Курс kiev-clrs – Лекция 17. Кратчайшие пути: алгоритм Дейкстры, поиск в ширину

Олег Смирнов

11 июля 2009 г.

Содержание

1 Цель лекции

- Задача кратчайшего пути в нагруженном графе
- Поиск в ширину

2 Задача кратчайшего пути

Задач поиска кратчайшего (минимального) пути в графе является приложением динамического программирования и жадных алгоритмов.

В ориентированном графе $G = (V, E)$ с весами дуг заданными как $w : E \to \mathbb{R}$, направленный путь $p = v_1 \to v_2 \to \ldots \to v_k$ имеет вес

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

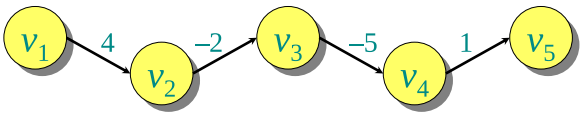

Рис. 1: Путь с весом $w(p) = -2$

Крачайший путь из u в v – это путь с минимальным возможным весом из u в v:

$$\delta(u, v) = \min\{w(p) : \text{ из } u \text{ в } v\}$$

Кратчайшие пути могут не существовать, когда в графе есть дуги с отрицательным весом. В этом случае $\delta(u, v) = -\infty$.

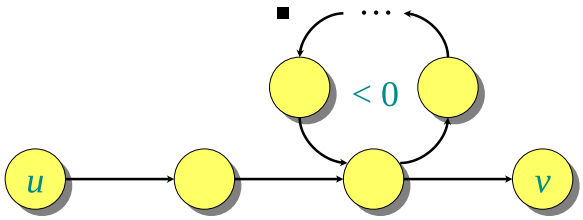

Рис. 2: Отрицательный путь

Отрицательный цикл в графе можно обходить бесконечное количество раз, уменьшая вес пути.

Кратчайшие пути могут не существовать, если граф несвязный. В этом случае, если путь из u в v отсутствует, обозначают $\delta(u, v) = \infty$.

Для реализации алгоритма поиска кратчайших путей нужно доказать две леммы о свойствах кратчайших путей.

1. Свойство оптимальной подструктуры: подпуть кратчайшего пути тоже является кратчайшим путём.

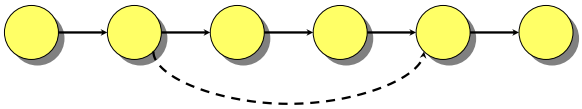

Рис. 3: Оптимальная подструктура

Допустим существует подпуть, которые короче рассматриваемого. Тогда удалив из исходного пути рассматриваемый и добавив более короткий, мы получим путь, который оптимальный исходного. Это противоречие доказывает лемму.

2. Неравенство треугольника: для всех вершин $u, v, x \in V$ выполняется неравенство

$$\delta(u, v) \leqslant \delta(u, x) + \delta(x, v)$$

Можно увидеть, что по определению, $\delta(u, v)$ является кратчайшим

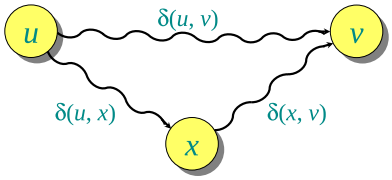

Рис. 4: Неравенство треугольника

из всех возможных путей из u в v, а значит он короче и пути из u в v через x (сумма в правой части неравнства).

3 Кратчайшие пути из одной вершины

Задача: найти длины всех кратчайших путей из заданной вершины $s \in V$ (исток) во все вершины $\delta(s, v)$ для $v \in V$.

Известно, что задача поиска кратчайшего пути между двумя вершинами такая же по сложности, как задача поиска из одной вершины во все.

Для упрощения задачи допустим, что в графе отсутствуют дуги отрицательного веса: $w(u, v) \geqslant 0$, $\forall u, v \in V$. Это означает, что кратчайшие пути существуют и $\delta(u, v) > -\infty$.

Идея: использовать жадный алгоритм.

- на каждом шаге поддерживается инвариант – множество S вершин, для которых уже известны длины кратчайших путей из истока s (в начале $s \in S$)
- на каждом шаге в множество S добавляется вершина $v \in V - S$, для которой оценка дистанции из s минимальна
- после этого происходит обновление (релаксация) оценок путей для вершин, смежных с v

Идея алгоритма в том, что из исходного графа выделяется подмножество вершин, кратчайшие расстояния к которым до s мы уже знаем. В отдельной структуре хранятся оценки расстояния от этого облака к остальному графу. Если оценка неизвестна, то она принимается равной ∞. На каждом шаге алгоритм выбирает из графа ту вершину, которая “ближе” (оценка расстояния минимальна) – это “жадный” выбор. После того, как эта вершина добавляется к множеству известных, оценки расстояний до других вершин корректируются (релаксируются).

4 Алгоритм Дейкстры

SHORTEST_PATHS(V)

```
1  d[s] ← 0
2  for each v ∈ V − {s}
3        do d[v] ← ∞ //d[x] – оценка пути из s в x
   //d[x] будет равно дистанции δ(s, x) если x ∈ S
4  S ← ∅
5  Q ← V //Q – очередь с приоритетом для V − S,
   //где ключём является d[v]. В начале содержит все вершины
6  while Q ≠ ∅
7        do u ← Extract_Min(Q)
8           S ← S ∪ {u}
9           for each v ∈ Adj[u]
10                do if d[v] > d[u] + w(u, v) //шаг релаксации
11                      then d[v] ← d[u] + w(u, v)
12 return d
```

Для доказательства корректности нужно показать, что шаг релаксации находит все кратчайшие пути в графе. Его условие является неравенством треугольника, доказанным выше. Внутри шага релаксации при уменьшении значения ключа $d[v]$ происходит реорганизация очереди с приоритетом.

Первый шаг – инициализация алгоритма: $S = \{\}$

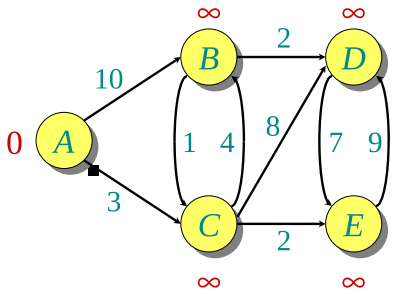

Рис. 5: Шаг 1

Q	**A**	**B**	**C**	**D**	**E**
	0	∞	∞	∞	∞

На втором шаге: $A \leftarrow Extract\_Min(Q)$.

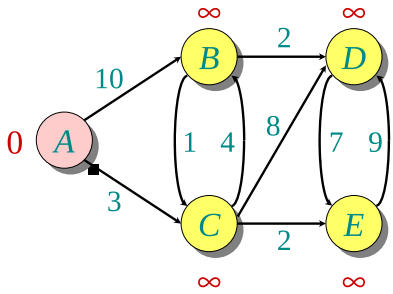

Рис. 6: Шаг 2

Q	A	**B**	**C**	**D**	**E**
	0	∞	∞	∞	∞

$$S = \{A\}$$

Релаксация дуг, смежных с A:

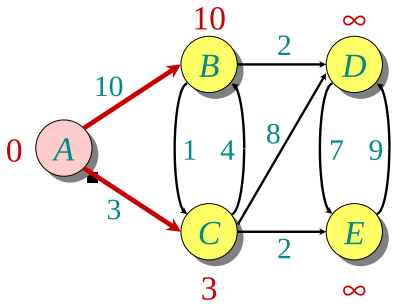

Рис. 7: Шаг 2.1

Q	A	**B**	**C**	**D**	**E**
	0	∞	∞	∞	∞
		10	3	∞	∞

Шаг три: минимальный приоритет в очереди имеет элемент $C \leftarrow Extract\_Min(Q)$.

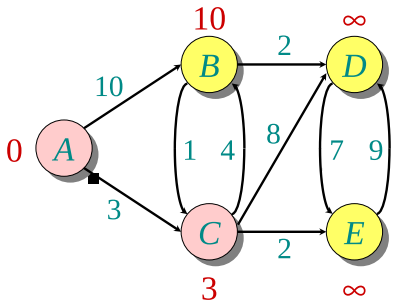

Рис. 8: Шаг 3

Q	A	**B**	C	**D**	**E**
	0	∞	∞	∞	∞
		10	**3**	∞	∞

$$S = \{A, C\}$$

Релаксация дуг, смежных с C:

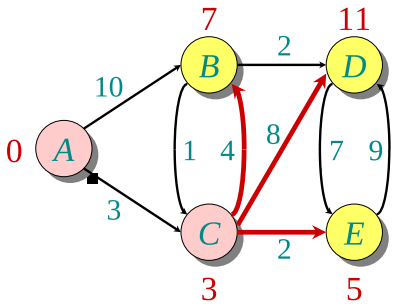

Рис. 9: Шаг 3.1

Q	A	**B**	C	**D**	**E**
	0	∞	∞	∞	∞
		10	**3**	∞	∞
		7		11	5

Шаг четыре: $E \leftarrow Extract\_Min(Q)$

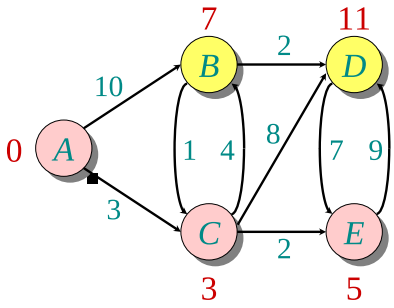

Рис. 10: Шаг 4

Q	A	**B**	C	**D**	E
	0	∞	∞	∞	∞
		10	**3**	∞	∞
		7		11	**5**

$$S = \{A, C, E\}$$

Релаксация дуг, смежных с E:

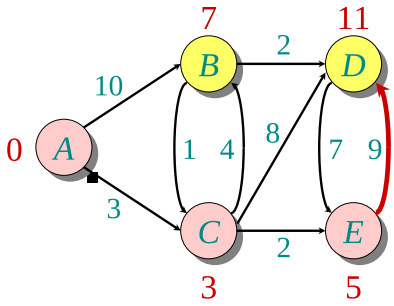

Рис. 11: Шаг 4.1

Q	A	**B**	C	**D**	E
	0	∞	∞	∞	∞
		10	**3**	∞	∞
		7		11	**5**
		7		11	

$B \leftarrow Extract\_Min(Q)$.

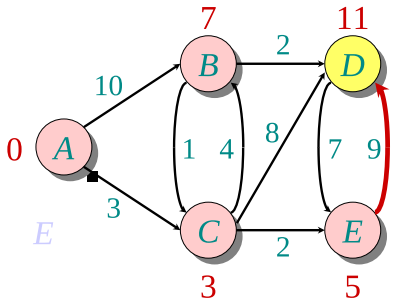

Рис. 12: Шаг 5

Q	A	B	C	**D**	E
	0	∞	∞	∞	∞
		10	**3**	∞	∞
		7		11	**5**
		7		11	

$S = \{A, C, E, B\}$

Релаксация дуг, смежных с B:

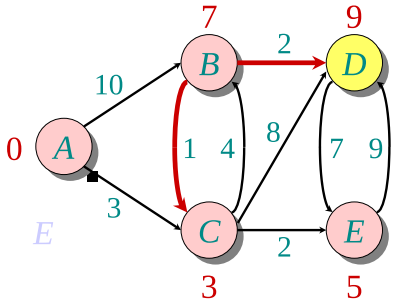

Рис. 13: Шаг 5.1

Q	A	B	C	**D**	E
	0	∞	∞	∞	∞
		10	**3**	∞	∞
		7		11	**5**
		7		11	
				9	

$D \leftarrow Extract\_Min(Q)$.

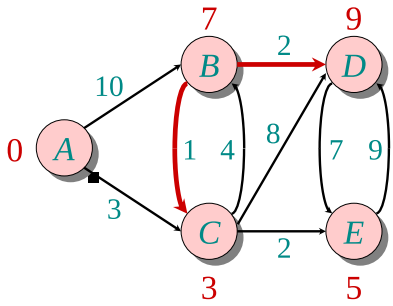

Рис. 14: Шаг 6

Q	A	B	C	D	E
	0	∞	∞	∞	∞
		10	**3**	∞	∞
		7		11	**5**
		7		11	
				9	

$$S = \{A, C, E, B, D\}$$

Если алгоритм корректен,то длины путей, содержащиеся в массиве d – минимально возможные в данном графе.

Для того, чтоб найти минимальные пути в графе, нужно рассмотреть для каждой вершины v последнюю дугу(u, v), которая подвергалась релаксации. Совокупность таких дуг образует дерево с вершиной в s и уникальными путями вниз до каждой из вершины v. Каждый путь в дереве соответствует минимальному в графе.

5 Корректность алгоритма

Корректность алгоритма доказывается по частям.

1. Отсутствие ошибок в цикле релаксации: $d[v]$ всегда служит верхней оценкой $\delta(s, v)$, т.е. инвариант $d[v] \geqslant \delta(s, v)$ выполняется для всех $v \in V$ на каждом из шагов после инициализации.

   Доказательство по индукции.

- Базовый случай: $d[s] \leftarrow 0$ и $d[v] \leftarrow \infty$, $\forall v \neq s$ выполняется, т.к. $\delta(s, s) = 0$ и $\delta(s, v) \leqslant \infty$
- Предположим, что инвариант не выполняется на одном из шагов. Пусть $d[v] < \delta(s, v)$ было получено на шаге релаксации $d[v] \leftarrow d[u] + w(u, v)$. Тогда:

$$\begin{aligned} d[u] + w(u, v) &\geqslant \\ \delta(s, u) + w(u, v) &\geqslant \\ \delta(s, u) + \delta(u, v) &\geqslant \\ &\geqslant \delta(s, v) \end{aligned}$$

  Противоречие доказывает лемму. Идея в том, что каждое присваивание на шаге релаксации оперирует с реальным путем в графе, а длина каждого пути всегда больше или равна длине кратчайшего пути.

2. Кратчайшие пути: пусть $s \to \ldots \to u \to v$ – кратчайший путь из s в v и пусть $d[u] = \delta(s, u)$. Тогда после шага релаксации дуги (u, v) выполнится $d[v] = \delta(s, v)$.

   По первой лемме, $d[v] \geqslant \delta(s, v)$. Если $d[v] > \delta(s, v)$, то выполнится условие релаксации и тогда $d[v] \leftarrow d[u] + w(u, v)$, т.е. $d[v] = \delta(s, v)$.

3. Завершение алгоритма: когда алгоритм Дейкстры заканчивает работу, $d[v] = \delta(s, v)$ для всех $v \in V$.

   Легко увидеть, что $d[v]$ не изменяется после того, как $v \in S$. Допустим, что перед добавлением u в S: $d[u] \neq \delta(s, u)$. Следовательно, по первой лемме $d[u] > \delta(s, u)$.

   Пусть p – кратчайший путь из s в u. Т.е. $w(p) = \delta(s, u)$. Рассмотрим первую дугу (x, y), в которой путь p выходит за пределы S.

$$\begin{aligned} &\text{Т.к. } x \in S: \\ &d[x] = \delta(s, x) \\ &\text{При релаксации по первой лемме:} \\ &d[y] = \delta(s, y) \leqslant \\ &\text{Т.к. путь до } y \text{ является подпутем до } u: \\ &\leqslant \delta(s, u) \\ &\text{Но по правилу выбора из очереди:} \\ &d[u] \leqslant d[y] \end{aligned}$$

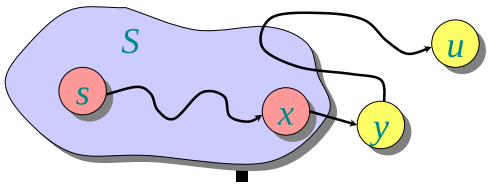

Рис. 15: Корректность завершения

6 Анализ алгоритма Дейкстры

1. Время инициализации алгоритма линейно: $O(V)$
2. Цикл с $Extract\_Min$ выполняется для каждой вершины, время работы: $O(V)$
3. Цикл для смежных вершин выполнятеся за $O(degree(u))$ итераций
4. Согласно лемме о рукопожатиях, цикл выполнит $\Theta(E)$ релаксаций

Время работы $T = \Theta(V) \cdot T_{Extract\_Min} + \Theta(E) \cdot T_{Decrease\_Key}$

Q	$T_{Extract\_Min}$	$T_{Decrease\_Key}$	Всего
массив	$O(V)$	$O(1)$	$O(V^2)$
двоичная куча	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Фиббоначева куча	$O(\lg V)$	$O(1)$	$O(E + V \lg V)$

7 Поиск в ширину

В ненагруженных графах, где можно считать что веса всех дуг $w(u, v) = 1$, алгоритм Дейкстры можно упросить, используя простую очередь (FIFO) вместо очереди с приоритетами.

Модификацией алгоритма для этого случая является поиск в ширину (Breadth-first search или BFS):

```
1  while Q ≠ ∅
2        do u ← Deq(Q)
3           for each v ∈ Adj[u]
4                 do if d[v] = ∞
5                       then d[v] ← d[u] + 1
6                            Enq(Q, v) //v в конец очереди
```

На каждом шаге, если вершина еще не была посещена ($d[v] = \infty$), то кратчайший путь до неё равен пути до предшественника u плюс единица. Затем v добавляется в конец очереди.

В начале работы очередь Q содержит только начальную вершину s.

Время работы алгоритма $T = O(V + E)$.

Инвариант алгоритма: на каждом шаге v находится в очереди после u, откуда следует, что $d[v] = d[u]$ или $d[v] = d[u] + 1$.