Дейкстра. Флойд. Форд-Беллман. 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. □

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.

```
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 Алгоритм Флойда

2.1 Алгоритм

Дан граф ориентированный взвешенный граф G. Необходимо найти кратчайшее расстояние между всеми парами вершин, если нет отрицательных циклов.

Алгоритм Флойда — это типичная динамика. Пусть $d_{i,j,k}$ — кратчайший путь из i в j, проходящий по вершинам из множества $\{1, 2, \ldots, k\} \cup \{i\} \cup \{j\}$. Легко понять, что переходы в такой динамике следующие:

$$dp_{i,j,k} = \min(dp_{i,j,k-1}, dp_{i,k,k-1} + dp_{k,j,k-1})$$

А сам алгоритм пишется в 4 строчки:

```
// d[i][j] = INF, if edge does not exists
for (int k = 0; k < n; ++k)
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
```

2.2 Восстанавливаем ответ

Но вот есть у нас Флойд, а можно как-то восстановить кратчайшие пути между каждой парой вершин?

Да, конечно же, можно. Нужно для каждой пары вершин поддерживать информацию о том, через какую вершину мы обновили минимум.

2.3 Отрицательные циклы

Но вот бывает такое, что в графе есть отрицательный цикл, как определить, что он есть?

Если он есть, то на какой-то диагональной клетке появится отрицательное число. Более того, они появятся во всех $d_{i,i}$ таких, что i содержится в отрицательном цикле, но могут появиться ещё и в вершинах, достижимых из такого отрицательного цикла.

Прошлый пункт помогает даже восстановить какой-нибудь отрицательный цикл.

3 Алгоритм Форда-Беллмана

3.1 Алгоритм

Дан граф ориентированный взвешенный граф G. Необходимо найти кратчайшее расстояние от s до всех остальных вершин при условии, что в графе нет отрицательных циклов.

Обратите внимание, тут разрешаются отрицательные рёбра, в отличии от Дейкстры.

Пусть $d_{v,i}$ – это кратчайшее расстояние от s до v за не более чем i шагов. Изначально $d_s = 0, d_{v \neq s} = +\infty$. Переходы очень простые:

$$d_{v,i} = \min \begin{cases} \min\limits_{(v,u) \in E} d_{u,i-1} + w(v,u) \\ d_{v,i-1} \end{cases}$$

Такая динамика считается за $O(nm)$ и требует $O(n)$ памяти, так как мы каждый раз обращаемся только к предыдущему слою, при этом нам достаточно посчитать $n-1$ слой (самый длинный по количеству вершин простой путь — состоит из n вершин и $n-1$ ребра)

Реализация:

```
vector<int> d(n, INF);
d[s] = 0;
for (int iter = 0; iter < n - 1; ++iter)
  for (int v = 0; v < n; ++v)
    for (auto [u, w] : gr[v])
      d[v] = min(d[v], d[u] + w);
```

3.2 Восстановление ответа

Ничего не мешает нам обновлять информацию о том, откуда мы обновили минимум — p_v. Так можно легко восстановить путь.

3.3 Отрицательные циклы

А давайте сделаем n итераций алгоритма. Тогда если на n-й что-то поменяется в массиве d, то это значит, что есть отрицательный цикл.

3.4 Больше, чем отрицательные циклы

А что если мы хотим решить следующую задачу классификации вершин:

1. Вершина лежит в достижимом отрицательном цикле
2. Вершина достижима из достижимого отрицательного цикла
3. Вершина не достижима
4. Вершина достижима, не содержится в отрицательном цикле и не достижима из него, то есть для неё корректно определено понятие расстояния

Оказывается, это NP-сложная задача. А всё из-за пункта 1 и 2. А вот если не пытаться разделить эти 2 пункта в разные, то задача решаема за $O(nm)$.

То есть хотим классифицировать следующим образом:

1. Вершина лежит в достижимом отрицательном цикле или достижима из достижимого отрицательного цикла
2. Вершина не достижима
3. Вершина достижима, не содержится в отрицательном цикле и не достижима из него

Всё что нам нужно сделать — это дополнительные n итераций Форда-Беллмана. Тогда гарантированно для всех вершин, достижимых из отрицательного цикла уменьшится расстояние относительно $(n-1)$-й итерации.

4 Ссылки

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