Лекция 1.
Мосты. Точки сочленения. Компоненты сильной связности

179 Школа, Овчинников Андрей

26 октября 2022 г.

Аннотация

Перед тем, как вы прочитаете то, что написано ниже обращу внимание на то, что в данном конспекте могут быть допущены ошибки и опечатки, если вы находите подобное, то пишите мне в личку или группу в телеграмм с упоминанием (*то есть через* @). Буду рад, если вы поможете довести конспект до хорошего безошибочного состояния.

Графы и их представление в памяти

Граф - *математическая абстракция, задающаяся множествами вершин и рёбер, каждое из которых связывает две определённые вершины.*
Напомним, что распространены три главных способа представления графов в памяти.

1. Матрица смежности
2. Список смежности
3. Список рёбер

Напомним читателю немного о том, когда применяется каждый из этих способов. Расссмотрим граф с n вершин и m рёбер.

Матрица смежности

Самый простой и естественный способ хранения графа, подходит как для неориентированных, так и для ориентированных. Не позволяет адекватно хранить кратные рёбра. Способ удобен, если необходимо инвертировать граф.
Необходимо $O(n^2)$ памяти.

Список смежности

Наиболее распространённый способ хранения графа. Подходит для любых графов и умеет в кратные рёбра.
Необходимо $O(m)$ памяти.

Список рёбер

Специфичный способ, по понятным причинам не позволяет естественным путём реализовать обход графа, однако очень полезен во множестве алгоритмов и в некоторых задачах по сути является необходимым *ключом* к решению.
Необходимо $O(m)$ памяти.

Дополнительно

Существуют другие способы хранения, в частности, таких способов много для деревьев. Например, деревья могут быть представлены в памяти, как список предков.

Обход графа

Поговорим о самом распространённом (*в моём понимании*) способе обхода графа, а именно любимый и незаменимый *dfs*.

Теория

Введём несколько новых понятий:

- *Дерево обхода в глубину (дерево dfs)* - заметим, что в процессе работы dfs, мы никогда не пройдём по ребру, которое замыкает цикл. Соответственно рассмотрим все рёбра, по которым *'прошёл'* dfs. Из замеченного ранее, эти рёбра будут образовывать дерево.
- *Прямое ребро* - это ребро дерева обхода dfs.
- *Обратное ребро* - это ребро графа, не вошедшее в дерево обхода

Обратим внимание читателя на то, что *обратные рёбра* **не** могут соединять вершины разных (невложенных) поддеревьев дерева обхода.

Реализация dfs

```
void
dfs(int vertex, vector<bool> &used, vector<vector<int>> &graph)
{
    used[vertex] = true;
    for (auto u : graph[vertex]) {
        if (!used[u]) {
            //делаем что-то...
            dfs(u, used, graph);
            //делаем что-то...
        }
    }
}
```

О важных *для данной темы* применениях dfs

Уделим особое внимание топологической сортировке ориентированного ациклического графа.
Топологической сортировкой *данного графа будем называть некоторую перенумерацию вершин, такую, что в новом порядке любой родитель стоит строго раньше (или строго позже) своего ребёнка.*

Реализация *Top Sort*

```
bool
top_sort(
        int vertex,
        vector<int> &ans,
        vector<bool> &used,
        vector<vector<int>> &graph)
{
    used[vertex] = true;
    bool has_cycle = false;
    for (auto u : gr[vertex]) {
        if (used[v]) {
            return true;
        } else {
            has_cycle = (has_cycle || top_sort(u, ans, used, graph));
        }
    }
    ans.push_back(vertex);
    return has_cycle;
}
```

Мосты в графах

Мостом *в данном графе будем называть такое ребро, при удалении которого увеличивается количество компонент связности.*
Поставим следующую задачу: *необходимо найти все мосты в данном графе состоящем из n вершин и m рёбер.*
Предложим несколько вариантов решения.

За $O(m * (n + m))$

Очевидное решение состоит в том, чтобы поочерёдно удалять рёбра, а после проверять граф на количество компонент связности, заранее посчитав их (необходимо, только если изначально компонент было несколько).

За $O(n + m)$

Рассмотрим дерево обхода, которое построил *dfs*. Для начала поймём, почему *обратное ребро* не может быть *мостом*. Это вполне очевидно, так как дерево *dfs* соединяет путём любые две вершины, а никакое обратное ребро не входит в этот путь.

Как же понять, что *прямое ребро* является мостом? Если мы удаляем мост, то граф распадается на большее количество компонент связности, но что же может не дать графу распасться?

Рассмотрим ребро *(u, v)*, пусть вершина v будет предком вершины u в дереве dfs'а, тогда любое обратное ребро из поддерева вершины u в вершину v или предка вершины v.

Теперь наша задача для каждого ребра проверить, имеется ли такое 'подходящее' *обратное ребро.*

Реализация

```
void
find_bridges(
        int vertex, int parent = -1, vector<int> &up,
        vector<int> &deep, vector<int> &used, vector<vector<int>> &graph)
{
    used[vertex] = true;
    if (parent == -1) {
        deep[vertex] = 0;
    } else {
        deep[vertex] = deep[parent] + 1;
    }
    up[vertex] = deep[vertex];
    for (auto u : graph[vertex]) {
        if (u == parent) {
            //ребро в предка от которого запустились
            continue;
        } else if (!used[u]) {
            //ребро в поддерево => надо обойти поддерево,
            //а после пересчитаться от лучшего
            //результата в поддереве, который вернёт нам функция
            find_bridges(u, vertex, up, deep, used, graph);
            up[vertex] = min(up[vertex], up[u]);
        } else {
            //обратное ребро из текущей вершины
            up[vertex] = min(up[vertex], deep[u]);
        }
    }
}
```

Точки сочленения в графе

Точкой сочленения *в данном графе будем называть вершину графа, при удалении которой количество компонент связности возрастает.*

Поставим следующую задачу: *необходимо найти все точки сочленения в данном графе состоящем из n вершин и m рёбер.*

За $O(n * (n + m))$

Просто удаляем, а лучше в массиве *used* помечаем вершину, как использованную. И запускаем обход графа с подсчётом компонент связности. И так для каждой вершины.

За $O(n + m)$

Сразу же опишем оптимальный аглоритм.

Как понять является ли вершина - точкой сочленения? Рассмотрим дерево обхода *dfs*'ом. Тогда вершина является точкой сочленения, если хотя бы одно из поддеревьев её сыновей не имеет связи по *обратному ребру* с вершинами, являющимися предками *рассматриваемой вершины.*

Реализация

Несложно допилить алгоритм для мостов. Только отдельно необходимо проверить корень дерева, который **НЕ** является точкой сочленения только в случае, когда количество сыновей у корня ≤ 1.

Компоненты сильной связности

Компонентой сильной связности *в данном ориентированном графе будем называть множество вершин, такое что можно добраться по рёбрам от любой вершины этого множества до любой другой в этом множестве.*

Поставим следующую задачу: *необходимо выделить все компоненты сильной связности в данном ориентированном графе состоящем из n вершин и m рёбер.*

Реализация

Я устал думать и писать...