23 Основные алгоритмы на графах

В этой главе описаны основные способы представления графов и алгоритмы обхода графов. Обходя граф, мы двигаемся по рёбрам и проходим все вершины. При этом накапливается довольно много информации, которая полезна для дальнейшей обработки графа, так что обход графа входит как составная часть во многие алгоритмы.

В разделе 23.1 обсуждаются два основных способа представления графа в памяти компьютера — с помощью списков смежных вершин и с помощью матрицы смежности. Раздел 23.2 описывает алгоритм поиска в ширину и соответствующее этому алгоритму дерево. В разделе 23.3 рассматривается другой порядок обхода вершин и доказываются свойства соответствующего алгоритма (называемого поиском в глубину). В разделе 23.4 мы используем поиск в глубину для решения задачи о топологической сортировке ориентированного графа без циклов. Другое применение поиска в глубину (отыскание сильно связных компонент ориентированного графа) описано в разделе 23.5.

23.1. Представление графов

Есть два стандартных способа представить граф $G = (V, E)$ — как набор списков смежных вершин или как матрицу смежности. Первый обычно предпочтительнее, ибо даёт более компактное представление для **разреженных** (sparse) графов — тех, у которых $|E|$ много меньше $|V|^2$. Большинство излагаемых нами алгоритмов используют именно это представление. Однако в некоторых ситуациях удобнее пользоваться матрицей смежности — например, для **плотных** (dense) графов, у которых $|E|$ сравнимо с $|V|^2$. Матрица смежности позволяет быстро определить, соединены ли две данные вершины ребром. Два алгоритма отыскания кратчайших путей для всех пар вершин, описанные в главе 26, используют представление графа с помощью матрицы смежности.

Представление графа $G = (V, E)$ в виде **списков смежных вершин** (adjacency-list representation) использует массив Adj из $|V|$ списков — по одному на вершину. Для каждой вершины $u \in V$ список смежных вершин $Adj[u]$ содержит в произвольном порядке (указатели на) все смежные с ней вершины (все вершины v, для которых $(u, v) \in E$).

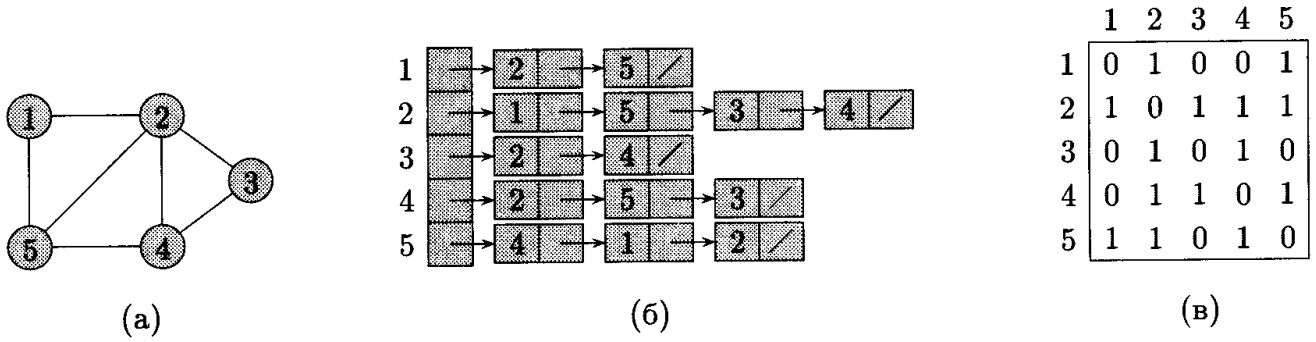

Рис. 23.1. Два представления неориентированного графа. (а) Неориентированный граф G с 5 вершинами и 7 рёбрами. (б) Представление этого графа с помощью списков смежных вершин. (в) Представление этого графа в виде матрицы смежности.

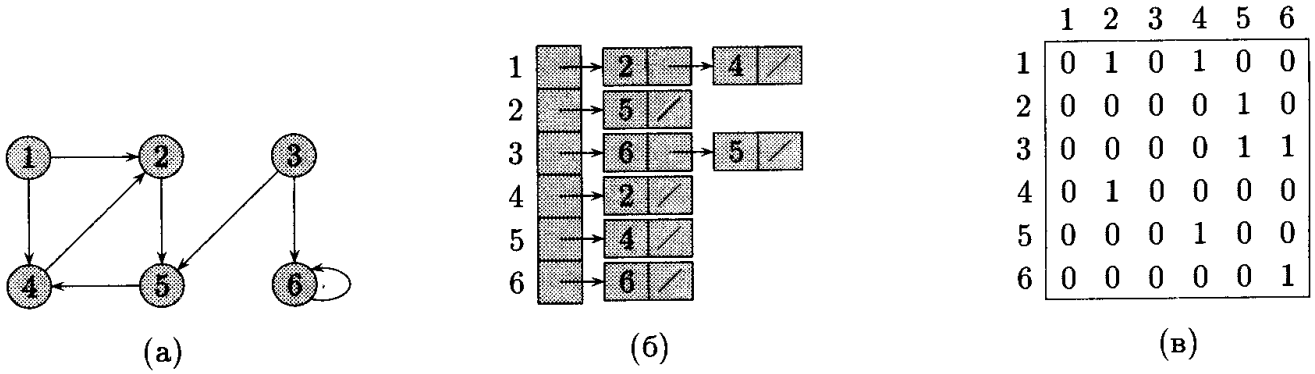

Рис. 23.2. Два представления ориентированного графа. (а) Ориентированный граф G с 6 вершинами и 8 рёбрами. (б) Представление этого графа с помощью списков смежных вершин. (в) Представление этого графа в виде матрицы смежности.

На рис. 23.1(б) показано представление неориентированного графа рис. 23.1(а) с помощью списков смежных вершин. Аналогичное представление для ориентированного графа рис. 23.2(а) изображено на рис. 23.2(б).

Для ориентированного графа сумма длин всех списков смежных вершин равна общему числу рёбер: ребру (u, v) соответствует элемент v списка $Adj[u]$. Для неориентированного графа эта сумма равна удвоенному числу рёбер, так как ребро (u, v) порождает элемент в списке смежных вершин как для вершины u, так и для v. В обоих случаях количество требуемой памяти есть $O(\max(V, E)) = O(V + E)$.

Списки смежных вершин удобны для хранения **графов с весами** (weighted graphs), в которых каждому ребру приписан некоторый вещественный **вес** (weight), то есть задана **весовая функция** (weight function) $w\colon E \to \mathbb{R}$. В этом случае удобно хранить вес $w(u, v)$ ребра $(u, v) \in E$ вместе с вершиной v в списке вершин, смежных с u. Подобным образом можно хранить и другую информацию, связанную с графом.

Недостаток этого представления таков: если мы хотим узнать, есть ли в графе ребро из u в v, приходится просматривать весь список $Adj[u]$ в поисках v. Этого можно избежать, представив граф в виде матрицы смежности — но тогда потребуется больше памяти.

При использовании **матрицы смежности** (adjacency matrix) мы нумеруем вершины графа (V, E) числами $1, 2, \ldots, |V|$ и рассматриваем матрицу $A = (a_{ij})$ размера $|V| \times |V|$, для которой

$$a_{ij} = \begin{cases} 1, & \text{если } (i,j) \in E, \\ 0 & \text{в противном случае.} \end{cases}$$

На рис. 23.1(в) и 23.2(в) показаны матрицы смежности неориентированного и ориентированного графов рис. 23.1(а) и 23.2а соответственно. Матрица смежности требует $\Theta(V^2)$ памяти независимо от количества рёбер в графе.

Для неориентированного графа матрица смежности симметрична относительно главной диагонали (как на рис. 23.1(в)), поскольку (u, v) и (v, u) — это одно и то же ребро. Другими словами, матрица смежности неориентированного графа совпадает со своей **транспонированной** (transpose). (Транспонированием называется переход от матрицы $A = (a_{ij})$ к матрице $A^T = (a_{ij}^T)$, для которой $a_{ij}^T = a_{ji}$.) Благодаря симметрии достаточно хранить только числа на главной диагонали и выше неё, тем самым мы сокращаем требуемую память почти вдвое.

Как и для списков смежных вершин, хранение весов не составляет проблемы: вес $w(u, v)$ ребра (u, v) можно хранить в матрице на пересечении u-й строки и v-го столбца. Для отсутствующих рёбер можно записать специальное значение NIL (в некоторых задачах вместо этого пишут 0 или ∞).

Для небольших графов, когда места в памяти достаточно, матрица смежности бывает удобнее — с ней часто проще работать. Кроме того, если не надо хранить веса, то элементы матрицы смежности представляют собой биты, и их можно размещать по нескольку в одном машинном слове, что даёт заметную экономию памяти.

Упражнения

23.1-1. Граф хранится в виде списков смежных вершин. Сколько операций нужно, чтобы найти а) число выходящих из данной вершины рёбер? б) число входящих в данную вершину рёбер?

23.1-2. Укажите представление в виде списков смежных вершин и матрицу смежности для графа, являющегося полным двоичным деревом с 7 вершинами (пронумерованными от 1 до 7 как при сортировке с помощью кучи в главе 7).

23.1-3. Что произойдёт с матрицей смежности ориентированного графа, если обратить направления стрелок на всех его рёбрах, заменив каждое ребро (u, v) на ребро (v, u)? Как обратить направления стрелок, если граф хранится в форме списков смежных вершин? Оцените требуемое для этого число операций.

23.1-4. Мультиграф $G = (V, E)$ представлен в виде списков смежных вершин. Как за время $O(V + E)$ преобразовать его в обычный неориентированный граф $G' = (V, E')$, заменив кратные рёбра на обычные и удалив рёбра-циклы?

23.1-5. **Квадратом** (square) ориентированного графа $G = (V, E)$ называется граф $G^2 = (V, E^2)$, построенный так: $(u, w) \in E^2$, если существует вершина $v \in V$, для которой $(u, v) \in E$ и $(v, w) \in E$ (две вершины соединяются ребром, если раньше был путь из двух рёбер). Как преобразовать G в G^2, если граф хранится в виде списков смежных вершин? как матрица смежности? Оцените время работы ваших алгоритмов.

23.1-6. Почти любой алгоритм, использующий матрицы смежности, требует времени $\Theta(V^2)$ (просто на чтение этой матрицы), но бывают и исключения. Покажите, что за время $O(V)$ по матрице смежности можно выяснить, содержит ли ориентированный граф «сток» (sink) — вершину, в которую ведут рёбра из всех других вершин и из которой не выходит ни одного ребра.

23.1-7. Назовём **матрицей инцидентности** (incidence matrix) ориентированного графа $G = (V, E)$ матрицу $B = (b_{ij})$ размера $|V| \times |E|$, в которой

$$b_{ij} = \begin{cases} -1, & \text{если ребро } j \text{ выходит из вершины } i, \\ 1, & \text{если ребро } j \text{ входит в вершину } i, \\ 0 & \text{в остальных случаях.} \end{cases}$$

Каков смысл элементов матрицы BB^T? (Здесь B^T — транспонированная матрица.)

23.2. Поиск в ширину

Поиск в ширину (breadth-first search) — один из базисных алгоритмов, составляющий основу многих других. Например, алгоритм Дейкстры поиска кратчайших путей из одной вершины (глава 25) и алгоритм Прима поиска минимального покрывающего дерева (раздел 24.2) могут рассматриваться как обобщения поиска в ширину.

Пусть задан граф $G = (V, E)$ и фиксирована **начальная вершина** (source vertex) s. Алгоритм поиска в ширину перечисляет все достижимые из s (если идти по рёбрам) вершины в порядке возрастания расстояния от s. Расстоянием считается длина (число рёбер) кратчайшего пути. В процессе поиска из графа выделяется часть, называемая «деревом поиска в ширину» с корнем s. Она содержит все достижимые из s вершины (и только их). Для каждой из них путь из корня в дереве поиска будет одним из кратчайших путей (из начальной вершины) в графе. Алгоритм применим и к ориентированным, и к неориентированным графам.

Название объясняется тем, что в процессе поиска мы идём вширь, а не вглубь (сначала просматриваем все соседние вершины, затем соседей соседей и т. д.).

Для наглядности мы будем считать, что в процессе работы алгоритма вершины графа могут быть белыми, серыми и чёрными. Вначале они все белые, но

в ходе работы алгоритма белая вершина может стать серой, а серая — чёрной (но не наоборот). Повстречав новую вершину, алгоритм поиска красит её, так что окрашенные (серые или чёрные) вершины — это в точности те, которые уже обнаружены. Различие между серыми и чёрными вершинами используется алгоритмом для управления порядком обхода: серые вершины образуют «линию фронта», а чёрные — «тыл». Более точно, поддерживается такое свойство: если $(u, v) \in E$ и u чёрная, то v — серая или чёрная вершина. Таким образом, только серые вершины могут иметь смежные необнаруженные вершины.

Вначале дерево поиска состоит только из корня — начальной вершины s. Как только алгоритм обнаруживает новую белую вершину v, смежную с ранее найденной вершиной u, вершина v (вместе с ребром (u, v)) добавляется к дереву поиска, становясь **ребёнком** (child) вершины u, а u становится **родителем** (parent) v. Каждая вершина обнаруживается только однажды, так что двух родителей у неё быть не может. Понятия **предка** (ancestor) и **потомка** (descendant) определяются как обычно (потомки — это дети, дети детей, и т. д.). Двигаясь от вершины к корню, мы проходим всех её предков.

Приведённая ниже процедура BFS (breadth-first search — поиск в ширину) использует представление графа $G = (V, E)$ списками смежных вершин. Для каждой вершины u графа дополнительно хранятся её цвет $color[u]$ и её предшественник $\pi[u]$. Если предшественника нет (например, если $u = s$ или u ещё не обнаружена), $\pi[u] = \text{NIL}$. Кроме того, расстояние от s до u записывается в поле $d[u]$. Процедура использует также очередь Q (FIFO, раздел 11.1) для хранения множества серых вершин.

```
BFS(G, s)
 1  for (для) каждой вершины u ∈ V[G] − {s}
 2        do color[u] ← БЕЛЫЙ
 3           d[u] ← ∞
 4           π[u] ← NIL
 5  color[s] ← СЕРЫЙ
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← {s}
 9  while Q ≠ ∅
10        do u ← head[Q]
11           for (для) всех v ∈ Adj[u]
12                 do if color[v] = БЕЛЫЙ
13                       then color[v] ← СЕРЫЙ
14                            d[v] ← d[u] + 1
15                            π[v] ← u
16                            ENQUEUE(Q, v)
17           DEQUEUE(Q)
18           color[u] ← ЧЁРНЫЙ
```

На рис. 23.3 приведён пример исполнения процедуры BFS.

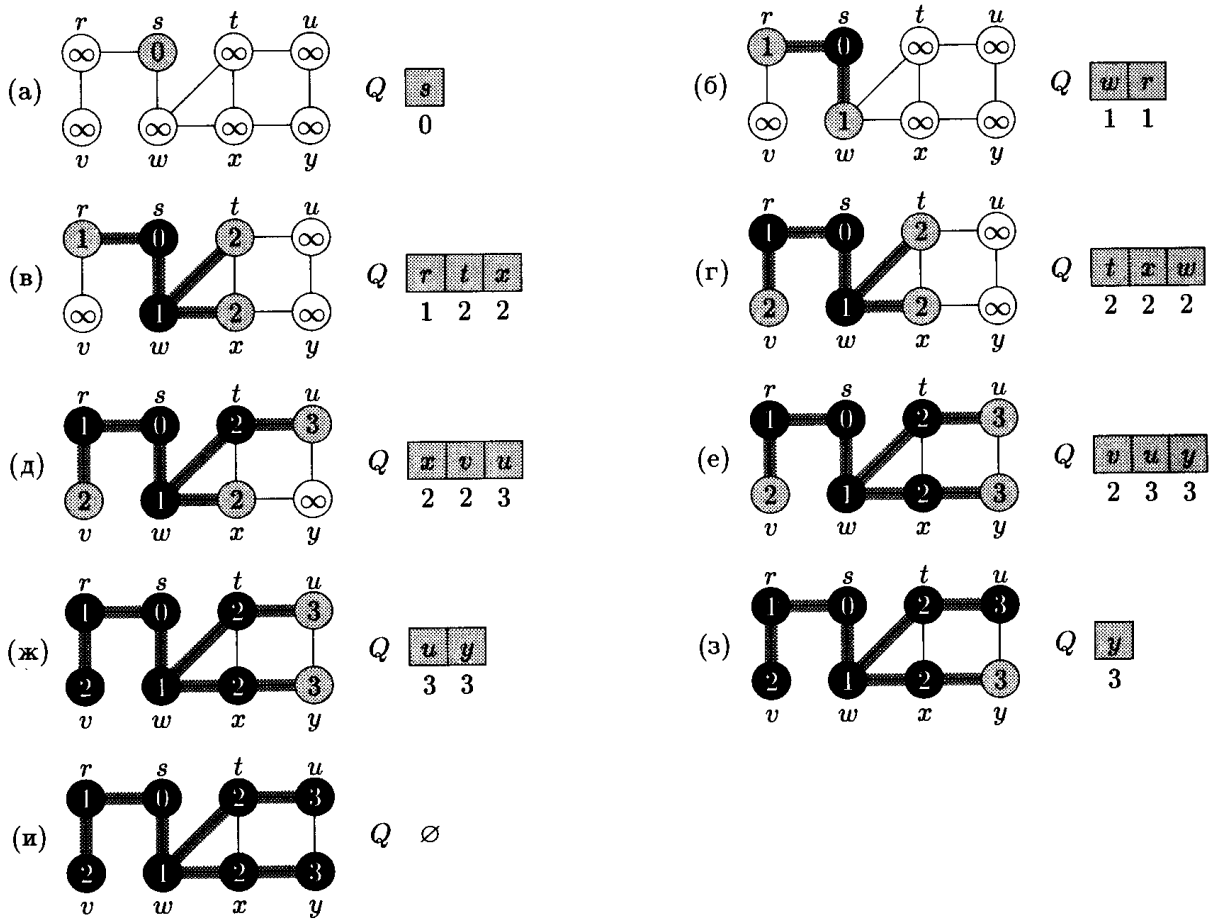

Рис. 23.3. Исполнение процедуры BFS для неориентированного графа. Рёбра формируемого дерева показаны серыми. Внутри каждой вершины u указано значение $d[u]$. Показано состояние очереди Q перед каждым повторением цикла в строках 9–18. Рядом с элементами очереди показаны расстояния от корня.

В строках 1–4 все вершины становятся белыми, все значения d бесконечными, и родителем всех вершин объявляется NIL. Строки 5–8 красят вершину s в серый цвет и выполняют связанные с этим действия: в строке 6 расстояние $d[s]$ объявляется равным 0, а в строке 7 говорится, что родителя у s нет. Наконец, в строке 8 вершина s помещается в очередь Q, и с этого момента очередь будет содержать все серые вершины и только их.

Основной цикл программы (строки 9–18) выполняется, пока очередь непуста, то есть существуют серые вершины (вершины, которые уже обнаружены, но списки смежности которых ещё не просмотрены). В строке 10 первая такая вершина помещается в u. Цикл **for** в строках 11–16 просматривает все смежные с ней вершины. Обнаружив среди них белую вершину, мы делаем её серой (строка 13), объявляем u её родителем (строка 15) и устанавливаем расстояние равным $d[u] + 1$ (строка 14). Наконец, эта вершина добавляется в хвост очереди Q (строка 16). После этого уже можно удалить вершину u из очереди Q, перекрасив эту вершину в чёрный цвет (строки 17–18).