Тема 6. Алгоритмы на графах

6.4. Остовные деревья

Остовное дерево (остов, каркас, стягивающее дерево) произвольного неориентированного связного графа $G = (V, E)$ есть суграф графа G, представляющий собой дерево $T = (V, E_T)$, где $E_T \subseteq E$, т. е. подграф, содержащий все вершины графа G. Если граф G несвязный, то множество, состоящее из остовных деревьев каждой связной компоненты, называется остовным лесом. Каждое дерево с n вершинами имеет в точности $n - 1$ ребер. Поэтому деревья можно считать минимально связными графами. Удаление любого ребра преобразует дерево в несвязный граф.

Для построения остовного дерева (леса) неориентированного графа G последовательно просматриваются ребра графа, оставляя те, которые не образуют циклов с уже выбранными. Очевидно, что для графа G можно построить множество остовных деревьев. Причем с увеличением числа вершин число остовных деревьев растет экспоненциально (полный граф с n вершинами имеет n^{n-2} остовных деревьев). Простейшим способом построения остовных деревьев является использование процедур поиска в глубину (*DFS*-дерево) и ширину (*BFS*-дерево).

6.4.1. *DFS-дерево*

DFS-дерево – это остовное дерево $T = (V, E_T)$, которое получается в результате поиска в глубину по неориентированному графу $G = (V, E)$, $E_T \subseteq E$. При этом ребра графа разбиваются на два множества: E_T – множество *ребер дерева* и E_B – множество ребер, не вошедших в остовное дерево, называемых *обратными ребрами*, так как они ведут назад в пройденные ранее вершины. Поиск в глубину вводит ориентацию на ребра графа G в соответствии с направлением прохождения, т. е. получается ориентированное остовное дерево. Если в *DFS*-дереве имеется путь из вершины v в вершину w, то v – предок w, а w – потомок v. Поскольку инцидентные вершине ребра графа могут выбираться при поиске в глубину в произвольном порядке, *DFS*-дерево для заданного графа не единственное. Построение остовного дерева способом поиска в глубину представлено алгоритмом 6.3.

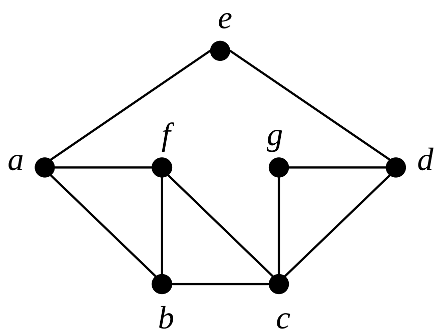
$\textbf{for } x \in V \textbf{ do } num(x) \leftarrow 0$ // инициализация
$i \leftarrow 0$
$E_T \leftarrow \varnothing$ // E_T – множество ребер
$E_B \leftarrow \varnothing$ // E_B – множество обратных ребер
$\textbf{for } x \in V \textbf{ do if } num(x) = 0 \textbf{ then } DFST(x, 0)$

$\textbf{procedure } DFST(v, u)$
  $i \leftarrow i + 1$
  $num(v) \leftarrow i$
  $\textbf{for } w \in \mathrm{Adj}(v) \textbf{ do}$
    $\textbf{if } num(w) = 0$
      $$\textbf{then}\begin{cases} \text{// } (v, w) - \text{ребро дерева} \\ E_T \leftarrow E_T \cup \{(v, w)\} \\ DFST(w, v) \end{cases}$$
      $\textbf{else if } num(w) < num(v) \textbf{ and } w \neq u$
        $$\textbf{then}\begin{cases} \text{// } (v, w) - \text{обратное ребро} \\ E_B \leftarrow E_B \cup \{(v, w)\} \end{cases}$$
$\textbf{return}$

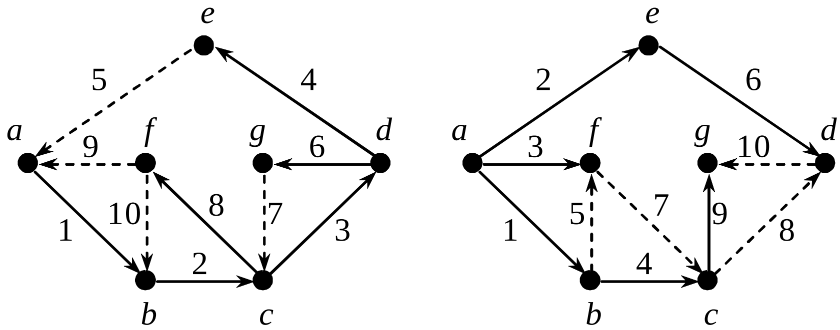
Алгоритм 6.3. Построение остовного дерева поиском в глубину

В алгоритме каждой вершине $v \in V$ сопоставлен элемент $num(v)$. Эти элементы служат для постепенной нумерации вершин числами от 1 до $|V|$ по мере их прохождения. Начальное значение $num(v) = 0$ для всех $v \in V$ показывает, что ни одна вершина не пройдена. Когда вершина v посещается в первый раз, $num(v)$ присваивается ненулевое значение. Рекурсивная процедура $DFST(v, u)$ реализует поиск в глубину в графе $G = (V, E)$, содержащем v, и строит DFS-дерево $T = (V, E_T)$, $E_T \subseteq E$; вершина u является отцом вершины v в дереве. Граф представляется структурой смежности. Элементы множества E_T – ребра дерева, а элементы множества E_B – обратные ребра. Если граф G несвязный, то T будет остовным лесом. Вершина, с которой начинается поиск, считается корнем соответствующего дерева.

DFS-дерево для неориентированного графа (см. рис. 6.2) представлено на рис. 6.3, *а*. Ребра дерева изображены сплошными линиями, обратные ребра – пунктирными. Числа около ребер указывают порядок включения ребер в множества E_T и E_B.



v	Adj(v)
a	b, e, f
b	a, c, f
c	b, d, f, g
d	c, e, g
e	a, d
f	a, b, c
g	c, d



а *б*

Рис. 6.3. Остовные деревья для графа на рис. 6.2
а – *DFS*-дерево; *б* – *BFS*-дерево

Необходимо обратить внимание, что если $num(w) = 0$, то (v, w) – ребро дерева. Если же $num(w) \neq 0$, то условием того, что (v, w) будет обратным ребром, является соотношение $num(w) < num(v)$ и $w \neq u$. Таким образом, нумерация вершин важна для выделения обратных ребер. В случае, если не требуется явное формирование множества E_B обратных ребер, нумерации вершин не требуется, достаточно только отличать уже пройденные вершины от еще не пройденных. Поэтому вместо $num(v)$ можно использовать элементы типа $new(v)$, как в алгоритме поиска в глубину (см. алгоритм 6.1). Нет необходимости также в хранении в элементе u отца вершины v. Таким образом, более простая версия алгоритма будет формировать только множество E_T ребер дерева.

Временная сложность алгоритма определяется сложностью поиска в глубину, т. е. $O(|V| + |E|)$.

6.4.2. BFS-дерево

BFS-дерево – это остовное дерево $T = (V, E_T)$, которое получается в результате поиска в ширину по неориентированному графу $G = (V, E)$, $E_T \subseteq E$. При этом ребра графа разбиваются на два множества: E_T – множество *ребер дерева* и E_C – множество ребер, не вошедших в остовное дерево, которые можно назвать *поперечными ребрами*, так как они соединяют вершины, не являющиеся в дереве ни предками, ни потомками друг друга. Поиск в ширину вводит ориентацию на ребра графа G в соответствии с направлением прохождения, т. е. получается ориентированное остовное дерево. Если граф $G = (V, E)$ несвязный, то $T = (V, E_T)$ будет остовным лесом. Построение *BFS*-дерева способом поиска в ширину представлено алгоритмом 6.4.

```
for x ∈ V do num(x) ← 0   // инициализация
i ← 0
E_T ← ∅   // E_T – множество ребер дерева
E_C ← ∅   // E_C – множество поперечных ребер
for x ∈ V do if num(x) = 0 then BFST(x)

procedure BFST(u)
   Q ← ∅   // очередь Q пуста
   Q ⇐ u
   i ← i + 1
   num(u) ← i
   while Q ≠ ∅ do
      v ⇐ Q
      for w ∈ Adj(v) do
         if num(w) = 0
            then
               // (v, w) – ребро дерева
               Q ⇐ w
               i ← i + 1
               num(w) ← i
               E_T ← E_T ∪ {(v, w)}
         else if num(w) > num(v)
            then
               // (v, w) – поперечное ребро
               E_C ← E_C ∪ {(v, w)}
return
```

Алгоритм 6.4. Построение остовного дерева поиском в ширину

Критерием распознавания поперечных ребер является соотношение $num(w) > num(v)$. Если $num(w) < num(v)$, это означает, что все инцидентные вершине w ребра уже исследованы и классифицированы.

BFS-дерево для неориентированного графа (см. рис. 6.2) представлено на рис. 6.3, *б*. Ребра дерева изображены сплошными линиями, поперечные ребра – пунктирными. Числа около ребер дерева указывают порядок включения ребер в множества E_T и E_C.

Временная сложность алгоритма определяется сложностью поиска в ширину, т. е. $O(|V| + |E|)$.