Графы

Часто возникает необходимость представления отношений между какими-либо объектами. Если объекты изобразить вершинами, а связи — рёбрами, получится граф. Например, графом может представляться компьютерная сеть, сеть дорог между городами, блок-схема программы и т.д. и т.п. Задача о графах возникла ещё в 1736 году — задача Эйлера о кёнигсбергских мостах. Задача — определить, можно ли пройти по всем мостам города, не проходя дважды ни по одному из них.

Графы бывают ориентированные и неориентированные.

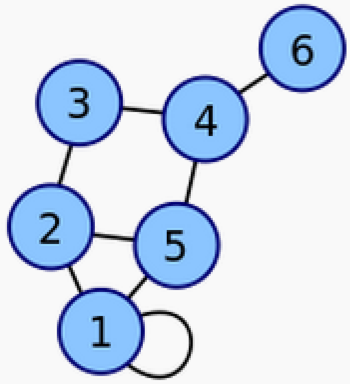
Ориентированный граф (орграф) G — это пара из множества вершин V и множества дуг E, где E — упорядоченная пара вершин (v, w) (т.е. E — бинарное отношение над множеством вершин). v называется началом, w — концом дуги. Рёбра вида (v, v) называются петлями.

Путём в орграфе называется последовательность вершин, для которых существуют дуги из предыдущей в следующую. Длина пути — количество дуг, составляющих путь. Путь называется простым, если все вершины на нём, за исключением, быть может, первой и последней, различны. Цикл — это простой путь длины не менее 1, который начинается и заканчивается в одной вершине. Если в графе нет циклов, он называется ациклическим. Граф может быть помеченным.

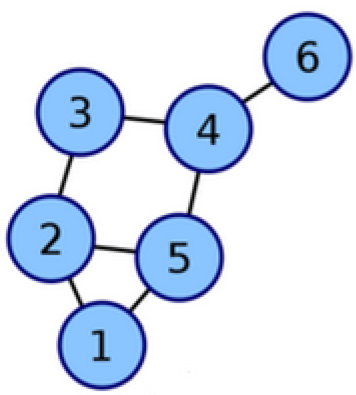
Неориентированный граф — это ориентированный граф, у которого для каждого ребра (v, w) существует противоположное ребро (w, v), то есть отношение E симметрично. Если e = (u,v) принадлежит E, то вершины u и v называются смежными в G, а ребро e и эти вершины называются инцидентными. Степенью вершины в неориентированном графе называется число смежных с ней вершин. Вершина степени 0 называется изолированной.

Представления графов

1. Матрица смежности — положим, что есть множество вершин {1, 2, ..., n}. Матрица смежности — матрица A размера n x n, где A[i, j] = true, если существует дуга между вершинами i и j. Если граф помеченный, в матрице вместо true/false может быть метка дуги. В этом случае нужно иметь зарезервированное значение, которое говорит, что дуги нет. Такое представление плохо, поскольку требует O(n^2) памяти, даже если дуг в графе значительно меньше, чем n^2.

Граф	Матрица смежности
	$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$

2. Матрица инцидентности — матрица B размером n x m, где B[i, j] = 1, если для некоторого k существует ребро e_j(i, k); B[i, j] = -1, если для некоторого k существует ребро e_j(k, i); B[i, j] = 2 — если это ребро-петля.

Граф	Матрица инцидентности
	$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

3. Список смежности — есть массив вершин, в каждой ячейке которого хранится список дуг, исходящих из вершины. В этом случае требуется всего O(m+n) памяти, но O(n) времени для поиска определённой дуги. Так что надо выбирать представление, наиболее удобное для каждого конкретного алгоритма.

Достижимость

Вершина w достижима из вершины v, если v = w или в G есть путь из v в w. Иначе говоря, отношение достижимости является рефлексивным и транзитивным замыканием отношения E. В неориентированном графе отношение достижимости является отношением эквивалентности над множеством вершин, классы эквивалентности называются компонентами связности. Для неориентированных графов эквивалентностью является отношение взаимной достижимости.

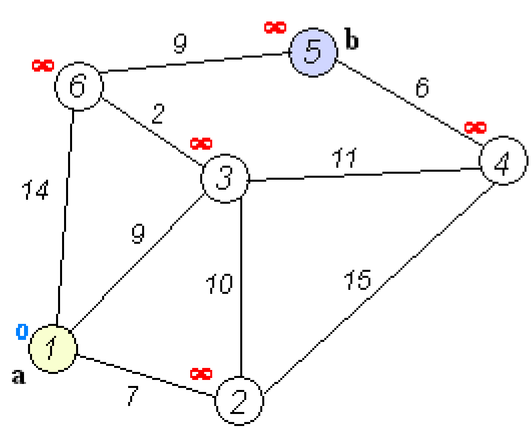
Задача о нахождении кратчайшего пути в графе

Пусть есть ориентированный граф G, у которого все вершины имеют неотрицательные метки (стоимости дуг), а одна вершина определена как источник. Задача — найти кратчайший путь от источника до всех остальных вершин (длина пути определяется как сумма стоимостей дуг, из которых этот путь состоит). Для решения этой задачи может быть использован алгоритм Дейкстры — классический пример "жадного" алгоритма. Жадный алгоритм — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным.

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, назовем 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 посещенную и повторим шаг алгоритма.



Псевдокод выглядит как-то так:

Обозначения

V — множество вершин графа

E — множество ребер графа

w[ij] — вес (длина) ребра ij

a — вершина, расстояния от которой ищутся

U — множество посещенных вершин

d[u] — по окончании работы алгоритма равно длине кратчайшего пути из a до вершины u

p[u] — по окончании работы алгоритма содержит кратчайший путь из a в u

Псевдокод

Присвоим $d[a] \leftarrow 0,\ p[a] \leftarrow a$

Для всех $u \in V$ отличных от *a*

  присвоим $d[u] \leftarrow \infty$

Пока $\exists v \notin U$ с $d[v] < \infty$

  Пусть $v \notin U$ — вершина с минимальным *d*[*v*]

  Добавим вершину *v* к *U*

  Для всех $u \notin U$ таких, что $vu \in E$если *d*[*u*] > *d*[*v*] + *w*[*vu*] то

    изменим $d[u] \leftarrow d[v] + w[vu]$

    изменим $p[u] \leftarrow p[v], u$

Как это может быть реализовано: множество U может быть представлено битовым вектором, d и p — просто массивы целых.

```
for (int i = 0; i < size; ++i)
{
    int w = findMin(distance, seen);
    seen[w] = true;
    for (int v = 0; v < size; ++v)
        if (!seen[v])
                if (distance[v] > distance[w] + m[w][v])
                {
                        distance[v] = distance[w] + m[w][v];
                        path[v] = w;
                }
}
```

Возможны варианты реализации, например, представлять множество "соседних" к просмотренному множеству вершин упорядоченным списком.

Обходы графов

1. Поиск в глубину (Depth-first search). Положим, что есть граф G, в котором все вершины первоначально помечены меткой unvisited (не посещались). Берём начальную вершину, помечаем её visited, затем для каждой вершины, смежной с вершиной v и не посещённой ранее, вызывается рекурсивно поиск в глубину. Как вариант — смежные вершины складываются в стек, на каждом шаге вершина снимается со стека и рассматривается. Для несвязных графов может потребоваться выбрать новую стартовую вершину.
2. Поиск в ширину — то же самое, что и нерекурсивный вариант поиска в глубину, только вместо стека — очередь.

Оба обхода работают за линейное время.

Методом поиска в глубину можно проверить граф на ацикличность — в процессе работы поиска в глубину получится "глубинный остовный лес". Если на каком-то шаге мы найдём дугу, идущую от потомка глубинного остовного дерева к предку ("обратную дугу"), то граф имеет цикл. Если нет — то не имеет. Чтобы быстро искать обратные дуги, можно нумеровать вершины в порядке обхода — "глубинная нумерация".

Доклады

1. Алгоритм Флойда-Уоршелла
2. Алгоритмы Прима и Краскала
3. Алгоритм A*
4. Алгоритм Кнута-Морриса-Пратта
5. Алгоритм Бойера-Мура
6. Алгоритм Рабина-Карпа