Представления графов. Остовные деревья. Обходы в ширину и глубину

Глоссарий

Введём некоторые наиболее важные определения, на которые будем опираться.

Граф это тройка $G = \{V, E, \varphi\}$, где V и E – множества вершин (от англ. **V**ertex) и рёбер (от англ. **E**dge) соответственно, а φ – функция инцидентности, сопоставляющая каждому ребру $e \in E$ пару вершин $u, v \in V$ и, возможно, некоторую дополнительную информацию.

Такой информацией может быть вес ребра, пропускная способность, и т.д. Множества вершин и рёбер обычно полагают конечными, в дальнейшем мы будем использовать V и E вместо $|V|$ и $|E|$ для обозначения мощности этих множеств, что будет понятно по контексту.

Это наиболее общее определение и оно требует ряда уточнений, чтобы однозначно описывать класс графов, с которыми мы работаем. Выделим следующие побочные определения теории графов:

Инцидентными называют вершину v и ребро e если v – один из концов e.

Смежными (англ. adjacent) называют вершины, инцидентные одному и тому же ребру.

Петлёй называют ребро, концы которого совпадают.

Кратными (параллельными) называют рёбра, инцидентные одной и той же паре вершин.

Простым называют граф, не содержащий петель и кратных рёбер.

Зачастую это свойство подразумевается по умолчанию, в то время, как графы, содержащие кратные рёбра называют **мультиграфами**, а графы, содержащие как кратные рёбра, так и петли – **псевдографами**. В случае простых графов можно отождествлять рёбра с парами вершин, которые им соответствуют и считать, что $E \subset V^2$.

В дальнейшем мы будем считать, что граф является простым, если явно не указано обратное.

Неориентированным называют граф, концы рёбер которого задаются неупорядоченными парами вершин.

Ориентированным называют граф, концы рёбер которого задаются упорядоченными парами вершин. Вершины ориентированных графов зачастую называют узлами (англ. node), а рёбра – дугами (англ. arrow). Также встречается сокращение орграф (англ. digraph).

Двудольным называют неориентированный граф, который можно раскрасить в два цвета таким образом чтобы никакое ребро не соединяло вершины одного цвета. Вершины одного цвета образуют **доли** этого графа.

Полным называют простой граф, содержащий все возможные рёбра. То есть, $V(V-1)$ и $V(V-1)/2$ рёбер для ориентированного и неориентированного графов соответственно. Может сочетаться с другими обозначениями, например, полный двудольный граф с долями размера a и b содержит $a \cdot b$ рёбер. Полный неориентированные графы на n вершинах принято обозначать K_n, а полные двудольные графы на долях размера a и b как $K_{a,b}$.

Такое обозначение для полных графов, по всей видимости, используется как дань уважения Казимежу Куратовскому, сформулировавшему в 1930г. критерий планарности графа, опирающийся на графы K_5 и $K_{3,3}$.

Степенью вершины называют число рёбер графа, инцидентных ей.

Взвешенным называют граф, каждому ребру которого приписано число – его вес.

Путём называют последовательность рёбер $e_1, e_2, \ldots, e_k$, такая что у каждого ребра выделены начальная и конечная вершины и при этом конечная вершина e_i совпадает с начальной вершиной e_{i+1}.

Циклом называют путь такой что $e_k \neq e_{i \bmod k+1}$, а также начало e_1 и конец e_k совпадают.

Простым называют путь (цикл), проходящий по любой вершине не больше одного раза.

Эйлеровым циклом называют цикл, который проходит по каждому ребру графа ровно один раз.

Гамильтоновым циклом называют цикл, который проходит по каждой вершине графа ровно один раз.

Ациклическим называют граф, не содержащий циклов.

Достижимой из v называют вершину u если из v в u есть путь.

Достижимость – рефлексивное и транзитивное отношение. То есть, если обозначать её как $v \to u$, то:

1. $v \to v$
2. $(a \to b) \wedge (b \to c) \implies (a \to c)$
3. (a) $(a \to b) \implies (b \to a)$ для неориентированных графов, то есть, там это отношение эквивалентности.

   (b) $(a \to b) \implies (b \not\to a)$ для ориентированных ациклических графов, то есть, там это отношение порядка.

Позже мы подробнее поговорим про отношение достижимости в ориентированных графах

Связным называют неориентированный граф, в котором из любой вершины есть путь в любую другую.

Деревом называют связный ациклический граф.

1. (2б) Докажите, что следующие утверждения эквивалентны.

1. Граф является деревом.
2. Граф связен и в нём ровно $V - 1$ ребро.
3. Для любой пары вершин u и v есть ровно один простой путь из u в v.
4. Рёбра графа можно ориентировать так, чтобы у всех вершин, кроме одной было ровно одно входящее ребро.

Слабо связным называют орграф, являющийся связным при игнорировании ориентации дуг.

Сильно связным называют орграф, в котором из любой вершины есть путь в любую другую.

2. (1б) Приведите пример слабо, но не сильно связного графа.

Турниром называют орграф, между любой парой вершин которого есть ровно одно ориентированное.

1°. (4б) 1. Докажите, что в любом турнире есть гамильтонов путь.

2. Докажите, что в любом сильно связном турнире есть гамильтонов цикл.

Подграфом исходного графа называют некоторое подмножество вершин графа и инцидентных им рёбер.

Остовом (остовным деревом) называют подграф, содержащий все вершины графа и являющийся деревом.

Расстоянием между вершинами u и v называют длину пути из u в v, имеющего наименьшую длину. Будем обозначать расстояние как $\rho(u, v)$. Если граф взвешен, то длина пути считается суммой весов рёбер графа. Иначе считается, что каждое ребро имеет вес 1. Если v не достижима из u, то $\rho(u, v) = +\infty$. Если в графе есть цикл отрицательного веса такой, что он достижим из u, а из него достижима v, то $\rho(u, v) = -\infty$.

Деревом кратчайших путей для графа G и вершины v называют корневое дерево, построенное на вершинах, достижимых из v, с корнем v, такое что для любой вершины u в этом дереве путь из v в u по дереву является одним из кратчайших путей из v в u в исходном графе G.

Представления графов

Все примеры в данных материалах будут использовать синтаксис C++.
По умолчанию считаем, что вершинам сопоставлены числа от 0 до $V - 1$.

Графы можно хранить в памяти разными способами. Какой именно способ использовать – вопрос, зависящий в первую очередь от решаемой задачи. Основные три представления:

Список рёбер. Просто храним список пар вершин $\{u_i, v_i\}$, которые соединяет i-е ребро.

Список смежности. Для каждой вершины храним список вершин, которые с ней смежны.

Матрица смежности. Удобный и наглядный, но неэффективный способ хранения.

Для пары вершин i и j значение a_{ij} равно 1 если между этими вершинами есть ребро и 0 если его нет.

Написанное выше – описание в целом. В частных случаях допустимы некоторые модификации, например, если у нас мультиграф, можно хранить не списки смежных вершин, а списки инцидентных данной вершине рёбер. Аналогично, в матрице смежности при необходимости можно вести запись по другому принципу – например, хранить в ней вес соответствующего ребра или список рёбер в каждой ячейке, если у нас мультиграф.

Структуры также могут получаться различными в зависимости от того, работаем мы с ориентированными графами или неориентированными. Можно пытаться смягчить смягчить недостатки матрицы смежности, используя в её основе вложенные сбалансированные деревья поиска, всё зависит от конкретных целей.

Если не оговорено что-то другое, под “дан граф” стоит считать, что дан его список рёбер. При этом в графовых алгоритмах мы почти всегда строим по этому списку списки смежности и работаем уже с ними.

Построение списка смежности по списку рёбер

```
vector<vector<int>> adjacency_list(int V, vector<pair<int, int>> E) {
    vector<vector<int>> G(V); // Массив из пустых списков
    for(auto it: E) {
        int u = it.first;
        int v = it.second;
        G[u].push_back(v); // Добавляет элемент v в конец списка G[u]
        G[v].push_back(u); // Только если граф неориентирован
    }
    return G;
}
```

Обход в ширину

Данный алгоритм также называют поиск в ширину, или на английском *breadth-first search*, сокращённо *BFS*. Приведём реализацию алгоритма, который находит все достижимые из данной вершины.

```
vector<int> reachable(vector<vector<int>> G, int s) { // G -- список смежности, s -- стартовая
    int V = G.size();
    vector<int> visited(V); // visited[i] = 0, для i = 0,...,V - 1
    queue<int> que;
    que.push(s);
    while(not que.empty()) {
        int v = que.back();
        que.pop();
        for(int u: G[v]) {
            if(!visited[u]) {
                visited[u] = true;
                que.push(u);
            }
        }
    }
    return visited; // visited[v] = 1 если v достижимо из s, иначе 0.
}
```

Его основная идея в том, чтобы рассматривать вершины “по уровням” — сначала рассматриваются все вершины на расстоянии 0 (это только s), затем – все на расстоянии 1, затем на расстоянии 2 и так далее.

3. (3б) Рассмотрим следующий вариант поиска в ширину:

```
vector<int> distance(vector<vector<int>> G, int s) {
    int V = G.size();
    vector<int> dist(V, V); // dist[i] = V, для i = 0,...,V - 1
    dist[s] = 0;
    queue<int> que;
    que.push(s);
    while(not que.empty()) {
        int v = que.back();
        que.pop();
        for(int u: G[v]) {
            if(dist[v] + 1 < dist[u]) {
                dist[u] = dist[v] + 1;
                que.push(u);
            }
        }
    }
    return dist; // dist[v] = V если пути из s в v нет.
                 // Иначе -- расстояние от s до v.
}
```

1. Докажите, что в `distance` будут рассмотрены все достижимые из s вершины и они будут рассмотрены в том же порядке, что и в `reachable`, то есть, алгоритмы эквивалентны.
2. Докажите, что в `dist` действительно будут храниться расстояния от s до каждой вершины.
3. Покажите, что обе процедуры работают за $O(E+V)$.

Обход в ширину тесно связан с так называемым деревом обхода в ширину. Мы можем каждой вершине u графа, кроме s сопоставить единственного прямого предка – ту вершину v, при рассмотрении которой мы добавили вершину u в очередь. Это и задаст искомый остов, который по совместительству будет деревом кратчайших путей для вершины s.

Обход в глубину

Также известен как *depth-first search* или *DFS*. Отличается лаконичной рекурсивной реализацией:

```
vector<int> reachable(vector<vector<int>> G, int s) {
    int V = G.size();
    vector<int> visited(V);
    function<void(int)> dfs = [&](int v) { // Этот кусок -- просто способ объявить функцию
                                           // внутри другой функции
        visited[v] = 1;
        for(auto u: G[v]) {
            if(!visited[u]) {
                dfs(u);
            }
        }
    };
    dfs(s);
    return visited;
}
```

В этом алгоритме надо поддерживать список посещённых вершин и “жадно” идти в любую непосещённую вершину как только мы её увидим. Поиск в глубину нельзя использовать для построения дерева кратчайших путей, но он обладает многими нетривиальными свойствами, которые могут быть даже важнее возможности искать кратчайшие пути.

4. (1б) Докажите, что `visited` будет посчитан корректно приведённым алгоритмом, то есть, $visited_i$ будет равен 1 если вершина достижима из s и 0 в противном случае.

Поиск циклов

5. (2б) Предложите алгоритм на основе *BFS* для проверки наличия и нахождения цикла в *неориентированном* графе. Будет ли этот алгоритм работать на ориентированных графах? Предложите алгоритм на основе *DFS* для поиска цикла в ориентированном графе и обоснуйте его.

6. (2б) Докажите, что в случае применения обхода в глубину к неориентированному графу, все рёбра, не вошедшие в дерево обхода в глубину, будут соединять вершину и некоторого её предка в дереве. Приведите пример когда это неверно в случае ориентированных графов.

Эйлеровость графов

Эйлеровым называют граф, содержащий эйлеров цикл. Имеют место следующие утверждения:

1. Неориентированный граф содержит эйлеров цикл $\iff$ граф связен и степени всех вершин чётны.
2. Неориентированный граф содержит эйлеров путь $\iff$ граф связен и степени всех вершин (кроме, быть может, двух) чётны.
3. Ориентированный граф содержит эйлеров цикл $\iff$ граф сильно связен и полустепени захода всех вершин равны их полустепеням исхода.

С алгоритмом поиска эйлерова пути можно ознакомиться по приведённой выше ссылке на конспекты ИТМО.

2°. (3б) Пусть дан *неориентированный* граф G. Предложите полиномиальный алгоритм, разбивающий его на наименьшее число рёберно не пересекающихся путей, которые покрывают все рёбра графа.

Подсказка: Если в графе есть эйлеров путь, то он и будет ответом. А если нет?

Минимальный остов

Минимальное остовное дерево (англ. MST: Minimum Spanning Tree) – это остовное дерево взвешенного графа, такое что суммарный вес входящих в него рёбер минимален. Основные алгоритмы для поиска:

1. Алгоритм Борувки
2. Алгоритм Краскала
3. Алгоритм Прима

7. (2б) Покажите, что остовное дерево является минимальным тогда и только тогда, когда для любого ребра, не входящего в остовное дерево, цикл, получаемый добавлением этого ребра к остовному дереву, не содержит ребра большего веса.

8. (2б) Покажите, что путь между любой парой вершин (u, v) по минимальному остовному дереву будет минимальным среди всех путей в графе если считать, что вес пути равен наибольшему ребру в нём. Верно ли, что любое дерево с таким свойством является минимальным остовом?

Потоки в транспортных сетях

Сеть (англ. flow network) – это ориентированный граф G, в котором каждое ребро (u, v) имеет положительную пропускную способность $c(u, v) > 0$. Если $(u, v) \notin E$, то $c(u, v) = 0$.

В транспортной сети всегда выделен *исток* s и *сток* t.

Потоком f в сети G называют функцию $f : V \times V \to R$, такую что:

1. $f(u, v) = -f(v, u)$
2. $|f(u, v)| \leq c(u, v)$
3. $\sum\limits_{v} f(u, v) = 0$ для всех вершин, кроме s и t

Величиной потока f называют $|f| = \sum\limits_{v} f(s, v)$

Теорема Форда-Фалкерсона

Основной теоремой, позволяющей изучать потоки является т. Форда-Фалкерсона. Она гласит, что следующие утверждения эквивалентны:

1. f – максимальный поток в G
2. В остаточной сети не существует пути из s в t
3. $|f| = c(S, T)$ для некоторого разреза (S, T) сети G

Обязательно разберитесь с определением остаточной сети. И в частности, с тем, что после того, как мы пускаем поток f по ребру (u, v), остаточная пропускная способность (u, v) уменьшается на $|f|$, а остаточная пропускная способность (v, u) увеличивается на ту же величину. За счёт после того, как мы пустим некоторый поток в сети, нам могут стать доступными для прохода некоторые рёбра, которые в начальном графе не были.

Алгоритм Форда-Фалкерсона

Согласно первому и второму пункту теоремы Форда-Фалкерсона, максимальный поток можно искать следующим образом:

1. Найти произвольный путь $s \to t$ в остаточной сети G_f. Если такого нет, то макс. поток уже найден.
2. Выбрать ребро (u, v) с *наименьшей* пропускной способностью в этом пути.
3. "Пустить" поток $c(u, v)$ по этому пути. То есть, для всех рёбер (a, b) уменьшить пропускную способность (a, b) на $c(u, v)$, а для обратных рёбер (b, a) увеличить её на ту же величину.

Данный алгоритм очень неэффективный. В случае целых пропускных способностей он работает за $O(|f| \cdot |E|)$, а для вещественных пропускных способностей он и вовсе можно не сойтись при любом числе итераций.

Алгоритм Эдмондса-Карпа

Чтобы превратить алгоритм Форда-Фалкерсона в полиномиальный, достаточно искать не произвольный путь из истока в сток, а кратчайший, тогда время работы алгоритма станет $O(|E|^2 \cdot V)$.

9. (2б) Согласно теореме Форда-Фалкерсона, наибольший поток в сети равен наименьшему разрезу между вершинами s и t. Предложите полиномиальный алгоритм, находящий сам минимальный разрез.

Теорема о декомпозиции

Любой поток можно представить в виде совокупности $O(E)$ путей и циклов в сети.

3°. (3б) Граф называют k-связным если удалением любых $k-1$ рёбер оставит граф связным. Эквивалентное определение: между любыми двумя вершинами в графе есть хотя бы k рёберно непересекающихся путей. Предложите полиномиальный алгоритм проверки неориентированного графа на k-связность.