будут порождать пути с минимальной длиной кратчайшей дуги.

Следует, однако, иметь в виду, что при использовании для решения задачи об узких местах алгоритма Флойда, алгоритма Данцига и алгоритма двойного поиска отсутствие в исходном графе

какой-либо дуги необходимо фиксировать, полагая длину несуществующей дуги равной $-\infty$, а не $+\infty$, как это делалось при решении задачи о кратчайшем пути. Кроме того, при решении задачи об узких местах длина кратчайшей дуги нулевого пути (т. е. пути, не содержащего дуг) полагается равной $-\infty$, в то время как в задаче о кратчайшем пути эта длина принимается равной нулю.

Таким образом, в задаче об узких местах все элементы матрицы D^0, с которой начинается работа алгоритмов Флойда и Данцига, полагаются равными $-\infty$ всякий раз, когда эти элементы отвечают отсутствующим в графе дугам. То же относится и к начальному вектору $d^0{}_1$ алгоритма двойного поиска. В задаче об узких местах компоненты этого вектора должны быть меньше соответствующих оптимальных значений (так как в отличие от задачи о кратчайших путях в данном случае приходится иметь дело с максимизацией, а не минимизацией).

Алгоритмы Флойда, Данцига и двойного поиска могут быть модифицированы также для решения некоторых других задач о путях, в частности решения задачи о путях с «усилениями». Сопоставим каждой дуге некоторое действительное число, называемое *коэффициентом усиления* дуги. Назовем *усилением пути* величину произведения коэффициентов усиления всех дуг, составляющих путь (при этом если некоторая дуга входит в путь многократно, то при определении величины усиления пути коэффициент усиления этой дуги повторяется в качестве сомножителя соответствующее число раз). *Задача о путях с усилениями* формулируется как задача нахождения такого пути между двумя вершинами, который дает наибольшую величину усиления.

ПРИМЕР 2. Финансист крупной компании решил вложить ее свободный капитал в облигации, от которых можно было бы иметь доход в последующем пятилетнем периоде. В распоряжении финансиста имеется несколько видов облигаций. В течение рассматриваемого пятилетнего периода можно реализовать облигации и высвобождающиеся средства вкладывать в облигации других видов. Как в такой ситуации финансист должен распределить во времени вложения имеющегося капитала?

Поставим в соответствие началу каждого года, когда по предположению производится вложение капитала, вершину графа. Каждому виду облигаций поставим в соответствие дугу, ведущую из вершины, представляющей момент приобретения облигация, в вершину, представляющую момент их реализации. Пусть коэффициент усиления каждой дуги совпадает с суммой, отнесенной к одному доллару, которая получается в результате реализации облигаций. Тогда на построенном графе задачу финансиста можно рассматривать как задачу поиска пути из вершины 1 в вершину 5, имеющего наибольшую величину усиления.

ПРИМЕР 3. Предположим, что при транспортировке товара от пункта производства к пункту продажи на каждом возможном участке маршрута теряется определенная доля товара. Каким следует выбрать маршрут перевозки, чтобы минимизировать величину потерь?

Данная задача может быть сформулирована как задача поиска пути с

максимальной величиной усиления, если для каждой дуги рассматриваемой транспортной сети положить коэффициент усиления равным доли товара, доставляемого в полной сохранности.

С помощью алгоритмов Флойда, Данцига и двойного поиска можно решать задачу поиска соответствующих путей с максимальной величиной усиления, если длиной дуги графа считать коэффициент усиления в данной дуге и если в этих алгоритмах операции сложения и сравнения на минимум заменить соответственно операциями умножения и сравнения на максимум. Указанные модификации алгоритмов не всегда позволяют решить задачу о путях с максимальной величиной усиления (так же как и исходные алгоритмы не всегда решают задачу поиска кратчайших путей). Если задача о кратчайших путях не может быть решена при существовании в исходном графе контуров отрицательной длины, то задача о путях с максимальной величиной усиления не может быть решена при существовании в исходном графе контура с величиной усиления, большей единицы. Данное обстоятельство связано с тем, что существование указанного контура позволяет, повторяя его неограниченное число раз, формировать (непростые) пути со сколь угодно большой величиной усиления.

С помощью алгоритмов Флойда, Данцига и двойного поиска можно также находить соответствующие пути с минимальной величиной усиления. При этом модификация данных алгоритмов должна была бы отличаться от их модификации для задачи о путях с максимальной величиной усиления только тем, что операция сравнения на минимум заменяется операцией сравнения на максимум. В задаче о путях с минимальной величиной усиления также возможен случай, когда решение найти нельзя. Это имеет место, если в исходном графе существует контур с величиной усиления, меньшей —1. Данное обстоятельство обусловливается тем, что наличие указанного контура позволяет, неограниченное число раз повторяя этот контур, формировать (непростые) пути со сколь угодно малой величиной усиления.

Подытоживая сказанное, можно сказать, что алгоритмы Флойда, Данцига и двойного поиска могут быть модифицированы для поиска не только кратчайших, но и путей, оптимальных по другим критериям. В частности, могут быть получены модификации указанных алгоритмов, решающие задачу об узких местах и задачу о путях с усилениями. Эти модификации получаются из исходных алгоритмов путем замены входящих в них операций сложения и сравнения на минимум некоторыми другими операциями при соответствующей интерпретации длин дуг исходного графа.

УПРАЖНЕНИЯ

1. Используйте алгоритм Дейкстры для поиска кратчайших путей, ведущих из вершины 1 в каждую другую вершину, на графе, изображенном на рис. 3.6.

2. Предположим, что после выполнения упражнения 1 вы обнаруживаете, что при расчетах считали отсутствующей дугу (4,2) длины 1. Можно ли при этом воспользоваться уже полученными результатами или необходимо вновь в полном объеме провести вычисления по алгоритму Дейкстры?

3. Используя алгоритм Флойда, найдите кратчайшие пути между всеми вершинами графа, изображенного на рис. 3.6.

4. Используя алгоритм Данцига, найдите кратчайшие пути между всеми парами вершин графа, изображенного на рис. 3.6. Сравните полученные результаты с результатами выполнения предыдущего упражнения.

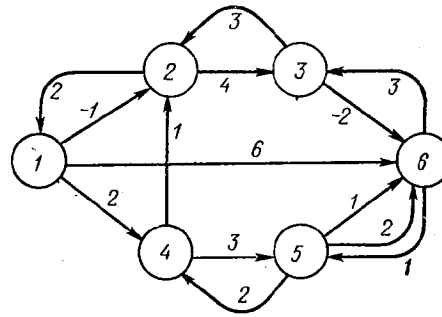


Рис. 3.6.

5. Используя обобщенный алгоритм Флойда, найдите первые три кратчайшие пути между всеми парами вершин на графе, изображенном на рис. 3.6. Выполните то же самое, используя обобщенный алгоритм Данцига.

6. Предположим, что перед вами стоит задача найти кратчайшие маршруты полетов между всевозможными парами аэропортов Европы. При этом необходимо учесть, что из-за отсутствия возможности получения въездной визы аэропорты некоторых стран могут быть закрыты для тех или иных пассажиров. Выберите наиболее эффективный способ решения данной задачи.

7. Влияет ли выбор нумерации вершин графа на эффективность алгоритма Флойда, алгоритма Данцига, алгоритма двойного поиска? Если влияет, то почему?

8. В графе, изображенном на рис. 3.6, найдите три первые кратчайшие пути между вершиной 1 и всеми остальными вершинами.

9. Покажите, что некоторая неоптимальная строка оценок может остаться неизменной в одной простой итерации алгоритма двойного поиска, однако уже в следующей простой итерации такая строка оценок обязательно изменится.

10. Предположим, что обобщенный алгоритм Флойда был использован для получения длин трех наилучших путей между всеми парами вершин в задаче поиска узких мест. Как полученные результаты могут быть использованы для поиска самих путей?

11. Какие значения следует присваивать компонентам вектора d_{ii} при решении задачи о путях с усилениями?

12. Пусть скорости перекачки нефти между различными емкостями нефтеперерабатывающего завода задаются следующей таблицей:

Емкость	А	Б	В	Г
А	0,00	0,13	0,14	0,15
Б	0,08	0,00	0,13	0,08
В	0,17	0,12	0,00	0,18
Г	0,10	0,06	0,13	0,00

Найдите два наилучших способа перекачки нефти из емкости A в емкость $Г$.

13. Предположим, что после выполнения всех вычислений по алгоритму двойного поиска вы обнаруживаете, что одна из длин дуг была задана неверно. При каких условиях часть полученных результатов может быть оставлена без изменений?

14. Фармацевтическая фирма планирует разработать в течение двенадцати месяцев новый препарат, который мог бы конкурировать с препаратом, недавно выпущенным главными конкурентами данной фирмы.

Разработка нового препарата осуществляется в четыре этапа, которые могут быть пройдены в медленном, нормальном или быстром темпе. Различные темпы прохождения этапов соответствуют различным уровням затрат времени и средств, которые задаются следующей таблицей (в таблице первая цифра — длительность в месяцах, вторая цифра — затраты в тыс. долл.):

Этап / Темп	Теоретические исследования	Лабораторные эксперименты	Одобрение правительства	Сбыт
Медленный	5, 5	3, 6	6, 1	5, 8
Нормальный	4, 7	2, 8	4, 1	4, 10
Быстрый	2, 10	1, 12	2, 3	3, 15

Найдите наилучший для фирмы способ разработки за двенадцать месяцев нового препарата при условии, что затраты фирмы не превысят 25 тыс. долл. Найдите второй наилучший способ достижения той же цели.

15. Управляющий гостиницей должен забронировать номера для новобрачных на следующий месяц. Он получил определенное количество заявок на бронирование с различными датами приезда и отъезда. Каждое возможное бронирование номеров дает гостинице определенный доход, зависящий от того, кем являются клиенты (студенты, служащие, персонал авиакомпаний и т. д.). Каким образом в данном случае использовать алгоритм Дейкстры для выработки расписания бронирования номеров, приносящего гостинице максимальный доход? (*Указание.* Представьте каждую заявку на бронирование номера дугой, соединяющей вершины, соответствующие датам приезда и отъезда; в соответствующем графе будут отсутствовать контуры, и потому к нему может быть применен алгоритм Дейкстры.)

ЛИТЕРАТУРА

1. Dantzig G. B. 1967. All Shortest Routes in a Graph, Theory of Graphs, International Symposium, Rome, Gordon and Breach, New York, pp. 91 — 92, 1966.
2. Dijksatra E. W., A Note on Two Problems in Connexion with Graphs, *Numer Math.*, **1**, pp. 269—271, 1959.
3. Dreyfus S. E., An Appraisal of Some Shortest Path Algorithms, *Operations Research*, **17**, pp. 395—412, 1969.
4. Floyd R. Z., Algorithm 97, Shortest Path, *Comm. ACM*, 5, p. 345, 1962.
5. Ford L. R., Network Flow Theory, Rand Corporation Report, p. 923, 1946.
6. Karp R. M., On the Computational Complexity of Combinatorial Problems, *Networks*, 5, pp. 45—68, 1975.
7. Knuth D. E., The Art Computer Programming, vol. 3, Sorting and Searching Addison-Wesley, Reading, 1973. [Имеется перевод: Кнут Д. Е. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. — М.: Мир, 1979.]
8. Lawler E. L., The Complexity of Combinatorial Computations: A Survey, Proc. 1971 Polytechnic Institute of Brooklyn Symposium on Computers and Automata, 1971.
9. Minieka E. T., On Computing Sets of Shortest Paths in a Graph, *Comm. ACM*, **17**, pp. 351—353, 1974.
10. Minieka E. T., Shier D. R., A Note on an Algebra for the k Best Routes in a Network, *J. IMA*, **11**, pp. 145 — 149, 1973.
11. Shier D. R., Iterative Methods for Determining the k Shortest Paths in a Network, *Networks*, 6, pp. 205 — 230, 1976.
12. Shier D. R., Computational Experience with an Algorithm for Finding the k Shortest Paths in a Network, *J. Res. Natl.* Bur. Std., 78B (July—September), pp. 139 — 165, 1974.
13. Williams T. A., White G. P., A Note on Yen's Algorithms for Finding the Length of All Shortest Paths in N—Node Nonnegative Distance Networks, *J. ACM*, **20**, No. 3, pp. 389 — 390, 1973.
14. Yen J. V., An Algorith for Finding Shortest Routes from All Source Nodes to a Given Destination in General Networks, *Q. Appl. Math.*, **27**, pp. 526 — 530, 1970.
15. Yen J. Y., Finding the Lengths of All Shortest Paths in N—Node Nonnegative Distance Complete Networks Using $1/2N^3$ Additions and N^3 Comparisons, *J. ACM*, **19**, No. 3, pp. 423 — 424, 1973.

ГЛАВА 4

Потоковые алгоритмы

4.1. Введение

Вообще говоря, поток определяет способ пересылки некоторых объектов из одного пункта в другой. Потоки, например, возникают при транспортировке готовой продукции от завода до распределительного склада, при движении людей от мест проживания к местам работы или при доставке писем от отправителей к получателям.

Несмотря на разнообразие ситуаций, связанных с потоками, в них возникает целый ряд достаточно общих проблем. Примерами таких проблем могли бы служить максимизация суммарного объема перевозки в некоторой системе из одной точки в другую; минимизация стоимости пересылки через некоторую систему определенного количества предметов из одной точки в другую; минимизация времени перевозок в заданной системе.

В данной главе потоки будут рассматриваться на графах. При этом будут исследованы задачи, подобные тем, которые были сформулированы выше. Если для исходных потоковых задач соответствующие графовые модели оказываются достаточно адекватными, то результаты реализации этих моделей с помощью соответствующих алгоритмов вполне применимы к исходным проблемам. Нет необходимости доказывать, что получаемые результаты обычно имеют такое же отношение к рассматриваемой практической задаче, что и предлагаемая для нее модель.

Применительно к графам, вообще говоря, поток задает способ пересылки некоторых объектов из одной вершины графа в другую по его дугам (перемещение по дуге осуществляется в заданном на ней направлении). Вершина, из которой начинается перемещение объектов, называется *источником* и обычно обозначается через s. Вершина, в которой заканчивается перемещение объектов, называется *стоком* и обычно обозначается через t. Объекты, которые перемещаются или «протекают» из источника в сток, будут называться *единицами потока* или просто *единицами*. Как указывалось выше, единицами потока в практических задачах могут быть готовые товары, люди, письма, да и многое другое.

Если количество единиц потока, которое может проходить по дуге (x, y), ограничено, то говорят, что дуга (x, y) имеет ограниченную *пропускную способность*. *Максимальную* величину пропуск-

ной способности будем обозначать через $c(x, y)$[1]). Кроме того, будем через $a(x, y)$ обозначать стоимость перемещения единицы потока по дуге (x, y). Заметим, что сеть — это граф, в котором каждой дуге приписана некоторая пропускная способность.

ПРИМЕР 1. В период ведения военных действий для материального обеспечения войск требуется провести конвой, включающий двенадцать транспортных единиц, от военной базы (источник) до места сосредоточения войск (сток). Система дорог, связывающая базу с войсками, может быть представлена графом, в котором дуги соответствуют незащищенным участкам дорог, а вершины — точкам пересечения этих дорог. Из соображений безопасности для каждого участка дорог устанавливается ограничение на число транспортных единиц, которые могут по нему пройти (т. е. задаются пропускные способности соответствующих дуг). Из предыдущего опыта организаторы конвоя знают о величине ожидаемых потерь для каждого участка рассматриваемой системы дорог.

Задачу поиска наилучших маршрутов движения для транспортных средств, составляющих конвой, можно сформулировать как задачу пересылки из источников в сток 12 единиц потока по сети, соответствующей рассматриваемой системе дорог. При этом не должна быть превышена пропускная способность ни в одной дуге и должны быть минимизированы суммарные ожидаемые потери.

Предположим, что имеется граф, в котором некоторое количество единиц потока проходит от источника к стоку и для каждой единицы потока известен маршрут движения. Назовем количество единиц, проходящих по дуге (x, y), потоком в данной дуге. Будем поток в дуге (x, y) обозначать через $f(x, y)$. Очевидно, $0 \leqslant f(x, y) \leqslant c(x, y)$. Дуги графа можно отнести к трем различным категориям: 1) дуги, в которых поток не может ни увеличиваться, ни уменьшаться (множество таких дуг обозначается через N); 2) дуги, в которых поток может увеличиваться (множество таких дуг обозначается через I); 3) дуги, в которых поток может уменьшаться (множество таких дуг обозначается через R).

Например, дуги, имеющие нулевую пропускную способность или значительную стоимость прохождения потока, должны принадлежать множеству N. Дуги, в которых поток меньше пропускной способности, должны принадлежать множеству I. Наконец, дуги, по которым уже проходит некоторый поток, должны принадлежать множеству R. Дуги из множества I называются *увеличивающими*, а дуги из множества R — *уменьшающими*. Очевидно, любая дуга графа принадлежит по крайней мере одному из трех введенных множеств — I, R или N. Вполне возможно, что какая-то дуга принадлежит как множеству I, так и множеству R. Это имеет место в том случае, когда по дуге уже протекает некоторый поток,

[1]) В дальнейшем величина c (x, y) будет называться просто пропускной способностью, как это принято в литературе на русском языке. *Прим. перев.*

который можно увеличивать или уменьшать. Соответствующие дуги называются *промежуточными*.

Обозначим через $i(x, y)$ максимальную величину, на которую может быть увеличен поток в дуге (x, y). Соответственно обозначим через $r(x, y)$ максимальную величину, на которую может быть уменьшен поток в дуге (x, y). Очевидно, $i(x, y) = c(x, y) - f(x, y)$ и $r(x, y) = f(x, y)$.

Предположим, что мы хотим переслать дополнительное количество единиц потока из источника в сток. В принципе возможны несколько способов выполнения данной задачи (если вообще она имеет решение). Первый способ мог бы быть реализован, если бы мы нашли путь P из вершины s в вершину t, целиком состоящий из увеличивающих дуг (рис. 4.1). Сколько в этом случае можно было бы дополнительно переслать единиц потока из s в t по пути P? Поскольку $i(x, y)$ представляет собой максимально возможное увеличение потока в дуге (x, y), то величина дополнительного потока из s в t по пути P составит самое большее

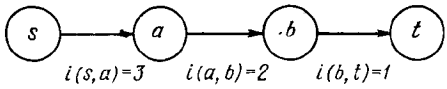


Рис. 4.1. Увеличивающая поток цепь, включающая только прямые дуги.

$$\min_{(x,\ y)\in P} \{i(x, y)\}.$$

Для данных примера, показанного на рис. 4.1, по пути P можно переслать из s в t максимум одну дополнительную единицу потока, поскольку

$$\min\{i(s, a),\ i(a, b),\ i(b,\ t)\} = \min\{3,\ 2,\ 1\} = 1.$$

Второй способ увеличения потока мог бы быть реализован, если бы мы нашли путь P из вершины t в вершину s, целиком состоящий из уменьшающих дуг (рис. 4.2). При этом можно было бы уменьшить поток в каждой дуге (x, y), что привело бы к уменьшению потока из вершины t в вершину s и, следовательно, к увеличению чистого потка из вершины s в вершину t. На какую максимальную величину можно было бы уменьшить поток из t в s по указанному пути? Поскольку в каждой дуге (x, y) пути P поток можно уменьшить самое большее на величину $r(x, y)$, то максимальное уменьшение потока вдоль пути P определяется величиной

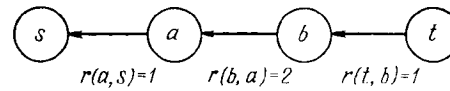


Рис. 4.2. Увеличивающая поток цепь, включающая только обратные дуги.

$$\min_{(x,\ y)\in P} \{r(x, y)\}.$$

Для данных примера, показанного на рис. 4.2, по пути P можно переслать обратно из t в s максимум одну единицу потока, поскольку

$$\min\{r(t, b),\ r(b, a),\ r(a, s)\} = \min\{1, 2, 1\} = 1.$$

Можно ли найти еще какой-либо способ увеличения чистого потока из вершины s в вершину t? Да, можно. Для этого следовало было бы скомбинировать два способа, описанные выше. Последнее означает, что необходимо найти цепь, соединяющую вер-

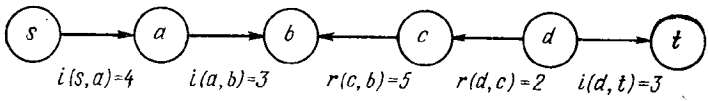


Рис. 4.3. Увеличивающая поток цепь, включающая прямые и обратные дуги.

шины s и t, дуги которой удовлетворяют следующим условиям: 1) все *прямые* дуги цепи, имеющие направление от s и t, принадлежат множеству I; 2) все *обратные* дуги цепи, имеющие направление от t к s, принадлежат множеству R.

Для примера рассмотрим цепь C, соединяющую вершины s и t, которая изображена на рис. 4. 3. Прямыми дугами этой цепи являются дуги (s, a), (a, b) и (d, t), а обратными — дуги (c, b) и (d, c). Если каждая прямая дуга принадлежит множеству I, а каждая обратная дуга — множеству R, то вдоль рассматриваемой цепи можно переслать дополнительный поток из s в t. Это осуществляется путем увеличения потока в прямых дугах, являющихся увеличивающими, и уменьшения потока в обратных дугах, являющихся уменьшающими. Максимальная величина дополнительного потока, который можно переслать вдоль соответствующей цепи из s в t, определится как минимум из следующих двух величин:

$\min\{i(x, y)$: (x, y) — прямая дуга$\}$
$\min\{r(x, y)$: (x, y) — обратная дуга$\}$.

Минимальная из двух указанных величин называется *максимальным увеличением потока* по соответствующей цепи. Для примера, показанного на рис. 4.3, имеем возможное увеличение потока в прямых дугах $i(s, a) = 4$, $i(a, b) = 3$ и $i(d, t) = 3$. Минимум из трех этих величин равен трем. Возможное уменьшение потока в обратных дугах определяют величины $r(c, b) = 5$ и $r(d, c) = 2$. Минимум из двух этих величин равен двум. Итак, максимальное

увеличение потока по цепи C равно $\min(3,2) = 2$. Таким образом, в результате увеличения на две единицы потока в прямых дугах и уменьшения на две единицы потока в обратных дугах по цепи C можно дополнительно переслать из s в t две единицы потока.

Каждая цепь из s в t любого из трех рассмотренных выше типов, по которой могут быть дополнительно посланы единицы потока, называется *увеличивающей цепью*. Каким образом можно определить, имеется ли в исходной сети увеличивающая цепь, ведущая из вершины s в вершину t? Это достаточно просто сделать с помощью описываемого ниже алгоритма поиска увеличивающей цепи. Основная идея алгоритма состоит в построении растущего из вершины s дерева, составленного из окрашенных дуг, по которому из вершины s могут посылаться дополнительные единицы потока. В процессе выполнения алгоритма могут возникнуть две различные ситуации. В первом случае сток t оказывается окрашенным. Тогда в построенном из окрашенных дуг дереве единственная цепь из s в t является увеличивающей поток цепью. Во втором случае сток t не удается окрасить, что означает, что при текущем распределении дуг по множествам R, I и N в исходной сети не существует увеличивающей цепи между s и t. С учетом сказанного дадим формальное описание алгоритма.

Алгоритм поиска увеличивающей цепи

Шаг 1. Определить состав множеств N, I и R. Дуги множества N из дальнейшего рассмотрения исключить (поскольку в них изменения потока невозможны). Окрасить вершину s.

Шаг 2. Окрашивать дуги и вершины в соответствии с приводимыми правилами до тех пор, пока либо не будет окрашена вершина t, либо окраска новых вершин станет невозможной.

Правила окрашивания вершины y и дуги (x, y) при уже окрашенной вершине x состоят в следующем: а) если $(x, y) \in I$, то окрашиваются вершина y и дуга (x, y); б) если $(y, x) \in R$, то окрашиваются вершина y и дуга (y, x); в) в противном случае окрашивание вершины y и дуги (x, y) не производится.

Как уже отмечалось, в случае окрашивания вершины t в сети находится единственная цепь из s в t, включающая окрашенные дуги. (Эта цепь является увеличивающей.) В противном случае, т. е. когда по окончании процедуры окрашивания вершина t не окрашивается, в сети отсутствуют увеличивающие цепи из s в t.

Обоснование алгоритма поиска увеличивающей цепи. Чтобы показать, что рассматриваемый алгоритм позволяет получить увеличивающую цепь из s в t (если такая цепь существует), необходимо доказать следующие три утверждения:

(1) Если в алгоритме вершина t окрашивается, то в исходной сети, действительно, имеется увеличивающая цепь из s в t.

(2) Если в алгоритме вершина t не может быть окрашена, то в исходной сети отсутствуют увеличивающие цепи из s в t.

(3) Выполнение алгоритма завершается за конечное число шагов.

Доказательство утверждения (1). В силу правил, по которым выполняется процедура окрашивания, дуга может быть окрашена только в том случае, если одна из ее концевых вершин окрашена, а другая — нет. Следовательно, в ходе выполнения алгоритма не может быть окрашена дуга, обе концевые вершины которой уже были ранее окрашены, и потому окрашенные дуги не могут образовывать циклы. Поскольку работа алгоритма начинается с окрашивания вершины s, то в последующем окрашенные дуги должны образовывать дерево, включающее вершину s. Поэтому если в алгоритме окрашивается вершина x, то, значит, должна существовать единственная цепь из s в x, включающая окрашенные дуги. В частности, если окрашивается вершина t, то найдется единственная цепь из окрашенных дуг, ведущая из вершины s в вершину t. Эта цепь должна быть увеличивающей, поскольку процедура окрашивания гарантирует, что прямые дуги этой цепи являются увеличивающими, а обратные дуги — уменьшающими.

Доказательство утверждения (2). Если имеется увеличивающая цепь C из s в t, то имеется по крайней мере одна увеличивающая цепь, ведущая из вершины s в каждую из вершин цепи C. Поэтому каждая из вершин цепи C должна быть окрашена, в частности должна быть окрашена и вершина t. Обратно, если вершина t не может быть окрашена, то не существует ни одной увеличивающей поток цепи из s в t.

Доказательство утверждения (3). Алгоритм должен завершиться за конечное число шагов, поскольку в нем по одному разу могут окрашиваться вершины и дуги, число которых конечно.

Итак, нами полностью проведено обоснование алгоритма нахождения увеличивающей поток цепи.

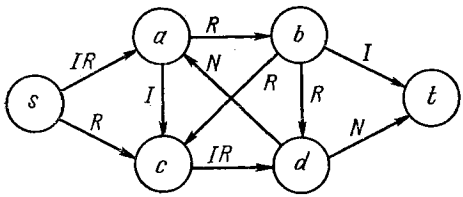


Рис. 4.4. Пример применения алгоритма поиска увеличивающей поток цепи.

ПРИМЕР 2 (применение алгоритма нахождения увеличивающей цепи). Используем рассмотренный только что алгоритм для определения увеличивающей цепи в графе, изображенном на рис. 4.4. Рядом с дугами графа указаны буквы I, R и N, устанавливающие принадлежность дуг соответствующим множествам. При решении данной конкретной задачи процедура окрашивания будет организована так, что для любой окрашенной ранее вершины вначале будет предприниматься попытка окрасить все

соседние с ней вершины и только потом осуществляться переход к выполнению аналогичной операции для другой окрашенной вершины.

Прежде всего окрашивается вершина s. Из вершины s могут быть окрашены вершина a и дуга (s, a), поскольку $(s, a) \in I$. Вершина c и дуга (s, c) не могут быть окрашены из вершины s, поскольку $(s, c) \notin I$. Это завершает процедуру окрашивания вершин и дуг из вершины s.

Далее будем окрашивать вершины и дуги из вершины a. Вершина b и дуга (a, b) не могут быть окрашены, поскольку $(a, b) \in I$. Вершина c и дуга (a, c) могут быть окрашены, поскольку $(a, c) \notin I$. Вершина d и дуга (d, a) не могут быть окрашены из вершины a, поскольку $(d, a) \notin R$. Это завершает процедуру окрашивания из вершины a.

Далее будем окрашивать вершины и дуги из вершины c. Поскольку вершины s и a уже окрашены, их можно не рассматривать. Вершина b и дуга

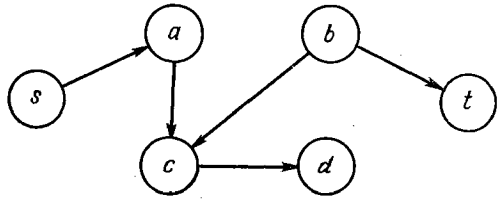


Рис. 4.5. Дерево, состоящее из окрашенных дуг.

(b, c) могут быть окрашены, так как $(b, c) \in R$. Так же могут быть окрашены вершина d и дуга (c, d), так как $(c, d) \in I$. Это завершает процедуру окрашивания из вершины c.

Далее будем окрашивать вершины и дуги из вершины b. Вершины a, c и d, которые уже были окрашены, можно не рассматривать. Вершина t и дуга (b, t) могут быть окрашены, так как $(b, t) \in I$.

На этом процедура окрашивания заканчивается, поскольку оказывается окрашенной вершина t. Подграф, состоящий из окрашенных вершин и дуг исходного графа, изображен на рис. 4.5. Из рисунка видно, что увеличивающей цепью из s в t является цепь (s, a), (a, c), (b, c), (b, t). Максимальное увеличение потока по этой цепи равно

$$\min\{i(s, a), i(a, c), r(b, c), i(b, t)\} = \min\{4, 3, 2, 2\} = 2.$$

Таким образом, потоки в прямых дугах найденной цепи, т. е. в дугах (s, a), (a, c) и (b, t), могут быть увеличены на две единицы. В обратной дуге (b, c) поток может быть уменьшен на две единицы. Последнее подразумевает, что указанные две единицы, ранее протекавшие по дуге (b, c), «возвращаются» по дуге (b, t), а соответствующая «недостача потока» в вершине c компенсируется за счет двух единиц, пропущенных по дуге (a, c).

Как можно видеть из примера 2, алгоритм поиска увеличивающей цепи не дает конкретного указания, в какой последовательности должны окрашиваться вершины и дуги. Это можно делать исходя из какой-либо заданной вершины, пытаясь окрасить максимальное число вершин (как в рассмотренном выше примере), или проводить окрашивание исходя из последней окрашенной вершины. Выбор наиболее эффективной организации процедуры окрашива-

ния является предметом специального исследования и часто зависит от характера более общей задачи, при решении которой в качестве подалгоритма используется алгоритм поиска увеличивающей цепи.

Каждый раз, когда в процессе выполнения рассмотренного алгоритма находится увеличивающая цепь из s в t, из источника в сток может быть послано дополнительное количество единиц потока, не превышающее максимального увеличения потока по данной цепи. При этом в увеличивающих дугах найденной цепи поток возрастает, а в уменьшающих дугах становится меньше на указанную максимальную величину.

4.2. Алгоритм поиска максимального потока

Задача о максимальном потоке состоит в поиске способа пересылки максимального количества единиц потока из источника в сток при условии отсутствия превышения пропускных способностей дуг исходного графа.

ПРИМЕР 1. Представитель бюро путешествий должен в один из дней организовать перелет десяти туристов из Чикаго в Стамбул. Имеется 7 свободных мест на прямой рейс Чикаго—Стамбул, 5 — на рейс Чикаго—Париж и 4 — на рейс Париж—Стамбул. Как должен действовать представитель этого бюро?

Соответствующая задача может быть сформулирована как задача о максимальном потоке, для чего построим сеть, каждая дуга которой соответствует рассматриваемым рейсам. В сети должно быть три дуги, представляющие рейсы Чикаго—Стамбул, Чикаго—Париж и Париж—Стамбул. Припишем каждой дуге пропускную способность, равную числу свободных мест на соответствующий рейс. Пусть эти пропускные способности составляют соответственно 7,5 и 4. Если через построенную сеть можно пропустить из источника (Чикаго) в сток (Стамбул) 10 единиц потока без превышения пропускных способностей дуг, то в выбранный день всю группу туристов можно переправить из Чикаго в Стамбул.

Как отмечалось в разд. 4.1, поток представляет собой любое перемещение соответствующих объектов из источника s в сток t. Будем, как и раньше, через $f(x, y)$ обозначать количество единиц потока, прошедших по дуге (x, y). Для любого потока из s в t количество единиц, выходящих из любой вершины x (при этом только $x \neq s$, $x \neq t$), должно быть равно количеству единиц, входящих в эту вершину x, т. е.

$$\sum_{y \in X} f(x, y) - \sum_{y \in X} f(y, x) = 0 \quad (x \neq s, \quad x \neq t). \tag{1}$$

(Напомним, что через X обозначается множество всех вершин рас-

сматриваемого графа)[1]. Кроме того, количество единиц потока, проходящих по каждой дуге (x, y), не должно превышать пропускной способности этой дуги, т. е.

$$0 \leqslant f(x, y) \leqslant c(x, y) \quad (x, y) \in A. \tag{2}$$

(Напомним, что через A обозначается множество всех дуг рассматриваемого графа.)

Наконец, суммарное число единиц потока, выходящих из источника, должно быть равно суммарному числу единиц потока, входящих в сток. Если указанное число обозначить через v, то это условие можно представить следующим образом:

$$\sum_{y \in X} f(x, y) - \sum_{y \in X} f(y, s) = v, \tag{3}$$

$$\sum_{y \in X} f(y, t) - \sum_{y \in X} f(t, y) = v. \tag{4}$$

Любой поток из s в t должен удовлетворять всем четырем условиям (1)—(4). И наоборот, если может быть найден набор величин $f(x, y)$ при $(x, y) \in A$, для которого выполняются эти четыре условия, то такой набор представляет собой поток из источника s в сток t. Здесь мы не останавливаемся на деталях, рассмотреть которые читатель может самостоятельно.

Итак, набор величин $f(x, y)$ при $(x, y) \in A$ представляет поток тогда и только тогда, когда он удовлетворяет условиям (1)—(4).

Теперь можно сказать, что задача о максимальном потоке состоит в поиске потока, удовлетворяющего условиям (1)—(4), для которого величина v максимальна. Читатель, знакомый с линейным программированием, заметит, что задача максимизации величины v при ограничениях (1)—(4) является задачей линейного программирования, и, следовательно, она могла бы быть решена с помощью симплекс-метода специально предназначенного для решения задач линейного программирования. Однако использование симплекс-метода для решения задачи о максимальном потоке напоминало бы стрельбу из пушек по воробьям. Существует более элегантный и естественный подход к решению рассматриваемой задачи. Мы имеем в виду алгоритм поиска максимального потока, разработанный Фордом и Фалкерсоном [2].

Идея, лежащая в основе алгоритма Форда и Фалкерсона, довольно проста и состоит в следующем. Выбирается некоторый на-

[1]) В соотношении (1) не совсем точно описаны множества суммирования. Здесь и далее следовало бы заменить X в первой сумме на $A(x)$ и во второй — на $B(x)$, где $A(x)$ — множество вершин y, для которых в исходном графе существует дуга (x, y), а $B(x)$ — множество вершин y, для которых в исходном графе существует дуга (y, x). — *Прим. перев.*

чальный поток из s в t и с помощью алгоритма поиска увеличивающей цепи выполняется поиск увеличивающей цепи. Если он оказывается успешным, то поток вдоль найденной цепи увеличивается до максимально возможного значения. Затем выполняется поиск новой увеличивающей цепи и т. д. Если на каком-то этапе этой процедуры увеличивающую поток цепь найти не удается, выполнение алгоритма заканчивается: текущий поток из s в t является максимальным.

Имея в виду высказанные выше соображения, мы можем теперь формально описать алгоритм поиска максимального потока из s в t в сети, представляющей собой граф с заданными на дугах пропускными способностями.

Алгоритм поиска максимального потока

Шаг 1. Выбрать любой начальный поток из вершины s (источник) в вершину t (сток), т. е. любой набор величин $f(x, y)$, удовлетворяющий соотношениям (1)—(4). Если ни один из начальных потоков из s в t неизвестен, задать начальный поток, положив для всех дуг (x, y) $f(x, y) = 0$.

Шаг 2. Для всех дуг (x, y) проделать следующее. Если $f(x, y) < c(x, y)$, то положить $i(x, y) = c(x, y) - f(x, y)$ и считать дугу (x, y) принадлежащей множеству I. Если $f(x, y) > 0$, то положить $r(x, y) = f(x, y)$ и считать дугу (x, y) принадлежащей множеству R.

Шаг 3. На множествах I и R, сформированных на предыдущем шаге, применить к исходному графу алгоритм поиска увеличивающей цепи. Если при этом увеличивающую поток цепь найти не удается, закончить процедуру алгоритма: текущий поток является максимальным. В противном случае осуществить максимально возможное увеличение потока вдоль найденной увеличивающей цепи. Затем возвратиться к шагу 2.

Обоснование алгоритма поиска максимального потока. Чтобы показать, что данный алгоритм позволяет получить максимальный поток из s в t, необходимо доказать следующие три утверждения:

(1) Алгоритм действительно формирует некоторый поток.

(2) Сформированный поток — максимальный.

(3) Выполнение алгоритма завершается за конечное число шагов.

Доказательство утверждения (1). Для доказательства этого утверждения достаточно отметить, что работа алгоритма начинается с рассмотрения некоторого потока, сформированного на шаге 1 на всех шагах выполнения алгоритма при увеличении потока вдоль соответствующей цепи условия (1)—(4) не нарушаются. Следовательно, по окончании выполнения алгоритма формируется некоторый поток из s в t.

Доказательство утверждения (*2*). Напомним приведенное в гл. 1 определение разреза: разрезом графа называется любое подмножество дуг, удаление которых из графа делает его несвязным. Напомним также, что простым разрезом считается разрез, не содержащий в себе никакого другого, отличного от него разреза. Например, в графе, изображенном на рис. 4.6, дуги (s, a) и (s, b) образуют разрез, так как удаление этих дуг из графа привело бы к образованию в нем двух компонентов. Указанный разрез является простым, поскольку ни одна из входящих в него дуг сама по себе разрезом не является.

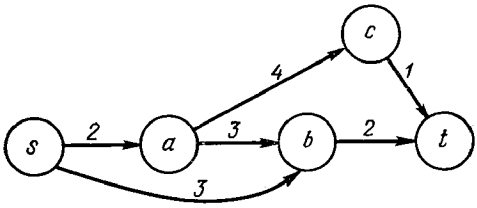


Рис. 4.6. Пример применения алгоритма поиска максимального потока.

Рассмотрим любой простой разрез, отделяющий источник и сток таким образом, что они оказываются в разных компонентах графа, образующихся после удаления из него дуг разреза. Обозначим тот компонент, который содержит вершину s, через X_s, и тот компонент, который содержит вершину t, через X_t. Каждая дуга рассматриваемого разреза либо (*а*) имеет конечную вершину в X_s и начальную вершину в X_t, либо (б) имеет конечную вершину в X_t и начальную вершину в X_s. Будем называть сумму пропускных способностей дуг типа (б) пропускной способностью разреза. Поскольку разрез отделяет вершину s от вершины t, то, очевидно, невозможно пропустить через сеть единиц потока больше, чем это определено величиной пропускной способности разреза: через разрез должны пройти все единицы потока. В общем случае максимальная величина потока из s в t обязательно должна быть меньше или равна наименьшей величине пропускной способности разрезов, отделяющих s от t.

Таким образом, чтобы показать, что в алгоритме действительно определяется максимальный поток, достаточно построить разрез, отделяющий s от t, пропускная способность которого равна величине потока, сформированного в процессе выполнения алгоритма. Это можно сделать с учетом следующих соображений. Выполнение алгоритма заканчивается, когда не может быть найдена ни одна увеличивающая поток цепь из s в t. При этом в результате последнего применения к соответствующему графу алгоритма поиска увеличивающей цепи вершина t не могла быть окрашена. Рассмотрим разрез, состоящий из дуг, у которых одна из конечных вершин оказывается окрашенной, а другая — неокрашенной в результате выполнения последней процедуры окрашивания вершин исход-

ного графа. Поскольку здесь вершина s окрашена, а вершина t не окрашена, то построенный разрез отделяет s от t. Пропускная способность того разреза равна сумме пропускных способностей дуг, которые имеют окрашенную начальную вершину и неокрашенную конечную вершину.

По окончании выполнения алгоритма поиска максимального потока в каждой дуге с окрашенной начальной и неокрашенной конечной вершинами величина потока совпадает с величиной пропускной способности; в противном случае неокрашенная концевая вершина каждой из этих дуг была бы окрашена в результате выполнения алгоритма поиска увеличивающей поток цепи. С другой стороны, по окончании выполнения алгоритма поиска максимального потока в каждой дуге с неокрашенной начальной вершиной и окрашенной конечной вершиной поток отсутствует; в противном случае неокрашенная концевая вершина каждой из этих дуг была бы окрашена в результате выполнения алгоритма поиска увеличивающей поток цепи.

Очевидно, каждая единица потока должна пройти через разрез по крайней мере один раз. Поскольку во всех дугах рассматриваемого разреза, направленных от стока к источнику, поток отсутствует, то через данный разрез каждая единица проходит ровно один раз. Поэтому общее количество единиц потока, прошедших через разрез от s к t, в точности равно величине его пропускной способности, так как в каждой дуге разреза, направленной от источника к стоку, величина потока совпадает с величиной пропускной способности.

Доказательство утверждения (*3*). Чтобы показать, что выполнение алгоритма завершается за конечное число шагов, необходимо сделать предположение о целочисленности пропускных способностей всех дуг и целочисленности начального потока. С практической точки зрения данное предположение не является слишком жестким, поскольку в большинстве случаев величины пропускных способностей могут быть округлены до целых чисел, что не оказывает существенного влияния на получаемые результаты.

Рассматриваемый алгоритм может не завершиться за конечное число шагов лишь в том случае, когда он формирует неограниченное количество увеличивающих цепей. Однако при поиске любой такой цепи значение v полного потока из s в t возрастает на положительную целую величину, так как потоки во всех дугах и их пропускные способности являются на протяжении всей процедуры выполнения алгоритма целочисленными. Но величина v ограничена сверху пропускной способностью любого разреза, отделяющего s от t. Следовательно, возможно лишь конечное число увеличений потока, что и завершает доказательство утверждения (3) и обоснование алгоритма поиска максимального потока.

ПРИМЕР 2. Применим алгоритм поиска максимального потока к графу, изображенному на рис. 4.6, где рядом с дугами указаны их пропускные способности. Опишем по шагам процедуру выполнения алгоритма в данном конкретном случае.

Шаг 1. Выполнение алгоритма начинается с задания нулевого потока, т. е. для всех дуг (x, y) рассматриваемого графа полагается $f(x, y) = 0$.

Шаг 2. Поскольку для всех дуг оказывается, что $f(x, y) < c(x, y)$, каждая дуга может быть отнесена к множеству I, при этом $i(x, y) = c(x, y) - f(x, y) = c(x, y)$. Поскольку для всех дуг $f(x, y) = 0$, на данном шаге ни одну из дуг нельзя отнести к множеству R.

Шаг 3. Используется алгоритм поиска увеличивающей цепи для определения соответствующей цепи из s в t. Легко видеть, что на данном этапе выполнения алгоритма на графе имеется несколько таких увеличивающих цепей. Предположим, что в используемом на данном шаге алгоритме формируется цепь (s, a), (a, b), (b, t). Максимальное увеличение потока по этой цепи составляет

$$\min\{i(s, a), i(a, b), i(b, t)\} = \min\{2, 3, 2\} = 2.$$

Итак, поток в каждой из трех дуг полученной цепи увеличивается на две единицы: $f(s, a) = 2$, $f(a, b) = 2$, $f(s, b) = 0$, $f(b, t) = 2$, $f(a, c) = 0$ и $f(c, t) = 0$. После этого осуществляется возврат к шагу 2.

Шаг 2. Для новых значений потока в дугах исходного графа пересчитываются величины $i(x, y)$ и $r(x, y)$; кроме того, корректируется состав множеств I и R. Ниже приводятся соответствующие выкладки:

$f(s, a) = 2 = c(s, a)$	$(s, a) \notin I$	$(s, a) \in R$, $r(s, a) = 2$,
$f(a, b) = 2 < c(a, b)$	$(a, b) \in I$, $i(a, b) = 1$	$(a, b) \in R$, $r(a, b) = 2$,
$f(b, t) = 2 = c(b, t)$	$(b, t) \notin I$	$(b, t) \in R$, $r(b, t) = 2$,
$f(s, b) = 0 < c(s, b)$	$(s, b) \in I$, $i(s, b) = 3$	$(s, b) \notin R$,
$f(a, c) = 0 < c(a, c)$	$(a, c) \in I$, $i(a, c) = 4$	$(a, c) \notin R$.
$f(c, t) = 0 < c(c, t)$	$(c, t) \in I$, $i(c, t) = 1$	$(c, t) \notin R$.

Шаг 3. Снова используется алгоритм поиска увеличивающей цепи для определения соответствующей цепи из s в t. В данном случае увеличивающая поток цепь является единственной — это цепь (s, b), (a, b), (a, c), (c, t). Ее и формирует указанный алгоритм. Максимально возможное увеличение потока вдоль найденной цепи равно

$$\min\{i(s, b), r(a, b), i(a, c), i(c, t)\} = \min\{3, 2, 4, 1\} = 1.$$

Таким образом, из вершины s в вершину t может быть дополнительно пропущена одна единица потока. При этом значение потока в каждой из трех прямых дуг (s, b), (a, c) и (c, t), принадлежащих найденной цепи, на единицу увеличится, а значение потока в обратной дуге (a, b) этой цепи на единицу уменьшится. Итак, теперь потоки в дугах рассматриваемого графа будут иметь слеующие значения: $f(s, a) = 2$, $f(a, b) = 1$, $f(b, t) = 2$, $f(s, b) = 1$, $f(a, c) = 1$, $f(c, t) = 1$. Каждая из трех единиц построенного к настоящему моменту потока проходит по следующим маршрутам:

1 единица проходит из s в t по пути (s, a), (a, b), (b, t);

1 единица проходит из s в t по пути (s, b), (b, t);

1 единица проходит из s в t по пути (s, a), (a, c), (c, t).

Далее осуществляется возврат к шагу 2.

Шаг 2. Для новых значений потока в дугах исходного графа пересчитываются величины $i(x, y)$ и $r(x, y)$. Кроме того, корректируется состав множеств I и R. Ниже приводятся соответствующие выкладки:

$f(s, a) = 2 = c(s, a)$	$(s, a) \notin I$	$(s, a) \notin R,\ r(s, a) = 2,$
$f(a, b) = 1 < c(a, b)$	$(a, b) \in I,\ i(a, b) = 2,$	$(a, b) \in R,\ r(a, b) = 1,$
$f(b, t) = 2 = c(b, t)$	$(b, t) \in I,$	$(b, t) \in R,\ r(b, t) = 2,$
$f(s, b) = 1 < c(s, b)$	$(s, b) \in I,\ i(s, b) = 2,$	$(s, b) \in R,\ r(s, b) = 1,$
$f(a, c) = 1 < c(a, c)$	$(a, c) \in I,\ i(a, c) = 3,$	$(a, c) \in R,\ r(a, c) = 1,$
$f(c, t) = 1 = c(c, t)$	$(c, t) \notin I,$	$(c, t) \in R,\ r(c, t) = 1.$

Шаг 3. Вновь используется алгоритм поиска увеличивающей цепи для определения соответствующей цепи из s в t. При этом обнаруживается, что при текущих значениях потока в дугах графа в нем не существует ни одной увеличивающей цепи. В процессе выполнения алгоритма в исходном графе оказываются окрашенными вершины s, b, a, c (причем в указанном порядке), однако вершина t не окрашивается. Поскольку увеличивающую поток цепь найти не удается, алгоритм поиска максимального потока заканчивает свою работу. Последний из построенных потоков является максимальным. Итак, в рассмотренном примере из s в t может быть пропущено 3 единицы потока.

Заметим, что последнее применение алгоритма поиска увеличивающей цепи приводит к разрезу из дуг (b, t), (c, t), имеющих по одной концевой и одной неокрашенной вершинам. Пропускная способность этого разреза равна $c(b, t) + c(c, t) = 2 + 1 = 3$, что совпадает с минимально возможным числом единиц потока из s в t.

В оставшейся части раздела будут описаны две важные модификации алгоритма поиска максимального потока. В первой модификации гарантируется конечность алгоритма поиска максимального потока в самом общем случае, когда нельзя предполагать целочисленность значений потоков и пропускных способностей дуг. Вторая модификация алгоритма позволяет решать задачу о максимальном потоке для графа, имеющего более одного источника и более одного стока.

Конечный вариант алгоритма поиска максимального потока

При доказательстве конечности алгоритма поиска максимального потока выдвигалось предположение о целочисленности значений начального потока и пропускных способностей всех дуг. Если значения пропускных способностей некоторых дуг не являются целыми числами, нет гарантии, что алгоритм поиска максимального потока закончит свою работу за конечное число шагов. Пример графа с нецелочисленными значениями пропускных способностей, применение к которому алгоритма поиска максимального потока приводит к неограниченному числу шагов увеличения потока, можно найти в работе Форда и Фалкерсона [2]. Однако Джонсон в 1966 г. [5] и Эдмондс и Карп в 1972 г. [1] предложили две раз-

личные модификации алгоритма поиска максимального потока, которые гарантируют конечность числа шагов увеличения потока. Ниже описывается модификация алгоритма поиска максимального потока, предложенная Эдмондсом и Карпом.

В процессе выполнения алгоритма поиска увеличивающей цепи существует возможность выбора следующей окрашиваемой дуги. В предлагаемой модификации общего алгоритма выбор следующей окрашиваемой дуги осуществляется вполне определенным образом. Будем нумеровать вершины в порядке их окрашивания (вершина s естественно получает номер 1). В первую очередь попытаемся окрасить дуги, инцидентные вершины 1. Затем будем пытаться окрашивать дуги, инцидентные вершине 2 и т. д.

Заметим, что если процедура окрашивания организована так, как это описано выше, то цепь из окрашенных дуг, соединяющая любую вершину с источником s, содержит минимально возможное количество дуг. Следовательно, каждая формируемая алгоритмом увеличивающая цепь содержит минимально возможное число дуг.

В увеличивающей цепи дуга (x, y) называется *узким местом*, если именно эта дуга ограничивает увеличение потока. Если дуга (x, y) является в соответствующей цепи узким местом, то при максимально возможном увеличении потока вдоль этой цепи величина $f(x, y)$ либо увеличивается до $c(x, y)$, либо уменьшается до 0.

Предположим, что дуга (x, y) является узким местом в двух формируемых соответствующим алгоритмом цепях C_1 и C_2. При этом будем считать, что C_2 формируется после C_1 и что ни в одной из увеличивающих цепей, формируемых «между» C_1 и C_2, дуга (x, y) не является узким местом. Без ограничения общности можно считать, что после увеличения потока по цепи C_1 получим $f(x, y) = C(x, y)$. Таким образом, дуга (x, y) является прямой в цепи C_1 и обратной в цепи C_2. Для $i = 1, 2$ обозначим число дуг в цепи C_i между вершинами m и n через $C_i(m, n)$. Из того факта, что модифицированная процедура окрашивания вершин и дуг приводит к построению увеличивающих цепей из s в любую вершину с минимально возможным числом дуг, вытекают следующие соотношения:

$$C_1(s, y) \leqslant C_2(s, y)$$
$$C_1(x, t) \leqslant C_2(x, t)$$
$$\begin{aligned} C_1(s, t) &= C_1(s, y) + C_1(x, t) - 1 \\ &\leqslant C_2(s, y) + C_2(x, t) - 1 \\ &= C_2(s, t) - 2. \end{aligned}$$

Итак, $C_1(s, t) \leqslant C_2(s, t) - 2$. Поэтому каждый раз, когда дуга (x, y) становится узким местом, количество дуг в соответствующей цепи увеличивается по меньшей мере на 2. Поскольку ни одна уве-

личивающая цепь из s в t не содержит более чем $(n-1)$ дугу (напомним, что n — число вершин графа), любая дуга в роли узкого места в соответствующих цепях не может выступать более $n/2$ раз. Но при каждом увеличении потока из s в t по крайней мере одна дуга является узким местом. Следовательно, общее количество увеличений потока не может превысить величины $n|A|/2$.

Аналогичный результат можно получить в случае, когда при изменении потока вдоль цепи C_1 поток в дуге (x, y) не увеличивается до $C(x, y)$, а уменьшается до 0.

ПРИМЕР 3. Используем модифицированную процедуру окрашивания для поиска максимального потока на графе, изображенном на рис. 4.6. Будем считать, что перед началом работы алгоритма во всех дугах поток является нулевым.

Пусть прежде всего окрашивается вершина s, которой присваивается номер 1. Просмотрим неокрашенные ребра, инцидентные вершине s. При этом выясняется, что могут быть окрашены ребро (s, a) и вершина a. Вершине a присваивается номер 2. Далее окрашиваются дуга (s, b) и вершина b. Вершине b присваивается номер 3. Этим завершается процедура окрашивания дуг, инцидентных вершине s.

Поскольку номер 2 получила вершина a, теперь необходимо просмотреть неокрашенные дуги, инцидентные этой вершине. В результате такого просмотра окрашиваются дуга (a, c) и вершина c. Вершине присваивается номер 4.

Поскольку номер 3 имеет вершина b, далее просматриваются неокрашенные дуги, инцидентные этой вершине. В результате такого просмотра окрашиваются дуга (b, t) и вершина t. Таким образом, определяется увеличивающая поток цепь (s, b), (b, t). По этой цепи поток из s в t может быть увеличен на две единицы.

После этого имеющаяся нумерация вершин и окраска вершин и дуг снимаются и вся процедура окрашивания повторяется. Она снова начинается с окрашивания вершины s, которая имеет номер 1. Просмотр неокрашенных дуг, инцидентных вершине s, приводит к окрашиванию дуги (s, a) и вершины a, а также к окрашиванию дуги (s, b) и вершины b. Вершинам a и b присваиваются соответственно номера 2 и 3.

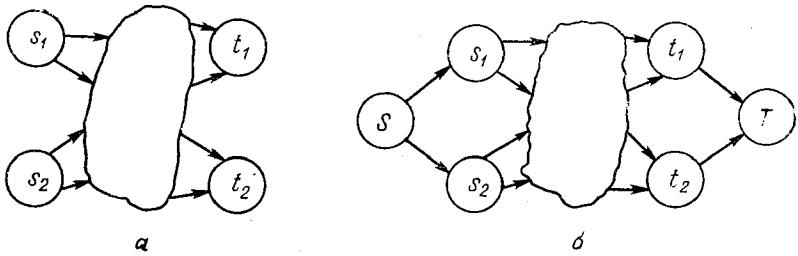
Далее просматриваются неокрашенные дуги, инцидентные вершине a. Это приводит к окрашиванию дуги (a, c) и вершины c. Вершине c присваивается номер 4.

Далее просматриваются неокрашенные дуги, инцидентные вершине b. В результате такого просмотра не удается окрасить ни одной новой дуги и вершины. Просмотр неокрашенных дуг, инцидентных следующей вершине c, приводит к окрашиванию дуги (c, t) и вершины t. При этом определяется увеличивающая цепь (s, a), (a, c), (c, t). По этой цепи поток из s в t может быть увеличен на единицу. Читателю остается убедиться в том, что дальнейшее увеличение сформированного потока невозможно.

Модификация алгоритма поиска максимального потока при нескольких источниках и стоках

В заключение рассмотрим задачу о максимальном потоке на графе, в котором может быть более одного источника и более одного стока. Можно ли в данном случае воспользоваться алгоритмом поиска максимального потока, который, вообще говоря, выпол-

няется на сети с одним источником и одним стоком? Оказывается, можно. Для этого необходимо ввести две новые вершины — главный источник S и главный сток T. Соединим главный источник S с каждым из источников исходной сети s_1, s_2 ... дугами (S, s_1), (S, s_2), ..., считая пропускную способность каждой из этих дуг не-



а *б*

Рис. 4.7. Граф с несколькими источниками и стоками: *а* — исходный граф; *б* — расширенный граф.

ограниченной. Соединим стоки исходной сети t_1, t_2, ... с главным стоком T дугами (t_1, T), (t_2, T), ..., считая пропускную способность каждой из этих дуг неограниченной.

Очевидно, любой поток в новом, расширенном графе соответствует потоку в исходном графе из имеющихся в нем источников в имеющиеся стоки; справедливо и обратное. Более того, максимальному потоку в расширенном графе соответствует максимальный поток в исходном графе. Таким образом, алгоритм поиска максимального потока может быть применен к расширенному графу, и полученный в результате этого поток будет определять максимальный поток в исходном графе (рис. 4.7).

4.3. Алгоритм поиска потока минимальной стоимости

В предыдущем разделе мы рассмотрели задачу о максимальном потоке, при решении которой определялось максимальное число единиц потока, пересылаемых из источника в сток в графе с заданными на дугах пропускными способностями. В данном разделе мы рассмотрим задачу, состоящую в организации пересылки с минимальными затратами заданного количества v единиц потока из источника в сток в графе с заданными на дугах пропускными способностями и стоимостями прохождения одной единицы потока.

ПРИМЕР 1. Некоторый предприниматель имеет возможность выбирать маршруты для перевозки готовых изделий от своего предприятия до склада. С выбором различных маршрутов перевозок связаны те или иные затраты. По каждому маршруту можно пропустить ограниченное количество изделий. Каков наилучший с точки зрения затрат способ перевозки готовых изделий от предприятия до склада?

Поставим в соответствие пунктам расположения предприятия вершину s и склада — вершину t некоторого графа. Представим отдельными вершинами каждое пересечение возможных маршрутов перевозки. Непересекающиеся отрезки маршрутов изобразим дугами, соединяющими соответствующие вершины. Пусть пропускная способность каждой дуги совпадает с максимальным количеством изделий, которое можно пропустить по соответствующему отрезку маршрута. Пусть стоимость пересылки единицы потока по каждой дуге совпадает с удельными затратами на перевозку каждого изделия по соответствующему отрезку.

Задачу предпринимателя можно теперь рассматривать как задачу поиска в построенном графе потока минимальной стоимости из s в t.

ПРИМЕР 2. Предположим, что агент бюро путешествий занимается организацией переезда группы из 75 человек, которую необходимо одновременно всю или по частям на следующий день отправить самолетом из Спрингфилда в Стамбул. Каков наилучший с точки зрения затрат вариант перелета группы между указанными пунктами?

Данная задача может быть сформулирована как задача о потоке минимальной стоимости на соответствующем графе. Вершины этого графа соответствуют аэропортам, которые могут быть промежуточными для полетов между Спрингфилдом и Стамбулом, а дуги представляют рейсовые полеты следующего дня между соответствующими аэропортами. Пропускная способность каждой дуги в построенном таким образом графе определяется количеством имеющихся свободных мест на соответствующий рейс, а затраты, характеризующие дуги, совпадают со стоимостью одного билета на соответствующий рейс.

Описываемый далее алгоритм решения задачи о потоке минимальной стоимости разработан Фордом и Фалкерсоном [2]. Этот алгоритм, соответственно названный *алгоритмом поиска потока минимальной стоимости*, является некоторым обобщением алгоритма поиска максимального потока, обсуждавшегося в разд. 3.2.

Итак, рассмотрим задачу о потоке минимальной стоимости и алгоритм ее решения. Пусть $a(x, y)$ обозначает стоимость прохождения единицы потока по дуге (x, y). Сначала будем предполагать, что каждая из величин $a(x, y)$ является целочисленной. Данное предположение не является слишком жестким, поскольку стоимости обычно выражаются в долларах или в центах, т. е. представляют собой целые числа. (В конце раздела будет представлена модификация алгоритма поиска потока минимальной стоимости, для которой допускается нецелочисленность величин стоимостей $a(x, y)$.)

Как и раньше, обозначим через $f(x, y)$ количество единиц потока, приходящих по дуге (x, y). Естественно, $f(x, y) \geqslant 0$. Обозначим, кроме того, через v количество единиц потока, которое нужно переслать из источника в сток.

Тогда задача о потоке минимальной стоимости может быть представлена следующим образом:

найти

$$\min \left\{ \sum_{(x,\ y)} a(x,\ y) f(x,\ y) \right\} \tag{5}$$

при ограничениях

$$\sum_y [f(s, y) - f(y, s)] = v, \tag{6}$$

$$\sum_y [f(x, y) - f(y, x)] = 0 \quad (x \neq s, x \neq t), \tag{7}$$

$$\sum_y [f(t, y) - f(y, t)] = -v, \tag{8}$$

$$0 \leqslant f(x, y) \leqslant c(x, y) \text{ для всех } x, y. \tag{9}$$

Сумма, входящая в соотношение (5), представляет собой общую стоимость потока. Уравнение (6) показывает, что чистый суммарный поток из источника s должен быть равен v. Уравнение (7) показывает, что чистый поток из любой вершины x, не совпадающей ни с источником s, ни со стоком t, должен быть равен 0. Уравнение (8) показывает, что чистый поток из стока t равен $(-v)$. Наконец, условие (9) описывает требование, согласно которому поток в каждой дуге должен иметь величину, находящуюся в интервале от 0 до значения пропускной способности данной дуги.

Как и задача о максимальном потоке, рассматриваемая здесь задача о потоке минимальной стоимости является задачей линейного программирования. (Заметим, что по существу задачу о максимальном потоке можно рассматривать как частный случай задачи о потоке минимальной стоимости, в котором все дуговые стоимости равны 0, а v совпадает с величиной максимального потока.)

Предположим, что соотношение (5) было бы заменено следующим соотношением:

$$\max \left\{ p - \sum_{(x, y)} a(x, y) f(x, y) \right\}, \tag{10}$$

где p — достаточно большое число (например, p может быть любым числом, бо́льшим максимальной стоимости прохождения единицы потока из источника s в сток t). Если интерпретировать величину p как доход, получаемый в результате пересылки одной единицы потока из s в t, то величину, определяемую соотношением (10), можно интерпретировать как максимально возможный чистый доход от пересылки v единиц потока из s в t за вычетом стоимости их пересылки через исходную сеть. В рамках данной интерпретации легко увидеть, что любой поток, доставляющий максимум в соответствии с соотношением (10) , одновременно будет обеспечивать минимум в соответствии с соотношением (5). Справедливо и обратное.

Алгоритм поиска потока минимальной стоимости выполняет следующие действия. Вначале из s в t пересылается как можно больше единиц потока, полная стоимость прохождения по сети каждой из которых равна 0. Затем из s в t пересылается как можно

больше единиц потока, полная стоимость прохождения по сети каждой из которых составляет 1. Далее процедура выполнения алгоритма продолжается аналогичным образом, при этом полная стоимость прохождения по сети единицы потока каждый раз увеличивается на единицу. Работа алгоритма заканчивается либо тогда, когда через сеть удается пропустить заданное число v единиц потока, либо когда ни одной дополнительной единицы потока по сети пропустить нельзя (в зависимости от того, какое из указанных событий произойдет раньше). Иначе говоря, в алгоритме поиска потока минимальной стоимости многократно решается задача, задаваемая соотношениями (6)—(10), сначала для $p = 0$, затем для $p = 1$, затем для $p = 2$ и т. д.

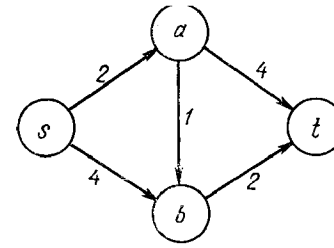


Рис. 4.8. Пример для задачи о потоке минимальной стоимости (рядом с дугами графа указаны их стоимости).

Предположим, что через исходную сеть было пропущено максимально возможное количество единиц потока, имеющих общую стоимость прохождения через сеть, равную $(p-1)$ или меньшей величине. Каким образом можно определить способ пересылки новых единиц потока, имеющих общую стоимость прохождения в p единиц? В соответствии с алгоритмом поиска потока минимальной стоимости это реализуется путем выявления увеличивающей цепи, стоимость прохождения по которой единицы потока составляет величину p. Поясним используемый в алгоритме прием на графе, изображенном на рис. 4.8. Наименее дорогим путем пересылки единицы потока из s в t в данном графе является путь (s, a), (a, b), (b, t). Стоимость прохождения единицы потока по этому пути составляет $2 + 1 + 2 = 5$ единиц. Если $v = 1$, то прохождение соответствующей единицы определяет поток минимальной стоимости.

Предположим, что в рассматриваемом примере $v = 2$. Легко видеть, что единственная увеличивающая поток цепь, по которой можно провести дополнительную единицу потока из s в t, — это цепь (s, b), (a, b), (a, t). Если воспользоваться для проведения второй единицы потока указанной цепью, то получается следующая картина прохождения обеих единиц потока через исходную сеть. Вторая единица потока как бы возвращает первую единицу потока в вершину a. При этом можно считать, что первая единица попадает из s в t по пути (s, a), (a, t). Далее вторая единица потока заменит первую единицу потока в вершине b. При этом можно считать, что вторая единица потока попадает из s в t по пути (s, b), (b, t). Полная стоимость прохождения обеих единиц потока из s в t определится

как $a(s, a) + a(a, t) + a(s, b) + a(b, t) = 2 + 4 + 4 + 2 = 12$; она превосходит стоимость прохождения одной единицы на 7. Это возрастание общей стоимости при использовании увеличивающей цепи (s, b), (a, b), (a, t) складывается из стоимости «+4» прохождения единицы потока по дуге (s, b), из стоимости «—1», связанной с освобождением дуги (a, b) от прохождения по ней одной единицы потока, и из стоимости «+4» прохождения единицы потока по дуге (a, t).

Таким образом, рассматриваемый алгоритм должен найти такую увеличивающую цепь, в которой сумма стоимостей для прямых дуг за вычетом суммы стоимостей обратных дуг равна p. Этот поиск осуществляется с помощью приписывания каждой вершине x целого числа $p(x)$. Эти числа выбираются так, что $p(s) = 0$, $p(t) = p$ для всех вершин x, отличных от s и t, и $0 \leqslant p(x) \leqslant p$. При этом изменения потока допускаются только в тех дугах, для которых

$$p(y) - p(x) = a(x, y). \tag{11}$$

Ясно, что если алгоритм находит увеличивающую цепь, полностью состоящую из дуг, удовлетворяющих соотношению (11), то приращение стоимости для каждой новой единицы потока, проходящей из s в t, будет равно p.

С учетом всех этих соображений мы теперь можем формально описать алгоритм поиска потока минимальной стоимости.

Алгоритм поиска потока минимальной стоимости

Шаг 1. (*Задание начальных условий.*) Принять значение $f(x, y)$ в каждой дуге (x, y) равным 0. Кроме того, положить $p(x) = 0$ для всех вершин x.

Шаг 2. (Выявление дуг, в которых допускается изменение потока.) Сформировать множество I, включить в него дуги (x, y), для которых

$$p(y) - p(x) = a(x, y) \text{ и } f(x, y) < c(x, y).$$

Сформировать множество R, включив в него дуги, для которых

$$p(y) - p(x) = a(x, y) \text{ и } 0 < f(x, y).$$

Сформировать множество N, включить в него дуги, не вошедшие ни в I, ни в R.

(На следующих этапах выполнения алгоритма изменения потока будут допускаться только в дугах, которые принадлежат I и R. Следовательно, потоки могут изменяться только в тех дугах, для которых выполняется соотношение (11).)

Шаг 3. (*Изменение потока.*) Применить алгоритм поиска максимального потока к исходной сети при найденном на шаге 2 рас-

пределении ее дуг по множествам I, R и N. Выполнение данного алгоритма заканчивается либо тогда, когда оказывается, что из s в t уже передано v единиц потока, либо тогда, когда выясняется, что при текущем разбиении дуг на множества I, R и N полученный поток максимален. В первом случае закончить выполнение процедуры алгоритма: полученный поток заданной суммарной величины v является потоком минимальной стоимости. Во втором случае проверить, не является ли полученный поток максимальным в исходном графе. (Это осуществляется путем исследования разреза, определяемого по окончании процедуры окрашивания в соответствии с алгоритмом поиска увеличивающей цепи[1]). Рассматриваемый поток является максимальным только в том случае, когда указанный разрез является насыщенным.) Если это так, закончить выполнение алгоритма: найденный поток является в исходном графе максимальным потоком из s в t и имеет при этом минимальную общую стоимость прохождения по соответствующей сети. В противном случае перейти к шагу 4.

Шаг 4. (*Изменение вершинных чисел.*) Исходной информацией для выполнения данного шага являются результаты окрашивания вершин в алгоритме поиска увеличивающей цепи. Напомним, что последний является подалгоритмом алгоритма поиска максимального потока.

Увеличить на $+1$ вершинные числа $p(x)$ для всех неокрашенных вершин. (Заметим, что при этом обязательно увеличится $p(t)$, поскольку вершина t является неокрашенной. Действительно, если бы вершина t была окрашена, то была бы найдена увеличивающая цепь.) Затем вернуться к шагу 2.

Обоснование алгоритма поиска потока минимальной стоимости. Доказательство того, что данный алгоритм действительно находит поток минимальной стоимости, состоящий из v единиц, будет опираться на условия дополняющей нежесткости линейного программирования, рассмотренные в разд. 3.1.

Как уже отмечалось ранее, задача о потоке минимальной стоимости может быть сформулирована в виде соотношений (6)—(10) как задача линейного программирования. Пусть $p(x)$ обозначает двойственную переменную, соответствующую уравнению (7) сохранения потока в вершине x. Пусть $p(x)$ обозначает двойственную переменную, соответствующую уравнению (6) сохранения потока в источнике s. Пусть $p(t)$ обозначает двойственную переменную, соответствующую уравнению (8) сохранения потока в стоке t. (Ниже

[1]) На самом деле сначала следует убедиться, что соответствующая совокупность дуг вообще образует разрез в исходном графе, поскольку алгоритм поиска потока минимальной стоимости выполняется на некотором подграфе исходного графа. Если выделенная совокупность не является разрезом в исходном графе, то можно сразу сказать, что соответствующий поток не достигает максимально возможного значения. — *Прим. перев.*

будет показано, что двойственные переменные $p(x)$ совпадают с вершинными числами $p(x)$, вычисляемыми алгоритмом. Поэтому совпадение в обозначениях не случайно.) Пусть далее $\gamma(x, y)$ обозначает двойственную переменную, соответствующую ограничению (9), накладываемому на пропускную способность дуги (x, y). Наконец, будем рассматривать параметр v как переменную, а не как константу.

Двойственная задача линейного программирования для задачи, определяемой соотношениями (6)—(10), имеет следующий вид:

минимизировать

$$\sum_{(x, y)} c(x, y)\,\gamma(x, y) \tag{12}$$

при ограничениях

$$-p(s) + p(t) = p, \tag{13}$$

$$p(x) - p(y) + \gamma(x, y) \geqslant -a(x, y) \text{ для всех } (x, y), \tag{14}$$

$$\gamma(x, y) \geqslant 0 \text{ для всех } (x, y). \tag{15}$$

$$p(x) \text{ не ограничена (для всех } x). \tag{16}$$

Уравнение (13) является двойственным ограничением, соответствующим неограниченной переменной прямой задачи v. Каждое неравенство (14) является двойственным ограничением, соответствующим переменной прямой задачи $f(x, y)$.

Условия дополняющей нежесткости для рассматриваемой пары двойственных задач линейного программирования имеют следующий вид:

$$p(x) - p(y) + \gamma(x, y) > -a(x, y) \Rightarrow f(x, y) = 0, \tag{17}$$

$$\gamma(x, y) > 0 \Rightarrow f(x, y) = c(x, y), \tag{18}$$

Если мы положим

$$p(s) = 0,\ p(t) = p \tag{19}$$

и для всех (x, y)

$$\gamma(x, y) = \max\{0,\ p(y) - p(x) - a(x, y)\}, \tag{20}$$

то условие дополняющей нежесткости (17) принимает вид

$$p(y) - p(x) < a(x, y) \Rightarrow f(x, y) = 0 \tag{21}$$

(поскольку в данном случае из соотношения (20) следует, что $\gamma(x, y) = 0$), а условие (18) принимает вид

$$p(y) - p(x) > a(x, y) \Rightarrow f(x, y) = c(x, y), \tag{22}$$

Таким образом, опираясь на свойства условий дополняющей нежесткости линейного программирования, в рамках обсуждаемого алгоритма достаточно построить такие значения $p(x)$ для всех вершин x и такие значения $f(x, y)$ для всех дуг (x, y), при которых удовлетворялись бы условия (19), (21) и (22). При этом, конечно, значения $f(x, y)$ должны представлять допустимое решение для задачи о потоке минимальной стоимости, т. е. удовлетворять условиям (6)—(10)[1]).

Перед началом выполнения алгоритма $p(x) = 0$ для всех x и, в частности, $p = 0 = p(t)$. Следовательно, в начальный момент условия дополняющей нежесткости выполняются. Теперь мы покажем, что условия дополняющей нежесткости выполняются на всех итерациях алгоритмов поиска потока минимальной стоимости. Это будет сделано в два этапа.

(а) Будет показано, что проводимые в алгоритме изменения потока не нарушают выполнения условий дополняющей нежесткости.

(б) Будет показано, что проводимые в алгоритме изменения вершинных чисел также не нарушают выполнения условий дополняющей нежесткости.

[Заметим, что в алгоритме всегда выполняются условия (19). Кроме того, совокупность значений $\{f(x, y)\}$ всегда представляет собой допустимый поток, поскольку для нее автоматически выполняются условия (6)—(9).]

В рассматриваемом алгоритме изменение потока допускается только в тех дугах (x, y), для которых $p(y) - p(x) = a(x, y)$. Поэтому изменения потока не могут привести к нарушению условий дополняющей нежесткости, связанных с теми дугами, для которых $p(y) - p(x) \neq a(x, y)$. Таким образом, этап (а) обоснования алгоритма проведен.

Остается показать, что изменение вершинных чисел в рассматриваемом алгоритме также не приводит к нарушению условий дополняющей нежесткости. (Напомним, что в алгоритме вершинные числа увеличиваются на $+1$ только для тех вершин, которые не могли быть окрашены. Кстати, невозможность окрашивания вершины означает, что в эту вершину нельзя переслать из источника ни одной дополнительной единицы потока.) Действительно, если для дуги (x, y) обе конечные вершины являются окрашенными или неокрашенными, то величина разности $p(y) - p(x)$ остается неизменной, а потому для этих дуг условие дополняющей нежесткости не нарушается. Если для некоторой дуги (x, y) вершина x является окрашенной, а вершина y — неокрашенной, то при этом выполняется одно из следующих трех условий:

---

[1]) Следует отметить, что величины $p(x)$ $(x \in X)$ посредством соотношения (20) определяют допустимые значения для всех переменных $\gamma(x, y)$.

(а) $p(y) - p(x) < a(x, y)$,
(б) $p(y) - p(x) > a(x, y)$,
(в) $p(y) - p(x) = a(x, y)$ и $f(x, y) = c(x, y)$.

Если имеет место условие (а), то после увеличения $p(y)$ на $+1$ оказывается $p(y) - p(x) \leqslant a(x, y)$, и условие (21) не нарушается. Если имеет место условие (б), то после увеличения $p(y)$ на $+1$ разность $p(y) - p(x)$ остается большей $a(x, y)$, и условие (22) не нарушается. Если же имеет место условие (в), то после увеличения $p(y)$ на $+1$ оказывается $p(y) - p(x) > a(x, y)$, и выполняется условие (22).

Если для некоторой дуги (x, y) вершина x оказывается неокрашенной, а вершина y — окрашенной, то при этом может выполняться одно из следующих трех условий:

(а) $p(y) - p(x) < a(x, y)$,
(б) $p(y) - p(x) > a(x, y)$,
(в) $p(y) - p(x) = a(x, y)$ и $f(x, y) = 0$.

Если имеет место условие (а), то после увеличения $p(x)$ на $+1$ разность $p(y) - p(x)$ останется меньше величины $a(x, y)$, и условие (21) не нарушится. Если имеет место условие (б), то после увеличения $p(x)$ на $+1$ окажется, что $p(y) - p(x) \geqslant a(x, y)$ и условие (22) не нарушится. Наконец, если имеет место условие (в), то после увеличения $p(x)$ на $+1$ окажется, что $p(y) - p(x) < a(x, y)$, и будет выполняться условие (21).

Итак, алгоритм поиска потока минимальной стоимости формирует допустимый поток, который при $p = p(t)$ удовлетворяет всем условиям дополняющей нежесткости. Когда после пересылки через сеть последней единицы потока выполнение алгоритма заканчивается, окончательное значение $p(t)$ будет равным общей стоимости прохождения по сети последней единицы потока. Следовательно, это значение $p(t)$ является достаточно большим для того, чтобы оптимальность найденного потока, определяемая соотношением (10), гарантировала оптимальность этого потока, определяемую соотношением (5). Итак, формируемый алгоритмом поток является потоком минимальной стоимости.

Мы почти завершили обоснование алгоритма поиска потока минимальной стоимости. Нам только осталось показать, что выполнение алгоритма заканчивается за конечное число шагов. Конечность алгоритма вытекает из следующих соображений. Выполнение алгоритма поиска потока минимальной стоимости завершается после того, как последняя единица заданного потока переправляется из источника в сток. При этом величина $p(t)$ совпадает с общей стоимостью прохождения по сети последней единицы потока. Если все стоимости, соответствующие дугам, являются конечными положительными целыми числами, то и величина $p(t)$ по завершении работы алгоритма должна быть конечным положительным целым числом. Следовательно, в рамках рассматриваемого алгоритма приходится не более чем $(p(t) + 1)$ раз применять алгоритм по-

иска максимального потока. Но для последнего, как мы знаем, существует модификация, заканчивающаяся за конечное число шагов. Следовательно, алгоритм поиска потока минимальной стоимости также заканчивается за конечное число шагов.

ПРИМЕР 3. Применим описанный только что алгоритм для поиска потока минимальной стоимости на графе, изображенном на рис. 4.9. Заметим, что первое число у каждой дуги представляет собой стоимость, а второе — ее пропускную способность.

Первоначально все вершинные числа равны нулю, а все вершины являются неокрашенными, за исключением вершины s, которая будет окрашенной на протяжении всей процедуры. Результаты выполнения алгоритма сведены в приводимую ниже таблицу.

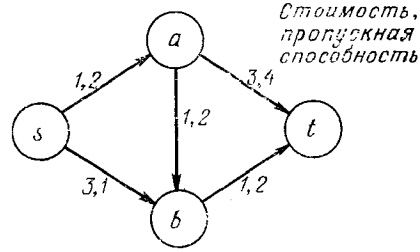


Рис. 4.9. Пример применения алгоритма поиска потока минимальной стоимости.

Итерация[a]	$p(s)$	$p(a)$	$p(b)$	$p(t)$	Окрашенные дуги	Окрашенные вершины
0	0	0	0	0	Нет	s
1	0	1	1	1	(s, a)	s, a
2	0	1	2	2	$(s, a), (a, b)$	s, a, b
3	0	1	2	3	$(s, a), (a, b), (b, t)$	s, a, b, t

[a] Итерация данного алгоритма включает применение к соответствующему графу алгоритма поиска увеличивающей цепи. — *Прим. перев.*

После 3-й итерации вершина t оказалась окрашенной. Посылаем две единицы потока из s в t по пути $(s, a), (a, b), (b, t)$. В результате $f(s, a) = 2$, $f(a, b) = 2$, $f(b, t) = 2$. Продолжаем итерации алгоритма.

4	0	1	2	3	Нет	s
5	0	2	3	4	$(s, b), (a, b)$	s, a, b
6	0	2	3	5	$(s, b), (a, b), (a, t)$	s, a, b, t

После 6-й итерации вершина t снова оказалась окрашенной. Посылаем одну единицу потока из s в t вдоль цепи $(s, b), (a, b), (a, t)$. В результате $f(s, a) = 2$, $f(s, b) = 1$, $f(a, b) = 1$, $f(a, t) = 1$, $f(b, t) = 2$. Продолжаем итерации алгоритма.

7	0	2	3	5	Нет	s

После 7-й итерации оказалось, что ни одна дополнительная единица потока не может быть передана из s в t, так как дуги, ведущие из окрашенных вершин в неокрашенные (а именно дуги (s, a) и (s, b)), являются насыщенными. Следовательно, полученный поток из трех единиц является максимальным потоком, имеющим минимальную стоимость прохождения по исходному графу.

До сих пор мы предполагали, что дуговые стоимости, приписанные дугам, являются положительными целыми числами. Теперь мы покажем, как следует модифицировать алгоритм поиска потока минимальной стоимости, чтобы он мог оперировать с нецелыми положительными числами, характеризующими дуги.

Уточним роль предположения о целочисленности дуговых стоимостей, приписанным дугам. Данное предположение приводит к тому, что единицы потока имеют стоимость прохождения по исходной сети, равную положительному целому числу. А это позволяет рассматривать только целочисленные значения p и изменять вершинные числа соответствующих вершин ровно на $+1$.

В том случае, когда значения стоимостей дуг не обязательно являются целыми, «общая стоимость» увеличивающей цепи из s в t тоже может не быть целым положительным числом. Поэтому в алгоритме необходимо рассматривать и нецелые значения p. Как при этом выбирать значения, присваиваемые параметру p, и какие приращения должны иметь вершинные числа?

Предположим, что алгоритм выполнен для некоторого значения $p = p(t)$. При этом для некоторых дуг одна из конечных вершин будет окрашена, а другая нет. Необходимо рассмотреть два случая.

Случай 1. Пусть у дуги (x, y) вершина x окрашена, а вершина y нет. Тогда дуга (x, y) может быть окрашена в следующей итерации только в том случае, если $(x, y) \in I$[1]) и допустимо такое изменение $p(y)$, после которого будет выполняться равенство $p(y) - p(x) = a(x, y)$. Но в силу условия (22) если $(x, y) \in I$, то $p(y) - p(x) \leqslant a(x, y)$. Следовательно, чтобы в следующей итерации дуга (x, y) могла быть окрашена, $p(y)$ должно возрасти на величину $[a(x, y) - p(y) + p(x)]$. Обозначим эту величину через $\delta(x, y)$, т. е. $\delta(x, y) = a(x, y) - p(y) + p(x)$.

Случай 2. Пусть у дуги (x, y) вершина y окрашена, а вершина x нет. Дуга (x, y) может быть окрашена в следующей итерации только в том случае, если $(x, y) \in R$[2]) и допустимо такое изменение $p(x)$, после которого будет выполняться $p(y) - p(x) = a(x, y)$. Но в силу

---

[1]) Здесь подразумевается множество I, используемое в алгоритме поиска максимального потока. Напомним, что в I входят только увеличивающие дуги. — *Прим. перев.*

[2]) Здесь также подразумевается множество R, которое использовано в алгоритме поиска максимального потока. В это множество входят только уменьшающие дуги. — *Прим. перев.*

условия (21) если $(x, y) \in R$, то $p(y) - p(x) \geqslant a(x, y)$. Следовательно, чтобы дуга (x, y) могла быть окрашена в следующей итерации, $p(x)$ должно возрасти на величину $[p(y) - p(x) - a(x, y)]$. Обозначим эту величину через $\delta(x, y)$, т. е. $\delta(x, y) = p(y) - p(x) - a(x, y)$.

Для каждой дуги (x, y), которая не попадает ни в одно из двух рассмотренных подмножеств дуг, будем полагать $\delta(x, y) = \infty$. Теперь определим величину

$$\delta = \min_{(x, y)} \{\delta(x, y)\} > 0. \qquad (23)$$

Итак, если двойственную переменную $p(x)$ для каждой неокрашенной вершины x увеличить на δ, то в следующей итерации будет окрашена по крайней мере одна новая дуга. При этом повышается вероятность обнаружения в исходной сети новой увеличивающей цепи. При указанном преобразовании двойственных переменных $p(t)$, очевидно, будет увеличено на δ. В последующем приращение двойственных переменных может определяться аналогичным образом, т. е. путем определения наименьшего приращения этих переменных, при котором в следующей итерации обеспечивается окрашивание хотя бы одной новой дуги. Если на каком-то этапе выполнения алгоритма получается $\delta = \infty$, то ни одна новая дуга окрашена быть не может. В этом случае текущий поток является максимальным.

Итак, введение описанной выше процедуры изменения вершинных чисел допускает возможность применения алгоритма поиска потока минимальной стоимости в случаях, когда стоимости, приписанные дугам, не являются целыми числами.

Будет ли полученный таким образом модифицированный алгоритм поиска потока минимальной стоимости давать решение за конечное число шагов? Да, будет. Действительно, поскольку $p(t)$ должно быть всегда равно сумме стоимостей дуг, составляющих некоторую цепь из s в t, и поскольку имеется лишь конечное число различных цепей из s в t, то $p(t)$ может принимать лишь конечное число различных значений. Следовательно, выполнение модифицированного алгоритма должно завершиться за конечное число шагов.

4.4. Алгоритм дефекта

Алгоритм поиска потока минимальной стоимости обладает рядом достаточно серьезных недостатков.

1. Выполнение алгоритма должно начинаться с задания нулевого потока, который путем последовательного увеличения на единицу на каждой итерации достигает максимальной величины.

2. Отсутствует простой способ корректировки результатов, к которым приводит алгоритм, в случае когда по окончании вы-

полнения алгоритма обнаруживается ошибка в задании стоимости или пропускной способности какой-либо дуги.

3. Применение алгоритма недопустимо в случаях, когда на потоки в дугах накладываются ограничения снизу или когда стоимости, приписанные дугам, отрицательны.

В данном разделе будет представлен еще один алгоритм решения задачи о потоке минимальной стоимости. Этот алгоритм, называемый *алгоритмом дефекта*, был предложен Фордом и Фалкерсоном [2]. Он лишен указанных выше недостатков алгоритма предыдущего раздела. Однако алгоритм дефекта обладает некоторыми другими специфическими недостатками, которые будут обсуждены ниже.

Пусть неотрицательное целое число $l(x, y)$ обозначает наименьшее количество единиц потока, которые должны протекать по дуге (x, y). Это число называется *нижней пропускной способностью* дуги (x, y). Введенная ранее пропускная способность $c(x, y)$ теперь будет называться *верхней пропускной способностью*.

ПРИМЕР 1. Агент бюро путешествий, который уже нам известен из ранее рассмотренных примеров, знает, что для того, чтобы заказать самолет для полета между городами x и y, необходимо закупить на соответствующий рейс не менее 25 билетов. В данной ситуации дуга (x, y) соответствующего графа должна иметь нижнюю пропускную способность $l(x, y)$, равную 25.

ПРИМЕР 2. В соответствии с местными ограничениями на валютные операции некоторая многонациональная компания хранит неконвертируемую валюту в блокированных счетах различных иностранных банков. Поскольку данная валюта может расходоваться только внутри соответствующей страны, затраты на вывоз капитала в эти страны могут в определенных пределах считаться равными нулю. Следовательно, дугам, отображающим перемещение капитала через границы стран, в которых имеются блокированные счета, могут быть приписаны нулевые стоимости.

ПРИМЕР 3. Цюрихский агент по продаже медицинского оборудования должен выработать маршрут передвижной выставки товаров, которая организуется им в различных крупных европейских городах. При этом агент рассматривает также варианты организации выставки в небольших городах, находящихся на возможных маршрутах. Стоимость переезда выставки из одного города в другой известна. Кроме того, для каждого города известен доход (или убытки) от получаемых заказов. Каков наилучший маршрут перемещения выставки, при использовании которого решаются основные рекламные задачи и попутно достигается максимальный доход?

Построим граф, в котором каждый рассматриваемый город представлен вершиной, а маршруты, связывающие различные города, — дугами. Далее разобъем каждую вершину, представляющую город, на две вершины (так, как это показано на рис. 4.10). Пусть верхние пропускные способности всех дуг равны 1. Пусть нижние пропускные способности дуг, соответствующие городам, в которых должна быть проведена выставка, равны 1, а нижние пропускные способности всех остальных дуг равны 0. Пусть, наконец, стоимости дуг, возникающих при разбиении графов на две вершины, равны ожидаемому доходу (отрицательному или положительному) от организации выставки в соответствующем городе. Пусть стоимость остальных дуг совпадает с затратами на перемещение выставки между соответствующими городами. Решение полученной задачи о потоке минимальной стоимости, в которой некоторые дуги имеют ненулевые нижние пропускные

способности и отрицательные стоимости, позволяет построить оптимальный маршрут перемещения выставки. (Отметим, что в гл. 7 будет рассмотрен более сложный вариант этой задачи об оптимальном маршруте.) Осуществить решение данной задачи можно с помощью алгоритма дефекта, к рассмотрению которого мы и переходим.

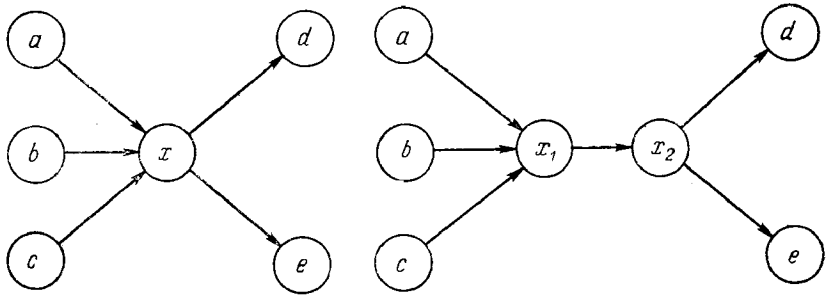


Рис. 4.10. Разбиение вершины [(x_1, x^2) — «внутригородская» дуга, остальные — «междугородные» дуги].

Алгоритм поиска потока минимальной стоимости решает соответствующую задачу в случае, когда $l(x, y) = 0$ для всех дуг(x, y). Алгоритм дефекта решает задачу о потоке минимальной стоимости в случае, когда $l(x, y) \geqslant 0$ для всех дуг (x, y). Более того, при использовании алгоритма дефекта допускаются отрицательные значения стоимостей дуг, в то время как алгоритм поиска потока минимальной стоимости оперирует только с положительными значениями. Следует, однако, отметить, что алгоритм дефекта не дает решения соответствующей задачи, когда в исходном графе присутствует контур, имеющий неограниченную пропускную способность и отрицательную общую стоимость. Например, на рис. 4.11 показан контур (a, b), (b, c), (c, a), имеющий неограниченную пропускную способность и общую стоимость, равную $1 + 1 - 3 = -1$. Следовательно, в графе, показанном на рис. 4.11, не существует потока минимальной стоимости, так как по указанному контуру можно передать неограниченное число единиц потока, прохождение каждой из которых будет иметь стоимость, равную -1.

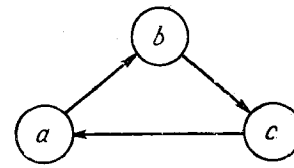


Рис. 4.11. Пример контура, имеющего отрицательную общую стоимость: $c(a,b) = c(b, c) = c(c, a) = a(a, b) = a(b, c) = 1$; $a(c, a) = -3$.

Чтобы упростить описание алгоритма дефекта, добавим к исходному графу «возвратную дугу» (t, s), ведущую из стока в источник. Пусть по этой дуге весь поток, который посылается из s в t, возвращается в источник s. Очевидно,

любой поток в исходном графе эквивалентен некоторому потоку в расширенном графе; справедливо также и обратное утверждение [1]). Если в задаче отыскивается поток минимальной стоимости из s в t, имеющий заданную общую величину v, то для возвратной дуги следует положить $l(t, s) = c(t, s) = v$ и $a(t, s) = 0$. Если же в задаче выполняется поиск потока минимальной стоимости из s в t, имеющий максимальную величину, то для возвратной дуги следует положить $l(t, s) = 0$, $c(t, s) = \infty$ и $a(t, s) = -p$, где p, как и раньше, достаточно большое число, превышающее максимальную стоимость прохождения через исходную сеть единицы потока.

Задача о потоке минимальной стоимости с ненулевыми нижними пропускными способностями при введении возвратной дуги может быть следующим образом сформулирована как задача линейного программирования:

минимизировать

$$\sum_{(x, y)} a(x, y) f(x, y) \tag{24}$$

при условии, что для всех вершин x

$$\sum_{y} f(x, y) - \sum_{y} f(y, x) = 0 \tag{25}$$

и что для всех дуг (x, y)

$$l(x, y) \leqslant f(x, y) \leqslant c(x, y). \tag{26}$$

Как и раньше, двойственную переменную, соответствующую уравнению (25) для вершины x, будем обозначать через $p(x)$. Двойственную переменную, соответствующую нижнему ограничению в (26) для дуги (x, y), обозначим через $\gamma_1(x, y)$, а двойственную переменную, соответствующую верхнему ограничению в (26) для дуги (x, y), обозначим через $\gamma_2(x, y)$. Для введенных переменных двойственная задача линейного программирования имеет следующий вид:

максимизировать

$$\sum_{(x, y)} [-c(x, y)\gamma_2(x, y) + l(x, y)\gamma_1(x, y)] \tag{27}$$

при условии, что для всех дуг (x, y) выполняются следующие соотношения:

$$-p(x) + p(y) + \gamma_1(x, y) - \gamma_2(x, y) \leqslant +a(x, y), \tag{28}$$

$$\gamma_1(x, y) \geqslant 0, \tag{29}$$

---

[1]) Поток в расширенном графе обладает тем свойством, что для него чистый поток в каждую вершину равен 0. Такой поток часто называют циркуляцией. — *Прим. перев.*

$$\gamma_2(x, y) \geqslant 0. \tag{30}$$

Разность $[\gamma_1(x,y) = \gamma_2(x, y)]$ присутствует во всех основных ограничениях (28) двойственной задачи, вследствие чего она может быть заменена новой переменной $\gamma(x, y)$, не ограниченной по знаку.

Положим, что для всех дуг (x, y) справедливо следующее соотношение:

$$\gamma(x, y) = a(x, y) + p(x) - p(y). \tag{31}$$

Заметим, что с учетом (31) соотношение (28) для любой дуги (x, y) имеет вид равенства.

Будем считать, что при $\gamma(x, y) \geqslant 0$ имеют место равенства

$$\gamma_1(x, y) = \gamma(x, y) \text{ и } \gamma_2(x, y) = 0.$$

При $\gamma(x, y) < 0$ будем считать, что

$$\gamma_1(x, y) = 0 \text{ и } \gamma_2(x, y) = -\gamma(x, y).$$

Условия дополняющей нежесткости для пары двойственных задач, соответствующих соотношениям (24) — (26) и (27) — (30), принимают следующий вид:

$$\gamma_1 > 0 \Rightarrow f(x, y) = l(x, y),$$

$$\gamma_2 > 0 \Rightarrow f(x, y) = c(x, y). \tag{32}$$

С использованием (31) эти условия для указанной пары задач линейного программирования могут быть представлены следующим образом:

$$p(y) - p(x) < a(x, y) \Rightarrow f(x, y) = l(x, y), \tag{33}$$

$$p(y) - p(x) > a(x, y) \Rightarrow f(x, y) = c(x, y). \tag{34}$$

Обратите внимание на схожесть приведенных условий с условиями дополняющей нежесткости (21) и (22).

Для большего удобства дальнейшего рассмотрения определим для всех дуг (x, y) величины

$$\alpha(x, y) = a(x, y) - p(y) + p(x). \tag{35}$$

С использованием (35) условия (33) и (34) могут быть переписаны следующим образом:

$$\alpha(x, y) > 0 \Rightarrow f(x, y) = l(x, y), \tag{36}$$

$$\alpha(x, y) < 0 \Rightarrow f(x, y) = c(x, y). \tag{37}$$

Итак, для того чтобы решить общую задачу о потоке минимальной стоимости, сформулированную выше, необходимо лишь

построить поток, удовлетворяющий уравнениям (25), и найти значения вершинных чисел, удовлетворяющие соотношениям (36) и (37) (заметим, что при этом условия (28) — (30) выполняются автоматически).

Предположим, что выбран некоторый поток, который удовлетворяет уравнениям (25). Другими словами, выбор величин $f(x, y)$ сделан так, что чистый поток через каждую вершину равен 0. Предположим также, что выбраны некоторые значения вершинных чисел $p(x)$. После этого любая дуга исходного графа может определяться одним из девяти различных состояний. Эти состояния описаны в приводимой ниже таблице.

Состояние		Величина дефекта
I $\alpha(x, y) < 0$	$f(x, y) < c(x, y)$	$\alpha(x, y)[f(x, y) - c(x, y)]$
II $\alpha(x, y) < 0$	$f(x, y) = c(x, y)$	0
III $\alpha(x, y) < 0$	$f(x, y) > c(x, y)$	$f(x, y) - c(x, y)$
IV $\alpha(x, y) = 0$	$f(x, y) < l(x, y)$	$l(x, y) - f(x, y)$
V $\alpha(x, y) = 0$	$l(x, y) \leqslant f(x, y) \leqslant c(x, y)$	0
VI $\alpha(x, y) = 0$	$f(x, y) > c(x, y)$	$f(x, y) - c(x, y)$
VII $\alpha(x, y) > 0$	$f(x, y) < l(x, y)$	$l(x, y) - f(x, y)$
VIII $\alpha(x, y) > 0$	$f(x, y) = l(x, y)$	0
IX $\alpha(x, y) > 0$	$f(x, y) > l(x, y)$	$\alpha(x, y)[f(x, y) - l(x, y)]$

(38)

Для каждого из указанных в таблице состояний приводится так называемая величина дефекта, которая характеризует степень дефектности дуги[1]). Обозначим величину дефекта через $k(x, y)$. Анализ девяти приведенных выше ситуаций показывает, что всегда $k(x, y) \geqslant 0$. Пусть k обозначает сумму величин дефектов всех дуг графа. Заметим, что величина дефекта тех дуг, для которых выполняется условие дополняющей нежесткости, равна 0. С другой стороны, если для некоторой дуги не выполняется условие дополняющей нежесткости, то величина ее дефекта строго положительна.

Основная идея, лежащая в основе алгоритма дефекта, состоит в том, чтобы последовательно уменьшать до нуля величину дефекта какой-либо дуги, не увеличивая при этом величину дефекта ни одной из оставшихся дуг. Когда это будет выполнено

[1]) Точнее, характеризует степень невыполнения для данной дуги соответствующего условия дополняющей нежесткости. — *Прим. перев.*

для каждой дуги с ненулевой величиной дефекта, все дуги исходного графа будут удовлетворять условиям дополняющей нежесткости и сформированный поток будет максимальным потоком минимальной стоимости.

Каким же образом в алгоритме дефекта осуществляется уменьшение до нуля величины дефекта различных дуг? Пусть, например, некоторая дуга (x, y) является дефектной, т. е. $k(x, y) > 0$. В соответствии с рассматриваемым алгоритмом прежде всего выявляется, что нужно делать: увеличивать или уменьшать поток в дуге (x, y), для того чтобы дуга перестала быть дефектной. Если поток в дуге (x, y) необходимо увеличивать, то выполняется поиск такой цепи из y в x, по которой поток может быть увеличен без возрастания величин дефекта каких-либо дуг исходного графа. Когда указанная цепь обнаруживается, она образует вместе с дугой (x, y) цикл. По этому циклу может быть послан дополнительный поток, в результате чего увеличивается поток по дуге (x, y), но не возрастают величины дефектов ни одной из дуг. Если указанное увеличение потока в дуге (x, y) не делает ее бездефектной, то описанный процесс повторяется. При необходимости он повторяется и далее до тех пор, пока либо дуга (x, y) не становится бездефектной, либо в процессе выполнения алгоритма не удается найти ни одной подходящей увеличивающей поток цепи. В последнем случае в соответствии с обсуждаемым алгоритмом увеличиваются некоторые вершинные числа (подобно тому, как это делается в алгоритме поиска потока минимальной стоимости), после чего снова ищутся подходящие увеличивающие поток цепи из y в x. Следует заметить, что увеличение вершинных чисел проводится так, что величина дефекта ни у одной из дуг не возрастает. В конце концов, либо дуга (x, y) становится бездефектной, либо обнаруживается, что в исходном графе не существует потока, который удовлетворял бы одновременно ограничениям, определяемым значениями верхних и нижних пропускных способностей всех дуг.

Мы рассмотрели случай, когда для устранения дефекта дуги (x, y) следовало увеличивать поток в этой дуге. Когда для той же цели поток в дуге (x, y) необходимо уменьшать, выполнение алгоритма осуществляется так же, как и в рассмотренном выше случае, за исключением того, что ищется увеличивающая поток цепь не из вершины y в вершину x, а из вершины x в вершину y. Соответствующая процедура повторяется до тех пор, пока каждая дефектная дуга не станет бездефектной.

Имея в виду приведенные соображения, мы теперь можем рассмотреть сам *алгоритм дефекта*.

Алгоритм дефекта

Шаг 1. (*Задание начальных условий*). Выбрать любой набор значений $f(x, y)$, для которого суммарный поток в каждую вершину равен 0, т. е. для которого выполняются соотношения (25). (Заметим, что выбранный в исходном графе поток не обязан удовлетворять верхним и нижним ограничениям на пропускную способность в отдельных дугах, т. е. условиям (26).) Выбрать далее любой набор значений для вершинных чисел $p(x)$.

Шаг 2. (*Определение величин дефектов.*) Для каждой дуги исходного графа (x, y) вычислить $\alpha(x, y)$ и $k(x, y)$, используя соответственно соотношения (35) и (38). Если для всех дуг (x, y) оказывается, что $k(x, y) = 0$, закончить выполнение алгоритма. В противном случае перейти к следующему шагу.

Шаг 3. (*Классификация дуг.*) Выделить на множестве всех дуг два подмножества: подмножество увеличивающих дуг I и подмножество уменьшающих дуг R. Дугу (x, y) отнести к уменьшающим дугам, если

(а) $\alpha(x, y) \geqslant 0$ и $f(x, y) > l(x, y)$

или

(б) $\alpha(x, y) \leqslant 0$ и $f(x, y) > c(x, y)$.

Дугу (x, y) отнести к увеличивающим дугам, если

(а) $\alpha(x, y) \geqslant 0$ и $f(x, y) < l(x, y)$

или

б) $\alpha(x, y) \leqslant 0$ и $f(x, y) < c(x, y)$.

Для дуг (x, y) из R положить

$r(x, y) = f(x, y) - l(x, y)$, если $\alpha(x, y) \geqslant 0$,

$r(x, y) = f(x, y) - c(x, y)$, если $\alpha(x, y) < 0$.

Для дуг (x, y) из I положить

$i(x, y) = l(x, y) - f(x, y)$, если $\alpha(x, y) > 0$,

$i(x, y) = c(x, y) - f(x, y)$, если $\alpha(x, y) \leqslant 0$.

Выбрать любую дугу (x, y), для которой $k(x, y) > 0$. Если $(x, y) \in I$, то считать вершиной s вершину y, а вершиной t — вершину x. Если $(x, y) \in R$, то считать вершиной s вершину y, а вершиной t — вершину x. (Заметим, что поскольку $k(x, y) > 0$, дуга (x, y) не может принадлежать одновременно I и R.)

Шаг 4. (*Применение алгоритма поиска максимального потока.*) Для сформированных множеств I, R и для полученных

значений $i(x, y)$ и $r(x, y)$ выполнить алгоритм поиска максимального потока из s в t. Сбалансировать полученный поток соответствующим потоком по дуге, соединяющей вершины s и t. Итерационную процедуру наращивания потока продолжать до тех пор, пока либо не будет получен поток, достаточный для перевода дуги, соединяющей вершины s и t, в разряд бездефектных, либо будет получен максимальный поток. В первом случае вернуться к шагу 2, а во втором — перейти к шагу 5.

Шаг 5. (*Увеличение вершинных чисел.*) Переход к данному шагу делается после того, как в алгоритме поиска максимального потока не удается построить увеличивающую поток цепь. Пусть C представляет собой множество вершин, окрашенных на последней итерации указанного алгоритма при поиске увеличивающей поток цепи. Пусть $\overline{C}$ представляет собой множество неокрашенных вершин для той же итерации алгоритма. Очевидно, $s \in C$ и $t \in \overline{C}$. Определить два подмножества дуг A_1 и A_2:

$$A_1 = \{(x, y) : x \in C,\ y \in \overline{C},\ \alpha(x, y) > 0,\ f(x, y) \leqslant c(x, y)\}, \tag{39}$$

$$A_2 = \{(y, x) : x \in C,\ y \in \overline{C},\ \alpha(y, x) < 0,\ f(y, x) \geqslant l(y, x)\}. \tag{40}$$

Если A_1 является пустым множеством, положить $\delta_1 = \infty$. В противном случае положить

$$\delta_1 = \min_{A_1} \{\alpha(x, y)\} > 0. \tag{41}$$

Если A_2 является пустым множеством, положить $\delta_2 = \infty$. В противном случае положить

$$\delta_2 = \min_{A_2} \{\alpha(x, y)\} > 0. \tag{42}$$

Наконец, положить

$$\delta = \min \{\delta_1, \delta_2\} > 0. \tag{43}$$

Если $\delta = \infty$, закончить выполнение алгоритма: в исходном графе не существует допустимого потока. Если $\delta < \infty$, то для $x \in \overline{C}$ заменить $p(x)$ на $p(x) + \delta$. Затем вернуться к шагу 3.

Если выполнение алгоритма заканчивается на шаге 2, то полученный к этому моменту поток $f(x, y)$ является потоком минимальной стоимости.

Обоснование алгоритма дефекта. Для обоснования данного алгоритма необходимо доказать следующие утверждения:

(1) Алгоритм формирует поток, удовлетворяющий соотношениям (25).

(2) В случае завершения выполнения алгоритма на шаге 2 полученный поток является потоком минимальной стоимости.

(3) Выполнение алгоритма завершается за конечное число шагов.

(4) В случае завершения алгоритма на шаге 5 для исходного графа не существует допустимого потока.

Докажем каждое из сформулированных утверждений.

Доказательство утверждения (1). Справедливость данного утверждения вытекает из того, что выполнение алгоритма начинается с рассмотрения потока, удовлетворяющего условиям (25), причем при всех изменениях потока в процессе работы алгоритма условия (25) не нарушаются.

Доказательство утверждения (2). Поскольку при завершении выполнения алгоритма на шаге 2 величины дефектов всех дуг равны 0, то для полученных дуговых потоков и вершинных чисел выполняются условия дополняющей нежесткости (36) и (37). В силу этого полученный поток является потоком минимальной стоимости.

Доказательство утверждения (4). Необходимо показать, что если на шаге 5 алгоритма оказывается $\delta = \infty$, то для исходного графа не существует потока, одновременно удовлетворяющего нижним и верхним ограничениям на пропускные способности во всех дугах. Для этого рассмотрим разрез, образованный дугами, у которых одна вершина принадлежит C, а другая — $\overline{C}$. Ни одна дуга, ведущая из C в $\overline{C}$, не является увеличивающей, ибо в противном случае могли бы быть окрашены соответствующие вершины в $\overline{C}$. Аналогично ни одна дуга, ведущая из $\overline{C}$ в C, не является уменьшающей, ибо в противном случае могли бы быть окрашены соответствующие вершины в $\overline{C}$. Кроме того, поскольку $\delta = \infty$, оба множества A_1 и A_2, определенные соответственно в (39) и (40), являются пустыми.

С учетом сказанного для потока в любой дуге (x, y), ведущей из C в $\overline{C}$, должно быть $f(x, y) \geqslant c(x, y)$, а для потока в любой дуге (x, y), ведущей из $\overline{C}$ в C, должно быть $f(x, y) \leqslant l(x, y)$. Отметим, что для потока в дуге, соединяющей вершины s и t, соответствующее неравенство является строгим.

Сложим теперь уравнения (25) для всех вершин, составляющих C. Полученную сумму можно интерпретировать как чистый поток из C, который должен быть равен 0. Но чистый поток из C равен общему потоку из вершин C в вершины $\overline{C}$, обозначаемому через $f(C, \overline{C})$, за вычетом общего потока из вершин $\overline{C}$ в вершины C, обозначаемого через $f(\overline{C}, C)$. Таким образом,

$$f(C, \overline{C}) - f(\overline{C}, C) = 0. \tag{44}$$

Обозначая через $l(X, Y)$ сумму нижних предельных значений пропускных способностей дуг, ведущих из X в Y, а через

$c(X, Y)$ — сумму верхних предельных значений пропускных способностей дуг, ведущих из X в Y, и учитывая приведенные выше неравенства, можем написать, что $f(\overline{C}, C) \leqslant l(\overline{C}, C)$ и $f(C, \overline{C}) \geqslant c(C, \overline{C})$. При этом по крайней мере одно из выписанных неравенств является строгим. Следовательно,

$$c(C, \overline{C}) - l(\overline{C}, C) < 0,$$

откуда

$$c(C, \overline{C}) < l(\overline{C}, C).$$

Последнее соотношение означает, что минимальный поток, который должен входить в множество вершин C, строго больше максимального потока, который может выходить из этого же множества вершин. Следовательно, в рассматриваемом случае (при $\delta = \infty$) не существует допустимого потока.

Доказательство утверждения (3). Остается показать, что выполнение алгоритма завершается за конечное число шагов.

Прежде всего отметим, что каждый раз, когда алгоритм производит изменения потока или вершинных чисел, величина дефекта ни для одной дуги не возрастает[1]). Поэтому в процессе работы алгоритма бездефектная дуга не может стать дефектной. Таким образом, в алгоритме потребовалось бы выполнение неограниченного числа шагов только в том случае, если бы нашлась дуга, для которой процедура устранения дефекта была бы бесконечной.

При устранении дефекта любой дуги в алгоритме выполняется ряд последовательных увеличений потока и вершинных чисел. Если предполагать, что пропускные способности всех дуг целочисленны, то число увеличений потока может быть лишь конечным, так как при каждом изменении поток в дефектной дуге увеличивается по крайней мере на единицу. Поэтому бесконечность процедуры устранения дефекта некоторой дуги могла бы быть связана только с неограниченным числом последовательных увеличений вершинных чисел[2]). Но увеличение вер-

---

[1]) То, что выполняемые алгоритмом изменения потока не приводят к возрастанию дефектов, достаточно легко проверить. Значительно сложнее убедиться в том, что величины дефектов не возрастают и при изменении вершинных чисел. — *Прим. перев.*

[2]) Если пропускные способности не могут быть представлены целочисленными значениями для всех дуг, то в алгоритме необходимо использовать конечную модификацию алгоритма поиска максимального потока. При этом в основном по тем же причинам, которые указывались при обосновании конечной модификации алгоритма поиска максимального потока, число увеличений потока становится конечным. Если же говорить точнее, то данное обстоятельство определяется следующим образом. Между двумя увеличениями потока, при которых дуга (x, y) является узким местом, мини-

шинных чисел приводит к тому, что либо окрашивается по крайней мере одна новая вершина, либо из множества $A_1 \cup A_2$ удаляется по крайней мере одна дуга. Поскольку исходный граф состоит из конечного числа вершин, а множество $A_1 \cup A_2$ включает конечное число дуг, то после конечного числа шагов либо должна быть окрашена вершина t, либо должно стать $\delta = \infty$. Таким образом, в алгоритме не может неограниченное число раз происходить увеличение вершинных чисел. Следовательно, весь алгоритм завершает свою работу за конечное число шагов.

На этом завершается обоснование алгоритма дефекта.

В начале данного раздела мы уже упоминали о том, что алгоритм дефекта лишен ряда недостаков, присущих алгоритму поиска потока минимальной стоимости, но имеет некоторые специфические недостатки. Какой же из них самый существенный? Основным недостатком алгоритма дефекта является то, что он должен начинать свою работу после задания вершинных чисел, а во многих случаях нет никаких соображений по поводу того, как это делать. В результате приходится задавать начальные значения для вершинных чисел произвольным образом, что может приводить к большим величинам дефекта у некоторых дуг и, следовательно, к большому числу увеличений потока, необходимых для устранения дефекта.

Однако, с другой стороны, применение алгоритма дефекта допускает отрицательность дуговых стоимостей и наличие нижних пропускных способностей дуг. Кроме того, если после применения алгоритма произошли бы изменения в стоимости или пропускных способностях некоторых дуг, то полученное ранее оптимальное решение могло бы быть использовано как начальное в случае применения алгоритма дефекта для решения задачи с измененными исходными данными.

4.5. Алгоритм поиска динамического потока

В предыдущих разделах данной главы мы рассматривали потоки, удовлетворяющие некоторым условиям. Последние определялись заданными на дугах пропускными способностями и стоимостями. В данном разделе мы будем рассматривать сети, дуги которых характеризуются еще одним показателем — временем прохождения потока. При этом мы будем рассматривать такие потоки на этих сетях, в которых каждая единица проходит из источника в сток за время, не превышающее заданное.

мальное количество дуг в увеличивающих цепях из s в t должно возрасти по крайней мере на 2 в подграфе, образованном дугами множества $I \cup R$, и должно остаться неизменным вне указанного подграфа. Поскольку в исходном графе можно выделить лишь конечное число подграфов, число увеличений потока может быть лишь конечным. — *Прим. перев.*

Припишем каждой дуге (x, y) графа $G = (X, A)$ целое положительное число $a(x, y)$, которое определяет количество некоторых временных интервалов, необходимых для прохождения единицы потока по дуге (x, y). Величина $a(x, y)$ называется *временем прохождения* по дуге (x, y). (Заметим, что ранее через $a(x, y)$ обозначалась стоимость дуги (x, y). Далее мы увидим, что как стоимость, так и время прохождения дуги играют в соответствующих алгоритмах одну и ту же роль. Поэтому для обозначения обоих параметров и используется один и тот же символ. Данное обстоятельство станет в последующем более ясным.) Будем через $c(x, y, T)$ обозначать максимальное число единиц потока, которое может входить в дугу (x, y) в момент времени T, где $T = 0, 1, \ldots$[1]). *Динамическим потоком* в графе G из вершины s в вершину t называется любой поток из s в t, который удовлетворяет ограничениям на пропускные способности дуг в каждый рассматриваемый момент времени. Точнее, динамическим потоком из s в t называется любой поток между указанными вершинами, для которого в каждую дугу (x, y) в любой рассматриваемый момент времени T входит не более чем $c(x, y, T)$ единиц потока. Отметим, что в динамическом потоке отдельные его единицы могут отправляться из источника в момент времени 0, 1, 2, ...

Максимальным динамическим потоком из вершины s в вершину t за период в p интервалов времени является такой динамический поток из s в t, для которого в сток t за период времени p проходит максимально возможное количество единиц потока.

ПРИМЕР 1. Пусть хорошо нам знакомый агент бюро путешествий должен переправить в течение 48 часов 75 пассажиров из Спрингфилда в Стамбул. Данная проблема может быть следующим образом сведена к задаче о максимальном динамическом потоке. Пусть в соответствующем графе Спрингфилду соответствует источник, а Стамбулу — сток. Пусть каждый аэропорт, принадлежащий возможному маршруту перелета из Спрингфилда в Стамбул, также представлен некоторой вершиной. В рассматриваемом графе соединим вершины x и y дугой только в том случае, если имеется беспосадочный рейс между соответствующими аэропортами. Пусть время прохождения каждой дуги (x, y) равно времени полета между соответствующими аэропортами, округленному до часов. (В длительность полета должно быть включено время пересадки в соответствующем аэропорту с одного рейса на другой.) Пусть пропускная способность $c(x, y, T)$ дуги (x, y) в момент времени T равна числу мест на соответствующий рейс с временем отправления T. Если указанного рейса нет, то полагается $c(x, y, T) = 0$.

Рассматриваемая практическая задача имеет решение, если в построенном выше графе существует динамический поток в 75 единиц из источника в сток за период в 48 интервалов времени и если мы можем этот поток построить.

---

[1]) В данном разделе время повсюду дискретно, так что T обозначает номер соответствующего временно́го интервала. Для краткости мы будем говорить о T просто как о моменте времени. — *Прим. перев.*

Очевидно, задача поиска максимального динамического потока является более сложной, чем задача поиска максимального потока. Это связано с тем, что при рассмотрении задачи о динамическом потоке необходимо прослеживать перемещение каждой единицы потока с тем, чтобы ни в один момент времени ни для одной дуги на входе не была превышена ее пропускная способность. К счастью, такое дополнительное усложнение задачи о динамическом потоке (по сравнению с задачей о статическом потоке) можно обойти путем сведения первой задачи ко второй в «развернутом во времени» варианте исходного графа.

Развернутый во времени вариант исходного графа $G = (X, A)$ *для периода в* p *интервалов времени* обозначается через G_p. Множество вершин графа G_p определяется как

$$X_p = \{x_i : x \in X,\ i = 0, 1, \ldots, p\}. \tag{45}$$

Множество дуг графа G_p определяется как

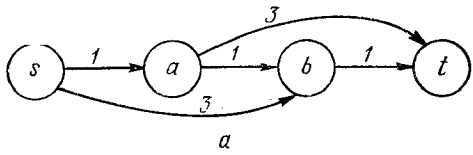
$$A_p = \{(x_i, y_j) : (x, y) \in A,\ i = 0, 1, \ldots, p - a(x, y),\ j = i + a(x, y)\}. \tag{46}$$

Положим

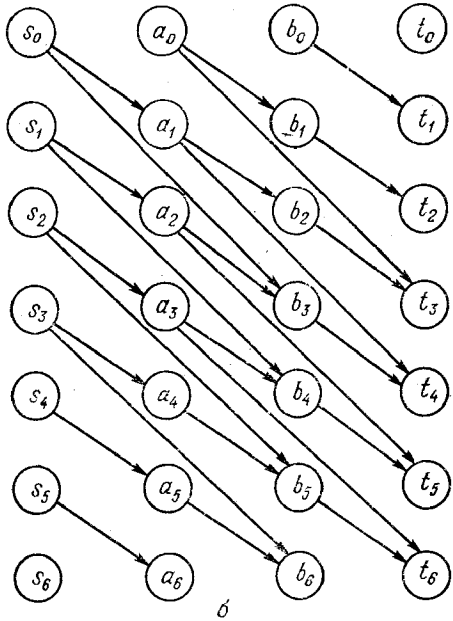
$$c(x_i, y_j) = c(x, y, i).$$

Заметим, что множество вершин X_p графа G_p формируется из вершин множества X, каждая из которых продублирована p раз для каждого момента времени в рассматриваемом периоде. В графе G_p вершины x_i и y_j соединяются дугой (x_i, y_j), если в исходном графе G поток может пройти из вершины x в вершину y за время $(j - i)$ Например, единица потока, выходящая из вершины x в момент времени 5 и затрачивающая при прохождении по дуге (x, y) 8 единичных интервалов времени, может быть представлена в графе G_p единицей потока, проходящей по дуге (x_5, y_{13}). На рис. 4.12 показаны некоторый граф и его развернутый во времени вариант для периода в 6 интервалов времени.

Очевидно, любой динамический поток из s в t в графе G эквивалентен потоку из группы источников в группу стоков в графе G_p. Справедливо и обратное утверждение. В приводимой ниже таблице даны динамический поток в графе, изображенном на рис. 4.12, и эквивалентный ему статический поток в растянутом во времени варианте этого графа ($p = 6$), который также изображен на рис. 4.12.



а



б

Динамический поток			Статический поток	
Путь	Время отправления	Количество	Путь	Количество
s, a, b, t	0	1 единица	s_0, a_1, b_2, t_3	1 единица
s, a, b, t	1	1 единица	s_1, a_2, b_3, t_4	1 единица
s, a, t	2	1 единица	s_2, a_3, t_6	1 единица

Рис. 4.12. *а* — исходный граф (пропускные способности всех дуг равны 1; время прохождения проставлено рядом с соответствующими дугами); *б* — растянутый во времени вариант исходного графа ($p = 6$; пропускные способности всех дуг равны 1).

Поскольку каждый динамический поток эквивалентен статическому потоку в развернутом во времени варианте исходного графа, то максимальный динамический поток за период в p интервалов времени может быть определен с помощью соответствующего алгоритма поиска максимального (статического) потока в развернутом во времени варианте исходного графа (развернутом на период времени p). Таким образом, в принципе нет необходимости в разработке нового алгоритма для решения задачи о динамическом потоке. Однако если p достаточно велико, то достаточно большим становится и граф G_p. Соответственно существенно возрастает объем вычислений, необходимых для поиска максимального потока в графе G_p.

К счастью, Форд и Фалкерсон [2] разработали алгоритм, названный ими *алгоритмом поиска максимального динамического потока*, который строит соответствующий поток значительно более эффективно, чем алгоритм поиска максимального потока после сведения задачи о динамическом потоке к обычной задаче о потоке. Однако необходимо иметь в виду, что алгоритм поиска максимального динамического потока может быть использован только для не зависящих от времени входных пропускных способностей, т. е. выполняться при условии, что $c(x, y, T) = c(x, y)$ для всех $T = 0, 1, \ldots, p$ и всех дуг $(x, y) \in A$.

Алгоритм поиска максимального динамического потока в качестве подалгоритма использует алгоритм поиска потока минимальной стоимости. Напомним, что в алгоритме поиска потока минимальной стоимости сначала из s в t посылается максимально возможное количество единиц потока, имеющих общую стоимость прохождения по сети, равную 1. Затем в соответствии с этим алгоритмом из s в t посылается максимально возможное количество единиц потока, имеющих общую стоимость прохождения по сети, равную 2. Далее процедура продолжается аналогично, до тех пор пока не будет получен максимальный поток. Пусть $F_1, F_2, \ldots, F_p$ обозначает результирующую последовательность потоков, формируемую алгоритмом поиска потока минимальной стоимости. (Детально способ формирования любого потока F_i из потока F_{i-1} был рассмотрен в разд. 4.3.) Каждый поток F_i в указанной последовательности может быть представлен определенным набором путей $f_{i,1}, f_{i,2}, \ldots, f_{i,r_i}$, ведущих из s в t, по которым пересылаются определенные количества единиц потока, составляющие общий поток F_i[1]). Обозначим через $n_{i,j}$ количество единиц потока, которое должно было бы проходить по пути $f_{i,j}$ в потоке F_i. Очевидно, ни один из путей $f_{i,j}$, ведущих из s в t, не имеет суммарную стоимость, большую чем i, иначе

[1]) Разложение потока F_i по путям $f_{i,1}, f_{i,2}, \ldots, f_{i,ri}$ связано с решением хотя и простой, но нетривиальной задачи. — *Прим. перев.*

этот путь не мог бы возникнуть в результате выполения алгоритма поиска потока минимальной стоимости. Обозначим через $a(f_{i,j})$ общую стоимость пути $f_{i,j}$.

Опираясь на введенные обозначения, мы может теперь формально описать *алгоритм поиска максимального динамического потока.*

Алгоритм поиска максимального динамического потока

Алгоритм формирует максимальный динамический поток за p единичных интервалов времени. Этот поток протекает от вершины s к вершине t в графе $G = (X, A)$, в котором $c(x, y, T) = c(x, y)$ для всех моментов времени $T = 0, 1, 2, \ldots, p$ и всех дуг $(x, y) \in A$.

Шаг 1. Считая стоимость дуги (x, y) равной времени $a(x, y)$ прохождения по этой дуге, применить к исходному графу алгоритм поиска потока минимальной стоимости. Выполнение алгоритма продолжать до формирования потока F_p.

Поток F_p определяется путями $f_{p,1}, f_{p,2}, \ldots, f_{p,r_p}$ из s в t, по которым протекает соответственно $n_{p,1}, n_{p,2}, \ldots, n_{p,r_p}$ единиц потока. Данная декомпозиция потока F_p является побочным результатом выполнения алгоритма поиска потока минимальной стоимости.

Шаг 2. Для $j = 1, 2, \ldots, r_p$ отправлять по каждому пути $f_{p,j}$ $n_{p,j}$ единиц потока в каждый из моментов времени $0, 1, \ldots, p - a(f_{p,j})$.

Полученный в результате выполнения шага 2 поток является максимальным динамическим потоком за p единичных интервалов времени.

Таким образом, алгоритм поиска максимального динамического потока сводится к алгоритму поиска потока минимальной стоимости, в котором длительности прохождения дуг выступают в роли дуговых стоимостей. Последний из полученных в этом алгоритме потоков подвергается декомпозиции на потоки вдоль определенных путей от s к t. После указанной декомпозиции вдоль каждого из полученных путей из s отправляется соответствующее количество единиц потока в моменты времени 0, 1 и т. д. вплоть до момента времени, для которого отправленный поток еще успевает достичь стока к моменту времени p.

В качестве примера рассмотрим максимальные динамические потоки в графе, изображенном на рис. 4.12, для $p = 4, 5, 6$. Результаты выполения алгоритма поиска потока минимальной стоимости в данном конкретном случае выглядят так:

p	F_p
0	0
1	0

2 0
3 1 единица вдоль пути s, a, b, t
4 1 единица вдоль пути s, a, b, t
5 1 единица вдоль пути s, b, t
1 единица вдоль пути s, a, b
6 1 единица вдоль пути s, b, t
1 единица вдоль пути s, a, t

На основе полученных результатов для каждого рассматриваемого значения p максимальный динамический поток может быть построен следующим образом:

$p = 0$ Поток не существует

$p = 1$ Поток не существует

$p = 2$ Поток не существует

$p = 3$ Отправить 1 единицу потока по пути s, a, b, t в момент времени $T = 0$. Данная единица потока попадает в сток в момент времени $T = 3$. Общий поток составляет 1 единицу

$p = 4$ Отправить 1 единицу потока по пути s, a, b, t в момент времени $T = 0$; данная единица потока попадает в сток в момент времени $T = 3$. Отправить 1 единицу потока по пути s, a, b, t в момент времени $T = 1$; данная единица потока попадает в сток в момент времени $T = 4$. Общий поток составляет 2 единицы

$p = 5$ Отправить 1 единицу потока по пути s, b, t в момент времени $T = 0$; данная единица потока попадает в сток в момент времени $T = 4$. Отправить 1 единицу потока по пути s, b, t в момент времени $T = 1$; данная единица потока попадает в сток в момент времени $T = 5$. Отправить 1 единицу потока по пути s, a, t в момент времени $T = 0$; данная единица потока попадает в сток в момент времени $T = 4$.
Отправить 1 единицу потока по пути s, a, t в момент времени $T = 1$; данная единица потока попадает в сток в момент времени $T = 5$. Общий поток составляет 4 единицы

$p = 6$ Отправить по одной единице потока по пути s, b, t в моменты времени $T = 0$, 1, 2; указанные единицы потока попадают в сток соответственно в моменты времени $T = 4$, 5, 6. Отправить по одной единице потока по пути s, a, t в моменты времени $T = 0$, 1, 2; указанные единицы потока попадают в сток в моменты времени $T = 4$, 5, 6. Общий поток составляет 6 единиц

Возвращаясь к общему описанию алгоритма поиска максимального динамического потока, заметим, что формируемый им поток является *повторенным во времени*. Действительно, формируе-

мый поток состоит из повторяющихся перемещений одного и того же количества единиц потока по определенным путям, ведущим из s в t. Конечно, динамический поток, генерируемый описанным выше способом, является не единственным максимальным динамическим потоком за период в p интервалов времени. Однако, как видно из структуры алгоритма, всегда существует максимальный динамический поток, являющийся повторенным во времени потоком.

Обоснование алгоритма поиска максимального динамического потока. Для обоснования данного алгоритма необходимо доказать следующие утверждения:

(а) Алгоритм формирует некоторый поток.

(б) Формируемый поток является максимальным динамическим потоком за период в p интервалов времени.

(в) Алгоритм завершается за конечное число шагов.

Доказательство утверждения (а). Структура алгоритма такова, что весь поток F_p распределен по путям, ведущим из вершины s в вершину t. При этом поток, входящий в любую вершину, за исключением источника и стока, равен потоку, выходящему из этой же самой вершины. Кроме того, отдельные единицы потока отправляются из источника только в том случае, если они могут за время p достичь стока.

Теперь важно выяснить, какое количество единиц потока, формируемого алгоритмом, входит в дугу (x, y) в каждый рассматриваемый момент времени. Для потока F_p количество единиц, протекающих по дуге (x, y), не превышает величину пропускной способности $c(x, y)$. Поэтому совокупность потоков вдоль путей $s_{p,1}, f_{p,2}, \ldots, f_{p,r_p}$ также не может чередовать через дугу (x, y) поток, превышающий ее пропускную способность. Но тогда и динамический поток, порождаемый алгоритмом, не может ни в один из моментов времени превышать пропускную способность какой-либо из дуг.

Итак, поскольку 1) все единицы потока, отправленные из источника в определенные моменты времени, достигают стока к моменту времени $T = p$, 2) при их перемещении по исходной сети ни в одной дуге не превышается ее пропускная способность и 3) ни в одной из вершин исходной сети не происходит накопления потока, рассматриваемый алгоритм строит динамический поток для периода в p интервалов времени.

Доказательство утверждения (б). Чтобы показать, что формируемый алгоритмом поток является максимальным динамическим потоком, рассмотрим эквивалентный ему стационарный поток в растянутом по времени варианте G_p исходного графа. При этом докажем, что соответствующий стационарный поток насыщает дуги некоторого разреза, отделяющего в графе G_p источники от стоков.

Пусть $p(x)$ представляет собой значение двойственной переменной для вершины x исходного графа перед началом выполнения $(p+1)$-й итерации алгоритма поиска потока минимальной стоимости. Тогда $p(t) = p + 1$. Пусть далее

$$C = \{x_i : x_i \in X_p,\ p(x) \leqslant i\}.$$

Заметим, что, поскольку $p(s) = 0$, все вершины-источники s_0, s_1, ..., s_p являются элементами C. Заметим также, что поскольку $p(t) = p + 1$, то ни одна из вершин-стоков t_0, t_1, ..., t_p элементом C не является. Множество всех дуг, одна концевая вершина которых принадлежит C, а другая не принадлежит C, образует разрез K, который отделяет в графе G_p источники от стоков.

Пусть дуга (x_i, y_j) принадлежит разрезу K, т. е. $x_i \in C$ и $y_j \notin C$. Тогда в силу определения K имеем $a(x, y) = j - i < p(y) - p(x)$. Отсюда с учетом вида условий дополняющей нежесткости для задачи о потоке минимальной стоимости вытекает, что $f(x, y) = c(x, y)$. Следовательно, потоки вдоль путей, участвующих в создании потока F_p, должны насыщать дугу (x, y). Но по каждому из указанных путей общее время прохождения потока (или соответственно общая стоимость прохождения) не превышает величины p. Более того, каждый из этих путей таков, что проходящие по нему единицы потока могут достичь вершины x_i в момент времени $T = i$ и попасть после этого в сток к моменту времени $T = p$. Таким образом, дуга (x_i, y_j) в графе G_p является насыщенной. Аналогичным образом можно показать, что при $x_i \notin C$ и $y_j \in C$ дуга (x_i, y_j) не содержит потока.

Итак, каждая дуга разреза K в графе G_p, ведущая из «области источника» в «область стока», является насыщенной, а каждая дуга того же разреза, ведущая из области стока в область источника, является пустой (не содержит потока). Следовательно, разрез K является насыщенным и формируемому алгоритмом потоку соответствует максимальный поток в графе G_p. Таким образом, формируемый в рассматриваемом алгоритме поток является максимальным динамическим потоком за период в p интервалов времени.

Доказательство утверждения (*в*). Выполнение алгоритма поиска потока минимальной стоимости завершается за конечное число шагов. Следовательно, и алгоритм поиска максимального динамического потока также должен завершаться за конечное число шагов, так как он включает алгоритм поиска потока минимальной стоимости и построение повторяющегося потока вдоль конечного числа путей, ведущих из вершины s в вершину t.

На этом заканчивается обоснование алгоритма поиска максимального динамического потока.

Заметим, что выше — при обсуждении динамических потоков — не рассматривалась возможность остановки или задержки в какой-либо вершине единицы потока в течение некоторого периода вре-

мени, прежде чем эта единица потока продолжит свое движение к стоку. Однако такая возможность является вполне реальной. В частности, в рассмотренном выше примере пассажиры, летящие из Спрингфилда в Стамбул, могли бы изъявить желание прервать свой перелет на несколько дней для остановки в одном из промежуточных городов.

Для случая когда допускается задержка потока, граф G следует скорректировать, добавив к нему дуги вида (x_i, x_{i+1}). При этом единицы потока, достигнув вершины x, могут быть отправлены из нее спустя некоторое время.

Интересно поставить следующий вопрос: изменится ли величина максимального динамического потока за период в p интервалов времени при условии допустимости задержек потока. Очевидно, возможность задержки потока не может привести к уменьшению этой величины. На самом деле, легко также показать, что величина максимального динамического потока за период в p единичных интервалов времени при наличии задержек не может и возрасти. Действительно, пусть задержки разрешены и граф G_p дополнен дугами указанного выше типа. Рассмотрим поток, формируемый алгоритмом поиска максимального динамического потока, и разрез K, насыщаемый этим потоком в графе G_p, еще не дополненном дугами (x_i, x_{i+1}). Разрез K является разрезом и в дополненном графе G_p, оставаясь в нем для соответствующего потока насыщенным, так как дополнительные дуги, попавшие в разрез, имеют направление от $\overline{C}$ к C и не содержат потока. Таким образом, возможность задержек потока не приводит к возрастанию величины максимального динамического потока.

Лексикографические динамические потоки

Пусть $V(p)$ обозначает максимальное количество единиц потока, которое может быть передано из источника в сток за p интервалов времени. Очевидно,

$$V(1) \leqslant V(2) \leqslant \ldots \leqslant V(p) \leqslant V(p+1).$$

В рассмотренном выше случае, когда времена прохождения дуг стационарны (т. е. остаются неизменными для всех моментов времени), максимальный динамический поток за период в p интервалов времени может быть получен в алгоритме поиска максимального динамического потока путем повторения во времени потока F_p, получаемого по завершении p-й итерации алгоритма поиска минимальной стоимости. Поскольку поток вдоль каждого пути $f_{p,i}$ повторяется $[p - a(f_{p,i})]$ раз, а путь $f_{p,i}$ пропускает $n_{p,i}$ единиц потока, то

$$V(p) = \sum_{i=1}^{i=r_p} [p - a(f_{p,i})]\, n_{p,i}.$$

Обозначим через V_p количество единиц потока, пересылаемых из s в b по окончании p-й итерации алгоритма поиска потока минимальной стоимости. Тогда

$$V(p) = pV_p - \sum_{i=1}^{i=r_p} a(f_{p,i})\, n_{p,i} = pV_p - \sum_{(x,\,y)} a(x,\,y)\, f_p(x,\,y). \qquad (48)$$

Отсюда следует, что

$$v(p+1) - v(p) = (p+1)V_{p+1} - pV_p - \sum_{(x,\,y)} a(x,\,y)\,[f_{p+1}(x,\,y) -$$

$$- f_p(x,\,y)] = V_{p+1} \geqslant V_p. \qquad (49)$$

Заметим, что $V(p+1) - V(p) = V(p)$ тогда и только тогда, когда поток F_{p+1} идентичен потоку F_p. Таким образом, с ростом p *максимальный динамический поток возрастает по крайней мере на* V_p *единиц.*

Потоком наискорейшего прибытия за период в p *интервалов времени* называется такой максимальный динамический поток за тот же период времени, который обеспечивает в каждый момент времени $i = 0, 1, 2, \ldots, p$ передачу в сток $V(i)$ единиц потока. Таким образом, поток наискорейшего прибытия за период в p интервалов времени (если он существует) является максимальным динамическим потоком для каждого периода, состоящего из $0, 1, 2, \ldots, p$ интервалов времени. Данное потоку название логически вполне оправданно, поскольку ни в одном другом потоке отдельные единицы потока не могли бы попасть в сток за более короткое время, чем в потоке наискорейшего прибытия.

ПРИМЕР 2. Агент бюро путешествий из предыдущего примера ставит перед собой задачу обеспечить доставку наибольшего количества туристов из Спрингфилда в Стамбул в течение первого часа, в течение первых двух часов и т. д. вплоть до первых 48 часов. Используя введенную выше терминологию, данную задачу можно сформулировать как задачу поиска потока наискорейшего прибытия за период в 48 интервалов времени.

Рассмотрим важную теорему.

ТЕОРЕМА 4.1. В графе G всегда существует поток наискорейшего прибытия за период в p интервалов времени.

Доказательство. Применим к графу G_p для $p = 0$ алгоритм поиска максимального потока из источника в сток. Полученный поток будет представлять собой максимальный динамический поток в исходном графе за период времени 0. Обозначим его через МДП_0. Далее, исходя из потока МДП_0, с помощью алгоритма по-

иска максимального потока построим в графе G_p для $p = 1$ поток $МДП_1$ из источников в стоки t_0 и t_1. Заметим, что в процессе формирования $МДП_1$ из $МДП_0$ единица потока, входящая в сток t_0, ни при каких обстоятельствах не будет передана в сток t_1 по другому маршруту, поскольку в соответствии с алгоритмом поиска максимального потока единица потока, уже попавшая в сток, никогда из него не изымается. Однако следует иметь в виду, что маршрут движения какой-либо единицы потока, попадающей в сток t_0, при формировании $МДП_1$ может измениться. Таким образом, $МДП_1$ будет максимальным динамическим потоком как за период в ноль интервалов времени, так и за период в один интервал времени.

Аналогичным образом, исходя из $МДП_1$, строится $МДП_2$, при этом $МДП_2$ будет максимальным динамическим потоком за период в ноль, один и два интервала времени.

Описанную процедуру можно продолжить до получения $МДП_p$, который будет искомым потоком наискорейшего прибытия. Теорема доказана.

Проведенное доказательство не только убеждает нас в том, что поток наискорейшего прибытия существует для любого графа G и любого $p = 0, 1, \ldots$, но и показывает, каким образом может быть построен данный поток. Однако соответствующая процедура для графов G большего размера или для бóльших величин p включает значительное и во многих случаях неприемлемое количество операций. К счастью, существует более эффективный алгоритм построения потока наискорейшего прибытия, правда для специального случая, когда пропускные способности дуг стационарны, т. е. $c(x, y, T) = c(x, y)$ для всех моментов времени T и всех дуг (x, y). Этот алгоритм так и называется алгоритмом поиска потока наискорейшего прибытия [7, 11].

Прежде чем описывать алгоритм поиска потока наискорейшего прибытия, выявим условия, которые позволяют судить о том, является или не является поток, построенный алгоритмом поиска максимального динамического потока, потоком наискорейшего прибытия. Сделаем это на примере графа, изображенного на рис. 4.12. Исследуем полученный в том графе максимальный динамический поток для $p = 6$. Этот поток включает 6 единиц: по одной единице потока отправляется из источника в моменты времени $T = 0, 1, 2$ по путям (s, a, t) и (s, b, t). В результате в сток прибывает по две единицы потока в моменты времени $T = 4, 5, 6$. Этот поток не может быть потоком наискорейшего прибытия, поскольку ни одна единица потока не пребывает в сток в момент времени $T = 3$[1]). Заметим, что на 5-й итерации ($p = 5$) выполняе-

[1]) Как будет видно несколько дальше (см. пример 3), в рассматриваемом графе существует динамический поток, который «достигает» стока в момент времени $T = 3$. — *Прим. перев.*

мого алгоритма поиска потока минимальной стоимости выявляется увеличивающая поток цепь (s, b, a, t). Эта цепь имеет общую стоимость (общее время прохождения), равную $3 - 1 + 3 = 5$. Она преобразует потоковый путь (s, a, b, t), имеющий общую стоимость $1+1+1=3$, в потоковый путь (s, a, t), имеющий общую стоимость $1 + 3 = 4$. Таким образом, единицы потока, протекавшие ранее по пути (s, a, b, t), будут переброшены на путь (s, a, t), имеющий по сравнению с первым на единицу большую стоимость. В этом и кроется причина того, что построенный максимальный динамический поток не является потоком наискорейшего прибытия.

Ключевая идея, лежащая в основе алгоритма поиска потока наискорейшего прибытия, заключается в том, чтобы пропускать максимальное количество единиц потока по коротким путям типа (s, a, b, t) прежде, чем переходить с той же целью к более длинным путям типа (s, a, t).

Чтобы упростить описание алгоритма поиска потока наискорейшего прибытия, будем предполагать, что все величины $c(x, y)$ являются целочисленными. Кроме того, заменим каждую дугу (x, y) на $c(x, y)$ дуг-дубликатов[1]. Любая введенная дуга имеет пропускную способность, равную 1. Поскольку в преобразованном графе пропускные способности всех дуг равны 1, то каждый путь из s в t будет пропускать по одной единице потока (в любой момент времени), а любая дуга будет либо насыщенной, либо пустой (т. е. не будет содержать потока).

Как и раньше, обозначим через F_i поток, сформированный по окончании i-й итерации алгоритма поиска потока минимальной стоимости. Пусть $f_{i,1}, f_{i,2}, \ldots, f_{i,r_i}$ по-прежнему обозначают пути, потоки вдоль которых составляют общий поток F_i. В силу сделанных предположений каждый из этих путей может пропускать по одной единице потока. (Таким образом, $n_{i,j} = 1$ для всех возможных i и j.) Обозначим через $P_{i,j}$-ю увеличивающую поток цепь из s в t, выявленную на i-й итерации алгоритма поиска потока минимальной стоимости. В рассмотренном выше примере для графа, изображенного на рис. 4.12, были определены $P_{3,1} = (s, a, b, t)$ и $P_{5,1} = (s, b, a, t)$.

С учетом высказанных соображений и данных определений мы теперь можем формально описать алгоритм поиска потока наискорейшего прибытия.

Алгоритм поиска потока наискорейшего прибытия

Шаг 1. (*Задание начальных условий.*) Выполнить p первых итераций алгоритма поиска потока минимальной стоимости, положив

[1]) Точнее, дуга (x, y) заменяется параллельно соединенными дугами, ведущими из x в y, причем число этих дуг совпадает с $c(x, y)$. — *Прим. перев.*

перед первой итерацией потоки во всех дугах равными нулю. При выполнении указанных итераций фиксировать каждый из потоков F_i вместе с соответствующим множеством путей $f_{i,1}, f_{i,2}, \ldots, f_{i,r_i}$, а также все выявляемые алгоритмом «пути прорыва» $P_{i,j}$ для $i = 0, 1, \ldots, p$ и $j = 1, 2, \ldots, w_i$.

Шаг 2. (*Построение потока.*) Провести следующую процедуру для каждого $\delta = 0, 1, \ldots, p$.

Выделить последовательность цепей

$$P_{0,1}, P_{0,2}, \ldots, P_{0,w_0}, P_{1,1}, P_{1,2}, \ldots, P_{1,w_1}, P_{\delta,1} P_{\delta,2}, \ldots, P_{\delta,w_\delta}.$$

В момент времени $T = \delta - i$ отправить одну единицу потока вдоль каждой из этих цепей $P_{i\,j}$, ведущих из s в t. Считая, что эта единица потока проходит дугу (x, y) в прямом направлении за $a(x, y)$ интервалов времени, а в обратном направлении — за $[-a(x, y)]$ интервалов времени, пометить каждую прямую дугу цепи $P_{i,j}$ моментом времени, в который единица потока входит в эту дугу, и снять ту метку у каждой обратной дуги, которая совпадает с моментом «выхода» единицы потока из данной дуги[1]).

По окончании описанной процедуры метки у дуг представляют собой моменты времени входа единиц потока в данную дугу и соответствуют потоку наискорейшего прибытия за период в p интервалов времени.

ПРИМЕР 3. Применим рассмотренный алгоритм для поиска потока наискорейшего прибытия при $p = 6$ в графе, изображенном на рис. 4.12. Напомним, что применение к этому графу алгоритма поиска потока минимальной стоимости приводит к выявлению всего лишь двух увеличивающих поток цепей, а именно цепи $P_{3,1} = (s, a, b, t)$ и цепи $P_{5,1} = (s, b, a, t)$. Результаты выполнения процедуры нанесения меток таковы:

для $\delta = 0$ меток нет,

для $\delta = 1$ меток нет,

для $\delta = 2$ меток нет,

для $\delta = 3$ вдоль цепи $P_{3,1}$ отправляется единица потока в момент времени $T = \delta - 3 = 0$.

Это дает следующие метки (рис. 4.13): 0 на дуге (s, a), 1 на дуге (a, b) и 2 на дуге (b, t).

Для $\delta = 4$ вдоль цепи $P_{3,1}$ отправляется единица потока в момент времени $T = \delta - 3 = 1$. Это дает следующие новые метки: 1 на дуге (s, a), 2 на дуге (a, b) и 3 на дуге (b, t).

Для $\delta = 5$ вдоль цепи $P_{3,1}$ отправляется единица потока в момент времени $T = \delta - 3 = 2$. Это дает следующие новые метки: 2 на дуге (s, a), 3 на дуге (a, b), 4 на дуге (b, t). Далее вдоль цепи $P_{5,1}$ отправляется единица потока в момент времени $T = \delta - 5 = 0$. Это дает следующие новые метки: 0 на дуге (s, b) и 2 на дуге (a, t). Кроме того, при просмотре цепи $P_{5,1}$ с дуги (a, b) снимается метка 2.

Для $\delta = 6$ вдоль цепи $P_{3,1}$ отправляется единица потока в момент времени $T = \delta - 3 = 3$. Это дает следующие новые метки: 3 на дуге (s, a), 4

[1]) Ниже будет показано, что каждая дуга всегда имеет метку, которую необходимо снять. — *Прим. перев.*

на дуге (a,b) и 5 на дуге (b, t). Далее вдоль цепи $P_{5,1}$ в момент времени $T = \delta - 5 = 1$ отправляется единица потока. Это дает следующие новые метки: 1 на дуге (s, b) и 3 на дуге (a, t). Кроме того, при просмотре цепи $P_{5,1}$ с дуги (a, b) снимается метка 3.

Полученное множество меток соответствует потоку наискорейшего прибытия в рассматриваемом графе за период в 6 интервалов времени.

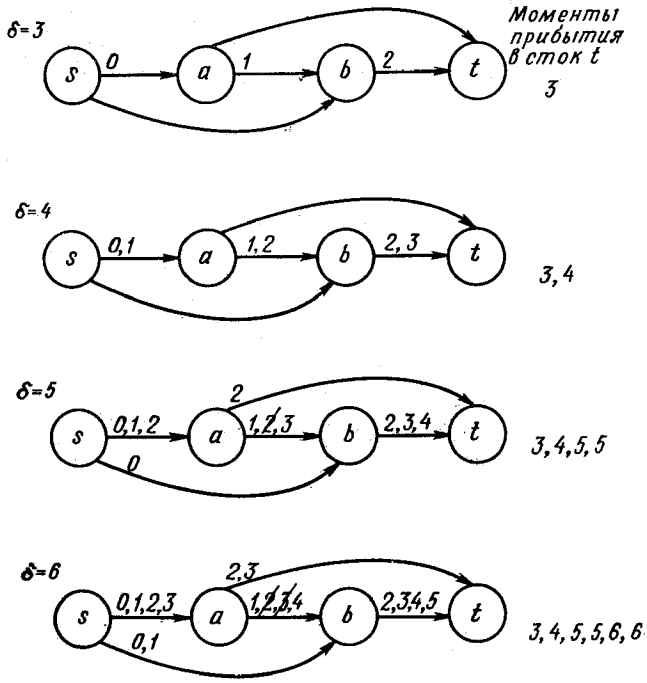


Рис. 4.13. Пример применения алгоритма поиска потока наискорейшего прибытия (пропускные способности всех дуг равны 1; $a(s, a) = a(a, b) = a(b, t) = 1$; $a(s, b) = a(a, t) = 3$).

Для обоснования алгоритма поиска потока наискорейшего прибытия нам понадобятся три вспомогательных результата.

ЛЕММА 4.1. Рассмотрим полную последовательность увеличивающих поток цепей, которая формируется алгоритмом поиска потока минимальной стоимости. Предположим, что две цепи этой последовательности P' и P''' содержат дугу (x, y) в качестве прямой. Тогда существует увеличивающая поток цепь P'', находящаяся в рассматриваемой последовательности между цепями P' и P''', которая содержит дугу (x, y) в качестве обратной.

Доказательство. Напомним, что пропускная способность дуги (x, y) равна единице. Не ограничивая общности, можно предпо-

лагать, что P' предшествует P''' в последовательности увеличивающих поток цепей, формируемых алгоритмом поиска потока минимальной стоимости. Поскольку дуга (x, y) в цепи P' является прямой, то после увеличения вдоль этой цепи потока данная дуга оказывается насыщенной. То же самое происходит, когда поток увеличивается вдоль цепи P'''. Поэтому дуга (x, y) должна быть обратной в некоторой увеличивающей поток цепи P'', формируемой после P', но перед P'''. Лемма доказана.

ЛЕММА 4.2. Предположим, что единица потока, проходя по увеличивающей цепи $P_{i,j}$ из s в t, затрачивает на прохождение каждой прямой дуги (x, y) время в $a(x, y)$ интервалов, а каждой обратной дуги (x, y) — время в $[-a(x, y)]$ интервалов. Тогда полное время прохождения единицы потока из s в t по цепи $P_{i,j}$ составит i интервалов.

Доказательство. Увеличивающая поток цепь $P_{i,j}$ генерируется на i-й итерации алгоритма поиска потока минимальной стоимости. При этом $p(s) = 0$, $p(t) = i$ и для всех дуг (x, y) $p(y) - p(x) = a(x, y)$. Отсюда непосредственно вытекает справедливость леммы.

ЛЕММА 4.3. Пусть $p_i(x)$ есть значение $p(x)$ на i-й итерации алгоритма поиска потока минимальной стоимости. Тогда при $i \leqslant j$

$$p_i(x) \leqslant p_j(x) \leqslant p_i(x) + j - i.$$

Доказательство. Перед началом каждой новой итерации алгоритма поиска потока минимальной стоимости любое вершинное число $p(x)$ либо увеличивается на $+1$, либо не меняется. Следовательно, $p_{i+1}(x) = p_i(x)$ либо $p_{i+1}(x) = p_i(x) + 1$, откуда и вытекает справедливость леммы.

Обоснование алгоритма поиска потока наискорейшего прибытия. Для обоснования данного алгоритма необходимо доказать следующие утверждения.

(а) Метки, формируемые алгоритмом, соответствуют некоторому потоку.

(б) Этот поток является потоком наискорейшего прибытия за период в p интервалов времени.

(в) Алгоритм завершает свою работу за конечное число шагов.

Доказательство утверждения (а). Чтобы показать, что формируемые метки соответствуют некоторому потоку, достаточно доказать следующие два утверждения:

(1) В процессе выполнения алгоритма не может возникнуть ситуации, при которой не будет сформирована метка, подлежащая устранению.

(2) На дугах не могут появляться одинаковые метки.

При доказательстве первого утверждения предположим, что дуга (x, y) является обратной дугой в некоторой цепи $P_{m,n}$. В силу леммы 4.1 дуга (x, y) должна быть прямой дугой в некото-

рой цепи, предшествующей цепи $P_{m,n}$ в формируемой последовательности увеличивающих поток цепей. Обозначим соответствующую цепь через $P_{i,j}$. Как было сказано, $i \leqslant m$. В соответствии с правилами выполнения алгоритма поиска потока наискорейшего прибытия дуга (x, y) — с учетом ее принадлежности цепи $P_{i,j}$ — получит метки $p_i(x)$, $p_i(x) + 1$, ..., $p_i(x) + m - i$, прежде чем возникнет необходимость (благодаря появлению цепи $P_{m,n}$) в снятии определенной метки с дуги (x, y). Принадлежность дуги (x, y) цепи $P_{m,n}$ указывает на то, что с нее следует снять метку $p_m(x)$, которая в силу леммы 4.3 уже сформирована. И вообще при просмотре цепи $P_{m,n}$, прежде чем потребуется снова снять метку с дуги (x, y), эта метка уже будет сформирована при просмотре цепи $P_{i,j}$. Поскольку формируемые метки образуют для отдельных дуг последовательности «соседних» целых чисел и поскольку снимаемые с дуг метки также образуют последовательности «соседних» целых чисел, то любая метка, которая должна быть снята с дуги (x, y), уже была для этой дуги сформирована. Это завершает доказательство первого утверждения.

Для доказательства второго утверждения предположим, что дуга (x, y) получила несколько одинаковых меток. Эти метки должны возникать при просмотре различных увеличивающих поток цепей, в которые дуга (x, y) входит в качестве прямой дуги. Пусть $P_{i,j}$ и $P_{m,n}$ обозначают две такие цепи. По лемме 4.1 существует увеличивающая поток $P_{k,l}$ цепь, занимающая промежуточное положение между цепями $P_{i,j}$ и $P_{m,n}$, в которую дуга (x, y) входит в качестве обратной дуги. При многократном просмотре цепи $P_{k,l}$ с дуги (x, y) снимаются метки $p_k(x)$, $p_k(x) + 1$, ..., $p_k(x) + m - k$, прежде чем при просмотре цепи $P_{m,n}$ дуге (x, y) начинают приписываться новые метки. Первая метка, которая будет получена при просмотре цепи $P_{m,n}$, — это метка $p_m(x)$, которая в силу леммы 4.3 уже была снята с дуги (x, y). Аналогично в процессе выполнения алгоритма при просмотре цепи $P_{k,l}$ каждая метка снимается с любой дуги всегда раньше, чем та же самая метка приписывается данной дуге при просмотре цепи $P_{m,n}$. Это завершает доказательство второго утверждения.

Доказательство утверждения (б). Покажем, что рассматриваемый алгоритм действительно формирует поток наискорейшего прибытия. Доказательство проведем по методу индукции. Очевидно, для $p = 0$ алгоритм формирует нужный поток. Предположим, что поток наискорейшего прибытия формируется алгоритмом и для $p = p_0 - 1$. Остается показать, что алгоритм формирует поток наискорейшего прибытия для $p = p_0$.

Поток, формируемый для периода в p_0 интервалов времени, состоит из потока, формируемого алгоритмом для периода в $(p_0 - 1)$ интервалов времени, и дополнительного количества единиц потока, прибывающих в сток в момент времени p_0. В силу леммы 4.2

по каждой из увеличивающих поток цепей $P_{0,1}$, $P_{0,2}$, ..., P_{0,w_0}, $P_{1,1}$, $P_{1,2}$, ..., P_{1,w_1}, $P_{\delta,1}$, $P_{\delta,2}$, ..., P_{δ,w_δ} единица потока может достичь стока к моменту времени P_0. Поскольку каждая из указанных цепей пропускает по одной единице потока, то общее их число составляет величину V_{p_0}.

В соответствии с методом индукции единицы сформированного алгоритмом потока, прибывающие в сток к моменту времени (p_0-1), составляют поток наискорейшего прибытия за период в $(p_0 - 1)$ интервалов времени и, следовательно, составляют также максимальный динамический поток за тот же период времени. С учетом этого, а также исходя из соотношения (49) можно заключить, что формируемый алгоритмом поток является не только максимальным динамическим потоком за период в p интервалов времени, но и потоком наискорейшего прибытия за тот же период времени, поскольку в момент времени p_0 в сток t прибывает V_{p_0} единиц потока.

Доказательство утверждения (*в*). Алгоритм завершается за конечное число шагов, поскольку метки возникают и исчезают при просмотре лишь конечного числа увеличивающих поток цепей. (Заметим, что число этих цепей конечно. Противное предположение означало бы, что алгоритм поиска потока минимальной стоимости не заканчивается за конечное число шагов.)

Таким образом, обоснование алгоритма поиска потока наискорейшего прибытия завершено.

В ряде ситуаций предпочтительнее обеспечивать прибытие в сток единиц потока в возможно более поздние моменты времени, а не в возможно более ранние, как в потоках наискорейшего прибытия. Например, если каждая единица потока представляет собой чек, по которому с вашего счета снимается соответствующая сумма, то для вас было бы предпочтительнее, чтобы чеки предъявлялись к оплате (приходили в сток) как можно позднее: тогда сумма вашего счета была бы в каждый момент времени максимально возможной. Можно привести и другой пример. Если представить единицами потока готовые изделия, хранение которых является весьма дорогостоящим делом, то было бы желательно принимать эти изделия на хранение (в стоке) как можно позже. Для ситуаций, аналогичных рассмотренным, следует интересоваться максимальным динамическим потоком, отдельные единицы которого передаются в сток как можно позже (до этого рассматривались потоки, отдельные единицы которого прибывали в сток как можно раньше).

Потоком наипоздней шего прибытия за период в p интервалов времени является такой максимальный динамический поток за этот период времени, который обеспечивает прибытие в сток наибольшего количества единиц в течение последнего интервала времени, в течение двух последних интервалов времени и т. д.

Всегда ли существует поток наипоздне́йшего прибытия за период в p интервалов времени? Да, всегда. Существование указанного потока может быть доказано с помощью конструкции, которая была использована при доказательстве существования потока наискорейшего прибытия. Отличие в данном случае состоит лишь в том, что сначала строится поток в сток t_p, затем в сток t_{p-1} и т. д.

Нет необходимости доказывать, что построение потока наипоздне́йшего прибытия с помощью упомянутой конструкции является при большом числе вершин и дуг в исходном графе или при большой величине p малоэффективным. Даже в специальном случае, когда пропускные способности всех дуг стационарны, видимо, не существует эффективного алгоритма построения потока наипоздне́йшего прибытия. Отметим, что в том же случае поток наискорейшего прибытия можно находить эффективно с помощью соответствующего алгоритма. Причина такого положения кроется в том, что при построении потока наипоздне́йшего прибытия необходимо, чтобы единицы потока покидали источник как можно позднее и задерживались в сети как можно дольше.

До сих пор мы рассматривали потоки, для которых исследовалась лишь картина прибытия отдельных единиц потока в сток и не исследовалась картина отправлений отдельных единиц потока из источника. Однако, естественно, существует немало ситуаций, в которых картина отправлений единиц потока из источника играет важную роль. Например, если вы распределяете готовую продукцию, сосредоточенную на складе, для вас, может быть, предпочтительнее отправлять продукцию со склада в распределительную сеть как можно раньше, чтобы минимизировать затраты на хранение. С другой стороны, если отдельные единицы потока представляют собой, например, учащихся, возвращающихся после летних каникул в школу (сток) из мест проживания (источников), то предпочтительнее задерживать отправление единиц потока из источников как можно дольше (если, конечно, принимать во внимание естественные желания школьников).

Аналогично тому, как определялись поток наискорейшего прибытия и поток позднейшего прибытия, мы можем определить *поток наискорейшего отправления* и *поток наипоздне́йшего отправления* за период в p интервалов времени. Доказательство существования каждого из этих потоков может быть проведено аналогично тому, как это делалось при доказательстве существования потока наискорейшего прибытия (при этом в доказательстве следует поменять местами источники и стоки).

Как и в случае потока наискорейшего (наипоздне́йшего) прибытия, конструкция, используемая при доказательстве существования соответствующего потока, не дает достаточно эффективного в вычислительном отношении метода построения этого потока. Тем не менее этот метод, видимо, является наилучшим.

Отметим, однако, то счастливое обстоятельство, что в специальном случае, когда пропускные способности дуг стационарны во времени, известен весьма эффективный способ построения потока позднейшего отправления. Соответствующая конструкция приводится ниже.

Пусть $G = (X, A)$ — произвольный граф. Определим обратный к нему граф G^{-1}. Граф G^{-1} имеет в качестве множества вершин множество X, а в качестве множества дуг множество

$$A^{-1}\{(y, x) : (x, y) \in A\}.$$

Таким образом, граф $G^{-1} = (X, A^{-1})$ представляет собой исходный граф G, в котором направления всех дуг просто заменены на обратные. Пусть пропускные способности и длительности прохождения для дуг графа G^{-1} совпадают с теми же характеристиками соответствующих дуг графа G.

Любой поток из s в t в графе G соответствует единственному потоку из t в s в графе G^{-1}, причем справедливо и обратное утверждение. Это следует из того, что маршрут каждой единицы потока в одном из графов совпадает с тем же маршрутом обратного направления в другом графе.

Важную роль для рассматриваемой конструкции играет следующий вспомогательный результат.

ЛЕММА 4.4. Предположим, что в графе G имеется максимальный динамический поток F за период в p интервалов времени. Пусть в этом потоке в момент времени i из источника отправляется x_i единиц, а в сток прибывает y_i единиц, где $i = 0, 1, \ldots, p$. Тогда в графе G^{-1} существует максимальный динамический поток за период в p интервалов времени, в котором в момент времени $(p - i)$ в сток прибывает x_i единиц, а из источника отправляется y_i единиц, где $i = 0, 1, \ldots, p$.

Доказательство. Поток в графе G^{-1}, который соответствует исходному потоку в графе G, обладает всеми необходимыми свойствами. Этот поток проходит по графу как бы в обратном времени, и поэтому время прибытия и отправления отдельных его единиц должно отсчитываться не от 0, а от p. Лемма доказана.

Из этой леммы следует, что потоку наискорейшего прибытия за период в p интервалов времени в графе G соответствует поток наипоздневшего отправления за тот же период времени в графе G^{-1}. Более того, поскольку $(G^{-1})^{-1} = G$, потоку наискорейшего прибытия за период в p интервалов времени в графе G^{-1} соответствует поток наипоздневшего отправления за тот же период в графе G.

Итак, поток наипоздневшего отправления за период в p интервалов времени в графе G может быть сформирован с помощью следующих процедур: 1) строится граф G^{-1} путем замены на обратные направлений всех дуг в графе G; 2) в графе G^{-1} определяется поток наискорейшего прибытия за период в p интервалов времени по

известному алгоритму; 3) определяется соответствующий поток в графе G (этот поток является в графе G потоком наипоздне́йшего отправления за период в p интервалов времени).

Таким образом, алгоритм поиска потока наискорейшего прибытия может быть использован и для поиска потока наипоздне́йшего отправления.

Теперь предположим, что для графа G получен поток наискорейшего прибытия за период в p интервалов времени с помощью соответствующего алгоритма. Следует ли для получения потока наипоздне́йшего отправления снова применить уже к графу G^{-1} алгоритм поиска потока наискорейшего прибытия? Следующая теорема дает ответ на этот вопрос.

ТЕОРЕМА 4.2. Если поток наискорейшего прибытия за период в p интервалов времени в графе G включает по y_i единиц, прибывающих в сток в моменты времени i, где $i = 0, 1, \ldots, p$, то поток наипоздне́йшего отправления за период в p интервалов времени в графе G включает по y_i единиц, отправляющихся из источника в моменты времени $p - i$, где $i = 0, 1, \ldots, p$.

Доказательство. Доказательство проведем от противного. Предположим, что в графе G для любого потока наипоздне́йшего отправления за период в p интервалов времени количество Z единиц потока, отправляющихся из источника за последние i интервалов времени, таково, что $Z \neq \sum_{j=0}^{j=i} y_j$ (противоречащее предположение). Тогда в силу леммы 4.4 в графе G^{-1} существует максимальный динамический поток за период в p интервалов времени, в котором Z единиц прибывает в сток в течение первых i интервалов времени. Однако максимальное количество единиц потока, которое может быть послано в графе G из источника в сток в течение последних i интервалов времени, равно максимальному количеству единиц потока, которое может быть послано в графе G^{-1} из источника в сток в течение первых i интервалов времени. Следовательно, $Z = \sum_{j=0}^{j=i} y_j$, что противоречит исходному предположению. Теорема доказана.

Итак, мы можем сделать вывод о том, что расписания потока наискорейшего прибытия и потока позднейшего отправления являются симметричными, точнее, количество единиц потока наискорейшего прибытия, попадающих в сток в момент времени i, совпадает с количеством единиц потока наипоздне́йшего отправления, покидающих источник в момент времени $(p - i)$.

Отметим благоприятное обстоятельство, заключающееся в том, что поток, построенный с помощью алгоритма поиска потока наискорейшего прибытия, обладает следующим важным свойством.

ТЕОРЕМА 4.3. Поток, формируемый алгоритмом поиска потока наискорейшего прибытия, также является и потоком наипоздней-шего отправления.

Доказательство. Пусть, как и раньше, через $P_{i,j}$ обозначена j-я увеличивающая поток цепь, выявленная на i-й итерации алгоритма поиска потока минимальной стоимости. В алгоритме поиска потока наискорейшего прибытия за период в p интервалов времени по цепи $P_{i,j}$ из источника в сток будет посылаться по одной единице потока в моменты времени $T = 0, 1, \ldots, p - i$ (см. лемму 4.2). Эти единицы потока также с учетом той же леммы будут прибывать в сток соответственно в моменты времени $T = i, i + 1, \ldots, p$. Поскольку это справедливо для любой увеличивающей поток цепи $P_{i,j}$, то каждой единице потока, прибывающей в сток в момент времени $p - k$, соответствует единица потока, отправляющаяся из источника в момент времени k. Следовательно, поток, формируемый алгоритмом поиска потока наискорейшего прибытия, имеет то же расписание, что и поток наипоздней-шего отправления. Теорема доказана.

В качестве примера рассмотрим поток, построенный алгоритмом поиска потока наискорейшего прибытия для графа, изображенного на рис. 4.12, *а*. Этот поток приведен на рис. 4.13 и имеет следующее расписание отправлений:

Момент времени	0	1	2	3	4	5	6
Количество единиц потока	2	2	1	1	0	0	0

Тот же поток имеет следующее расписание прибытий:

Момент времени	0	1	2	3	4	5	6
Количество единиц потока	0	0	0	1	1	2	2

Обратите внимание на симметрию двух приведенных расписаний.

К сожалению, между потоками наипоздней-шего прибытия и наискорейшего отправления не существует симметрии, аналогичной той, что имеет место между потоками наискорейшего прибытия и наипоздней-шего отправления. Продемонстрируем это на примере графа, изображенного на рис. 4.14. В этом графе за период в $p = 5$ интервалов времени можно переслать из источника в сток максимум 4 единицы потока. Непосредственный анализ показы-

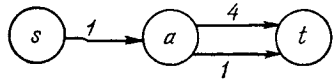


Рис. 4.14. Контрпример (пропускные способности всех дуг равны 1, время прохождения проставлено рядом с соответствующими дугами).

вает, что соответствующий поток имеет следующее расписание отправлений:

Момент времени	0	1	2	3	4	5
Количество единиц потока	1	1	1	1	0	0

Тот же поток, как показывает анализ, имеет следующее расписание прибытий:

Момент времени	0	1	2	3	4	5
Количество единиц потока	0	0	0	1	1	2

Рассмотрим некоторое множество $S = \{a, b, c, \ldots\}$. Будем говорить, что на множестве S задано отношение *лексикографического предпочтения*, если любая единица, относящаяся к элементу a, предпочтительнее любого количества единиц, относящихся к остальным элементам b, c, ...; любая единица, относящаяся к элементу b, предпочтительнее любого количества единиц, относящихся к остальным элементам c, ..., и т. д. Поток наискорейшего прибытия называется *лексикографическим потоком*, поскольку он определяет лексикографическое предпочтение на множестве стоков t_0, t_1, t_2, ..., t_p относительно прибывающих в них единиц потока. Аналогично поток наипоздневшего прибытия, поток наискорейшего отправления и поток наипоздневшего отправления также называются лексикографическими потоками, поскольку для каждого из этих потоков существует лексикографическое предпочтение для источников или стоков относительно рассматриваемых единиц потока.

Мы завершаем данный раздел представлением результата, который показывает, что в максимальных динамических потоках

расписания отправлений единиц потока из источника и расписания прибытия единиц потока в сток взаимозаменяемы.

ТЕОРЕМА 4.4. Пусть F' и F'' — два любых максимальных динамических потока в графе G. Тогда в этом графе существует максимальный динамический поток, у которого расписание отправлений из источника то же самое, что и потока F', а расписание прибытий то же самое, что и потока F''.

Доказательство. Выберем некоторый минимальный разрез, отделяющий в графе G источник от стока. Так как F' и F'' являются максимальными динамическими потоками, каждый из них насыщает данный разрез. Сформируем из F' и F'' смешанный поток, который совпадает с потоком F' в той части разрезанного графа, где находится источник, и совпадает с потоком F'' в той части графа, где находится сток. Этот смешанный поток имеет расписание отправлений то же, что и потока F', а расписание прибытий то же, что и потока F''. Таким образом, сформированный поток является тем самым потоком, о существовании которого утверждает теорема.

ЛЕММА 4.5.[1]). Существуют следующие динамические потоки:

1. Поток наискорейшего отправления из стока и наискорейшего прибытия в сток.
2. Поток наискорейшего отправления из стока и наипоздней-шего прибытия в сток.
3. Поток наипоздней-шего отправления из стока и наискорей-шего прибытия в сток.
4. Поток наипоздней-шего отправления из стока и наипоздней-шего прибытия в сток.

Доказательство. Сформулированный в лемме результат непосредственно получается после применения теоремы 4.4 к графу G.

Как следует из теоремы 4.3, с помощью алгоритма поиска потока наискорейшего прибытия может быть построен поток наискорейшего прибытия в сток и поток наипоздней-шего отправления из источника, т. е. поток, занимающий третью позицию в списке леммы 4.5. Другие три потока из того же списка (позиции 1, 2 и 4) можно было бы построить путем совмещения двух разных потоков, как это делалось при доказательстве теоремы 4.4. К сожалению, вопрос о существовании более эффективного метода построения этих трех потоков остается открытым.

---

[1]) Точнее было бы назвать данное утверждение следствием теоремы 4.4. — *Прим. перев.*

4.6. Потоки с усилениями

В предшествующих рассмотрениях поток, входивший в любую дугу, не изменялся: единица вошла — единица вышла. При прохождении потока через дугу новые единицы не создавались, но и старые не исчезали. В данном разделе мы отказываемся от предположения, согласно которому при прохождении по дугам поток остается неизменным. Напротив, мы допускаем, что количество единиц в потоке, проходящем по дуге, может увеличиваться или уменьшаться. Точнее, мы будем считать, что если в любую дугу (x, y) в вершине x входит $f(x, y)$ единиц потока, то из этой дуги в вершине y выйдет $k(x, y)\, f(x, y)$ единиц потока. Можно считать, что каждая единица потока, проходящая по дуге (x, y), умножается на величину $k(x, y)$. Эта величина называется *коэффициентом усиления* или просто *усилением дуги* (x, y).

ПРИМЕР 1. Некоторый питомник по разведению саженцев должен отправлять выращенные растения своим потребителям. При перевозке к одним потребителям доля погибающих растений велика, что связано с неблагоприятными климатическими условиями на маршруте перевозки. При перевозке к другим потребителям ввиду благоприятных климатических условий растения, вообще говоря, значительно вырастают за время перевозки.

Маршруты, на которых растения частично погибают, можно рассматривать как дуги с коэффициентами усиления, меньшими единицы, а маршруты, на которых растения развиваются, можно рассматривать как дуги с коэффициентами усиления, большими единицы.

ПРИМЕР 2. Финансист некоторой корпорации должен решить вопрос об использовании имеющегося капитала в различных возможных вариантах инвестиций. Каким образом можно построить сеть с усилениями для использования ее при решении данной проблемы инвестирования?

Финансист может представлять каждый вариант инвестирования в виде дуги некоторого графа, которая соединяет вершины, соответствующие началу и концу периода инвестирования. Если, скажем, некоторый вариант инвестирования приносит доход в 8%, то с соответствующей дугой следует связать коэффициент усиления, равный 1,08. Пропускную способность каждой дуги следует положить равной максимальной величине капитала, который может быть инвестирован по соответствующему варианту. Если каждый инвестируемый доллар представлять в виде единицы потока, протекающей в построенной таким образом сети, то проблема инвестиций может быть сформулирована как задача пересылки в данной сети из источника в сток максимального количества единиц потока при условии ограниченного запаса этих единиц в источнике (источником является вершина, представляющая текущий момент времени, а стоком — вершина, представляющая момент времени, к которому должен истекать срок всех инвестиций).

Если $k(x, y) > 1$, то поток в дуге (x, y) *усиливается*. Если $k(x, y) = 1$, то поток в дуге (x, y) остается *неизменным*. Если $0 < k(x, y) < 1$, то поток в дуге (x, y) *ослабляется*. Если $k(x, y) = 0$, то поток в дуге (x, y) *теряется* и данная дуга может рассматриваться как сток. В дальнейшем мы будем всегда предполагать, что все дуги (x, y), для которых $k(x, y) = 0$, заменены стоками. Если $k(x, y) <$

< 0, то для каждой единицы потока, входящей в вершину x, в вершину y должно попадать $-k(x, y)$ единиц потока, т. е. в данном случае дуга (x, y) может рассматриваться как *порождающая спрос на поток*.

Центральной задачей данного раздела является задача о потоке минимальной стоимости в сети с усилениями. Как можно понять из названия, будет рассматриваться задача поиска оптимального с точки зрения затрат способа пересылки через сеть потока из источника в сток при заданном количестве V единиц потока в источнике и при наличии усиления в дугах сети. Следует при этом иметь в виду, что если из источника в сеть отправляется V единиц потока, то количество единиц потока, прибывающих в сток из-за усиления в дугах, не обязательно совпадает с V. (Заметим, что во всех потоковых задачах, которые рассматривались ранее, количество единиц потока, выходивших в сеть из источника, всегда равнялось количеству единиц потока, прибывавших в сток.)

Как и раньше, для дуги (x, y) задается максимальное $c(x, y)$ и минимальное $l(x, y)$ количество единиц потока, которое может входить в дугу (x,y), а также стоимость $a(x, y)$ прохождения по дуге каждой вошедшей в нее единицы потока. По-прежнему $f(x, y)$ будет обозначать количество единиц потока, входящих в дугу (x, y).

Задача о потоке минимальной стоимости в сети с усилениями может быть сформулирована следующим образом:

минимизировать

$$\sum_{(x, y)} a(x, y) f(x, y) \tag{50}$$

при условии, что

$$\sum_{y} f(x, y) - \sum_{y} k(y, x) f(y, x) = \begin{cases} V, & \text{если } x = s, \\ 0, & \text{если } x \neq s,\ x \neq t. \end{cases} \tag{51}$$

$$l(x, y) \leqslant f(x, y) \leqslant c(x, y) \text{ [для всех } (x, y)]. \tag{52}$$

Выражение (50) определяет общую стоимость потока. Уравнение (51) показывает, что чистый поток из вершины s должен быть равен V, а чистый поток из любой другой вершины, не считая вершины t, должен быть равен 0. Соотношение (52) показывает, что величина потока в каждой дуге должна находиться в интервале между нижней и верхней границами пропускной способности.

Задача о потоке минимальной стоимости в сети с усилениями, описываемая соотношениями (50) — (52), является задачей линейного программирования. Пусть $p(x)$ обозначает двойственную переменную, соответствующую уравнению (51) для вершины x. Пусть $\gamma_1(x, y)$ обозначает двойственную переменную, соот-

ветствующую верхнему ограничению на пропускную способность дуги (x, y) в соотношении (52). Пусть $\gamma_2(x, y)$ обозначает двойственную переменную, соответствующую нижнему ограничению на пропускную способность дуги (x, y) в соотношении (52). В данных переменных задача линейного программирования, двойственная к задаче, описываемой соотношениями (50) — (52), формулируется следующим образом:

максимизировать

$$Vp(s) - \sum_{(x, y)} c(x, y)\,\gamma_1(x, y) + \sum_{(x, y)} l(x, y)\,\gamma_2(x, y) \tag{53}$$

при условии, что для всех дуг

$$p(x) - k(x, y)\,p(y) - \gamma_1(x, y) + \gamma_2(x, y) \leqslant a(x, y), \tag{54}$$

$$\gamma_1(x, y) \geqslant 0, \tag{55}$$

$$\gamma_2(x, y) \geqslant 0 \tag{56}$$

и что для всех вершин

$$p(x) \text{ может иметь любой знак.} \tag{57}$$

Если задать значения двойственных переменных $p(x)$, где $x \in X$, то значения остальных двойственных переменных $\gamma_1(x, y)$ и $\gamma_2(x, y)$ для всех дуг (x, y) должны удовлетворять соотношению

$$-\gamma_1(x, y) + \gamma_2(x, y) \leqslant a(x, y) - p(x) + k(x, y)\,p(y) \triangleq \zeta(x, y). \tag{58}$$

Для удобства правая часть неравенства (58) обозначена через $\zeta(x, y)$.

При максимизации выражения (53) для целевой функции двойственной задачи выбор значений переменных $\gamma_1(x, y)$ и $\gamma_2(x, y)$ можно осуществлять в соответствии со следующими правилами:

если $\zeta(x, y) \geqslant 0$, то

$$\gamma_1(x, y) = 0,\ \gamma_2(x, y) = \zeta(x, y) \tag{59}$$

а если $\zeta(x, y) < 0$, то

$$\gamma_1(x, y) = -\zeta(x, y),\quad \gamma_2(x, y) = 0. \tag{60}$$

Приведенные правила обосновываются тем, что переменные $\gamma_1(x, y)$ и $\gamma_2(x, y)$ входят лишь в одно общее для них ограничение (54).

Таким образом, значения переменных $\gamma_1(x, y)$ и $\gamma_2(x, y)$ для всех дуг (x, y) определяются значениями двойственных переменных $p(x)$. Следовательно, решение двойственной задачи (53) — (57) сводится лишь к поиску набора оптимальных значений двойственных вершинных переменных $p(x)$.

Условия дополняющей нежесткости для рассматриваемой здесь пары двойственных задач линейного программирования имеют следующий вид:

$$\gamma_1(x, y) > 0 \Rightarrow f(x, y) = c(x, y), \tag{61}$$

$$\gamma_2'(x, y) > 0 \Rightarrow f(x, y) = l(x, y). \tag{62}$$

Поскольку значения переменных $\gamma_1(x, y)$ и $\gamma_2(x, y)$ определяются величиной $\zeta(x, y)$, то условия дополняющей нежесткости (61) и (62) могут быть следующим образом переформулированы:

$$\zeta(x, y) < 0 \Rightarrow f(x, y) = c(x, y), \tag{63}$$

$$\zeta(x, y) > 0 \Rightarrow f(x, y) = l(x, y). \tag{64}$$

Итак, чтобы решить задачу о потоке минимальной стоимости на сети с усилениями, необходимо найти такие допустимые значения потоков $f(x, y)$ и такие значения двойственных вершинных переменных $p(x)$, при которых условия дополняющей нежесткости (63) и (64) выполняются для всех дуг (x, y).

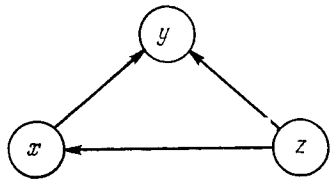


Рис. 4.15. Генерирующий цикл с направлением обхода по часовой стрелке $[k(x, y) = 2, k(y, z) = \frac{1}{3}, k(z, x) = 3]$.

Рассмотрим цикл, изображенный на рис. 4.15, независимо от остальной части включающего его графа. Если одна дополнительная единица потока прибывает в вершину x и входит в дугу (x, y), то после прохождения по дуге (x, y) соответствующего потока количество единиц его в вершине y возрастет до двух, затем это количество изменится в вершине z до величины 2/3 и, наконец, при возврате в вершину x станет равным 2 единицам потока. Таким образом, каждая дополнительная единица потока, проходящая по рассматриваемому циклу в направлении движения часовой стрелки, превращается в две единицы потока. В связи с этим указанный цикл называется *генерирующим циклом с направлением обхода по часовой стрелке*. В общем случае цикл C называется генерирующим циклом с направлением обхода по часовой стрелке, если

$$k_C \triangleq \frac{\prod\limits_{(x, y) \in C_1} k(x, y)}{\prod\limits_{(x, y) \in C_2} k(x, y)}, \tag{65}$$

где C_1 и C_2 обозначают множества соответственно прямых и обратных дуг для указанного направления обхода цикла C. Ана-

логично можно определить *генерирующий цикл с направлением обхода против часовой стрелки* как цикл, для которого выполняется неравенство (65) при обходе его против часовой стрелки. Генерирующие циклы с направлением обхода как по часовой, так и против часовой стрелки играют важную роль, поскольку предоставляют возможность генерирования дополнительных единиц потока.

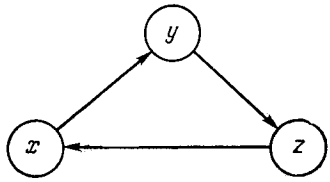


Рис. 4.16. Поглощающий цикл с направлением обхода по часовой стрелке $[k(x, y) = 1,\ k(z, y) = -\frac{1}{2}, k(z, x) = \frac{1}{4}]$.

Рассмотрим теперь цикл, изображенный на рис. 4.16. Если одна дополнительная единица потока прибывает в вершину x и входит в дугу (x, y), а затем проходит весь указанный цикл по часовой стрелке, то в вершине x обнаружится потребность в $^1/_2$ единицы потока. Это обусловливается тем, что при прохождении дополнительной единицы потока по дуге (x, y) одна единица потока попадает в вершину y. Приход в вершину y одной дополнительной единицы потока требует увеличения потока в дуге (z, y) до двух единиц. Но в силу возрастания потока в дуге (z, y) до двух единиц в вершине x дополнительно должна быть 1/2 единицы потока. Таким образом, если в вершину x приходит $1^1/_2$ единицы потока, то одна единица потока может быть отправлена в рассматриваемый цикл с обходом его по часовой стрелке, а оставшаяся 1/2 единицы потока может быть использована для удовлетворения дополнительной потребности в вершине x. Итак, данный цикл поглощает поток и называется *поглощающим циклом с направлением обхода по часовой стрелке*. В общем случае цикл C называется поглощающим циклом с направлением обхода по часовой стрелке, если

$$k_C \triangleq \frac{\prod\limits_{(x, y)\in C_1} k(x, y)}{\prod\limits_{(x, y)\in C_2} k(x, y)}, \tag{66}$$

где C_1 и C_2 имеют тот же смысл, что и в (65). Аналогично можно определить *поглощающий цикл с направлением обхода против часовой стрелки* как цикл, для которого неравенство (66) выполняется при обходе его против часовой стрелки.

Поскольку граф может содержать генерирующий и (или) поглощающий циклы, нельзя гарантировать, что каждая единица потока, покидающая источник, в конце концов придет в сток (она может быть поглощена соответствующим циклом) или что каждая единица потока, прибывающая в сток, первоначально

вышла из источника (она могла возникнуть в некотором генерирующем цикле).

Отметим, что если величина k_C вычисляется при обходе цикла C по часовой стрелке, то аналогично вычисляемая величина при обходе цикла C против часовой стрелки составит $1/k_C$. Это обусловливается тем, что изменение направления обхода цикла делает прямые дуги обратными, а обратные дуги прямыми. Поэтому в выражении для k_C числитель и знаменатель меняются местами.

Если исходный граф не содержит поглощающих или генерирующих циклов, т. е. при обходе в любом направлении каждого цикла $Ck_C = 1$, то задача о потоке минимальной стоимости в сети с усилениями может быть сведена к обычной задаче о потоке минимальной стоимости (на сети без усилений), которая решается с помощью алгоритма дефекта. Покажем теперь, как преобразовать задачу о потоке минимальной стоимости при наличии усилений в обычную задачу о потоке минимальной стоимости.

Рассмотрим ограничения (52) на пропускную способность дуг для задачи с усилениями и уравнения сохранения (51). Обозначим через E матрицу коэффициентов, входящих в уравнения (51). Если какой-либо столбец матрицы E умножить на отличную от нуля константу, то множество допустимых решений соответствующей системы линейных соотношений фактически не изменится. Уточним данное утверждение. Предположим, что столбец матрицы E, соответствующей дугам (x, y), умножается на отличную от нуля константу c. Тогда в исходной системе равенств (51) соответствующие переменные $f(x, y)$ следует заменить на $cf(x, y)$. Если величины $l(x, y)$ и $c(x, y)$ также умножить на c, то $cf(x, y)$ можно заменить новой переменной $f'(x, y)$. При этом ограничения (52) в преобразованной задаче будут иметь тот же самый вид, что и в исходной.

Можно ли найти для столбцов и строк матрицы E такие множители, чтобы после соответствующей замены переменных новая задача стала обычной потоковой задачей на сети без усилений? Следующая теорема, принадлежащая Труемперу [10], дает на этот вопрос утвердительный ответ.

ТЕОРЕМА 4.5. Рассматриваемая потоковая задача на сети с усилениями может быть преобразована в аналогичную потоковую задачу на сети без усилений тогда и только тогда, когда существуют такие вершинные числа $m(x)$, что для всех дуг (x, y)

$$\frac{m(x)\,k(x, y)}{m(y)} = 1. \tag{67}$$

Доказательство. Предположим, что существуют вершинные числа $m(x)$, такие, что для всех дуг (x, y) выполняется соотношение (67). Умножим уравнение сохранения потока (51) для каж-

дой вершины x на величину $1/m(x)$. Положим для всех дуг (x, y) $f'(x, y) = f(x, y)/m(x)$. Перепишем теперь соотношения (50) — (52) в новых переменных f'. После этого рассматриваемые соотношения примут вид соотношений (24) — (26), описывающих обычную задачу о потоке минимальной стоимости.

Чтобы доказать обратное утверждение, предположим, что задача о потоке на сети с усилениями может быть преобразована в задачу о потоке на сети без усилений. Тогда для каждого столбца матрицы E должен найтись множитель, который преобразует ограничения (51) задачи с усилениями в соответствующие ограничения задачи без усилений. Обозначим этот множитель для ограничения (51) в вершине x через $1/m(x)$.

В задаче без усилений коэффициент при каждой переменной $f'(x, y)$ в уравнении сохранения (25) должен быть равен $+1$; следовательно, $f'(x, y) = f(x, y)/m(x)$. В той же задаче коэффициент при каждой переменной $f'(y, x)$ в уравнении сохранения (25) должен быть равен -1; следовательно, $f'(y, x) = -k(y, x) f(y, x)/m(x) = -k(y, x) f'(y, x)/m(x)$, откуда для всех дуг (x, y) должно быть $k(y, x) m(y)/m(x) = 1$. Последнее соотношение совпадает с (67). Теорема доказана.

Таким образом, задача о потоке на сети с усилениями может быть преобразована в задачу о потоке на сети без усилений, если существуют вершинные числа, удовлетворяющие (67). В каком случае такие вершинные числа существуют? Каким образом их можно определить? Ответы на эти вопросы даются ниже.

ТЕОРЕМА 4.6. Вершинные числа, удовлетворяющие условиям теоремы 4.5, существуют в том и только том случае, когда исходный граф не содержит ни поглощающих, ни генерирующих циклов, т. е. $k_C = 1$ для любого цикла C.

Доказательство. Предположим, что существует такой набор вершинных чисел, что для всех дуг (x, y)

$$m(x, y) = k(x, y)\, m(x)/m(y) = 1.$$

Рассмотрим любой цикл C. С учетом сделанного предположения

$$k_C = \frac{\prod\limits_{(x, y)\in C_1} k(x, y)}{\prod\limits_{(x, y)\in C_2} k(x, y)} = \frac{\prod\limits_{(x, y)\in C_1} \frac{m(x)\, k(x, y)}{m(y)}}{\prod\limits_{(x, y)\in C_2} \frac{m(x)\, k(x, y)}{m(y)}} = \frac{\prod\limits_{(x, y)\in C_1} m(x, y)}{\prod\limits_{(x, y)\in C_2} m(x, y)} = 1, \tag{68}$$

где C_1 и C_2 имеют тот же смысл, что и в (65), (66). Следовательно, если для всех дуг (x, y) все величины $m(x, y)$ равны 1, в графе не может существовать ни одного поглощающего или генерирующего цикла.

Обратно, предположим, что в исходном графе не существует

ни одного генерирующего или поглощающего цикла. Выберем в графе любое покрывающее дерево[1]). Пусть $m(s) = 1$ и пусть $m(x, y) = 1$ для всех дуг в дереве T. Известным способом пройдем по дугам дерева T, вычисляя для всех вершин величины $m(x)$. Эти величины определяются однозначно.

Выберем теперь любую дугу (x, y), не входящую в дерево T. Эта дуга образует единственный цикл C с дугами дерева T. Поскольку цикл C не может быть ни генерирующим, ни поглощающим и поскольку для всех дуг (i, j) этого цикла, входящих в T, $m(i, j) = 1$, то в силу (68) величина $m(x, y)$ также равна 1. Таким образом, $m(x, y) = 1$ для всех дуг, не входящих в дерево T. Теорема доказана.

Итак, если сеть не содержит ни поглощающих, ни генерирующих циклов, то задача о потоке минимальной стоимости на сети с усилениями может быть сведена к задаче о потоке минимальной стоимости на сети без усилений, которая в общем случае решается алгоритмом дефекта или алгоритмом поиска потока минимальной стоимости в случае равенства нулю нижних пропускных способностей дуг. Указанное сведение одной задачи к другой осуществляется путем умножения уравнений сохранения потока (51) для каждой вершины x на коэффициенты $1/m(x)$, где $m(x)$ определяется так же, как это делалось при доказательстве теоремы 4.6.

Для сетей, содержащих поглощающие и генерирующие циклы, рассматриваемая задача усложняется. Далее мы представим алгоритм Джевелла [4], предназначенный для решения данной задачи. Этот алгоритм называется *алгоритмом поиска потока с усилениями*. Он включает три основных шага.

Шаг 1. (*Задание начальных условий.*) На этом шаге выполняется поиск таких значений величин потоков $f(x\ y)$ и таких значений вершинных чисел $p(x)$, при которых выполняются условия дополняющей нежесткости (63), (64) и условия допустимости потока (51), (52). Однако при этом из источника может отправляться количество единиц потока, меньшее V. Итак, на шаге 1 отыскивается решение, которое удовлетворяет всем прямым и двойственным ограничениям, а также условиям дополняющей нежесткости. Единственное условие, которое не удовлетворяется, — это требование выхода из источника заданного количества единиц потока. (Как будет видно несколько позже, шаг 1 можно упростить путем построения графа, являющегося некоторым расширением исходного графа.)

Шаг 2. (*Увеличение потока.*) На этом шаге увеличивается количество единиц потока, выходящих из источника, при выпол-

[1]) По смыслу потоковых задач исходный граф должен быть связным. Поэтому указанное дерево всегда существует.— *Прим. перев.*

нении всех прямых и двойственных ограничений, а также условий дополняющей нежесткости.

Шаг 3. (*Изменение двойственных переменных.*) Данный шаг предписывает, каким образом изменять значения двойственных переменных, чтобы большее число единиц потока можно было отправить из источника в сеть.

В рассматриваемом алгоритме прежде всего выполняется шаг 1, связанный с поиском начального потока. Затем выполняется шаг 2, на котором из источника в сеть посылается максимальное количество единиц потока таким образом, чтобы выполнялись все условия дополняющей нежесткости. После этого выполняется шаг 3, на котором двойственные переменные изменяются так, что появляется возможность еще большего увеличения потока, выходящего из источника. Затем осуществляется возврат к шагу 2. Каждый раз, когда на шаге 2 увеличение потока, выпускаемого в сеть, становится невозможным, проводится шаг 3. Данная процедура повторяется до тех пор, пока из источника в сеть не будет отправлено V единиц потока. Если допустимого потока, в котором из источника отправляется V единиц, не существует, то алгоритм обнаруживает это при выполнении шага 3 и завершает свою работу.

Используя приведенные выше соображения, мы теперь можем формально описать алгоритм поиска потока с усилениями.

Алгоритм поиска потока с усилениями

Шаг 1. (*Задание начальных условий.*) Данный шаг показывает, как определить множество значений величин потоков $f(x, y)$ и двойственных переменных $p(x)$ для сети, эквивалентной исходной. При этом удовлетворяются все условия допустимости (51), (52), и все условия дополняющей нежесткости (63), (64), однако чистый поток из источника может быть меньше или равен заданному количеству V единиц потока, отправляемого из источника.

Произвольно задать значения для двойственных переменных $p(x)$, где $x \in X$. Обратиться к условиям дополняющей нежесткости

$$\zeta(x, y) = a(x, y) - p(x) + k(x, y)\, p(y) < 0 \Rightarrow f(x, y) = c(x,y) \quad (69)$$

$$\zeta(x, y) = a(x, y) - p(x) + k(x, y)\, p(y) > 0 \Rightarrow f(x, y) = l(x, y) \quad (70)$$

с тем чтобы определить, какие из значений $f(x, y)$ с ними несовместимы. Для каждой дуги (x, y) выбрать такие значения потока $f(x, y)$, которые совместимы с соответствующими условиями (69) и (70).

Далее вычислить чистый излишек потока $V(x)$ в каждой вершине x, определяя величину $V(x)$ следующим образом:

$$V(s) = \sum_y f(x, y) - \sum_y k(y, x) f(y, x) - V,$$

$$V(x) = \sum_y f(x, y) - \sum_y k(y, x) f(y, x). \qquad (71)$$

Если $V(x) = 0$ для всех x, то текущее решение является максимальным, так как оно представляет собой допустимый поток, который удовлетворяет всем условиям дополняющей нежесткости для выбранных значений $p(x)$. (Заметим, что, как правило, на данном этапе не удается получить оптимального потока.)

Построить новую сеть, добавив к исходной сети вершину S и дуги (S, x), ведущие из S в каждую вершину x, для которой $V(x) \neq 0$. Для каждой дуги (S, x) положить $c(S, x) = |V(x)|$. Кроме того, положить $k(S, x) = +1$, если $V(x) < 0$, и $k(S, x) = -1$, если $V(x) > 0$. Наконец, для всех дуг (S, x) положить $a(S, x) = 0$.

Пусть поток в каждой добавленной дуге (S, x) равен 0, а во всех остальных дугах поток тот же, каким он был выбран в исходной сети. Очевидно, в новой сети условие сохранения потока во всех промежуточных вершинах не выполняется, поскольку по крайней мере в одной из таких вершин — вершине x — получается $V(x) \neq 0$. Однако если бы можно было отправить в новую сеть из вершины S достаточное количество единиц потока, которые насытили бы все дуги (S, x), то возникающий в новой сети (недопустимый) поток соответствовал бы допустимому потоку в исходной сети. Последнее связано с тем, что указанный поток доставлял бы в вершину x дополнительное количество единиц потока, в точности совпадающее с излишком чистого потока в вершине x. Более того, потоку минимальной стоимости в новой сети, насыщающему все дуги (S, x), соответствует поток минимальной стоимости в исходной сети, так как все величины $a(S, x)$ равны 0. Следовательно, определяя оптимальный поток в новой сети, мы можем найти оптимальный поток в исходной сети.

Формулировка задачи о потоке минимальной стоимости для новой сети в виде задачи линейного программирования эквивалентна аналогичной формулировке той же задачи для исходной сети, которая описывается соотношениями (50) — (52). Отличие обеих формулировок состоит лишь в том , что в задаче для новой сети в качестве источника фигурирует новая вершина S, а в уравнении (51) для вершины x 0 в правой части заменяется величиной $V(x)$. Условия дополняющей нежесткости для новой сети аналогичны таким же условиям для исходной сети. Если для вершинных чисел $p(x)$ сохранить те значения, которые они имели

в исходной сети, и положить величину $p(S)$ равной достаточно большому по модулю отрицательному числу, то условия дополняющей нежесткости будут выполняться и для новой сети. Таким образом, первоначальный выбор значений потока и двойственных переменных в исходной сети позволяет сделать такой выбор значений потоков и двойственных переменных в новой сети, при котором потоки и двойственные переменные удовлетворяют условиям допустимости и условиям дополняющей нежесткости. Итак, теперь необходимо увеличить количество единиц потока, выходящих из вершины S.

Перейти к шагу 2.

Шаг 2. (*Увеличение потока.*) На данном шаге поток из вершины S увеличивается настолько, насколько это возможно. Причем увеличение потока осуществляется так, что условия дополняющей нежесткости (69) — (70) не нарушаются и значения двойственных переменных не изменяются.

Для каждой дуги (x, y) определить, допускают ли условия дополняющей нежесткости (69), (70) увеличение или уменьшение значения потока $f(x, y)$. Сформулировать множество увеличивающих дуг I и множество уменьшающих дуг R (далее на данном шаге будут рассматриваться лишь дуги из множеств I и R).

Чтобы определить, можно ли из S отправить дополнительные единицы потока, используя только дуги из множеств I и R, необходимо последовательно наращивать дерево с корнем в вершине S аналогично тому, как это делалось в алгоритме поиска максимального потока.

Дуги, включаемые в упомянутое дерево, будут окрашиваться. Если некоторая вершина покрывается деревом, то это означает, что дополнительные единицы потока могут быть отправлены из источника в эту вершину по дугам построенного дерева. При этом каждый раз, когда дуга, инцидентная вершине x, будет добавляться к дереву, а значит, окрашиваться, вершина x будет получать метку $f(x)$. Эта метка указывает на количество единиц потока, которые достигли бы вершины x, если из источника по соответствующей цепи была отправлена одна единица потока.

Если помечается сток, то это означает, что увеличивающая поток цепь из источника в сток найдена. По этой цепи из источника в сток отправляется максимально возможное количество единиц потока.

Если в процедуре окрашиваний может быть окрашена дуга, образующая цикл с ранее уже окрашенными дугами, то необходимо проверить, не может ли соответствующий цикл поглощать поток, отправленный из источника. Если может, то в этот цикл из источника посылается максимально возможное количество единиц потока, которое будет в нем поглощаться. Если же в указанном цикле поток, посланный из источника, не может погло-

щаться, то с одной из дуг цикла снимается окраска и процесс окрашивания продолжается.

Процедура наращивания дерева включает последовательное окрашивание дуг и присвоение вершинам меток $f(x)$. Перед началом процедуры все дуги являются неокрашенными и все вершины — непомеченными, кроме вершины S, для которой $f(S) = 1$. В рассматриваемой процедуре операции по окрашиванию дуг и разметке вершин выполняются следующим образом.

Окрасить дугу (x, y) при выполнении одного из четырех условий:

(1) Вершина x помечена, $f(x) > 0$ и $(x, y) \in I$;

(2) Вершина x помечена, $f(x) < 0$ и $(x, y) \in R$;

(3) Вершина y помечена, $f(y) > 0$ и либо $(x, y) \in R$, $k(x, y) > 0$, либо $(x, y) \in I$, $k(x, y) < 0$;

(4) Вершина y помечена, $f(y) < 0$ и либо $(x, y) \in I$, $k(x, y) > 0$, либо $(x, y) \in R$, $k(x, y) < 0$.

Если дуга (x, y) окрашивается в силу выполнения условия (1) или условия (2) (при этом мы будем говорить, что дуга (x, y) окрашивается из вершины x), то пометить вершину y величиной $f(y) = f(x)\, k(x, y)$. Если дуга (x, y) окрашивается в силу выполнения условия (3) или условия (4) (при этом мы будем говорить, что дуга (x, y) окрашивается из вершины y), то пометить вершину x величиной $f(x) = f(y)/k(x, y)$.

Подчеркнем, что метка вершины $f(x)$ определяет количество единиц потока, которое накапливается в вершине x в результате прохождения одной единицы потока по цепи из окрашенных дуг, соединяющей вершины S и x и генерирующей метку $f(x)$.

Процедуру окрашивания и формирования меток продолжать до выполнения одного из следующих трех условий:

(1) Помечен сток.

(2) Некоторая вершина получает две несовпадающие метки.

(3) Нельзя окрасить ни одной новой дуги и пометить ни одной новой вершины.

Если имеет место условие (3), перейти к шагу 3.

Если имеет место условие (1) (а это значит, что найдена единственная цепь из окрашенных дуг, соединяющая источник и сток), отправить по данной цепи максимально возможное количество единиц потока. Затем вернуться к началу шага 2.

Если имеет место условие (2), т. е. если некоторая вершина x получила две различные метки $f_1(x)$ и $f_2(x)$ (не ограничивая общности, будем считать, что $f_1(x) < f_2(x)$), рассмотреть два случая: первый — $f_1(x)$ и $f_2(x)$ имеют разные знаки; второй — $f_1(x)$ и $f_2(x)$ имеют один и тот же знак (в обоих случаях метки соответствуют двум различным цепям, соединяющим вершины S и x, причем эти цепи могут частично совпадать).

Если $f_1(x)$ и $f_2(x)$ имеют разные знаки (в этом случае единица

потока, посланная из S по цепи, генерирующей метку $f_1(x)$, вызывает необходимость накопления в вершине x $|f_1(x)|$ единиц потока, а та же единица потока, посланная из S по цепи, генерирующей метку $f_2(x)$, «превращается» в $f_2(x)$ единиц потока в вершине x), послать по соответствующим цепям максимально возможное количество единиц потока, подобрав их в каждой из цепей так, чтобы необходимость накопления в вершине x, вызванная одним из них, удовлетворялась потоком, приходящим по другой[1]). После этого вернуться к началу шага 2.

Если $f_1(x)$ и $f_2(x)$ имеют один и тот же знак (в этом случае уже нельзя гарантировать, что найден поглощающий цикл, который может принять поток, посланный из источника S), снова рассмотреть два различных случая: первый — метка $f_2(x)$ порождена меткой $f_1(x)$ (т. е. при вычислении $f_2(x)$ на определенном этапе использовалась метка $f_1(x)$); второй — метка $f_2(x)$ не порождена меткой $f_1(x)$ (в обоих случаях, не ограничивая общности, можно считать, что $|f_1(x)|<|f_2(x)|$).

Если метка $f_2(x)$ порождена меткой $f_1(x)$ (в этом случае из вершины x был помечен цикл C, включающий x, который в силу $k_C = f_1(x_1)/f_2(x) < 1$, что следует из соотношения $|f_1(x)|<|f_2(x)|$, является поглощающим циклом), направить из вершины x в соответствии поглощающий цикл максимально возможное количество единиц потока. После этого перейти к началу шага 2.

Если метка $f_2(x)$ не порождена меткой $f_1(x)$ (в этом случае были найдены две различные цепи, соединяющие вершины S и x, но не был найден какой-либо поглощающий цикл), следует убрать метку $f_2(x)$ и все метки, ею порожденные, а также снять окраску с соответствующих дуг. Затем следует продолжить процедуру окрашивания и формирования меток.

Шаг 3. (*Изменение двойственных переменных.*) На данном шаге определяются новые значения $p'(x)$ для соответствующих двойственных переменных: при этих значениях выполняются условия дополняющей нежесткости. После вычисления новых значений двойственных переменных, детально описываемого ниже, осуществляется переход к шагу 2 в попытке увеличить количество единиц потока, отправляемых из источника.

Для каждой вершины x следующим образом определить величину $q(x)$:

$$q(x) = \begin{cases} \dfrac{1}{f(x)}, & \text{если } x \in L, \\ 0, & \text{если } x \notin L, \end{cases} \tag{72}$$

[1]) В рассмотренном случае две указанные цепи фактически образуют поглощающий цикл. — *Прим. перев.*

где L — множество вершин, помеченных на последней итерации шага 2[1]).

Далее определить

$$\delta = \min \left\{ \frac{\zeta(x, y)}{[q(x) - k(x, y) q(y)]} \right\}, \tag{73}$$

где минимизация проводится по всем дугам, для которых числитель и знаменатель в приведенном выражении имеют один и тот же знак. Если знаменатель соответствующей дроби для всех дуг равен нулю, то следует закончить выполнение алгоритма: решаемая задача не имеет допустимого решения. В противном случае следующим образом определить для всех вершин x новые значения $p'(x)$ двойственных переменных:

$$p'(x) = p(x) + \delta q(x).$$

Затем возвратиться к шагу 2.

Обоснование алгоритма поиска потока с усилениями. Мы должны показать, что по окончании выполнения рассматриваемого алгоритма формируется либо оптимальное решение, либо такое двойственное решение, которое показывает, что соответствующая задача не имеет допустимого решения.

На шаге 1 рассматриваемого алгоритма задача о потоке минимальной стоимости для исходной сети с усилениями преобразуется в аналогичную задачу для новой сети, в которой искомый поток должен насыщать каждую дугу, инцидентную стоку. Поток минимальной стоимости в новой сети, для которого каждая дуга, инцидентная источнику, является насыщенной, определяет оптимальное решение задачи о потоке минимальной стоимости для исходной сети. Следовательно, мы должны только показать, что (а) алгоритм находит оптимальное решение задачи о потоке минимальной стоимости для новой сети (при этом оптимальный поток насыщает все дуги, инцидентные стоку) или что (б) указанная задача не имеет допустимых решений.

(а) Чтобы показать, что выполнение алгоритма заканчивается определением оптимального решения соответствующей задачи для новой сети, необходимо убедиться в выполнении условий дополняющей нежесткости на протяжении всей процедуры алгоритма, пока не будет насыщена каждая дуга, инцидентная источнику (если насытить указанные дуги не удается, то задача не имеет допустимого решения).

Выполнение алгоритма на шаге 1 начинается с решения, для которого выполняются все условия дополняющей нежесткости.

---

[1]) Множество L включает концевые вершины всех дуг, составляющих ту часть наращиваемого в алгоритме дерева, которая получена на последней итерации шага 2. — *Прим. перев.*

Более того, условия дополняющей нежесткости не нарушаются при всех изменениях потока на шаге 2, поскольку здесь ни в одной дуге не допускается изменение потока, которое может привести к нарушению соответствующего условия дополняющей нежесткости.

Но не нарушаются ли условия дополняющей нежесткости на шаге 3 алгоритма? Рассмотрим любую дугу $(x, y) \in A$. После изменения на шаге 3 вершинных чисел величины $\zeta(x, y)$ меняются на $\zeta'(x, y)$, совпадающие с

$$a(x, y) - p'(x) + k(x, y)\, p'(y) = a(x, y) - p(x) - \delta q(x) +$$

$$+ k(x, y)[p(y) + \delta q(y)] = \zeta(x, y) + \delta[-q(x) + k(x, y)\, q(y)]. \quad (74)$$

Возможны следующие случаи.

Случай 1. Если дуга $(x, y) \in T$, где T — часть наращиваемого в алгоритме дерева, построенная в последней итерации шага 2, то $q(x, y) = k(x, y)\, q(y)$ и $\zeta'(x, y) = \zeta(x, y) + \delta[-q(x) + q(x)] = \zeta(x, y)$. Поскольку изменение на шаге 3 двойственных переменных не вызывает изменения величины $\zeta(x, y)$, то условия дополняющей нежесткости для дуги (x, y) остаются выполненными.

Случай 2. Если дуга $(x, y) \notin T$, $x \notin L$, $y \notin L$, то $q(x) = q(y) = 0$ и $\zeta'(x, y) = \zeta(x, y) + 0\delta = \zeta(x, y)$. Поскольку изменение на шаге 3 двойственных переменных не вызывает изменения величины $\zeta(xc\ y)$, то условия дополняющей нежесткости для дуги (x, y) остаются выполненными. Возможны еще три случая:

$(x, y) \notin T, \quad x \in L, \quad y \notin L$

$(x, y) \notin T, \quad x \notin L, \quad y \in L$

$(x, y) \notin T, \quad x \in L, \quad y \in L$

Мы предоставляем читателю возможность убедиться в том, что в каждом из этих случаев изменение значений двойственных переменных не нарушает выполнимости условий дополняющей нежесткости для дуги (x, y). Читатель может сделать это точно так же, как в рассмотренных выше случаях 1 и 2.

(б) Предположим, что алгоритм завершает свою работу на шаге 3 в результате невозможности определения конечного значения для δ. Мы покажем, что в данном случае не существует допустимого потока.

Дополнительный поток из источника при требовании выполнимости условий дополняющей нежесткости для текущих значений двойственных переменных $p(x)$ может достичь только вершин из множества L. Можно ли изменить значения двойственных переменных так, чтобы ко множеству L добавился хотя бы один

новый элемент и чтобы тем самым увеличилась вероятность пометки стока или окрашивания некоторого поглощающего цикла?

Если не учитывать условий дополняющей нежесткости, то можно указать пять возможных случаев, в которых допустимо окрашивание дуги (x, y):

1. $x \in L$, $y \notin L$, $f(x) > 0$, $f(x, y) < c(x, y)$
2. $x \in L$, $y \notin L$, $f(x) < 0$, $f(x, y) > l(x, y)$
3. $x \notin L$, $y \in L$, $f(y) > 0$, $f(x, y) > l(x, y)$
4. $x \notin L$, $y \in L$, $f(y) < 0$, $f(x, y) < c(x, y)$
5. $x \in L$, $y \in L$, $(x, y) \notin T$ и дуга (x, y)

образует поглощающий цикл с уже окрашенными дугами.

Тщательный анализ каждого из этих случаев показывает, что в них величина $\zeta(x, y)$ и величина $q(x) - k(x, y)q(y)$ должны быть одного знака. Если же δ определить нельзя, то это означает, что не существует неокрашенной дуги (x, y), для которой указанные величины имеют один и тот же знак. Следовательно, в этом случае ни одна из дуг не может быть окрашена и поток из источника не может распространиться за пределы построенного дерева. Итак, в случае невозможности определить величину δ допустимого потока не существует. Это завершает рассмотрение случая (б) и обоснование алгоритма.

Представляя алгоритм поиска потока с усилениями и его обоснование, мы ничего не говорили о количестве шагов, которое включает данный алгоритм. На самом деле алгоритм поиска потока с усилениями в том виде, в каком он описан, не обязательно завершится за конечное число шагов. Далее мы покажем, как модифицировать данный алгоритм, чтобы гарантировать его конечность. Предварительно дадим ряд определений.

Поток с усилениями будем называть *каноническим*, если ни один связный компонент, состоящий из промежуточных для этого потока дуг (т. е. дуг, в которых поток строго больше нижней пропускной способности и строго меньше верхней пропускной способности), не содержит какую-либо из следующих конфигураций:

1) цикл C, у которого $k_C = 1$;
2) два различных, хотя, возможно, и пересекающихся цикла;
3) источник и цикл;
4) сток и цикл;
5) источник и сток.

Отметим, что если связный компонент, состоящий из промежуточных дуг, содержит любую из перечисленных конфигураций, то поток в дугах этого компонента может быть изменен так,

что а) одна или более дуг компонента перестанут быть промежуточными; б) чистый поток из каждой вершины (возможно, исключая источник и сток) останется неизменным.

Пусть, например, промежуточные дуги составляют два цикла, которые пересекаются или соединены цепью из других промежуточных дуг (конфигурация 2). Тогда можно было бы увеличить поток в одном из циклов, а затем возникающие при этом «излишки» потока в некоторых вершинах «поглотить» с помощью другого цикла.

Читателю осталось убедиться в том, что при наличии указанных конфигураций поток всегда можно изменить таким образом, что чистый поток из источника либо останется неизменным, либо увеличится.

Итак, если некоторый поток с усилениями не является каноническим, то соответствующее ему множество промежуточных дуг содержит одну из перечисленных выше конфигураций. Поэтому поток может быть изменен таким образом, что количество промежуточных дуг уменьшится, а чистый поток из источника разве что возрастет. Если полученный после такого изменения поток также не является каноническим, то данная процедура может быть повторена. После последовательных повторений описанной процедуры в конце концов будет построен канонический поток, при этом чистый поток из источника не уменьшится. Таким образом, любой неканонический поток может быть преобразован в канонический поток без уменьшения чистого потока из источника.

ТЕОРЕМА 4.7. Существует лишь конечное число различных канонических потоков в графе G.

Доказательство. Число различных способов выбора множества промежуточных дуг M конечно, поскольку M является подмножеством конечного множества дуг графа G. Потоки в каждой дуге $(x, y) \notin M$ совпадают либо с $l(x, y)$, либо с $c(x, y)$, т. е. в дугах, не принадлежащих M, потоки могут принимать лишь конечное число различных значений.

Покажем, что при фиксированном множестве M и при фиксированных значениях потока в дугах, не принадлежащих M, ни один из потоков не может быть каноническим, если множество всех допустимых в данных условиях потоков является бесконечным. Это докажет теорему.

Значения потоков в некоторых дугах M однозначно определяются условиями сохранения потока. Пусть M' представляет собой подмножество M, в дугах которого значения потока не определяются однозначно условиями сохранения потока. Множество M', если оно не пусто, должно содержать цикл или цепь, соединяющую вершины s и t. (В противном случае значения потоков во всех дугах из M' могли бы быть последовательно

определены, начиная с некоторой вершины, инцидентной лишь одной дуге в M'.)

Если множество M' содержит цепь из s в t, то ни один из возможных потоков не является каноническим. Если множество M' содержит цикл C, у которого $k_C = 1$, то ни один из возможных потоков также не является каноническим. Итак, если множество M' не пусто, то оно должно содержать цикл C, для которого $k_C \neq 1$.

Если возможно существование бесконечного числа различных потоков, то бесконечное число различных потоков должно существовать и в цикле C. Поскольку для этого цикла $k_C \neq 1$, то избыток или недостаток потока, вызываемый потоками, циркулирующими в C, должен компенсироваться либо за счет потока из источника, либо за счет потока в сток, либо за счет потока в некотором другом цикле. Итак, множество M' должно включать конфигурации 2, 3 или 4. Следовательно, ни один из возможных потоков не является каноническим. Теорема доказана.

С учетом представленных выше результатов мы теперь можем формально описать конечную модификацию алгоритма поиска потока с усилениями.

Конечная модификация алгоритма поиска потока с усилениями

Преобразовать начальный поток, сформированный на шаге 1 алгоритмом поиска потока с усилениями, в канонический поток, не уменьшая при этом чистый поток из источника.

При выполнении шага 2 алгоритма поиска потока с усилениями не окрашивать непромежуточную дугу, если вместо нее можно окрасить некоторую промежуточную дугу.

Обоснование. Прежде всего покажем, что если выполнение шага 2 начинается с канонического потока, то получающийся в результате проводимых на данном шаге изменений поток также является каноническим. Для этого предположим противное: пусть получаемый на шаге 2 поток не является каноническим. Тогда множество промежуточных дуг для получаемого потока должно содержать по крайней мере одну из пяти перечисленных выше конфигураций. Обозначим множество (промежуточных) дуг, составляющих некоторую из указанных конфигураций, через J. Пусть J' — та часть множества J, в которую входят дуги, являющиеся промежуточными и для исходного потока. Пусть J'' — оставшаяся часть множества J, в которую входят дуги, не являющиеся промежуточными для исходного потока. При изменении потока на шаге 2 дуги J'' должны были быть окрашены. Отсюда следует, что при использовании конечной модификации

алгоритма поиска потока с усилениями ни одна дуга из J' не может быть окрашена в процессе окрашивания дуг из J''. Однако тщательный анализ каждой из пяти возможных конфигураций показал бы, что недопустимость окраски дуг из J' невозможна: это противоречит правилам выполнения конечной модификации алгоритма поиска потока с усилениями.

Итак, на шаге 2 при исходном каноническом потоке могут порождаться также только канонические потоки. Поскольку в каждом следующем потоке, формируемом на шаге 2 алгоритма, чистый поток из источника увеличивается, на шаге 2 никакой канонический поток не будет порождаться дважды. Поскольку в силу теоремы 4.7 существует лишь конечное число различных канонических потоков, шаг 2 включает конечное число итераций. Итак, единственная «возможность» для алгоритма не закончиться за конечное число шагов состоит в том, что количество итераций на шаге 3 не ограничено.

Поскольку в алгоритме поток может изменяться лишь конечное число раз, неограниченное число итераций на шаге 3 может возникнуть только между двумя последовательными изменениями потока. Однако после каждой итерации шага 3 либо окрашивается по крайней мере одна новая дуга, либо оказывается неопределяемым параметр δ, что означает отсутствие допустимого потока. Поскольку в графе имеется лишь конечное множество дуг, шаг 3 для одного и того же потока может выполняться только конечное число раз. Следовательно, общая процедура алгоритма состоит из конечного числа шагов.

Некоторые разновидности потоков с усилениями

До сих пор мы рассматривали лишь задачу поиска потока минимальной стоимости, в котором из источника отправляется заданное количество единиц потока. Предположим теперь, что вместо указанной задачи нас интересует задача поиска потока минимальной стоимости, которым в сток доставляется заданное количество единиц потока. Можно ли использовать алгоритм поиска потока с усилениями для решения данной задачи? Да, можно. Это осуществляется с помощью перехода к обратному графу. Напомним, что обратный по отношению к G граф G^{-1} определялся как граф с множеством вершин X и множеством дуг

$$A^{-1} = \{(y, x) : (x, y) \in A\}.$$

Пусть стоимость дуги $(y, x) \in A^{-1}$ равна $a(x, y)/k(x, y)$. Пусть усиление дуги $(y, x) \in A^{-1}$ равно $1/k(x, y)$. Пусть, наконец, $l(y, x) = l(x, y)/|k(x, y)|$ и $c(y, x) = c(x, y)/|k(x, y)|$. Заметим, что обратный по отношению к G^{-1} граф совпадает с исходным гра-

фом G. Пусть далее $b(x)$ обозначает общее количество единиц потока, запасаемых в вершине x.

Для заданного в графе G потока $f(x, y)$ мы можем следующим образом в графе G^{-1} определить обратный поток $f^{-1}(y, x)$, $((y, x) \in \in A^{-1})$:

если $k(y, x) > 0$, то $f^{-1}(y, x) = f(x, y)k(x, y)$;

если $k(y, x) < 0$, то $f^{-1}(y, x) = [c(x, y) - f(x, y)][-k(x, y)]$. (Поясним, что $c(x, y)$ единиц может попадать в вершину x и $k(x, y)$ $c(x, y)$ единиц — в вершину y.)

Почему потребовалось вводить для обратного потока специальное определение? Дело в том, что если для потока в графе G $f(x, y)$ единиц потока направлены из вершины x по дуге (x, y), то $k(x, y)$ $f(x, y)$ единиц потока попадает по данной дуге в вершину y. Для обратного же потока в графе G^{-1} если $f(x, y)$ единиц потока попадает в вершину x по дуге (y, x), то $k(x, y)$ $f(x, y)$ единиц потока должно быть отправлено по той же самой дуге из вершины y (убедитесь в этом, используя определение обратного потока). Таким образом, обратный поток прямо противоположен исходному потоку. При этом источник s в графе G становится стоком в графе G^{-1}, а сток t в графе G становится источником в графе G^{-1}.

Предположим, что некоторый поток $\{f(x, y)\}$ в графе G удовлетворяет всем ограничениям на пропускную способность дуг и дает запас единиц потока в вершине x в количестве $b(x)$. Тогда соответствующий поток $\{f^{-1}(y, x)\}$ в графе G удовлетворяет всем ограничениям на пропускную способность дуг и дает запас единиц потока в вершине x в количестве $[-b(x)]$. Более того, оба эти потока имеют одинаковую общую стоимость.

Итак, поток минимальной стоимости в графе G^{-1}, в котором из источника выходит V единиц, соответствует потоку минимальной стоимости в графе G, в котором в сток приходит V единиц. Таким образом, задача поиска в сети с усилениями потока минимальной стоимости, в котором в сток прибывает заданное количество единиц потока, может быть решена с помощью алгоритма поиска потока с усилениями, который применен к графу, являющемуся обратным по отношению к исходному графу (при этом в обратном графе из источника отправляется то же самое количество единиц потока, которое прибывает в сток в исходном графе).

Когда алгоритм поиска потока с усилениями используется для решения соответствующей задачи, невозможно предсказать, какое количество единиц потока из V единиц, отправленных из источника, прибудет в сток (V единиц потока могут либо все «дойти» до стока, либо частично поглотиться при прохождении сети). Существует ли какой-нибудь способ обеспечения того, чтобы алгоритм в сети с усилениями формировал поток минимальной стоимости, в котором в сток прибывает по крайней мере

W единиц потока? Да, существует. Достаточно лишь дополнить исходную сеть новой вершиной T и дугой (t, T), соединяющей сток t исходной сети с вершиной T. Пусть $l(t, T) = W$, $c(t, T) = \infty$, $k(t, T) = 1$ и $a(t, T) = 0$, и пусть вершина T становится стоком в дополненной сети. Каждый допустимый поток в дополненной сети соответствует потоку в исходной сети, в котором в сток прибывает не менее чем W единиц потока. Конечно, требование того, чтобы в сток попадало не менее W единиц потока, вполне возможно, приведет к увеличению общей стоимости потока.

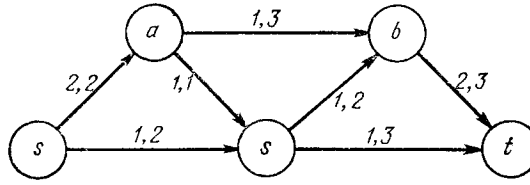
Предположим, что не выдвигается никаких требований в отношении того, какое количество единиц потока должно прибыть в сток. Возможно, существует не один поток минимальной стоимости, в котором из источника выходит V единиц потока. Можно ли найти поток минимальной стоимости, в котором из источника выходит V единиц потока и при этом в сток попадает максимально возможное количество единиц потока? Да, можно. Такой поток генерируется в исходной сети, дополненной, как и выше, вершиной T и дугой (t, T). Пусть для добавленной дуги $l(t, T)=0$, $c(t, T)=\infty$, $k(t, T) = 1$ и $a(t, T) = \varepsilon$, где ε — достаточно небольшое по модулю отрицательное число. Когда ε достаточно мало по модулю, то поток максимальной стоимости в дополненной сети, генерируемый алгоритмом поиска потока с усилениями, соответствует потоку минимальной стоимости в исходной сети. Если существует не один поток минимальной стоимости, то указанный алгоритм, будучи применен к дополненной сети, определит такой поток минимальной стоимости, в котором в сток попадает максимально возможное количество единиц (поскольку с каждой дополнительной единицей потока общая стоимость уменьшается на величину $|\varepsilon|$, что связано с прохождением этой единицы по дуге (t, T)).

Обратно, если мы хотим минимизировать количество единиц потока, прибывающих в сток, то следует положить ε равным небольшой по модулю положительной величине.

Специальный случай рассмотренной в данном разделе задачи, в которой стоимости прохождения для всех дуг равны 0, исследован у Гринолда [3].

УПРАЖНЕНИЯ

1. Составьте список всех увеличивающих поток цепей из s в t в следующем графе (рядом с дугами указаны текущие значения потока и пропускные способности).



2. В графе из упражнения 1, исходя из текущего потока, постройте максимальный поток из s в t. Найдите минимальный разрез, который разделяет вершины s и t. Насыщает ли максимальный поток все дуги этого разреза?

3. Сформулируйте условия, при которых в произвольном графе существует увеличивающая поток цепь из источника в сток, целиком состоящая из обратных или прямых дуг.

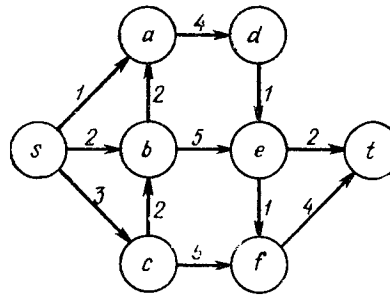
4. Агент по снабжению должен решить задачу максимизации объема перевозок от склада до различных объектов розничной торговли. После изучения гл. 4 он может сформулировать данную задачу как задачу поиска максимального потока между вершиной, представляющей склад, и вершинами, представляющими соответствующие объекты. Одна из промежуточных вершин в графе рассматриваемой задачи представляет город, расположенный на пересечении трех магистральных дорог. С целью уменьшения уровня загрязнения окружающей среды в данном городе был проведен закон, ограничивающий тоннаж грузов, перевозимых через город. Каким образом следует учесть это дополнительное ограничение в исходном графе, чтобы можно было применять к нему алгоритм поиска максимального потока?

5. Если алгоритм поиска максимального потока выявляет увеличивающую поток цепь, которая включает некоторую обратную дугу, то определенные единицы потока будут «сняты» с этой дуги и переправлены в сток по новому маршруту, не включающему данную дугу.

а) Возможно ли, чтобы на протяжении всего времени работы алгоритма поиска потока максимальной стоимости не происходило изменения маршрута движения ни одной единицы потока?

б) При каких условиях можно быть уверенным, что для единиц потока, «назначенных» к прохождению по определенной дуге, на последующих итерациях алгоритма поиска максимального потока не будут изменены маршруты движения?

6. Постройте поток минимальной стоимости из вершины s в вершину t в приводимом ниже графе, используя а) алгоритм поиска потока минимальной стоимости, б) алгоритм дефекта. (В приведенном графе верхние пропускные способности всех дуг равны 1, а нижние — 0. Стоимости прохождения указаны рядом с соответствующими дугами.)



7. Предположим, что мы уже вычислили поток минимальной стоимости для достаточно большого графа. Анализируя полученные результаты, мы обнаружили, что

1) некоторая дуга графа была пропущена;

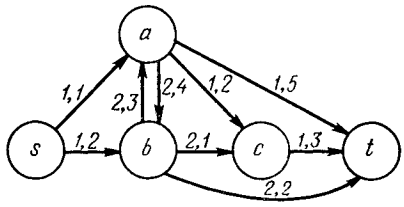
2) пропускная способность некоторой дуги была завышена на 5 единиц;

3) стоимость некоторой дуги была занижена на 2 единицы. Ответьте на следующие вопросы:

а) Можно ли определить, какая из указанных ошибок влияет на оптимальное решение?

б) Можно ли для каждой из ошибок скорректировать полученный результат, не проводя заново всех вычислений? Можно ли то же самое сделать в случае, когда были допущены сразу все указанные ошибки?

8. Постройте максимальный динамический поток за период в 10 интервалов времени в приводимом ниже графе. Числа, стоящие рядом с дугами этого графа, определяют для них соответственно пропускную способность и время прохождения.



9. Предположим, что время прохождения дуги (a, c) в графе из упражнения 8 в момент времени x меняется с двух единиц до одной единицы. При каких значениях x данное изменение оставит прежними результаты выполнения упражнения 8?

10. Докажите, что отдельные единицы динамического потока не могут перемещаться в «обратном времени». Может ли развернутый во времени вариант исходного графа содержать циклы?

11. Обобщите алгоритм поиска максимального динамического потока на случай, когда время прохождения дуги не является целым числом.

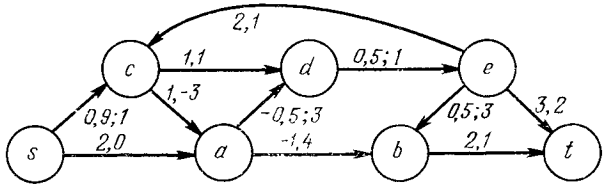
12. При каких условиях алгоритм поиска максимального динамического потока генерирует поток наискорейшего прибытия?

13. При каких условиях количество единиц потока, прибывающих в сток в каждый момент времени, постоянно?

14. Постройте контрпример, опровергающий следующее утверждение: теорема 4.4 из разд. 4.5 справедлива для немаксимальных потоков F' и F''.

15. Для графа, изображенного на рис. 4.13, постройте четыре лексикографических потока, указанных в лемме 4.5.

16. Перечислите все поглощающие и генерирующие циклы в приводимом ниже графе. В этом графе нижние пропускные способности всех дуг равны 0, а верхние — 1. Коэффициенты усиления и стоимости приведены рядом с соответствующими дугами.



17. Примените алгоритм поиска потока с усилениями для определения наилучшего с точки зрения затрат способа, обеспечивающего вытекание 5 единиц потока из источника s, представленного на графе из упражнения 16.

18. Каким образом можно использовать алгоритм поиска потока с усилениями для выявления цикла, по которому можно переправлять единицы потока в сток?

19. Проведите детальное доказательство того, что условия дополняющей нежесткости сохраняются при изменении вершинных чисел на шаге 3 алгоритма поиска потока с усилениями.

20. В данной главе было рассмотрено семь алгоритмов:

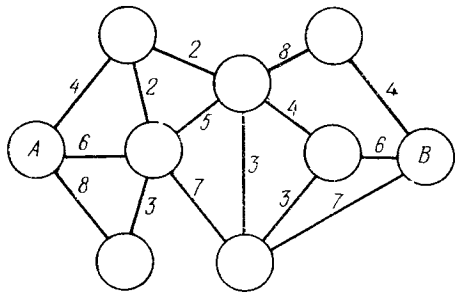
1) алгоритм поиска увеличивающей поток цепи;
2) алгоритм поиска максимального потока;
3) алгоритм поиска потока минимальной стоимости;
4) алгоритм дефекта;
5) алгоритм поиска максимального динамического потока;
6) алгоритм поиска потока наискорейшего прибытия;
7) алгоритм поиска потока с усилениями.

а) Какие из этих алгоритмов взаимозаменяемы?

б) Какие из алгоритмов представленного списка являются частными случаями других алгоритмов того же списка?

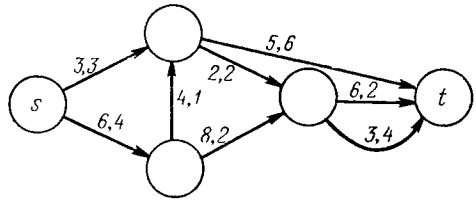
в) Какие из алгоритмов представленного списка являются подалгоритмами других алгоритмов того же списка?

21. Большое число людей путешествует на автомобилях из города A в Мексике в город B в США. Возможные маршруты, связывающие A и B, представлены на приведенной ниже схеме. Служба пограничного контроля должна иметь достаточное количество контрольных пунктов, размещенных таким образом, чтобы ни один автомобиль не смог проехать из A в B, не встречая на маршруте движения хотя бы одного такого пункта. Стоимость сооружения контрольного пункта зависит от места его размещения. Стоимости размещения пунктов для различных участков маршрутов указаны на приведенной схеме. Найдите наименее дорогостоящее размещение контрольных пунктов.



22. Программа обучения в области управления может быть построена различными способами, как показано в приведенной ниже схеме. Каждая дуга этой схемы (графа) представляет определенную подпрограмму обучения. Каждый студент начинает обучение в s и проходит курс по любому пути, заканчивающемуся в t. Количество студентов, которые могут обучаться по отдельной программе, ограничено. Хотя каждая такая частная программа имеет продолжительность в один месяц, стоимость обучения для них различна. На схеме у соответствующих дуг указаны минимальное количество студентов в формируемых группах и стоимости обучения для одного студента. Каково максимальное количество студентов, которые могут пройти

обучение по полной программе в течение пяти месяцев при затратах на обучение, не превышающих для каждого студента 1000 долл.[3]



ЛИТЕРАТУРА

1. Edmonds J., Karp R. M., Theoretical Improvements in Algorithm Efficiency for Network Flow Problems, J. ACM, vol. 19, N. 2, pp. 218 — 264, 1972.
2. Ford L. R., Fulkerson D. R., Flows in Networks, Princeton Press, Princeton, 1962. [Имеется перевод: Форд Л. Р., Фалкерсон Д. Р., Потоки в сетях.— М.: Мир, 1966].
3. Grinold R. C. Calculating Maximal Flows in a Network with Positive Gains, Operations Research, 21, pp. 528 — 541, 1973.
4. Jewell W. S., Optimal Flow through a Network with Gains, Operations Research, 10, pp. 476 — 499, 1962.
5. Johnson E. L., Networks and Basic Solutions, Operations Research, 14, pp. 619 — 623, 1966.
6. Maurras J. F., Optimization of the Flow through Networks with Gains, Math. Programming, 3, pp. 135 — 144, 1972.
7. Minieka E. T., Maximum, Lexicographic and Dynamic Network Flows, Operations Research, 21, pp. 517 — 527, 1973.
8. Minieka E. T., Optimal Flow in a Network with Gains, INFOR, 10, pp. 171 — 178, 1972.
9. Truemper K., Optimum Flow in Networks with Positive Gains, Ph. D. Thssis, Case-Western Reserve University, 1973.
10. Truemper K., An Efficient Scaling Procedure for Gains Networks, Networks, 6, pp. 151 — 160, 1976.
11. Wilkinson W. L., An Algorithm for Universal Maximal Dynamic Flows in a Network, ORSA, 19, pp. 1602 — 1612, 1971.

ГЛАВА 5

Алгоритмы поиска паросочетаний и покрытий

5.1. Введение

Паросочетанием графа называется некоторое множество его дуг, такое, что каждая вершина графа инцидентна не более чем одной дуге этого множества.

Покрытием графа называется некоторое множество дуг в графе, такое, что каждая его вершина инцидентна по крайней мере одной дуге этого множества. Например, на рис. 5.1 каждое из множеств дуг $\{\alpha, \gamma\}$, $\{\beta, \varepsilon\}$, $\{\delta, \beta\}$, $\{\alpha\}$, $\{\beta\}$ образует паросочетание графа, а каждое из множеств $\{\alpha, \varepsilon, \gamma\}$, $\{\alpha, \beta, \delta, \varepsilon\}$, $\{\beta, \delta, \varepsilon\}$ — покрытие графа.

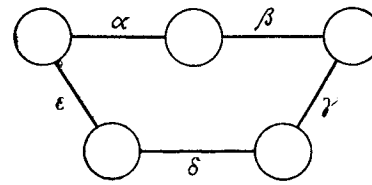


Рис. 5.1.

Очевидно, что любое подмножество паросочетаний графа также является его паросочетанием, а любое множество дуг, включающее в качестве подмножества покрытие графа, является покрытием этого графа.

С отысканием паросочетаний и покрытий часто приходится сталкиваться на практике.

ПРИМЕР 1. Во время второй мировой войны многие летчики из других стран бежали в Англию для того, чтобы поступить на службу в Королевские военно-воздушные силы (КВВС). Для выполнения полета самолета КВВС требуется наличие двух пилотов, у которых умение пилотирования самолета сочетается со знанием языка. КВВС заинтересованы в том, чтобы в воздухе одновременно находилось максимальное количество самолетов.

Для решения этой задачи построим граф, каждая из вершин которого отображает пилота КВВС. Соединим дугой две любые вершины графа, соответствующие пилотам, которые могут летать совместно. Любое паросочетание этого графа представляет допустимое множество самолетов, которые одновременно могут находиться в воздухе. Таким образом, КВВС были заинтересованы в отыскании такого паросочетания этого графа, которое содержит наибольшее количество дуг. Иначе говоря, требуется найти *максимальное по мощности паросочетание графа.*

ПРИМЕР 2. Агентство по продаже недвижимого имущества имеет для продажи целый ряд домов и некоторое количество потенциальных покупателей. Каждый такой покупатель может проявлять интерес к более чем одному из домов. Агент по продаже недвижимого имущества может достаточно точно

оценить, сколько каждый покупатель заплатит за каждый из представляющих для него интерес дом.

Поскольку агент по продаже недвижимого имущества получает 7% комиссионных отчислений от каждой сделки, он заинтересован в максимизации общего объема совершенных им продаж в долларах. Возникает вопрос: каким образом может быть достигнут этот максимум? Пусть каждый покупатель и каждый дом представляются отдельными вершинами графа; соединим две соответствующие вершины дугой в случае, когда конкретный покупатель желает приобрести определенный дом. Каждая дуга при этом представляет возможную сделку. Присвоим каждой дуге графа вес, равный размеру комиссионных отчислений, которые агент по продаже недвижимого имущества должен получить от реализации соответствующей сделки.

Агент может максимизировать свои заработки при реализации сделок, соответствующих паросочетанию графа с наибольшим общим весом. Иными словами, перед ним стоит задача поиска *паросочетания с максимальным весом.*

ПРИМЕР 3. Некоторый комитет должен быть сформирован таким образом, чтобы в него входили по крайней мере по одному представителю от каждого из 50 штатов и по крайней мере по одному представителю каждой из 65 имеющихся в США основных этнических групп. Свои услуги для участия в работе комитета в общенациональном масштабе предложили 170 человек. Необходимо определить состав комитета, включающего наименьшее число претендентов и удовлетворяющего всем отмеченным выше требованиям.

Построим граф, в котором каждый штат представляется вершиной и каждая этническая группа также представляется вершиной. Таким образом, граф будет иметь $50 + 65 = 115$ вершин. Пусть каждый человек, изъявивший желание участвовать в работе комитета, представляется дугой графа, соединяющей вершины, соответствующие штату, в котором он проживает, и этнической группе, к которой он принадлежит. Любой комитет, удовлетворяющий всем географическим и этническим требованиям, соответствует покрытию этого графа. Комитет с наименьшим количеством членов соответствует покрытию графа с наименьшим количеством дуг, иными словами — *покрытию минимальной мощности.*

ПРИМЕР 4. Служба знакомств создает возможность встречи каждому обратившемуся к ней лицу по крайней мере с одним подходящим для него претендентом на знакомство. Размер затрат службы на каждую встречу различен в зависимости от конкретных мероприятий, требуемых для ее организации (время и место встречи, система предпочтений встречающихся и т. д.).

Каким образом служба знакомств может с минимальными издержками выполнить все обязательства перед своими клиентами?

Построим граф, в котором каждому клиенту соответствует вершина и каждой совместимой паре соответствует дуга. Каждой дуге припишем вес, равный издержкам на соответствующую встречу.

Любое покрытие этого графа представляет собой способ организации по крайней мере одной подходящей встречи для каждого клиента службы знакомств.

Служба должна найти покрытие графа с наименьшими общими издержками[1], другими словами — *покрытие с минимальным весом.*

В примерах 1—4 проиллюстрированы четыре типа широко распространенных задач о паросочетаниях и покрытиях:

1. Задача о покрытии максимальной мощности.

[1]) Общие издержки равны сумме издержек на каждую из встреч. — *Прим. перев.*

2. Задача о паросочетании с максимальным весом.
3. Задача о покрытии минимальной мощности.
4. Задача о покрытии с минимальным весом.

Однако имеются и другие задачи о паросочетании и покрытиях; к ним, в частности, относятся следующие.

Задача о паросочетаниях минимальной мощности. Очевидно, что паросочетание, имеющее мощность, равную нулю (т. е. не содержащее ни одной дуги), является паросочетанием минимальной мощности.

Задача о паросочетаниях с минимальным весом. Если все дуги в графе имеют неотрицательные веса, то паросочетание нулевой мощности является и паросочетанием с минимальным весом. Если же веса некоторых дуг отрицательны, то задача о паросочетаниях с минимальным весом может быть решена следующим образом:

а) исключаются дуги, имеющие неотрицательный вес,

б) изменяются на противоположные знаки значений веса оставшихся дуг и

в) находится паросочетание с максимальным весом.

Очевидно, что дуги, входящие в это паросочетание, представляют собой паросочетание с минимальным весом, соответствующее исходному графу.

Таким образом, задача о паросочетаниях с минимальным весом эквивалентна задаче о паросочетаниях с максимальным весом.

Задача о покрытии максимальной мощности. Очевидно, что в тех случаях, когда в графе отсутствуют изолированные вершины, множество всех вершин графа представляет собой покрытие максимальной мощности.

Задача о покрытии с максимальным весом. Ясно, что покрытие с максимальным весом должно включать все дуги, имеющие положительный вес. Если все дуги с положительными значениями веса не составляют покрытия, то в покрытие с максимальным весом должны быть включены некоторые дуги с неположительными значениями веса.

Эти дуги должны покрывать оставшиеся непокрытые вершины. Они могут быть выбраны в результате отыскания покрытия с минимальным весом для оставшихся непокрытых вершин после изменения знаков значений весов на противоположные (т. е. путем присвоения рассматриваемым дугам неотрицательных значений веса).

Таким образом, решение задачи о покрытии с минимальным весом определяет и решение задачи о покрытии с максимальным весом.

Кстати, решение задачи о паросочетании максимальной мощности дает решение задачи о покрытии минимальной мощности,

и наоборот, задача о покрытии минимальной мощности дает решение задачи о паросочетании максимальной мощности. То есть если паросочетание максимальной мощности M^* известно, то с его помощью легко может быть получено покрытие минимальной мощности C^*, и наоборот, если известно покрытие минимальной мощности C^*, то может быть легко получено паросочетание максимальной мощности M^*. Каким образом это можно сделать?

Построение 1. (*От паросочетания к покрытию.*) Пусть M — произвольное паросочетание. Выберем произвольную вершину v, не инцидентную ни одной из дуг, входящих в M. Присоединим к паросочетанию любую дугу, инцидентную v. Будем повторять эту процедуру до тех пор, пока не будут рассмотрены все вершины v, не инцидентные ни одной из дуг, входящих в M. Полученное в результате множество дуг C' является некоторым покрытием графа.

Построение 2. (*От покрытия к паросочетанию.*) Пусть C — произвольное покрытие. Пусть v — произвольная вершина, инцидентная более чем одной дуге, входящей в C. Исключим из C любую дугу, инцидентную v. Будем повторять эту процедуру до тех пор, пока не останется ни одной вершины, инцидентной более чем одной дуге. Получаемое в результате множество дуг M' есть паросочетание.

ТЕОРЕМА 5.1. Если M — паросочетание максимальной мощности, тогда C' — покрытие минимальной мощности. Если C — покрытие минимальной мощности, тогда M' — паросочетание максимальной мощности.

Доказательство. Поскольку M — паросочетание максимальной мощности,

$$|C'| = |M| + (|X| - 2|M|) = |X| - |M|,$$

где через $|X|$ обозначается число вершин в графе.

Так как C — покрытие минимальной мощности, то

$$|M'| = |C| - |2|C| - |X|) = |X| - |C|.$$

Таким образом,

$$|M| + |C'| = |X| \tag{1}$$

и

$$|C| + |M'| = |X|. \tag{2}$$

Предположим, что C' не является покрытием минимальной мощности. Тогда существует полученное из C' с помощью построения 2 паросочетание, имеющее мощность, большую чем мощность M, определяемую из уравнения (1). Это, однако, противоречит пред-

положению, что M — паросочетание максимальной мощности. Кроме того, если M' не является паросочетанием максимальной мощности, то существует полученное из M' с помощью построения 1 покрытие, которое имеет мощность, меньшую чем мощность C, определяемую из (2). Но это противоречит предположению о том, что C есть покрытие минимальной мощности. Следовательно, теорема доказана.

Таким образом, задачи о паросочетании максимальной мощности и о покрытии минимальной мощности эквивалентны друг другу. К сожалению, между задачами о паросочетании с максимальным весом и о покрытии с минимальным весом аналогичных отношений, по-видимому, не существует.

Ниже в разд. 5.2 рассматривается алгоритм решения задачи о паросочетании максимальной мощности, а в разд. 5.3 — алгоритм решения задачи о паросочетании с максимальным весом. В разд. 5.4 приводится алгоритм решения задачи о покрытии с минимальным весом.

5.2. Алгоритм решения задачи о паросочетании максимальной мощности

Граф, множество вершин X которого допускает разбиение на два подмножества X' и X'' так, что ни одна из дуг графа не соединяет вершины одного и того же подмножества, называется двудольным. Таким образом, в двудольном графе одна из вершин, соединяемых каждой из дуг, содержится в X', а другая вершина — в X''.

Например, на рис. 5.2 представлен двудольный граф, для которого $X' = \{a, b, c\}$ и $X'' = \{d, e, f\}$. Заметим, что у каждой дуги одна из соединяемых ею вершин находится в X', а другая — в X''.

В двудольном графе каждый цикл должен содержать четное число дуг (четный цикл). Это следует из того, что вершины каждой дуги находятся в различных подмножествах, а цикл должен замкнуться в том подмножестве в котором лежит его начало.

Если граф $G = (X, E)$ является двудольным, то задача о паросочетании максимальной мощности легко может быть решена путем сведения к задаче о максимальном потоке, рассмотренной в разд. 4.2.

Для этого необходимо:

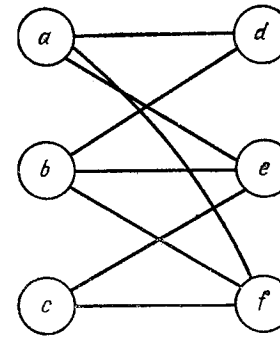


Рис. 5.2. Двудольный граф.

1) ориентировать все ребра в направлении от X' к X'';

2) ввести вершину — источник s и дуги (s, x), ориентированные от источника s к каждой вершине $x \in X'$;

3) ввести вершину — сток t и дуги (x, t), ориентированные от каждой вершины $x \in X''$ к стоку t;

4) пропускную способность каждой из дуг исходного графа принять равной 1 (рис. 5.3).

Обозначим построенный граф через G'. В дугах всех цепей, по которым происходит увеличение потока из s, будет протекать

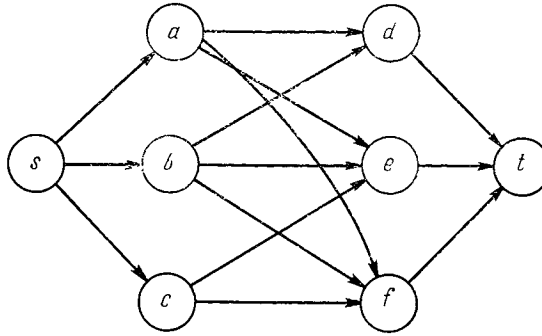


Рис. 5.3. Сеть, в которой ищется максимальный поток (граф G').

поток, равный единице, если решение задачи о максимальном потоке в графе G' начинается с нулевого потока.

Поток в дугах этих цепей равен единице вследствие равенства единице пропускных способностей всех дуг графа G'. Таким образом, решение задачи о максимальном потоке будет таким, что в каждой дуге графа G' будет протекать поток, равный либо нулю, либо единице. Дуги, исходящие из вершин подмножества X' к вершинам подмножества X'' в графе G', по которым протекает единичный поток, соответствуют паросочетанию в графе G. Более того, каждому паросочетанию в G может быть поставлен в соответствие такой поток в G', что ребрам паросочетания соответствуют дуги графа G', по которым протекает поток, равный единице. Следовательно, паросочетание в G, соответствующее максимальному потоку в G', должно быть паросочетанием максимальной мощности в G. В противном случае если бы это паросочетание не имело максимальную мощность, то должен был бы найтись в G' еще больший поток, соответствующий паросочетанию максимальной мощности, что, однако, противоречит принятым предположениям. Итак, для двудольного графа задача поиска паросочетания максимальной мощности может быть сведена к задаче о максимальном потоке.

Если граф G не является двудольным, то G должен содержать цикл с нечетным числом ребер (нечетный цикл). В противном случае множество вершин графа G, как описано выше, могло бы быть разбито на подмножества X' и X''.

Наличие нечетных циклов усложняет дело, поскольку в этом случае отсутствует очевидный способ сведения задачи о паросочетании к потоковой задаче. Оценим степень трудности решения задачи поиска паросочетания максимальной мощности при постановке ее как задачи линейного программирования. Рассмотрим для этого следующую задачу линейного программирования:

максимизировать

$$\sum_{(i,\ j)} x(i,\ j) \tag{3}$$

при ограничениях

$$\sum_{j} [x(j,\ i) + x(i,\ j)] \leqslant 1 \text{ (для всех вершин } i), \tag{4}$$

$$0 \leqslant x(i,\ j) \leqslant 1 \text{ (для всех ребер } (i,\ j)), \tag{5}$$

где $x(i, j)$ определяет количество включений ребра (i, j) в паросочетание. Каждое ограничение (4) требует, чтобы число входящих в паросочетание ребер, инцидентных вершине i, не превышало единицы[1]).

Каждое ограничение (5) требует, чтобы каждое ребро (i, j) не входило в паросочетание более одного раза.

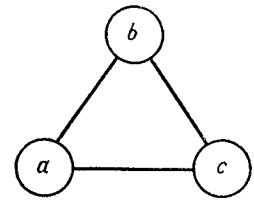


Рис. 5.4.

Ясно, что каждое паросочетание удовлетворяет ограничениям (4) и (5), если положить $x(i, j) = 1$ для (i, j), входящих в него. Однако ограничениям (4) и (5) могут удовлетворять и нецелочисленные значения переменных $x(i, j)$. Например, рассмотрим граф, представленный на рис. 5.4. Решение $x(a, b) = x(b, c) = x(c, a) = {}^1/_2$ удовлетворяет ограничениям (4) и (5). Значение целевой функции (3) для этого решения равно ${}^1/_2 + {}^1/_2 + {}^1/_2 = 1\ {}^1/_2$. При более детальном изучении оказывается, что наибольшее значение целевой функции для решения, соответствующего паросочетанию, равно 1. Например, для паросочетания $x(a, b) = 1$, $x(b, c) = 0$, $x(c, a) = 0$ значение целевой функции (3) равно $1 + 0 + 0 = 1$. Следовательно, мы никогда не можем быть уверены, что оптимальное решение задачи линейного про-

[1]) Напомним, что ребро, соединяющее вершины i и j, обозначается как (i, j) или (j, i).

граммирования (3) — (5) будет соответствовать паросочетанию. (В разд. 5.3 в задачу линейного программирования, описываемую соотношениями (3) — (5), будут введены такие дополнительные ограничения, при которых ее оптимальное решение всегда будет паросочетанием. Однако чрезмерно большое количество дополнительных ограничений приведет к значительному росту размерности задачи линейного программирования, что может существенно снизить эффективность методом ее решения.) Увы, для решения задачи о паросочетании максимальной мощности на графах, не являющихся двудольными, не могут быть использованы ни потоковые алгоритмы, ни методы линейного программирования. Поэтому необходимо рассмотреть алгоритм Эдмондса [5], разработанный специально для решения задачи о *паросочетании максимальной мощности*.

Если задано паросочетание M в графе G, то *чередующейся цепью* называется простая цепь, в которой из любых двух смежных ребер одно принадлежит, а другое не принадлежит паросочетанию M. Например, для паросочетания, приведенного на рис. 5.5, *а*, цепь (a, b), (b, c), (c, f) является чередующейся, так как первое и третье ребра (a, b) и (c, f) не входят в M, а второе ребро (b, c) принадлежит M.

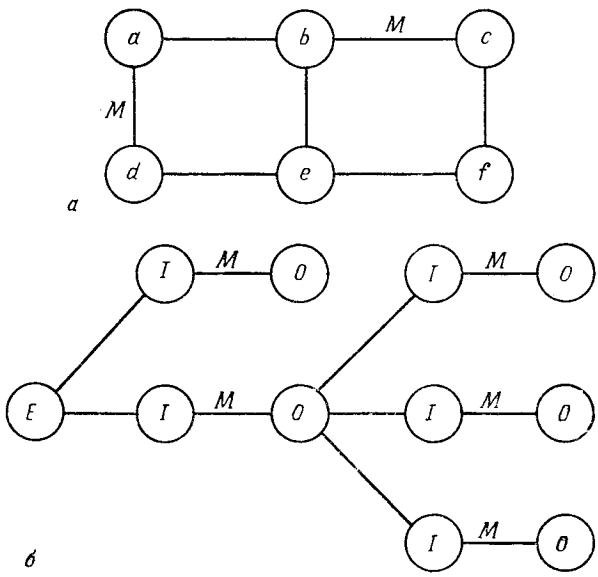


Рис. 5.5. Пример (*а*) увеличивающих цепей и (*б*) чередующегося дерева. E — открытая вершина, I — внутренняя вершина чередующегося дерева, O — внешняя вершина чередующегося дерева, M — ребро, входящее в паросочетание.

Цепь (d, a), (a, b), (b, c), (c, f) также является чередующейся, так как ее первое и третье ребра принадлежат M, а второе и четвертое не принадлежат M.

Вершина называется открытой, если она не инцидентна ни одному из ребер, входящих в это паросочетание.

Вершина называется вершиной *паросочетания*, если она инцидентна некоторому входящему в паросочетание ребру.

Чередующаяся цепь называется *увеличивающей*, если ее первая и последняя вершины являются открытыми. Например, на рис. 5.5, *а* цепь (e, b), (b, c), (c, f) является увеличивающей. Если ребра увеличивающей цепи, не принадлежащие паросочетанию M, включить в него, а ее ребра, принадлежащие M, исключить из него, то полученное в результате паросочетание будет содержать на одно ребро больше, чем в исходном паросочетании. Следовательно, если паросочетание содержит увеличивающую цепь, то оно не может быть паросочетанием максимальной мощности. Более того, верно и обратное утверждение.

ТЕОРЕМА 5.2. Если паросочетание M не является паросочетанием максимальной мощности, то для него может быть найдена увеличивающая цепь.

Доказательство. Пусть M^* — паросочетание максимальной мощности, которое содержит наибольшее количество общих с паросочетанием M ребер. Пусть через G' обозначен подграф, множество ребер которого состоит из всех ребер, входящих только в одно из двух паросочетаний M и M^*. Ни одна из вершин G' не может иметь более двух инцидентных ей ребер. В противном случае в одном из паросочетаний одной вершине будут инцидентны два ребра, что невозможно по определению паросочетания. По этой причине каждый связный компонент G' должен быть либо простой цепью, либо простым циклом.

При этом любой связный компонент G' не может быть нечетным циклом. В противном случае одной вершине в этом нечетном цикле были бы инцидентны две дуги одного и того же паросочетания, что невозможно. Более того, связный компонент G' не может быть и четным циклом. Если связный компонент G' был бы четным циклом, то каждое ребро в нем, принадлежащее M^*, может стать принадлежащим M, и наоборот. Это новое паросочетание содержало бы такое же количество ребер, как и M^*, но имело бы большее чем M^* количество ребер, общих с M, что невозможно в силу предположения о свойстве M^*. Следовательно в графе G' ни один из компонентов связности не может быть циклом.

Таким образом, каждый связный компонент графа G' является простой цепью. Рассмотрим первое и последнее ребра в этой цепи. Если оба эти ребра принадлежат M, то цепь является увеличивающей цепью для M^*, но это противоречит

предположению о том, что M^* — паросочетание максимальной мощности. Если одно из этих ребер принадлежит M, а другое — M^*, то можно изменить принадлежность ребер этой цепи: те, которые ранее принадлежали M, будут принадлежать M^*, и наоборот. В результате получается новое паросочетание, имеющее мощность, равную мощности M^*, и число ребер, общих с M, большее чем в M^*. Но это противоречит введенным предположению о том, что M^* есть паросочетание с наибольшим числом ребер, общих с M. Следовательно, и первое и последнее ребра этой цепи должны принадлежать M^*, а это означает, что данная цепь является увеличивающей цепью для M. Теорема доказана.

Таким образом, некоторое паросочетание является паросочетанием максимальной мощности тогда и только тогда, когда для него не существует увеличивающей цепи. Этот результат является основой для алгоритма поиска паросочетания максимальной мощности. Алгоритм выбирает некоторую открытую вершину и отыскивает увеличивающую чередующуюся цепь, соединяющую эту вершину с некоторой другой открытой вершиной. Если такая цепь найдется, то не входившие в паросочетание ребра этой цепи включаются в него. Остальные ребра цепи исключаются из паросочетания. В результате получается новое паросочетание большей мощности. Если увеличивающую цепь найти не удается, то аналогично проверятся наличие такой цепи, включающей другую открытую вершину. Работа алгоритма заканчивается, когда все открытые вершины проверены на наличие увеличивающей цепи.

Алгоритм поиска паросочетания минимальной мощности при построении дерева с корнем в открытой вершине отыскивает увеличивающую цепь, начинающуюся в этой вершине. Это дерево называется *чередующимся деревом*, так как в нем каждая цепь, начинающаяся в корне (открытой вершине), является чередующейся цепью. (Первое ребро в такой цепи не принадлежит паросочетанию, так как оно начинается в открытой вершине.) Все вершины в чередующемся дереве в свою очередь делятся на *внешние* и *внутренние*. Рассмотрим одиночную цепь, соединяющую **корень** дерева с некоторой вершиной x. Если последнее ребро в этой цепи принадлежит паросочетанию, то вершина x называется внешней вершиной чередующегося дерева. Если же последнее ребро в этой цепи не принадлежит паросочетанию, то вершина x называется внутренней вершиной чередующегося дерева. Корень дерева является его внешней вершиной (рис. 5.5, *б*). Описываемая ниже процедура определяет способ построения чередующегося дерева.

Алгоритм построения чередующегося дерева

Шаг 1. (*Выбор корня дерева.*) Выбрать любую вершину v, не принадлежащую паросочетанию M. Определить вершину v как корень дерева и пометить ее как внешнюю его вершину. Все другие вершины считать непомеченными и все ребра — неокрашенными. (Окраска ребра в оранжевый цвет будет означать включение его в чередующееся дерево.)

Шаг 2. (*Построение дерева.*) Выбрать любое неокрашенное ребро (x, y), инцидентное внешней вершине x. (Если такое ребро отсутствует, перейти к шагу 4.) Здесь возможны 3 случая:

а) y — внутренняя вершина дерева,

б) y — внешняя вершина дерева,

в) y — непомеченная вершина.

Если имеет место случай (а), то окрасить (x, y) в синий цвет (т. е. ребро не принадлежит дереву) и повторить шаг 2.

Если имеет место случай (б), то (x, y) окрасить в оранжевый цвет и перейти к шагу 3.

Если же имеет место случай (в), то (x, y) следует окрасить в оранжевый цвет. Если y — открытая вершина, то увеличивающая чередующаяся цепь оранжевых ребер от v к y найдена. Если же вершине y инцидентна дуга, входящая в паросочетание, то следует окрасить в оранжевый цвет дугу паросочетания (y, z), инцидентную y. Пометить вершину y как внутреннюю и вершину z как внешнюю. Повторить шаг 2.

Шаг 3. (*Выявление нечетного цикла.*) Шаг выполняется только тогда, когда ребро (x, y) инцидентно двум внешним вершинам и дерево окрашено в оранжевый цвет. В этом случае имеет место цикл C, состоящий из дуг, окрашенных в оранжевый цвет. Цикл C должен быть нечетным, так как в нем между внешними вершинами x и y содержатся чередующиеся внешние и внутренние вершины. Обнаружен нечетный цикл, поэтому работа алгоритма заканчивается.

Шаг 4. (*Венгерское дерево.*) Этот шаг выполняется только тогда, когда дальнейшее окрашивание ребер оказывается невозможным. Оранжевые ребра образуют чередующееся дерево, которое называется венгерским деревом. Выявлением этого дерева алгоритм заканчивается.

Заметим, что алгоритм построения чередующегося дерева заканчивается каждый раз тогда, когда

а) найдена увеличивающая чередующаяся цепь,

б) обнаружен нечетный цикл,

в) построено венгерское дерево.

ПРИМЕР 1. Пусть нами построено чередующееся дерево, приведенное на рис. 5.6, с корнем в открытой вершине a.

Шаг 1. Вершину a пометим как внешнюю вершину дерева.

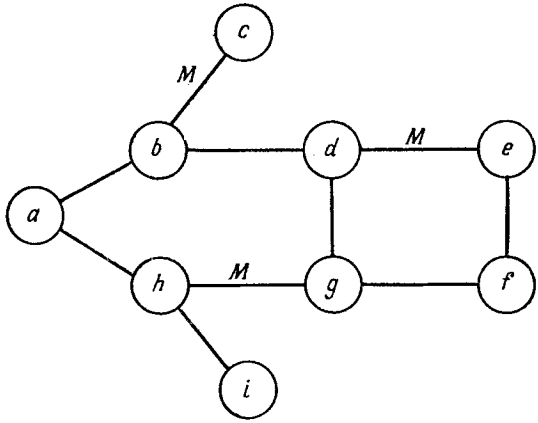


Рис. 5.6. (M — ребро, входящее в паросочетание).

Шаг 2.

Рассматриваемое ребро	Внешние вершины дерева	Внутренние вершины дерева	Оранжевые ребра
Начало	a	Отсутствует	Отсутствуют
(a, h)	a, g	h	(a, h), (h, g)
(g, d)	a, g, e	h, d	(a, h), (h, g), (g, h), (d, e)
(a, b)	a, g, e, c	h, d, b	(a, h), (h, g), (g, d), (d, e), (a, b), (b, c)
(e, f)	a, g, e, c	h, d, b	(a, h), (h, g), (g, d), (d, e), (a, b), (b, c), (e, f)

Алгоритм заканчивается, так как найдена увеличивающая цепь (a, h), (h, g), (g, d), (d, e), (e, f).

Следует отметить, что часто на шаге 2 имеется возможность просмотра нескольких ребер. В этом случае произвольно выбирается одно из них. Если бы ребро (g, f) просматривалось вместо ребра (g, d), то была бы найдена увеличивающая цепь (a, h), (h, g), (g, f). Следовательно, результат работы алгоритма построения чередующегося дерева может зависеть от того, какое ребро будет выбрано следующим для просмотра. И наконец, отметим, что если на шаге 2 пара смежных ребер окрашивается в

оранжевый цвет, то одно из них принадлежит, а другое не принадлежит паросочетанию. Работа алгоритма поиска паросочетания максимальной мощности может начинаться в предположении, что найдено произвольное паросочетание. Затем выбирается некоторая открытая вершина и с помощью описанного выше алгоритма строится чередующееся дерево с корнем в этой вершине. Если найдется увеличивающая цепь, то ребра, принадлежащие паросочетанию, из него исключаются и вместо них включаются ребра этой цепи ранее не входившие в паросочетание. В результате получается новое паросочетание, мощность которого превышает мощность исходного паросочетания. В новое паросочетание входит и вершина, являвшаяся корнем чередующегося дерева.

Если алгоритм построения чередующегося дерева приводит к построению венгерского дерева, то отсутствует увеличивающая цепь, включающая корень дерева. (Для проверки см. приводимое далее доказательство того, что алгоритм обеспечивает построение паросочетания максимальной мощности.) Если в результате работы алгоритма построения чередующегося дерева обнаруживается нечетный цикл, то этот цикл стягивается в одну вершину. Работа алгоритма продолжается на новом, прореженном, графе, полученном в результате стягивания этого нечетного цикла.

Паросочетание, построенное в результате исследования с помощью алгоритма построения чередующегося дерева всех открытых вершин, является паросочетанием максимальной мощности для результирующего графа[1]. Далее восстанавливаются все сжатые циклы и определяются на ребрах каждого из этих циклов паросочетания. Можно показать, что после восстановления всех стянутых в точки циклов получаемое в результате паросочетание является паросочетанием максимальной мощности для исходного графа. С учетом данных пояснений мы теперь можем перейти к формальному описанию *алгоритма построения паросочетания максимальной мощности.*

Алгоритм построения паросочетания максимальной мощности

Шаг 1. (*Определение начальных условий и обозначения.*) Обозначить рассматриваемый граф через G_0. Выбрать любое паросочетание M_0 графа G_0. Все открытые вершины считать *нерассмотренными*. Положить $i = 0$.

Шаг 2. (*Исследование открытой вершины.*) Если в графе

[1] Графа, получаемого после сжатия всех нечетных циклов. —*Прим. перев.*

G_i имеется только одна неисследованная открытая вершина, то перейти к шагу 6. В противном случае выбрать любую неисследованную открытую для паросочетания вершину v. Далее, используя описанный алгоритм, построить чередующееся дерево с корнем в вершине v.

Если при построении этого дерева находится увеличивающая цепь, то перейти к шагу 3. Если же при построении обнаруживается нечетный цикл, то перейти к шагу 4. И наконец, если алгоритм построения чередующегося дерева приведет к построению венгерского дерева, то перейти к шагу 5.

Шаг 3. (*Построение увеличивающей цепи.*) Этот шаг выполняется только после того, как с помощью алгоритма построения чередующегося дерева найдена увеличивающая цепь. Для выполнения шага 3 следует исключить из паросочетания M_i ребра, входящие в найденную увеличивающую цепь, и включить в него оставшиеся ребра этой цепи. В результате мощность паросочетания M возрастает на единицу и вершина v становится инцидентной ребру, входящему в паросочетание.

Шаг 4. (*Построение нечетного цикла.*) Этот шаг выполняется только после того, как находится нечетный цикл с помощью алгоритма построения чередующегося дерева. Положить $i = i + 1$. Обозначить найденный нечетный цикл через C_i. Стянуть C_i в фиктивную вершину a_i. Напомним (см. гл. 2), что каждое ребро, инцидентное вершине в C_i, становится теперь инцидентным вершине a_i. Граф, полученный в результате стягивания C_i в вершину, назовем графом G_i. Пусть паросочетание M_i включает все ребра паросочетания M_{i-1}, входящие в множество ребер графа G_i. (Заметим, что, кроме одной, все вершины в цикле C_i инцидентны ребрам этого цикла, принадлежащим паросочетанию. Следовательно, после стягивания C в вершину a_i в лучшем случае одно ребро, принадлежащее паросочетанию, будет инцидентно a_i.) Вернуться к шагу 2, выбирая для исследования в качестве открытой вершины отображение вершины v в графе G_i. Отметим, что на шаге 2 большая часть помеченных вершин и раскрашенных ребер одной итерации алгоритма построения чередующегося дерева может быть повторно использована на следующей за ней итерации этого алгоритма.

Шаг 5. (*Построение венгерского дерева.*) Этот шаг выполняется только после того, как с помощью алгоритма построения чередующегося дерева будет построено венгерское дерево с корнем в открытой вершине v. Считать вершину v просмотренной. Вернуться к шагу 2.

Шаг 6. (*Восстановление стянутых в вершину нечетных циклов.*) Этот шаг выполняется только после того, как все вершины (кроме открытых) просмотрены или стали инцидентными ребрам паросочетания.

В результате повторения шага 4 была получена последовательность графов G_1, G_2, ..., G_t и последовательность фиктивных вершин a_1, a_2, ..., a_t. Кроме того, была получена соответственно последовательность паросочетаний M_1, M_2, ..., M_t в графах G_1, G_2, ..., G_t.

Паросочетание M_t является паросочетанием максимальной мощности в графе G_t. Положить $M_t^* = M_t$. Для $j = t$, t —

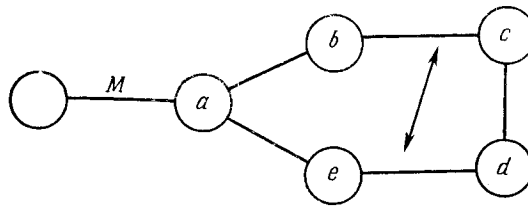


Рис. 5.7. Выбор включаемых в паросочетание ребер нечетного цикла C_i. Добавить два ребра (b, c) и (d, e) к M^*_{j-1}.

— 1, ..., 1 построить из паросочетания M_j^* в графе G_j паросочетание M_{j-1}^* в графе G_{j-1} следующим образом:

(а) Если вершина a_j инцидентна ребру, принадлежащему паросочетанию M_j^*, то M_{j-1}^* включает все ребра паросочетания M_j^* совместно с единственным множеством ребер нечетного цикла C_i, которые инцидентны всем открытым относительно M_j^* вершинам цикла C_i (рис. 5.7).

(б) Если вершина a_j не инцидентна ребру, принадлежащему паросочетанию M_j^*, то M_{j-1}^* включает все ребра паросочетания M_j^* совместно с любым множеством ребер в C_i, инцидентных всем, кроме одной, вершинам в C_i. Выбор вершины в C_i, остающейся открытой, произволен (рис. 5.8). *Паросочетание M_0^* является паросочетанием максимальной мощности исходного графа G_0.*

Доказательство. Доказательство того, что M_0^* является паросочетанием максимальной мощности в графе G_0, распадается на две части, в которых необходимо: 1) доказать, что паросочетание M_t^* является паросочетанием максимальной мощности в графе G_t; 2) доказать, что если паросочетание M_i^* является паросочетанием максимальной мощности в графе G_i,

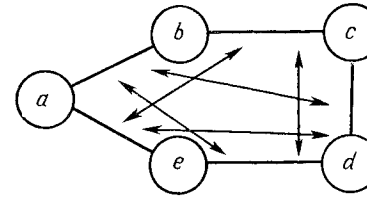


Рис. 5.8. Выбор включаемых в паросочетание ребер нечетного цикла C_i. Добавить к M^*_{j-1} любые два несмежных ребра. Например, если вершина b должна остаться открытой, то следует включить в паросочетание ребра (a, c) и (c, d). Все вершины нечетного цикла C_i являются открытыми в M^*_j.

то паросочетание M_{i-1}^* является паросочетанием максимальной мощности в графе G_{i-1}.

Доказательство того, что M_0^* является паросочетанием максимальной мощности в графе G_0, следует из последовательного применения результата доказательства (2) к паросочетаниям M_t^*, M_{t-1}^*, M_{t-2}^*, ..., M_0^*.

Часть 1. Для того чтобы доказать, что M_t^* является паросочетанием максимальной мощности в графе G_t, достаточно показать, что в G_t отсутствует увеличивающая цепь, начинающаяся в любой открытой вершине графа G_t и оканчивающаяся в отличной от нее открытой вершине этого графа. Предположим, что для паросочетания M_t^* имеется увеличивающая цепь C, соединяющая открытые вершины v и w в графе G_t. По крайней мере одна из этих двух вершин, скажем вершина v, при построении M_t^* должна была быть исследована на шаге 2 алгоритма поиска паросочетания максимальной мощности. Это исследование должно было закончиться на шаге 5 построением венгерского дерева с корнем в вершине v. Проследим цепь C от вершины v к вершине w. Пусть через (x, y) обозначается первое ребро в C, не принадлежащее венгерскому дереву. Таким образом, вершина x должна быть внешней вершиной дерева. Вершина y может быть (а) открытой, (б) непомеченной и инцидентной ребру, входящему в паросочетание, (в) помеченной внешней или (г) помеченной внутренней.

Рассмотрим каждый из четырех случаев.

(а) Вершина y не может быть открытой; в противном случае шаг 2 закончился бы нахождением увеличивающей цепи от v к y, а не построением венгерского дерева с корнем в вершине v.

(б) Вершина y не может быть непомеченной и инцидентной ребру паросочетания; в противном случае при построении с помощью описанного выше алгоритма чередующегося дерева она была бы помечена как внутренняя вершина.

(в) Вершина y не может быть внешней вершиной; в противном случае при построении с помощью описанного выше алгоритма чередующегося дерева ребро (x, y) было бы окрашено и был бы образован нечетный цикл.

(г) Если вершина y — внутренняя вершина чередующегося дерева, то чередующаяся цепь от корня v до вершины y представляет часть другой увеличивающей цепи C' от v к w. Увеличивающая цепь C' состоит из чередующейся цепи от v до y в венгерском дереве и части C от y к w.

Повторим проведенный выше анализ для цепи C'. Этот анализ приведет либо к противоречию, как в случаях (а), (б) и (в), либо к другой увеличивающей цепи C''. Каждая полученная с помощью такой процедуры увеличивающая цепь должна иметь меньше ребер, непринадлежащих венгерскому дереву, чем в предыдущей цепи.

Таким образом, мы, в конце концов, либо придем к противоречию, либо покажем, что при использовании алгоритма построения чередующегося дерева вместо венгерского дерева была окрашена увеличивающая чередующаяся цепь. Следовательно, если существует увеличивающая цепь, то исследование вершины v не может приводить к построению венгерского дерева. Следовательно, первая часть доказательства выполнена.

Часть 2. Предположим, что в графе G_{i-1} паросочетание M^*_{i-1} не является паросочетанием максимальной мощности. Тогда имеется некоторая увеличивающая цепь, соединяющая две открытые вершины в графе G_{i-1}. Легко заметить, что отображение этой цепи на графе G_i также является увеличивающей цепью по отношению к паросочетанию M^*_i. Но это противоречит предположению, что M^*_i является паросочетанием максимальной мощности на графе G_i. Следовательно, вторая часть доказательства завершена.

ПРИМЕР 2. Пусть нами найдено паросочетание максимальной мощности для графа, представленного на рис. 5.9, *а*. Отметим, что в этом графе имеется 9 вершин, и, следовательно, ни одно из паросочетаний не может содержать более четырех ребер. Рассмотрим на этом примере работу алгоритма поиска паросочетания максимальной мощности.

Шаг 1. В качестве исходного выбрать паросочетание, показанное на рис. 5.9, *а*.

Шаг 2. Для исследования в качестве открытой вершины выбрать вершину a и воспользоваться алгоритмом построения чередующегося дерева. Вершину a пометить как внешнюю. Окрасить ребра (a, d) и (d, b). Вершины d и b пометить соответственно как внутреннюю и внешнюю. Окрасить ребро (a, b) между внешними вершинами a и b. В результате найден нечетный цикл (a, d), (d, b), (b, a). Перейти к шагу 4.

Шаг 4. Стянуть найденный на шаге 2 нечетный цикл в фиктивную вершину a_1. Новый граф G_1 показан на рис. 5.9, *б*. Вернуться к шагу 2.

Шаг 2. Для просмотра выбрать открытую вершину a_1 (отображение вершины a на графе G_1) и воспользоваться алгоритмом построения чередующегося дерева. Вершину a пометить как внешнюю. Окрасить ребро (a, c). В результате найдена увеличивающая цепь (a, c). Перейти к шагу 3.

Шаг 3. Добавить ребро (a, c) к паросочетанию (рис. 5.9, *в*). Вернуться к шагу 2.

Шаг 2. Для просмотра выбрать вершину g и воспользоваться алгоритмом построения чередующегося дерева. Вершину g пометить как внешнюю. Окрасить ребра (g, h) и (h, f). Вершины h и f пометить соответственно как внутреннюю и внешнюю. Окрасить ребро (f, i). В результате найдена увеличивающая цепь (g, h), (h, f), (f, i). Перейти к шагу 3.

Шаг 3. Исключить ребро (h, f) из паросочетания. Включить в паросочетание ребра (g, h) и (f, i) (рис. 5.9, *г*). Перейти к шагу 2.

Шаг 2. Открытой является единственная вершина e. Перейти к шагу 6.

Шаг 6. Преобразовать вершину a в соответствующий ей нечетный цикл. Он включает ребра (a, d), (d, b), (b, a). Так как вершина b инцидентна ребру паросочетания, то в паросочетание включается ребро (a, d).

Получаемое в результате паросочетание представлено на рис. 5.9, *д*.

Отметим, что ни одно из ребер, входивших в начальное паросочетание, не вошло в результирующее паросочетание.

Следует отметить также, что возможны и другие паросочетания макси-

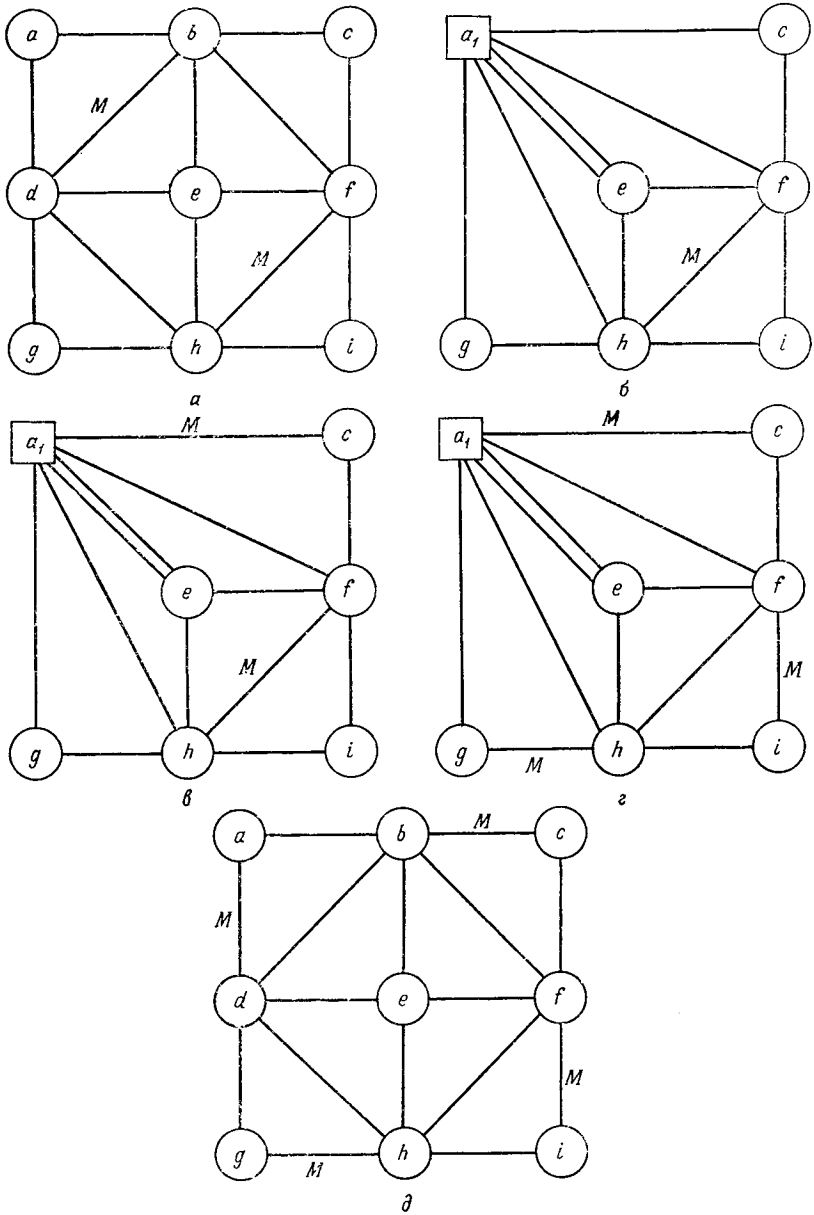


Рис. 5.9. Пример выполнения алгоритма поиска чередующегося дерева.

мальной мощности. Например, паросочетание $\{(a, b), (c, f), (i, h), (d, g)\}$ также является паросочетанием максимальной мощности.

Получаемое с помощью рассмотренного алгоритма паросочетание существенно зависит от начального выбора открытой вершины и отсутствия ограничений при выборе ребер для их окраски в соответствии с алгоритмом построения чередующегося дерева.

5.3. Алгоритм выбора паросочетания с максимальным весом

В этом разделе представлен предложенный Эдмондсом и Джонсоном [6] алгоритм выбора для графа $G = (X, E)$ паросочетания с максимальным весом.

Основной операцией этого алгоритма, как и алгоритма поиска паросочетания максимальной мощности, рассмотренного в разд. 5.2, является построение чередующегося дерева. Как мы видели в разд. 5.2, увеличение мощности паросочетания возможно в основном благодаря наличию увеличивающих чередующихся цепей. Аналогичный результат справедлив и для паросочетаний с максимальным весом.

Взвешенной увеличивающей чередующейся цепью является чередующаяся цепь, в которой общий вес ребер, не входящих в паросочетание, превышает общий вес ребер, входящих в это паросочетание; первая вершина в цепи является открытой, если первое ребро в этой цепи не принадлежит паросочетанию; последняя вершина в цепи является открытой, если последнее ребро в этой цепи не принадлежит паросочетанию.

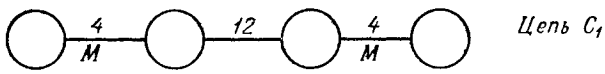
Заметим, что если заменить входившие в паросочетание ребра не входившими в него ребрами, то новое паросочетание имеет больший вес, чем исходное. Например, на рис. 5.10 цепи C_1, C_2 и C_3 являются чередующимися. В каждой цепи ребра, входящие и не входящие в паросочетание, имеют суммарный вес, соответственно равный 8 и 12 единицам. Все они являются взвешенными увеличивающими чередующимися цепями. Цепь C_1 называется *слабой увеличивающей чередующейся цепью*, так как в ней число ребер, принадлежащих паросочетанию, превышает число ребер, не принадлежащих ему.

Цепь C_2 называется *нейтральной увеличивающей чередующейся цепью*, так как в ней число ребер, принадлежащих паросочетанию, равно числу ребер, не принадлежащих ему. Цепь C_3 называется *сильной увеличивающей чередующейся цепью*, так как в ней число ребер, не входящих в паросочетание, превосходит число ребер, входящих в него.

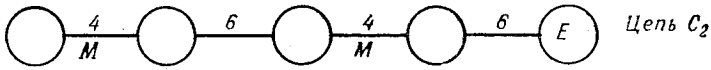
Рассмотрим теорему, аналогичную теореме 5.2.

ТЕОРЕМА 5.3. Паросочетание M тогда и только тогда является паросочетанием с максимальным весом, когда для него не существует взвешенных увеличивающих цепей.

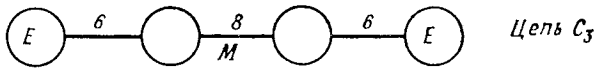
Доказательство. Если для паросочетания M имеется взвешенная увеличивающая цепь C, то M не может быть паросочетанием с максимальным весом. Действительно, паросочетание M', полученное при замене ребер цепи C, входивших в M, не входившими в него ребрами этой цепи, имеет больший вес, чем паросочетание M. Для того чтобы доказать вторую часть теоремы, предположим, что M^* — произвольное паросочетание с весом бо́льшим, чем вес паросочетания M. Как и в теореме 5.2, рассмотрим множество всех ребер, которые входят только в одно из паросочетаний M и M^*. Из доказательства теоремы 5.2 мы знаем, что каждый связный компонент, состоящий из ребер этого множества, должен быть либо циклом, либо цепью. Так как все циклы могут рассматриваться как цепи, то будем рассматривать каждый связный компонент как некоторую цепь. Поскольку паросочетание M^* имеет вес, превышающий вес паросочетания M, то в одной из этих цепей ребра, входящие в паросочетание M^*, должны иметь больший суммарный вес, чем ребра, принадлежащие паросочетанию M. Следовательно, эта цепь должна быть взвешенной увеличивающей цепью для паросочетания M, что противоречит условиям теоремы. Теорема доказана.



Слабая увеличивающая чередующаяся цепь



Нейтральная увеличивающая чередующаяся цепь



Сильная увеличивающая чередующаяся цепь

Рис. 5.10. Взвешенные увеличивающие чередующиеся цепи.

Пусть $V = \{V_1, V_2, \ldots, V_z\}$ — множество всех подмножеств вершин графа, включающих нечетное количество вершин. Пусть T_m — множество всех ребер, обе вершины которых принадлежат подмножеству V_m и $T = \{T_1, T_2, \ldots, T_z\}$. Обозначим количество вершин вV_m через $2n_m + 1$. Тогда справедливо следующее утверждение.

Ни одно паросочетание не может содержать более чем n_m *ребер, принадлежащих множеству* T_m. Пусть через $a(i, j)$ обозначен вес ребра (i, j) и $x(i, j) = 1$, если ребро (i, j) принадлежит паросочетанию; в противном случае $x(i, j) = 0$.

Для того чтобы легче было понять алгоритм поиска паросочетания с максимальным весом, мы должны рассмотреть сначала постановку задачи поиска паросочетаний с максимальным весом как следующую задачу линейного программирования: максимизировать

$$\sum_{(i, j)} a(i, j)\, x(i, j) \tag{6}$$

$$\sum_{j} [x(i, j) + x(j, i)] \leqslant 1, \text{ для всех } i \in X \tag{7}$$

$$\sum_{i,\, j \in T_m} x(i, j) \leqslant n_m, \quad m = 1, 2, \ldots, z \tag{8}$$

$$x(i, j) \geqslant 0, \text{ для всех } (i, j). \tag{9}$$

(Заметим, что ребро соединяющее вершины i и j, обозначается через (i, j) или (j, i).)

Ограничение (7) требует, чтобы каждой вершине i было инцидентно не более чем одно ребро паросочетания. Ограничение (8) означает, что в паросочетание должно входить не более n_m ребер из множества T_m. Целевая функция (6) представляет собой суммарный вес ребер паросочетания. Каждое паросочетание удовлетворяет ограничениям (7) — (9). Однако практически невозможно перечислить все эти ограничения даже для графов небольшой размерности. К счастью, применение алгоритма построения паросочетания с максимальным весом дает в результате паросочетание, являющееся оптимальным решением задачи линейного программирования (6) и (9). Каким же образом можно показать, что это действительно так?

Это можно сделать путем поиска допустимого решения двойственной задачи, которое совместно с решением прямой задачи удовлетворяет всем условиям дополняющей нежесткости. Задача линейного программирования, двойственная по отношению к задаче (6) — (9), имеет следующий вид:

минимизировать

$$\sum_{i \in X} y_i + \sum_{m=1}^{m=z} n_m z_m \tag{10}$$

при ограничениях

$$y_i + y_j + \sum_{m:\,(i,\,j) \in T_m} z_m \geqslant a(i,\,j), \text{ для всех } (i,\,j) \tag{11}$$

$$y_i \geqslant 0, \text{ для всех } i \in X \tag{12}$$

$$z_m \geqslant 0, \quad m = 1,\, 2,\, \ldots,\, z\,. \tag{13}$$

Двойственная переменная y_i соответствует ограничению (7) прямой задачи для вершины i. Двойственная переменная z_m соответствует ограничению (8) прямой задачи для T_m.

Условия дополняющей нежесткости для этой пары (прямой и двойственной) задач линейного программирования имеют вид

$$x(i,\,j) > 0 \Rightarrow y_i + y_j + \sum_{m:\,(i,\,j) \in T_m} z_m = a(i,\,j), \text{ для всех } (i,\,j) \tag{14}$$

$$y_i > 0 \Rightarrow \sum_{j \in X} [x(i,\,j) + x(j,\,i)] = 1, \text{ для всех } i \in X \tag{15}$$

$$z_m > 0 \Rightarrow \sum_{(i,\,j) \in T_m} x(i,\,j) = n_m, \quad m = 1,\, 2,\, \ldots,\, z \tag{16}$$

Рассмотрим работу алгоритма поиска паросочетания с максимальным весом.

Выполнение алгоритма начинается с нулевого паросочетания [все $x(i,\,j) = 0$] и допустимых значений двойственных переменных y_i, $i \in X$ и z_m, $m = 1,\, 2,\, \ldots,\, z$, удовлетворяющих условиям дополняющей нежесткости (14) и (16). Только условие (15) остается невыполненным. На каждой итерации алгоритма паросочетание и (или) значения двойственных переменных изменяются так, что все условия (7) — (9), (11) — (13), (14) и (16) продолжают выполняться, а условие (15) выполняется дополнительно по крайней мере еще для одной двойственной переменной y_i. Поскольку имеется только $|X|$ двойственных переменных y_i, то не более чем через $|X|$ итераций все условия (15) становятся выполненными. Благодаря выполнению условий дополняющей нежесткости получаемое в результате паросочетание должно быть паросочетанием с максимальным весом. Рассмотрим более внимательно условие (15). Оно устанавливает, что если двойственная переменная y_i для вершины i имеет положительное значение, то вершина i инцидентна ребру, входящему в паросочетание. Таким образом, условие (15) нарушается только для открытых вер-

шин, которым соответствуют положительные значения двойственных переменных.

Рассмотренный алгоритм определяет какую-либо открытую вершину v, такую, что $y_v > 0$, и использует алгоритм построения чередующегося дерева с корнем в вершине v. Как мы видели в разд. 5.2, работа этого алгоритма завершается

а) нахождением увеличивающей чередующейся цепи,

б) определением нечетного цикла,

в) построением венгерского дерева.

Если алгоритм обнаруживает увеличивающую цепь, то эта цепь является сильной увеличивающей цепью. Входившие в паросочетание ребра цепи заменяются не входившими в него ребрами этой же цепи. Это приводит к увеличению общего веса паросочетания и включению в него ребра, инцидентного v.

В результате построения увеличивающей чередующейся цепи для вершины v выполняется условие (15). Если с помощью алгоритма построения чередующегося дерева находится нечетный цикл, этот цикл стягивается в фиктивную вершину, и алгоритм продолжает работу на преобразованном графе. И наконец, если находится венгерское дерево, то изменяются двойственные переменные таким образом, что остаются выполненными все условия прямой и двойственной задач и условие дополняющей нежесткости, за исключением, может быть, условия (15). В результате к чередующемуся дереву может быть добавлено не принадлежащее ему ребро. В итоге или значение y_v уменьшается до нуля, в результате чего выполняется условие (15), или вершина v становится инцидентной ребру паросочетания.

В процессе работы алгоритма нечетные циклы стягиваются в фиктивные вершины. Впоследствии все эти фиктивные вершины обратно преобразуются (растягиваются) в соответствующие исходные нечетные циклы. Однако порядок обратного преобразования фиктивных вершин в нечетные циклы может не совпадать с порядком, в котором они формировались.

Благодаря таким стягиваниям нечетных циклов и обратным преобразованиям фиктивных вершин в нечетные циклы алгоритм будет порождать последовательность графов $G_0, G_1, \ldots, G_t$. Рассмотренных основных положений достаточно теперь для формального изложения алгоритма построения паросочетания с максимальным весом.

Алгоритм построения паросочетания с максимальным весом

Шаг 1. (*Выбор начальных значений.*) Пусть сначала M_0 — пустое множество ребер, а все двойственные переменные z_m равны нулю, $(m = 1, 2, \ldots, z)$. Выбрать такие начальные значения двойственных переменных y_i, $i \in X$, что $y_i + y_j \geqslant a(i, j)$

для всех ребер (i, j). (Например, можно выбрать ребро, имеющее максимальное значение веса, и каждое y_i принять равным половине значения этого веса.) Положить $k = 0$ и обозначить исходный граф через $G_k = (X_k, E_k)$.

Шаг 2. (*Поиск открытой вершины.*) Выбрать в графе G_k любую нефиктивную открытую вершину с $y_v > 0$. Если такой вершины не существует, то перейти к шагу 6. В противном случае, выделить в G_k множество E^* ребер (i, j), для которых выполняется условие

$$y_i + y_j + \sum_{(i, j) \in T_m} z_m = a(i, j). \tag{17}$$

С помощью алгоритма построения чередующегося дерева построить дерево с корнем в вершине v, содержащее только ребра из множества E^*. Если при этом найдется увеличивающая чередующаяся цепь, то перейти к шагу 3. Если будет построен нечетный цикл, то перейти к шагу 4. И наконец, если будет построено венгерское дерево, то перейти к шагу 5.

Шаг 3. (*Преобразование паросочетания.*) Этот шаг выполняется только тогда, когда с помощью алгоритма построения чередующегося дерева отыскивается увеличивающая цепь. Входящие в паросочетание M_k ребра цепи заменить не входящими в него ребрами этой же цепи. Вершина v перестает быть открытой. Вернуться к шагу 2.

Шаг 4. (*Построение нечетного цикла.*) Этот шаг выполняется только при завершении алгоритма построения чередующегося дерева нахождением нечетного цикла. Положить $k = k + 1$. Обозначить этот нечетный цикл через C_k. Стянуть C_k в фиктивную вершину a_k. Обозначить полученный новый граф через $G_k = (X_k, E_k)$. Пусть M_k — паросочетание, включающее все ребра, которые одновременно принадлежат паросочетанию M_{k-1} и графу G_k. В дальнейшем при расстановке пометок все вершины, которые заменены фиктивной вершиной a_k, помечать одинаково с a_k. Вернуться к шагу 2 и продолжить построение чередующегося дерева с корнем в вершине, являющейся отображением вершины v в G_k (даже если эта вершина является фиктивной). Заметим, что разметка вершин и раскраска ребер, полученных на последней итерации алгоритма построения чередующегося дерева, могут оказаться полезными на последующей итерации.

Шаг 5. (*Венгерское дерево.*) Этот шаг выполняется только тогда, когда результатом алгоритма построения чередующегося дерева является венгерское дерево.

Принять

$$d_1 = \min \{y_i + y_j - a(i, j)\}, \tag{18}$$

где минимум берется по всем (i, j), таким, что $i \in X_0$ является внешней вершиной дерева и вершина $j \in X_0$ не помечена[1].

Положить

$$d_2 = \frac{1}{2} \min \{y_i + y_j - a(i, j)\}, \tag{19}$$

где минимум берется по всем (i, j), таким, что $i \in X_0$ и $j \in X_0$ являются внешними вершинами дерева, но они не стянуты в одну и ту же фиктивную вершину[2].

Принять $$d_3 = \frac{1}{2} \min \{z_m\}, \tag{20}$$

где минимум берется по всем множествам V_m вершин, мощность которых равна нечетному числу и которые стянуты в фиктивную вершину a_k, являющуюся внутренней вершиной дерева[3]

Положить $$d_4 = \min \{y_i\}, \tag{21}$$

где минимум берется по всем внешним вершинам $i \in X_0$ дерева[4]. И наконец, определить

$$d = \min \{d_1, d_2, d_3, d_4\}. \tag{22}$$

Изменить двойственные переменные y_i, z_m следующим образом:
(а) Переменные y_i, соответствующие внешним вершинам дерева, уменьшить на величину d.
(б) Переменные y_i, соответствующие внутренним вершинам дерева, увеличить на d.
(в) Увеличить на $2d$ каждую двойственную переменную z_m, соответствующую внешней фиктивной вершине дерева в графе G_k.
(г) Уменьшить на $2d$ каждую двойственную переменную z_m, соответствующую внутренней фиктивной вершине дерева в графе G_k.

При этом если $d = d_1$, то в E^* включить ребро (i, j), для которого достигается минимум в (18). Это ребро теперь может быть окрашено в процессе реализации алгоритма построения чередующегося дерева. Поэтому следует вернуться к шагу 2 и продолжить построение чередующегося дерева с корнем в вершине v.

Если $d = d_2$, то в E^* включить ребро (i, j), для которого достигается минимум в (19). Это ребро теперь может быть окрашено в результате реализации алгоритма построения чередующегося дерева, что приводит к обнаружению нечетного цикла. Поэтому

---

[1] Если множество таких ребер (i, j) пустое, то $d_1 = \infty$.
[2] Если множество таких ребер (i, j) пустое, то $d_2 = \infty$. — *Прим. перев.*
[3] Если множество таких вершин пустое, то $d_3 = \infty$.
[4] Если множество таких вершин пустое, то $d_4 = \infty$. — *Прим. перев.*

следует вернуться к шагу 2 и продолжить построение чередующегося дерева с корнем в вершине v.

Если $d = d_3$, то какая-либо из двойственных переменных z_i становится равной нулю. Преобразовать в исходный нечетный цикл фиктивную вершину, соответствующую этой двойственной переменной. Положить $k = k + 1$. Полученный после обратного преобразования фиктивной вершины граф обозначить через $G_k = (X_k, E_k)$. Положить паросочетание M_k состоящим из всех ребер паросочетания M_{k-1} и n_i ребер множества T_i, инцидентных $2\,n_i$ вершинам множества V_i. (Оставшаяся одна вершина множества V_i инцидентна ребру паросочетания M_k, так как все внутренние вершины дерева в графе C_{k-1} инцидентны ребрам паросочетания M_{k-1}.) После такого преобразования чередующегося дерева следует вернуться к шагу 2 и продолжить построение чередующегося дерева с корнем в v.

Если $d = d_4$, то двойственная переменная y_i, соответствующая какой-либо из внешних вершин i, становится равной нулю. Тогда в чередующемся дереве цепь, соединяющая корень v с вершиной i, является нейтральной увеличивающей цепью. В паросочетании M_k произвести замену входящих в него ребер цепи невходящими ребрами этой же цепи. Вершина v становится инцидентной ребру паросочетания, а вершина i становится открытой, что допустимо из-за равенства нулю y_i. Вернуться к шагу 2.

Шаг 6. (*Обратное преобразование фиктивных вершин в нечетные циклы.*) Этот шаг выполняется только после того, как на шаге 2 просмотрены все вершины, для которых не выполняется условие (15). Рассмотреть все оставшиеся фиктивные вершины в полученном к этому моменту графе. Произвести преобразование фиктивных вершин в нечетные циклы в порядке, обратном их отысканию (полученная последней фиктивная вершина преобразовывается в нечетный цикл первой и т. д.) и образовать паросочетание максимальной мощности на каждом из этих нечетных циклов. Полученное в результате паросочетание является паросочетанием с максимальным весом для начального графа G_0. На этом работа алгоритма заканчивается.

Доказательство оптимальности получаемого решения. Алгоритм начинает свою работу с паросочетания нулевой мощности и на каждой итерации обеспечивает получение некоторого паросочетания. Нам остается только доказать, что получаемое на последней итерации паросочетание является паросочетанием с максимальным весом. А это доказывается тем, что окончательные значения двойственных переменных y_i, $i \in X$, и z_m, $m = 1, 2, \ldots, z$, удовлетворяют ограничениям двойственной задачи (11) — (13) и условиям дополняющей нежесткости (14) — (16). Так как равенства (18) — (22) гарантируют, что ни одна из двойственных переменных не уменьшится до отрицательного значе-

ния и так как на шаге 1 начальные значения двойственных переменных выбираются неотрицательными, то на всех итерациях алгоритма условия (12) и (13) будут выполнены.

Для доказательства того, что условие (11) также выполняется на всех итерациях алгоритма, заметим следующее:

1. Начальные значения двойственных переменных выбраны так, что выполняются условия (11).

2. Ребро (i, j) должно входить либо (а) в фиктивную вершину, либо (б) не в фиктивную вершину, но в множество E^*, либо (в) не должно входить ни в фиктивную вершину, ни в множество E^*.

В случае (а) изменение значений двойственных переменных не приводит к изменению левой части условия (11), так как и y_i и y_j изменяются на величину d, а двойственная переменная, соответствующая фиктивной вершине, включающей (i, j), изменяется на $-2d$.

В случае (б) изменение значений двойственных переменных приводит к увеличению значений двойственных переменных, соответствующих внутренним вершинам дерева, и уменьшению значений двойственных переменных, соответствующих внешним вершинам дерева. Следовательно, значение левой части условия (11) остается неизменным.

В случае (в) всегда

$$y_i + y_j + \sum_{m:\,(i,\,j)\in T_m} z_m > a(i, j).$$

Если i и j не помечены или они имеют разные пометки, то изменения значений двойственных переменных не приводят к нарушению приведенного выше неравенства. Если одна из вершин i, j имеет метку внутренней вершины, а другая не помечена или если обе вершины i и j являются внутренними вершинами дерева, то левая часть условия (11) возрастает после изменения значений двойственных переменных. Если одна из вершин i и j помечена как внешняя вершина дерева, а другая не помечена, то, согласно равенству (18), в результате изменения значений двойственных переменных не произойдет изменения знака приведенного выше неравенства на противоположный. Если i и j являются внешними вершинами дерева, то, согласно равенству (19), изменение двойственных переменных также не приведет к изменению знака в предыдущем неравенстве на противоположный. Таким образом, при всех возможных изменениях значений двойственных переменных условие (11) выполняется.

Для доказательства того, что результирующие значения переменных $x(i, j)$, y_i, z_m для всех i, j, m удовлетворяют условиям (14), мы должны рассмотреть следующие два случая:

(а) Ребро (i, j), принадлежащее паросочетанию, к началу

шага 6 не входит в нечетный цикл, стянутый в фиктивную вершину;

(б) Ребро (i, j), принадлежащее паросочетанию, содержится к началу шага 6 в нечетном цикле, стянутом в фиктивную вершину.

В случае (а) в соответствии с алгоритмом ребро $(i, j) \in E^*$ и условие (14) выполняется.

В случае (б) ребро (i, j) содержится в некотором нечетном цикле, стянутом в фиктивную вершину, после того как произведено последнее изменение значений двойственных переменных, и поэтому условие (11) выполняется как равенство. Следовательно, условие (14) выполняется при значениях двойственных переменных, полученных в конце работы алгоритма.

Условие (15) выполняется к концу работы алгоритма для всех вершин; в противном случае следовало бы повторить шаг 2 для любой вершины, для которой не выполняется условие (15).

Наконец, осталось показать, что выполняется условие (16). z_m может стать положительным только тогда, когда вершины множества V_m стягиваются в фиктивную вершину. На шаге 6 каждая фиктивная вершина обратно преобразуется в нечетный цикл, на котором определяется паросочетание максимальной мощности. Таким образом, условие (16) к концу работы алгоритма выполняется. Что и требовалось доказать.

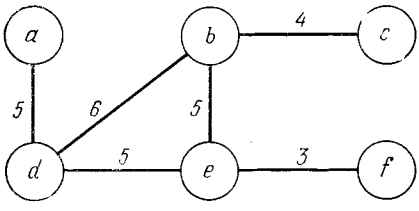


Рис. 5.11. Пример выполнения алгоритма поиска паросочетания с максимальным весом.

ПРИМЕР 3. Пусть необходимо найти паросочетание с максимальным весом для графа, приведенного на рис. 5.11.

Шаг 1. Так как наибольший вес ребра равен 6, примем $y_i = {}^6/_2 = 3$ для всех вершин i графа. Пусть все $z_i = 0$. Сначала паросочетание не содержит ни одного ребра (пустое паросочетание).

Шаг 2. Для просмотра выбирается открытая вершина d. Строится чередующееся дерево: $E^* = \{(d, b)\}$. Вершина d помечается как внешняя вершина дерева. Далее помечается вершина b, и в результате оказывается построенной увеличивающая чередующаяся цепь (d, b). Переходим к шагу 3.

Шаг 3. В паросочетание включается ребро (d, b).

Шаг 2. Для просмотра выбирается открытая вершина e.

Строится чередующееся дерево: $E^* = \{(d, b)\}$. Вершина e получает метку внешней вершины дерева. Новые вершины не могут быть помечены.

Чередующееся дерево, включающее только вершину e, является венгерским деревом. Переходим к шагу 5.

Шаг 5. Изменяются значения двойственных переменных:

$$d_1 = \min \{y_d + y_e - a(d, e),\ y_b + y_e - a(b, e),$$
$$y_f - y_e - a(f, e)\} =$$

$= \min\{3+3-5,\ 3+3-5,\ 3+3-3\} = 1,$

$d_2 = \infty,\ d_3 = \infty,\ d_4 = y_e = 3,$

$d = \min\{d_1,\ d_2,\ d_3,\ d_4\} = \min\{1,\ \infty,\ \infty,\ 3\} = 1.$

Таким образом, y_e уменьшается на 1 (табл. 5. 1).

Таблица 5.1

Изменения двойственных переменных в алгоритме определения паросочетания с максимальным весом

	y_a	y_b	y_c	y_d	y_e	y_f	z_g	Паросочетание
Начальные значения	3	3	3	3	3	3	0	Пустое
Просмотр d	3	3	3	3	3	3	0	(d, b)
Просмотр e	3	3	3	3	2	3	0	(d, b)
Просмотр g	3	2	3	2	1	3	2	(a, g)
Просмотр c	3	2	2	2	1	3	2	(a, g)
	2	3	1	3	2	3	0	(a, d), (e, b)
Просмотр f	2	3	1	3	2	1	0	(a, d), (f, e), (b, c)

Шаг 2. Продолжается просмотр открытой вершины e.

Строится чередующееся дерево: $E^* = \{(d, b), (d, e), (e, b)\}$. Вершина e получает метку внешней вершины дерева. Вершина b помечается как внутренняя, вершина d — как внешняя вершина дерева и ребра (e, b) и (b, d) окрашиваются. Далее окрашивается ребро (d, e), соединяющее внешние вершины дерева d и e. Получен нечетный цикл. Переходим к шагу 4.

Шаг 4. Нечетный цикл $(e, b), (d, b), (d, e)$ стягивается в фиктивную вершину g. После стягивания получается новый граф, показанный на рис. 5.12.

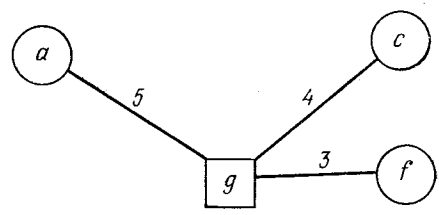


Рис. 5.12. Граф, полученный после стягивания в вершину g нечетного цикла.

Шаг 2. Просматривается открытая фиктивная вершина g. Строится чередующееся дерево: $E^* = \{(d, b), (d, e), (e, b)\}$. Вершина g помечается как внешняя вершина дерева. (Следовательно, вершины d, b, e также получают метку внешней вершины.) Дальнейшая метка вершин невозможна. Дерево, состоящее только из вершины g, является венгерским деревом. Переходим к шагу 5.

Шаг 5. Изменяются значения двойственных переменных:

$d_1 = \min\{y_b + y_c - a(b, c),\ y_a + y_d - a(a, d),\ y_e + y_f - a(e, f)\} =$

$= \min\{3+3-4,\ 3+3-5,\ 2+3-3\} = \min\{2,\ 1,\ 2\} = 1,$

$d_2 = \infty,\ d_3 = \infty$ (отсутствуют внутренние фиктивные вершины дерева),

$d_4 = \min\{y_b, y_d, y_e\} = \min\{3, 3, 2\} = 2,$

$d = \min\{1, \infty, \infty, 2\} = 1.$

Таким образом, $y_d = 3 - 1 = 2$, $y_e = 2 - 1$, $y_b = 3 - 1 = 2$, $z_g = 2d = 2$. Заметим, что для каждого ребра, принадлежащего E^*, условие (11) выполняется как равенство, например для ребра (d, e)

$y_d + y_e + z_g = 2 + 1 + 2 = 5 = a(d, e).$

Шаг 2. Продолжается просмотр открытой фиктивной вершины g.

Строится чередующееся дерево: $E^* = \{(d, e), (e, b), (b, d), (a, d)\}$. Вершина g помечается как внешняя вершина дерева. Таким образом, вершины b, e, d также помечаются как вершины дерева. Ребро (a, g) окрашивается в оранжевый цвет, и так как вершина a является открытой, то увеличивающая цепь найдена. Переходим к шагу 3.

Шаг 3. В паросочетание включается ребро (a, g).

Шаг 2. Для просмотра выбирается открытая вершина c.

Строится чередующееся дерево: $E^* = \{(d, e), (e, b), (b, d), (a, d)\}$. Вершина g помечается как внешняя вершина дерева. Дальнейшая метка вершин невозможна. Дерево, состоящее только из вершины c, является венгерским деревом. Переходим к шагу 5.

Шаг 5. Изменяются значения двойственных переменных:

$d_1 = y_c + y_b - a(c, d) = 3 + 2 - 4 = 1,$

$d_2 = \infty, \quad d_3 = \infty, \quad d_4 = y_c = 3,$

$d = \min\{1, \infty, \infty, 3\} = 1.$

Таким образом, $y_c = 3 - 1 = 2$.

Шаг 2. Продолжается просмотр открытой вершины c. Строится чередующееся дерево: $E^* = \{(d, e), (e, b), (b, d), (a, d), (c, b)\}$. Вершина c помечается как внешняя вершина дерева. Вершина g получает разметку внутренней, а вершина a — внешней вершины дерева. Ребра (c, g) и (g, a) окрашиваются в оранжевый цвет. (Так как g — внутренняя вершина дерева, то b, e, d также помечаются как внутренние вершины.) Дальнейшая разметка вершин невозможна. Полученное дерево является венгерским деревом. Переходим к шагу 5.

Шаг 5. Изменяются значения двойственных переменных:

$d_1 = \infty, \quad d_2 = \infty, \quad d_3 = 1/2\, z_g = 1, \quad d_4 = \min\{y_c, y_a\} = \min\{2, 3\} = 2,$

$d = \min\{\infty, \infty, 1, 2\} = 1.$

Таким образом, $y_c = 2 - 1 = 1$, $y_a = 3 - 1 = 2$, $y_b = 2 + 1 = 3$, $y_d = 2 + 1 = 3$, $y_e = 1 + 1 = 2$, $z_g = 2 - 2(1) = 0$.

Так как значение z_g стало равным нулю, то следует фиктивную вершину g преобразовать в нечетный цикл. Такое преобразование вершины g приводит к исходному графу, приведенному на рис. 5.11. Теперь мы должны найти паросочетание максимальной мощности на нечетном цикле, соответствующем вершине g. Поскольку ребро (d, a) принадлежит паросочетанию, то внешними для него являются только вершины e и b. Таким образом, ребро (e, b) включается в паросочетание.

Шаг 2. Для просмотра выбирается открытая вершина f. Строится чередующееся дерево: $E^* = \{(d, e,) (e, b), (b, d), (a, d), (b, c)\}$. Вершина f помечается как внешняя вершина дерева. Дальнейшая пометка вершин невозможна. Дерево, состоящее только из вершины f, является венгерским деревом. Переходим к шагу 5.

Шаг 5. Изменяются значения двойственных переменных:

$d_1 = y_f + y_e - a(f, e) = 3 + 2 - 3 = 2$, $d_2 = \infty$, $d_3 = \infty$, $d_4 = y_f = 3$,

$d = \min\{2, \infty, \infty, 3\} = 2$.

Таким образом, $y_f = 3 - 2 = 1$. Значения двойственных переменных остаются неизменными.

Шаг 2. Продолжается просмотр открытой вершины f. Строится чередующееся дерево: E^* состоит из всех ребер графа. Вершина f помечается как внешняя вершина дерева. Вершина e помечается как внутренняя и вершина b — как внешняя вершина дерева. Ребра (f, e) и (e, b) окрашиваются. Далее окрашивается ребро (b, c). Найдена увеличивающая чередующаяся цепь (f, e), (e, b), (b, c). Переходим к шагу 3.

Шаг 3. Исключается из паросочетания ребро (e, b), а ребра (f, e) и (b, c) включаются в него. Теперь паросочетание содержит ребра (a, d), (f, e) и (b, c).

Шаг 2. Не просмотренные открытые вершины отсутствуют. Переходим к шагу 6.

Шаг 6. В полученном последнем графе отсутствуют фиктивные вершины. Задача решена.

Заметим, что окончательные значения всех двойственных переменных $z_m = 0$ и $\Sigma y_i = 2 + 3 + 1 + 3 + 2 + 1 = 12$. Таким образом, значение целевой функции (16) прямой задачи равно 12 и значение целевой функции (10) двойственной задачи также равно 12. Значит, в соответствии с уравнением (4) из гл. 1 полученные решения прямой и двойственной задач должны быть оптимальными.

5.4. Алгоритм построения покрытия с минимальным весом

В этом разделе представлен алгоритм [7], с помощью которого на графе $G = (X, E)$ находится покрытие с минимальным весом. Основной процедурой в этом алгоритме, как и в алгоритмах выбора оптимальных паросочетаний, рассмотренных в разд. 5.2 и 5.3, является процедура построения чередующегося дерева.

Пусть, как и ранее, $V = \{V_1, V_2, \ldots, V_z\}$ — множество всех подмножеств множества X, состоящих из нечетного числа элементов. Пусть через $2n_m + 1$ обозначено число вершин в подмножестве V_m, а через U_m — множество таких ребер, что *хотя бы одна* из инцидентных им вершин принадлежит X_m. Справедливо следующее утверждение: *каждое покрытие должно содержать в* U_m *по крайней мере* $(n_m + 1)$*ребер.*

Пусть, как и ранее, $a(i, j)$ — вес ребра (i, j). Пусть $x(i, j) = 1$, если ребро (i, j) принадлежит покрытию; в противном случае $x(i, j) = 0$.

Предположим сначала, что веса всех ребер положительны. Рассмотрим следующую задачу линейного программирования: минимизировать

$$\sum_{(i,j)} a(i, j)\, x(i, j) \qquad (23)$$

при ограничениях

$$\sum_j [x(i, j) + x(j, i)] \geqslant 1 \text{ для всех } i \in X \tag{24}$$

$$\sum_{(i, j) \in U_m} x(i, j) \geqslant n_m + 1, \quad m = 1, 2, \ldots, z \tag{25}$$

$$x(i, j) \geqslant 0 \text{ для всех } (i, j). \tag{26}$$

В соответствии с ограничением (24) требуется, чтобы по крайней мере одно ребро, принадлежащее покрытию, было бы инцидентно вершине i. Ограничение (25) означает, что по крайней мере $(n_m + 1)$ ребер, принадлежащих множеству U_m, должно входить в покрытие.

Целевая функция (23) представляет собой общий вес покрытия. Очевидно, что любое покрытие удовлетворяет ограничениям (24) — (26). Задача, двойственная задаче (23) — (26), формируется следующим образом:
максимизировать

$$\sum_{i \in X} y_i + \sum_{m=1}^{z} (n_m + 1) z_m \tag{27}$$

при ограничениях

$$y_i + y_j + \sum_{m:\,(i, j) \in U_m} z_m \geqslant a(i, j) \text{ для всех } (i, j) \tag{28}$$

$$y_i \geqslant 0 \text{ для всех } i \in X \tag{29}$$

$$z_m \geqslant 0, \quad m = 1, 2, \ldots, z. \tag{30}$$

Условия дополняющей нежесткости для этой пары задач линейного программирования имеют следующий вид:

$$x(i, j) > 0 \Rightarrow y_i + y_j + \sum_{m:\,(i, j) \in U_m} z_m = a(i, j) \text{ для всех } (i, j), \tag{31}$$

$$y_i > 0 \Rightarrow \sum_{j \in X} [x(i, j) + x(j, i)] = 1 \text{ для всех } i \in X, \tag{32}$$

т. е. вершина i покрывается только одним ребром.

$$z_m > 0 \Rightarrow \sum_{(i, j) \in U_m} x(i, j) = n_m + 1, \quad m = 1, 2, \ldots, z. \tag{33}$$

Таким образом, $2n_m + 1$ вершин в X_m покрывается n_m такими ребрами, что обе инцидентные каждому из них вершины при-

надлежат X_m, и одним таким ребром, что одна инцидентная ему вершина принадлежит X_m.

Отметим, что двойственная переменная y_i соответствует ограничению (24) для вершины i, и, следовательно, y_i может рассматриваться как двойственная переменная для вершины i. Отметим также, что двойственная переменная z_m соответствует ограничению (25) для подмножества вершин нечетной мощности V_m и может рассматриваться как двойственная переменная для подмножества вершин V_m. Таким образом, имеются двойственные переменные, связанные с каждой вершиной графа и каждым подмножеством его вершин нечетной мощности.

Как и предыдущий алгоритм этой главы, алгоритм построения покрытия с минимальным весом будет стягивать нечетные циклы в фиктивные вершины.

Таким образом, каждая фиктивная вершина a_k будет содержать подмножество вершин X_k нечетной мощности. Двойственная переменная z_k, связанная с X_k, может рассматриваться как двойственная переменная для фиктивной вершины a_k. Из ограничений (29) — (30) следует, что

$$y_i \leqslant \min_j \{a(i, j), a(j, i)\}$$

Вершина i называется *насыщенной*, если имеет место $y_i = \min \{a(i, j), a(j, i)\}$, т. е. если y_i принимает наибольшее допустимое значение. В противном случае вершина i называется *ненасыщенной*. Если $y_i = 0$, то вершина i называется *пустой*. Отметим, что условие (32) не относится к пустым вершинам и, следовательно, только пустые вершины могут быть инцидентны более чем одному ребру, принадлежащему покрытию. Отметим также, что каждая насыщенная вершина должна быть соединена ребром с пустой вершиной.

Алгоритм построения покрытия с минимальным весом выполняется в два этапа. На первом этапе строится некоторое паросочетание; на втором этапе это паросочетание преобразуется в покрытие, которое является покрытием с минимальным весом.

В начале первого этапа предполагается, что все $x(i, j) = 0$, т. е. имеется паросочетание нулевой мощности, все двойственные переменные $z_m = 0$ и значения двойственных переменных y_i удовлетворяют ограничениям (28) и (29). На каждой итерации первого этапа исследуется внешняя открытая ненасыщенная вершина v. В результате выполнения итерации либо ребро, инцидентное вершине v, включается в паросочетание, либо двойственные переменные изменяются таким образом, что вершина v становится насыщенной.

Условия (28) — (31) и (33) остаются выполненными в процес-

се реализации первого этапа. Второй этап работы алгоритма начинается после исключения всех открытых ненасыщенных вершин. Паросочетание, полученное в результате выполнения первого этапа, преобразуется на втором этапе в покрытие путем включения в него ребер, каждое из которых соединяет насыщенную вершину с соответствующей пустой вершиной. Для полученного покрытия выполняются все ограничения прямой и двойственной задач, а также условия дополняющей нежесткости; следовательно, это покрытие является покрытием с минимальным весом.

Зададимся вопросом: как на первом этапе следует изменить паросочетание и (или) двойственные переменные, чтобы вершина перестала быть открытой или стала насыщенной? Это достигается построением чередующегося дерева[1] с корнем в вершине v с помощью описанного в разд. 5.2 алгоритма построения чередующегося дерева.

Как и ранее, алгоритм построения чередующегося дерева заканчивается одним из трех результатов:
(а) построением увеличивающей цепи,
(б) нахождением нечетного цикла,
(в) построением венгерского дерева.

В случае (а) ребра цепи, входившие в паросочетание, заменяются ребрами этой цепи, не входившими ранее в него. Вершина v становится инцидентной ребру паросочетания. В случае (б) полученный нечетный цикл стягивается в фиктивную вершину, и алгоритм построения чередующегося дерева продолжает свою работу на преобразованном графе, получаемом путем стягивания нечетного цикла в точку. В случае (в) двойственные переменные изменяются так, что (в1) к построенному чередующемуся дереву может быть добавлено новое ребро или (в2) становится насыщенной внешняя вершина дерева j.

В случае (в1) продолжается построение чередующегося дерева с корнем в вершине v. В случае же (в2) чередующаяся цепь от v к j образует нейтральную увеличивающую цепь. В паросочетании производится замена входящих в него ребер цепи ребрами этой же цепи, не входящими ранее в это паросочетание. В результате вершина v становится инцидентной ребру, входящему в паросочетание, а вершина j становится открытой насыщенной вершиной.

Таким образом, вершина v в конечном счете является либо инцидентной ребру паросочетания, либо насыщенной. Как уже говорилось, алгоритм построения чередующегося дерева стягивает нечетные циклы в фиктивные вершины. В дальнейшем фик-

[1] Чередующееся дерево строится только на ребрах, удовлетворяющих соотношению (28) как равенству.

тивные вершины с помощью алгоритма построения покрытия с минимальным весом преобразуются обратно в соответствующие исходные нечетные циклы. Однако порядок обратного преобразования этих вершин в циклы лишь частично соответствует порядку, в котором они формировались. Следовательно, алгоритм построения покрытия с минимальным весом имеет дело с последовательностью графов G_0, G_1, ..., G_t.

На основе полученных общих сведений мы теперь подготовлены к формальному изложению *алгоритма построения покрытия* с минимальным весом.

Алгоритм построения покрытия с минимальным весом

Этап 1. (*Выбор паросочетания.*)

Шаг 1. (*Выбор обозначений и начальных значений.*) Положить $k = 0$. Обозначить через $G_k = (X_k, E_k)$ рассматриваемый граф. Положить паросочетание M_k пустым (не содержащим ребер). Для сокращения записи формул вводится переменная

$$w_i = y_i + \sum_{n: i \in V_n} z_n \text{ для всех } i \in X_0 \tag{34}$$

Принять двойственные переменные $z_m = 0$, $m = 1, 2, \ldots, t$. Для двойственных переменных y_i выбрать любое значение, при котором выполнялись бы условия

$$w_i + w_j \leqslant a(i, j) \text{ для всех } (i, j) \in E_0 \tag{35}$$

$$w_i \geqslant 0 \text{ для всех } i \in X_0 \tag{36}$$

Например, $y_i = 0$ будет удовлетворять всем требованиям.

Шаг 2. (*Просмотр открытой ненасыщенной вершины.*) Пусть $E^* = \{(i, j) : w_i + w_j = a(i, j),\ i \in X_0,\ j \in X_0\}$. Выбрать любую, не являющуюся фиктивной, открытую ненасыщенную вершину $v \in X_k$, если таких вершин нет, то перейти к этапу 2. В противном случае, используя алгоритм построения чередующегося дерева, на множестве ребер E^* графа G_k построить чередующееся дерево с корнем в вершине v. Если выполнение алгоритма построения чередующегося дерева заканчивается отысканием увеличивающей цепи, перейти к шагу 3. Если работа этого алгоритма заканчивается отысканием нечетного цикла, перейти к шагу 4. Если алгоритм приводит к построению венгерского дерева, то перейти к шагу 5.

Шаг 3. (*Построение увеличивающей цепи.*) Этот шаг выполняется только после того, как с помощью алгоритма построения чередующегося дерева обнаруживается увеличивающая цепь. В паросочетании M_k заменить ребра увеличивающей цепи не

принадлежащими ему ребрами этой же цепи. Открытая корневая вершина v становится инцидентной ребру паросочетания. Вернуться к шагу 2 для исследования другой открытой ненасыщенной вершины.

Шаг 4. (*Построение нечетного цикла.*) Этот шаг выполняется только после того, как с помощью алгоритма построения чередующегося дерева найдется некоторый нечетный цикл. Положить $k = k + 1$. Стянуть найденный нечетный цикл в фиктивную вершину a_k. Граф, полученный в результате стягивания этого цикла в вершину, обозначить через G_k. Обозначить через M_k паросочетание, включающее все ребра M_{k-1}, оставшиеся в графе G_k.

Вершины, входящие в стянутый нечетный цикл при всех последующих разметках по алгоритму построения чередующегося дерева, получают те же метки, что и фиктивная вершина a_k. Вернуться к шагу 2 и продолжить построения чередующегося дерева с корнем в вершине, являющейся отображением вершины v в графе G_k^1. В этом случае метки и окраска ребер последней итерации могут оказаться полезными на следующей итерации алгоритма построения чередующегося дерева.

Шаг 5. (*Венгерское дерево.*) Этот шаг выполняется только после того, как работа алгоритма построения чередующегося дерева заканчивается построением венгерского дерева.

Определить

$$d_1 = \min \{a(i, j) - w_i - w_j\}, \tag{37}$$

где минимум берется по всем внешним вершинам дерева $i \in X_0$ и всем непомеченным вершинам $j \in X_0$.

Определить

$$d_2 = {}^1/_2 \min \{a(i, j) - w_i - w_j\}, \tag{38}$$

где минимум берется по всем таким внешним вершинам дерева $i \in X_0$ и $j \in X_0$, что $(i, j) \notin E^*$.

Определить

$$d_3 = {}^1/_2 \min \{z_m\}, \tag{39}$$

где минимум берется по всем m, таким, что вершины подмножества V_m представляют внутреннюю фиктивную вершину дерева.

Определить

$$d_4 = \min \{a(i, j) - w_i\}, \tag{40}$$

где минимум берется по всем таким нефиктивным внешним вер-

[1] Это может быть только тогда, когда фиктивная вершина является корнем чередующегося дерева.

шинам дерева $j \in X_k$ и всем таким внутренним вершинам дерева $j \in X_0$, что $(i, j) \in E^*$.

Определить

$$d_5 = \min \left\{ w_i - \sum_{m: i \in V_m} z_m \right\}, \tag{41}$$

где минимум берется по вершинам $i \in X_0$, которые входят в нечетный цикл, стянутый во внешнюю фиктивную вершину дерева.

И наконец, определить

$$d = \min \{d_1, d_2, d_3, d_4, d_5\}. \tag{42}$$

Изменить двойственные переменные следующим образом:

1. Переменные w_i соответствующие внешней вершине дерева, увеличить на d.

2. Переменные, соответствующие внутренней вершине дерева, уменьшить на d.

3. Увеличить на $2d$ переменную z_m, соответствующую каждой внешней фиктивной вершине дерева в графе G_k.

4. Уменьшить на $2d$ двойственную переменную, соответствующую каждой внутренней фиктивной вершине дерева в графе G_k.

При этом если $d = d_1$, то к E^* добавляется ребро и процесс построения чередующегося дерева может быть продолжен. Вернуться к шагу 2 для продолжения построения чередующегося дерева с корнем в вершине v.

Если $d = d_2$, то к E^* добавляется ребро, которое формирует нечетный цикл. Вернуться к шагу 2 для продолжения построения чередующегося дерева с корнем в вершине v.

Если $d = d_3$, то значение двойственной переменной для некоторой внутренней фиктивной вершины дерева снова становится равным нулю. Положить $k = k + 1$. Преобразовать обратно эту фиктивную вершину в исходный нечетный цикл. Она инцидентна одному из ребер паросочетания, так как имела метку внутренней вершины дерева. Включить в паросочетание ребра нечетного цикла таким образом, чтобы все вершины этого цикла были инцидентны ребрам паросочетания. Принять полученный новый граф за G_k и новое паросочетание за M_k. Вернуться к шагу 2 для продолжения построения чередующегося дерева с корнем к вершине v.

Если $d = d_4$, то некоторая внешняя вершина j дерева становится насыщенной. Тогда чередующаяся цепь в чередующемся дереве, соединяющая корень v с вершиной j, является нейтральной увеличивающей цепью. Произвести замену ребер этой цепи, входивших в паросочетание, не входящими в него ребрами этой же цепи. Вершина j при этом становится открытой и насыщенной, а вершина v — инцидентной ребру, принадлежащему паросоче-

танию. Вернуться к шагу 2 и просмотреть другую открытую ненасыщенную вершину.

Если $d = d_5$, то y_i становится равным нулю для некоторой вершины i, содержащейся во внешней фиктивной вершине дерева a_m. Чередующаяся цепь от корня v до вершины a_m является нейтральной увеличивающей цепью. Произвести замену ребер цепи от v до a_m, входящих в паросочетание, не входящими в него ребрами этой же цепи. Вершина v становится инцидентной ребру, принадлежащему паросочетанию, а фиктивная вершина a_m — открытой. Вернуться к шагу и просмотреть другую открытую ненасыщенную вершину.

Этап 2. (*Преобразование паросочетания в покрытие.*)

Шаг 6. (*Открытые нефиктивные вершины.*) Этот шаг выполняется только после удаления на шаге 2 из графа G_k всех открытых ненасыщенных нефиктивных вершин. Для его выполнения необходимо включить в M_k ребра, соединяющие в G_k каждую открытую насыщенную нефиктивную вершину с пустой вершиной. (Пустая вершина не содержится ни в одной из фиктивных вершин, инцидентных ребру паросочетания. См. лемму 5.2.) Теперь открытыми в G_k являются только фиктивные вершины. Перейти к шагу 7.

Шаг 7. (*Обратное преобразование фиктивных вершин в нечетные циклы.*) Если в графе G_k отсутствуют фиктивные вершины, то M_k является покрытием с минимальным весом для графа G_0. В противном случае последнюю из оставшихся фиктивных вершин a_m обратно преобразовать в исходный нечетный цикл. Положить $k = k + 1$. Граф, полученный в результате преобразования фиктивной вершины, считать графом G_k.

Здесь возможны два случая: а) в результате выполнения первого этапа алгоритма вершина a_m была покрыта ребром графа; б) в результате выполнения первого этапа алгоритма вершина a_m оказалась открытой. В случае (а) одна из вершин в цикле, соответствующем фиктивной вершине a_m, инцидентна ребру, принадлежащему паросочетанию, а остальные вершины в этом цикле являются открытыми. В M_k включаются все ребра, принадлежащие M_{k-1}, совместно с ребрами нечетного цикла, стянутого в a_m, которые инцидентны входящим в него открытым вершинам. Повторить шаг 7. В случае (б) для некоторой вершины i нечетного цикла, соответствующего фиктивной вершине a_m, $y_i = 0$ (см. лемму 5.1). Положить M_k содержащим ребра, принадлежащие M_{k-1}, совместно с обоими ребрами нечетного цикла, инцидентными вершине i, и остальными ребрами этого нечетного цикла, принадлежащим паросочетанию. Повторить шаг 7.

Прежде чем приступить к доказательству того, что описанный алгоритм позволяет построить покрытие с минимальным весом, необходимо доказать две леммы.

ЛЕММА 5.1. В результате выполнения первого этапа алгоритма каждый нечетный цикл, соответствующий открытой фиктивной вершине, включает некоторую пустую вершину.

Доказательство. Пусть a_m — любая фиктивная открытая вершина, полученная в результате выполнения первого этапа алгоритма. В момент образования фиктивной вершины она была либо открытой, либо инцидентной ребру, принадлежащему паросочетанию. Если вершина a_m была открытой, то она являлась корнем строящегося чередующегося дерева. Это чередующееся дерево строилось до тех пор, пока a_m либо не стала инцидентной ребру, принадлежащему паросочетанию, либо одна из вершин, входящих в соответствующий ей нечетный цикл, не стала пустой. (См. шаг 5, случай $d = d_5$.)

Если a_m стала инцидентной ребру, принадлежащему паросочетанию, то она может стать открытой только благодаря тому, что одна из вершин, входящих в соответствующий ей нечетный цикл, станет пустой. (См. шаг 5, случай $d = d_5$.) Наконец, если a_m совместно с вершинами некоторого нечетного цикла стягивается в другую фиктивную вершину a_n, то a_n к концу выполнения первого этапа алгоритма должна быть обратно преобразована в нечетный цикл. В результате такого преобразования a_n фиктивная вершина a_m оказывается инцидентной ребру, принадлежащему паросочетанию (См. шаг 5, случай $d = d_3$) Лемма доказана.

ЛЕММА 5.2. Пустая вершина, полученная в результате выполнения первого этапа алгоритма и смежная с насыщенной открытой вершиной, не содержится ни в одной из фиктивных вершин, инцидентных ребру паросочетания.

Доказательство. Предположим, что после выполнения первого этапа алгоритма вершина i является насыщенной и открытой, а вершина j — пустой вершиной. Для доказательства леммы мы должны показать, что вершина j не содержится ни в одном из нечетных циклов, которые стянуты в фиктивные вершины, инцидентные ребрам, принадлежащим паросочетанию.

Поскольку $y_i = a(i, j)$, $y_i = 0$ и $\Sigma z_m = 0$, то из (28) и (34) следует, что $\sum_{m:j \in V_m} z_m = 0$. Таким образом, значения двойственных переменных равны нулю для любой фиктивной вершины, содержащей вершину j. Рассмотрим тот шаг процесса присвоения меток, на котором фиктивная вершина a_m была помечена в последний раз. Если бы вершина a_m получила метку внутренней вершины дерева, то она на пятом шаге алгоритма должна была бы быть преобразована обратно в нечетный цикл, так как на указанном шаге 5 все фиктивные вершины с нулевыми значениями двойственных переменных преобразуются в соответствующие им нечетные циклы.

Если бы вершина a_m получила метку внешней вершины дерева, то чередующаяся цепь, соединяющая корень дерева с вершиной a_m, была бы нейтральной увеличивающей цепью и a_m стала бы открытой. (См. шаг 5, случай $d = d_5$.) Следовательно, фиктивная вершина a_m никогда не может быть инцидентна ребру, принадлежащему паросочетанию. Лемма доказана.

Доказательство оптимальности алгоритма построения покрытия с минимальным весом. На первом этапе алгоритм строит некоторое паросочетание. На втором этапе это паросочетание преобразуется в покрытие для графа G_0. Нам достаточно показать, что полученное в результате выполнения алгоритма покрытие является покрытием с минимальным весом. Для этого в свою очередь достаточно показать, что окончательные значения двойственных переменных y_i и z_m и значения прямых переменных для полученного покрытия удовлетворяют ограничениям (28) — (30) и условиям дополняющей нежесткости (31) — (33).

Докажем, допустимость полученных значений y_i, z_m и $x(i, j)$ для каждого из ограничений.

Ограничение (28) может быть переформулировано в виде

$$y_i + y_j + \sum_{k:\,(i,\,j)\in U_k} z_k = y_i + y_j + \sum_{m:\,i\in V_m} z_m + \sum_{m:\,j\in V_m} z_m -$$

$$- \sum_{m:\,i,\,j\in V_m} z_m = w_i + w_j - \sum_{m:\,i,\,j\in V_m} z_m \leqslant a(i, j) \qquad (28')$$

Первоначально все $z_m = 0$ и $w_i + w_j \leqslant a(i,j)$. Условие (28′) удовлетворяется как строгое неравенство для всех ребер, не принадлежащих множеству E^*. После включения ребра (i, j) в E^* вершины i и j (а) либо имеют различные метки, (б) либо содержатся в одной и той же фиктивной вершине, (в) либо не имеют меток вообще.

В случае (а) при изменении значений двойственных переменных изменение значения y_i компенсируется противоположным изменением значения y_i и $z_m = 0$ для всех m, для которых $i \in V_m$ и $j \in V_m$. В случае (б) значения w_i и w_j изменяются на величину d и значение двойственной переменной z фиктивной вершины, включающей вершины i и j, изменяется в противоположном направлении на $2d$. В случае (в) значения w_i, w_j и z_m $i \in V_m$, $j \in V_m$ остаются неизменными. Следовательно, условие (28) при принятых изменениях двойственных переменных всегда остается выполненным.

Доказательство для условия (29). Из выражения (41) для определения d_5 следует, что в процессе выполнения алгоритма всегда поддерживается соотношение.

$$w_i \geqslant \sum_{m: i \in V_m} z_m.$$

Следовательно, $y_i \geqslant 0$ на всех итерациях алгоритма.

Доказательство для условия (30). Из уравнения (39) следует, что z_k никогда не может уменьшиться до отрицательной величины.

Доказательство для условия (31). Ребро (i, j) включается в полученное покрытие с помощью алгоритма в одном из следующих трех случаев: (а) на первом этапе, (б) на шаге 6, (в) на шаге 7.

В случае (а) ребро (i, j) принадлежит множеству E^*, а вершины i и j к концу выполнения первого этапа не принадлежат одной и той же фиктивной вершине. Таким образом, $\sum_{m: i,j \in V_m} z_m = 0$ и из (28′) следует, что условие (31) выполняется.

В случае (б) вершина i насыщена, а j — пустая, и обе вершины к концу выполнения первого этапа не являются фиктивными. Следовательно, $y_i = a(i, j)$, а $y_i = 0$ и условие (3) выполняется.

В случае (в) ребро (i, j) к концу выполнения первого этапа содержится в фиктивной вершине. Следовательно, ребро $(i, j) \in E^*$ и условие (31) выполняется.

Доказательство для условия (32). К концу выполнения первого этапа все вершины насыщены или инцидентны ребрам, входящим в паросочетание. На шагах 6 и 7 более одного раза покрываются ребрами только пустые вершины.

Следовательно, только пустые вершины в покрытии инцидентны более чем одному ребру, а все другие вершины инцидентны точно одному ребру.

Доказательство для условия (33). Если к концу выполнения первого этапа $z_m > 0$, то вершины, входящие в множество V_m, стягиваются в фиктивную вершину a_t. Соответствующие нечетному циклу вершины множества V_t на шаге 7 покрываются n_t ребрами, входящими в этот нечетный цикл, и еще одним дополнительным ребром. Таким образом, в полученном покрытии присутствует $n_t + 1$ ребер, входящих в множество U_m, и, следовательно, выполнено условие (33). Что и требовалось доказать.

ПРИМЕР 4. Пусть на графе, представленном на рис. 5.13, необходимо найти покрытие с минимальным весом; рядом с каждым ребром указан его вес.

Шаг 1. (Выбор начальных значений). Пусть $z_m = 0$ для

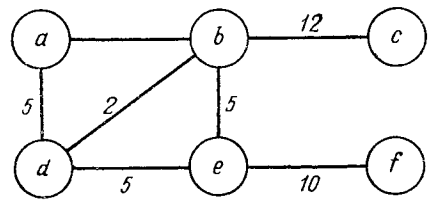


Рис. 5.13. Пример применения алгоритма поиска покрытия с минимальным весом.

$m = 1, 2, \ldots, z$ и все $w_i = {}^1/_2 \min\{a(i, j)\} = 1$. Первоначально паросочетанию не принадлежит ни одного ребра.

Шаг 2. (*Просмотр открытой, ненасыщенной вершины d.*) Алгоритм построения чередующегося дерева: $E^* = \{(d, b)\}$. Вершина d получает метку внешней вершины дерева. Ребро (d, b) окрашивается. Увеличивающая цепь найдена.

Шаг 3. Ребро (d, b) включается в паросочетание.

Шаг 2. (*Просмотр открытой ненасыщенной вершины e.*) Строится чередующееся дерево: $E^* = \{(d, b)\}$. Вершина e получает метку внешней вершины дерева. Другие вершины не могут быть помечены. Таким образом, построено венгерское дерево. Перейти к шагу 5.

Шаг 5. Производится изменение двойственных переменных:

$$d_1 = \min\{a(e, b) - w_e - w_b,\ a(e, d) - w_e - w_d,\ a(e, f) - w_e - w_f\} =$$
$$= \min\{5 - 1 - 1,\ 5 - 1 - 1,\ 10 - 1 - 1\} = 3,$$

$$d_2 = d_3 = d_4 = d_5 = \infty,$$

$$d = \min\{d_1, d_2, d_3, d_4, d_5\} = 3.$$

В результате получаем $w_e = 1 + 3 = 4$. Двойственные переменные, соответствующие остальным вершинам, остаются неизменными (табл. 5.2).

Таблица 5.2

Изменения двойственных переменных в алгоритме определения покрытия с минимальным весом

	w_a	w_b	w_c	w_d	w_e	w_f	Z_1	Множество ребер
Начальные значения	1	1	1	1	1	1	0	Пустое
Просмотр d	1	1	1	1	1	1	0	(d, b)
Просмотр e	1	1	1	1	4	1	0	Пустое
Просмотр a_1	1	2	1	2	5	1	2	»
Просмотр a	3	2	1	2	5	1	2	(a, a_1)
Просмотр c	3	2	10	2	5	1	2	(a, a_1)
Повторный просмотр c	4	1	11	1	4	1	0	(a, d), (b, c)
Повторный просмотр c	5	0	12	0	5	1	0	(a, d), (b, c)
Просмотр f	5	0	12	0	5	5	0	(a, q), (b, c), (e, f)
Стадия 2	5	0	12	0	5	5	0	(a, d), (b, c), (e, f)

Шаг 2. (*Продолжение просмотра вершины e.*) Строится чередующееся дерево: $E^* = \{(d, b), (e, b), (d, e)\}$. Вершина e получает метку внешней вершины дерева. Ребра (e, b) и (b, d) окрашиваются, вершина b (получает метку внутренней, а вершина d — внешней вершин дерева. Ребро (d, e) также окрашивается, поскольку вершины d и e являются внешними вершинами дерева. Следовательно, найден нечетный цикл, состоящий из ребер (e, b), (b, d) (d, e). Переходим к шагу 4.

Шаг 4. Нечетный цикл (e, b), (b, d), (d, e) стягивается в фиктивную вершину a_1 (рис. 5.14). Пусть $z_1 = 0$ — двойственная переменная, соответствующая фиктивной вершине a_1. Теперь в паросочетание не входит ни одно из ребер преобразованного графа.

Шаг 2. (*Просмотр открытой фиктивной вершины* a_1.) Строится чередующееся дерево: $E^* = \{(e, b), (b, d), (d, e)\}$. Вершина a_1 получает метку внешней вершины дерева, и, следовательно, вершины e, b, d должны иметь такую же метку. Другие вершины не могут быть помечены. Дерево, состоящее только из вершины a_1, является венгерским деревом. Переходим к шагу 5.

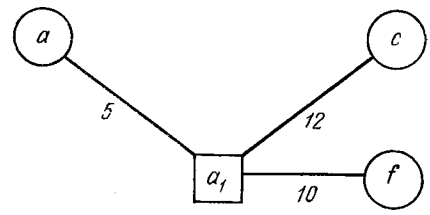


Рис. 5.14. Граф, полученный в результате стягивания в вершину нечетного цикла.

Шаг 5. Изменяются значения двойственных переменных:

$$d_1 = \min\{a(e, f) - w_e - w,\ a(b, c) - w_b - w_e,\ a(a, d) - w_a - w_d\} =$$

$$= \min\{10 - 1 - 1,\ 12 - 1 - 1,\ 5 - 1 - 1\} = 3,$$

$$d_2 = d_3 = d_4 = \infty,$$

$$d_5 = \min\{w_e, w_b, w_d\} = \min\{4, 1, 1\} = 1,$$

$$d = \min\{d_1, d_2, d_3, d_4, d_5\} = 1.$$

Значения двойственных переменных становятся равными $w_l = 4 + 1 = 5$. $w_b = 1 + 1 = 2$, $w_d = 1 + 1 = 2$, $z_1 = 0 + 2 \cdot 1 = 2$. Так как $d = d_5$, вершина a_1 может быть оставлена открытой. Заметим, что $y_b = w_b - z_1 = 2 - 2 = 0$ и $y_d = w_d - z_1 = 2 - 2 = 0$. Переходим к шагу 2.

Шаг 2 (*Просмотр открытой ненасыщенной вершины* a.) Строится чередующееся дерево: $E^* = \{(e, b), (b, d), (d, e)\}$. Вершина a получает метку внешней вершины. Другие вершины не могут быть помечены. Полученное дерево состоит из одной вершины и является венгерским деревом. Переходим к шагу 5.

Шаг 5. Изменяются значения двойственных переменных:

$$d_1 = \min\{a(a, d) - w_a - w_d\} = 5 - 1 - 2 = 2,$$

$$d_2 = d_3 = d_4 = d_5 = \infty,$$

$$d = d_1 = 2.$$

Значение двойственной переменной $w_a = 1 + 2 = 3$. Переходим к шагу 2.

Шаг 2. Продолжается просмотр открытой ненасыщенной вершины a. Строится чередующееся дерево: $E^* = \{(d, b), (e, b), (d, e), (a, d)\}$. Вершина a получает метку внешней вершины дерева. Ребро (a, a_1) окрашивается. Найдена увеличивающая цепь (a, a_1). Это ребро включается в паросочетание. Повторяется шаг 2.

Шаг 2 (*Просмотр открытой ненасыщенной вершины* c.) Строится чередующееся дерево: E^* совпадает с E^* на предыдущем шаге. Вершина c получает метку внешней вершины дерева. Другие вершины не могут быть помечены. Полученное дерево состоит из одной вершины c и является венгерским деревом. Переходим к шагу 5.

Шаг 5. Изменяются значения двойственных переменных:

$$d_1 = \min\{a(c, b) - w_c - w_b\} = 12 - 1 - 2 = 9,$$

$$d_2 = d_3 = d_4 = d_5 = \infty, \quad d = d_1 = 9.$$

Значение двойственной переменной $w_c = 1 + 9 = 10$. Значения остальных двойственных переменных не изменяются. Переходим к шагу 2.

Шаг 2. Продолжается просмотр открытой ненасыщенной вершины.

Строится чередующееся дерево: $E^* = \{(d, b), (e, b), (d, e), (a, d), (c, b)\}$. Вершины c и a получают метки внешних, а вершина a_1 — внутренней вершин дерева. Ребра (c, b) и (a_1, a) окрашиваются. Другие вершины не могут быть помечены. Полученное дерево, состоящее из ребер (c, b) и (a_1, a), является венгерским деревом. Переходим к шагу 5.

Шаг 5. Изменяются значения двойственных переменных:

$$d_1 = d_2 = d_3 = \infty,$$

$$d_3 = 1/2 \min \{z_1\} = 1/2 \cdot 2 = 1,$$

$$d_4 = \min \{a(c, b) - w_c,\ a(a, d) - w_a\} = \min \{12 - 10,\ 5 - 3\} = 2,$$

$$d_5 = d_3 = 1.$$

Значения двойственных переменных становятся равными:

$$w_a = 3 + 1 = 4, \qquad w_e = 5 - 1 = 4,$$

$$w_b = 2 - 1 = 1, \qquad w_f = 1,$$

$$w_c = 10 + 1 = 11, \qquad z_1 = 2 - 2(1) = 0.$$

$$w_d = 2 - 1 = 1,$$

Так как $d = d_3$, z_1 становится равным нулю, и фиктивная вершина a_1 может быть преобразована обратно в исходный нечетный цикл. Полученный в результате такого преобразования граф совпадает с исходным графом, представленным на рис. 5.13. В паросочетание наряду с включенным в него ранее ребром (a, d) включается ребро (b, e), инцидентное двум оставшимся открытым вершинам b и e нечетного цикла (b, d), (d, e), (e, b). Переходим к шагу 2.

Шаг 2. Продолжается просмотр открытой ненасыщенной вершины c.

Строится чередующееся дерево: E^* включает все ребра графа, кроме (e, f). Вершины c и e получают метки внешних, а вершина b — внутренней вершин дерева. Ребра (c, b) и (b, e) окрашиваются. Далее вершина получает метку внутренней, а вершина a — внешней вершин дерева. Ребра (e, d) и (d, a) также окрашиваются. Оставшаяся вершина не может быть помечена. Полученное дерево состоит из ребер (c, d), (b, e), (e, d), (d, a) и является венгерским деревом. Переходим к шагу 5.

Шаг 5. Изменяются значения двойственных переменных:

$$d_1 = \min \{a(e, f) - w_e - w_f\} = 10 - 4 - 1 = 5,$$

$$d_2 = d_3 = d_5 = \infty,$$

$$d_4 = \min \{a(c, b) - w_c,\ a(e, b) - w_e,\ a(e, d) - w_e,\ (a, d) - w_a\} =$$

$$= \min \{12 - 11,\ 5 - 4,\ 5 - 4,\ 5 - 4\} = 1.$$

Шаг 2. Значения двойственных переменных становятся равными

$$w_a = 4 + 1 = 5, \quad w_b = 1 - 1 = 0, \quad w_c = 11 + 1 = 12,$$

$$w_d = 1 - 1 = 0, \quad w_e = 4 + 1 = 5, \quad w_f = 1, \quad z_1 = 0.$$

Вершина c становится насыщенной, и в дальнейшем нет необходимости ее просматривать.

Шаг 2. (*Просмотр открытой вершины f.*) Строится чередующееся дерево: $E^* = \{(a, d), (d, e), (e, b), (b, c)\}$. Вершина f получает метку внешней

вершины дерева. Остальные вершины графа не могут быть помечены. Построенное дерево состоит из одной вершины f и является венгерским деревом. Переходим к шагу 5.

Шаг 5. Изменяются значения двойственных переменных:

$$d_1 = a(f, e) - w_f - w_e = 10 - 1 - 5 = 4$$

$$d_2 = d_3 = d_4 = d_5 = \infty,$$

$$d = d_1 = 4.$$

Значение двойственной переменной $w_f = 1 + 4 = 5$.

Шаг 2. Продолжается просмотр открытой ненасыщенной вершины f. Строится чередующееся дерево: $E^* = \{(a, d), (d, e), (e, b), (b, c), (e, f)\}$. Вершины f и b получают метки внешних, а вершина e — внутренней вершин дерева. Ребра (f, e) и (e, b) окрашиваются. Далее получает метку вершина c и окрашивается ребро (b, c). В результате построена увеличивающая цепь (f, e), (e, b), (b, c). Переходим к шагу 3.

Шаг 3. В паросочетание включаются ребра (f, e) и (b, c). Из паросочетания исключается ребро (e, b). В результате получается паросочетание, состоящее из ребер (a, d), (f, e), (b, c). Переходим к шагу 2.

Шаг 2. Открытые вершины отсутствуют. Переходим к этапу 2.

Этап 2. Фиктивные вершины отсутствуют. Работа алгоритма заканчивается, так как полученный набор ребер является покрытием с минимальным весом.

Вес покрытия, состоящего из ребер (a, d), (f, e), (b, c), равен $5 + 12 + 10 = 27$. Получаемое в результате значение целевой функции (27) двойственной задачи равно $\Sigma y_i = \Sigma w_i = 5 + 0 + 12 + 0 + 5 + 5 = 27$. Таким образом, значения целевых функций прямой и двойственной задач линейного программирования одинаковы, поэтому в соответствии с уравнением (4) гл. 1 полученные решения прямой и двойственной задач должны быть оптимальными.

Отрицательные веса ребер

При рассмотрении алгоритма построения покрытия с минимальным весом предполагалось, что веса всех ребер являются неотрицательными числами. В противном случае оптимальное решение задачи линейного программирования (23) — (26), для которой разработан рассмотренный выше алгоритм, достигалось бы при $x(i, j) = \infty$, когда $a(i, j) < 0$.

Очевидно, что любое ребро, имеющее отрицательный вес, должно входить в прикрытие с минимальным весом. Кроме того, каждое ребро нулевого веса может быть также включено в покрытие с минимальным весом. Как же найти покрытие с минимальным весом, когда в графе присутствуют «отрицательные» ребра? Для его отыскания достаточно приписать нулевое значение веса ребрам, имеющим отрицательный вес. Затем с помощью изложенного выше алгоритма находится покрытие с минимальным весом на графе с неотрицательными значениями весов ребер. В полученное покрытие следует включить все не вошедшие в него ребра, имевшие до преобразования отрицательные значения веса. Покрытие, полученное в результате такого включения ребер с отрицательным значением веса, является по-

крытием с минимальным весом. Заметим, что если $a(i, j) = 0$, то в процессе выполнения алгоритма постоянны соотношения $w_i = 0$, $w_j = 0$, $w_i + w_j = 0$ и, следовательно, постоянно $(i, j) \in E^*$. Кроме того, вершины i и j на каждом шаге алгоритма являются насыщенными и не должны подвергаться просмотру на шаге 2.

УПРАЖНЕНИЯ

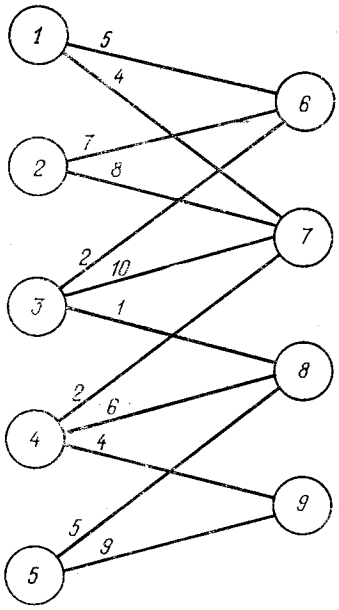


Рис. 5.15.

1. Построить граф, в котором паросочетание максимальной мощности не является паросочетанием с максимальным весом. Построить граф, в котором покрытие минимальной мощности не является покрытием с минимальным весом.

2. Для двудольного графа, представленного на рис. 5.15, построить:
   а) паросочетание максимальной мощности,
   б) паросочетание с максимальным весом,
   в) покрытие минимальной мощности,
   г) покрытие с минимальным весом.

3. Для графа, представленного на рис. 5.16, построить:
   а) паросочетание максимальной мощности,
   б) паросочетание с максимальным весом,
   в) покрытие минимальной мощности,
   г) покрытие с минимальным весом.

4. Каким образом алгоритм построения паросочетания максимальной мощности удалит ошибочно включенное в решение ребро графа? Каким образом алгоритм построения паросочетания с максимальным весом удалит ошибочно включенное в решение ребро графа? Каким образом это будет достигнуто с помощью алгоритма построения покрытия с минимальным весом?

5. Туристская группа из тридцати человек должна быть размещена в гостинице. Каждый номер гостиницы имеет две двуспальные кровати. Администратор гостиницы желает распределить гостей в возможно меньшем числе номеров таким образом, чтобы в одном номере не находились лица противоположного пола, не связанные родственными узами. Допускаются сочетания соседей по номеру типа муж — жена, отец — дочь, брат — сестра. Каким образом администратору гостиницы решить стоящую перед ним задачу?

6. В чем состоят основные различия и сходства алгоритма построения паросочетания с максимальным весом и первого этапа алгоритма построения покрытия с минимальным весом?

7. На участке металлообработки имеется шесть сверлильных станков. В определенный день появилась необходимость в выполнении на них пяти

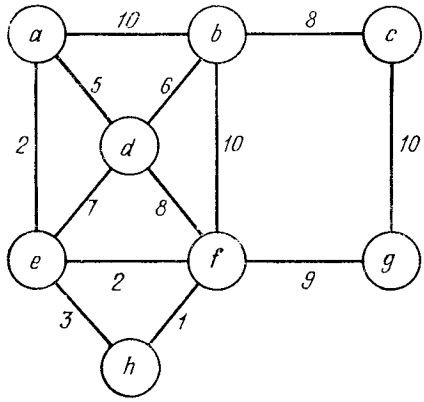


Рис. 5.16.

работ. Трудозатраты (в человеко-часах), связанные с выполнением каждой из работ на каждом станке, приведены в таблице.

Станок \ Работа	А	Б	В	Г	Д
1	5	7	6	4	9
2	8	10	3	4	7
3	6	11	5	4	7
4	5	8	7	3	9
5	3	6	4	2	7
6	3	7	5	3	7

Необходимо назначить каждую из работ станкам таким образом, чтобы на каждом станке выполнялась одна работа и при этом общее время, затраченное на выполнение всех работ, было минимальным.

8. ООН субсидирует программы сотрудничества городов-побратимов; согласно этим программам, города разделяются на пары для осуществления между ними обмена в области культуры и образования.

В текущем году привлечено для этих целей 10 новых городов. Каким образом разделить эти 10 городов на пары так, чтобы минимизировалось суммарное расстояние между городами-побратимами.

Оценки расстояний между городами представлены в приводимой таблице.

От города \ До города	1	2	3	4	5	6	7	8	9	10
1	0	80	70	70	60	45	90	110	85	155
2		0	75	95	90	80	90	160	70	45
3			0	65	70	60	100	80	80	55
4				0	80	80	70	170	200	250
5					0	110	170	190	270	300
6						0	100	150	110	200
7							0	75	95	100
8								0	90	100
9									0	50
10										0

ЛИТЕРАТУРА

1. Balinski M., Labelling to Obtain a Maximum Matching, Combinatorial Mathematics and Its Applications (Bose and Dowling, eds.), University of North Caroline Press, Chapel Hill, pp. 585 — 602, 1969.
2. Berge C. Two Theorems in Graph Theory, Proc. Natl. Acad. Sci. USA, vol. 43, pp. 842 — 844, 1957.
3. Brown J. R., Maximum Cardinality Matching, Kent State University, не опубликовано.
4. Edmonds J., Paths, Trees and Flowers, Can. J. Math., vol. 17, pp. 449 — 467, 1965.
5. Edmonds J., Maximum Matching and Polyhedra with 0-1 Vertices, J. Res. N. B. S., vol. 69B, No. 1, 2, pp. 125 — 130, 1965.
6. Edmonds J., Johnson Ellis, Matching: A Well Solved Class of Integer Linear Programs, Combinatorial Structures and Their Applications, Gordon and Breach, New York, pp. 89 — 92, 1970.
7. White L. J., A Parametric Study of Matchings and Coverings in Weighted Graphs, Ph. D. Thesis, University, of Michigan, 1967.

ГЛАВА 6

Задача почтальона

6.1. Введение

Любой почтальон, перед тем как отправиться в путь, должен подобрать на почте письма, относящиеся к его участку, после этого он должен их доставить адресатам, располагающимся вдоль маршрута его следования, и вернуться на почту, чтобы возвратить оставшуюся неврученной корреспонденцию. Каждый почтальон, желая расходовать меньше сил, хотел бы покрыть свой маршрут кратчайшим путем. Короче говоря, задача почтальона заключается в том, чтобы пройти все улицы маршрута и вернуться в его начальную точку, минимизируя при этом длину пройденного пути.

Очевидно, что такая задача стоит не только перед почтальоном, но и при выполнении многих видов курьерских заданий. Например, полицейскому хотелось бы знать наиболее эффективный путь патрулирования улиц своего района, фермеру было бы весьма полезно знать наилучший маршрут движения сельскохозяйственных машин по его полям при посеве, бригада ремонта заинтересована в выборе наилучшего пути своего перемещения по всем дорогам.

Первая публикация, посвященная решению подобной задачи, появилась в одном китайском журнале, где она была названа задачей почтальона. Иногда ее называют задачей китайского почтальона.

Задача почтальона может быть переформулирована в терминах теории графов. Для этого построим граф $G = (X, E)$, в котором каждое ребро соответствует улице в маршруте движения почтальона, а каждая вершина — стыку двух улиц. Эта задача представляет собой задачу поиска кратчайшего маршрута, включающего каждое ребро по крайней мере один раз и заканчивающегося в вершине движения.

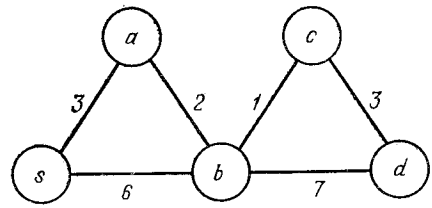


Рис. 6.1. Граф, в котором существует эйлеров маршрут.

Пусть s — начальная вершина маршрута и $a(i, j) > 0$ — длина ребра (i, j). В

графе, представленном на рис. 6.1, имеется несколько путей, по которым почтальон может обойти все ребра и вернуться обратно в вершину s. Сформулированным требованиям удовлетворяет, например, каждый из приведенных ниже четырех маршрутов:
Маршрут 1: (s, a), (a, b), (b, c), (c, d), (d, b), (b, s)
Маршрут 2: (s, a), (a, b), (b, d), (d, c), (c, b), (b, s)
Маршрут 3: (s, b), (b, c), (c, d), (d, b), (b, a), (a, s)
Маршрут 4: (s, b), (b, d), (d, c), (c, b), (b, a), (a, s)
В любой из четырех указанных маршрутов каждое ребро входит только один раз. Таким образом, общая длина каждого маршру-

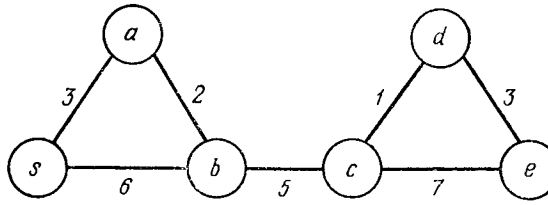


Рис. 6.2. Граф, в котором отсутствует эйлеров маршрут.

та равна $3 + 2 + 1 + 3 + 7 + 6 = 22$. Лучших маршрутов для почтальона не существует. Маршрут, в котором каждое ребро обходится точно один раз, называется эйлеровым маршрутом (по имени выдающегося математика Леонарда Эйлера, упомянутого в гл. 1 в связи с задачей о Кенигсбергских мостах).

Рассмотрим граф, представленный на рис. 6.2. Очевидно, что на нем отсутствуют пути почтальона, в которые бы ребро (b, c) входило только один раз, т. е. отсутствует эйлеров маршрут. Возможным оптимальным маршрутом (т. е. маршрутом наименьшей длины) на данном графе является путь (s, a), (a, b), (b, c) (c, d), (d, e), (e, c), (c, b), (b, s). Общая длина этого маршрута равна $3 + 2 + 5 + 1 + 3 + 7 + 5 + 6 = 32$. Попробуем установить, существуют ли какие-либо другие оптимальные маршруты почтальона на этом графе? Очевидно, что в несвязном графе (т. е. графе, имеющем несколько компонент) не существует маршрута почтальона (не говоря уже о существовании эйлерова маршрута). Поэтому в дальнейшем в этой главе мы будем всегда предполагать, что рассматриваемый граф является связным. Очевидно также, что количество приходов почтальона в некоторую вершину должно быть равно количеству уходов его из этой вершины. При этом если почтальон не проходит более одного раза по ребру, инцидентному некоторой вершине, то эта вершина должна быть инцидентна четному числу ребер. Общее количество ребер, инцидентных вершине x, назовем степенью вершины x и будем

обозначать через $d(x)$. Если графе G все вершины имеют четную степень, то граф называется четным. В ориентированном графе число дуг, входящих в вершину x, называется *внутренней степенью* вершины x[1]); эту степень мы будем обозначать через $d^-(x)$. Число дуг, выходящих из вершины x, называется *внешней степенью* $d^+(x)$ вершины x[2]). На рис. 6.8 $d^-(a) = 2$, $d^+(a) = 1$. Если в графе для всех вершин $d^+(x) = d^-(x)$, то такой граф называется *симметричным*[3]). Предположим, что мы знаем оптимальный маршрут почтальона на графе G, который начинается и заканчивается в вершине s. Каким будет оптимальный маршрут почтальона, если в качестве начальной выбрать некоторую другую вершину, скажем вершину t? Ответ на этот вопрос находится из следующих соображений. Любой оптимальный маршрут R, начинающийся в вершине s, в конечном счете когда-то впервые проходит через вершину t. Назовем эту часть маршрута R_1, а оставшуюся часть обозначим через R_2. Заметим, что R_1 начинается в вершине s и оканчивается в вершине t. В свою очередь R_2 начинается в вершине t и заканчивается в вершине s. Сформируем новый маршрут R', состоящий из R_2 и следующего за ним R_1. Маршрут R' начинается в вершине t, оканчивается также в вершине t и имеет такую же суммарную длину, как и маршрут R. Следовательно, R' должен быть оптимальным маршрутом, начинающимся из вершины t. Таким образом, мы можем сделать следующее утверждение.

ТЕОРЕМА 6.1. Суммарная длина оптимального маршрута почтальона не зависит от того, какая из вершин этого маршрута будет выбрана в качестве начальной.

Ниже, в разд. 2 этой главы описывается алгоритм поиска оптимального маршрута почтальона для неориентированного графа (ребра соответствуют улицам, по которым можно перемещаться в обоих направлениях). В разд. 3 приводится алгоритм поиска оптимального маршрута почтальона для ориентированного графа (ребра соответствуют улицам, по которым можно перемещаться только в определенном направлении). В разд. 4 рассматривается задача почтальона на графе, в котором некоторые дуги ориентированы, а остальные — неориентированы (т. е. имеются улицы с односторонним и двусторонним движением).

---

[1]) Более распространенный термин — «полустепень захода». — *Прим. ред.*

[2]) Более распространенный термин — «полустепень исхода». — *Прим. ред.*

[3]) Автор имеет в виду граф, в котором не обязательно каждой дуге соответствует дуга противоположного направления, в последнем случае граф был бы симметричным. — *Прим. ред.*

6.2. Задача почтальона для неориентированного графа

В этом разделе описывается алгоритм решения задачи почтальона для любого неориентированного графа $G = (X, E)$, в котором ребра могут обходиться в каждом из двух направлений.

Необходимо рассмотреть отдельно следующие два случая:

Случай А. Граф G четный.

Случай Б. Граф G нечетный.

Случай А. Если граф четный, то оптимальное решение задачи почтальона является эйлеровым маршрутом. В этом случае почтальон не должен обходить более одного раза любое ребро графа.

Как же найти на графе G эйлеров маршрут, в котором вершина s является начальной вершиной? Для этого необходимо пройти любое ребро (s, x), инцидентное вершине s, а затем любое не использованное еще ребро, инцидентное вершине x. Этот процесс прохождения неиспользованных ребер повторяется до тех пор, пока не происходит возврата в вершину s. (Мы должны обязательно прийти в вершину s, поскольку каждая вершина имеет четную степень и каждое посещение (приход и уход) некоторой вершины оставляет четное число неиспользованных ребер, инцидентных этой вершине. Следовательно, каждый раз, когда почтальон приходит в некоторую вершину, имеется неиспользованное ребро для того, чтобы покинуть эту вершину.) Ребра, по которым почтальоном совершен обход, образуют цикл C_1. Если в цикл C_1 вошли все ребра графа G, то C_1 является эйлеровым маршрутом (а значит, и оптимальным решением задачи).

В противном случае следует образовать цикл C_2, состоящий из неиспользованных ребер и начинающийся с произвольного неиспользованного ребра. Образование циклов C_3, C_4, ..., состоящих из неиспользованных ребер, продолжается до тех пор, пока не будут использованы все ребра графа. Далее следует соединить все циклы C_1, C_2, ... в один цикл C, который содержит все ребра графа G. В цикл C каждое ребро графа входит только по одному разу, и поэтому он является оптимальным решением задачи почтальона.

Два цикла C_1 и C_2 могут быть соединены в один только тогда, когда они содержат общую вершину x. Для соединения двух таких циклов необходимо выбрать в качестве исходного произвольное ребро цикла C_1 и двигаться вдоль его ребер до достижения вершины x. Затем надо пройти все ребра цикла C_2 и вернуться в вершину x. Наконец, следует продолжить обход ребер цикла C_1 до возвращения назад к начальному ребру. Пройденный маршрут является циклом, полученным в результате соединения циклов C_1 и C_2. Эта процедура легко может быть расширена на случай соединения любого количества циклов и может выполняться

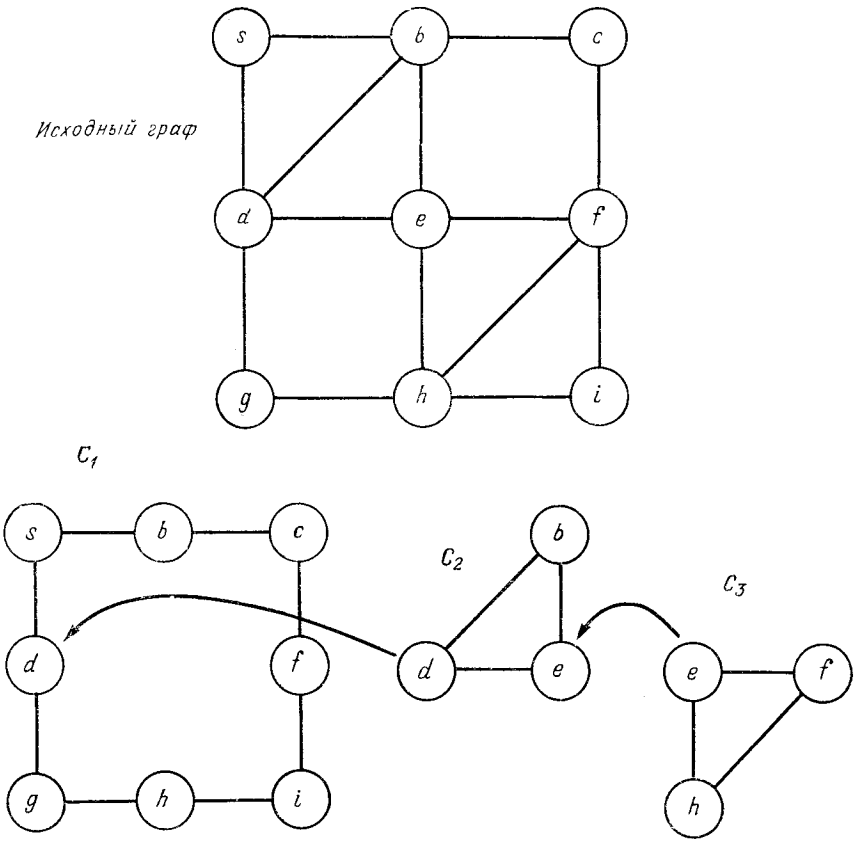


Рис. 6.3. Соединение нескольких циклов в один.

до тех пор, пока не образуются два их подмножества, не имеющие общих вершин.

ПРИМЕР 1. Необходимо найти оптимальный маршрут почтальона на четном графе, представленном на рис. 6.3. Начиная с вершины s, совершим обход ребер графа по периметру до возвращения в вершину s. В результате обхода этих ребер будет сформирован цикл $C_1 = (s, b), (b, c), (c, f), (f, i), (i, h), (h, g), (g, d), (d, s)$. Ребра цикла C_1 будем считать использованными. Далее, начиная с неиспользованного ребра (d, b), обойдем неиспользованные ребра верхнего треугольного цикла $C_2 = (d, b), (b, e), (e, d)$, после чего ребра цикла C_2 будем также считать использованными. Наконец, начиная с неиспользованного ребра (e, f), обойдем ребра нижнего треугольного цикла $C_3 = (e, f), (f, h), (h, e)$. Теперь все ребра графа использованы.

Соединим найденные выше циклы следующим образом: цикл C_2 вставим между ребрами (g, d) и (d, s) цикла C_1; а цикл C_3 — между ребрами (b, e), и (e, d) цикла C_2. В результате получим цикл $C = (s, b), (b, c), (c, f), (f, i)$

(i, h), (h, g), (g, d), (d, b), (b, e), (e, f), (f, h), (h, e), (e, d), (d, s). Очевидно, что C — оптимальное решение задачи почтальона для рассмотренного графа, так как C содержит каждое ребро ровно один раз, а начинается и заканчивается в вершине s. Отметим, что в этом примере не учитывались длины ребер.

Случай Б. Граф G не является четным графом. В любом маршруте почтальона число входов в вершину равно числу выходов из нее. Следовательно, если вершина x имеет нечетную степень, то по крайней мере одно ребро, инцидентное вершине x, должно обходиться почтальоном повторно.

Пусть $f(i, j)$ — число дополнительных прохождений почтальоном ребра (i, j). Значит, ребро (i, j) обходится почтальоном $f(i, j) + 1$ раз. Очевидно, что $f(i, j) + 1$ должно быть неотрицательным целым числом. Заметим, что $f(i, j)$ не содержит информации о направлении движения вдоль ребра (i, j).

Построим новый граф $G^* = (X, E^*)$, в котором ребро (i, j) графа G повторено $f(i, j) + 1$ раз. Очевидно, что эйлеров маршрут в графе G^* соответствует маршруту почтальона в графе G. Почтальон желает выбрать значения переменных $f(i, j)$ такими, чтобы (а) граф G^* был четным графом, (б) $\Sigma\, a(i, j)\, f(i, j)$ — общая длина повторно обходимых ребер — была минимальной.

Если вершина x в графе G имеет нечетную степень, то для того, чтобы в графе G^* вершина x имела четную степень, почтальон должен повторно обойти нечетное число ребер, инцидентных этой вершине. Аналогично, если вершина x в графе G имеет четную степень, то для того, чтобы в графе G^* вершина x имела четную степень, почтальон должен повторно обойти четное число ребер, инцидентных этой вершине (нуль является четным числом). Вспомним упражнение 2 гл. 1, в котором утверждается, что граф G имеет четное число вершин с нечетной степенью. Если мы проследим до конца начинающуюся в вершине с нечетной степенью цепь повторно обходимых ребер, то она обязательно должна закончиться в другой вершине с нечетной степенью. Таким образом, повторно обходимые ребра образуют цепи, началом и концом которых являются вершины с нечетной степенью. Конечно, любая такая цепь может содержать в качестве промежуточных вершины с четной степенью. Следовательно, почтальон должен (а) решить, какие вершины с нечетной степенью будут соединены цепью повторно обходимых ребер и (б) знать точный состав каждой такой цепи.

Почтальон может с помощью каждого из алгоритмов построения кратчайшего пути, предложенных Флойдом и Данцигом и рассмотренных в разд. 3.2, определить на графе G кратчайший путь между каждой парой вершин с нечетной степенью.

Для определения пары вершин с нечетной степенью, которые должны быть соединены цепью повторно обходимых ребер, поч-

тальон может поступить следующим образом. Построить граф $G' = (X', E')$, множество вершин которого состоит из всех вершин с нечетной степенью графа G, а множество ребер соединяет каждую пару вершин. Присвоить каждому ребру графа вес, равный некоторому очень большому числу за вычетом длины кратчайшего пути между соответствующими вершинами графа G, вычисленной с помощью алгоритмов Флойда или Данцига.

Далее на графе G' следует построить паросочетание с максимальным весом, используя для этого алгоритм, рассмотренный в разд. 5.4. Так как граф G' имеет четное число вершин и каждая пара вершин в G' соединена ребром, то паросочетание с максимальным весом будет покрывать каждую вершину в точности одним ребром. Это паросочетание связывает на графе G пары вершин с нечетными степенями. Почтальон повторно должен обходить ребра, составляющие цепь кратчайшей длины и соединяющие пару связанных в паросочетании вершин. Поскольку это паросочетание имеет наибольший общий вес, то получаемый в результате маршрут почтальона должен иметь минимальную общую длину.

Таким образом, задачу почтальона для неориентированного графа можно решить с помощью алгоритмов Флойда или Данцига и алгоритма построения паросочетания с максимальным весом. Никакого нового алгоритма для этого не требуется.

ПРИМЕР 2. Найдем оптимальный маршрут почтальона на неориентированном графе, который представлен на рис. 6.4. Вершины графа *a*, *c*, *d* и *f* имеют нечетную степень. Длины кратчайших цепей между всеми парами вершин с нечетными степенями задаются следующей таблицей:

Таблица 6.1

Матрица путей наименьшей длины

Паросочетание	Вес
(a, c), (d, f)	4+3=7
(a, d), (c, f)	2+4=6
(a, f), (c, d)	3+2=5

	a	b	c	d	e	f
a	0	1	4	2	4	3
b	1	0	5	3	5	2
c	4	5	0	2	7	4
d	2	3	2	0	6	3
e	4	5	7	6	0	3
f	3	2	4	3	3	0

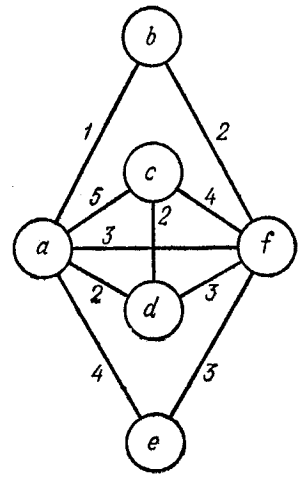


Рис. 6.4.

Читателю предлагается проверить эти величины с помощью алгоритмов Флойда или Данцига.

Сформируем граф G', показанный на рис. 6.5. Вершинами графа G' являются вершины a, c, d и f графа G, имеющие нечетную степень. G' является полным графом, так как в нем представлены все возможные ребра. К счастью, поскольку в G' вершин немного, нам легче получить паросочетание с минимальным весом на графе G' с помощью перебора, чем используя алгоритм нахождения паросочетания с максимальным весом. В приведенном на рис. 6.5 графе возможны три паросочетания, указанные в табл. 6.1. Следовательно, паросочетанием с минимальным весом является (a, f), (c, d).

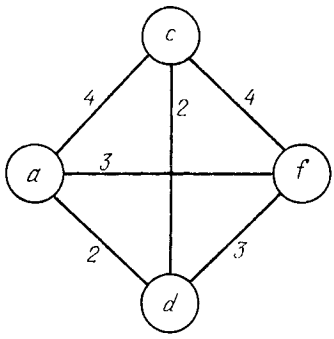


Рис. 6.5. Граф G'.

Таким образом, почтальону следует выбирать для повторного обхода кратчайший путь от a к f, представляющий собой ребро (a, f) в графе G, и кратчайший путь от c к d, который является ребром (c, d) в том же графе G. На рис. 6.6 показан граф G^*, в котором ребра (a, f) и (c, d) один раз продублированы. Все вершины в графе G^* имеют четную степень, и оптимальный маршрут почтальона на графе G (рис. 6.4) соответствует эйлерову маршруту в графе G^* (рис. 6.6). Метод решения задачи почтальона, описанный выше для случая A, может быть применен к графу G^*. В рассматриваемом примере оптимальным маршрутом почтальона является путь (a, b), (b, f), (f, e), (e, a), (a, c), (c, f), (f, d), (d, c), (c, d), (d, a), (a, f), (f, a), в котором каждое ребро графа G^* обходится ровно один раз, а каждое ребро в графе G обходится по крайней мере один раз. В графе G повторно обходятся только ребра (a, f) и (c, d). Общая длина этого маршрута равна 34 единицам, и она на 5 единиц больше, чем сумма длин ребер графа G.

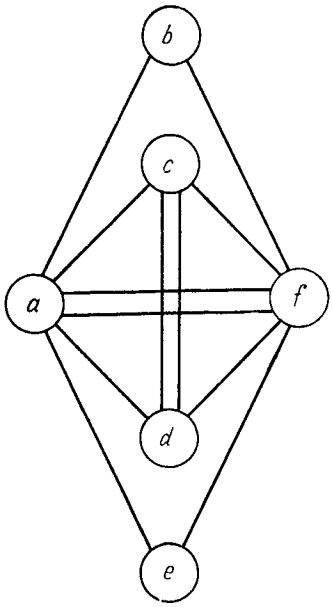


Рис. 6.6. Граф G^*.

Заметим, что если бы использовался алгоритм построения паросочетания с максимальным весом, то вес каждого ребра в графе G' следовало бы принять равным некоторому большому числу, скажем M, минус длина кратчайшего пути между соответствующими двумя конечными точками в графе G. Таким образом, паросочетание (a, c), (d, f) имело бы общий вес, равный $(M - 4) + (M - 3) = 2M - 7$. Паросочетание (a, d), (c, f) имело бы общий вес, равный $(M - 2) + (M - 4) = 2M - 6$. Паросочетание (a, f), (c, d) имело бы общий вес, равный $(M - 3) + (M - 2) = 2M - 5$ и было бы выбрано в качестве паросочетания с максимальным весом. Использование большого числа M просто может рассматриваться как способ преобразования задачи построения паросочетания с минимальным весом, покрывающего все вершины графа, в задачу построения паросочетания с максимальным весом.

6.3. Задача почтальона для ориентированного графа

Ориентированный граф $G = (X, A)$ для задачи почтальона соответствует физическим ситуациям, в которых все улицы допускают лишь одностороннее движение. Ориентация дуги графа определяет направление, в котором можно проходить по соответствующей улице.

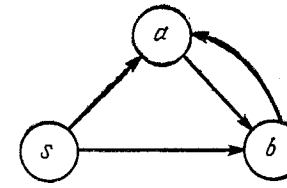


Рис. 6.7. Маршрута почтальона не существует.

В отличие от неориентированных графов задача почтальона на ориентированном графе может не иметь решения. Для примера рассмотрим граф, изображенный на рис. 6.7. Как только почтальон достигает вершины a или вершины b, он не может вернуться в вершину s, так как из вершин множества $\{a, b\}$ не выходит ни одной дуги к вершинам, не принадлежащим этому множеству. Вообще решения задачи почтальона не существует в тех случаях, когда имеется такое множество вершин s, что отсутствуют дуги, выходящие из вершин множества s к вершинам, не принадлежащим s. Если же таких множеств s нет, то у почтальона всегда есть возможность завершить свой маршрут, каким бы длинным он ни оказался, поскольку при движении по дугам графа его нигде не ждет «ловушка» вроде множества $\{a, b\}$, изображенного на рис. 6.7.

В ориентированном графе, как и в неориентированном, число приходов почтальона в некоторую вершину должно быть равно числу уходов его из этой вершины. Следовательно, если число дуг, входящих в вершину x, превышает число выходящих [т. е. $d^-(x) > (d^+(x)]$, то почтальон должен обходить некоторые выходящие из x дуги более одного раза. Аналогично почтальон должен более одного раза обходить некоторые входящие в вершину x дуги, если число выходящих из нее дуг превышает число входящих [т. е. $d^+(x) > d^-(x)$]. Таким образом, если для некоторой вершины x имеет место условие $d^+(x) \neq d^-(x)$, то эйлерова маршрута не существует.

Для решения задачи почтальона на ориентированном графе необходимо отдельно рассмотреть следующие два случая:

Случай А. Граф G симметричный [т. е. $d^+(x) = d^-(x)$ для всех x].

Случай *Б*. Граф G несимметричный.

Случай А. Если граф G — симметричный, то почтальон может пройти свой маршрут без повторного обхода какой-либо из дуг, т. е. оптимальным решением задачи почтальона является эйлеров маршрут.

На графе $G = (X, A)$ эйлеров маршрут может быть найден с помощью метода, аналогичного методу определения такого же цикла для неориентированного четного графа. Для этого необходимо, начиная из исходной вершины s, обходить дуги в направлении, совпадающем с их ориентацией (без повторных обходов) до тех пор, пока не произойдет возврат в вершину s. Пройденный при этом путь образует некоторый контур C_1. Затем, начав с любой неиспользованной дуги и обходя только такие дуги в направлении заданной их ориентации, строится другой контур C_2. Описанная процедура построения контуров повторяется до тех пор, пока не будет завершен обход всех дуг графа. Наконец, все найденные контуры необходимо соединить в один большой контур C, как это сделано на рис. 6.3. Контур C включает каждую дугу графа ровно один раз и представляет в случае A оптимальное решение задачи почтальона.

Случай Б. Как и ранее, пусть $f(i, j)$ обозначает число повторных обходов дуги (i, j). Почтальону необходимо выбрать такие неотрицательные значения переменных $f(i, j)$, которые минимизируют общую длину повторно обходимых дуг

$$\sum a(i, j) f(i, j) \tag{1}$$

при условии, что в каждую вершину он входит столько раз, сколько выходит из нее, т. е.

$$d^-(i) + \sum_j f(j, i) = d^+(i) + \sum_j f(i, j). \tag{2}$$

Преобразовав уравнение (2), получаем

$$\sum_j [f(i, j) - f(j, i)] = d^-(i) - d^+(i) = \underline{D}(i). \tag{3}$$

Таким образом, почтальон желает минимизировать выражение (1) при условии, что для всех вершин графа G удовлетворяется равенство (3). Эта задача представляет одну из модификаций задачи о потоке минимальной стоимости. Вершины, для которых $\underline{D}(i) < 0$, являются стоками с предельным значением суммарного входящего потока, равным $-\underline{D}(i)$. Вершины, где $\underline{D}(i) > 0$, представляют собой источники с предельным значением суммарного выходящего потока, равным $\underline{D}(i)$. Вершины, для которых $\underline{D}(i) = 0$, являются промежуточными вершинами. Значения пропускных способностей всех дуг не ограничены.

Сформулированная задача о потоке минимальной стоимости может быть решена путем введения в граф дополнительного источника и стока. Дополнительный источник связан дугами со всеми другими источниками, и все стоки связаны дугами с дополнительным стоком. Значение пропускной способности каж-

дой дуги, выходящей из дополнительного источника, равно предельному значению суммарного потока источника, который инцидентен этой дуге. Значение пропускной способности каждой дуги, входящей в дополнительный сток, равно предельному значению суммарного входящего потока в сток, который инцидентен этой дуге. Теперь для получения решения задачи о потоке минимальной стоимости можно использовать алгоритм, рассмотренный в разд. 4.3. Так как правые части всех равенств, (3) — целые числа, то можно утверждать, что с помощью описанного алгоритма определения потока минимальной стоимости будут получены неотрицательные целые значения $f(i, j)$.

Определив целочисленные оптимальные значения переменных $f(i, j)$, построим граф G^*, в котором дуга (i, j), соединяющая вершины i и j для всех $(i, j) \in A$, повторяется $f(i, j) + 1$ раз. В соответствии с равенством (3) такой граф G^* является симметричным. Теперь для графа G^* с помощью метода, описанного для случая A, может быть найден эйлеров маршрут. Последний на графе G^* соответствует маршруту почтальона на графе G, в котором дуга (i, j) обходится $[f(i, j) + 1]$ раз. Так как оптимальные значения $f(i, j)$ минимизируют выражение (1), получаемый на графе G^* эйлеров маршрут должен соответствовать оптимальному маршруту почтальона на графе G.

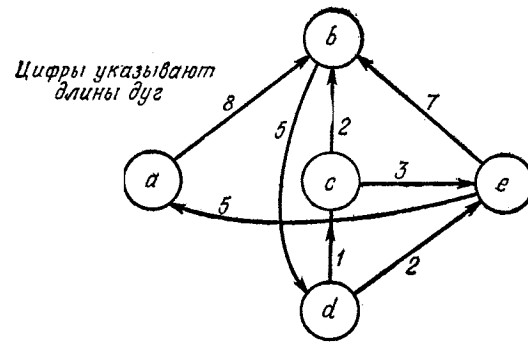


Рис. 6.8. Задача почтальона на ориентированном графе.

ПРИМЕР 1. Найдем оптимальный маршрут почтальона для ориентированного графа, представленного на рис. 6.8.

Определяя внешние и внутренние степени вершин, получим:

$d^-(a) = 1 = d^+(a)$; вершина a — промежуточная;

$d^-(b) = 3 > d^+(b) = 1$; вершина b — источник с предельным значением общего выходящего потока, равным $3 - 1 = 2$;

$d^-(c) = 1 < d^+(c) = 2$; вершина c — сток с предельным значением суммарного входящего потока, равным $2 - 1 = 1$;

$d^-(d) = 1 < d^+(d) = 2$; вершина d — сток с предельным значением суммарного входящего потока, равным $2 - 1 = 1$;

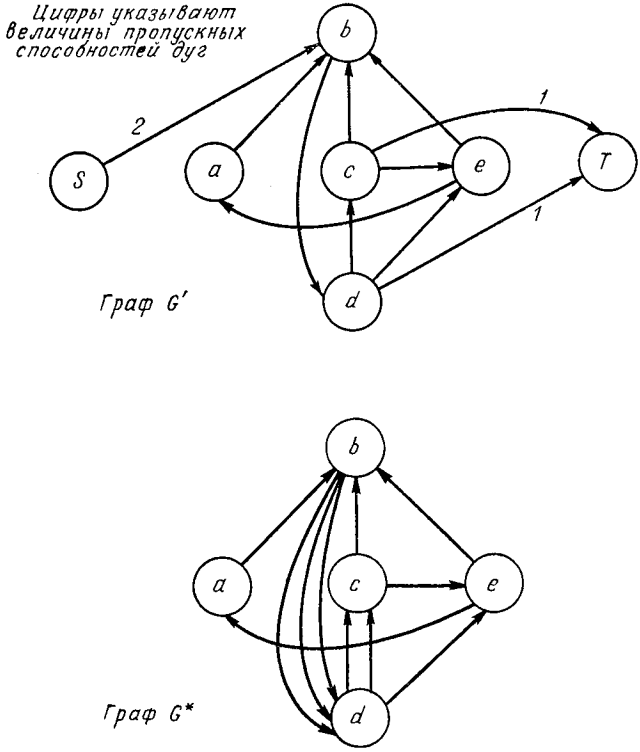


Рис. 6.9.

$d^-(e) = 2 = d^+(e)$, вершина e — промежуточная.

Введем дополнительный источник S и соединим его дугой (S, b) с источником b. Дуга (S, b) имеет значение пропускной способности, равное 2 единицам. Введем также дополнительный сток T и соединим вершины c и d с T дугами (c, T) и (d, T). Дуга (c, T) имеет значение пропускной способности, равное величине предельного значения суммарного входящего в c потока, т. е. одной единице. Дуга (c, T) имеет значение пропускной способности, равное величине предельного значения суммарного входящего в c потока, которая также составляет одну единицу. Значения пропускных способностей всех других дуг не ограничены. Полученный в результате таких преобразований граф показан на рис. 6.9. Таким образом, мы видим, что от S к T может протекать не более 2 единиц потока и весь поток из S' должен проходить через вершину b. В вершину T может прийти не более одной единицы потока через вершину c и не более одной единицы потока через вершину d. Дуги, составляющие путь потока из S в T, являются именно теми дугами, которые почтальон должен обойти более одного раза. В результате применения алгоритма определения потока минимальной стоимости или алгоритма ограниченного перебора допустимых потоков можно установить, что одна единица оптимального потока протекает по дугам (S, b), (b, d), (d, T) с затратами, равными $0 + 5 + 0 = 5$; вторая единица оптимального потока протекает по дугам (S, b), (b, d), (d, c), (c, T) с затра-

тами, равными $0 + 5 + 1 + \_ = 6$. Следовательно, $f(b, d) = 2$, $f(d, c) = 1$; для всех остальных (i, j) имеет место $f(i, j) = 0$. Таким образом, почтальон должен дважды повторить обход дуги (b, d) и один раз — обход дуги (d, c), добавляя к протяженности своего маршрута $5 + 5 + 1 = 11$ единиц. На рис. 6.9 показан граф G^*, который включает все дуги графа G совместно с двумя дополнительными дугами (b, d) и одной дополнительной дугой (d, c). Заметим, что граф G^* является симметричным и, как мы знаем из случая A, на нем существует эйлеров маршрут. Например, эйлеров маршрут на графе G^*, являющийся оптимальным маршрутом почтальона на графе G, включает дуги (a, b), (b, d), (d, c), (c, b), (b, d), (d, c), (c, e), (e, b), (b, d), (d, e), (e, a). Общая длина этого маршрута равна 44 единицам: 33 единицы приходятся на сумму длин всех дуг в графе G и 11 единиц — на сумму длин всех повторно обходимых дуг в этом графе.

6.4. Задача почтальона для смешанного графа

Ниже рассматривается задача почтальона на графе, часть дуг которого ориентирована, а часть — нет, что характерно для смешанного графа. Почтальон должен обходить ориентированные дуги графа только в направлении их ориентации (улицы с односторонним движением), а неориентированные дуги (улицы с двусторонним движением) в любом направлении (а если нужно, то и в обоих). В дальнейшем при вычислении внешних и внутренних степеней вершин графа неориентированные дуги не рассматриваются.

Возникает вопрос: всегда ли существует маршрут почтальона на смешанном графе? Оказывается, не всегда. Дело в том, что на графе может существовать такое множество вершин S, что все дуги, соединяющие вершины из этого множества с вершинами, не принадлежащими множеству S, будут направлены в сторону вершин множества S. В этом случае, как только почтальон достигнет вершины, принадлежащей множеству S, он никогда не сможет достичь ни одной из вершин, не принадлежащих этому множеству. Он попадает в «ловушку», и, следовательно, задача почтальона не имеет решения. Если такого множества S не существует, то почтальон может продолжать обход дуг (может быть, довольно долго) до тех пор, пока не обойдет все дуги и не вернется в вершину, из которой он начал маршрут.

Для смешанного графа $G = (X, A)$ следует рассмотреть отдельно три случая:

Случай А. Граф G четный и симметричный.

Случай Б. Граф G четный, но несимметричный.

Случай В. Граф G нечетный и несимметричный.

Случай А наиболее прост для анализа, так как способ решения задачи в этом случае основывается на сочетании методов ее решения для симметричного ориентированного и четного неориентированного графов, рассмотренных соответственно в разд. 6.3 и 6.2.

Для решения задачи следует выбрать любую ориентированную дугу в графе G и построить некоторый контур из ориентированных дуг в соответствии с процедурой четных ориентированных графов, описанной в разд. 6.3 (это возможно, поскольку граф G — симметричный). Указанная процедура повторяется

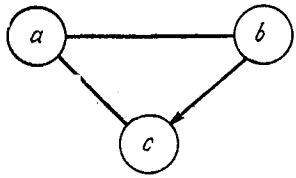


Рис. 6.10. Смешанный граф, в котором существует эйлеров маршрут.

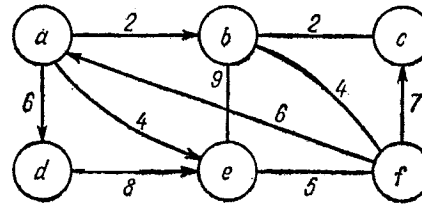


Рис. 6.11. Смешанный граф, в котором отсутствует эйлеров маршрут.

до тех пор, пока не будут использованы все ориентированные дуги (это также возможно благодаря тому, что неиспользованные дуги всегда представляют четный симметричный граф). Далее та же процедура повторяется для неориентированных дуг в графе G (это опять-таки возможно, поскольку неиспользованные дуги образуют четный граф). После того как все дуги графа G будут использованы, производится соединение всех полученных выше контуров и циклов в один маршрут C. Этот маршрут представляет эйлеров маршрут графа G и для случая A является оптимальным решением задачи почтальона.

Случай Б. Граф G четный, но несимметричный. В этом случае нелегко заранее ответить на вопрос, должен ли почтальон повторно обходить какие-либо дуги графа. Например, на рис. 6.10 оптимальным решением для четного несимметричного графа является эйлеров маршрут, включающий неориентированную дугу (a, b), которая обходится в направлении от a к b, и неориентированную дугу (c, a), которая обходится в направлении от c к a. С другой стороны, оптимальное решение для четного несимметричного графа, представленного на рис. 6.11, не может быть эйлеровым маршрутом, так как дуга (f, a) должна обходиться трижды, для того чтобы почтальон мог обойти дуги (a, b), (a, d) и (a, e).

В алгоритме решения задачи почтальона на смешанном графе направление обхода каждой неориентированной дуги в графе G выбирается произвольно. В результате граф G преобразуется в четный ориентированный граф G_d. Для четного ориентированного графа G_d может быть применен метод решения задачи

почтальона, рассмотренный в разд. 6.3. Однако из-за произвольности выбора направления дуги возникает необходимость корректировки направления некоторых из них.

Алгоритм решения задачи почтальона на смешанном графе

Пусть $G = (X, A)$ — любой четный смешанный граф. Оптимальный маршрут почтальона (если он существует) может быть найден следующим образом.

Обозначим через U множество всех неориентированных дуг, а через V — множество всех ориентированных дуг в графе G. Для каждой дуги в U выберем исходные направления. Назовем полученный ориентированный граф графом G_d. Для каждой вершины i графа G_d вычислим

$$\underline{D}(i) = d^-(i) - d^+(i). \qquad (4)$$

Если $\underline{D}(i) < 0$, то вершина i является стоком с предельной величиной суммарного входящего потока, равной $\underline{D}(i)$. Если $\underline{D}(i) > 0$, то вершина i является источником с предельной величиной суммарного выходящего потока $\underline{D}(i)$. Если $\underline{D}(i) = 0$, то вершина i является промежуточной. Если в графе G_d все вершины являются промежуточными, то граф G_d — четный симметричный ориентированный граф. Для получения оптимального маршрута почтальона на таком графе может быть применен метод решения задачи, рассмотренный в разд. 6.3. С помощью этого метода находится эйлеров маршрут на графе G_d, который соответствует оптимальному маршруту почтальона на графе G. В противном случае необходимо построить граф $G' = (X, A')$, в котором:

(а) Каждая дуга $(i, j) \in V$ замещается дугой $(i, j) \in A'$ с неограниченной величиной пропускной способности и стоимостью, равной длине дуги (i, j).

(б) Каждой дуге $(i, j) \in U$ соответствует пара дуг (i, j) и (j, i) в A'. Каждая из этих дуг имеет неограниченную величину пропускной способности и стоимость, равную длине дуги $(i, j) \in A$.

(в) Каждой дуге $(i, j) \in U$ соответствует также ориентированная дуга $(j, i)_1 \in A'$, направление которой противоположно направлению дуги, принятому в G_d. Эта дуга называется фиктивной дугой. Припишем фиктивной дуге стоимость, равную нулю, и величину пропускной способности, равную 2.

С учетом определенных выше для графа G_d предельных значений потоков, выходящих из источников и входящих в стоки, воспользуемся алгоритмом построения потока минимальной стоимости для поиска на графе G' потока минимальной стоимости, удовлетворяющего потребности всех стоков. Если такого

потока не существует, то не существует и маршрута почтальона. В противном случае пусть через $f(i, j)$ обозначается величина потока, который протекает по дуге (i, j) графа G', полученного в результате решения задачи о потоке минимальной стоимости. Напомним, что поток, полученный с помощью описанного выше алгоритма построения потока минимальной стоимости, является неотрицательным и целочисленным. В ходе доказательства того, что алгоритм обеспечивает получение решения задачи почтальона, будет показано, что по каждой фиктивной дуге протекает поток, равный 0 или 2.

Построим далее граф G^*, в котором:

(а) Каждая нефиктивная дуга $(i, j) \in V$ графа G' заменяется $[f(i, j) + 1]$ дугами (i, j) графа G^*.

(б) Каждая нефиктивная дуга $(i, j) \in U$ графа G' заменяется $f(i, j)$ дугами (i, j) графа G^*.

(в) Если поток в фиктивной дуге равен двум единицам, то в граф G^* включается одна такая дуга.

(г) Если поток в фиктивной дуге равен нулю, то в граф G^* включается одна такая дуга с противоположной ориентацией. Таким образом, если в фиктивной дуге поток равен нулю, то выбранное исходное направление дуги графа G_d сохраняется и в графе G^*, если же поток в фиктивной дуге равен двум единицам, то направление ориентации соответствующей дуги графа G_d в графе G^* изменяется на противоположное.

Граф G^* является четным симметричным ориентированным графом. Описанный в разд. 6.3 метод решения для четных симметричных ориентированных графов теперь может быть применен для поиска эйлерова маршрута на графе G^*. Этот эйлеров маршрут на графе G^* соответствует оптимальному маршруту почтальона на исходном графе G.

Докажем, что построенный маршрут почтальона является оптимальным.

Так как граф G не обязательно является симметричным, некоторые вершины могут иметь избыточное количество входящих дуг, другие вершины, наоборот, могут иметь излишнее число выходящих дуг. В идеальном варианте нам хотелось бы назначить направления всех неориентированных дуг так, чтобы полученный ориентированный граф был симметричным. Тогда можно было бы применить описанный в разд. 6.3 метод решения задачи почтальона для поиска эйлерова маршрута в четном симметричном ориентированном графе G_d. Однако может возникнуть ситуация, когда преобразование неориентированного графа в ориентированный симметричный граф невозможно. В этом случае некоторые из дуг (ориентированные или неориентированные) должны обходиться почтальоном более одного раза. Конечно, для

почтальона необходимо так выбрать повторно обходимые дуги, чтобы их общая длина была наименьшей.

В алгоритме выбирается некоторое исходное направление для каждой неориентированной дуги графа G. В итоге получается ориентированный граф G_d. Метод решения задачи почтальона для ориентированного графа, рассмотренный в разд. 6.3, мог бы быть применен для получения решения задачи на графе G_d. Но получаемое с помощью этого метода решение зависит от направлений, приписанных неориентированным дугам графа G_d, и поэтому всегда существует опасность построения неоптимального решения задачи почтальона.

Далее алгоритм порождает граф G' и, используя такие же, как в графе G_d, предельные значения суммарных потоков, выходящих из источников и входящих в стоки, находит поток минимальной стоимости, удовлетворяющий требования каждого из стоков.

Граф G' включает:

(а) Дуги, соответствующие ориентированным дугам графа G. (Эти дуги имеют ненулевую стоимость и неограниченную величину пропускной способности.)

(б) Нефиктивные дуги, соответствующие неориентированным дугам графа G. (Эти дуги также имеют ненулевую стоимость и неограниченную величину пропускной способности.)

(в) Фиктивные дуги, соответствующие неориентированным дугам графа G. (Эти дуги имеют нулевую стоимость и пропускную способность, равную 2.)

Значения оптимального потока $f(i, j)$ для дуг типа (а) и (б) графа G' равны числу повторных обходов почтальоном соответствующих ориентированных дуг графа G. Таким образом, почтальон каждую дугу $(i, j) \in V$ будет обходить $[f(i, j) + 1]$ раз, а дуги (i, j), принадлежащие множеству U, $-[f(i, j) + f(j, i) + 1)]$ раз.

Как будет показано далее, каждой фиктивной дуге типа (в) соответствует оптимальный поток, равный нулю или двум единицам. Если в фиктивной дуге $(j, i)_1$ оптимальный поток равен двум единицам, то выходящий из вершины j общий поток уменьшается на две единицы, а выходящий из вершины i общий поток возрастает на две единицы. Этот же эффект мог бы быть достигнут при выборе в графе G_d обратной исходной ориентации дуги (i, j). Следовательно, алгоритм изменяет исходную ориентацию этой дуги на противоположную.

Если же в фиктивной дуге $(j, i)_1$ оптимальный поток равен нулю, то поток в этой дуге не влияет на величины выходящих суммарных потоков из вершин i и j, что эквивалентно сохранению выбранной исходной ориентации этой дуги графа G_d. Следовательно, алгоритм сохраняет первоначальную ориентацию

этой дуги. После определения в дугах графа G' значений потока минимальной стоимости $f(i, j)$ формируется граф G^*.

Остается показать, что

(а) граф G^* является четным и симметричным.

(б) эйлеров маршрут графа G^* соответствует оптимальному маршруту почтальонов на графе G.

(в) на графе G не существует маршрута почтальона, если с помощью алгоритма построения потока минимальной стоимости на графе G' не удается найти поток, удовлетворяющий требования всех стоков.

Доказательство (*а*). Так как граф G_d четный, то $d^+(i) + d^-(i)$ является четным числом для всех вершин i в G_d. В свою очередь $d^+(i)$ и $d^-(i)$ либо оба нечетны, либо оба четны, и поэтому в любом случае $D(i)$ должно быть четным. Значит, в графе G' величины суммарных потоков, выходящих из каждого источника и входящих в каждый из стоков, являются четным числом (нуль считается четным числом). Значения пропускных способностей всех дуг также либо четные, либо неограниченные числа. Следовательно, с помощью алгоритма определения потока минимальной стоимости будет получен оптимальный поток, значения которого в каждой из дуг графа G' четные числа. Таким образом, значение потока в каждой дуге кратно двум. Значение $d(i)$ степени вершины i в графе G^* равно значению $d(i)$ в графе G плюс величина потока, втекающего в вершину i и вытекающего из нее по нефиктивным дугам. Так как поток в каждой дуге является четным числом, $d(i)$ в графе G^* также является четным числом, а граф G^* — четным графом. Каждые две единицы потока, входящего в вершину i по нефиктивной дуге, увеличивают значение $d^-(i)$ в графе G^* на две единицы. Каждые две единицы потока, выходящие из вершины i по нефиктивной дуге, также увеличивают значение $d^+(i)$ в графе G^* на две единицы.

Две единицы потока, входящего в вершину i по фиктивной дуге $(j, i)_1$, вызывают изменение исходной ориентации дуги (i, j) графа G_d на обратную, что эквивалентно увеличению значения $d^-(i)$ в графе G^* на две единицы. Две единицы потока, выходящего из вершины i графа G^* по фиктивной дуге $(i, j)_1$, вызывают изменение исходной ориентации дуги (j, i) графа G_d на обратную, что эквивалентно увеличению значения $d^+(i)$ на две единицы в графе G^*. Таким образом, каждая единица потока, втекающая в вершину i, увеличивает значение $d^-(i)$ на единицу и каждая единица потока, вытекающая из вершины i, увеличивает на единицу значение $d^+(i)$. Так как поток минимальной стоимости удовлетворяет равенству (3) для всех вершин графа G, из этого следует, что граф G^* является симметричным.

Доказательство (*б*). Предположим, что с помощью алгоритма, описанного для смешанного графа, найден неоптимальный марш-

рут почтальона. В таком случае должен существовать другой маршрут почтальона, в котором повторно обходимые дуги имеют меньшую общую длину. Этому маршруту на графе G' должен соответствовать поток, имеющий меньшую стоимость, чем тот, который получается с помощью алгоритма построения потока минимальной стоимости, что невозможно.

Доказательство (*в*). Так как равенство $\Sigma\, d^-(i) = \Sigma\, d^+(i)$ справедливо для любого графа G_d, суммарный выходящий из всех источников поток должен быть равен суммарному потоку, входящему во все стоки графа G_d. Таким образом, для удовлетворения потребностей всех стоков из всех источников должен вытекать суммарный поток, имеющий предельное значение. Если в графе G существует дуга из i в j, то граф G' должен иметь дугу (i, j) с неограниченной величиной пропускной способности.

Предположим, что с помощью алгоритма построения потока минимальной стоимости не удается определить поток, удовлетворяющий потребность всех стоков. Пусть через S обозначено множество всех вершин, окрашенных на последней итерации алгоритма построения потока минимальной стоимости. Из рассмотрения алгоритма построения увеличивающего потока, который является частью алгоритма построения потока минимальной стоимости, мы знаем, что по всем дугам, выходящим из некоторой вершины $a \in S$ и входящим в вершины, не принадлежащие множеству S, должен протекать поток, равный величине пропускной способности. Это, однако, невозможно, поскольку пропускная способность некоторых из этих дуг не ограничена. Поэтому в графе таких дуг не может быть, и все дуги с одной окрашенной и одной неокрашенной вершинами направлены к вершинам, принадлежащим множеству S. Следовательно, как только почтальон попадет в любую вершину из множества S, он не сможет выйти из этого множества, а значит, маршрута почтальона не существует. Что и требовалось доказать.

ПРИМЕР. Найдем оптимальный маршрут почтальона для четного несимметричного смешанного графа G, показанного на рис. 6.11. Сначала произвольно выберем направление ориентации каждой из неориентированных дуг графа G. Полученный граф G_d показан на рис. 6.12. В графе G_d:

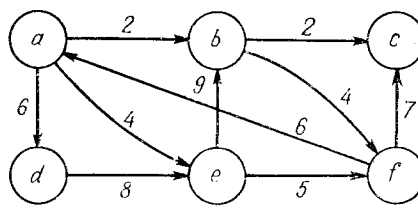


Рис. 6.12. Граф G_d.

$d^+(a) = 3,\ d^-(a) = 1,\ \underline{D}(a) = -2;$ вершина a — сток с величиной суммарного входного потока, равной двум;

$d^+(b) = 2,\ d^-(b) = 2,\ \underline{D}(b) = 0;$ вершина b — промежуточная;

$d^+(c) = 0,\ d^-(c) = 2,\ \underline{D}(c) = 2;$ вершина c — источник, с величиной суммарного выходного потока, равной двум;

$d^+(d) = 1,\ d^-(d) = 1,\ \underline{D}(d) = 0;$ вершина d — промежуточная;

$d^+(e) = 2,\ d^-(e) = 2,\ \underline{D}(e) = 0;$ вершина e — промежуточная;

$d^+(f) = 2,\ d^-(f) = 2;\ \underline{D}(f) = 0;$ вершина f — промежуточная.

Граф G' показан на рис. 6.13. Первое число, расположенное над дугой, указывает ее стоимость. Если пропускная способность дуги является конечной, то ее значение указывается над дугой вторым числом. Необходимо с помощью алгоритма построения потока минимальной стоимости найти поток минимальной стоимости, по которому протекают 2 единицы потока от источника c к стоку a. (Заметим, что величины суммарных потоков, выходящих из каждого источника и входящих в каждый из стоков, являются четными числами и сумма величин потоков, выходящих из источников, равна сумме потоков, входящих в стоки.) Легко проверить, что 2 единицы потока минимальной стоимости протекают от c к a вдоль пути (c, b), (b, f), (f, a). Стоимость прохождения единицы потока равна $0 + 4 + 6 = 10$ или для общего потока — 20. Таким образом, значения оптимального потока в дугах $(c, b)_1$, (b, f), (f, a) равны $f(c, b)_1 = f(b, f) = f(f, a) = 2$, а потоки во всех остальных дугах равны нулю.

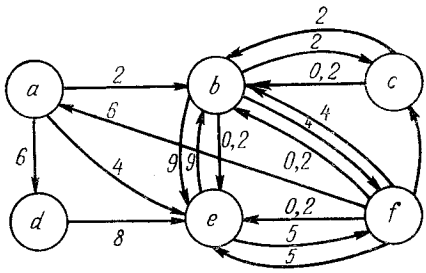


Рис. 6.13. Граф G'.

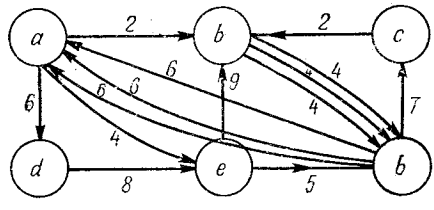


Рис. 6.14. Граф G^*.

Так как $f(c, b)_1 = 2$, мы должны изменить исходную ориентацию этой дуги в графе G_d на противоположную. Первоначальная ориентация всех остальных дуг остается неизменной. Кроме того, дуги (b, f) и (f, a) должны обходиться почтальоном дважды, так как $f(b, f) = f(f, a) = 2$.

Граф G^* показан на рис. 6.14. Заметим, что он является четным симметричным ориентированным графом. Для поиска эйлерова маршрута графа G^* может быть применен метод, рассмотренный в разд. 6.3. Этот маршрут соответствует оптимальному маршруту почтальона на графе G. Поскольку стоимость оптимального потока на графе G' равна 20, то в маршруте почтальона на графе G будут повторно обходиться дуги с общей стоимостью, равной 20.

Случай В. Граф G не является ни четным, ни симметричным. Насколько известно автору, в настоящее время для этого случая отсутствует метод решения задачи почтальона. Эту исходную задачу можно было бы приближенно заменить двухэтапной зада-

чей, при решении которой на первом этапе некоторым оптимальным образом исходный граф преобразуется в четный, а на втором — полученный четный граф с помощью процедур, рассмотренных в этом разделе для случая Б, преобразуется в симметричный. Однако нет гарантий, что оптимальное решение, полученное на каждом из этапов, приведет к оптимальному решению исходной задачи.

УПРАЖНЕНИЯ

1. Городские власти объявили, что с данного момента на определенной улице с двусторонним движением будет введено одностороннее движение. Обязательно ли это приведет к возрастанию протяженности оптимального маршрута почтальона, обслуживающего эту улицу? Сформулируйте условия, при которых это произойдет.

2. Городские власти объявили, что с данного момента на определенной улице с односторонним движением будет введено двустороннее движение. Обязательно ли это приведет к уменьшению протяженности оптимального маршрута почтальона, обслуживающего эту улицу? Сформулируйте условия, при которых это произойдет.

3. Найдите оптимальный маршрут почтальона на графе рис. 6.15.

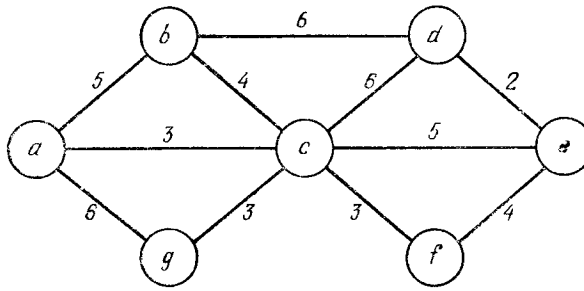


Рис. 6.15.

4. Докажите, что почтальон всегда может выбрать маршрут на неориентированном графе, некоторая дуга которого повторяется не более одного раза. Верно ли это также для ориентированных графов?

5. Покажите, что для решения задачи почтальона на неориентированном графе необходимо не более k итераций алгоритма Флойда или Данцига, где k — число вершин графа, имеющих четную степень. (Необходимо учесть, что вершина с нечетной степенью никогда не может входить в качестве промежуточной вершины в любой путь, включающий повторно обходимые ребра.)

6. Найдите оптимальный маршрут почтальона на ориентированном графе рис. 6.12.

7. Насколько почтальон может изменить протяженность своего маршрута, изменив начальный пункт маршрута?

8. Почтальон хочет начать свой маршрут в определенной вершине и закончить в вершине, отличной от начальной. Каким образом эта задача может быть сведена к задаче почтальона?

9. Почтальон должен обойти улицу Б после обхода улицы А. Каким образом эта задача может быть сведена к задаче почтальона на (а) неориентированном графе, (б) ориентированном графе, (в) смешанном графе?

10. Решите задачу почтальона для смешанного графа рис. 6.11, используя начальную ориентацию неориентированных дуг, отличную от принятой в рассмотренном примере.

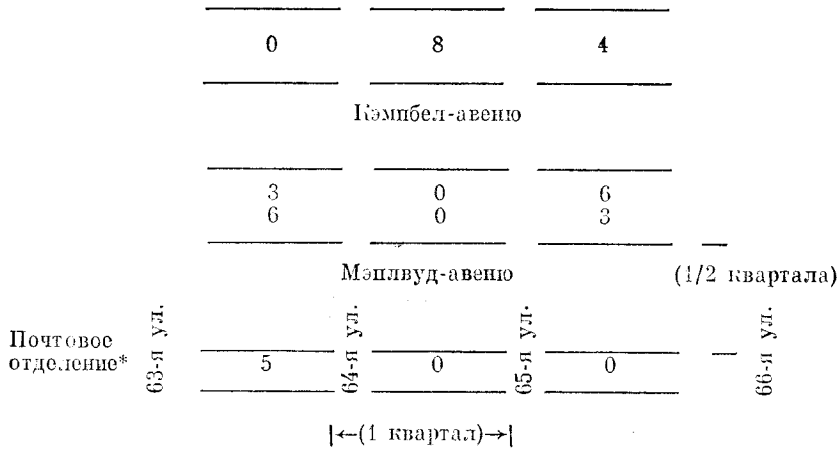
11. При решении задачи почтальона на смешанном графе в оптимальном потоке, полученном с помощью алгоритма построения потока минимальной стоимости, по каждой дуге протекает поток величины, кратной двум. Предположим, что существует другой оптимальный поток, такой, что не по каждой дуге протекает поток величины, кратной двум.

Что можно сказать о фиктивной дуге, по которой протекает поток, равный только одной единице?

(Ответ: эта дуга графа G^* должна быть неориентированной.)

12. Может быть решена задача о Кенигсберских мостах, описанная в разд. 1.1, без повторного обхода хотя бы одного из мостов?

13. Разносчик газет должен доставить газеты подписчикам, проживающим на обеих сторонах Кэмпбел-авеню и Мэплвуд-авеню между 63-й и 66-й улицами. Количество газет, которое он должен доставить на каждую из сторон обеих улиц, показано на диаграмме. Затраты времени разносчика газет распределяются следующим образом: 1 мин. — проезд на велосипеде одного квартала и 6 с — передача газеты каждому подписчику. Он должен начать и закончить свой маршрут на углу 63-й улицы и Мэплвуд-авеню. Каков наилучший маршрут разносчика газет?



ЛИТЕРАТУРА

1. Edmonds J., Johnson Ellis, L., Matching, Euler Tours and the Chinese Postman, *Math. Prog.*, vol. 5, pp. 88 — 124, 1973.

ГЛАВА 7

Задача коммивояжера

7.1. Формулировка и некоторые свойства решений задачи коммивояжера

Коммивояжеру требуется посетить каждый город в пределах конкретной зоны обслуживания и возвратиться домой. Не приходится и говорить, что коммивояжеру хотелось бы выбрать такой порядок обхода клиентов, при котором его путь был возможно короче. Таким образом, коммивояжер сталкивается с задачей поиска маршрута минимальной протяженности, длительности или стоимости.

Задача коммивояжера может быть сформулирована как задача на графе в следующей постановке: построить граф $G(X, A)$, вершины которого соответствуют городам в зоне коммивояжера, а дуги отображают коммуникации, соединяющие пары городов. Пусть длина $a(x, y) \geqslant 0$ каждой дуги $(x, y) \in A$ равна расстоянию, стоимости или времени. Контур, включающий каждую вершину графа G хотя бы один раз, называется маршрутом коммивояжера. Контур, включающий каждую вершину графа G ровно один раз, называется гамильтоновым контуром (по имени ирландского математика Вильяма Роуана Гамильтона, который в 1859 г. первым начал изучение этих задач).

Общей задачей коммивояжера называется задача поиска маршрута наименьшей общей длины.

Задачей коммивояжера называется задача поиска гамильтонова контура наименьшей общей длины.

ПРИМЕР 1. Курьер банка ежедневно должен доставлять письма из отделения банка, в котором он служит, во все другие отделения банка. Обычно, когда он приносит письма в какое-либо отделение, ему вручают дополнительную корреспонденцию, подлежащую доставке в следующее на его пути отделение. Отрицательно относясь к этой дополнительной работе, курьер хотел бы узнать, в каком порядке ему следует посещать пункты назначения, чтобы минимизировать общее число дополнительно разносимых писем. Курьер решает свою задачу путем решения общей задачи коммивояжера на графе, вершины которого соответствуют отделениям банка, а дуги — возможным переездам между ними. Пусть в этом графе длина дуги (x, y) равна прогнозируемому количеству дополнительных писем, которое курьеру пришлось бы перевезти из отделения x в отделение y. Предположим, что курьер хочет продвинуться по службе и для этого должен максимально увеличить общее количество дополнительно доставляемых писем. В этом случае он мог бы, конечно, беспрестанно перемещаться между всеми отделениями банка и в итоге доставить практически неограниченное количество писем. Однако предположим, что ему разрешается каждое отделение посетить только один раз. Пусть M равно некоторому очень большому числу

и пусть длина каждой дуги теперь равна M минус ее первоначальная длина. Если имеется n отделений банка, то каждый гамильтонов контур графа включает n дуг. Теперь задача курьера решается путем поиска гамильтонова контура, имеющего наименьшую общую длину.

ПРИМЕР 2. В цехе, имеющем n различных станков, изделие должно быть обработано на каждом из n станков в некоторой произвольной последовательности. При этом время переналадки, необходимое для передачи изделия от станка x к станку y, равно $a(x, y)$. Какую выбрать последовательность обработки на каждом из станков, для того чтобы минимизировать общее время обработки изделия?

Эта *задача решается как задача* коммивояжера на графе G, каждая вершина которого соответствует станку, а каждая пара (x, y) вершин соединяется дугой, имеющей длину $a(x, y)$. Контур коммивояжера, имеющий наименьшую длину, называется *оптимальным маршрутом коммивояжера* и является оптимальным решением общей задачи коммивояжера. Гамильтонов контур наименьшей длины называется *оптимальным гамильтоновым контуром* и является оптимальным решением задачи коммивояжера.

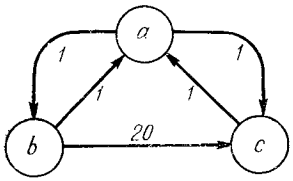


Рис. 7.1. Оптимальный путь коммивояжера.

Оптимальный маршрут коммивояжера не обязательно является гамильтоновым контуром. Например, рассмотрим граф, показанный на рис. 7.1. В этом графе имеется только один гамильтонов контур (a, b), (b, c), (c, a) с общей длиной, равной $20 + 1 + 1 = 22$. Оптимальный же маршрут коммивояжера (a, b), (b, a), (a, c), (c, a) проходит через каждую вершину дважды и имеет общую длину $1 + 1 + 1 + 1 = 4$.

Возникает вопрос, в каких случаях гамильтонов контур является решением общей задачи коммивояжера?

ТЕОРЕМА 7.1. Если для каждой пары вершин (x, y) графа G выполняется условие

$$a(x, y) \leqslant a(x, z) + a(z, y) \text{ для всех } z \neq x,\ z \neq y, \tag{1}$$

то гамильтонов контур является решением общей задачи коммивояжера на графе G (если решение вообще существует).

Условие (1) говорит только о том, что расстояние непосредственно от x до y никогда не превышает расстояния от x до y через любую другую вершину z. Условие (1) называется неравенством треугольника.

Доказательство. Предположим, что не существует оптимального решения общей задачи коммивояжера, представляющего собой гамильтонов контур. Пусть C — любой оптимальный маршрут коммивояжера. Так как C не является гамильтоновым конту-

ром, то некоторая вершина, скажем вершина z, повторяется в маршруте C по крайней мере дважды. Предположим, что первый раз коммивояжер в вершину z пришел из вершины x и вышел из нее в направлении вершины y. Изменим маршрут C таким образом, чтобы коммивояжер из вершины x, минуя z, направился прямо к вершине y. Полученный в результате маршрут C' также является маршрутом коммивояжера, поскольку в нем каждая вершина посещается по крайней мере один раз. Кроме того, в соответствии с условием (1) общая длина C' не превышает длины C. Заменяя C на C' и повторяя эту процедуру, мы получим другой маршрут C'' и т. д. В конце концов этот процесс приведет нас к оптимальному маршруту, являющемуся гамильтоновым контуром, так как каждый последующий маршрут имеет число дуг на единицу меньшее, чем его предшественник. Теорема доказана.

Из теоремы 7.1 следует, что если граф G удовлетворяет неравенству треугольника, то оптимальное решение задачи коммивояжера на графе G является оптимальным решением для общей задачи коммивояжера на этом графе.

Есть простой способ избежать разработки двух разных методов решения для задачи коммивояжера и общей задачи коммивояжера. Если граф G не удовлетворяет неравенству треугольника, то следует заменить длину $a(x, y)$ каждой дуги (x, y), не удовлетворяющей этому неравенству, длиной кратчайшего пути от вершины x к вершине y.

Заметьте, что длина дуги от x к y отображает теперь передвижение коммивояжера от вершины x к вершине y вдоль кратчайшего пути, соединяющего эти вершины, а не напрямую из x в y, и длина удовлетворяет неравенству треугольника. Если оптимальное решение задачи коммивояжера на графе G включает некоторую дугу (x, y), длина которой уменьшена описанным выше способом, то в оптимальном решении эту дугу следует заменить дугами, входящими в кратчайший путь между вершинами x и y. Таким образом, нам необходимо разработать метод решения только для задачи коммивояжера. Например, на рис. 7.1 $a(b, c) = 20 > a(b, a) > a(a, c) = 1 + 1$. Отсюда видно, что для дуги (b, c) условие (1) не выполняется. Если длину дуги (b, c) уменьшить до 2 (длины кратчайшего пути от b к c), то в полученном графе появится только один гамильтонов контур (a, b), (b, c), (c, a), длина которого равна $1 + 2 + 1 = 4$. Заменив дугу (b, c) дугами (b, a), (a, c), получаем маршрут (a, b), (b, a), (a, c), (c, a), который является оптимальным маршрутом коммивояжера на исходном графе.

Предположим, что мы желаем вместо гамильтонова контура наименьшей длины найти гамильтонов контур, имеющий максимальную длину. Например, банковский курьер пожелает макси-

мизировать, а не минимизировать общее количество доставляемых им дополнительных писем. Может ли эта задача быть сведена к задаче коммивояжера?

Пусть через M обозначена наибольшая длина дуги в графе G. Пусть далее

$$a'(x, y) = M - a(x, y) \geqslant 0 \text{ для всех дуг } (x, y). \qquad (2)$$

Каждый гамильтонов контур на графе $G = (X, A)$ содержит ровно $|X|$ дуг. Оптимальный (в смысле наименьшей длины) гамильтонов контур C включает дуги, преобразованные в соответствии с соотношением (2), и его общая длина равна

$$\sum_{(x, y) \in C} a'(x, y) = \sum_{(x, y) \in C} [M - a(x, y)] = M \mid X \mid = \sum_{(x, y) \in C} a(x, y)$$

Таким образом, гамильтонов контур минимальной длины, состоящий из преобразованных дуг, эквивалентен гамильтонову контуру максимальной длины, составленному из тех же дуг с исходной (не преобразованной) длиной.

Следовательно, для нахождения гамильтонова контура максимальной длины достаточно решить задачу коммивояжера, предварительно преобразовав длины дуг графа в соответствии с (2). Однако не все графы имеют гамильтонов контур. Например, граф, состоящий только из двух вершин x и y и одной дуги (x, y), такого контура не имеет. Условия существования в графе гамильтонова контура описываются в разд. 7.2. В разд. 7.3 рассматриваются методы вычисления нижних границ длины оптимального гамильтонова контура. И наконец, в разд. 7.4 описываются методы отыскания оптимального гамильтонова контура.

7.2. Условия существования гамильтонова контура

Как показано в разд. 7.1, решением задачи коммивояжера является оптимальный гамильтонов контур. К сожалению, не все графы содержат гамильтонов контур. Следовательно, прежде чем приступить к поиску оптимального гамильтонова контура, мы должны по крайней мере попытаться установить факт его наличия в данном графе. Весьма глубокое рассмотрение условий существования гамильтоновых контуров в графах можно найти в книге Бержа [2].

Введем ряд определений, которые впоследствии нам понадобятся. Граф называется сильно связным, если в нем для любых двух вершин x и y существует путь от x к y. Подмножество вершин x_i некоторого графа называется сильно связным, если для любых пар вершин $x \in x_i$ и $y \in x_i$ существует путь из x в y и x_i не является подмножеством никакого другого множества вершин, обладающего такими же свойствами.

Подграф, порожденный сильно связным подмножеством вершин, называется сильно связной компонентой исходного графа. Например, граф, показанный на рис. 7.1, является сильно связным графом, так как имеется путь от каждой вершины к лю-

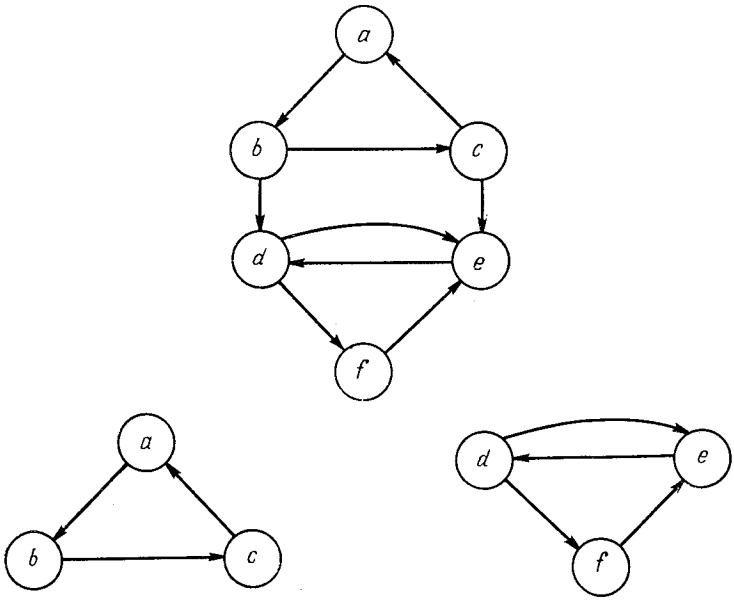


Рис. 7.2. Граф и его сильно связные компоненты.

бой другой вершине этого графа. Читателю предоставляется возможность проверить это самостоятельно.

Рассмотрим далее граф, изображенный на рис. 7.2. Этот граф не является сильно связным, так как на нем отсутствует путь из вершины d к вершине b, хотя существует цепь от d к b, представляющая собой дугу (d, b). Вершины $\{a, b, c\}$ образуют сильно связное подмножество, так как существует путь от каждой из этих вершин к любой другой вершине этого же подмножества. Кроме того, ни одна дополнительная вершина не может быть к нему добавлена без потери этого свойства. Например, вершина d не может быть добавлена по той причине, что не существует пути от вершины d к вершине a. Подграф, порожденный подмножеством $\{a, b, c\}$, показан на рис. 7.2. Этот подграф является сильно связной компонентой исходного графа.

Рассмотрим подмножество вершин $\{d, e\}$. На графе существует путь от d к e и от e к d. Однако $\{d, e\}$ не является сильно связным подмножеством вершин, так как к нему может быть добавлена

вершина f без потери свойства сильной связности. В то же время к множеству $\{d, e, f\}$ не может быть добавлено ни одной вершины без потери этого свойства. Следовательно, $\{d, e, f\}$ является сильно связным подмножеством вершины. Сильно связный компонент графа, полученный из $\{d, e, f\}$, также показан на рис. 7.2. Если граф не является сильно связным, то на нем отсутствует гамильтонов контур. Это следует из того, что гамильтонов контур включает путь между каждой парой вершин графа. Таким образом, необходимым условием существования гамильтонова контура является наличие в графе сильной связности. Напомним, что петлей является любая дуга, начальная и конечная вершины которой совпадают, т. е. дуга вида (x, x). Гамильтонов контур не может содержать петлю, и, следовательно, на существование гамильтонова контура в графе G не влияет добавление или исключение петель. Если две вершины, скажем x и y, соединены более чем одной дугой $(x, y)_1$, $(x, y)_2$... одинакового направления, то исключение всех дуг, кроме одной дуги с наименьшей длиной, не влияет ни на существование гамильтонова контура, ни на длину оптимального гамильтонова контура графа (если он существует).

По этой причине мы впредь будем предполагать, что в графе G отсутствуют петли и что в нем имеется не более одной дуги от x к y для любых x и y.

Пусть через $D^-(x)$ обозначено множество всех таких вершин графа $G = (X, A)$, что $(x, y) \in A$, т. е. множество всех вершин, «падающих на» вершину x. Пусть через $D^+(x)$ обозначено множество всех таких вершин y, что $(x, y) \in A$, т. е. множество всех вершин, «падающих из» вершины x.

Вершины, принадлежащие множествам $D^-(x)$ и $D^+(x)$, называются соответственно предшественниками и преемниками вершины x. Пусть $D(x) = D^-(x) \nleftrightarrow UD^+(x)$. Как и ранее, пусть $d^-(x) = |D^-(x)|$, $d^+(x) = |D^+(x)|$ и $d(x) = |D(x)|$. И наконец, пусть n — число вершин в графе G. Имея в виду эти определения, мы можем сформулировать следующую весьма общую теорему Гуйя-Ури [6].

ТЕОРЕМА 7.2. Если граф $G(X,A)$ удовлетворяет условиям:
I) граф G сильно связный,
II) $d(x) \geqslant n$ для всех $x \in X$,
то он содержит гамильтонов контур.

Доказательство. Доказательство проводится методом индукции по числу n вершин в графе G.

Справедливость теоремы для $n = 2$ и $n = 3$ очевидна. Мы будем предполагать, что теорема справедлива для всех графов с числом вершин меньше $n (n > 3)$ и использовать это предположение для того, чтобы показать справедливость теоремы также и для всех графов с числом вершин n.

Пусть G — произвольный граф с n вершинами, удовлетворяющий условиям (I) и (II). Пусть через C обозначен простой контур на графе G с наибольшим возможным числом дуг. Пусть $x_1, x_2, \ldots$ x_m — последовательность, в которой коммивояжер посещает вершины графа G, входящие в контур C. Если $m = n$, то теорема справедлива. В противном случае $m < n$ и C не является гамильтоновым контуром. Пусть $X_0 = \{x_1, x_2, \ldots, x_m\}$. Обозначим через $X_1, X_2, \ldots, X_p$ сильно связные компоненты подграфа, порожденного подмножеством вершин $X - X_0$.

Утверждение (а). Каждый сильно связный компонент $X_1, X_2, \ldots, X_p$ содержит гамильтонов контур.

Доказательство. Достаточно показать, что степень каждой вершины $x \in X_i$ по крайней мере равна $|x_i|$ для всех $i = 1, 2, \ldots p$. Если это так, то компонент X_i будет удовлетворять условиям (I) и (II) в соответствии с индуктивным предположением содержит гамильтонов контур. Мы знаем, что $d(x) \geqslant n$. Рассмотрим произвольные вершины $x \in X_i$ и $y \in X_j$, $j \neq i$. Дуги (x,y) и (y, x) не могут одновременно принадлежать A; в противном случае X_i и X_j образовали бы один сильно связный компонент. Для $k = 1, 2, \ldots, m$ не могут одновременно существовать дуги (x, x_k) и (x_k, x), так как в противном случае контур C мог бы быть расширен путем включения в него вершины x. Следовательно, число дуг, соединяющих любую вершину $x \in X_i$ с вершинами, не принадлежащими X_i, не может превышать $|X| - |X_i|$. Таким образом, степень вершины $x \in X_i$ в X_i не меньше чем $|X_i|$. Утверждение (а) доказано.

Утверждение (б). Для $i = 1, 2, \ldots, p$ $|X_i| \leqslant |X_0|$.

Доказательство. Если бы это утверждение было неверным, то в X_i должен был бы существовать гамильтонов контур, содержащий большее количество дуг, чем C, что противоречит принятым относительно C предположениям.

Утверждение (в). Каждую вершину $x \in X_i$ соединяет с вершинами множества X_0 не менее $2 + |X_0| - |X_i| \geqslant 2$ дуг.

Доказательство. Как уже упоминалось в утверждении (а), вершина $x \in X_i$ соединяется с вершинами в X_j не более чем $|X_j|$ дугами для всех $j \neq i$, $j \neq 0$. Кроме того, вершина $x \in X_i$ может соединяться с другими вершинами в X_i не более чем $2(|X_i| - 1)$ дугами. Поскольку $d(x) \geqslant n$, по крайней мере $2 + |X_0| - |X_i|$ дуг соединяют вершину $x \in X_i$ с вершинами в X_0, и, следовательно, утверждение (в) справедливо.

Утверждение (г). В графе G существует такой компонент X_i, что для него имеется пара дуг, одна из которых направлена из некоторой вершины в X_i к вершине в X_0, а другая направлена из некоторой вершины в X_0 к вершине в X_i.

Доказательство. Для $p = 1$ утверждение должно быть верно в силу того, что G — сильно связный граф.

В случае, когда $p>1$, предположим, что все дуги, соединяющие X_1 и X_0, направлены от X_1 к X_0. Рассмотрим подграф, порожденный объединением множеств вершин $X_1 \cup X_0$. Так как некоторая вершина в X_1 соединяется не более чем одной дугой с каждой вершиной, не принадлежащей $X_1 \cup X_0$, то вершины в X_1 удовлетворяют условию (II) в этом подграфе. Из доказательства утверждения (а) следует, что вершины в X_0 также удовлетворяют условию (II) в этом подграфе. Поскольку граф G является сильно связным, то имеется такой путь P из вершины $x \in X_0$ к вершине $y \in X_1$, что его промежуточные вершины не принадлежат $X_1 \cup X_0$. Путь P должен содержать по крайней мере две дуги. Если к подграфу, порожденному множеством вершин $X_0 \cup X_1$, была бы добавлена дуга (x, y), то этот подграф стал бы сильно связным. Следовательно, этот подграф совместно с дугой (x, y) удовлетворял бы индуктивным предположениям и поэтому на нем существовал бы гамильтонов контур, содержащий дугу (x, y). Заменим в подграфе дугу (x, y) дугами, входящими в путь P. В результате получим контур, содержащий большее количество дуг, чем контур C, что невозможно. Следовательно, каждый компонент $X_1, X_2, \ldots, X_p$ должен быть соединен с X_0 дугами обоих направлений. Утверждение (г) доказано.

Утверждение (*д*). Если X_1 — компонент, удовлетворяющий утверждению (г), то для каждой вершины $y \in X_1$ имеются дуга, направленная из y к вершине в X_0, и дуга, направленная из вершины в X_0 к вершине y.

Доказательство. Пусть C_1 — гамильтонов контур на X_1. Пусть последовательность $y_1, y_2, \ldots, y_q$ обозначает порядок, в котором коммивояжер посещает вершины в X_1. (Контур C_1 существует в соответствии с утверждением (а).) Предположим, что не существует ни одной дуги, выходящей из любой вершины в X_0 к вершине y_j. Будем обходить контур C_1 по вершинам $y_{j+1}, y_{j+2}, \ldots$ до тех пор, пока не встретится такая вершина y_{j+t}, что существует дуга (x_k, y_{j+t}), начальная вершина которой принадлежит X_0. Рассмотрим вершину y_{j+t-1}. Ни одна дуга не может быть направлена из y_{j+t-1} к вершинам $x_{k+1}, x_{k+2}, \ldots, x_{k+q}$. В противном случае контур C должен был бы слиться с $q-1$ дугой контура C_1 от вершины y_{j+t} к вершине y_{j+t-1}, образуя контур с числом дуг, превышающим m, что невозможно. Следовательно, ни одна из дуг не может быть направлена из вершины X_0 к вершине y_{j+t-1} и не более $m-q$ дуг направлены из вершины y_{j+t-1} к вершинам в X_0. Так как $m = |X_0|$ и $q = |X_1|$, то не выполняется соотношение, доказанное в утверждении (в). Очевидно, что предположение об отсутствии дуги, идущей из вершины X_0 в вершину y_j, приводит к противоречию. Таким образом, утверждение (д) справедливо.

Утверждение (*е*). (*Заключение.*) Как и ранее, пусть последо-

вательность $x_1, x_2, \ldots, x_m$ обозначает порядок посещения коммивояжера вершин в C и пусть последовательность $y_1, y_2, \ldots, y_q$ определяет порядок посещения вершин в C_1. Пусть

$$a(i, j) = \begin{cases} 1, \text{ если } (x_i, y_j) \in A, \\ 0 \text{ в противном случае,} \end{cases}$$

$$b(i, j) = \begin{cases} 1, \text{ если } (y_{j-1}, x_{i+1}) \in A, \\ 0 \text{ в противном случае.} \end{cases}$$

Так как в соответствии с утверждением (в) имеется по крайней мере $q(m - q + 2)$ дуг, соединяющих X_0 и X_1, то справедливо соотношение

$$\sum_{i,j} a(i,j) + b(i, j) \geqslant q(m - q + 2).$$

Для некоторого j_0

$$\sum_{i} a(i, j_0) + b(i, j_0) \geqslant m - q + 2. \tag{3}$$

Из утверждения (д) следует, что имеется по крайней мере одна дуга (x_{i0}, y_{j0}), направленная из X_0 к вершине $y_{j0} \in X_1$, но поскольку контуров, содержащих больше чем m дуг, не существует, то справедливо, что $b(i_0, j_0) = 0$, $b(i_0 + 1, j_0) = 0$, ..., $b(i_0 + q - 1, j_0) = 0$.

Таким образом, имеется не более $m - q + 1$ дуг, направленных из вершины y_{j_0-1} к вершинам множества X_0. Кроме того, если $a(i_0 + 1, j_0) = 1$, то $b(i_0 + q, j_0) = 0$; если $a(i_0 + 2, j_0) = 1$, то $b(i_0 + q + 1, j_0) = 0$ и т. д. Итак, наличие каждой дополнительной дуги, направленной из вершины в X_0 к вершине y_{j0}, препятствует существованию по крайней мере одной дополнительной дуги, исходящей из вершины y_{j0} к вершинам в X_0. Следовательно,

$$\sum_{i} a(i, j_0) + b(i, j_0) \leqslant m - q + 1,$$

что противоречит неравенству (3). Значит, контур C не может включать меньше чем n дуг. Теорема доказана.

На практике достаточно легко установить, удовлетворяет ли некоторый граф условиям (I) и (II) теоремы 7.2.

Условие (I) проверяется применением к каждой паре вершин алгоритма Флойда или алгоритма Данцига для построения кратчайшего пути (см. гл. 3). Условие (I) выполняется, если для каждой пары вершин в графе существует связывающий их путь конечной длины.

Условие (II) легко проверяется подсчетом числа дуг, инцидентных каждой вершине графа.

Если граф G — неориентированный, то коммивояжеру разрешается обходить дуги в произвольном направлении. В этом случае решение задачи коммивояжера состоит в нахождении гамильтонова цикла, т. е. некоторого простого цикла наибольшей длины, проходящего через все вершины графа.

Достаточные условия существования гамильтонова цикла на неориентированном графе сформулированы Квателем [4]. Пусть, как и ранее, через n обозначено число вершин в графе. Пронумеруем вершины так, чтобы выполнялось соотношение

$$d(x_1) \leqslant d(x_i) \leqslant \ldots \leqslant d(x_n).$$

ТЕОРЕМА 7.3. Граф $G = (X, E)$ содержит гамильтонов цикл, если $n \geqslant 3$ и

$$d(x_k) \leqslant k \leqslant \frac{1}{2} n \Rightarrow d(x_{n-k}) \geqslant n - k. \tag{4}$$

Доказательство. Предположим, что граф G удовлетворяет условиям (4) теоремы, но не содержит гамильтонова цикла. Рассмотрим произвольное ребро (x_i, x_j), которое не принадлежит графу. Если добавление этого ребра не приводит к образованию гамильтонова цикла, то включим его в граф G. Повторим эту процедуру до тех пор, пока ни одно из ребер нельзя будет дополнительно включить в граф G. Заметим, что после добавления каждого нового ребра в граф G условие (4) остается допустимым, так как его включение в граф не понижает степени любой из вершин. Назовем граф, получаемый в результате добавления ребер, графом $G^* = (X, E^*)$. В дальнейшем мы используем этот граф G^* для того, чтобы показать наличие противоречия при принятом предположении. Пусть u и v — такие произвольные несмежные вершины графа, что $d(u) + d(v)$ имеет максимальное значение. Предположим без нарушения общности, что $d(u) \leqslant d(v)$. Пусть C — простая цепь наибольшей длины, соединяющая вершины u и v. Поскольку в граф G^* нельзя дополнительно включить ни одной дуги без образования гамильтонова цикла, то C включает все вершины в множестве X. Пусть последовательность $u = u_1, u_2, \ldots, u_{n-1}, u_n = v$ определяет порядок, в котором вершины обходятся в цепи C.

Пусть через S обозначено множество всех таких вершин u_i, что вершина u_{i+1} смежна с вершиной u.

Через T обозначим множество всех вершин, смежных с вершиной v. Отметим, что ни одна из вершин y_j не может одновременно принадлежать S и T, иначе цикл $u_j, u_{j-1}, \ldots, u_1, u_{j+1}, u_{j+2}, \ldots u_n, u_j$ был бы гамильтоновым циклом. Кроме того, $S \cup T = \{u_1, u_2, \ldots, u_{n-1}\}$. Поэтому $d(u) + d(v) = |S| + |T| \leqslant n$. Следовательно, $d(u) \leqslant \frac{1}{2} n$. Поскольку ни одна из вершин u_j не может одновременно входить в S и T, то u_j и v не смежны, если $u_j \in S$. Из максимальных $d(u) + d(v)$ следует, что $d(u_j) \leqslant d(u)$.

Таким образом, существует по крайней мере $|S|$ вершин, степень которых не превышает степени вершины u. Примем $k = d(u)$. Значит, $d(x_k) \leqslant k$. Из условия (4) следует, что $d(x_{n-k}) \geqslant n - k$. Таким образом, должно быть по крайней мере $k + 1$ вершин, степень которых не меньше чем $(n - k)$. Так как $d(u) = k$, то вершина u не смежна ни с одной из этих вершин. Значит, существует некоторая вершина w, которая не смежна с u и при этом $d(w) \geqslant n - k$. Отсюда $d(u) + d(w) > d(u) + d(v)$, что противоречит принятому предположению. Следовательно, граф G^* должен содержать гамильтонов цикл. Теорема доказана.

Условие (4) легко проверить. Необходимо упорядочить вершины по возрастанию их степеней и проверить, выполняется ли условие для первых $^n/_2$ вершин.

7.3. Нижние границы

Рассмотрим теперь методы расчета нижних границ длины оптимальных гамильтоновых контуров на ориентированном графе $G = (X, A)$. Если из длины любого гамильтонова контура вычесть значение нижней границы, то получаемая разность равна максимальному значению, на которое длина данного гамильтонова контура может превышать длину оптимального гамильтонова контура. Знание этой величины полезно для оценки разности длин найденного и оптимального гамильтонова контуров.

Дуги, входящие в гамильтонов контур, должны обладать следующими двумя свойствами:

1. Каждая вершина должна быть инцидентна двум дугам, одна из которых направлена к ней, а другая от нее.

2. Все дуги должны быть связны. Дуги, входящие в любое множество не связных контуров, содержащих все вершины, обладают свойством 1. Любое связное множество дуг обладает свойством 2. И только гамильтонов контур обладает свойствами 1 и 2 одновременно.

Рассмотрим семейство F всех подмножеств множества A, обладающих свойством 1. (Мы будем предполагать, что F не пусто.) Понятно, что каждый гамильтонов контур входит в F. Таким образом, нижней границей длины оптимального гамильтонова контура является наименьшее значение суммы длин дуг, образующих подмножества в F. Обозначим через L_1 эту нижнюю границу. Нижняя граница оптимального гамильтонова контура может быть вычислена следующим образом.

Шаг 1. Строится граф $G' = (X', A')$. Для этого каждой вершине $x \in X$ ставится в соответствие пара вершин $x_1 \in X'$ и $x_2 \in X'$. Каждой дуге $(x, y) \in A$ ставится в соответствие промежуточная дуга $(x_1, y_2) \in A'$. Пусть каждая промежуточная дуга

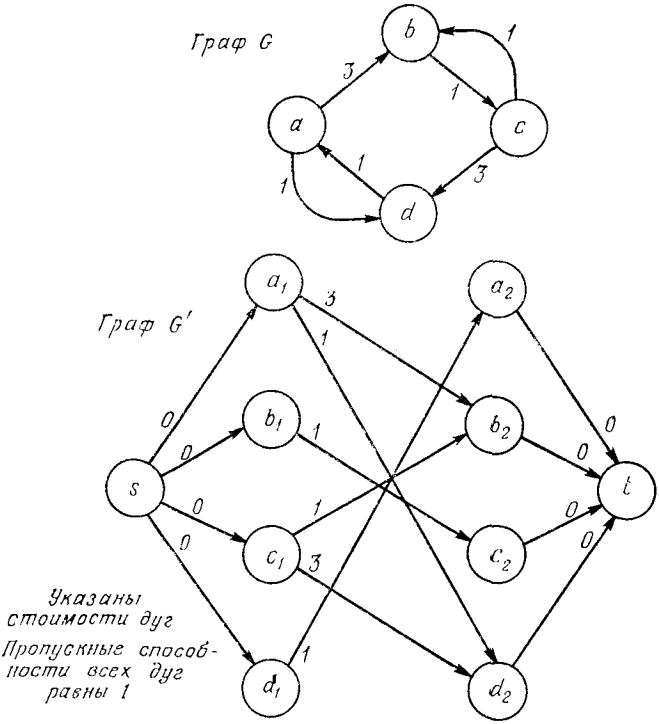


Рис. 7.3. Расчет нижних границ длины оптимального гамильтонова контура.

имеет пропускную способность, равную 1, и стоимость, равную длине соответствующей дуги в A. Введем вершину-источник s и вершину-сток t и соединим вершину s направленными из нее дугами (s, x_1) с каждой из вершин $x_1 \in X'$. Соединим далее вершину t с каждой из вершин $x_2 \in A'$ направленными в нее дугами (x_2, t). Пусть каждая из дуг (s, x_1) и (x_2, t) для всех x_1 и $x_2 \in X'$ имеет пропускную способность, равную единице, и нулевую стоимость (рис. 7.3).

Шаг 2. Определим на графе G' максимально возможный поток из s в t, имеющий минимальную общую стоимость. Такой поток находится с помощью алгоритма построения потока минимальной стоимости (см. разд. 4.3). Пусть L_1 равна стоимости полученного потока. Так как пропускные способности всех дуг в графе G' равны единице, полученный поток состоит из единичных потоков, протекающих в дугах графа G'' от вершины s к вершине t. Кроме того, каждая из промежуточных вершин (т. е. всех вершин, за исключением s и t) инцидентна не более чем одной дуге,

по которой протекает единичный поток. Далее, каждой промежуточной дуге в A' соответствует некоторая дуга в A. Рассмотрим множество всех промежуточных дуг, по которым протекает единичный поток. Пусть через M обозначено соответствующее множество дуг в A. Дуги в множестве M образуют не связные контуры в G, которые включают все вершины графа G. В противном случае существовала бы некоторая вершина $x \in X$, которая не была бы инцидентна направленной к ней или из нее дуге в M. Это соответствовало бы тому, что в графе G' в вершину x_2 не входили бы или из вершины x_1 не выходил бы ни один из единичных потоков. В каждом из этих двух случаев суммарный поток из s в t состоял бы менее чем из $|X|$ единичных потоков. Но это невозможно, так как каждому подмножеству в $\overline{F}$ соответствует поток в G', равный $|X|$ единиц, и, кроме того, множество $\overline{F}$ не пусто. Поскольку непулевую стоимость имеют только промежуточные дуги, то суммарная стоимость найденного потока равна сумме длин дуг, входящих в соответствующие подмножества в $\overline{F}$. Поскольку найденный поток имеет наименьшую суммарную стоимость, то соответствующее ему подмножество в $\overline{F}$ должно иметь минимальную сумму длин дуг. Таким образом, L_1 является нижней границей длины оптимального гамильтонова контура в графе G.

Если в граф G входят неориентированные дуги, то каждая неориентированная дуга может быть заменена парой противоположно направленных дуг, соединяющих ту же пару вершин. Пусть каждая из этих направленных дуг имеет длину, равную длине исходной неориентированной дуги. Получаемый в результате граф будет содержать только направленные дуги, и для него может быть применен описанный выше метод вычисления L_1.

Если множество M образует гамильтонов контур, то мы нашли не только нижнюю границу длины оптимального гамильтонова контура, но сам этот оптимальный контур.

Читатель, хорошо знакомый с методами решения задач линейного программирования, заметит, что граница L_1 может быть также найдена путем решения задачи о назначениях. Кристофидесом [3] разработан метод получения лучшей оценки границы L_1. Этот метод использует решение задачи о назначениях.

ПРИМЕР 1. Требуется вычислить нижнюю границу длины оптимального гамильтонова контура в графе G, представленном на рис. 7.3. Граф G' показан также. Очевидно, что из источника s в сток t в графе G' может протекать не более четырех единиц потока, так как пропускная способность всех направленных к стоку дуг равна четырем. При этом стоимость каждой промежуточной дуги не ниже единицы. Поскольку граф G' имеет небольшую размерность, поток минимальной стоимости удобнее найти посредством простого просмотра, а не с помощью алгоритма построения потока минимальной стоимости.

Поток минимальной стоимости имеет следующие характеристики:

Количество	Путь	Стоимость
1 единица	(s, a_1), (a_1, d_2), (d_2, t)	0+1+0=1
1 единица	(s, d_1), (b_1, c_2), (c_2, t)	0+1+0=1
1 единица	(s, c_1), (c_1, b_2), (b_2, t)	0+1+0=1
1 единица	(s, d_1), (d_1, a_2), (a_2, t)	0+1+0=1

Суммарная стоимость этого потока равна четырем, следовательно, $L_1 = 4$. Дугами в графе G, соответствующими дугам с ненулевым потоком в графе G', являются (a, d), (d, a), (b, c), (c, b). Заметим, что вершины в этих контурах обладают свойством 1, но не обладают свойством 2. Поэтому множество этих дуг входит в F. Граф G имеет единственный гамильтонов контур (a, b), (b, c), (c, d), (d, a) с суммарной длиной $3 + 1 + 3 + 1 = 8$. Вычислим теперь в графе G другую нижнюю границу L_2 длины оптимального гамильтонова контура. Эта нижняя граница вычисляется при ослаблении требований свойства 1, а именно не определяется заранее, сколько ориентированных дуг, инцидентных каждой вершине, должно быть направлено из этой вершины.

Напомним, что согласно определению, данному в гл. 2, ориентированным деревом является такое дерево, в котором в любую вершину направлена не более чем одна дуга. Единственная вершина в ориентированном дереве, в которую не направлены ни одна из инцидентных ей дуг, называется корнем ориентированного дерева. Рассмотрим произвольное множество дуг, являющихся покрывающим ориентированным деревом графа G с корнем в некоторой вершине, скажем в вершине x, совместно с некоторой дугой, направленной в эту вершину. Пусть через H обозначено семейство всех таких множеств дуг (предполагается, что H не пусто). Каждое множество дуг в H обладает свойством 2, но не обязательно обладает свойством 1. В то же время каждый гамильтонов контур на графе G принадлежит H. Для нахождения в графе G покрывающего ориентированного дерева наименьшей суммарной длины с корнем в вершине x может быть использован алгоритм построения максимального ориентированного леса, рассмотренный в гл. 2.

Присоединим к этому ориентированному дереву входящую в вершину x дугу наименьшей длины. Назовем полученное множество дуг множеством T. Множество T должно иметь минимальную суммарную длину среди всех множеств, входящих в H. Следовательно, суммарная длина дуг в множестве T является нижней границей длины оптимального гамильтонова контура на графе G. Эту нижнюю границу будем обозначать через L_2.

Если множество дуг T образует гамильтонов контур, то этот контур является оптимальным гамильтоновым контуром.

ПРИМЕР 2. Определим нижнюю границу L_2 длины гамильтонова контура для графа, изображенного на рис. 7.3. Поскольку этот граф имеет небольшую размерность, то минимальное покрывающее дерево с корнем в вершине a легче найти простым просмотром графа, а не с помощью алгоритма построения максимального ориентированного леса. Таким деревом является множество дуг (a, b), (b, c) и (a, d) с суммарной длиной, равной $3 + 1 + 1 = 5$. Дугой, имеющей наименьшую длину и направленной в вершину a, является дуга (d, a) с длиной, равной 1. Таким образом, множество дуг T включает дуги (a, b), (b, c), (a, d), (d, a), суммарная длина которых равна $3 + 1 + 1 + 1 = 6$, т. е. $L_2 = 6$. Заметим, что полученное множество T не образует гамильтонова контура. Вспомним из предыдущего примера, что $L_1 = 4 \neq L_2$, т. е. L_1 не всегда равно L_2.

Если граф G является неориентированным графом, то процедура вычисления L_2 может быть упрощена. Для этого удалим из графа произвольную вершину x. Используем затем алгоритмы построения максимального покрывающего дерева (гл. 2) для нахождения минимального покрывающего дерева на графе, полученном в результате удаления вершины x. Выберем две дуги, инцидентные вершине x и имеющие наименьшую длину; присоединим эти дуги к покрывающему дереву. Назовем полученное множество дуг множеством U. Множество U обладает свойством 2, но не обязательно обладает свойством 1. Суммарная длина дуг, входящих в U, дает нижнюю границу L_2 длины оптимального гамильтонова цикла в неориентированном графе. Следовательно, для неориентированных графов при определении L_2 алгоритм построения максимального ориентированного леса может быть заменен менее сложным алгоритмом построения максимального покрывающего дерева. Припишем каждой вершине x число $u(x)$. Пусть

$$a'(x, y) = a(x, y) + u(x) + u(y) \text{ для всех } (x, y). \qquad (5)$$

В результате такого преобразования длина каждого гамильтонова цикла изменилась на одну и ту же величину. Таким образом, гамильтонов цикл наименьшей длины на графе, длина ребер которого равна $a(x, y)$, будет также гамильтоновым циклом наименьшей длины на том же графе со значением длин ребер, равным $a'(x, y)$. Хелд и Карп [7, 8] разработали методы нахождения таких значений $u(x)$, $x \in X$, что получаемое с их учетом множество U является гамильтоновым циклом. Эти методы основаны на решении задач линейного программирования, и их дальнейшее обсуждение увело бы нас слишком далеко.

7.4. Методы решения задачи коммивояжера

Известны многие различные методы решения задачи коммивояжера. Среди них можно выделить методы, разработанные Беллмором и Немхаузером [1], Гарфинкелем и Немхаузером [5], Хел-

дом и Карпом [8], Стекханом [9]. Все эти методы относятся к одному из двух классов:

(а) Методы, которые всегда приводят к нахождению оптимального решения, но требуют для этого в наихудшем случае недопустимо большого числа операций.

(б) Методы, которые не всегда приводят к нахождению оптимального решения, но требуют для получения такого решения приемлемого числа операций.

В основу многих известных методов решения задачи коммивояжера положены результаты, полученные в теории линейного целочисленного и динамического программирования. Описание этих методов увело бы нас слишком далеко от целей, которые автор ставил перед собой при изложении этого раздела. В этом разделе мы ограничимся рассмотрением двух различных методов решения задачи коммивояжера, опирающихся только на результаты, которые получены в предыдущих главах этой книги. Первый из этих методов — метод ветвей и границ — относится к классу (а), второй — метод последовательного улучшения решения — к классу (б).

Метод ветвей и границ

Пусть рассматривается граф $G = (X, A)$. В разд. 7.4 показано, что для получения нижней границы L_1 длины гамильтонова контура наименьшей длины на графе G может быть использован описанный в разд. 4.3 алгоритм построения потока минимальной стоимости. Если полученный с помощью этого алгоритма оптимальный поток соответствует некоторому одному контуру на графе G, то этот контур является оптимальным гамильтоновым контуром, и на этом решение задачи коммивояжера заканчивается. Однако существует достаточно высокая вероятность того, что оптимальный поток, полученный для любого практически встречающегося графа, будет соответствовать нескольким несвязным контурам. В этом случае произвольно выберем один контур и обозначим через $X_c = \{x_1, x_2, \ldots, x_k\}$ множество входящих в него вершин.

В оптимальном решении коммивояжер, выходя из вершины $x_1 \in X_c$, перемещается либо в вершину, принадлежащую множеству X_c, либо в вершину, не входящую в множество X_c. Если он пришел в вершину $x \in X_c$, то из нее он снова перемещается либо в вершину из множества X_c, либо в вершину, не принадлежащую X_c, и т. д. Таким образом, оптимальное решение должно быть представлено по крайней мере в одном из следующих графов:

1. Графе G, из которого исключены все дуги (x_1, y), $y \in X_c$.
2. Графе G, из которого исключены все дуги (x_1, y), $y \notin X_c$ и дуги (x_2, z), $z \in X_c$.

3. Графе G, из которого исключены все дуги (x_1, y), (x_2, y), $y \notin X_c$ и дуги $(x_3\ z)$, $z \in X_c$.

. . . . . .

. . . . . .

k. Графе G, из которого исключены все дуги (x_1, y), (x_2, y), ..., (x_{k-1}, y), $y \notin X_c$ и дуги (x_k, z), $z \in X_c$.

(Заметим, что исключение дуги из графа эквивалентно приписыванию ей бесконечной длины.)

Назовем перечисленные выше графы соответственно G_1, G_2, ..., G_k. Поскольку длина каждой дуги в графе G_i, $i = 1, 2, \ldots, k$ не меньше длины соответствующей дуги в графе G, длина оптимального гамильтонова контура в графе G_i не может быть меньше длины оптимального гамильтонова контура в графе G. Более того, оптимальное решение на графе G является также оптимальным решением по крайней мере на одном из графов G_1, G_2, ..., G_k. Определим далее с помощью алгоритма нахождения потока минимальной стоимости нижнюю границу L_1 длины оптимального гамильтонова контура для каждого из графов $G_1, G_2, \ldots, G_k$. Назовем эти нижние границы соответственно $L_1(G_1)$, $L_1(G_2)$, ..., $L_1(G_k)$. (Читатель, знакомый с методами решения задач линейного программирования, может вместо решения задачи потока минимальной стоимости решать задачу о назначениях.)

Если полученный для некоторого графа G_c оптимальный поток протекает по дугам, образующим гамильтонов контур, то мы не должны в дальнейшем рассматривать никакой граф G_j, для которого $L_1(G_j) \geqslant L_1(G_c)$, так как $L_1(G_c)$ является достижимой нижней границей. Таким путем некоторые из графов G_1, G_2, ..., G_k исключаются из дальнейшего рассмотрения.

Для каждого из графов G_i, для которых нижняя граница имеет меньшее значение, чем $L(G_c)$, повторяется описанная выше процедура. При этом граф G_i заменяется графами G_{i1}, G_{i2}, ... На каждом из графов G_{ij} вычисляется нижняя граница $L_1(G_{ij})$ длины оптимального гамильтонова контура. Затем, как и ранее, исключается из дальнейшего рассмотрения как можно большее количество вновь образованных графов.

В конце концов один граф, в котором найден гамильтонов контур минимальной длины, исключит из дальнейшего рассмотрения все оставшиеся графы. Этот гамильтонов контур должен быть гамильтоновым контуром минимальной длины в исходном графе G.

Рассмотренный метод решения задачи коммивояжера называется методом ветвей и границ.

ПРИМЕР. Воспользуемся методом ветвей и границ для нахождения оптимального гамильтонова контура в графе, изображенном на рис. 7.4. Так как граф G имеет малую размерность, мы сразу можем обнаружить, что в ре-

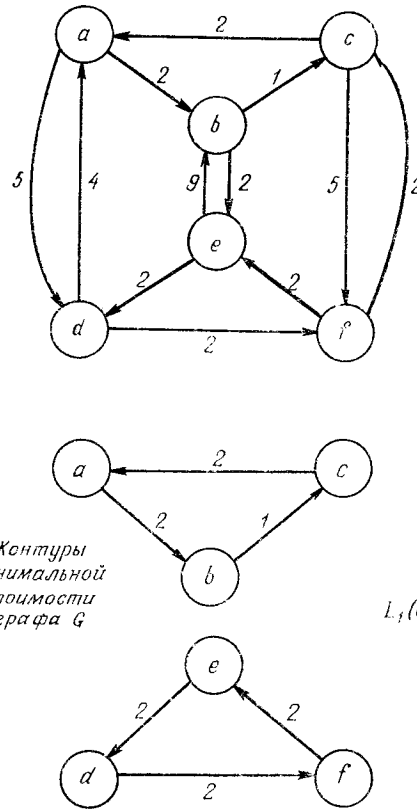


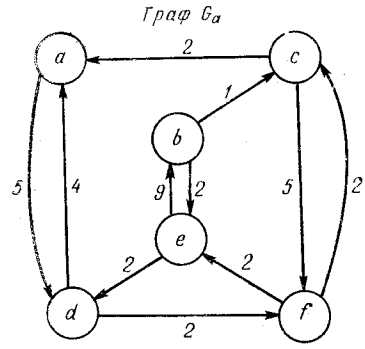
Рис. 7.4. Пример использования метода ветвей и границ для нахождения оптимального гамильтонова контура.

зультате использования алгоритма нахождения потока минимальной стоимости на графе G (подробнее см. разд. 7.3) получается поток, соответствующий двум контурам (a, b), (b,c), (c, a) и (f, e), (e, d), (d, f). Общая длина этих двух контуров равна 11. Таким образом, $L_1(G) = 11$. Используя вершины $X_c = \{a, b, c\}$ первого контура, мы можем построить три графа G_a, G_b, G_c, изображенные на рис. 7.4 (продолжение). Граф G_a получен путем исключения из графа G дуги, направленной из a в b или c, т. е. дуги (a, b). Граф G_b получен путем исключения из графа G всех дуг, направленных из a в d, e, f, и всех дуг, направленных из b в a и c. Таким образом, граф G_b получен путем исключения из графа G дуг (a, d) и (b, c). Граф G_c получается путем исключения из графа G всех дуг, направленных из a и b в d, e, f, и всех дуг, направленных из c в a и b. Таким образом, граф G_c получен путем исключения графа G дуг (a, d), (b, c) и (c, a).

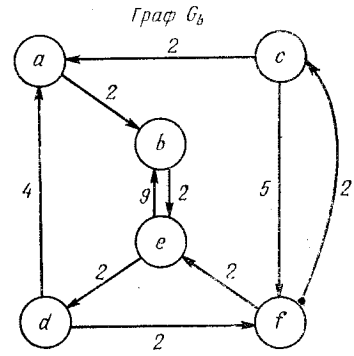
Вследствие небольшой размерности графов мы можем сразу увидеть, что использование алгоритма нахождения потока минимальной стоимости привело бы к следующим контурам и нижним границам длины оптимальных гамильтоновых контуров в графах G_a, G_b, G_c.

Граф	Контур	Нижняя граница
G_a	(a, d), (d, f), (f, e), (e, b), (b, c), (c, a)	$L_1(G_a)=21$
G_b	(a, b), (b, e), (e, d), (d, f), (f, e), (c, a)	$L_1(G_b)=12$
G_c	(a, b), (b, c) (c, f), $(f, e,)$ (e, d), (d, a)	$L_1(G_c)=16$

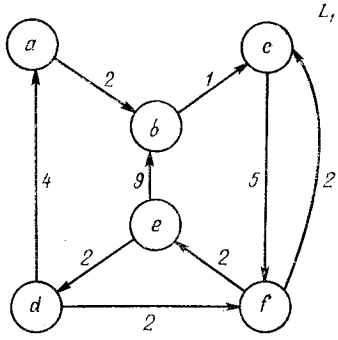
Так как при определении нижней границы длины оптимального гамильтонова контура в графе G_b получен гамильтонов контур длины 12, то графы G_a и G_c могут в дальнейшем не рассматриваться (действительно, $L_1(G_b) < L^1(G_c) < L_1(G_a)$). Следовательно, гамильтонов контур (a, b), (b, e), (e, d), (d, f), (f, c), (c, a) является гамильтоновым контуром в исходном графе G.



Контур минимальной стоимости
a, d, f, e, b, c, a
$L_1(G_a) = 21$



Контур минимальной стоимости
a, b, e, d, f, c, a
$L_1(G_b) = 12$



Граф G_c

Контур минимальной стоимости
a, b, c, f, e, d, a
$L_1(G_c) = 16.$

Рис. 7.4. (продолжение)

Метод последовательного улучшения решения

Метод последовательного улучшения решения задачи коммивояжера используется для нахождения в графе G близкого к оптимальному (и иногда оптимального) гамильтонова контура и состоит в следующем. Для начала берется некоторый произвольный гамильтонов контур. Пусть $x_1, x_2, \ldots, x_n$ обозначает последовательность, в которой в этом контуре обходятся вершины графа G.

Для $i = 1, 2, \ldots, n-1$ и $j = i+1, \ldots, n$ определить, уменьшается ли длина гамильтонова контура при перестановке в заданной выше последовательности обхода вершин пары вершин x_i и x_j. Если это приводит к уменьшению длины гамильтонова контура, то внести такую перестановку в порядок обхода вершин графа G. Повторять этот процесс до тех пор, пока с помощью попарных перестановок достигается уменьшение длины гамильтонова контура.

Перестановка вершин i и j в действительности означает замену дуг

$(x_{i-1}, x_i), (x_i, x_{i+1}), (x_{j-1}, x_j) (x_j, x_{j+1})$

дугами

$(x_{i-1}, x_j), (x_j, x_{i+1}), (x_{j-1}, x_i), (x_i, x_{j+1})$.

Процесс попарных перестановок должен быть конечным, так как (а) имеется только конечное число различных способов обхода n вершин графа (различных перестановок) и (б) каждый раз формируется новая последовательность обхода, отличная от всех предыдущих. При этом получается новый гамильтонов контур, имеющий строго меньшую длину, чем предыдущий.

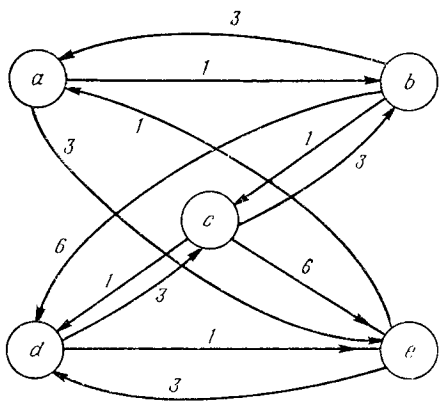


Рис. 7.5. Пример использования метода последовательного улучшения решения.

Следует заметить, что при использовании метода последовательного улучшения решения результирующий гамильтонов контур зависит от того, каким был выбран начальный гамильтонов контур. Рассмотрим для примера граф, изображенный на рис. 7.5. Этот граф содержит пять вершин, следовательно, каждый гамильтонов контур также должен включать пять

дуг. Так как каждая дуга имеет длину, не меньшую чем единица, каждый гамильтонов контур должен иметь длину не менее пяти единиц. Гамильтонов контур, в котором вершины обходятся в порядке a, b, c, d, e, имеет длину, равную пяти единицам, и поэтому он должен быть оптимальным гамильтоновым контуром.

Предположим, что в методе последовательного улучшения решения в качестве начального выбран гамильтонов контур, в котором порядок обхода вершин соответствует a, e, d, c, b. Читатель может убедиться, что в этом случае никакая новая последовательность обхода вершин, получаемая перестановкой любой пары вершин, не соответствует гамильтонову контуру в графе G (например, перестановка вершин a и e приводит к тому, что последовательности e, a, d, c, b нельзя поставить в соответствие контур, так как в графе отсутствует дуга из a в d). Таким образом, если последовательное улучшение решения начинается с гамильтонова контура, в котором вершины обходятся в порядке a, e, d, c, b, то в результате получаем тот же самый гамильтонов контур с длиной, равной $3 + 3 + 3 + 3 + 3 = 15$. Очевидно, что этот контур не является оптимальным гамильтоновым контуром.

Теперь предположим, что в качестве начальной последовательности обхода вершин графа G выбрана последовательность a, b, d, c, e. Соответствующий ей гамильтонов контур имеет длину $1 + 6 + 3 + 6 + 1 = 17$. Если произвести перестановку вершин c и d, то получаемая в результате последовательность a, b, c, d, e соответствует оптимальному гамильтонову контуру с длиной, равной $1 + 1 + 1 + 1 + 1 = 5$.

Как видно из рассмотренного выше примера, длина начального гамильтонова контура не обязательно служит показателем качества результирующего гамильтонова контура. Действительно, в первом случае в качестве начального был выбран гамильтонов контур длины 15 и получился в результате гамильтонов контур с длиной 15. В то же время при выборе в качестве начального гамильтонова контура с длиной 17 в результате получился контур длины 5.

В этом примере просто было установить, оптимален ли получаемый контур. Однако в общем случае мы не можем быть уверены, что гамильтонов контур, получаемый с помощью метода последовательного улучшения решения, является оптимальным.

В этом случае для определения максимального значения разности между длиной получаемого и оптимального гамильтонова контуров могут быть использованы значения нижних границ L_1 и L_2.

Усовершенствование метода последовательного улучшения решения можно найти в работах Стекхана [9], а также Лина и Кернигана [10].

УПРАЖНЕНИЯ

1. Покажите, что множество оптимальных решений задачи коммивояжера не зависит от того, какая из вершин выбрана в качестве начальной.

2. Предположим, что коммивояжер должен посетить вершину b сразу же после посещения вершины a. Как это может быть учтено при решении задачи коммивояжера?

3. Предположим, что коммивояжер должен посетить вершину b после посещения вершины a (но не обязательно сразу же). Как это может быть учтено при решении задачи коммивояжера?

4. Используйте результаты, полученные в разд. 7.3, для того, чтобы удостовериться в наличии гамильтоновых контуров в графах, изображенных на рис. 7.6 и 7.7.

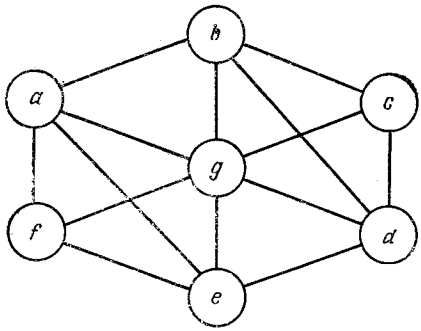


Рис. 7.6.

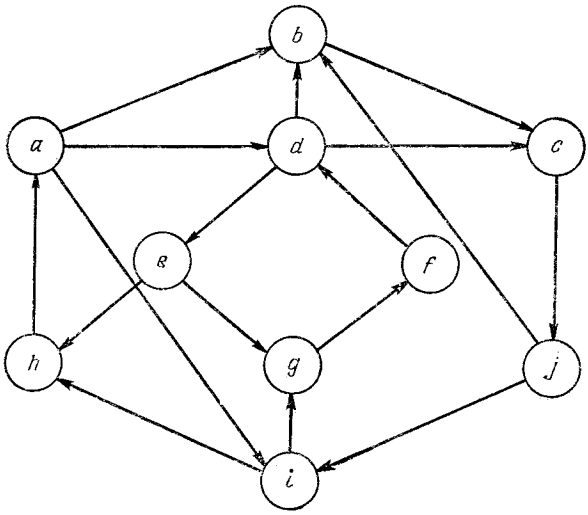


Рис. 7.7.

5. Мы нашли, что для графа, изображенного на рис. 7.3, $L_1 > L_2$. Сконструируйте граф, для которого $L_1 = L_2$.

6. Найдите методом ветвей и границ оптимальное решение задачи коммивояжера на графе рис. 7.8.

7. Найдите методом последовательного улучшения решения достаточно хороший маршрут коммивояжера на графе рис. 7.8. В качестве на-

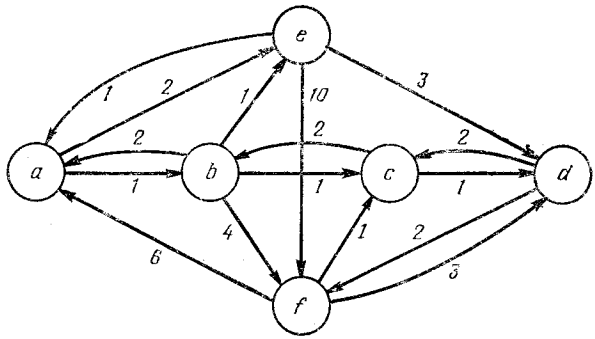


Рис. 7.8.

чального выберите гамильтонов контур, в котором вершины обходятся в последовательности d, c, b, e, a, f.

8. Предположим, что два коммивояжера при использовании метода последовательного улучшения решения выбирают в качестве начального один и тот же неоптимальный гамильтонов контур. Последовательности попарных перестановок вершин, используемые каждым из коммивояжеров, отличаются друг от друга. Покажите, что решением задачи в этом случае могут быть различные гамильтоновы контуры.

9. Прежде каждый сотрудник отдела исследования операций большого предприятия получал копию документа, касающегося всех сотрудников этого отдела. В целях экономии начальник отдела решил, что следует изготовлять только одну копию такого документа, которая должна передаваться внутри отдела по заранее определенному маршруту. Расстояния между рабочими столами сотрудников отдела даны в приведенной ниже таблице. Каким следует выбрать маршрут движения документа? (Документ каждый раз должен возвращаться к сотруднику, который первым его передал для дальнейшего рассмотрения. Это необходимо для того, чтобы отправитель документа мог быть уверен, что документ не утерян в процессе движения по маршруту.)

От исполнителя \ К исполнителю	1	2	3	4	5	6
1	0	9	8	7	6	10
2		0	10	9	15	20
3			0	5	15	20
4				0	20	5
5					0	20

(Расстояние указано в метрах.)

10. Для изготовления изделия в механической мастерской необходимо выполнить четыре операции на машине. Эти операции могут быть произведены в любой последовательности. Однако время переналадки машины зависит от того, какой была предыдущая операция по изготовлению изделия. Длительности переналадок приведены в таблице. Какова наилучшая последовательность выполнения операций?

Предыдущая операция \ Последующая операция	А	Б	В	Г
А	x	15	20	5
Б	30	x	30	15
В	25	25	x	15
Г	20	35	10	x

ЛИТЕРАТУРА

1. Bellmore M., Nemhauser G. L. The Travelling Salesman Problem: A Survey, *ORSA*, vol. 16, pp. 538 — 558, 1968.
2. Berge C., Graphs and Hupergraphs (English translation by E. Minieka), North-Holland, Chap. 10, pp. 186 — 227, 1973.
3. Christofides N., Bounds for the Travelling-Salesman Problem, *ORSA*, vol. 20, No. 6, pp. 1044 — 1056, 1972.
4. Chvatal V., On Hamilton Ideals, *J. Combinatorial Theory, Series B*, vol. 12, No. 2, pp. 163 — 168, 1972.
5. Garfinkel R., Namhauser G. L., Integer Programming, John Wiley, Inc., New York, pp. 354 — 360, 1972.
6. Ghouila-Houri A., Une condition suffisante d'existense d'un circuit hamiltonien, *Cahiers du Research Acad. Sci.*, vol. 251, pp. 494 — 497, 1960.
7. Held M., Karp R., The Travelling-Salesman Problem and Minimum Spanning Trees, *ORSA*, vol. 18, pp. 1139 — 1162, 1970.
8. Held M., Karp R., The Travelling-Salesman Problem and Minimum Spanning Trees, Part II, *Math. Programming*, vol. 1, No. 1, pp. 6 — 25, 1971.
9. Steckhan H. A Theorem on Symmetric Travelling Salesman Problems, *ORSA*, vol. 18, pp. 1163 — 1167, 1970.
10. Lin S., Kernighan B. W. An Effective Heuristic Algorithm for the Travelling-Sales man Problem, *ORSA*, vol. 21, No. 2, pp. 498 — 516, 1973.

ГЛАВА 8

Задачи размещения

8.1. Введение

Задачи размещения связаны с решением проблем наилучшего расположения в определенных регионах таких систем обслуживания, как торговые центры, посты пожарной охраны, фабрики, аэропорты, склады и т. д. Математическая структура задачи размещения определяется конфигурацией области допустимых точек и способом оценки качества размещения. Вследствие этого существует много разнообразных задач размещения, и техническая литература изобилует методами их решения. В этом разделе мы ограничимся рассмотрением только таких задач размещения, для которых областью допустимых точек размещения центров обслуживания является некоторый граф, т. е. эти центры могут располагаться в какой-либо вершине или на какой-либо дуге графа. Кроме того, мы будем ограничиваться только такими задачами размещения, которые являются не очень сложными с математической точки зрения. Рассмотрим несколько примеров, встречающихся на практике задач размещения.

ПРИМЕР 1. Администрация округа решила построить новый пост пожарной охраны, который должен обслуживать все шесть городов округа. Этот пост должен располагаться возле какой-либо из автомагистралей, но так, чтобы минимизировать расстояние до наиболее отдаленного от нее города. Если автомагистраль округа изображается ребром графа, то задача размещения пожарного поста становится задачей такого размещения точки на ребре графа, при котором минимизируется расстояние вдоль ребер (автомагистралей) от этой точки до наиболее удаленной вершины графа (города).

Рассмотрим граф, представленный на рис. 8.1. Если в качестве точки размещения выбрать вершину *3*, то наиболее отдаленной от вершины *3* является вершина *6*. Расстояние от вершины *3* до вершины *6* равно 3 единицам.

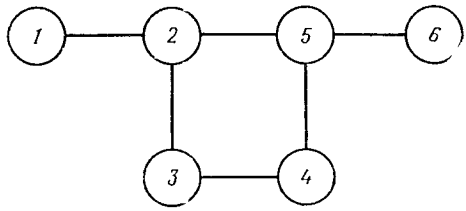


Рис. 8.1. (Все дуги имеют длину, равную единице.)

Более удачным является выбор вершины *2*, которая отдалена от любой вершины не более чем на 2 единицы. Еще более удачным является выбор в качестве точки размещения средней точки ребра (2, 5). Эта точка отдалена от вершин *1*, *3*, *4*, *6*, на 1,5 единицы.

ПРИМЕР 2. Предположим, что в том же округе должна быть размещена почта таким образом, чтобы минимизировалось суммарное расстояние от нее до каждого из городов округа. Размещение почты в средней точке ребра (2,5) дало бы суммарное расстояние, равное $1{,}5 + 0{,}5 + 1{,}5 + 1{,}5 + 0{,}5 + 1{,}5 = 7$ единицам, так как расстояние от средней точки ребра (2,5) до вершин *1*, *2*, *3*, *4*, *5*, *6* соответственно равно 1,5; 0,5; 1,5; 1,5; 0,5; 1,5 единицы.

В примерах 1 и 2 по существу рассмотрены одинаковые задачи. Они отличаются только критерием оценки качества размещения. В примере 1 минимизируется максимальное расстояние; в примере 2 минимизируется сумма расстояний. Точка размещения, выбранная в соответствии с первым критерием, т. е. точка, в которой минимизируется максимальное расстояние до всех вершин графа, называется центром. Точка же, выбранная в соответствии со вторым критерием, т. е. точка, в которой минимизируется сумма расстояний до всех вершин графа, называется медианой. Более строгие определения этих понятий будут даны позднее.

ПРИМЕР 3. Предположим, что в том же округе необходимо разместить станцию автомобилей-тягачей для оказания помощи водителям, попавшим в аварию на какой-либо из автомагистралей округа. Предположим, также, что критерием качества размещения этой станции является минимум максимального расстояния, которое автомобиль-тягач должен преодолеть до возможного места аварии. В этом случае вместо максимального расстояния до всех вершин графа (как это делается в примере 1) должно рассматриваться максимальное суммарное расстояние до всех точек всех ребер графа.

ПРИМЕР 4. Предположим, что в том же округе для размещения телефонной станции должно быть выбрано место в каком-либо из графов или вдоль какой-либо автомагистрали. Место размещения телефонной станции должно быть выбрано так, чтобы минимизировалась суммарная протяженность всех телефонных линий, которые будут проложены между станцией и шестью городами. Усложним задачу, считая, что города имеют различную численность населения и количество требуемых линий между станцией и каждым из городов находится в пределах от 1 до 5.

Заметим, что если для соединения каждого города с телефонной станцией требуется только одна линия, то поставленная в примере 4 задача имеет математическую структуру, аналогичную структуре задачи, поставленной в примере 2. Однако теперь следует точнее учесть численность населения определенных городов и влияние на выбор места размещения станции более густонаселенных городов.

Прежде чем ввести более строгие определения подлежащих рассмотрению различных типов размещений, следует ввести некоторые определения, необходимые для описания точек на дугах и различных расстояний в графе.

Множество рассматриваемых вершин в графе G содержит вершины с номерами от 1 до n. Рассмотрим произвольную дугу

(i, j), длина которой равна $a(i, j) > 0$. Пусть f обозначает точку на дуге (i,j), которая для всех $0 \leqslant f \leqslant 1$ отстоит на $fa(i, j)$ единиц от вершины i и на $(1 - f)a(i, j)$ единиц от вершины j. Назовем ее *f-точкой*. Таким образом, четверть-точкой дуги (i, j) является точка, отстоящая от вершины i на $^1/_4$ длины дуги (i, j)

Нуль-точка дуги (i, j) является вершиной i, а единичная точка дуги (i, j) — вершиной j. Следовательно, вершины графа также могут рассматриваться как точки дуг. Точки дуг, которые не являются вершинами, называются *внутренними точками*. Любая точка дуги должна быть либо внутренней точкой, либо вершиной. Как и ранее, обозначим через X множество всех вершин графа. Пусть через P обозначается множество всех точек. Таким образом, $P - X$ является множеством всех внутренних точек.

Пусть $d(i, j)$ обозначает длину кратчайшего пути из вершины i в вершину j. Тогда через D обозначается матрица $n \times n$, в которой элементом (i, j) является $d(i, j)$. Элементы в матрице D называются *расстояниями вершина — вершина*. Напомним, что для вычисления элементов матрицы D может быть использован любой из двух рассмотренных в разд. 3.2 алгоритмов: алгоритм Флойда или алгоритм Данцига. Пусть через $d(f - (r, s)j)$ обозначена длина кратчайшего пути от f-точки на дуге (r, s) до вершины j. Эта величина называется *расстоянием точка — вершина*. Если дуга (r, s) неориентированная, т. е. допустим ее обход в обоих направлениях, то в качестве $d(f - (r, s),j)$ должно быть выбрано наименьшее из следующих двух расстояний:

(а) расстояние от точки до вершины r плюс расстояние от вершины r до вершины j,

(б) расстояние от f-точки до вершины s плюс расстояние от вершины s до вершины j.

Таким образом,

$$d(f - (r, s), j) = \min \{fa(r, s) + d(r, j),\ (1 - f)\, a(r, s) + d(s, j)\}. \tag{1а}$$

Если дуга (r, s) ориентированная, т. е. ее обход допустим только из r в s, то первый член в (1а) может быть исключен и

$$d(f - (r, s), j) = (1 - f)\, a(r, s) + d(s, j). \tag{1б}$$

Заметим, что для вычисления всех расстояний точка — вершина необходимо знать только значения длин всех дуг и значения всех элементов матрицы D.

Для заданной дуги (r, s) и вершины j расстояние точка — вершина как функция от f на графике должна иметь один из трех типов зависимостей, показанных на рис. 8.2. Заметим, что угол наклона этой кусочно-линейной кривой равен $+a(r, s)$ или

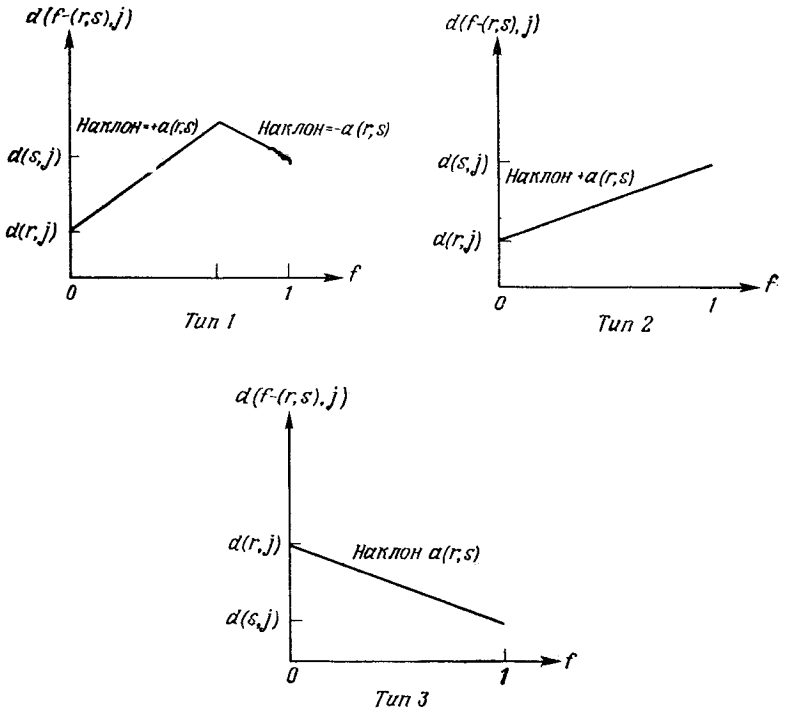


Рис. 8.2. Графики функций, характеризующих расстояния точка — вершина.

— $a(r, s)$ и его величина может не более одного раза измениться от $+a(r, s)$ до $-a(r, s)$.

Рассмотрим далее наименьшее расстояние от вершины j до каждой точки на дуге (r, s). Для некоторой точки на дуге (r, s) это расстояние принимает максимальное значение. Оно обозначается через $d'(j, (r, s)$ и называется *расстоянием вершина — дуга*. Если дуга (r, s) неориентированная, то имеются два маршрута движения из вершины j в f-точку на дуге (r, s): через вершину r или вершину s. Естественно, мы выбираем кратчайший из этих двух маршрутов. Если эти два маршрута из вершины j в f-точку на дуге (r, s) имеют различную протяженность, то некоторые точки, соседние с f-точкой на дуге (r, s), отстоят еще дальше от вершины j. Например, на рис. 8.1 четверть-точка на дуге (3, 4) отстоит от вершины *6* на 1,25 единицы или на 2,75 единицы в зависимости от того, движетесь ли вы через вершину *3* или через вершину *4*. Если f возрастает с 0,25 до 0,26, то наименьшее расстояние от

вершины *2* до значения 0,26 на дуге (3, 4) равно $\min\{1{,}26,\ 2{,}74\} = 1{,}26$.

Поэтому два расстояния от вершины j до некоторой точки на дуге равны между собой, если эта точка является наиболее удаленной от вершины j.

Заметим, что сумма этих расстояний всегда равна

$$d(j, r) + fa(r, s) + d(j, s) + (1 - f)\, a(r, s) = d(j, r) + d(j, s) + a(r, s).$$

Откуда следует, что

$$d'(j, (r, s)) = \frac{d(j, r) + d(j, s) + a(r, s)}{2}. \qquad (2а)$$

С другой стороны, если дуга (r, s) ориентированная, то некоторая точка на дуге (r, s) может быть достигнута только через вершину r. Следовательно, наиболее удаленной от любой вершины графа точкой на дуге (r, s) является точка, ближайшая к вершине s (т. е. f-точка на дуге (r, s), для которой $f = 1$). В этом случае

$$d'(j, (r, s)) = d(j, r) + a(r, s). \qquad (2б)$$

Пусть в графе имеется m дуг. Обозначим через D' матрицу размерности $n \times m$, у которой элемент, стоящий на пересечении i-й строки и k-столбца, является расстоянием вершина — дуга от j-й вершины до k-й дуги. Заметим, что значения элементов матрицы D' могут быть вычислены с помощью равенств (2а) и (2б), если известны расстояния вершина — вершина, задаваемые матрицей D, и длины дуг графа.

Пусть $d'(f - (r, s), (t, u))$ обозначает максимальное расстояние от f-точки на дуге (r, s) до точек на дуге (t, u). Это расстояние называется *расстоянием точка — дуга.*

Если дуга (r, s) неориентированная и если $(r, s) \neq (t, u)$, то маршрут от f-точки на дуге (r, s) до наиболее отдаленной точки на дуге (t, u) должен проходить либо через вершину r, либо через вершину s. Отсюда следует, что

$$d'(f - (r, s), (t, u)) = \min\{fa(r, s) + d'(r, (t, u)),\ (1 - f)\, a(r, s) + d'(s, t, u))\}. \qquad (3а)$$

Если же дуга (r, s) ориентированная и $(r, s) \neq (t, u)$, то первый член в формуле (8.3а) может быть исключен и

$$d'(f - (r, s), (t, u)) = (1 - f)\, a(r, s) + d'(s, (t, u)). \qquad (3б)$$

Если $(r, s) = (t, u)$ и дуга (r, s) ориентированная, то наиболее удаленная точка на дуге (r, s) от f-точки на (r, s) является g-точ-

кой, где g стремится к f со стороны значений, меньших чем f. Таким образом, в этом случае

$$d'(f-(r, s), (r, s)) = 1 - fa(r, s) + d(s, r). \qquad (3в)$$

Если же $(r, s) = (t, u)$ и дуга (r, s) неориентированная, то максимальное расстояние от f-точки на дуге (r, s) до g-точки на дуге (r, s) в случае, когда $g < f$, не может превышать

$$A \equiv \min\{fa(r, s), {}^1/_2[a(r, s) + d(s, r)]\}.$$

Первый член в этом выражении равен длине маршрута от f-точки до g-точки в пределах дуги (r, s), второй член равен длине маршрута от f-точки на дуге (r, s) до g-точки на дуге (r, s), но проходящего через вершину s.

Аналогично в случае, когда $g > f$, максимальное расстояние от f-точки на дуге (r, s) до g-точки (r, s) не может превышать (r, s):

$$B \equiv \min\{(1-f)a(r, s), {}^1/_2[a(r, s) + d(r, s)]\}.$$

Первый член в выражении для B равен длине маршрута от f-точки до g-точки в пределах дуги (r, s), а второй — равен длине маршрута от f-точки на дуге (r, s) до g-точки на дуге (r, s), проходящего через вершину r.

Следовательно, если дуга (r, s) неориентированная, то $d'(f-$

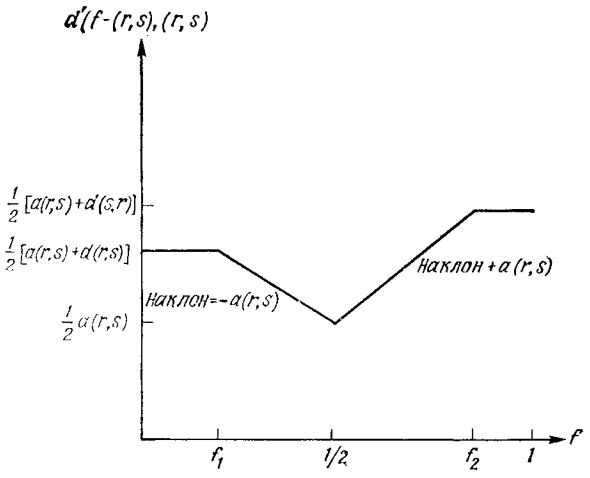


Рис. 8.3. График функции, характеризующей расстояния тока — дуга $[d'(f-(r, s), (r, s)]$.

$-(r, s), (r, s)) = \max\{A, B\}$, или, что эквивалентно,

$$d'(f-(r, s), (r, s)) = \max\{\min\{fa(r, s), {}^1/_2[a(r, s) + d(s, r)]\}, \min\{(1-f)a(r, s), {}^1/_2[a(r, s) + d(r, s)]\}\}. \quad (3г)$$

Если изобразить расстояние $d'(f-(r, s), (t, u))$ точка — дуга как функцию f для всех $(r, s) \neq (t, u)$, то соответствующая кривая на графике будет иметь тот же вид, что и кривая расстояний точка — вершина, представленная на рис. 8.2, так как уравнения (3а) и (3б) имеют соответственно тот же вид, что и уравнения (1а) и (1б). Отличаются они только на постоянные величины. С другой стороны, если $d'(f-(r, s), (r, s))$ для любой неориентированной дуги (r, s) представить как функцию от f, то кривая функции будет иметь вид, показанный на рис. 8.3. Это следует из уравнения (3г).

Таким образом, введенные нами определения могут быть систематизированы в форме следующей таблицы:

Обозначение	Наименование	Способ определения
$a(i, j)$	Длина дуги	Задана
$d(i, j)$	Расстояние вершина-вершина (ВВ)	Алгоритм Флойда или Данцига
$d'(f-(r, s), j)$	Расстояние точка-вершина (ТВ)	Уравнения (1а), (1б)
$d'(j, (r, s))$	Расстояние вершина-дуга (ВД)	Уравнения (2а), (2б)
$d'(f-(r, s), (t, u))$	Расстояние точка-дуга (ТД)	Уравнения (3а), (3б), (3в), (3г)

Пусть $\text{МВВ}(i) = \max_j \{d(i, j)\}$ (4)

— максимальное расстояние от вершины i до вершин графа, т. е. расстояние от вершины i до наиболее отдаленной вершины графа, и

$$\text{СВВ}(i) = \sum_j d(i, j) \quad (5)$$

— суммарное расстояние от вершины i до всех вершин графа.

Пусть аналогично $\text{МТВ}(f-(r, s)) = \max_j \{d(f-(r, s), j\}$ (6)

— максимальное расстояние от f-точки на дуге (r, s) до вершин графа, т. е. расстояние от f-точки на дуге (r, s) до наиболее отдаленной вершины графа, и

$$\text{СТВ}(f-(r, s)) = \sum_j d(f-(r, s), f) \quad (7)$$

— сумма расстояний от f-точки на дуге (r, s) до всех вершин графа. Аналогично мы можем определить МВД(i), СВД(i), МТД(f—(r, s)), СТД(f—(r, s)), взяв максимум или сумму по всем дугам, а не по всем вершинам, как это сделано в выражениях (4) — (7).

Введя определения всех этих расстояний, их максимумов и сумм, мы теперь готовы к тому, чтобы дать строгие определения для рассматриваемых далее различных типов размещений.

1. *Центром* графа G являются любая вершина x этого графа, такая, что

$$\text{МВВ}(x) = \min_i \{\text{МВВ}(i)\}. \tag{8}$$

Таким образом центр — это любая вершина, расстояние от которой до наиболее отдаленной от нее вершины минимально.

2. *Главным центром* графа G является любая вершина x этого графа, такая, что

$$\text{МВД}(x) = \min_i \{\text{МВД}(i)\}, \tag{9}$$

т. е. главный центр — это любая вершина, расстояние от которой до наиболее удаленной точки на дугах графа минимально.

3. *Абсолютным центром* графа G является любая f-точка на произвольной дуге (r, s) этого графа, такая, что

$$\text{МТВ}(f-(r, s)) = \min_{f-(t, u)\in P} \{\text{МТВ}(f-(t, u))\}, \tag{10}$$

т. е. абсолютный центр — это любая точка на дуге, расстояние от которой до наиболее отдаленной вершины графа минимально.

4. *Главным абсолютным центром* графа G является f-точка на произвольной дуге (r, s) этого графа, такая, что

$$\text{МТД}(f-(r, s)) = \min_{f-(t, u)\in P} \{\text{МТД}(f-(t, u))\}. \tag{11}$$

Таким образом, главный абсолютный центр — это любая точка, расстояние от которой до наиболее удаленной точки минимально.

По аналогии с каждым из четырех типов определенных здесь размещений мы можем определить (5) *медиану*, (6) *главную медиану*, (7) *абсолютную медиану* и (8) *главную абсолютную медиану*.

Определения типов размещений (5) — (8) совершенно аналогичны определениям соответствующих предыдущих типов размещений, за исключением того, что везде оператор максимизации [т. е. МВВ(i), МВД(i), МТВ(f—(i, u)), МТД(f—(t, u))] заменяется оператором суммирования [т. е. СВВ(i), СВД(i), СТВ(f—(t, u)) СТД(f—(t, u))].

Заметим, что в примере 1 требуется найти абсолютный центр, в примере 2 — абсолютную медиану, в примере 3 — обобщенный абсолютный центр в примере 4 — взвешенную абсолютную медиану, которая вводится и обсуждается в разд. 8.4.

Ниже в разд. 8.2 излагаются методы нахождения четырех определенных выше типов центров. В разд. 8.3 излагаются методы нахождения четырех рассматривавшихся выше типов медиан.

8.2. Задачи поиска центра

В этом разделе рассматривается метод поиска каждого из описанных в разд. 8.1 четырех типов центров.

Центр

Вспомним, что центром является любая вершина x с наименьшим значением МВВ(x), т. е. центр — это любая вершина x, такая, что расстояние от нее до наиболее отдаленной вершины минимально. Центр отыскивается как один из элементов матрицы D, значение i, j-го элемента которой —$d(i, j)$ — определяет кратчайшее расстояние от вершины i к вершине j. Элементы матрицы могут быть вычислены с помощью алгоритма Флойда или алгоритма Данцига, рассмотренных в разд. 3.2. Максимальное расстояние МВВ(i) от вершины i до любой вершины графа является элементом i-й строки матрицы D, имеющим максимальное значение. Центром является произвольная вершина x с наименьшим среди всех вершин графа значением МВВ(x), т. е. центр — это произвольная вершина x, которой соответствует строка матрицы D, содержащая элемент с наименьшим максимальным значением.

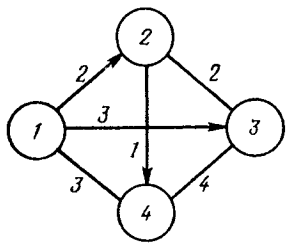


Рис. 8.4. Пример вычисления центра. В примере 1, где рассматривается изображенный на рисунке граф, устанавливается соответствие между порядковыми номерами дуг и их графовыми обозначениями: 1 — (1,2); 2 — (1,3); 3 — — (1,4); 4 — (2,4); 5 — (2,3); 6 — (3,4).

ПРИМЕР 1. Найдем центр для графа, изображенного на рис. 8.4. Мы оставляем читателю возможность проверки того, что с помощью алгоритма Флойда или алгоритма Данцига получается следующая матрица:

$$D = \begin{Vmatrix} 0 & 2 & 3 & 3 \\ 4 & 0 & 2 & 1 \\ 6 & 2 & 0 & 3 \\ 3 & 5 & 4 & 0 \end{Vmatrix}$$

Таким образом,

МВВ (1) $= \max\{0, 2, 3, 3\} = 3$,

МВВ (2) $= \max\{4, 0, 2, 1\} = 4$,

МВВ (3) $= \max\{6, 2, 0, 3\} = 6$,

МВВ (4) $= \max\{3, 5, 4, 0\} = 5$,

откуда min {(МВВ(i)} = min {3, 4, 5, 6} = 3 = МВВ(1). Следовательно, вершина *1* является центром графа, а расстояние от вершины *1* до наиболее отдаленной вершины графа равно 3 единицам.

Главный центр

Вспомним, что главный центр — это произвольная вершина x с минимальным среди всех вершин значением МВД(x), т. е. главным центром является такая вершина x, что расстояние от x до наиболее удаленной точки на дугах графа минимально. Для нахождения главного центра необходимо в матрице D' выбрать строку с наименьшим максимальным значением ее элемента. Эта строка соответствует вершине, являющейся главным центром, поскольку МВД(i) равен наибольшему элементу в i-й строке матрицы D' расстояний вершина — дуга.

ПРИМЕР 2. Найдем главный центр для графа, изображенного на рис. 8.4. Дуги этого графа пронумерованы от *1* до *6*. Используя полученную в предыдущем примере матрицу D расстояний вершина-вершина и длины дуг, приведенные на рис. 8.4, мы можем с помощью равенства (2) рассчитать элементы матрицы D':

$$D' = \begin{Vmatrix} 2 & 3 & 3 & 3 & 3\,{}^1\!/_2 & 5 \\ 6 & 7 & 4 & 1 & 2 & 3\,{}^1\!/_2 \\ 8 & 9 & 6 & 3 & 2 & 3\,{}^1\!/_2 \\ 5 & 6 & 3 & 6 & 5\,{}^1\!/_2 & 4 \end{Vmatrix}$$

Например, из равенства (2а) получаем, что

$d'(1, (3, 4)) = {}^1\!/_2 [d(1, 3) + d(1,4) + a(3, 4)] = {}^1\!/_2 (3 + 3 + 4) = 5$.

Из равенства (2б)

$d'(1, (2, 4)) = d(1, 2) + a(2, 4) = 2 + 1 = 3$.

Таким образом,

МВД (1) $= \max\{2, 3, 3, 3, 3\,{}^1\!/_2, 5\} = 5$,

МВД (2) $= \max\{6, 7, 4, 1, 2, 3\,{}^1\!/_2\} = 7$,

МВД (3) $= \max\{8,\ 9,\ 6,\ 3,\ 2,\ 3\ ^1/_2\} = 9,$

МВД (4) $= \max\{5,\ 6,\ 3,\ 6,\ 5\ ^1/_2,\ 4\} = 6.$

Отсюда $\min$ МВД$(i) = \min\{5,\ 7,\ 9,\ 6\} = 5 =$ МВД(1). Следовательно, вершина *1* является главным центром графа. Наиболее удаленная от вершины *1* точка лежит на дуге (3, 4) и расстояние от вершины *1* до этой точки равно 5 единицам.

Абсолютный центр

Вспомним, что абсолютный центр — это любая точка на дуге, расстояние от которой до наиболее отдаленной вершины графа минимально. Для поиска абсолютного центра мы должны найти такую точку $f - (r, s)$, что МТВ$(f - (r, s)) = \min\{\text{МТВ}(f - (t, u))\}$. Задача поиска абсолютного центра является более трудной, чем задачи поиска центра и главного центра графа, так как в процессе ее решения необходимо просматривать не только вершины, но и все точки дуг графа. Рассматривая свойства решения задачи, отметим, что ни одна из внутренних точек ориентированной дуги не может быть абсолютным центром. Действительно, из того, что обход ориентированной дуги может производиться только в одном направлении, следует, что конечная вершина, в которую входит эта дуга, ближе к каждой вершине графа, чем любая внутренняя точка этой дуги. Следовательно, при поиске абсолютного центра мы должны рассматривать только вершины графа и внутренние точки его неориентированных дуг. Рассмотрим произвольную неориентированную дугу (r, s). Расстояние $d(f - (r, s), j)$ от точки f на дуге (r, s) до вершины j определяется из уравнения (1а). Это расстояние легко представить как функцию от f. Ее график представляет собой кусочно-линейную кривую с не более чем двумя линейными отрезками (рис. 8.2). Построим графики $d(f - (r, s), j)$ для $0 \leqslant f \leqslant 1$ и для всех j. Участки кривых каждого из графиков, которые лежат выше соответствующих участков кривых всех других графиков, представляют собой значения $\max d(f - (r, s), j)$ для всех f. Точка f^* дуги (r, s), в которой достигается минимальное по f значение $\max d(f - (r, s), j)$, является наилучшим претендентом на размещение в ней абсолютного центра на дуге (r, s). С помощью этой процедуры может быть найдена точка размещения наилучшего претендента на каждой неориентированной дуге графа.

Абсолютным центром является любая точка $f^* - (r, s)^*$, для которой расстояние до максимально удаленной вершины графа минимально, т. е.

$$\max_j \{d(f^* - (r, s)^*, j)\} = \min_{(i, u)} \{\max_j \{d(f^* - (t, u), j)\}\}.$$

Таким образом, абсолютный центр находится в два этапа. Сначала находятся на каждой неориентированной дуге точки — претенденты на размещение в них абсолютного центра. Точками-претендентами являются точки на дуге, расстояние от которых до наиболее удаленных вершин минимально. Затем среди этих претендующих точек и всех вершин графа в качестве абсолютного центра выбирается такая точка или вершина, расстояние от которой до максимально удаленной вершины графа минимально. Для выбора наилучшей претендующей точки на каждом ребре требуется построение функций, характеризующих расстояния точка — вершина для всех f-точек. Это относительно не сложная процедура, так как функции являются кусочно-линейными кривыми, содержащими не более двух отрезков. К сожалению, по-видимому, не существует неграфических методов определения наилучших точек-претендентов, где можно было бы избежать сравнения на графиках расстояний точка — вершина для различных вершин графа. Рассмотренный метод отыскания абсолютного центра был предложен Хакими в 1964 г. [3].

ПРИМЕР 3. Найдем абсолютный центр в графе, изображенном на рис. 8.4. Мы знаем, что все абсолютные центры (их может быть несколько) должны быть либо вершинами, либо внутренними точками на неориентированных дугах. Среди вершин графа наилучшим претендентом на размещение абсолютного центра была бы вершина, являющаяся центром графа. В примере 1 мы нашли, что таким центром является вершина *1*, а все другие вершины графа отдалены от центра не более чем на 3 единицы. Таким образом, вершина *1* является наилучшим претендентом с максимальным расстоянием до всех вершин графа, равным 3 единицам.

Остается теперь исследовать внутренние точки трех неориентированных дуг (1,4), (2,3) и (3,4).

Исследуем сначала ребро (3,4). Из уравнения (1а) следует:

$$d(f-(3,4),1)=\min\{fa(3,4)+d(3,1),(1-f)a(3,4)+d(4,1)\}=$$

$$=\min\{4f+6,\ 4(1-f)+3\}=\begin{cases}4f+6 & \text{для } f\leqslant 1/8\\ 7-4f & \text{для } f\geqslant 1/8.\end{cases}$$

$$d(f-(3,4),2)=\min\{fa(3,4)+d(3,2),(1-f)a(3,4)+d(4,2)\}=$$

$$=\min\{4f+2,\ 4(1-f)+5\}=\begin{cases}4f+2 & \text{для } f\leqslant 7/8\\ 9-4f & \text{для } f\geqslant 7/8.\end{cases}$$

$$d(f-(3,4),3)=\min\{fa(3,4)+d(3,3),(1-f)a(3,4)+d(4,3)\}=$$

$$=\min\{4f,\ 4(1-f)+4\}=4f \text{ для всех } f\colon 0\leqslant f\leqslant 1.$$

$$d(f-(3,4),4)=\min\{fa(3,4)+d(3,4),(1-f)a(3,4)+d(4,4)\}=$$

$$=\min\{4f+3,\ 4(1-f)+0\}=\begin{cases}4f+3 & \text{для } f\leqslant 1/8\\ 4-4f & \text{для } f\geqslant 1/8.\end{cases}$$

Эти функции, характеризующие расстояния точка — вершина,

изображены на рис. 8.5. Наименьшее значение, принимаемое функцией $\max_j d(f-(r, s), j)$, достигается в точке, для которой $d(f-(3, 4), 1) = d(f-(3, 4)2)$.

Таким образом, $7-4f^* = 4f^* + 2$, $f^* = 5/8$,

$d(5/8-(3, 4), 1) = d(5/8-(3,4),2) = 4\,1/2$.

Следовательно, 5/8-точка является наилучшим претендентом для размещения абсолютного центра на дуге (3, 4), и не существует вершины, отстоящей от указанной точки на ребре (3, 4) более чем на $4\,1/2$ единицы.

На рис. 8.6 показан результат вычисления наилучшего претендента на роль абсолютного центра на ребре (1, 4). Заметим, что наилучшим претендентом в этом случае явля-

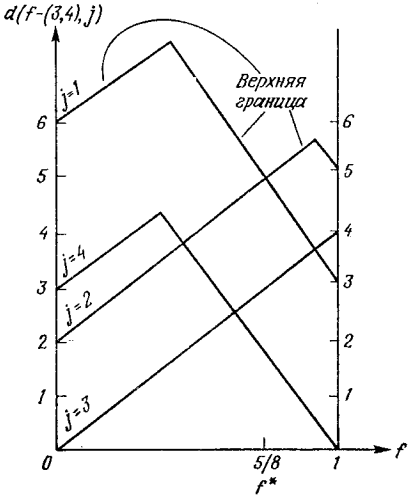


Рис. 8.5. График функций, характеризующих расстояния точка — вершина $d(f-(3, 4), j)$.

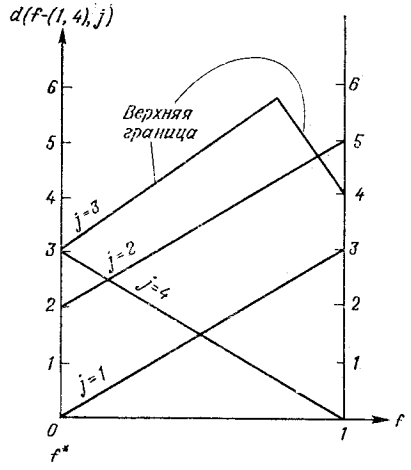


Рис. 8.6. График функций, характеризующих расстояния точка — вершина $d(f-(1, 4), j)$.

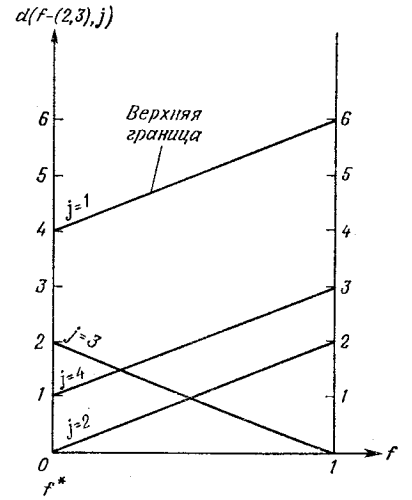


Рис. 8.7. График функций, характеризующих расстояния точка — вершина $d(f-(2,3), j)$.

ется 0-точка, которая совпадает с вершиной *1* графа. Мы уже знаем, что каждая вершина графа отдалена от вершины *1* не более чем на 3 единицы.

На рис. 8.7 показан результат вычисления наилучшего претендента для абсолютного центра на ребре (2, 3). Заметим, что наилучшим претендентом и в этом случае является 0-точка, которая совпадает с вершиной *2* графа. Следовательно, наилучшим претендентом среди внутренних точек является 5/8-точка ребра (3, 4) с максимальным отдалением от всех вершин графа, равным $4^1/_2$ единицам. Наилучшим претендентом среди вершин является вершина *1* с максимальным отдалением от всех других вершин графа, равным 3 единицам. Значит, вершина *1* является абсолютным центром графа.

Главный абсолютный центр

Напомним, что главным абсолютным центром является любая точка x, такая, что расстояние от нее до максимально отдаленной точки на дуге графа минимально.

Для нахождения главного абсолютного центра нам необходимо найти такую точку $f-(r, s)$, что

$$\text{МТД}(f-(r, s)) = \min_{f-(t, u)\in P} \{\text{МТД}(f-(t, u))\}.$$

Как и ранее, заметим, что *ни одна из внутренних точек на ориентированной дуге не может быть главным абсолютным центром.* Действительно, обход каждой ориентированной дуги может производиться только в одном направлении, поэтому конечная вершина ее находится ближе к любой дуге графа, и, следовательно, она является лучшим (по сравнению с любой внутренней точкой) претендентом на размещение главного абсолютного центра. Итак, при определении точки для размещения главного абсолютного центра мы должны рассматривать только вершины графа и внутренние точки его неориентированных дуг. Задача определения главного абсолютного центра идентична задаче определения абсолютного центра. Единственным отличием является лишь то, что в первой задаче рассматриваются на графе расстояния точка — вершина, а во второй — расстояния точка — дуга. Как отмечено в разд. 8.1, функции, характеризующие расстояния точка — дуга, за исключением функции $d'(f-(r, s), (r, s))$, имеют тот же вид, что и функции, характеризующие расстояния точка — вершина (см. рис. 8.2). Вид функции $d'(f-(r, s), (r, s))$ показан на рис. 8.3.

В большинстве практических задач точка, наиболее отдаленная от f-точки на ребре (r, s), не будет лежать на том же ребре (r, s). Если это так, то мы можем просто не рассматривать в дальнейшем функции расстояния точка — дуга $d'(f-(r, s), (r, s))$, и зада-

ча поиска главного абсолютного центра может быть решена с помощью описанного выше метода поиска абсолютного центра. Единственным отличием является то, что вместо функции расстояний точка — вершина при поиске главного абсолютного центра будут использоваться функции расстояний точка—дуга. Однако следует отметить, что для нахождения главного абсолютного центра требуется рассматривать большее, чем в предыдущей задаче, количество кривых ($d(f - (r, s), (t, u))$, так как число дуг в графе обычно превышает число его вершин. Но в тех случаях, когда точка, наиболее удаленная от f-точки на ребре (r, s), может лежать также на ребре (r, s), для определения наилучших точек претендентов на ребре (r, s) необходимо включать в рассмотрение еще и кривые функции $d'(f - (r, s), (r, s))$. Для построения этих кривых может быть использовано уравнение (3г). К счастью, эта функция также представляется в виде кусочно-линейной кривой, содержащей не более четырех линейных отрезков (см. рис. 8.3).

Таким образом, метод решения задачи поиска главного абсолютного центра аналогичен методу решения задачи поиска абсолютного центра с той лишь разницей, что в последнем случае вместо расстояний точка — вершина используются расстояния точка — дуга.

8.3. Задачи поиска медиан

Ниже рассматриваются методы поиска четырех типов медиан, описанных в разд. 8.1.

Медиана

Вспомним, что медиана — это вершина x, такая, что сумма расстояний от нее до всех остальных вершин графа минимальна, т. е.

$$\mathrm{CBB}(x) = \min\{\mathrm{CBB}(i)\}.$$

Заметим, что сумма значений элементов i-й строки матрицы D расстояний вершина — вершина равна сумме расстояний от вершины ко всем остальным вершинам графа, т. е. равна $CBB(i)$. Следовательно, медиана соответствует любой строке матрицы D, для которой сумма значений элементов минимальна.

ПРИМЕР 1. Найти медиану графа, представленного на рис. 8.4. Мы знаем из предыдущих примеров, что матрица D для этого графа имеет вид

$$D = \begin{Vmatrix} 0 & 2 & 3 & 3 \\ 4 & 0 & 2 & 1 \\ 6 & 2 & 0 & 3 \\ 3 & 5 & 4 & 0 \end{Vmatrix}$$

Суммы ее элементов в каждой строке равны

$$\text{СВВ}(1) = 0 + 2 + 3 + 3 = 8,$$

$$\text{СВВ}(2) = 4 + 0 + 2 + 1 = 7,$$

$$\text{СВВ}(3) = 6 + 2 + 0 + 3 = 11,$$

$$\text{СВВ}(4) = 3 + 5 + 4 + 0 = 12.$$

Следовательно, $\min\{\text{СВВ}(i)\} = \min\{8, 7, 11, 12\} = 7 = \text{СВВ}(2)$, и вершина *2* является медианой этого графа. Сумма расстояний от вершины *2* ко всем остальным вершинам равна 7 единицам.

Главная медиана

Главная медиана — это такая вершина x, что сумма расстояний от нее до каждой из дуг минимальна. Под расстоянием от вершины до дуги понимается максимальное расстояние от вершины до точек на этой дуге. Таким образом, главная медиана — это такая вершина x, что

$$\text{СВД}(x) = \min_i \{\text{СВД}(i)\}.$$

Сумма значений элементов i-й строки матрицы D' расстояний вершина — дуга равна сумме расстояний от вершины i до всех дуг, т. е. равна $CBD(i)$. Следовательно, главная медиана соответствует любой строке матрицы D', для которой сумма значений элементов минимальна.

ПРИМЕР 2. Найти главную медиану на графе, изображенном на рис. 8.4. Из предыдущих примеров мы знаем, что для этого графа матрица D' имеет вид

$$D' = \begin{Vmatrix} 2 & 3 & 3 & 3 & 3\,1/2 & 5 \\ 6 & 7 & 4 & 1 & 2 & 3\,1/2 \\ 8 & 9 & 6 & 3 & 2 & 3\,1/2 \\ 5 & 6 & 3 & 6 & 5\,1/2 & 4 \end{Vmatrix}$$

Суммы элементов матрицы D' в каждой строке равны

$$\text{СВД}(1) = 2 + 3 + 3 + 3 + 3\,1/2 + 5 = 19\,1/2,$$

$$\text{СВД}(2) = 6 + 7 + 4 + 1 + 2 + 3\,1/2 = 23\,1/2,$$

$$\text{СВД}(3) = 8 + 9 + 6 + 3 + 2 + 3\,1/2 = 31\,1/2,$$

$$\text{СВД}(4) = 5 + 6 + 3 + 6 + 5\,1/2 + 4 = 29\,1/2.$$

Следовательно, $\min_i \{\text{СВД}(i)\} = \min\{19\,1/2,\ 23\,1/2,\ 31\,1/2,\ 29\,1/2\} = 19\,1/2 = \text{СВД}(1)$, т. е. вершина *1* является главной медианой этого графа. Суммарное расстояние от вершины *1* до всех вершин графа равно $19\,1/2$ единицам.

Абсолютная медиана

Абсолютная медиана — это такая точка на дуге графа, что сумма расстояний от нее до всех вершин графа минимальна.

ТЕОРЕМА 8.1. В графе всегда существует вершина, являющаяся абсолютной медианой.

Доказательство. Рассмотрим функцию, характеризующую расстояния точка — вершина $d(f-(r, s), j)$ для каждой вершины j.

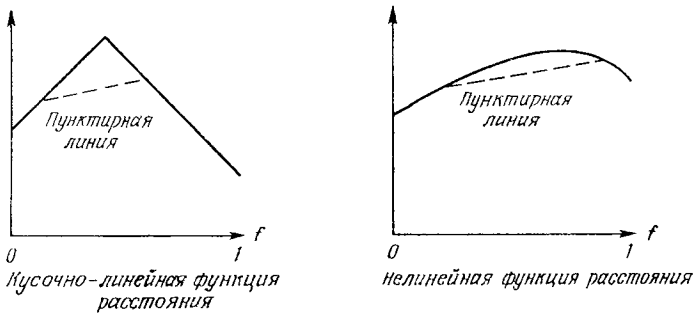


Рис. 8.8. Вогнутые функции расстояний.

В графическом представлении эта функция имеет вид кривой, показанной на рис. 8.2. Заметим, что если соединить любые две точки этой кривой отрезком прямой линии (рис. 8.8), то этот отрезок всегда будет целиком лежать на кривой или под ней. Любая функция, обладающая таким свойством, называется *вогнутой*. Кроме того, минимальное значение вогнутой функции всегда находится в одной из ее граничных точек, т. е. либо в точке $f = 0$, либо в точке $f = 1$.

Рассмотрим далее значение СТВ $(f-(r, s)) = \Sigma d(f-(r, s)\, j)$ как функцию от f. Поскольку эта функция равна сумме вогнутых функций, она также должна быть вогнутой. Таким образом, СТВ$(f-(r, s))$ достигает минимума либо при $f = 0$, либо при $f = 1$. Следовательно, ни одна из внутренних точек дуги (r, s) не может стать абсолютной медианой, а наилучшим претендентом является одна из ее конечных вершин. Теорема доказана.

Отметим, что доказательство теоремы 8.1 остается справедливым и в тех случаях, когда вместо расстояний точка — вершина, определяемых уравнением (1), рассматривается любая вогнутая функция от f, характеризующая расстояния точка — вершина. Из теоремы 8.1 следует, что при поиске абсолютной медианы достаточно исследовать только вершины графа. Таким образом, *любая медиана является также и абсолютной медианой*, и поэтому для ее отыскания не требуется никаких новых методов.

Главная абсолютная медиана

Главная абсолютная медиана — это такая точка на дуге графа, сумма расстояний от которой до всех дуг графа минимальна. Как и ранее, будем считать, что в качестве расстояния от некоторой точки до дуги принимается максимальное из расстояний от данной точки до всех точек рассматриваемой дуги. Таким образом, главная абсолютная медиана — это точка $f-(r, s)$, такая, что

$$\text{СТД}\,(f-(r, s)) = \min_{f-(f, u)\in P} \{\text{СТД}\,(f-(t, u))\}.$$

Теорема 8.1 устанавливает, что в графе всегда существует вершина, являющаяся абсолютной медианой. В доказательстве этой теоремы используется вогнутость функций расстояния точка — вершина. Если бы оказались вогнутыми еще и функции расстояния точка — дуга, то можно было бы доказать аналогичную теорему для главной абсолютной медианы. К сожалению, как можно увидеть на рис. 8.3, функция $d'(f-(r, s), (r, s))$ расстояний точка — дуга не является вогнутой. Все остальные функции расстояний точка — дуга являются вогнутыми, как это видно из рис. 8.2. Следовательно, вполне возможно, что все главные абсолютные медианы являются внутренними точками на дугах. (Пример, иллюстрирующий этот случай, будет приведен ниже.) Однако имеется возможность исключить внутренние точки некоторых дуг. Во-первых, ни одна из внутренних точек ориентированной дуги не может быть главной абсолютной медианой. Это, как и ранее, следует из того, что вершина, в которую входит дуга, является лучшим претендентом на роль главной абсолютной медианы, чем любая внутренняя точка этой ориентированной дуги. Кроме того, справедлива следующая теорема.

ТЕОРЕМА 8.2. Если для неориентированной дуги (r, s) выполняется неравенство

$$|\,\text{СВД}\,(r) - \text{СВД}\,(s)\,| > {}^1/_2\,[d\,(r, s) + d\,(s, r)]\,, \tag{12}$$

то ни одна из внутренних точек этой дуги не может быть главной абсолютной медианой.

Доказательство. Суммарное расстояние от вершины r до всех дуг графа равно

$$\text{СВД}\,(r) = d'\,(r, (r, s)) + \sum_{(x, y)\,:\,(x, y)\neq(z, s)} d'\,(r, (x, y))\,. \tag{13}$$

Аналогично

$$\text{СВД}\,(s) = d'\,(s, (r, s)) + \sum_{(x, y)\,:\,(x, y)\neq(r, s)} d'\,(s, (x, y)). \tag{14}$$

Можно предположить без ограничения общности, что $\text{СВД}(r) \leqslant$

$\leqslant$ СВД(s). Так как $d'(f-(r, s), (t, u))$ — вогнутая функция для всех $(t, u) \neq (r, s)$, то точки функции $\sum\limits_{(x, y):(x, y)\neq(r, s)} d'(f-(r, s), (x, y))$ лежат выше прямой, проходящей через две ее граничные точки. Значения функции $\sum\limits_{(x,y);\ (x,y)\neq(r, s)} d'(f-(r, s), (x, y))$ в этих точках равны

$$\sum_{(x, y):(x, y)\neq(r, s)} d'(0-(r, s), (x, y)) = \sum_{(x, y):(x, y)\neq(r, s)} d'(r, (x, y)),$$

$$\sum_{(x, y):(x, y)\neq(r, s)} d'(1-(r, s), (s, y)) = \sum_{(x, y):(x, y)\neq(r, s)} d'(s, (x, y)).$$

Таким образом, вследствие вогнутости функции при $f = 1/2$

$$\sum_{(x, y):(x, y)\neq(r, s)} d'(1/2-(r, s), (x, y)) \geqslant 1/2 \sum_{(x, y):(x, y)\neq(r, s)} [d'(s, (x, y)) + d'(r, (x, y))].$$

Следовательно, при возрастании f от 0 до $1/2$ функция

$$\sum_{(x, y):(x, y)\neq(r, s)} d'(f-(r, s), (x, y))$$

возрастает по крайней мере на величину

$$1/2 \sum_{(x, y):(x, y)\neq(r, s)} [d'(s(x, y)) - d'(r, (x, y))].$$

Рассмотрим далее функцию $d'(f-(r, s), (r, s))$. Она достигает минимального значения в точке $f = 1/2$. Это значение равно $d'(1/2-(r, s), (r, s))$ (см. уравнение (3г) и рис. 8.8). При изменении f от 0 до $1/2$ $d'(f-(r, s), (r, s))$ уменьшается на величину $d'(r, (r, s)) = 1/2\, a(r, s)$. Если для некоторого значения $0 < f < 1$ функция $\sum\limits_{(x, y)} d'(f-(r, s), (x, y)) <$ СВД(r), то необходимо, чтобы максимальное уменьшение $d'(f-(r, s), (r, s))$ в точке $f = 1/2$ было по крайней мере не меньше минимального возрастания $\sum\limits_{(x, y):(x, y)\neq(r, a)} d'(f-(r, s), (x, y))$ в точке $f = 1/2$. Или иначе, если внутренняя точка ребра (r,s) является лучшим претендентом на роль главной абсолютной медианы, чем вершина r, то должно быть справедливо неравенство

$$d'(r, (r, s)) - 1/2\, a(r, s) \geqslant 1/2 \sum_{(x, y):(x, y)\neq(r, s)} [d'(s, (x, y)) - d'(r, (x, y))]. \qquad (15)$$

Аналогичное неравенство справедливо для вершины s и, следовательно,

$$d'(s, (r, s)) + d'(r, (r, s)) - a(z, s) \geqslant \text{СВД}(s) - \text{СВД}(r). \quad (16)$$

Подставляя уравнение (2а) в неравенство (16), получаем

$$^1/_2 (d(s, r) + a(r, s)) + {}^1/_2 (d(r, s) + a(r, s)) - a(r, s) \geqslant$$
$$\geqslant \text{СВД}(s) - \text{СВД}(r). \quad (17)$$

После упрощения неравенство (17) принимает вид

$$^1/_2 (d(r, s) + d(s, r)) \geqslant \text{СВД}(s) - \text{СВД}(r). \quad (18)$$

Если бы мы предположили, что СВД(s) $\leqslant$ СВД(r), то неравенство (18) имело бы вид

$$^1/_2 (d(r, s) + d(s, r)) \geqslant \text{СВД}(r) - \text{СВД}(s). \quad (19)$$

Объединение неравенств (18) и (19) приводит к противоречию с неравенством (12). Значит, внутренняя точка на дуге (r, s) не может быть лучшим претендентом, чем вершина. Теорема доказана.

Теорема 8.2 указывает простой способ исключения из дальнейшего рассмотрения некоторых ребер, внутренние точки которых не могут быть главной абсолютной медианой. Для проверки выполнения условия (12) необходимо только знать значение элементов матрицы D расстояний вершина — вершина и матрицы D' расстояний вершина — дуга. Возникает следующий вопрос: если с помощью теоремы 8.2 из графа исключаются не все ребра, то возможно ли их дальнейшее исключение? Ответ на вопрос дает следующая лемма.

ЛЕММА 8.1. Для любой внутренней точки f на произвольном ребре (r, s) справедливы соотношения

$$\text{СТД}(f - (r, s)) \geqslant \text{СТД}(r) - {}^1/_2 d(r, s), \quad (20)$$

$$\text{СТД}(f - (r, s)) \geqslant \text{СТД}(s) - {}^1/_2 d(s, r). \quad (21)$$

Доказательство. Из доказательства теоремы 8.2 мы знаем, что при возрастании f от 0 до $^1/_2$ расстояние $d'(f - (r, s), (r, s))$ уменьшается на величину $d'(r, (r, s)) = {}^1/_2 a(r, s)$, которая в соответствии с уравнением (2а) равна $^1/_2 d(r, s)$, т. е. условие (20) выполняется.

Если f убывает от 1 до $^1/_2$, то $d'(f - (r, s), (r, s))$ уменьшается на величину $d'(s, (r, s)) - {}^1/_2 a(r, s) = {}^1/_2 d(s, r)$, и условие (21) также выполняется. Лемма доказана.

Лемма 8.1 может быть использована при получении нижней границы суммарного расстояния до каждой точки произвольного ребра, которое не было исключено из рассмотрения с помощью теоремы 8.2. Каждая из этих нижних границ может быть сравнена с наименьшим суммарным расстоянием от некоторой вершины до каждой из дуг графа, а именно с $\min\{\text{СВД}(i)\}$. Если нижняя

граница для некоторого ребра превышает наименьшее значение суммарного расстояния от вершины, то это ребро может быть исключено из дальнейшего рассмотрения. Для каждого оставшегося не исключенным ребра (r, s) необходимо определить точку наилучшего претендента, т. е. точку, в которой достигается минимальное по всем f значение СТД$(f-(r, s))$. Если суммарное расстояние от этой точки до всех ребер графа будет меньше, чем нижняя граница некоторых ребер, для которых еще не вычислялись значения СТД, то последние могут быть также исключены из дальнейшего рассмотрения.

В конечном счете все ребра графа должны оказаться принадлежащими либо к множеству исключенных из рассмотрения, либо к множеству ребер, для которых определены минимальные по f значения СТД.

Главный абсолютный центр выбирается из множества претендентов — вершин и внутренних точек, не исключенных из рассмотрения ребер графа.

ПРИМЕР 3. Найти главную абсолютную медиану для графа, представленного на рис. 8.1. Используя алгоритм Флойда или алгоритм Данцига, строим матрицу D расстояний вершина — вершина:

$$D = \begin{Vmatrix} 0 & 1 & 2 & 3 & 2 & 3 \\ 1 & 0 & 1 & 2 & 1 & 2 \\ 2 & 1 & 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 & 1 & 2 \\ 2 & 1 & 2 & 1 & 0 & 1 \\ 3 & 2 & 3 & 2 & 1 & 0 \end{Vmatrix}$$

Устанавливаем следующую нумерацию дуг:
1 — (1,2); 2 — (2,3); 3 = (3,4); 4 — (4,5); 5 — (5,6), 6 — (2,5).
Из уравнения (2а) можно вычислить расстояния d' вершина — дуга как элементы матрицы D':

$$D' = \begin{Vmatrix} 1 & 2 & 3 & 3 & 3 & 2 \\ 1 & 1 & 2 & 2 & 2 & 1 \\ 2 & 1 & 1 & 2 & 3 & 2 \\ 3 & 2 & 1 & 1 & 2 & 2 \\ 2 & 2 & 2 & 1 & 1 & 1 \\ 3 & 3 & 3 & 2 & 1 & 2 \end{Vmatrix}$$

Суммы элементов строк этой матрицы равны

СВД (1) $= 1+2+3+3+3+2=14$
СВД (2) $= 1+1+2+2+2+1=9$
СВД (3) $= 2+1+1+2+3+2=11$
СВД (4) $= 3+2+1+1+2+2=11$
СВД (5) $= 2+2+2+1+1+1=9$
СВД (6) $= 3+3+3+2+1+2=14$

Следовательно, вершины *2* и *5* являются наилучшими претендентами на размещение главной абсолютной медианы, так как для каждой из этих вершин суммарное расстояние до всех других вершин графа равно 9 единицам. Далее попытаемся с помощью условия (12) теоремы 8.2 исключить некоторые ребра графа. Заметим, что в рассмотренном графе все дуги являются неориентированными и, следовательно, правая часть условия (12) равна $d(r, s)$.

1. Ребро (1,2) исключается из рассмотрения, так как

$$|\,\text{СВД}(1) - \text{СВД}(2)\,| = |\,14 - 9\,| = 5 > 1 = d(1,\ 2).$$

2. Ребро (2,3) также исключается из рассмотрения, поскольку

$$|\,\text{СВД}(2) - \text{СВД}(3)\,| = |\,9 - 11\,| = 2 > 1 = d(2,\ 3).$$

3. Ребро (3,4) не исключается, в связи с тем что

$$|\,\text{СВД}(3) - \text{СВД}(4)\,| = |\,11 - 11\,| = 0 < 1 = d(3,\ 4).$$

4. Ребро (4,5) исключается из рассмотрения, так как

$$|\,\text{СВД}(4) - \text{СВД}(5)\,| = |\,11 - 9\,| = 2 > 1 = d(4,\ 5).$$

5. Ребро (5,6) исключается из рассмотрения, поскольку

$$|\,\text{СВД}(5) - \text{СВД}(6)\,| = |\,9 - 14\,| = 5 > 1 = d(5,\ 6).$$

6. Ребро (2,5) не исключается, так как

$$|\,\text{СВД}(2) - \text{СВД}(5)\,| = |\,0 - 0\,| = 0 < 1 = d(2,\ 5).$$

Таким образом, дальнейшему рассмотрению подлежат только ребра (3,4) и (2,5). Проверим далее с помощью условий (20) и (21), имеется ли возможность исключить из рассмотрения оставшиеся ребра.

1. Ребро (3,4) может быть исключено из рассмотрения в соответствии с условием (20), так как

$$\text{СВД}(f - (3,\ 4)) \geqslant \text{СВД}(3) - {}^1/_2\, d(3,\ 4) = 11 - {}^1/_2 = 10\,{}^1/_2.$$

То есть нижняя граница СВД (f—(3,4)) превышает величину 9 единиц, достигаемую при выборе вершины *2* графа в качестве главной абсолютной медианы.

2. Ребро (2,5) не может быть исключено из дальнейшего рассмотрения, так как полученная с помощью условий (20) и (21) нижняя граница не превышает значения минимального суммарного расстояния от наилучшего претендента среди вершин до всех дуг графа.

Действительно,

$$\text{СТД}(f - (2,5)) \geqslant \text{СТД}(2) - {}^1/_2\, d(2,5) = 9 - {}^1/_2 < 9,$$

$$\text{СТД}(f - (2,\ 5)) \geqslant \text{СТД}(5) - {}^1/_2\, (5,\ 2) = 9 - {}^1/_2 < 9.$$

Следовательно, для дальнейшего рассмотрения остается только ребро (2,5). Для получения расстояний точка — дуга на дуге (2,5) могут быть использованы уравнения (3а) и (3г):

$$d'(f - (2,5),\ (1,2)) = 1 + f \qquad d'(f - (2,5),\ (4,5)) = 1 + (1 - f)$$
$$d'(f - (2,5),\ (2,3)) = 1 + f \qquad d'(f - (2,5),\ (5,6)) = 1 + (1 - f)$$
$$d'(f - (2,5),\ (3,4)) = 2 \qquad d'(f - (2,5),\ (2,5)) = 1 + (1 - f)$$

Сложив эти расстояния точка — дуга, получаем

$$\text{СТД}(f - (2,\ 5)) = (1 + f) + (1 + f) + 2 + (2 - f) + (2{}^1_2 - f) +$$

$+ \max\{f,\ (1-f)\} = 8 + \max\{(1-f),\ f\}$.

Поскольку $\min\limits_{0 \leqslant f \leqslant 1} \max\{f, (1-f)\}$ достигается при $f = 1/2$, то наилучшим претендентом на размещение главной абсолютной медианы на ребре (2,5) является $1/2$-точка, так как СТД $1/2 - (2,5)) = 8 1/2$. Наилучшему претенденту среди вершин соответствует суммарное расстояние от вершины до всех дуг, равное 9 единицам. Поэтому мы выбираем половинную точку на дуге (2,5) в качестве главной абсолютной медианы этого графа; суммарное расстояние этой точки от всех дуг графа равно 8,5 единицы.

8.4. Обобщения

Взвешенное размещение

В предыдущих разделах при решении задачи размещения предполагалось, что каждая вершина и каждая дуга графа имеют одинаковый вес. Однако, как показано в примере 4 разд. 8.1, в практике встречаются ситуации, когда целесообразно приписать вершинам графа различные веса и умножать расстояние до вершины на вес этой вершины. Существуют также ситуации, в которых следует умножать расстояние до дуги на вес дуги, например если дуги представляют участки автомагистрали, которые должны быть обслужены центральной станцией аварийного обслуживания. Каждому участку автомагистрали следовало бы приписать вес, соответствующий интенсивности движения по нему транспорта. Методы решения задач размещения, рассмотренные в разд. 8.2 и 8.3, легко могут быть обобщены для нахождения решения задач размещения при наличии весов вершин и дуг, т. е. задач нахождения взвешенного центра, взвешенного главного центра, взвешенного абсолютного центра и т. д. Для этого достаточно умножить каждое из расстояний (т. е. каждое расстояние вершина — вершина, вершина — дуга, точка — вершина, точка — дуга) на вес, приписанный конечной вершине или конечной дуге, до которых измеряется расстояние. Если вместо исходных невзвешенных использовать взвешенные расстояния, то с помощью методов, рассмотренных в разд. 8.2 и 8.3, будут получаться соответствующие решения задач взвешенных размещений.

Кратные центры и кратные медианы

В разд. 8.2 и 8.3 рассматривались задачи выбора в качестве центра или медианы ровно одной вершины или точки на дуге графа.

Предположим, что в отличие от рассмотренных случаев условия задачи позволяют выбрать место размещения нескольких пунктов обслуживания. Тогда каждая вершина (или дуга) должна быть поставлена в соответствие ближайшему к ней пункту обслуживания. Решение таких задач размещения значительно усложняет-

ся тем, что оно включает две стадии: (а) разбиение множества вершин (или дуг) на подмножества и (б) выбор наилучшего размещения для всех элементов каждого из подмножеств вершин (или дуг).

К сожалению, все применяемые для решения этих задач методы в конечном счете сводятся к решению задач целочисленного программирования (Кристофидес и Виола [1], Марстен [6], Майника [7, 8], Ревель и Свэйн [9], Торегас [10]). Любое сколько-нибудь детальное обсуждение этих методов увело бы нас слишком далеко. Для ознакомления с ними следует обратиться к специальной литературе по целочисленному программированию.

Однако стоит отметить один результат, относящийся к нахождению кратных абсолютных медиан (Хакими [4], Голдман [2]). Предположим, что мы ищем такое расположение p точек размещения ($p>1$), что каждая вершина приписана к ближайшему пункту и суммарное расстояние от каждого пункта до приписанных к нему вершин минимально. Множество таких точек размещения называется абсолютной p-медианой.

ТЕОРЕМА 8.3. Существует абсолютная p-медиана, которая состоит только из вершин графа.

Доказательство. В теореме 8.1 этот результат установлен для случая $p = 1$. Для $p>1$ все вершины разбиваются на p подмножеств, каждому из которых соответствует одна медиана. Мы знаем, что поскольку любая сумма функций расстояний точка — вершина является вогнутой функцией, то медианой является вершина. Теорема доказана.

УПРАЖНЕНИЯ

1. Города a, b, c, d соединяются дорогами, как показано на рис. 8.9. Постройте для этого графа матрицы D и D' и определите в нем:

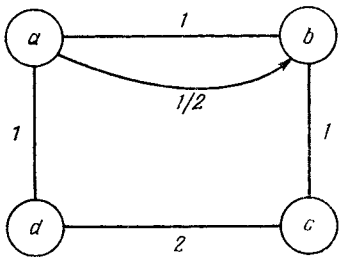


Рис. 8.9.

а) центр,
б) абсолютный центр,
в) главный центр,
г) главный абсолютный центр,
д) медиану,
е) абсолютную медиану,
ж) главную медиану,
з) главную абсолютную медиану.

2. Предположим, что между городами b и d предыдущего примера построена дополнительная дорога протяженностью в $1^1/_2$ единицы. Повторите для этого случая упражнение 1, считая, что по новой дороге допускается движение в обоих направлениях.

3. Предполагается, что по каждому ребру (x, y) допускается движение в обоих направлениях, но благодаря действию ветра расстояние от x к y эквивалентно 5 единицам, а расстояние от y к x — 7 единицам. Каким образом можно включить в нашу модель определения центров и медиан такое ребро, как (x, y)? (Покажите, почему это ребро (x, y) нельзя заменить парой дуг (x, y) и (y, x)?)

4. Предположим, что город a имеет численность населения вдвое большую, чем город b, а город b вдвое большую, чем город c, численность населения которого одинакова с городом d. Повторите упражнение 1, используя приведенные выше веса.

5. Упростите рассмотренный в разд. 8.2 метод нахождения абсолютного центра в случае, когда рассматриваемый граф является деревом.

6. Покажите, что в качестве абсолютного центра или абсолютной медианы никогда не может быть выбрана внутренняя точка ориентированной дуги.

7. Определите на сколько можно увеличить длину ребра (1,4) графа на рис. 8.4 без изменения местоположения его центра медианы? На сколько можно увеличить длину ребра (2,3) этого графа без изменения местоположения его абсолютного центра? Абсолютной медианы?

ЛИТЕРАТУРА

1. Christofides N., Viola P., The Optimum Location of Multicentres of a Graph, *O. R. Quart.*, **22**, pp. 45 — 54, 1971.
2. Gold man A. J., Optimum Locations for Centers in a Network, *Transp. Sci.*, **4**, pp. 352 — 360, 1969.
3. Hakimi S. L., Optim um Locations of Switching Centers and the Absolute Centers and Medians of a Graph, *ORSA*, **12** pp. 450 — 459, 1964.
4. Hakimi S. L., Optimum Distribution of Switching Centers in a Communications Network and Some Graph Theoretic Problems, *ORSA*, **13**, pp. 462 — 475, 1965.
5. Levy J., An Extended Theorem for Location in a Network, *O. R. Quart.*, **18**, pp. 433 — 442, 1967.
6. Marsten R., An Algorithm for Finding Almost All of the Medians of a Network, **DP № 23**, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, November, 1972.
7. Minieka E., The m-Center Problem, *SIAM Rev.*, **12**, pp. 138— 139, 1970.
8. Minieka E., The General Centers and Medians of a Graph, *ORSA*, **25**, pp. 641 — 650, 1977.
9. ReVelle G., Swain R., Central Facility Location, *Geographical Analysis*, **2**, No. **1**, 30 — 42, 1970.
10. Toregas C., Revelle C., Swain R., Bergman L., The Location of Emergency Facilities, *ORSA*, **19**, pp. 1363 — 1973, 1971.
11. Wendell R. S., Hurter A. P., Jr., Optimal Locations on a Network, *Transp. Sci.*, **7**, pp. 18 — 33, 1973.

ГЛАВА 9

Сетевые графики

9.1. Метод критического пути (МКП)

Многие крупные проекты, такие, как строительство дома, разработка автоматизированной системы бухгалтерского учета, обучение в институте и даже приготовление обеда, можно разбить на большое количество различных операций (работ). Некоторые из этих операций могут выполняться одновременно, другие — только последовательно: одна операция после окончания другой. При строительстве дома, например, можно совместить во времени внутренние отделочные работы и работы по благоустройству территории, однако возводить стены можно только после закладки фундамента. При изучении институтской программы каждый курс лекций можно рассматривать как операцию. Некоторые курсы допускают совместное изучение, другие же должны изучаться только последовательно. При приготовлении обеда вы можете сервировать стол, пока мясо будет находиться в духовке, но нельзя жарить картофель, если он не будет отмыт от грязи.

Задача управления проектом состоит, таким образом, в том, чтобы обеспечить его своевременное завершение с учетом времени, необходимого для выполнения каждой операции, и определенных взаимосвязей, характеризующих последовательность их выполнения. Если, например, подрядчик задержит закладку фундамента, то дом не будет построен к сроку. Если студент будет слишком долго усваивать основные курсы, он не успеет вовремя окончить институт. Если вы займетесь сервировкой стола и забудете положить мясо в духовку, обед запоздает. Другими словами, необходимо выяснить, какие операции являются *критическими* для своевременного завершения всего проекта.

Каждый проект можно представить в виде некоторого графа, называемого *сетевым графиком*. Для этого зададим:

1. Перечень всех операций проекта.
2. Время, необходимое для выполнения каждой операции.
3. Перечни операций, непосредственно предшествующих каждой операции.

Представим теперь каждую операцию в виде дуги ориентированного графа, построенного по следующему правилу: если некоторая операция представляется дугой (x, y), то в вершину x входят только дуги, представляющие операции, *непосредственно*

предшествующие данной операции. При необходимости введем фиктивные дуги, не представляющие никакой операции.

ПРИМЕР 1. Предположим, что строительство нового дома включает следующие операции:

Операция	Время выполнения	Предшествующие операции	Дуга графа
1. Расчистка участка	1	Нет	(a, b)
2. Закладка фундамента	4	Расчистка участка	(b, c)
3. Возведение стен	4	Закладка фундамента	(c, d)
4. Монтаж электропроводки	3	Возведение стен	$(d, e)_1$
5. Штукатурные работы	4	Монтаж электропроводки, настил крыши	(e, f)
6. Благоустройство территории	6	Возведение стен	(d, g)
7. Отделочные работы	4	Штукатурные работы, настил крыши	(f, g)
8. Настил крыши	5	Возведение стен	$(d, e)_2$

Заметим, что условие, требующее, чтобы настил крыши предшествовал отделочным работам, излишне и его можно исключить. Действительно, на-

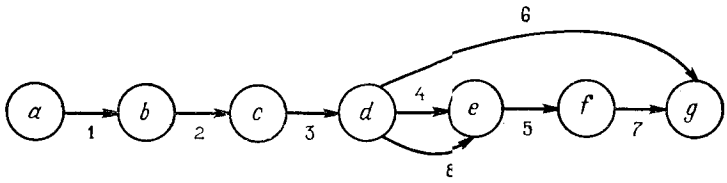


Рис. 9.1. Сетевой график строительства дома.

стил крыши должен быть выполнен до начала штукатурных работ, которые в свою очередь должны закончиться до начала отделочных работ. Сетевой график такого проекта изображен на рис. 9.1. Он содержит две параллельные дуги, соединяющие вершины d и e. На практике параллельные дуги обычно либо заменяются одной дугой, представляющей совместную операцию (в нашем случае монтаж электропроводки и настил крыши), либо одна из параллельных дуг разбивается на две следующие друг за другом дуги. Для облегчения последующего изложения мы не будем осуществлять этих преобразований.

ПРИМЕР 2. Рассмотрим следующий абстрактный проект:

Операция	Предшествующие операции
A	—
B	A
C	A
D	A
E	B, C
F	B, C, D
G	E, F

Операции, предшествующие операции E, образуют подмножество множества операций, предшествующих операции F. Чтобы обеспечить предшествование операций B и C операции F, в этом случае необходимо ввести фиктивную операцию и соответствующую ей фиктивную дугу (c, d), как по-

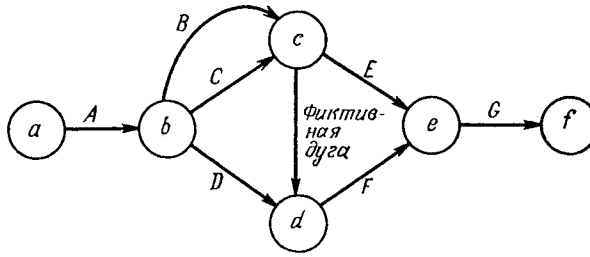


Рис. 9.2. Сетевой график с фиктивной дугой.

казано на рис. 9.2. Время выполнения фиктивных операций всегда полагается равным нулю.

ПРИМЕР 3. Для освоения курса исследования операций студент должен изучить следующие дисциплины: (*а*) вычислительную математику (два семестра — ВЫЧ 1 и ВЫЧ 2), (б) статистику (три семестра — СТАТ 1, СТАТ 2 и СТАТ 3), (в) линейное программирование (один семестр — ЛП), (г) нелинейное программирование (один семестр — НП), (д) стохастическое программирование (один семестр — СП). При этом очевидно, что к изучению вычислительной математики во втором семестре (ВЫЧ 2) можно приступить только после освоения материала семестра 1 (ВЫЧ 1), к изучению СТАТ 3 — после освоения СТАТ 2, к изучению СТАТ 2 — после освоения СТАТ 1 и ВЫЧ 2, к изучению СТАТ 1 — после освоения ВЫЧ 1. Для изучения линейного программирования (ЛП) не требуется знания никаких предварительных дисциплин, для изучения нелинейного программирования (НП) требуется знание ВЫЧ 2 и ЛП, для изучения стохастического программирования (СП) — знание ВЫЧ 2, СТАТ 2 и ЛП.

Сетевой график этой учебной программы изображен на рис. 9.3. Введение в нем фиктивной дуги (c, d) потребовалось потому, что изучение СТАТ 2 требует освоения СТАТ 1 и ВЫЧ 2, а изучение НП — ВЫЧ 2 и ЛП.

На всю эту учебную программу студенту потребуется самое меньшее 5 семестров, если он последовательно, семестр за семестром, пройдет ВЫЧ 1, СТАТ 1, СТАТ 2, СТАТ 3 и СП. Любая задержка в изучении указанной последовательности дисциплин приведет к такой же по времени задержке окончания всего курса обучения.

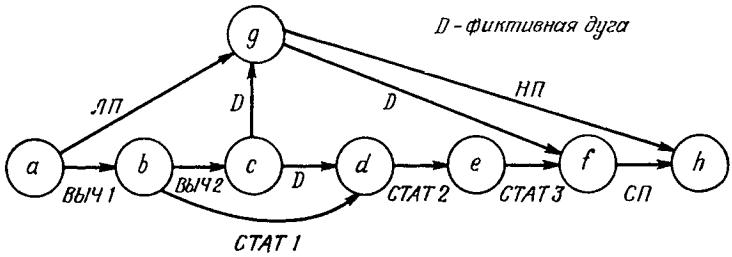


Рис. 9.3. Сетевой график программы изучения курса исследования операций.

Граф, изображающий отношения предшествования между операциями проекта, называется *сетевым графиком*. Дуги сетевого графика всегда являются ориентированными и представляют реальные или фиктивные операции. Вершины сетевого графика называются *событиями*. Говорят, что событие *произошло*, если все операции, которые отображаются дугами, входящими в соответствующую вершину, полностью завершены.

Сетевой график не может содержать контуров. Действительно, пусть в нем имеется контур (a, b), (b, c), ..., (r, s), (s, a). Тогда событие a не может произойти, пока не произошло событие s, событие s не может произойти, пока не произошло событие r, и т. д. Очевидно, что в этом случае проект никогда не может быть закончен. Но каждый реальный проект должен допускать возможность его завершения, следовательно, в сетевом графике проекта должны отсутствовать контуры.

Рассмотрим сетевой график $G = (X, A)$ с множеством событий X и множеством операций A. Для облегчения дальнейшего изложения предположим, что граф G содержит ровно одно событие, в которое не входит ни одной дуги, и ровно одно событие, из которого не выходит ни одной дуги. Будем называть эти события *начальным* и *конечным* событиями соответственно (они аналогичны источнику и стоку сети).

Отсутствие контуров в G позволяет перенумеровать события 1, 2, ... таким образом, чтобы для каждой дуги (i, j) соблюдать условие $i < j$. Такая нумерация может быть получена при помощи следующего алгоритма.

Алгоритм нумерации событий

Шаг 1. Присвоить начальному событию номер 1.
Шаг 2. Присвоить следующий номер любому неперенумерованному событию, для которого все предшествующие события перенумерованы (такое событие всегда существует благодаря отсутствию контуров). Повторять шаг 2 до тех пор, пока все события не будут перенумерованы. Конечное событие при этом всегда получит последний (наибольший) номер.

ПРИМЕР 4. (*Алгоритм нумерации событий.*) Перенумеруем события сетевого графика, изображенного на рис. 9.2. Событие a — начальное, присвоим ему номер 1 (шаг 1). На шаге 2 событие b получает номер 2, так как все предшествующие ему события (т. е. событие a) перенумерованы. Далее событие c получает номер 3, событие d — номер 4, событие e — номер 5 и конечное событие f — номер 6. Отметим, что в этом примере отсутствовал выбор события, которому должен был быть присвоен следующий номер.

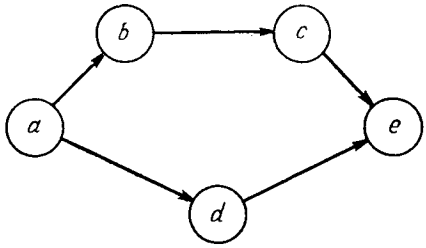


Рис. 9.4. Сетевой график с тремя возможными нумерациями событий.

Для сетевого графика, изображенного на рис. 9.4, возможны несколько различных нумераций событий:

Событие	Первая нумерация	Вторая нумерация	Третья нумерация
a	1	1	1
b	3	2	2
c	4	4	3
d	2	3	4
e	5	5	5

Обозначим время, необходимое для выполнения операции (x, y), через $t(x, y) \geqslant 0$. Если (x, y) — фиктивная операция, положим $t(x, y) = 0$.

Проанализируем теперь сетевой график, чтобы определить, как быстро может быть закончен изображенный на нем проект и какие операции являются критическими для его своевременного окончания.

Для каждого события $x \in X$ пусть $E(x)$ обозначает *наиболее ранний* из возможных сроков его наступления, а $L(x)$ — *наиболее поздний* срок появления события x, еще допускающий своевременное окончание всего проекта.

На рис. 9.3, например $E(b) = 1$, так как событие b может произойти не ранее окончания первого семестра. Пусть полный срок обучения (срок

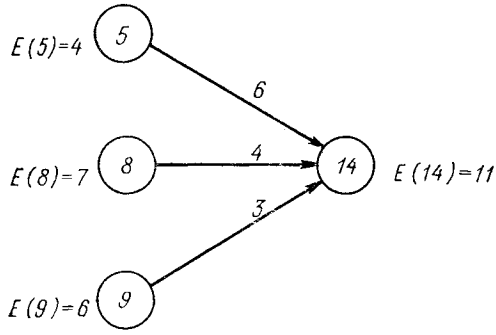


Рис. 9.5. Вычисление наиболее ранних сроков наступления событий.

окончания проекта) ограничен пятью семестрами, тогда $L(b) = 1$, так как если событие b произойдет позднее окончания первого семестра, то СТАТ 1, СТАТ 2, СТАТ 3 и СП не будут изучены к сроку.

В качестве другого примера рассмотрим рис. 9.5. Пусть событию 14 непосредственно предшествуют ровно три события: 5, 8 и 9, причем $E(5) = 4$, $E(8) = 7$, $E(9) = 6$. Время наступления события 14 не может быть меньше 10, так как $E(5) + t(5, 14) = 4 + 6 = 10$. Одновременно это время не может быть меньше 11, так как $E(8) + t(8, 14) = 7 + 4 = 11$, и меньше 9, так как $E(9) + t(9, 14) = 6 + 3 = 9$. Таким образом, $E(14) = \max\{10, 11, 9\} = 11$.

В общем случае

$$E(j) = \max_{i:(i,j)\in A} \{E(i) + t(i, j)\}. \qquad (1)$$

Соотношение (1) дает возможность построить *алгоритм расчета наиболее ранних сроков наступления событий.*

Алгоритм расчета наиболее ранних сроков наступления событий

Шаг 1. Пронумеровать события $1, 2, \ldots, n = |X|$ таким образом, чтобы для каждой операции (i, j) выполнялось условие $i < j$. Использовать для этого алгоритм нумерации событий. Положить $E(1) = 0$.

Шаг 2. Для $j = 2, 3, \ldots, n$ вычислить

$$E(j) = \max_{i:(i,j)\in A} \{E(i) + t(i, j)\}.$$

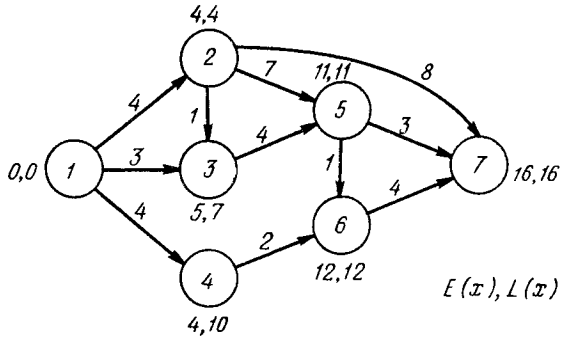


Рис. 9.6. Пример расчета сроков наступления событий.

ПРИМЕР 5. (*Алгоритм расчета ранних сроков наступления событий.*) Вычислим ранние сроки событий для сетевого графика, изображенного на рис. 9.6. Событиям уже присвоены номера от 1 до 7.

Шаг 1. $E(1) = 0$.

Шаг 2. $E(2) = E(1) + t(1, 2) = 0 + 4 = 4$.

$$E(3) = \max[E(1) + t(1, 3),\ E(2) + t(2, 3)] =$$
$$= \max[4 + 1,\ 0 + 3] = 5.$$

$$E(4) = E(1) + t(1, 4) = 0 + 4 = 4.$$

$$E(5) = \max[E(2) + t(2, 5),\ E(3) + t(3, 5)] =$$
$$= \max[4 + 7,\ 5 + 4] = 11.$$

$$E(6) = \max[E(4) + t(4, 6),\ E(5) + t(5, 6)] =$$
$$= \max[4 + 2,\ 11 + 1] = 12.$$

$$E(7) = \max[E(2) + t(2, 7),\ E(5) + t(5, 7),\ E(6) + t(6, 7)] =$$
$$= \max[4 + 8,\ 11 + 3,\ 12 + 4] = 16.$$

Проект, таким образом, не может завершиться раньше чем через 16 единиц времени.

Вычислим теперь поздние сроки наступления событий. Рассмотрим рис. 9.7. Событие 17 предшествует ровно трем событиям: 20, 24 и 29. Время наступления события 17 не должно превышать 11; в противном случае событие 20 будет задержано и произойдет позднее $L(20) = 16$. Аналогично время события 17 не должно превышать 15, так как $L(24) - t(17,24) = 19 - 4 = 15$, и не должно превышать 14, так как $L(29) - t(17,29) = 22 - 8 = 14$.

Мы видим, что в общем случае

$$L(i) = \min_{j:(i,\ j)\in A} \{L(j) - t(i, j)\}. \qquad (2)$$

Опираясь на соотношение (2), мы можем теперь описать *алгоритм расчета наиболее поздних сроков событий.*

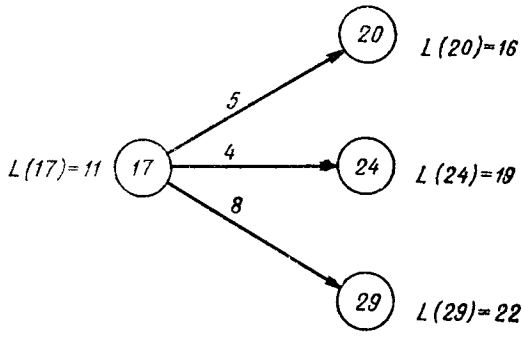


Рис. 9.7. Вычисление наиболее поздних сроков наступления событий

Алгоритм расчета наиболее поздних сроков ссбытий

Шаг 1. Перенумеровать события 1, 2, ..., $n = |X|$ таким образом, чтобы для каждой операции (i, j) выполнялось условие $i < j$. Использовать для этого алгоритм нумерации событий.

Положить $L(n)$ равным заданному времени завершения проекта (во всех реальных ситуациях $L(n) \geqslant E(n)$).

Шаг 2. Для $i = n-1, n-2, \ldots, 1$ вычислить

$$L(i) = \min_{j:(i,j)\in A} \{L(j) - t(i, j)\}.$$

ПРИМЕР 6. (*Алгоритм расчета наиболее поздних сроков наступления событий.*) Вычислим наипоздпейшие сроки событий для сетевого графика, изображенного на рис. 9.6. Пусть $L(7) = E(7) = 16$. Это означает, что проект должен быть закончен как можно раньше, а именно к моменту времени 16.

$L(7) = 16.$

$L(6) = L(7) - t(6,7) = 16 - 4 = 12.$

$L(5) = \min\{L(7) - t(5,7),\ L(6) - t(5,6)\} = \min\{16 - 3,\ 12 - 1\} = 11.$

$L(4) = L(6) - t(4,6) = 12 - 2 = 10.$

$L(3) = L(5) - t(3,5) = 11 - 4 = 7.$

$L(2) = \min\{L(7) - t(2,7),\ L(5) - t(2,5),\ L(3) - t(2,3)\} = \min\{16 - 8,\ 11 - 7,\ 7 - 1\} = 4.$

$L(1) = \min\{L(4) - t(1,4),\ L(3) - t(1,3),\ L(2) - t(1,2)\} = \min\{10 - 4,\ 7 - 3,\ 4 - 4\} = 0.$

Следовательно, чтобы закончить проект к моменту времени 16, необходимо начать его в момент времени 0.

Из алгоритма расчета наиболее поздних сроков наступления событий непосредственно следует, что увеличение наиболее позднего срока окончания всего проекта $L(n)$ на t единиц приведет к увеличению наиболее поздних сроков для всех событий также на t единиц.

Наиболее ранний срок $E(x)$ события x можно интерпретировать как длину пути наибольшей длины от начального события к событию x, а разность $L(n) - L(x)$ — как длину пути наибольшей длины от события x к конечному событию. Отметим еще, что если $L(n) \geqslant E(n)$, то $L\ x) \geqslant E(x)$ для всех событий x.

Алгоритм расчета ранних сроков наступления событий требует одной операции сложения для каждой дуги графика и одной операции сравнения на максимум для каждого события, кроме первого. Алгоритм расчета наиболее поздних сроков событий требует одной операции вычитания для каждой дуги и одной операции сравнения на минимум для каждого события, кроме события n.

Рассмотрим некоторую операцию (x, y). Какое максимальное количество времени можно выделить для ее выполнения без задержки своевременного окончания выполнения всего проекта? Операция (x, y) может начаться не ранее $E(x)$ и должна закончиться не позднее $L(y)$. Таким образом, без задержки окончания проекта на выполнение операции (x, y) можно выделить не более $L(y) - E(x)$ единиц времени. Следовательно, при выполнении этой операции можно допустить максимальную задержку $L(y) - E(x) - t(x, y) \geqslant 0$.

Величина

$$L(y) - E(x) - t(x, y) \tag{3}$$

называется *полным резервом времени операции* (x, y). Очевидно, что задержка выполнения операции, полный резерв времени которой равен нулю, приведет к такой же по времени задержке выполнения всего проекта.

Какое максимальное количество времени может быть выделено для выполнения операции (x, y) без введения дополнительных временны́х ограничений на последующие операции? Для соблюдения этого условия операция (x, y) должна быть закончена к моменту времени $E(y)$. Поскольку операция (x, y) может начаться не ранее $E(x)$, на ее выполнение без введения дополнительных временных ограничений на последующие операции можно выделить не более $E(y) - E(x)$ единиц времени.

Величина

$$E(y) - E(x) - t(x, y) \tag{4}$$

называется *свободным резервом времени* операции (x, y). Свободный резерв времени равен максимальной задержке выполнения операции (x, y), не влияющей на выполнение последующих операций.

Из (1) следует, что свободный резерв времени всегда неотрицателен.

Какое максимальное количество времени может быть выделено для выполнения операции (x, y) без введения дополнительных временных ограничений на любую операцию проекта? Для выполнения этого условия операция (x, y) должна начаться как можно позднее (т. е. в момент времени $L(x)$) и закончиться как можно раньше (т. е. в момент времени $E(y)$). Следовательно, на выполнение операции (x, y) в этом случае можно выделить не более $E(y) - L(x)$ единиц времени.

Величина

$$E(y) - L(x) - t(x, y) \tag{5}$$

называется *независимым резервом времени операции* (x, y). Независимый резерв времени равен максимальной задержке, которую можно допустить при выполнении операции (x, y) без введения дополнительных временных ограничений на любую другую операцию проекта. Отрицательное значение независимого резерва означает, что любая задержка в выполнении операции приведет к дополнительным ограничениям на выполнение других операций.

Как соотносятся между собой значения трех видов резервов? Из соотношений (3) — (5), поскольку $L(x) \geqslant E(x)$ для всех событий x, следует, что для любой операции (x, y)

$$\text{полный резерв} \geqslant \text{свободный резерв} \geqslant \text{независимый резерв} \tag{6}$$

Значения всех трех резервов для операций сетевого графика, изображенного на рис. 9.6, приведены в следующей таблице:

Операция	Полный резерв	Свободный резерв	Независимый резерв
(1,2)	4—0—4=0	4—0—4=0	4—0—4=0 (критическая операция)
(1,3)	7—0—3=4	5—0—3=2	5—0—3=2
(1,4)	10—0—4=6	4—0—4=0	4—0—4=0
(2,3)	7—4—1=2	5—4—1=0	5—4—1=0
(2,5)	11—4—7=0	11—4—7=0	11—4—7=0 (критическая операция)
(2,7)	16—4—8=4	16—4—8=4	16—4—8=4
(3,5)	11—5—4=2	11—5—4=2	11—7—4=0
(4,6)	12—4—2=6	12—4—2=6	12—10—2=0
(5,6)	12—11—1=0	12—11—1=0	12—11—1=0 (критическая операция)
(5,7)	16—11—3=2	16—11—3=2	16—11—3=2
(6,7)	16—12—4=0	16—12—4=0	16—12—4=0 (критическая операция)

Операция называется *критической*, если любая задержка в ее выполнении приводит к задержке завершения всего проекта. Другими словами, критическая операция — это операция, полный резерв времени которой равен нулю.

Знание всех критических операций необходимо для предотвращения задержки их выполнения, вызывающей задержку окончания проекта. Для некритических операций можно допустить задержку, не превышающую полного резерва, не нарушая своевременности окончания проекта.

Напомним, что $E(n)$ равно длине пути наибольшей длины от начального к конечному событию сетевого графика. Если $E(n) = L(n)$, то каждая операция этого пути является критической. Путь, состоящий только из критических операций, называется *критическим путем*.

Например, для сетевого графика, изображенного на рис. 9.6, операции (1,2), (2,5), (5, 6) и (6, 7) имеют полные резервы, равные нулю, т. е. каждая из этих операций является критической. Заметим, что критические операции составляют путь от события 1 к событию 7, длина которого равна $4 + 7 + 1 + 4 = 16$. Этот путь является путем наибольшей длины в сетевом графике. Задержка в выполнении любой операции этого пути приводит к такой же по величине задержке окончания всего проекта.

В качестве другого примера рассмотрим сетевой график, изображенный на рис. 9.8. Пути (1, 2), (2, 5) и (1, 3), (3, 5) имеют одинаковую длину, равную 8. Следовательно, оба пути являются критическими.

При выполнении некритических операций (1, 4) и (4, 5) можно допустить некоторые задержки при условии, что они не задержат окончания проекта позднее момента времени 8. Насколько же можно задерживать выполнение операции (1, 4), чтобы при этом время окончания всего проекта не превысило 8?

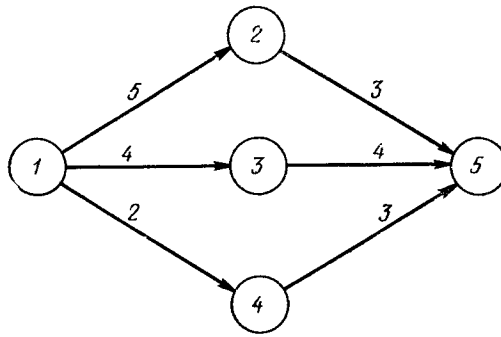


Рис. 9.8. Сетевой график с несколькими критическими путями.

До сих пор мы предполагали, что для каждой операции (x, y) точно известно время ее выполнения $t(x, y)$. Очевидно, что такое предположение мало соответствует действительности. Вряд ли мы можем точно знать, например, за какое время механическая мастерская выполнит работу? Насколько мы можем быть уверены, что студент за один семестр изучит некоторую дисциплину? Знаем ли мы достоверно, сколько времени потребует благоустройство территории дома или приготовление пятифунтового куска мяса?

Для учета неопределенности длительности выполнения операций был разработан метод, известный под названием PERT (Program Evaluation Review Technique). Метод PERT в сущности повторяет описанный метод критического пути с той разницей, что детерминированные длительности выполнения операций заменяются на ожидаемые. Для вычисления ожидаемого времени выполнения операции используются три временные оценки:

A — *оптимистическая оценка времени выполнения*,

B — *реалистическая оценка времени выполнения*,

C — *пессимистическая оценка времени выполнения*.

Ожидаемое время выполнения операции оценивается как

$$\frac{A}{6} + \frac{4}{6} B + \frac{C}{6} \tag{7}$$

Другими словами, берется взвешенная сумма трех оценок, причем оценкам A и C приписываются веса $\frac{1}{6}$, а оценке B — вес $\frac{4}{6}$.

Дисперсия времени выполнения операции задается выражением

$$\left(\frac{C-A}{6}\right)^2. \tag{8}$$

К сетевому графику, в котором длительности выполнения операций вычисляются согласно (7), можно теперь применить метод критического пути (МКП). Тогда $E(x)$ будет обозначать ожидаемый наиболее ранний срок, а $L(x)$ — ожидаемый наиболее поздний срок осуществления события x. $E(n)$, таким образом, будет равно ожидаемому наиболее раннему сроку окончания выполнения всего проекта.

Действительный наиболее ранний срок окончания выполнения всего проекта считается нормально распределенной случайной величиной с математическим ожиданием $E(n)$ и дисперсией, равной сумме дисперсий операций пути наибольшей длины на графике от события 1 к событию n (если существует более одного такого пути, как, например, на рис. 9.8, то выбираются пути с максимальной суммой дисперсий). Такое предположение позволяет сделать ряд вероятностных оценок действительного времени завершения проекта. Детали читатель может найти в любой популярной книге по статистике.

При использовании метода PERT необходимо помнить, что гипотеза о нормальном распределении действительного времени окончания проекта имеет тем меньше оснований, чем больше статистическая зависимость длительностей выполнения отдельных операций. Кроме того, теоретическое обоснование выражений (7) и (8) опирается на весьма сомнительное предположение о бета-распределении продолжительности выполнения операций. Метод PERT тем не менее нашел широкое практическое применение.

9.2. Определение длительности выполнения операций из условия обеспечения минимальной стоимости

Если проект должен быть завершен к возможно более раннему сроку, то время, которое может быть выделено для выполнения каждой операции, весьма ограничено. В самом деле, критические операции, как мы видели в разд. 9.1, должны быть выполнены без какой бы то ни было задержки, а некритические операции (x, y) — к моменту времени $L(y)$. Следовательно, при управлении выполнением проекта возникает задача оптимального распределения задержек между некритическими операциями. В этом разделе рассматривается предложенная Фалкерсоном [3] модель для решения такой задачи по критерию минимизации стоимости.

Предположим, что время $t(x, y)$ выполнения операции (x, y) должно удовлетворять требованиям

$$0 \leqslant r(x, y) \leqslant t(x, y) \leqslant s(x, y). \tag{9}$$

Предположим также, что стоимость выполнения операции (x, y) равна

$$K(x, y) - k(x, y)\, t(x, y), \tag{10}$$

где $K(x, y)$ — произвольная постоянная, а $k(x, y)$ — положительный коэффициент. При увеличении времени выполнения операции на единицу ее стоимость уменьшается на $k(x, y)$. В многих ситуациях предположение о линейности функции стоимости (10) вполне реалистично. Например, время выполнения операции может быть сокращено путем выделения дополнительных одинаково оплачиваемых рабочих.

Возникает вопрос: какое время $t(x, y)$ необходимо выделить для выполнения каждой операции (x, y), чтобы весь проект был закончен к заданному времени T, а общая стоимость выполнения операций была минимальной?

Решение этой задачи состоит в выборе $p(x)$ — оптимального времени возникновения каждого события x — при условиях

$$p(1) = 0, \quad p(n) = T, \tag{11}$$

$$p(y) - p(x) \geqslant r(x, y) \text{ для всех } (x, y). \tag{12}$$

Далее время $t(x, y)$ выполнения каждой операции (x, y) выбирается как можно большим, т. е. $t(x, y)$ полагается равным

$$\min \{p(y) - p(x), s(x, y)\}. \tag{13}$$

Следовательно, для того чтобы решить задачу определения длительности выполнения операций при минимальной стоимости исполнения проекта, необходимо найти только оптимальные значения $p(x)$, удовлетворяющие соотношениям (11), (12).

Задача определения времени выполнения операции при минимальной стоимости описывается следующей задачей линейного программирования:

минимизировать

$$\sum_{(x, y)} [K(x, y) - k(x, y)]\, t(x, y)] \tag{14}$$

при ограничениях

$$p(x) + t(x, y) \leqslant p(y) \text{ для всех } (x, y), \tag{15}$$

$$r(x, y) \leqslant t(x, y) \quad \text{для всех } (x, y), \tag{16}$$

$$t(x, y) = s(x, y) \quad \text{для всех } (x, y), \tag{17}$$

$$p(n) - p(1) \leqslant T. \tag{18}$$

Построим на основе сетевого графика $G = (X, A)$ граф $G' = (X', A')$ следующим образом. Заменим дугу, соответствующую каждой операции $(x, y) \in A$, двумя дугами $(x, y)_1 \in A'$ и $(x, y)_2 \in A'$. Введем также обратную дугу $(n, 1) \in A'$. Пусть $a(x, y)_i$ обозначает стоимость единицы потока по дуге $(x, y)_i$, $i = 1, 2$, где

$$\begin{aligned} a(x, y)_1 &= -s(x, y), \\ a(x, y)_2 &= -r(x, y), \\ a(n, 1) &= T. \end{aligned} \tag{19}$$

Пусть $c(x, y)_i$ обозначает пропускную способность дуги $(x, y)_i$, $i = 1, 2$, где

$$\begin{aligned} c(x, y)_1 &= k(x, y), \\ c(x, y)_2 &= \infty, \\ c(n, 1) &= \infty. \end{aligned} \tag{20}$$

ТЕОРЕМА 9.1. Применим алгоритм дефекта (разд. 4.4) для отыскания потока минимальной стоимости в графе G'. Пусть $p_t(x)$ — значение двойственной переменной для вершины x графа, полученное после окончания работы алгоритма.

Тогда значения $p(x) = -(p_t(x) - p_t(1))$ для всех $x \in X$ являются оптимальными моментами наступления событий.

Доказательство. Поскольку $\sum_{(x,\,y)} K(x, y)$ постоянна, целевую функцию (14) можно преобразовать к виду
максимизировать

$$\sum_{(x,\,y)} k(x, y)\, t(x, y). \tag{21}$$

Выражения (15)—(18), (21) представляют собой задачу линейного программирования относительно переменных $t(x, y)$, $(x, y) \in A$ и $p(x)$, $x \in X$. Запишем двойственную к ней задачу. Для каждой операции (x, y) введем двойственные переменные $f(x, y)$, $h(x, y)$, $g(x, y)$, сопоставленные ограничениям (15), (16), (17) соответственно, и v — двойственную переменную, отвечающую ограничению (18).

Двойственная задача имеет вид

минимизировать

$$Tv - \sum_{(x,\,y)} r(x, y)\, h(x, y) + \sum_{(x,\,y)} s(x, y)\, g(x, y) \tag{22}$$

при ограничениях

$$\sum_{y} [f(x, y) - f(y, x)] = \begin{cases} 0 & \text{для } x \neq 1, n \\ v & \text{для } x = 1 \\ -v & \text{для } x = n, \end{cases} \tag{23}$$

$$f(x, y) + g(x, y) - h(x, y) = k(x, y) \quad \text{для всех } (x, y), \tag{24}$$

$$f(x, y) \geqslant 0 \quad \text{для всех } (x, y), \tag{25}$$

$$g(x, y) \geqslant 0 \quad \text{для всех } (x, y), \tag{26}$$

$$h(x, y) \geqslant 0 \quad \text{для всех } (x, y). \tag{27}$$

Из $g(x, y) - h(x, y) = k(x, y) - f(x, y)$ и $r(x, y) \leqslant s(x, y)$ следует, что в любом оптимальном решении двойственной задачи

$$g(x, y) = \max\{0, k(x, y) - f(x, y)\}, \tag{28}$$

$$h(x, y) = \max\{0, f, y) - k(x, x)\}. \tag{29}$$

Используя (28)—(29), целевую функцию (22) двойственной задачи можно переписать в виде
минимизировать

$$Tv + \sum [s(x, y) \max\{0, k(x, y) - f(x, y)\}] -$$

$$- \sum [r(x, y) \max\{0, f(x, y) - k(x, y)\}]. \quad (30)$$

Если $f(x, y)$ возрастает от 0 до $k(x, y)$, то целевая функция (30) убывает со скоростью $s(x, y)$; при дальнейшем возрастании $f(x, y)$ целевая функция (30) убывает со скоростью $r(x, y)$. Так как $r(x, y) \leqslant s(x, y)$ и целевая функция (30) минимизируется, возрастание $f(x, y)$ от 0 до $k(x, y)$ оказывает более благоприятный эффект, чем последующее возрастание $f(x, y)$.

Таким образом, мы можем рассматривать операцию (x, y) как дугу графа, обладающую следующим свойством: пропускание по ней первых $k(x, y)$ единиц потока требует затрат в размере $s(x, y)$, а последующее пропускание потока требует затрат $r(x, y)$ на единицу потока.

Заменим теперь каждую дугу (x, y) парой дуг $(x, y)_1$ и $(x, y)_2$. Положим

$$f(x, y) = f(x, y)_1 + f(x, y)_2. \quad (31)$$

На дугах $(x, y)_1$ и $(x, y)_2$ определим, согласно (19) и (20), стоимости и пропускные способности.

Двойственная задача может быть теперь переписана в виде: минимизировать

$$Tv + \sum_{(x, y)i} a(x, y)_i f(x, y)_i \quad (32)$$

при ограничениях

$$\sum_y \sum_{i=1, 2} [f(x, y)_i - f(y, x)_i] = \begin{cases} 0 \text{ для } x \neq 1, n \\ v \text{ для } x = 1 \\ -v \text{ для } x = n, \end{cases} \quad (33)$$

$$0 \leqslant f(x, y)_i \leqslant c(x, y)_i \text{ [для всех } (x, y)_i. \quad (34)$$

Двойственную задачу (32)—(34) можно преобразовать в широко известной форме следующим образом. Введем в граф «обратную» дугу, идущую из вершины n в вершину 1. На этой дуге определим стоимость и пропускную способность в соответствии с (19) и (20), тогда задача (32)—(34) примет вид:

минимизировать

$$\sum_{(x, y)} a(x, y) f(x, y) \quad (35)$$

при ограничениях

$$\sum_y [f(x, y) - f(y, x)] = 0 \text{ для всех } x, \quad (36)$$

$$0 \leqslant f(x, y) \leqslant c(x, y) \text{ для всех } (x, y), \tag{37}$$

где индексы дуг для краткости опущены. Задача линейного программирования (35)—(37) представляет собой не что иное, как задачу о потоке минимальной стоимости (гл. 4, выражения (24)—(26)) для графа G'.

Каждая единица потока должна пройти от вершины 1 до вершины n и возвратиться обратно к вершине 1 по дуге $(n, 1)$. При этом на дуге $(n, 1)$ целевая функция получает положительное приращение $+T$, а на всех остальных дугах, по которым единица потока проходит от вершины 1 к вершине n, — только отрицательные приращения (см. соотношение (19)). Так как целевая функция (35) должна быть минимизирована, единицу потока целесообразно пропускать через граф от вершины 1 к вершине n только в том случае, если при этом суммарное приращение целевой функции не превышает $-T$.

Для решения задачи (35)—(37) может быть применен алгоритм дефекта (см. разд. 4.4) для графа G'. После окончания работы этот алгоритм найдет не только допустимый поток в графе G', удовлетворяющий условиям (36)—(37), но и значения двойственных переменных $p_t(x)$, $x \in X$, удовлетворяющие условиям дополняющей нежесткости:

$$p_t(y) - p_t(x) < a(x, y) \Rightarrow f(x, y) = 0, \tag{38}$$

$$p_t(y) - p_t(x) > a(x, y) \Rightarrow f(x, y) = c(x, y) \tag{39}$$

для всех $(x, y) \in A'$ (см. гл. 4, условия (33)—(34), где $l(x, y) = 0$ для всех (x, y)).

Если для выполнения каждой операции (x, y) будет выделено максимальное возможное время $s(x, y)$, то общее время выполнения проекта превысит T (в противном случае исходная задача тривиальна, так как оптимальным решением будет $t(x, y) = s(x, y)$ для всех (x, y)). Так как $a(x, y)_1 = -s(x, y)$ для всех (x, y), то в оптимальном решении (35)—(37) от вершины 1 к вершине n и обратно по дуге $(n, 1)$ обязательно должен быть пропущен некоторый поток. Следовательно, $f(n, 1) > 0$, и из (38)—(39) следует, что

$$p_t(1) - p_t(n) = a(n, 1) = T. \tag{40}$$

Без ограничения общности можно считать, что $p_t(1) = 0$. Если это не так, то без нарушения справедливости условий (38)—(39) можно вычесть $p_t(1)$ из всех $p_t(x)$. Следовательно, $p_t(n) = -T$.

Поскольку ограничения (36) отличаются от ограничений (25) гл. 4 коэффициентом (-1), оптимальные значения двойственных переменных, выработанные алгоритмом определения дефекта, необходимо умножить на -1. Тогда мы получим оптимальные значения двойственных переменных для задачи (35)—(37).

Очевидно, что оптимальному потоку задачи (35)—(37) соответствует оптимальный поток задачи (22)—27), равный $f(x, y) = f(x, y)_1 + f(x, y)_2$. Чтобы завершить доказательство, мы должны показать, что значения $p(x) = -[p_t(x) - p_t(1)]$ допустимы для исходной задачи (15)—(18), (21). Кроме того, мы должны показать, что значения $p(x)$ удовлетворяют условиям дополняющей нежесткости для пары двойственных задач: прямой (15)—(18), (21) и двойственной (22)—(27). Эти условия имеют вид

$$p(y) - p(x) - t(x, y) > 0 \Rightarrow f(x, y) = 0. \tag{41}$$

$$r(x, y) < t(x, y) \Rightarrow h(x, y) = 0. \tag{42}$$

$$t(x, y) < s(x, y) \Rightarrow g(x, y) = 0. \tag{43}$$

Для доказательства допустимости отметим, что поток по дуге $(x, y)_2$ никогда не может достичь величины пропускной способности, равной ∞. Поэтому

$$p_t(y) - p_t(x) \leqslant a(x, y)_2 = -r(x, y).$$

Следовательно, $p(y) - p(x) \geqslant r(x, y)$.
Таким образом, значения $p(x)$ дают допустимое решение исходной задачи, ибо $t(x, y) = \min\{p(y) - p(x), s(x, y)\}$.

Проверим теперь выполнение условия дополняющей нежесткости (41). Из (13) следует

$$p(y) - p(x) - t(x, y) > 0 \Rightarrow t(x, y) = s(x, y),$$

т. е. $-p_t(y) + p_t(x) + a(x, y)_1 > 0$ и из (38) $f(x, y)_1 = 0$, $f(x, y)_2 = 0$. Таким образом, $f(x, y) = 0$, и условие (41) выполнено.

Для проверки условия (42) отметим, что

$$r(x, y) < t(x, y) \Rightarrow r(x, y) < p(y) - p(x) \Rightarrow -p_t(y) + p_t(x) > r(x, y) = -a(x, y)_2$$

и из (38) следует, что $f(x, y)_2 = 0$. Поэтому $f(x, y) = f(x, y)_1 \leqslant k(x, y)$, и с учетом (27), (29) $h(x, y) = 0$. Условие (42), таким образом, выполнено.

Условие дополняющей нежесткости (43) проверяется аналогично. Теорема доказана.

ПРИМЕР. Используя теорему 1, вычислим оптимальные сроки выполнения операций для сетевого графика G, изображенного на рис. 9.9. Соответствующий граф G' изображен на рис. 9.10. Он содержит все дуги G, повторенные дважды, и, кроме того, обратную дугу.

Оптимальное решение, найденное при помощи алгоритма дефекта, изображено на рис. 9.11. Проверку условий дополняющей нежесткости (38)—(39) для найденных значений потоков и двойственных переменных мы оставляем читателю.

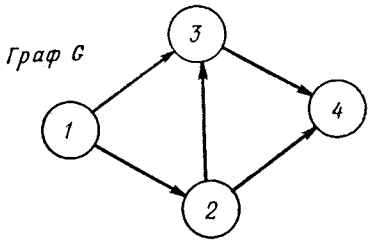


Рис. 9.9. Сетевой график G с переменными длительностями выполнения операций.

Длительности	Стоимости
$2 \leqslant t(1, 2) \leqslant 8$	$k\ (1, 2) = 1$
$5 \leqslant t(1, 3) \leqslant 10$	$k\ (1, 3) = 8$
$1 \leqslant t(2, 3) \leqslant 4$	$k\ (2, 3) = 6$
$4 \leqslant t(2, 4) \leqslant 8$	$k\ (2, 4) = 7$
$6 \leqslant t(3, 4) \leqslant 8$	$k\ (3, 4) = 5$

$T = 15$

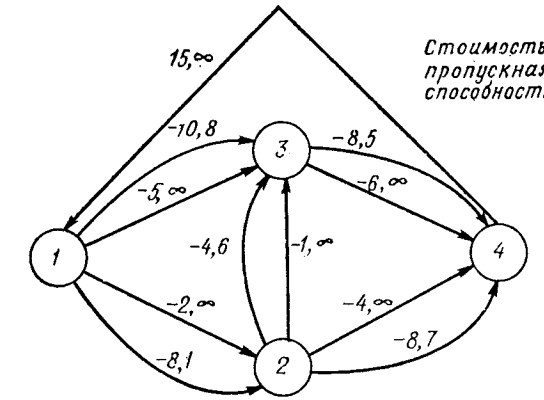


Рис. 9.10. Граф G'.

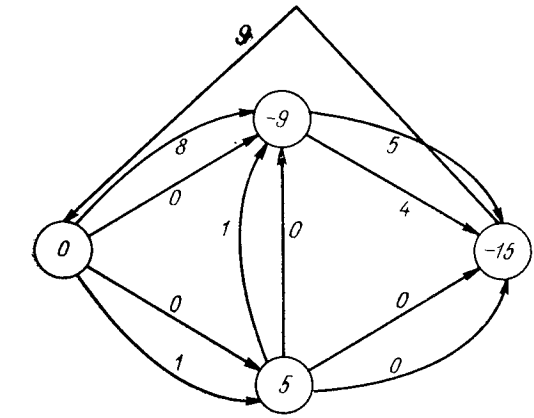


Рис. 9.11. Оптимальное решение, найденное по алгоритму дефекта (показаны потоки в дугах и двойственные переменные, соответствующие вершинам).

Оптимальные моменты наступления событий равны

$$p(1) = -(p_t(1) - p_t(1)) = 0.$$

$$p(2) = -(p_t(2) - p_t(1)) = -(-5-0) = 5.$$

$$p(3) = -(p_t(3) - p_t(1)) = -(-9-0) = 9.$$

$$p(4) = -(p_t(4) - p_t(1)) = -(-15-0) = 15.$$

Оптимальные длительности выполнения операций:

$$t(1, 2) = \min\{s(1, 2),\ p(2) - p(1)\} = \min\{8,\ 5-0\} = 5.$$

$$t(1, 3) = \min\{s(1, 3),\ p(3) - p(1)\} = \min\{10,\ 9-0\} = 9.$$

$$t(2, 3) = \min\{s(2, 3),\ p(3) - p(2)\} = \min\{4,\ 9-5\} = 4.$$

$$t(2, 4) = \min\{s(2, 4),\ p(4) - p(2)\} = \min\{8,\ 15-5\} = 8.$$

$$t(3, 4) = \min\{s(3, 4),\ p(4) - p(3)\} = \min\{8,\ 15-9\} = 6.$$

Заметим, что если происходит изменение стоимостей или пропускных способностей дуг графа (при изменении длительностей или стоимостей в исходной задаче), то в качестве начального решения для алгоритма дефекта можно использовать предыдущее оптимальное решение. Если происшедшие изменения незначительны, то такое начальное решение оказывается обычно более эффективным, чем нулевой начальный поток.

Может, например, возникнуть необходимость закончить проект на два дня раньше, чем предполагалось. В этом случае мы должны уменьшить T на два. Если в качестве начального решения использовать предыдущее оптимальное решение, полученное алгоритмом дефекта, то обратная дуга будет иметь дефект в две единицы, а на всех остальных дугах дефекты будут отсутствовать. Детально этот метод нахождения оптимальных сроков завершения операций освещен в работе [5].

9.3. Обобщенные сетевые графики

До сих пор мы предполагали, что (а) любая операция, следующая за некоторым событием, может быть выполнена при условии, что выполнены все операции, предшествующие этому событию, и (б) должны быть выполнены все операции проекта.

Предположение (а), однако, будет излишним, если, например, изучение студентом хотя бы одной из нескольких дисциплин достаточно для изучения данной дисциплины. Получение хотя бы одного из некоторого числа чеков даст вам возможность начать коммерческую операцию. Аналогично успех в получении хотя бы одной из нескольких субсидий обеспечит финансирование проекта исследований.

Предположение (б) будет излишним, например, для институтской программы, допускающей факультативные курсы, или при

выполнении станочной операции, для которой в зависимости от результатов контроля качества могут понадобиться одна, две или три операции сверления. На основе результатов анализа рынка можно определить, какой политики рекламирования необходимо придерживаться.

Мы видим, таким образом, что многие проекты не могут быть адекватно описаны в терминах ограничений, наложенных на сетевые графики в разд. 9.1. По этой причине были разработаны обобщенные сетевые графики, для которых рассмотренные выше предположения не вводятся. Их детальное описание читатель может найти в работах [1, 2, 6, 7, 8].

В отличие от сетевых графиков, содержащих лишь вершины одного типа, называемые событиями, обобщенные сетевые графики содержат вершины различных типов, которые называются *решающими узлами* (*РУ*). Решающий узел характеризуется условиями, налагаемыми на входящие в него и выходящие из него операции. На операции, входящие в РУ, могут быть наложены три различных условия:

(а) «Вход И»: событие, соответствующее данному РУ, считается происшедшим, если выполнены все входящие в РУ операции.

(б) «Включающий вход»: событие, соответствующее данному РУ, считается происшедшим, если выполнена по крайней мере одна из входящих в РУ операций.

(в) «Исключающий вход»: событие, соответствующее данному РУ, считается происшедшим, если выполнена ровно одна из входящих в РУ операций.

На операции, выходящие из РУ, могут быть наложены два различных условия:

(а) «Детерминированный выход»: после того как произошло событие, соответствующее данному РУ, выполняются все выходящие из него операции.

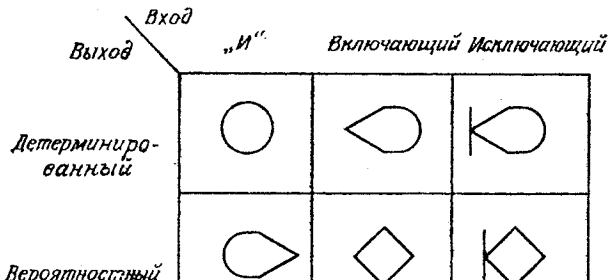


Рис. 9.12. Шесть типов решающих узлов.

(б) «Вероятностный выход»: после того как произошло событие, соответствующее данному РУ, выполняется ровно одна из выходящих из него операций.

Таким образом, имеется $3 \times 2 = 6$ различных типов РУ. Их графические изображения представлены на рис. 9.12.

Для сетевых графиков мы задавали времена $t(x, y)$, требуемые для выполнения каждой операции (x, y). Для обобщенных сетевых графиков должны быть заданы как времена $t(x, y)$, так и вероятности $p(x, y)$ выполнения каждой операции (x, y). Значение $p(x, y)$ есть вероятность того, что после появления события, соответствующего решающему узлу x, будет выполняться операция (x, y). Если этот РУ имеет детерминированный выход, то $p(x, y)$ должна быть равна 1 и операция (x, y) обязательно выполняется. Если же он имеет вероятностный выход, то сумма вероятностей выполнения операций, выходящих из x, не должна превышать единицы.

ПРИМЕР 1. Рассмотрим проект, обобщенный сетевой график которого изображен на рис. 9.13. Решающий узел *1* имеет детерминированный выход,

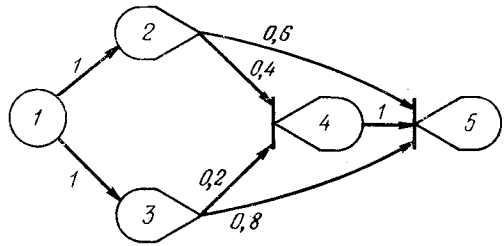


Рис. 9.13. Обобщенный сетевой график (числа на дугах задают вероятности выполнения операций).

следовательно, обе операции (1,2) и (1,3) должны быть выполнены, т. е. $p\ (1,2) = 1$, $p\ (1,3) = 1$. Решающий узел *2* имеет вход И: соответствующее ему событие появится после выполнения операции (1,2). Вслед за этим с вероятностью 0,6 будет выполняться операция (2,5), а с вероятностью 0,4 — операция (2,4), причем должна быть выполнена только одна из этих двух операций. Событие, соответствующее решающему узлу 4, произойдет, только если будут выполнены либо операция (2,4), либо операция (3,4) (но не обе они вместе). Если не будет выполнена ни (2,4), ни (3,4), то это событие не произойдет.

Возможна, кроме того, такая ситуация, когда будет выполнена как операция (2,5), так и операция (3,5). В этом случае событие, соответствующее решающему узлу 5, не произойдет.

ПРИМЕР 2. Журнал просит авторов присылать рукописи для публикации. После получения рукописи она направляется на рассмотрение одновременно двум рецензентам, каждый из которых может дать один из трех различных ответов (отвергнуть, принять или неопределенный ответ) с вероятностями 0,5; 0,4 и 0,1 соответственно.

Если журнал получает хотя бы один отрицательный отзыв, он отвергает рукопись. Если оба рецензента рекомендуют принять рукопись, то журнал принимает ее для публикации. В остальных случаях рукопись направляется третьему рецензенту.

Обобщенный сетевой график такой процедуры показан на рис. 9.14. Дуги, над которыми не надписаны названия, соответствуют фиктивным опе-

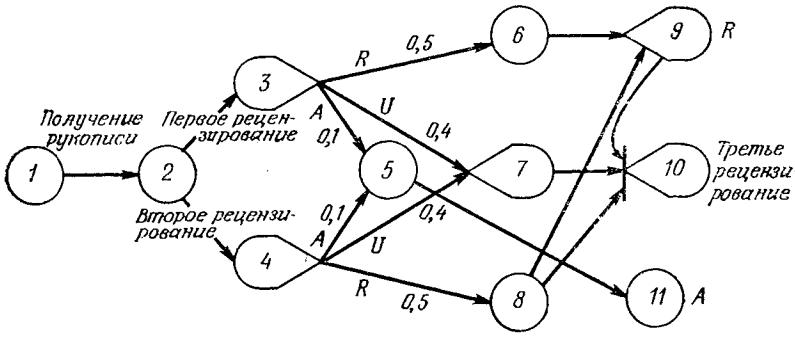


Рис. 9.14. Обобщенный сетевой график процедуры одновременного рецензирования (*R* — «отвергнуть», *A* — «принять», *U* — «неопределенный ответ»).

рациям, необходимым для построения всех возможных ситуаций. Отметим, что на графике представлены все три типа входов и два типа выходов решающих узлов (РУ).

Теперь изменим ситуацию. Предположим, что редактор не посылает рукопись второму рецензенту до тех пор, пока первый рецензент не даст одного из двух ответов: «принять» или «неопределенный». Обобщенный сетевой график для такой ситуации показан на рис. 9.15.

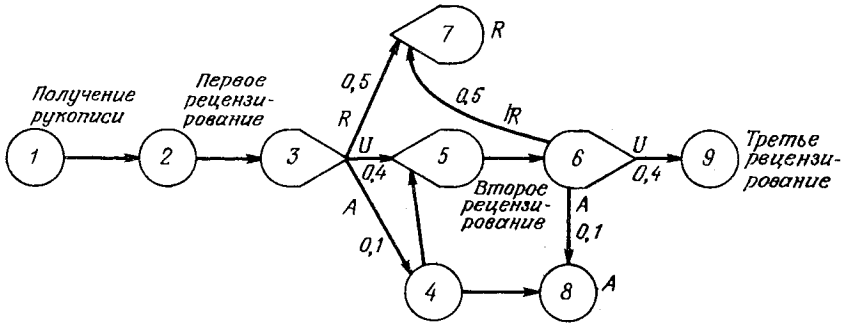


Рис. 9.15. Обобщенный сетевой график процедуры последовательного рецензирования (*R* — «отвергнуть», *A* — «принять», *U* — «неопределенный ответ»).

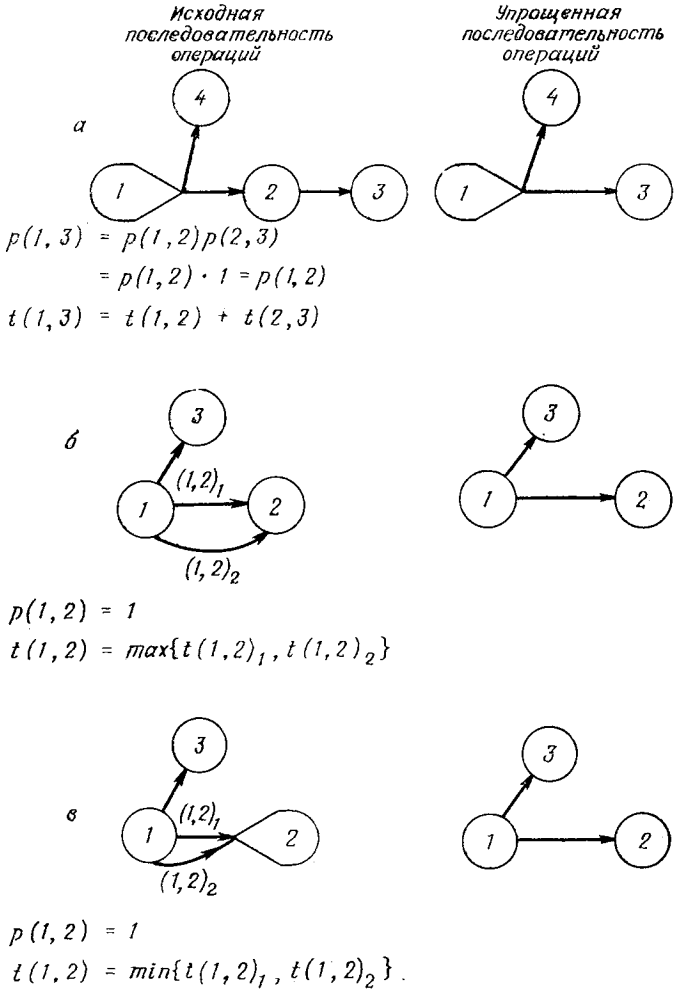


Рис. 9.16. Упрощения обобщенных сетевых графиков.

Все события, изображенные на сетевых графиках, рано или поздно происходят. Для обобщенных сетевых графиков это, вообще говоря, не так: не обязательно все операции должны быть выполнены и, следовательно, не обязательно все события должны появляться. На рис. 9.14, например, изображен проект, который будет закончен только в одном из РУ: 9, 10 или 11.

Может также случиться, что проект будет закончен не в РУ, а после выполнения некоторой операции. Если, например, на

рис. 9.13 будут выполнены операции (2, 4) и (3, 4), то проект будет закончен без наступления события, соответствующего РУ 4. Это, однако, может произойти только для графиков, содержащих РУ с исключающими входами.

Перед тем как перейти к дальнейшему анализу обобщенных сетевых графиков, отметим, что часто возможно упростить график, преобразовав его к эквивалентному графику с меньшим числом операций. Такое упрощение возможно для операций, образующих последовательные или параллельные комбинации. Эти упрощения представлены на рис. 9.16.

Вследствие наличия различных возможностей окончания проекта, представленного обобщенным сетевым графиком, лицу, управляющему проектом, естественно, важно знать вероятности появления событий, соответствующих некоторым РУ, или вероятности фактического выполнения некоторых операций. Более того, часто важно знать математическое ожидание времени появления события, соответствующего данному РУ (в предположении, что событие появится). (Заметим, что для метода PERT времена, требуемые для выполнения операций, были случайными величинами, но мы были уверены, что рано или поздно каждая операция будет выполнена. Для обобщенных сетевых графиков имеет место обратная ситуация: времена, требуемые для выполнения операций, предполагаются неслучайными, но сами операции выбираются для выполнения случайным образом.)

Вычисление этих вероятностей и математических ожиданий наталкивается на трудности, связанные с присутствием РУ с вероятностными выходами. Для такого РУ может быть выполнена только одна из множества выходящих из него операций. Поэтому вероятности выполнения операций и появления событий, следующих за РУ с вероятностным выходом, не являются статистически независимыми, и, следовательно, мы не можем применить традиционные и удобные вероятностные правила вычислений, предполагающие статистическую независимость.

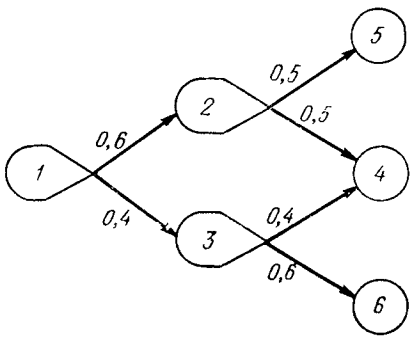


Рис. 9.17. Статистическая зависимость операций.

На рис. 9.17, например, для наступления события, соответствующего РУ 4, должны быть выполнены операции (2, 4) и (3, 4). Вероятность выполнения операции (2, 4) равна $0{,}6 \times 0{,}5 = 0{,}3$, а операции (3, 4) — $0{,}4 \times 0{,}4 = 0{,}16$. Мы могли бы сделать поспешный вывод о том, что вероятность

появления соответствующего РУ 4 события равна $0{,}3 \times 0{,}16$. Однако более пристальное рассмотрение показывает, что операции (2, 4) и (3, 4) не являются статистическими независимыми. Они обе могут быть выполнены только в случае совместного выполнения операции (1, 2) и (1, 3), что невозможно. (Напомним, что для РУ с вероятностным выходом выполняется только одна из выходящих операций.) Таким образом, событие, соответствующее РУ 4, никогда не произойдет.

Отсутствие статистической независимости между вероятностями выполнения операции делает большие графики практически неподдающимися расчету. Даже если мы изменим определение РУ с вероятностным выходом таким образом, чтобы могли выполняться несколько выходящих из него операций (каждая со своей вероятностью), то возникнут те же вычислительные трудности. Например, за данной операцией, выходящей из РУ с вероятностным выходом, могут последовать несколько операций, которые будут статистически зависимы друг от друга, что приведет к той же ситуации.

Положение, кроме того, осложняется тем, что многие проекты порождают обобщенные сетевые графики, содержащие контуры. Проект, например, может содержать операцию, которая должна повторяться до тех пор, пока она не будет выполнена правильно. Примером такой операции может служить изучение необходимого курса институтской программы. Присутствие контуров еще более осложняет вычисления. Некоторые методы преобразования циклов в эквивалентные ациклические конфигурации рассмотрены в работах [2, 7, 8].

УПРАЖНЕНИЯ

1. Ваша организация решила установить новую, более эффективную вычислительную систему. Старая вычислительная система должна быть демонтирована. Переход необходимо выполнить таким образом, чтобы в каждый момент времени по крайней мере одна из систем была работоспособна. Более того, необходимо перевести на язык новой системы все программы библиотеки обучить персонал работе с этой системой.

Необходимо составить подробный сетевой график, описывающий эту операцию.

2. Для сетевого графика, изображенного на рис. 9.18,

а) вычислить наиболее ранние сроки событий;

б) вычислить наиболее поздние сроки событий, при условии что проект должен быть закончен как можно раньше;

в) найти критический путь;

г) вычислить полный резерв времени для каждой операции;

д) вычислить свободный резерв времени для каждой операции;

е) вычислить независимый резерв времени для каждой операции.

3. Вы обнаружили, что время выполнения операции (3,5) (рис. 9.18) должно быть равно 5. Пересчитайте результаты упражнения 2, не повторяя при этом всех вычислений. Насколько может возрасти время выполнения

операции (3,5) в условиях отсутствия изменения наиболее раннего срока окончания всего проекта?

4. На рис. 9.19 три числа, изображенные рядом с каждой дугой, обозначают соответственно оптимистическую, реалистическую и пессимистиче-

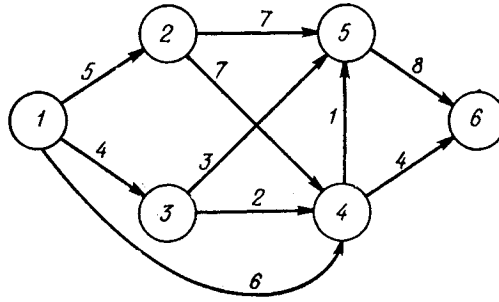


Рис. 9.18. К упражнению 9.2 (числа на дугах задают времена выполнения операций).

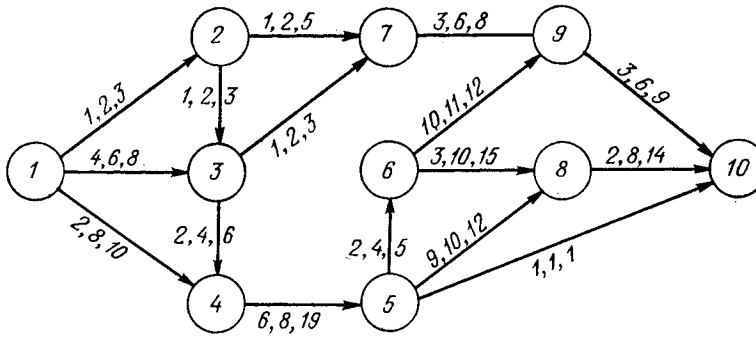


Рис. 9.19. К упражнению 9.4.

скую оценки времени выполнения операции. Используя метод PERT, вычислить:

а) ожидаемое время выполнения каждой операции;

б) дисперсию времени выполнения каждой операции;

в) ожидаемые наиболее ранние сроки наступления событий;

г) ожидаемые наиболее поздние сроки наступления событий, при условии что проект должен быть закончен к ожидаемому наиболее раннему времени окончания;

д) критический путь;

е) дисперсию наиболее раннего времени окончания всего проекта.

5. На рис. 9.20 числа, изображенные рядом с каждой операцией, указывают соответственно минимальное время, необходимое для выполнения операции, максимальное время, необходимое для выполнения операции,

и стоимость уменьшения продолжительности выполнения операции на одну единицу времени (в долл.).

Вычислить оптимальные ресурсы времени, которые должны быть выделены на выполнение каждой операции, при условии что весь проект должен быть закончен за 25 единиц времени.

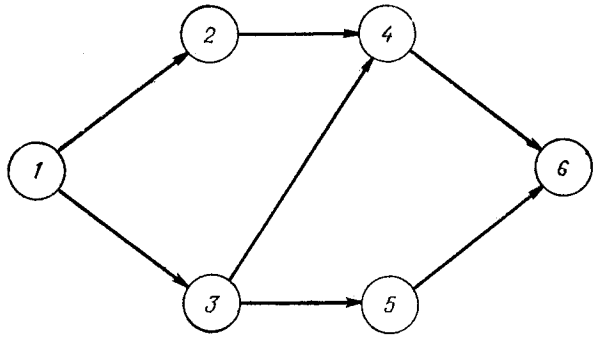


Рис. 9.20. К упражнению 9.5.

1—2 (4, 8, 7); 1—3 (6, 10, 5); 2—4 (3, 5, 6); 3—4 (3, 6, 3); 3—5 (2, 8, 1); 4—6 (9, 15, 6); 5—6 (8, 10, 5).

6. Показать, что алгоритмы расчета наиболее ранних и наиболее поздних сроков наступления событий неприменимы, если сетевой график содержит контуры.

7. В детали автомобиля необходимо просверлить два отверстия. Если оба отверстия имеют заданную глубину, деталь считается законченной. Если глубина первого отверстия оказывается меньше стандартной, второй сверлится с особой точностью в специальном режиме работы сверлильного станка. Если глубина первого отверстия оказывается больше стандартной, рабочий должен проверить, соответствует ли это отверстие всем поршням из некоторого запаса. Если это так, сверление второго отверстия производится в обычном режиме. В противном случае с вероятностью 0,6 контролирующий качество инженер отбракует деталь, а с вероятностью 0,4 примет решение сверлить второе отверстие в специальном режиме.

Постройте для этой процедуры обобщенный сетевой график.

8. Проект состоит из семи операций A, ..., G. Между операциями заданы следующие отношения предшествования:

Операции	Предшествующие операции	Операции	Предшествующие операции
A	—	E	AC
B	A	F	A
C	—	G	ED
D	B		

Постройте сетевой график проекта.

9. Постройте сетевой график, в котором значение независимого резерва для некоторой операции отрицательно.

Интерпретируйте это значение.

ЛИТЕРАТУРА

1. Eisner H., A Generalized Network Approach to the Planning and Scheduling of a Research Project, *ORSA*, **10**, pp. 115 — 125, 1962.
2. Elmaghraby S., An Algebra for the Analysis of Generalized Activity Networks, *Mgmt. Sci.*, **10**, N 3, pp. 494 — 514, 1964.
3. Ford L. R., Fulkerson D. R., Flows in Networks, Princeton Press, Princeton, pp. 151 — 161. [Имеется перевод: Форд Л. Р., Фалкерсон Д. Р. Потоки в сетях. — М.: Мир, 1966, с. 215 — 231.]
4. Kelley Jr., Critical-Path Planning and Scheduling: Mathematical Basis, *ORSA*, **9**, pp. 296 — 320, 1969.
5. Moder J., Phillips C., Project Management with CPM and PERT, Van Nostrand Reihold Company, New York, 2nd, ed., 1970. (Прекрасное всеобъемлющее вводное рассмотрение сетевых графиков.)
6. Pritsker A. B., Modeling and Analysis Using Q-GERT Networks, Halsted Press, New York, 1977.
7. Pritsker A. B., Happ W. W. GERT: Graphical Evaluation Review Technique: Part I.Fundamentals, *J. of Ind. Eng.*, **17**, pp. 267 — 274, 1966.
8. Pritsker A. B., Whitehouse G., GERT: Graphical Evalution Review Technique: Part II, Probabilistic and Industrial Engineering Applications, *J. of Ind Eng.*, **17**, pp. 293 — 301, 1966.

Предметный указатель

Оглавление

Уважаемый читатель!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И-110, ГСП, 1-й Рижский пер., д. 2, издательство «Мир».

Майника Э.

Алгоритмы оптимизации на сетях и графах

Ст. научный редактор А. А. Харитонов
Младший научный редактор Л. С. Сысоева
Художник В. И. Харламов
Художественный редактор Л. Е. Безрученков
Технический редактор Л. П. Бирюкова
Корректор Т. П. Пашковская

ИБ № 2387

Сдано в набор 25.08.80. Подписано к печати 01.04.81. Формат 60×90¹/₁₆. Бумага типографская № 2. Гарнитура обыкнов. Печать высокая. Объем 10,25 бум. л. Усл. печ. л. 20,50. Усл. кр.-отт. 20,74. Уч.-изд. л. 20,54. Изд. № 20/0951. Тираж 10 000 экз. Зак. 736. Цена 1 р. 50 к.

ИЗДАТЕЛЬСТВО «МИР»

Москва, 1-й Рижский пер., 2.

Ярославский полиграфкомбинат Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли.
150014. Ярославль, ул. Свободы, 97.