

Курс kiev-clrs – Лекция 19. Кратчайшие пути:
все пары вершин, умножение матриц, алгоритм
Флойда-Маршала

Олег Смирнов

25 июля 2009 г.

Содержание

1	Цель лекции	2
2	Кратчайшие пути между всеми парами вершин	3
3	Динамическое программирование	3
4	Умножение матриц	5
5	Улучшенный алгоритм умножения матриц	6
6	Алгоритм Флойда-Уоршелла	6
7	Транзитивное замыкание графа	7
8	Алгоритм Джонсона	8

1 Цель лекции

- Поиск кратчайших путей между всеми вершинами графа
- Смежные задачи (транзитивное замыкание и др.)

Для поиска всех кратчайших путей в графе из одной вершины используются такие алгоритмы:

- ненагруженный граф
 - поиск в ширину (BFS): $O(E + V)$
- граф с неотрицательными дугами
 - алгоритм Дейкстры: $O(E + V \lg V)$
- общий случай
 - алгоритм Беллмана-Форда $O(VE)$
- направленный ациклический граф
 - топологическая сортировка и один проход алгоритма Беллмана-Форда: $O(V + E)$

Алгоритм Дейкстры работает за почти линейное время, а алгоритм Беллмана-Форда – как минимум за квадратичное.

Для того, чтоб найти кратчайшие пути из каждой вершины в каждую, нужно выполнить соответствующий алгоритм для каждой вершины, т.е. $|V|$ раз:

- ненагруженный граф
 - поиск в ширину $|V|$ раз: $O(VE + V^2)$
- граф с неотрицательными дугами
 - алгоритм Дейкстры $|V|$ раз: $O(VE + V^2 \lg V)$
- общий случай
 - алгоритм Беллмана-Форда $|V|$ раз: $O(V^2E)$
 - известно три алгоритма, которые будут рассмотрены ниже

2 Кратчайшие пути между всеми парами вершин

Для ориентированного нагруженного графа $G = (V, E)$, где $V = 1, 2, \dots, n$ с заданной функцией весов $w : E \rightarrow \mathbb{R}$, найти матрицу $n \times n$ кратчайших путей $\delta(i, j)$ для всех $i, j \in V$.

Идея: выполнить алгоритм Беллмана-Форда для каждой вершины. Время работы: $O(V^2E)$. Тогда для плотного графа (в котором n^2 дуг), время выполнения будет $\Theta(n^4)$.

3 Динамическое программирование

Рассмотрим матрицу смежности $A = (a_{ij})$ размерности $n \times n$ для ориентированного графа, где

$$a_{ij} = \begin{cases} w(i, j), & \text{если дуга } (i, j) \in E \\ \infty, & \text{иначе} \end{cases}$$

Для использования динамического программирования, нужно определить оптимальную подструктуру (задачи меньшего размера). Алгоритм Беллмана-Форда на каждом i -м шаге строит кратчайшие пути длины i . Это было доказано в виде леммы на предыдущей лекции.

Можно определить:

$$d_{ij}^{(m)} = \text{вес кратчайшего пути из } i \text{ в } j, \text{ используя } \leq m \text{ дуг}$$

Например, при $m = 0$:

$$d_{ij}^{(0)} = \begin{cases} 0, & \text{если } i = j \\ \infty, & \text{если } i \neq j \end{cases}$$

Из алгоритма Беллмана-Форда известно, что начиная с $n - 1$:

$$d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \dots = \delta(i, j)$$

если в графе нет циклов отрицательного веса. Т.е. ответ будет получен не более чем за n итераций.

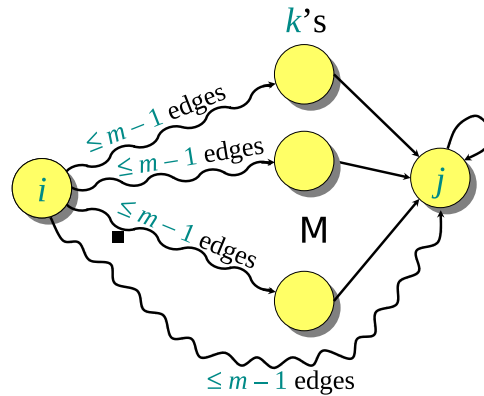


Рис. 1: Алгоритм Беллмана-Форда

Формула для рекурсивного вычисления $d_{ij}^{(m)}$:

$$d_{ij}^{(m)} = \min_k \{d_{ik}^{(m-1)} + a_{kj}\}$$

Очевидно, что формула перебирает все возможные пути в j , рассматривая только смежные вершины. Если существует путь без промежуточной смежной вершины (длиной $m-1$), то он тоже будет рассмотрен, т.к. вес дуги из j в j считается равным 0.

Алгоритм является очередным примером использования релаксации:

```

N_BELLMAN_FORD( $G, w$ )
1  for  $m \leftarrow 1$  to  $n-1$ 
2      do for  $i \leftarrow 1$  to  $n$ 
3          do for  $j \leftarrow 1$  to  $n$ 
4              do for  $k \leftarrow 1$  to  $n$ 
5                  do if  $d_{ij} > d_{ik} + a_{kj}$  // шаг релаксации
6                      then  $d_{ij} \leftarrow d_{ik} + a_{kj}$ 
7  return  $d$ 

```

Время работы: $O(n^4)$, т.е. такое же, как и у алгоритма Беллмана-Форда в худшем случае.

4 Умножение матриц

Алгоритм поиска кратчайших путей можно улучшить с помощью умножения матриц.

Для матриц C , A и B размера $n \times n$, произведение $C = A \cdot B$:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

выполняется за время $\Theta(n^3)$ стандартным алгоритмом.

Если заменить операцию сложения “+” на взятие минимума “min”, а умножение “.” на “+”, то можно формула умножения

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}$$

будет иметь такую же форму, как

$$d_{ij}^{(m)} = \min_k \{d_{ik}^{(m-1)} + a_{kj}\}$$

Таким образом, вычисление матрицы путей можно представить в виде:

$$D^{(m)} = D^{(m-1)} \otimes A$$

Единичная матрица:

$$A^0 = I = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^0 = (d_{ij}^{(0)})$$

Операция \otimes обладает свойством **ассоциативности**, т.к. $(\mathbb{R}, \min, +)$ – **замкнутое полукольцо**, (\mathbb{R}, \min) – коммутативный моноид, $(\mathbb{R}, +)$ – моноид.

Вычисление матрицы путей D является возведением в степень матрицы A :

$$D^{(1)} = D^{(0)} \otimes A = A^1 \tag{1}$$

$$D^{(2)} = D^{(1)} \otimes A = A^2 \tag{2}$$

$$\dots \tag{3}$$

$$D^{(n-1)} = D^{(n-2)} \otimes A = A^{n-1} \tag{4}$$

$$D^{(n-1)} = (\delta(i, j)) \tag{5}$$

Время работы: $\Theta(n \cdot n^3) = \Theta(n^4)$ – такое же как и для n вызовов Беллмана-Форда.

5 Улучшенный алгоритм умножения матриц

Использовать алгоритм Страссена в этом случае нельзя, т.к. этот алгоритм использует вычитание – обратную операцию к сложению. В полукольце $(\mathbb{R}, \min, +)$ обратной операции для \min не существует (иначе это было бы кольцо).

Однако можно использовать возведение в степень методом “разделяй-и-властвуй” – последовательное возведение в квадрат.

Необходимо вычислить

$$A^{2^{\lceil \lg(n-1) \rceil}}$$

так как $A^{n-1} = A^n = A^{n+1} = \dots$

Это можно сделать за $\Theta(n^3 \lg n)$ итераций. Чтоб обнаружить циклы отрицательного веса, после выполнения алгоритма нужно проверить диагональ на наличие отрицательных чисел. Это можно сделать за время $O(n)$.

6 Алгоритм Флойда-Уоршелла

Решение динамическим программированием можно сделать еще лучше. Идея в том, чтоб определить подзадачи более эффективным способом. Вместо вычисления минимума из k значений, можно вычислять минимум из двух. Тогда время работы алгоритма станет кубическим.

Пусть c_{ij}^k – вес кратчайшего пути из i в j , используя только вершины из множества $\{1, 2, \dots, k\}$.

Тогда $\delta(i, j) = c_{ij}^{(n)}$ и $c_{ij}^{(0)} = a_{ij}$.

На шаге релаксации проверяется условие: станет ли путь из i в j короче, если в него добавить промежуточную вершину с номером k ?

$$c_{ij}^{(k)} = \min_k \{c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\}$$

Все промежуточные вершины на рисунке – из множества $\{1, 2, \dots, k\}$.

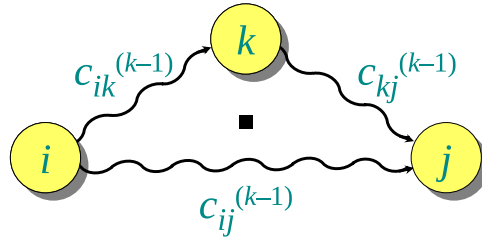


Рис. 2: Алгоритм Флойда-Уоршелла

```

FLOYD_WARSHALL( $G, w$ )
1  $C \leftarrow A$ 
2 for  $k \leftarrow 1$  to  $n$ 
3     do for  $i \leftarrow 1$  to  $n$ 
4         do for  $j \leftarrow 1$  to  $n$ 
5             do if  $c_{ij} > c_{ik} + c_{kj}$  // шаг релаксации
6                 then  $c_{ij} \leftarrow c_{ik} + c_{kj}$ 
7 return  $d$ 

```

Алгоритм выполняется за $\Theta(n^3)$ итераций и весьма эффективен на практике. В случае, если граф не плотный, алгоритм намного лучше Беллмана-Форда. Отрицательные циклы графа можно обнаружить, например, выполнив еще один раунд алгоритма.

7 Транзитивное замыкание графа

Необходимо найти транзитивное замыкание графа, т.е. вычислить матрицу T , где

$$t_{ij} = \begin{cases} 1, & \text{если есть путь из } i \text{ в } j \\ 0, & \text{иначе} \end{cases}$$

Можно выполнить алгоритм Флойда-Уоршелла, используя операции (\vee, \wedge) вместо $(\min, +)$:

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

Время выполнения: $\Theta(n^3)$. Т.к. операции образуют кольцо на множестве элементов $\{0, 1\}$, можно использовать алгоритм Страссена, который выполняется за субкубическое время.

8 Алгоритм Джонсона

Алгоритм Дейкстры очень удобен для получения всех кратчайших путей в графе, т.к. для n вершин он выполняется за: $O(VE + V^2 \lg V)$ – почти квадратичное время. Для его использования нужно предварительно избавиться от дуг отрицательного веса.

Просто добавить к весу каждой дуги некоторую константу не получится, т.к. разные пути содержат разное количество дуг. Вместо этого можно назначить “вес” каждой вершине.

Теорема: используя функцию $h : V \rightarrow \mathbb{R}$, можно “перевзвесить” каждую дугу $(u, v) \in E$ задав $w_h(u, v) = w(u, v) + h(u) - h(v)$. Тогда для любых двух вершин все пути между ними будут “перевзвешены” на одинаковое значение.

Пусть $p = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$ – путь в графе G . Тогда:

$$\begin{aligned} w_h(p) &= \sum_{i=1}^{k-1} w_h(v_i, v_{i+1}) = \\ &= \sum_{i=1}^{k-1} (w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1})) = \\ &= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) + h(v_1) - h(v_k) = \\ &= w(p) + \underbrace{h(v_1) - h(v_k)}_{\text{константа} \geq 0} \end{aligned}$$

Следствие:

$$\delta_h(u, v) = \delta(u, v) + h(u) - h(v)$$

Если найти функцию $h : V \rightarrow \mathbb{R}$, такую, что $w_h(u, v) \geq 0$ для всех $(u, v) \in E$, то можно выполнить алгоритм Дейкстры для каждой вершины графа с измененными весами. Функция должна сделать веса всех дуг неотрицательными.

Можно заметить, что:

$$w_h(u, v) \geq 0 \iff h(v) - h(u) \leq w(u, v)$$

Эта задача разностных ограничений (частный случай задачи линейного программирования), которую можно решить с помощью алгоритма Беллмана-Форда.

Схема алгоритма Джонсона:

1. Найти функцию $h : V \rightarrow \mathbb{R}$ такую, что $w_h(u, v) \geq 0$ для всех $(u, v) \in E$. Это можно сделать с помощью алгоритма Беллмана-Форда, решив задачу разностных ограничений $h(v) - h(u) \leq w(u, v)$.
 - Время: $O(VE)$
2. Выполнить алгоритм Дейкстры, используя функцию w_h для каждой вершины $u \in V$, чтобы вычислить $\delta_h(u, v)$ для всех $v \in V$.
 - Время: $O(VE + V^2 \lg V)$
3. Для всех $(u, v) \in V \times V$ вычислить $\delta(u, v) = \delta_h(u, v) - h(u) + h(v)$
 - Время: $O(V^2)$

Тогда общее время работы алгоритма Джонсона: $O(VE + V^2 \lg V)$ – практически квадратичное.

Алгоритм работает только для графов, в которых нет циклов отрицательного веса.