Глава 4

Пути в графах

4.1. Расстояния в графе

Поиск в глубину не только быстро находит все вершины, достижимые из начальной, но и строит дерево, содержащее пути к ним (рис. 4.1). Однако эти пути не обязательно будут *кратчайшими*: например, от S до C можно добраться по одному ребру, в то время как поиск в глубину находит путь из трёх. А как искать кратчайшие, мы изучим в этой главе.

Расстоянием (distance) между двумя вершинами неориентированного графа будем называть длину кратчайшего пути между ними, измеренную в рёбрах. У этого понятия есть простой физический смысл. Представим себе граф из шариков (вершин), соединённых нитками одинаковой длины (рёбрами). Потянем граф вверх за вершину s; за ней потянутся все вершины, достижимые из s. Чтобы найти расстояния от них до s, мы можем теперь просто измерить, насколько они ниже s.

Например, на рис. 4.2 расстояние между вершинами S и B равно 2, и есть два кратчайших пути между ними. Когда граф подвешен за S, все рёбра этих

Рис. 4.1. (a) Простой граф и (b) его дерево поиска в глубину.

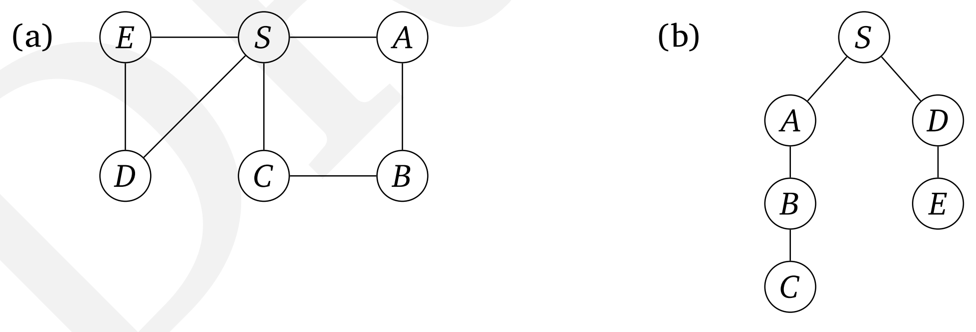
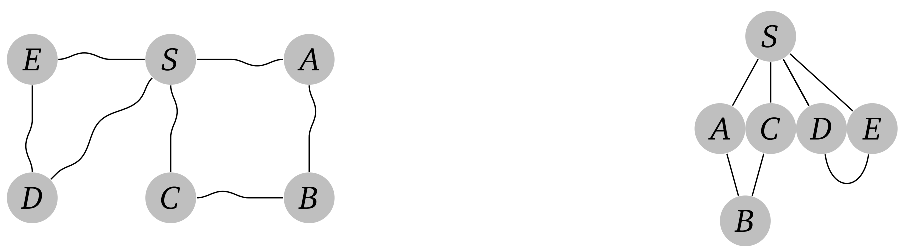


Рис. 4.2. Физическая модель графа.



двух путей натягиваются. А ребро (D, E) провисает, поскольку не входит ни в один из кратчайших путей из вершины S.

4.2. Поиск в ширину

Вершины графа рис. 4.2, подвешенного за S, разбиваются на слои: сама S, вершины на расстоянии 1 от S, вершины на расстоянии 2 и так далее. Можно находить расстояния от S до всех вершин, переходя от уровня к уровню. Когда вершины уровней $0, 1, 2, \ldots, d$ определены, легко найти вершины уровня $d+1$: это просто ещё не просмотренные вершины, смежные с вершинами уровня d. Эти рассуждения наталкивают нас на алгоритм, работающий в каждый момент с двумя уровнями: некоторым уровнем d, который уже полностью известен, и уровнем $d+1$, который находится просмотром соседей вершин уровня d.

Поиск в ширину (breadth-first search, BFS) непосредственно реализует эту простую идею (рис. 4.3). Изначально очередь Q содержит только вершину S, то есть вершину на расстоянии 0. Для каждого последующего расстояния $d = 1, 2, 3, \ldots$ найдётся момент времени, в который Q содержит все вершины на расстоянии d и только их. Когда все эти вершины будут обработаны (извлечены из очереди), их непросмотренные соседи окажутся добавленными в конец очереди, то есть мы перейдём к следующему значению d.

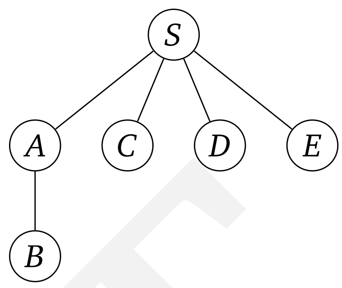
Запустим этот алгоритм для вершины S в графе рис. 4.1. Считаем, что соседи каждой вершины обрабатываются в алфавитном порядке. На рис. 4.4 слева показан порядок обхода вершин, а справа изображено дерево поиска в ширину. Оно содержит только рёбра, при переходе по которым обнаруживались новые вершины. В отличие от дерева поиска в глубину все пути данного дерева с началом в S являются кратчайшими. Поэтому оно называется *деревом кратчайших путей* (shortest-path tree).

Рис. 4.3. Поиск в ширину.

```
процедура BFS(G, s)
{Вход: граф G(V, E), вершина s ∈ V.}
{Выход: для всех вершин u, достижимых из s,
  dist[u] будет равно расстоянию от s до u.}
для всех вершин u ∈ V:
  dist[u] ← ∞
dist[s] ← 0
Q ← {s} {очередь из одного элемента}
пока Q не пусто:
  u ← EJECT(Q)
  для всех рёбер (u, v) ∈ E:
    если dist[v] = ∞:
      INJECT(Q, v)
      dist[v] ← dist[u] + 1
```

Рис. 4.4. Результат поиска в ширину для графа рис. 4.1.

порядок посещения	содержание очереди после обработки вершины
	$[S]$
S	$[A\ C\ D\ E]$
A	$[C\ D\ E\ B]$
C	$[D\ E\ B]$
D	$[E\ B]$
E	$[B]$
B	$[\,]$



Этот алгоритм можно применять и к ориентированным графам. В них тоже можно определить расстояние от вершины S до вершины T как минимальное число рёбер, которое надо пройти (в их направлении), чтобы из S попасть в T. Расстояние при этом уже не будет симметрично, но понятие кратчайшего пути имеет смысл, и наш алгоритм по-прежнему годится.

Корректность и время работы

Убедимся в корректности алгоритма. Мы утверждаем, что

> для всех $d = 0, 1, 2, \ldots$ найдётся момент времени, когда: 1) для всех вершин на расстоянии не более d это расстояние уже помещено в массив `dist`; 2) для всех оставшихся вершин значения в `dist` равны ∞; 3) очередь содержит в точности вершины на расстоянии d.

Данное утверждение легко доказывается по индукции (проверьте).

Как и в случае поиска в глубину, время работы поиска в ширину линейно, то есть равно $O(|V| + |E|)$. Действительно, каждая вершина помещается в очередь ровно один раз — при её обнаружении. Поэтому общее количество операций с очередью есть $2|V|$. Вся оставшаяся работа производится во внутреннем цикле алгоритма. На это требуется время $O(|E|)$, поскольку каждое ребро в данном цикле просматривается один раз (для ориентированных графов) или два раза (для неориентированных).

Теперь, когда мы знаем и поиск в глубину, и поиск в ширину, интересно сравнить их способы обхода графа. Поиск в глубину стремится вперёд, и возвращается назад, чтобы пойти вбок, только если впереди уже нет новых вершин. Поэтому он может долго добираться до вершины, которая находится совсем близко (рис. 4.1). Поиск в ширину обходит вершины в порядке увеличения расстояний до них от начальной вершины, как фронт волны на воде.

Если в алгоритме поиска в ширину заменить очередь на стек (вынимается первым тот элемент, который положен последним), то он превратится в поиск в глубину (в нерекурсивном варианте).

При поиске в ширину нас интересуют расстояния от s, поэтому мы не перезапускаем поиск в других связных компонентах (недоступные вершины просто игнорируются).