ЛКШ, ЛКШ.2018.Август B'
B', конспект лекции

Собрано 10 августа 2018 г. в 10:14

Содержание

1. Дейкстра **1**
- 1.1. Описание алгоритма . . . 1
- 1.2. Доказательство корректности . . . 1
- 1.3. Асимптотика . . . 1
- 1.4. $\mathcal{O}(V^2)$ . . . 1
- 1.5. $\mathcal{O}(V \log V + E \log V)$ . . . 2
- 1.6. K-ичная куча . . . 2

2. Флойд **3**
- 2.1. Описание алгоритма . . . 3
- 2.2. Об отрицательных циклах . . . 3

3. Форд-Беллман **4**
- 3.1. Описание алгоритма . . . 4
- 3.2. Оптимизации ФБ . . . 4
  - 3.2.1. ФБ с очередью (SPFA) . . . 4
  - 3.2.2. ФБ с break . . . 4

4. Примеры задач **5**
- 4.1. Первым делом самолеты, ну а поезда потом . . . 5
- 4.2. Минимальный путь по масимальному ребру . . . 5
- 4.3. Минимальный путь по сумме двух максимальных ребер . . . 5
- 4.4. Получаем N из +a +b +c . . . 5
- 4.5. Обменник . . . 5

Тема #1: Дейкстра

9 августа

1.1. Описание алгоритма

Вспомним BFS. Там мы обрабатывали вершинки по удаленности от стартовой. А работает ли такой алгоритм для произвольных весов ребер? Оказывается да, и он называется алгоритмом Дейкстры.
Алгоритм Дейкстры ищет расстояние от одной вершины до всех на графе без отрицательных ребер.
Псевдокод:

```
d[..] = INF // массив расстояний. Изначально заполнен бесконечностями
d[s] = 0 // расстояние до стартовой вершины ноль
used[..] = False // массив посещенных вершин
while Есть непосещенные, достижимие из s вершины:
    v = непосещенная вершина с минимальным d[v]
    used[v] = True
    for (<u, w> : gr[v]):
        d[u] = min(d[u], d[v] + w)
```

1.2. Доказательство корректности

Докажем, что на каждой итерации алгоритма ответ для v посчитан верно.
Доказывать будем по индукции.
База. Расстояние до s посчитано правильно.
Переход. Пусть расстояние посчитано правильно для какого-то множества вершин S. Хотим доказать, что если взять сейчас вершину такую, для которой текущее расстояние минимально, то это будет правильно посчитанное для нее расстояние.
Пусть это вершина v. Тогда $d[v] = d[u] + w$ для какой-то вершины u из S и ребра $u \to v$ веса w. Пусть это расстояние неверное и существует какой-то путь в эту вершину из s минимальной стоимости $d'[v]$. Тогда этот путь какое-то время идет по вершинам из S, потом выходит из S, а потом идет как-то.
TODO: нужна картинка.
Найдем первый момент, когда этот путь выходит из S. Если он идет в v, то мы победили. Если не в v, то в какую-то вершину t. Тогда $d[t] \geqslant d[v]$. Но $d'[v] = d[t] + path$, где $path \geqslant 0$. Из этих двух неравенств получаем $d'[v] \geqslant d[v] + path$. Но мы предположили, что $d'[v] < d[v]$ и $path \geqslant 0$. Противоречие.

1.3. Асимптотика

Нам нужна структура данных, которая умеет изменять значение элемента в множестве, удалять элемент из множества, находить минимальный элемент во множестве.
Операцию нахождения и удаления минимального элемента назовем $extractMin$, операцию изменения значения элемента $decreaseKey$ (decrease потому что в нашем случае значения элементов уменьшаются).
Асимптотика алгоритма будет $\mathcal{O}(V \cdot extractMin + E \cdot decreaseKey)$

1.4. $\mathcal{O}(V^2)$

Массив умеет делать $extractMin$ за $\mathcal{O}(V)$ (простой проход) и $decreaseKey$ за $\mathcal{O}(1)$ (поменять по индексу).
Получили $\mathcal{O}(V^2 + E)$

1.5. $\mathcal{O}(V \log V + E \log V)$

Куча (set, priority_queue) умеет каждую из этих операций делать за логарифм. Получили $\mathcal{O}(V \log V + E \log V)$.
Код:

```
set<pair<int, int>> Q;
Q.insert({s, 0});
vector<int> dist(V, INF);
dist[s] = 0;
vector<bool> used(V, false);
while (!Q.empty()) {
    auto tmp = *Q.begin();
    Q.erase(Q.begin());
    int v = tmp.second;
    int d = tmp.first;
    used[v] = true;
    for (auto e : gr[v]) {
        if (!used[e.to]) {
            Q.erase({dist[e.to], e.to});
            dist[e.to] = min(dist[e.to], d + e.w);
            Q.insert({dist[e.to], e.to});
        }
    }
}
```

Этот алгоритм можно соптимизировать, если удалять из сета только в случае, когда расстояние уменьшается.

1.6. K-ичная куча

TODO

Тема #2: Флойд

9 августа

2.1. Описание алгоритма

Алгоритм Флойда считает расстояния между всеми парами вершин. Делает он это с помощью динамики $dp[i][j][k]$ — кратчайший путь от i до j, если промежуточные в пути вершины из промежутка $[0, k]$.
Переход: $dp[i][j][k] = min(dp[i][j][k-1], dp[i][k][k-1] + dp[k][j][k-1])$
Заметим, что третья размерность нам не нужна, так как если ее убрать, мы будем перебирать пути из i в j по вершинам из $[0; k]$ и какие-то ещё $\Rightarrow$ ответ не ухудшится.
Получили простой код:

```
for (int k = 0; k < V; ++k) {
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
        }
    }
}
```

Время работы $\mathcal{O}(V^3)$

2.2. Об отрицательных циклах

TODO

Тема #3: Форд-Беллман

9 августа

3.1. Описание алгоритма

Алгоритм Форда-Беллмана умеет вычислять расстояние от одной вершины до всех других. При этом в графе могут быть отричательные ребра.
Будем считать динамику $dp[v][len]$ — длина кратчайшего пути из s в v, проходящего не более чем по len ребрам.
Переход: перебираем ребра из v, пытаемся обнобвить $dp[e.to][len + 1]$
Стандартная оптимизация: если убрать len, ответ не ухудшится.
Сколько итераций нужно? Если в графе нет циклов отрицательного веса, то легко доказать, что $V - 1$ итерации точно хватит.
При реализации проще всего хранить граф в виде списка ребер и пытаться обновить все .

```
1 vector<int> dist(V, INF);
2 vector<Edge> gr(E);
3 dist[s] = 0;
4 for (int i = 0; i < V - 1; ++i) {
5     for (Edge e : gr) {
6         dist[e.to] = min(dist[e.to], dist[e.from] + e.w);
7     }
8 }
```

3.2. Оптимизации ФБ

3.2.1. ФБ с очередью (SPFA)

TODO

3.2.2. ФБ с break

TODO

Тема #4: Примеры задач

9 августа

4.1. Первым делом самолеты, ну а поезда потом

TODO

4.2. Минимальный путь по масимальному ребру

Дейкстра умеет работать не только с суммой, но и с максимумом. Конец.

4.3. Минимальный путь по сумме двух максимальных ребер

TODO

4.4. Получаем N из +a +b +c

TODO

4.5. Обменник

TODO