# Дейкстра. 1-k BFS.

Сапожников Денис

## Contents

# 1 Алгоритм Дейкстры

Дан ориентированный взвешенный граф $G(V, E)$ с рёбрами неотрицательного веса. Алгоритм Дейкстры находит длину кратчайшего пути от вершины $s$ до всех остальных вершин.

## 1.1 Дейкстра за $O(n^2)$

Пусть $U$ — множество, расстояние до которого уже посчитано корректно. На каждой итерации выбирается $u \notin U$ с минимальной оценкой расстояния до $s$. Вершина $u$ добавляется в $U$, после чего происходит релаксация (обновление) оценок расстояния.

Приведём алгоритм, а потом докажем его корректность:

```
vector<int> d(n, INF), p(n, -1), used(n);
d[s] = 0;
for (int i = 0; i < n; ++i) {
  int v = -1;
  for (int j = 0; j < n; ++j)
    if (!used[j] && (v == -1 || d[j] < d[v]))
      v = j;
  used[v] = true;
  for (auto [u, w] : gr[v])
    if (d[u] > d[v] + w) {
      d[u] = d[v] + w;
      p[u] = v;
    }
}
```

*Proof.* Итак, пусть $d_v$ – текущая оценка расстояния от вершины $s$, до вершины $u$. Докажем, по индукции, что для найденной $v$ верно, что $d_v = \rho(v, s)$.

База: в самом начале мы вытаскиваем вершину $s$, для неё действительно $d_s = \rho(s, s) = 0$.

Переход: Пусть для $n$ первых шагов алгоритм сработал верно и на $n+1$ шаге выбрана вершина $u$. Докажем, что в этот момент $d_u = \rho(s, u)$. Для начала отметим, что для любой вершины $v$, всегда выполняется $d_v \geq \rho(s, v)$ (алгоритм не может найти путь короче, чем кратчайший из всех существующих). Пусть $P$ — кратчайший путь из $s$ в $u$, $v$ — первая непосещённая вершина на $P$, $z$ — предшествующая ей (следовательно, посещённая). Поскольку путь $P$ кратчайший, его часть, ведущая из $s$ через $z$ в $v$, тоже кратчайшая, следовательно $\rho(s, v) = \rho(s, z) + w(z, v)$.

По предположению индукции, в момент посещения вершины $z$ выполнялось $d_z = \rho(s, z)$, следовательно, вершина $v$ тогда получила метку не больше чем $d_z + w(z, v) = \rho(s, z) + w(z, v) = \rho(s, v)$, следовательно, $d_v = \rho(s, v)$. С другой стороны, поскольку сейчас мы выбрали вершину $u$, её метка минимальна среди непосещённых, то есть $d_u \leq d_v = \rho(s, v) \leq \rho(s, u)$, где второе неравенство верно из-за ранее упомянутого определения вершины $v$ в качестве первой непосещённой вершины на $P$, то есть вес пути до промежуточной вершины не превосходит веса пути до конечной вершины вследствие неотрицательности весовой функции. Комбинируя это с $d_u \geq \rho(s, u)$, имеем $d(u) = \rho(s, u)$, что и т.д. $\square$

## 1.2 Дейкстра за $O(m \log n)$

Этот алгоритм отличается от предыдущего ровно одной оптимизацией: каждый раз до этого мы пробегались по всем $d$ и искали минимум. Вместо этого будем поддерживать *set* пар $\{d, to\}$, из которого будем вытаскивать минимум.

```cpp
vector<int> d(n, INF), p(n, -1);
d[s] = 0;
set<pair<int, int>> q;
q.insert({ 0, s });
for (int i = 0; i < n; ++i) {
  auto [dist, v] = *q.begin();
  q.erace(q.begin());
  for (auto [u, w] : gr[v])
    if (d[u] > d[v] + w) {
      q.erase({ d[u], u });
      d[u] = d[v] + w;
      p[u] = v;
      q.insert({ d[u], u });
    }
}
```

## 1.3 Оптимизации Дейкстры

1. priority_queue вместо set. Да, асимптотически это ухудшает оценку до $O(m \log m)$, но на практике работает быстрее раза в 2-3.

2. Иногда надо найти кратчайшее расстояние только между парой точек, в таком случае можно запускать Дейкстру от $s$ и $t$ одновременно, тогда Дейкстра будет меньше «расти в ширь».

3. Алгоритм A*. Пусть нас интересует Дейкстра на плоскости и опять между парой вершин. Тогда мы будем поддерживать в $q$ оценки вида $d(v) + dist(p_i, t)$, где $dist$ – Евклидово расстояние. И каждый раз опять же вытаскивать минимум. Это корректно, и не очень сложно доказывается. А ещё это очень ускоряет Дейкстру.

# 2 BFS

## 2.1 Обычный BFS

Наверняка многие знают, что это, но напомнить стоит. Типичная задача — задача о коне. Дан шахматный конь, нужно сказать, за какое минимальное количество ходов он доберётся из стартовой клетки в заданную. Для этого мы заведём структуру данных queue — очередь, которая умеет в 2 типа запросов, которые равны обычной очереди в супермаркете.

- pop — кассир обслужил первого в очереди человека, который после сразу ушёл.
- push — пришёл новый человек в конец очереди.

С помощью такой структуры мы можем легко решить задачу.

Будем поддерживать полуинвариант: в очереди сначала лежат клетки на расстоянии $k$ от заданной, а затем на расстоянии $k+1$. Берём первую клетку из очереди (она на расстоянии k) и добавляем в конец очереди всех её непосещённых соседей (действительно, они на расстоянии $k+1$ и полуинвариант сохранился). Так обходим всё поле и решаем задачу сразу для любой финальной клетки. Сложность решения — $O(nm)$

## 2.2 1-k BFS

Бывает такое, что в задаче нужно найти кратчайшее расстояние от $s$ до всех остальных вершин в неориентированном взвешенном графе, и при этом веса рёбер — целые числа от 1 до $k$. В таком случае иногда быстрее будет работать 1-k BFS.

Так как веса рёбер ограниченны, то максимальный по весу путь равен $(n-1)k$. Заведём $(n-1)k$ очередей. Каждая очередь будет означать список вершин на этом расстоянии. Изначально в 0-й очереди находится стартовая вершина. Кроме того, мы поддерживаем вершин, расстояние до которых уже точно посчитано.

Пусть мы обработали все вершины на расстоянии меньше $L$. Тогда будем вытаскивать из очереди все вершины на расстоянии $L$, если расстояние до неё уже посчитано, то пропускаем вершину иначе расстояние до вершины строго равно $L$ (идея аналогична Дейкстре). От всех вершин $v$, расстояние до которых равно $L$, добавим в очередь всех соседей $u$ с ребром веса $w$ в очередь номер $L+w$.

Оптимизация: заметим, что в каждый момент времени у нас используется не больше $k+1$ очереди, поэтому мы можем создать всего $k+1$ очередь и засовывать новые вершины в очередь номер $(L+w) \bmod (k+1)$.

Время работы алгоритма — $O(nk+m)$ и $O(n+k)$ памяти.

```
vector<queue<int>> q(k + 1);
q[0].push(st);
vector<int> dist(n, -1);
for (int i = 0; i < n * k; ++i) {
  int nq = i % (k + 1);
  while(q[nq].size()) {
    int v = q[nq].front();
    q[nq].pop_front();

    if (dist[v] == -1)
      dist[v] = i;
    else
      continue;

```

```
    for (auto [u, w] : gr[v])
      if (dist[u] == -1)
        q[(nq + w) % (k + 1)].push(u);
  }
}
```

Частным случаем 1-k BFS является 0-1 bfs, он пишется ещё проще и более распространён.

# 3 Ссылки

1. Дейкстра
2. Форд-Беллман
3. Флойд
4. A*
5. 1-k BFS