Определения

- **Разрез графа** — множество рёбер, удаление которых делит граф на два изолированных подграфа, один из которых, в частности, может быть отдельным узлом.
- **Ребро безопасное для мн-ва A** – Пусть T – минимальные остов для G(T ⊆ E) и A ⊆ T, тогда e – безопасное ребро для A, если существует остов T' минимальной длинны, возможно совпадающий с T, такой что A U e ⊆ T'.
- **Эйлеров цикл** — это цикл графа, проходящий через каждое ребро (дугу) графа ровно по одному разу.
- **Эйлеров граф** — граф, содержащий эйлеров цикл.
- Если граф имеет простой цикл, содержащий все вершины графа по одному разу, то такой цикл называется **гамильтоновым циклом**, а граф называется **гамильтоновым графом**.
- **Потоковая сеть** – (s,t,V,E,c), где s – начало (исток), t – конец (сток), а c – это пропускные способности дуг графа, (V,E) – ориентированный граф.
- **Условие неразрывности сети** – грубо говоря, входящий поток в каждую вершину должен быть равен исходящему потоку из этой вершины.
- **Разрез потока** — такая пара множеств вершин (S,T), что
    1. $S \cup T = V$, где V — множество вершин графа
    2. $S \cap T = \emptyset$
    3. $s \in S, t \in T$, где s — исток, t — сток.
- **Величиной разреза** называется сумма пропускных способностей таких рёбер (i, j), что $i \in S, j \in T$.
- **Пропускная способность разреза (A,B)** — сумма пропускных способностей всех рёбер из A в B $\sum_{u \in A} \sum_{v \in B} c(u, v)$.

Поиск в ширину

1. Поместить узел, с которого начинается поиск, в изначально пустую очередь.
2. Извлечь из начала очереди узел u и пометить его как развёрнутый.
    - Если узел u является целевым узлом, то завершить поиск с результатом «успех».
    - В противном случае, в конец очереди добавляются все преемники узла u, которые ещё не развёрнуты и не находятся в очереди.
3. Если очередь пуста, то все узлы связного графа были просмотрены, следовательно, целевой узел недостижим из начального; завершить поиск с результатом «неудача».
4. Вернуться к п. 2.

Поиск в глубину

Пусть задан граф $G = (V, E)$, где V — множество вершин графа, E — множество ребер графа. Предположим, что в начальный момент времени все вершины графа окрашены в *белый* цвет. Выполним следующие действия:

1. Пройдём по всем вершинам $v \in V$.
   - Если вершина v белая, выполним для неё `DFS(v)`.

Процедура DFS (параметр — вершина $u \in V$)

1. Перекрашиваем вершину u в *серый* цвет.
2. Для всякой вершины w, смежной с вершиной u и окрашенной в белый цвет, рекурсивновыполняем процедуру `DFS(w)` [2].
3. Перекрашиваем вершину u в *чёрный* цвет.

Часто используют двухцветные метки — без серого, на 1-м шаге красят сразу в чёрный цвет.

Метод Краскала (Крускала)

Нужен для построения минимального остовного дерева взвешенного связного неориентированного графа.

Вначале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса. Подробное описание алгоритма приведено в книге

Алгоритм Прима.

1 $U := \{a\}$, где a – произвольная вершина графа;
2 $F := \varnothing$;
3 **while** $U \neq V$ **do**
4 найти ребро наименьшего веса (x, y) среди всех ребер, у которых $x \in U$, $y \in \overline{U}$;
5 добавить к *F* ребро (x, y), а к *U* вершину *y*.

Теорема об алгоритме Прима. *Подграф* (U, F), *построенный с помощью алгоритма Прима, является каркасом наименьшего веса для данного графа.*

Теорема о числе различных каркасов

В любом связанном графе существует <= n^{n-2} различных каркасов.

Оценка достижима для полного графа, а для неполного является верхней.

Теорема об эйлеровом графе

Граф – эйлеров тогда и только тогда, когда он связанный и каждая вершина имеет чётную степень.

Алгоритм Флёри

(поиска эйлерова пути в графе)

Пусть задан граф G=(V,E). Начинаем с некоторой вершины p\in V и каждый раз вычеркиваем пройденное ребро. Не проходим по ребру, если удаление этого ребра приводит к разбиению графа на две связные компоненты (не считая изолированных вершин), т.е. необходимо проверять, является ли ребро мостом или нет.

Достаточный критерий Гамильтоновости графа

Если для любой пары несмежных вершин графа, их сумма будет больше либо равна N, где N – количество вершин графа, то граф – Гамильтонов.

Теорема Дирака:

Пусть G — неориентированный граф и δ — минимальная степень его вершин. Если $n \geqslant 3$ и $\delta \geqslant n/2$, то G — гамильтонов граф.

Теорема Поша:

Пусть граф G имеет $n \geqslant 3$ вершин и выполнены следующие два условия:

- $\forall\ k,\ 1 \leqslant k < (n-1)/2$, число вершин со степенями, не превосходящими k, меньше чем k
- для нечетного n число вершин степени $(n-1)/2$ не превосходит $(n-1)/2$

тогда G — гамильтонов граф.

Алгоритм Дейкстры:

Каждой вершине из *V* сопоставим метку — минимальное известное расстояние от этой вершины до *a*. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Инициализация. Метка самой вершины *a* полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от *a* до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.

Шаг алгоритма. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина *u*, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых *u* является предпоследним пунктом. Вершины, в которые ведут рёбра из *u*, назовём *соседями* этой вершины. Для каждого соседа вершины *u*, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки *u* и длины ребра, соединяющего *u* с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину *u* как посещённую и повторим шаг алгоритма.

Алгоритм Беллмана–Форда

Алгоритм поиска кратчайшего пути во взвешенном графе.

Решим поставленную задачу на графе, в котором заведомо нет отрицательных циклов.

Для нахождения кратчайших путей от одной вершины до всех остальных, воспользуемся методом динамического программирования. Построим матрицу A_{ij}, элементы которой будут обозначать следующее: A_{ij} — это **длина кратчайшего пути из *s* в *i***, содержащего не более *j* рёбер.

Путь, содержащий 0 рёбер, существует только до вершины *s*. Таким образом, A_{i0} равно 0 при *i* = *s*, и +∞ в противном случае.

Теперь рассмотрим все пути из *s* в *i*, содержащие ровно *j* рёбер. Каждый такой путь есть путь из $j-1$ ребра, к которому добавлено последнее ребро. Если про пути длины $j-1$ все данные уже подсчитаны, то определить *j*-й столбец матрицы не составляет труда.

Так выглядит алгоритм поиска длин кратчайших путей в графе без отрицательных циклов:

```
for v ∈ V
  do d[v] ← +∞
d[s] ← 0
for i ← 1 to |V| − 1
  do for (u, v) ∈ E
    if d[v] > d[u] + w(u, v)
      then d[v] ← d[u] + w(u, v)
return d
```

Здесь *V* — множество вершин графа *G*, *E* — множество его рёбер, а *w* — весовая функция, заданная на ребрах графа (возвращает длину дуги, ведущей из вершины *u* в *v*), d - массив, содержащий расстояния от вершины s до любой другой вершины.

Алгоритм Беллмана–Форда позволяет очень просто определить, существует ли в графе *G* отрицательный цикл, достижимый из вершины *s*. Достаточно произвести внешнюю итерацию цикла не $|V|-1$, а ровно |*V*| раз. Если при исполнении последней итерации длина кратчайшего пути до какой-либо вершины строго уменьшилась, то в графе есть отрицательный цикл, достижимый из *s*. На основе этого можно предложить следующую оптимизацию: отслеживать изменения в графе и, как только они закончатся, сделать выход из цикла (дальнейшие итерации будут бессмысленны).

Можно выполнить каждый шаг за время O(*E*), где *E* — число рёбер в графе, тогда общее время работы алгоритма ограничено O(*E***f*).

Алгоритм Форда — Фалкерсона

решает задачу нахождения максимального потока в транспортной сети.

1. Обнуляем все потоки. Остаточная сеть изначально совпадает с исходной сетью.
2. В остаточной сети находим любой путь из источника в сток. Если такого пути нет, останавливаемся.
3. Пускаем через найденный путь (он называется ***увеличивающим путём*** или ***увеличивающей цепью***) максимально возможный поток:
    1. На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью $c_{\min}$.
    2. Для каждого ребра на найденном пути увеличиваем поток на $c_{\min}$, а в противоположном ему — уменьшаем на $c_{\min}$.
    3. Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.
4. Возвращаемся на шаг 2.

Эдмондса — Карпа

решает задачу нахождения максимального потока в транспортной сети. Алгоритм представляет собой частный случай метода Форда — Фалкерсона и работает за время $O(VE^2)$.

1. Обнуляем все потоки. Остаточная сеть изначально совпадает с исходной сетью.
2. В остаточной сети находим *кратчайший* путь из источника в сток. Если такого пути нет, останавливаемся.
3. Пускаем через найденный путь (он называется **увеличивающим путём** или **увеличивающей цепью**) максимально возможный поток:
    1. На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью $c_{\min}$.
    2. Для каждого ребра на найденном пути увеличиваем поток на $c_{\min}$, а в противоположном ему — уменьшаем на $c_{\min}$.
    3. Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.
4. Возвращаемся на шаг 2.

Чтобы найти кратчайший путь в графе, используем поиск в ширину.

Эти алгоритмы сходятся для любых вещественных весов за время $O(|V||E|^2)$ и $O(|V|^2|E|)$ соответственно.