Как и в случае поиска в глубину, процедуру WS можно использовать без всяких модификаций и тогда, когда списки инцидентности ЗАПИСЬ$[v], v \in V$, определяют некоторый ориентированный граф. Очевидно, что тогда посещаются только те вершины, до которых существует путь от вершины, с которой мы начинаем поиск (см. задачу 2.12).

2.4 Стягивающие деревья (каркасы)

Деревом называется произвольный неориентированный связный граф без циклов. Для произвольного связного неориентированного графа $G = \langle V, E\rangle$ каждое дерево $\langle V, T\rangle$, где $T \subseteq E$, будем называть *стягивающим деревом*[2] графа G. Ребра такого дерева будем называть *ветвями*, а все остальные ребра графа будем называть *хордами* (очевидно, что о ветвях и хордах в графе мы можем говорить только в контексте фиксированного стягивающего дерева).

Отметим, что каждое дерево с n вершинами имеет $n-1$ ребер. Этот факт можно просто доказать, применяя индукцию относительно n. Очевидно, что это справедливо для $n = 1$. Если $n > 1$, то отметим сначала, что в каждом дереве с n вершинами существует «висячая» вершина, т.е. вершина степени 1. Действительно, рассмотрим в таком дереве произвольный путь $v_1 - v_2 - \ldots - v_k$ с максимальной длиной ($v_i \neq v_j$ для $i \neq j$). Вершины v_i, v_k «висячие», так как ни от одной из них нельзя провести второго ребра ни к одной из вершин $v_1, \ldots, v_n$ (дерево не содержит циклов), ни к любой другой вершине (путь максимальный). Удаляя из нашего дерева вершину v_k и ребро $\{v_{k-1}, v_k\}$ (либо вершину v_1 и ребро $\{v_1, v_2\}$), получаем, очевидно, дерево с $n-1$ вершинами, которое в силу индуктивного предположения имеет $n-2$ ребер. Следовательно, наше первоначальное дерево имело $n-2+1 = n-1$ ребер.

Процедуры поиска в глубину и в ширину можно простым способом использовать для нахождения стягивающих деревьев. В обоих случаях достижение новой вершины u из вершины v вызывает включение в дерево ребра $\{u, v\}$.

Алгоритм 2.3. (Нахождение стягивающего дерева связного графа методом поиска в глубину.)

Данные: Связный граф $G = \langle V, E\rangle$, заданный списками инцидентности ЗАПИСЬ$[v], v \in V$.

Результаты: Стягивающее дерево $\langle V, T\rangle$ графа G.

```
1 procedure WGD(v);
(*поиск в глубину, соединенный с нахождением ребра дерева;
переменные НОВЫЙ, ЗАПИСЬ, T — глобальные*)
```

[2] В литературе по теории графов широко распространены также термины *каркас* и *остов* — *Прим. перев.*

```
2 begin НОВЫЙ[v]:=ложь;
3       for u ∈ ЗАПИСЬ[v] do
4           if НОВЫЙ[u] then (*{v,u} — новая ветвь*)
5             begin T := T ∪ {v,u}; WGD(u)
6             end
7 end; (* WGD*)
8 begin (* главная программа*)
9       for u ∈ V do НОВЫЙ[u]:=истина; (*инициализация*)
10      T := ∅; (*T=множество найденных к этому моменту ветвей*)
11      WGD(r) (*r — произвольная вершина графа*)
12 end
```

Для доказательства того, что алгоритм 2.3 корректно строит стягивающее дерево произвольного связного графа, достаточно отметить следующие три факта. Во-первых, в момент добавления к множеству T новой ветви $\{v,u\}$ (строка 5) в $\langle V,E\rangle$ существует путь из r в v (этот факт мы доказываем по индукции). Таким образом, алгоритм строит связный граф. Во-вторых, каждая новая ветвь $\{u,v\}$, добавляемая к множеству T, соединяет уже рассмотренную вершину v (т.е. НОВЫЙ[v]=**ложь**) с новой вершиной u. Отсюда следует, что построенный граф $\langle V,T\rangle$ не содержит циклов. Действительно, последняя ветвь, «замыкающая» цикл, должна была бы соединить две уже рассмотренные вершины. И наконец, в-третьих, из свойства поиска в глубину следует, что процедура WGD просматривает все вершины связного графа. Следовательно, граф $\langle V,T\rangle$, построенный нашим алгоритмом, есть стягивающее дерево графа G. Вычислительная сложность алгоритма есть, очевидно, $O(n+m)$, т.е., того же порядка, что и поиск в глубину. Пример стягивающего дерева, построенного алгоритмом 2.3, приведен на рис. 2.7, а. Каждое стягивающее дерево, построенное указанным алгоритмом, имеет любопытное свойство, которое мы сейчас опишем. Вершину r, с которой мы начинаем поиск в графе, назовем ***корнем*** стягивающего дерева $\langle V,T\rangle$. Очевидно, что в дереве $\langle V,T\rangle$ существует в точности один путь от произвольной вершины к корню (см. задачу 10).

Для двух различных вершин v и u дерева $\langle V,T\rangle$ будем говорить, что u является ***потомком*** вершины v, если v лежит на пути (в дереве $\langle V,T\rangle$) из u в корень. Если при этом имеет место $v-u$, то будем говорить, что u — ***сын*** вершины v, а v — ***отец*** вершины u.

Теорема 2.4. Пусть $\langle V,T\rangle$ — стягивающее дерево связного графа $G=\langle V,E\rangle$, построенное алгоритмом 2.3, и пусть $\{u,v\}\in E$. Тогда либо u — потомок v, либо v - потомок u.

Доказательство. Предположим без ограничения общности, что вершина v будет просмотрена раньше, чем u. Рассмотрим процесс поиска в глубину, начиная с вершины v. Очевидно, что по окончании его должно быть НОВЫЙ[u]=**ложь**,

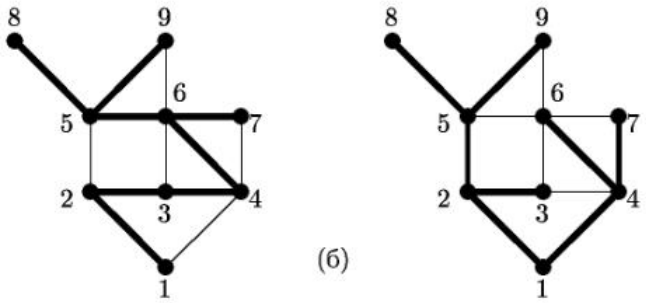

Рис. 2.7: Стягивающие деревья, построенные с помощью алгоритма 2.3 (а) и алгоритма 2.5 (б).

ибо $v - u$. Но это означает, что ребра, добавленные в множество T в течение этого процесса, содержат путь из v в u, откуда следует, что v лежит на пути из u в корень.

Подобным же способом можно построить стягивающее дерево, используя метод поиска в ширину.

Алгоритм 2.5. (Нахождение стягивающего дерева связного графа методом поиска в ширину.)

Данные: Связный граф $G = \langle V, E\rangle$, представленный списками инцидентности ЗАПИСЬ$[v], v \in V$.

Результаты: Стягивающее дерево $\langle V, T\rangle$ графа G.

```
1 begin
2       for u ∈ V do НОВЫЙ[u]:=истина; (*инициализация*)
3       T := ∅; (*T — множество найденных к этому моменту ветвей*)
4       ОЧЕРЕДЬ := ∅; ОЧЕРЕДЬ ⇐ r;
5       НОВЫЙ[r]:=ложь; (*корень стягивающего дерева*)
6       while ОЧЕРЕДЬ ≠ ∅ do
7             begin v ⇐ ОЧЕРЕДЬ;
8                   for u ∈ ЗАПИСЬ[v] do
9                       if НОВЫЙ[u] then (*{v,u} = новая ветвь*)
10                        begin ОЧЕРЕДЬ ⇐ u; НОВЫЙ[u]:=ложь;
                                T := T ∪ {v,u}
11                        end
12            end
13 end
```

Способом, аналогичным использованному для алгоритма 2.3, можно доказать, что данный алгоритм корректно строит стягивающее дерево для произвольного связнего графа за $O(m+n)$ шагов.

На рис. 2.7, б дан пример стягивающего дерева, построенного алгоритмом 2.5.

Рассуждения разд. 2.3 приводят непосредственно к следующей теореме:

Теорема 2.6. Пусть $\langle V, T\rangle$ — стягивающее дерево связного графа $G = \langle V, E\rangle$, построенное при помощи алгоритма 2.5. Тогда путь в $\langle V, T\rangle$ из произвольной вершины u до корня r является кратчайшим путем из u в r в графе G.

В гл. 3 особо обсуждается более общая задача отыскания кратчайших путей в графе, ребрам которого приписаны «длины», не обязательно равные единице.

Рассуждения данного раздела легко переносятся на произвольные графы, не обязательно связные. Максимальный подграф без циклов произвольного графа G называется *стягивающим лесом* графа G. Очевидно, что стягивающий лес графа с k компонентами связности определяется через стягивающие деревья этих компонент и, следовательно, содержит $n-k-1$ ребер.

2.5 Отыскание фундаментального множества циклов в графе

Тесно связана с задачей нахождения стягивающего дерева задача построения фундаментального множества циклов. Если к стягивающему дереву $\langle V, T\rangle$ графа $G = \langle V, E\rangle$ мы добавим произвольную хорду $e \in E\backslash T$, то нетрудно отметить, что возникший при этом подграф $\langle V, T \cup \{e\}\rangle$ содержит в точности один цикл[3], который мы будем обозначать через C_e. Очевидно, что C_e содержит ребро e. Множество $\mathcal{C} = \{C_e : e \in E\backslash T\}$ будем называть *фундаментальным множеством циклов* графа G (относительно стягивающего дерева $\langle V, T\rangle$). Название «фундаментальный» связано с тем фактом, что, как мы докажем позднее, каждый цикл графа G можно некоторым естественным способом получить из циклов множества $\mathcal{C}$.

Введем для произвольных множеств A и B операцию

$$A \oplus B = (A \cup B)\backslash(A \cap B).$$

Множество $A \oplus B$ будем называть *симметрической разностью* множеств A и B.

Отметим, что симметрическая разность множеств $A_1, \ldots, A_k$ содержит независимо от расстановки скобок в точности те элементы, которые появляются в

[3] В этом разделе под циклом мы всегда будем понимать элементарный цикл

нечетном числе множеств A_i. Действительно, это нетрудно доказать индукцией по k. Для $k = 1$ и $k = 2$ это, очевидно, так. Для $k > 2$ наша симметрическая разность имеет вид $A \oplus B$, где A является симметрической разностью множеств $A_1, \ldots, A_p$ и B — симметрической разностью множеств $A_{p+1}, \ldots, A_{p+q}$, где $p, q < k$ и $p+q = k$. Множество $A \oplus B$ содержит в точности те элементы, которые появляются только в одном из множеств A или B. Пользуясь индуктивным предположением, сделаем вывод, что $A \oplus B$ является множеством тех элементов, которые появляются в нечетном числе множеств из $A_1, \ldots, A_p$ и в четном числе из $A_{p+1}, \ldots, A_{p+q}$, или же в нечетном числе множеств из $A_{p+1}, \ldots, A_{p+q}$ и в четном числе множеств из $A_1, \ldots, A_p$. А это в точности множество тех элементов, которые появляются в нечетном числе множеств из $A_1, \ldots, A_k$.

Из доказанного выше свойства следует, что мы можем опустить скобки в симметрической разности произвольного числа множеств и писать $A_1 \oplus A_2 \oplus \ldots \oplus A_k$. Множество C ребер графа называется *псевдоциклом*, если каждая вершина графа $\langle V, C \rangle$ имеет четную степень. Примером псевдоцикла является пустое множество и произвольный цикл графа.

Лемма 2.7. *Симметрическая разность произвольного числа псевдоциклов является псевдоциклом.*

Доказательство. Очевидно, что достаточно рассмотреть случай двух псевдоциклов C_1 и C_2. Обозначим для произвольной вершины v через $S_1(v)$ и $S_2(v)$ множество ребер соответственно из C_1 и C_2, инцидентных с v. Множества $S_1(v)$ и $S_2(v)$ имеют четную мощность, четной является также мощность множества ребер из $C_1 \oplus C_2$, инцидентных с v, так как $|S_1(v) \oplus S_2(v)| = |S_1(v)| + |S_2(v)| - 2|S_1(v) \cap S_2(v)|$

Теперь мы можем доказать теорему о фундаментальном множестве циклов.

Теорема 2.8. *Пусть $G = \langle V, E \rangle$ — связный неориентированный граф, а $\langle V, T \rangle$ — его стягивающее дерево. Произвольный цикл графа G можно однозначно представить как симметрическую разность некоторого числа фундаментальных циклов. В общем случае произвольный псевдоцикл C графа G можно однозначно выразить как*

$$C = \bigoplus_{e \in C \setminus T} C_e. \tag{2.1}$$

Доказательство. Симметрическая разность $\bigoplus_{e \in C \setminus T} C_e$, являющаяся псевдоциклом в силу предыдущей леммы, состоит из множества хорд $C \setminus T$ и некоторых ветвей. Это вытекает из того факта, что каждая хорда $e' \in C \setminus T$ принадлежит в точности к одному фундаментальному циклу, а именно к C_e. Множество

$$C \oplus \bigoplus_{e \in C \setminus T} C_e. \tag{2.2}$$

является также в силу предыдущей леммы псевдоциклом, причем он может содержать только ветви. Однако ни один непустой псевдоцикл не может содержаться в T, так как каждый непустой подграф без циклов содержит вершину степени 1 («висячую»). Отсюда следует, что множество (2.2) пустое, что эквивалентно равенству (2.1). Однозначность представления (2.1), легко следует из того, что цикл C_e является единственным фундаментальным циклом, содержащим хорду e ($e \in E \backslash T$)

Теорема, которую мы доказали, имеет очень простую интерпретацию в терминах линейных пространств. Пусть $E = \{e_1, \ldots, e_m\}$ и каждому псевдоциклу C поставлен в соответствие вектор $\langle a_1, \ldots, a_n \rangle$, где

$$a_i = \begin{cases} 1, & \text{если } e_i \in C, \\ 0 & \text{в противном случае} \end{cases}$$

Сумме таких векторов в предположении, что суммы координат берутся по модулю 2, соответствует симметрическая разность некоторых псевдоциклов. Лемма 2.7 утверждает, что векторы, поставленные в соответствие псевдоциклам, образуют подпространство m-мерного линейного пространства над двухэлементным полем (состоящим из 0 и 1 с операциями сложения по модулю 2 и умножения). Отметим, что произвольная линейная комбинация в пространстве над двухэлементным полем соответствует симметрической разности, так как единственным ненулевым элементом (коэффициентом) может быть только 1. В свою очередь теорема 2.8 говорит, что фундаментальные циклы определяют базис нашего подпространства.

Нахождение фундаментального множества циклов имеет существенное значение при анализе электрических цепей. Точнее говоря, каждому фундаментальному циклу в графе, соответствующему данной электрической цепи, мы можем сопоставить уравнение, определяющее *закон Кирхгофа для напряжений* (см., например, [61]): сумма падения напряжений вдоль цикла равна нулю. Тогда ни одно из этих уравнений не зависит от остальных, от них же зависит произвольное уравнение, определяющее закон Кирхгофа для произвольного цикла графа.

Опишем теперь простой алгоритм нахождения множества фундаментальных циклов. Этот алгоритм основывается на поиске в глубину и имеет структуру, аналогичную рекурсивному алгоритму нахождения стягивающего дерева (алгоритм 2.3). Каждая новая вершина, встречающаяся в процессе поиска, помещается в стек, представленный таблицей СТЕК, и удаляется из стека после использования. Очевидно, что стек всегда содержит последовательность вершин с рассматриваемой в данный момент вершины v до корня. Поэтому же если анализируемое нами ребро $\{v, u\}$ замыкает цикл (т.е. $WGN[v] > WGN[u] > 0$ и u не находится непосредственно под верхним элементом стека), то вершина u —

также в силу теоремы 2.4 — находится в стеке и цикл, замыкаемый ребром $\{v, u\}$ представлен верхней группой элементов стека, начиная с v и кончая вершиной u.

Алгоритм 2.9. (Нахождение множества элементарных циклов графа.)

Данные: Граф $G = \langle V, E\rangle$, представленный списками инцидентности ЗАПИСЬ$[v]$, $v \in V$.

Результаты: Множество элементарных циклов графа G.

```
1. procedure ЦИКЛ(u);
(* нахождение фундаментального множества циклов
для компоненты связности, содержащей вершину u,
переменные d, num, СТЕК, ЗАПИСЬ, WGN — глобальные*)
2. begin d := d + 1; CTEK[d] := v; num := num + 1; WGN[v] := num;
3.         for u ∈ ЗАПИСЬ[v] do
4.             if WGN[u] = 0 then ЦИКЛ(u)
5.             else if (u ≠ СТЕК[d − 1]) and (WGN[v] > WGN[u]) then
                       (* {v, u} замыкает новый цикл *)
6.                     выписать цикл с вершинами
7.                     СТЕК[d], СТЕК[d − 1], ..., СТЕК[c], где СТЕК[c] = u;
8.         d := d − 1 (*использованная вершина v удаляется из стека *)
9. end; (*ЦИКЛ*)
10. begin (* главная программа*)
11.         for v ∈ V do WGN[v] := 0; num := 0; (*инициализация*)
12.             d := 0; СТЕК[0] := 0; (*d=число элементов в стеке*)
13.         for v ∈ V do
14.             if WGN[v] = 0 then ЦИКЛ(u)
15. end
```

Оценим теперь вычислительную сложность этого алгоритма. Отметим сначала, что общее число шагов, не считая выписывания циклов (строки 6, 7) — как и во всех алгоритмах, основанных на поиске в глубину, — имеет порядок $O(n+m)$. К этому следует прибавить суммарную длину всех циклов. Эта длина не превосходит $(m - n + 1)n$, что дает общую сложность алгоритма $O(nm + n)$ (или $O(nm)$, если отбросить вырожденный случай $m = 0$).

2.6 Нахождение компонент двусвязности

Вершину a неориентированного графа $G = \langle V, E\rangle$ будем называть *точкой сочленения*, если удаление этой вершины и всех инцидентных ей ребер ведет к увеличению числа компонент связности графа. Равнозначное утверждение состоит в том, что a является точкой сочленения, если существуют вершины v

(а) б)

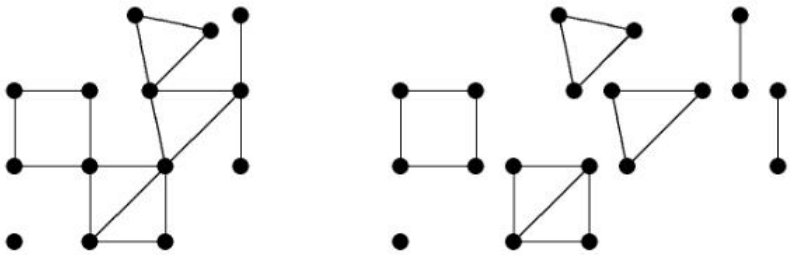

Рис. 2.8: а) Граф с точками сочленения; б) Блоки этого графа

и u, отличные от a, такие, что каждый путь из u в v, а мы предполагаем, что существует по крайней мере один такой путь, проходит через вершину a. Неориентированный граф называется *двусвязным*, если он связный и не содержит точек сочленения. Произвольный максимальный двусвязный подграф графа G называется *компонентой двусвязности* или *блоком* этого графа.

Двусвязность графа — очень желательный признак для некоторых приложений. Представим себе, что вершины графа изображают узлы некоторой информационной сети, а ребра соответствуют линиям передачи. Если наш граф двусвязный, то выход из строя отдельного узла w никогда не приведет к потере соединения между любыми двумя узлами, отличными от w. Знание блоков графа также очень важно, если принять во внимание то, что многие графовые задачи, такие как нахождение всех элементарных циклов или установление факта планарности графа (граф называется планарным, если его можно так начертить на плоскости, чтобы никакие два ребра не пересекались), приводят естественным путем к аналогичным задачам для блоков данного графа.

Отметим, что если $\langle V_1, B_1\rangle$, $\langle V_2, B_2\rangle$ — два разных блока графа G, то $V_1 \cap V_2 = \varnothing$ или $V_1 \cap V_2 = \{a\}$, где a — точка сочленения графа G. Действительно, рассмотрим подграф $\langle V_1 \cup V_2, B_1 \cup B_2\rangle$. Если было бы $|V_1 \cap V_2| \geq 2$, этот подграф был бы двусвязным вопреки предположению о максимальности $\langle V_1, B_1\rangle$ и $\langle V_2, B_2\rangle$. Невозможен также случай $V_1 \cap V_2 = \{a\}$, где a не является точкой сочленения графа G. Ибо тогда в G существует путь между вершинами $u \in V_1$, $u \neq a$ и $u \in V_2$, $v \neq a$, не включающий вершину a. Нетрудно отметить, что подграф,

возникающий из $\langle V_1 \cup V_2, B_1 \cup B_2\rangle$ добавлением вершин и ребер этого пути, будет двусвязным, что противоречит предположению о максимальности $\langle V_1, B_1\rangle$ и $\langle V_2, B_2\rangle$. На рис. 2.8 дан пример графа, имеющего точки сочленения и блоки.

Нахождение точек сочленения и блоков графа является классической задачей, которую можно эффективно решить погружением в процедуру поиска в глубину (см. Тарьян [66]). Прежде чем представить частный алгоритм, сделаем предварительно несколько замечаний.

Теорема 2.10. *Пусть $D = \langle V, T\rangle$ — стягивающее дерево с корнем r связного графа $G = \langle V, E\rangle$, построенное с помощью поиска в глубину (алгоритм 2.3). Вершина $v \in V$ является точкой сочленения графа G тогда и только тогда, когда либо $v = r$ и r имеет по крайней мере двух сыновей в D, либо $v \neq r$ и существует сын w вершины u, такой что ни w, ни какой-либо его потомок не связаны ребром ни с одним предком вершины v.*

Доказательство. Рассмотрим сначала случай $v = r$. Если вершина v имеет только одного сына (либо является изолированной вершиной), то ее устранение не увеличит числа компонент связности, ибо это не нарушает связности дерева. Если она имеет по меньшей мере двух сыновей, то после удаления вершины v эти сыновья лежат в разных компонентах связности. Действительно, из теоремы 2.4 вытекает, что каждый путь между двумя различными сыновьями должен проходить через корень — в противном случае он содержал бы хорду $\{u, t\}$, где ни u не является потомком t, ни t не является потомком u. Аналогично, для случая $v \neq r$. Если после устранения вершины v существует путь от ее сына w до корня, то, как и в предыдущем случае из теоремы 2.4, следует, что этот путь должен содержать ребро, соединяющее w или некоторого потомка вершины w с некоторым предком вершины u.

Идея алгоритма следующая: ведем поиск в глубину в графе, начиная с некоторой вершины r, вычисляя для каждой вершины u два параметра: $WGN[v]$ и $L[v]$. Первый из них — это просто номер вершины v в порядке, в котором вершины посещаются при поиске в глубину, начиная с вершины r. Если через $D = \langle V, T\rangle$ мы обозначим дерево, соответствующее нашему поиску в глубину, то другой параметр определяет наименьшее значение $WGN[u]$, где $u = v$ или вершина u связана хордой с вершиной v или ее произвольным потомком в D. Параметр $L[v]$ легко вычислить по индукции относительно дерева D, если мы знаем $L[w]$ для всех сыновей w вершины v. Обозначив

$$A = \min\{L[w] : w \text{ — сын вершины } v\},$$
$$B = \min\{WGN[u] : \{u, v\} \in E \backslash T\},$$

имеем $L[v] = \min\{WGN[v], A, B\}$.

Из теоремы 2.10 следует, что v является точкой сочленения или корнем тогда и только тогда, когда $L[w] \geq WGN[v]$ для некоторого сына u вершины v

(полагаем, что $n > 1$).

Алгоритм 2.11. (Нахождение компонент двусвязности графа).

Данные: Граф $G = \langle V, E\rangle$ без изолированных вершин, представленный списками инцидентности ЗАПИСЬ$[v]$, $v \in V$.

Результаты: Множества ребер всех компонент двусвязности.

```
1. procedure ДВУСВ(v,p);
(* поиск в глубину, начиная с вершины v и полагая, что p
является отцом вершины u в этом процессе;
параллельно выписываются ребра найденных компонент двусвязности;
переменные num, L, WGN, СТЕК — глобальные*)
2. begin num := num + 1; WGN[v] := num;
3.    L[v] := WGN[v];
4.    for u ∈ ЗАПИСЬ[v] do
5.       if WGN[u] = 0 then (*вершина u — новая, {v,u} — ветвь*)
6.          begin СТЕК ⇐ {v,u}; ДВУСВ(u,v);
7.             L[v] := min(L[v], L[u]);
8.             if L[u] ≥ WGN[v] then (*v есть корень или точка сочленения,
а верхняя часть стека до {v,u} включительно содержит
компоненту двусвязности*)
9.                begin (*выписать ребра компоненты двусвязности*)
10.                  repeat e ⇐ СТЕК; write(e)
11.                  until e = {v,u};
12.                  write(';')(*знак конца компоненты двусвязности*)
13.               end
14.         end
15.      else (*WGN[u]>0, т.е. u уже была посещена*)
16.         if (u ≠ p) and (WGN[u] < WGN[u]) then [4]
(* ребро {v,u} — хорда и не включается в стек*)
17.               begin СТЕК ⇐ {v,u};
18.                  L[v] := min{L[v], WGN[u])
19.               end
20. end; (*ДВУСВ*)
21. begin (*главная программа*)
22.       for u ∈ V do WGN[u] := 0; (*инициализация*)
23.       CTEK := ∅; num := 0;
24.       for u ∈ V do
25.          if WGN[u] = 0 then ДВУСВ(u,0)
26. end
```

Покажем теперь, что вызов процедуры ДВУСВ$(v, 0)$ (строка 25) для еще не

рассматривавшейся вершины v влечет за собой выделение и фиксирование всех блоков компоненты связности графа, содержащей вершину v. Доказательство проводится методом индукции по числу блоков этой компоненты. Если компонента связности не содержит точек сочленения, то нетрудно отметить, что вызов ДВУСВ$(v, 0)$ приводит к засылке в стек всех ребер этой компоненты, а затем (см. цикл 10) запись этих ребер. Предположим теперь, что наша компонента содержит $k > 1$ блоков и что алгоритм работает корректно для произвольной компоненты, содержащей менее k блоков. Рассмотрим первое встретившееся ребро $\{v, u\}$, такое что $L[u] \geq WGN[v]$ в строке 8. Согласно нашим предыдущим рассуждениям это неравенство означает, что v — корень или точка сочленения. Ни один из потомков вершины u (в дереве поиска в глубину, реализованном соответствующим алгоритмом) не является точкой сочленения, и в результате ребра верхней части стека до $\{v, u\}$ включительно являются ребрами графа, соединяющими потомков вершины v вместе с самой u, и тем самым создают блок, который выписывается в цикле 10. Модифицируем теперь нашу компоненту, удалив описанный выше блок (не удаляя вершину v). Модифицированная таким образом компонента имеет $k-1$ блок, и выполненение для нее процедуры ДВУСВ$(v, 0)$ вызывает в силу индуктивного предположения корректную запись этих блоков. Действия алгоритма для модифицированной компоненты отличаются от действий в случае с исходной компонентой только тем, что для первой встреченной точки сочленения (исходной компоненты) v цикл 4 не выполняется для вершин u, принадлежащих удаленному блоку. Отсюда легко следует, что ДВУСВ$(u, 0)$ корректно определяет все k блоков нашей компоненты связности, а тем самым весь алгоритм корректно определяет все блоки графа.

Оценим теперь вычислительную сложность алгоритма. Циклы 22 и 25 требуют $O(n)$ шагов, причем для второго цикла не учитываются шаги, выполняемые при вызове ДВУСВ$(v, 0)$ для каждой еще нерассмотренной вершины. Такой вызов для компоненты связности с n_i вершинами и m_i ребрами требует $O(n_i + m_i)$ шагов, не считая шагов в цикле 10, так как такая процедура ведет в компоненте поиск в глубину, требуя число шагов, ограниченное константой для каждого просматриваемого ребра. Каждое ребро удаляется из стека и попадает в список ребер блока в точности один раз, что дает в сумме $O(m)$ шагов, совершаемых циклом 10 во время его выполнения всего алгоритма. Суммируя все эти слагаемые, получаем в итоге общую сложность алгоритма, равную $O(n + m)$.

2.7 Эйлеровы пути

Эйлеровым путем в графе называется произвольный путь, проходящий через каждое ребро графа в точности один раз, т.е. путь $v_1, \ldots, v_{m+1}$, такой что каждое ребро $e \in E$ появляется в последовательности $v_1, \ldots, v_{m+1}$ в точности один