Введение в дискретную математику и математическую логику

Лекция №8
Поиск кратчайшего пути из одного источника

- Апанович Зинаида Владимировна

Взвешенные графы

Многие проблемы можно смоделировать с помощью графов, ребрам которых присвоены веса.

Например, их можно использовать для моделирования системы авиаперевозок .

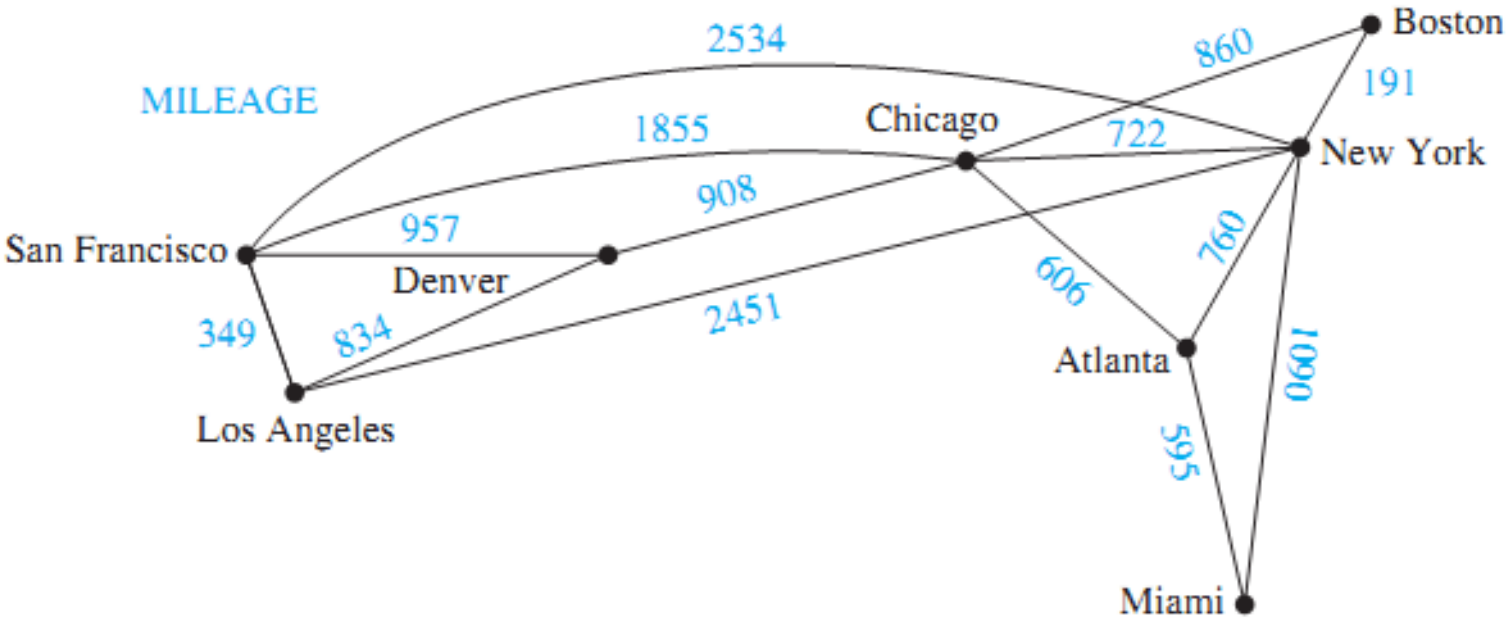
Мы создаем базовую модель графа, представляя города вершинами, а рейсы — ребрами.

Задачи, связанные с расстояниями, можно моделировать, присваивая ребрам расстояния между городами.

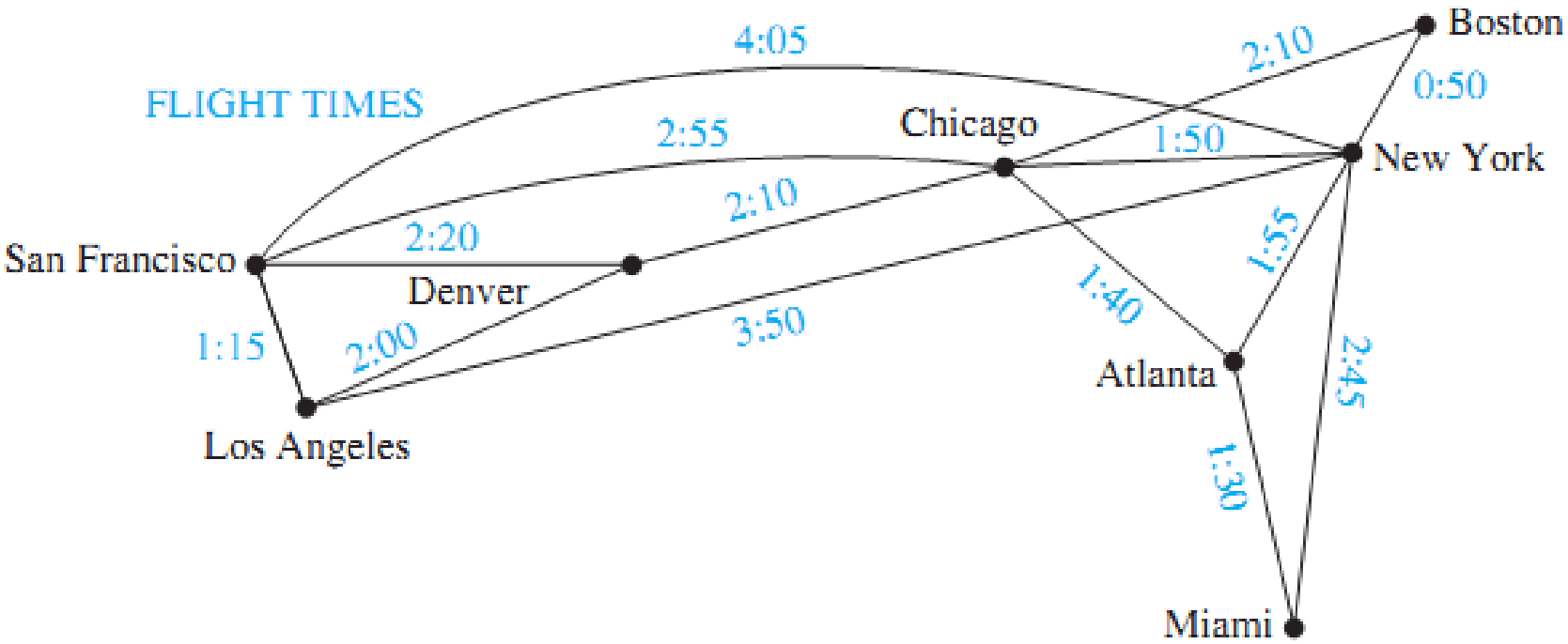
Задачи, связанные со временем полета, можно моделировать, присваивая время полета.

Проблемы, связанные с тарифами, можно моделировать, присваивая ребрам тарифы.

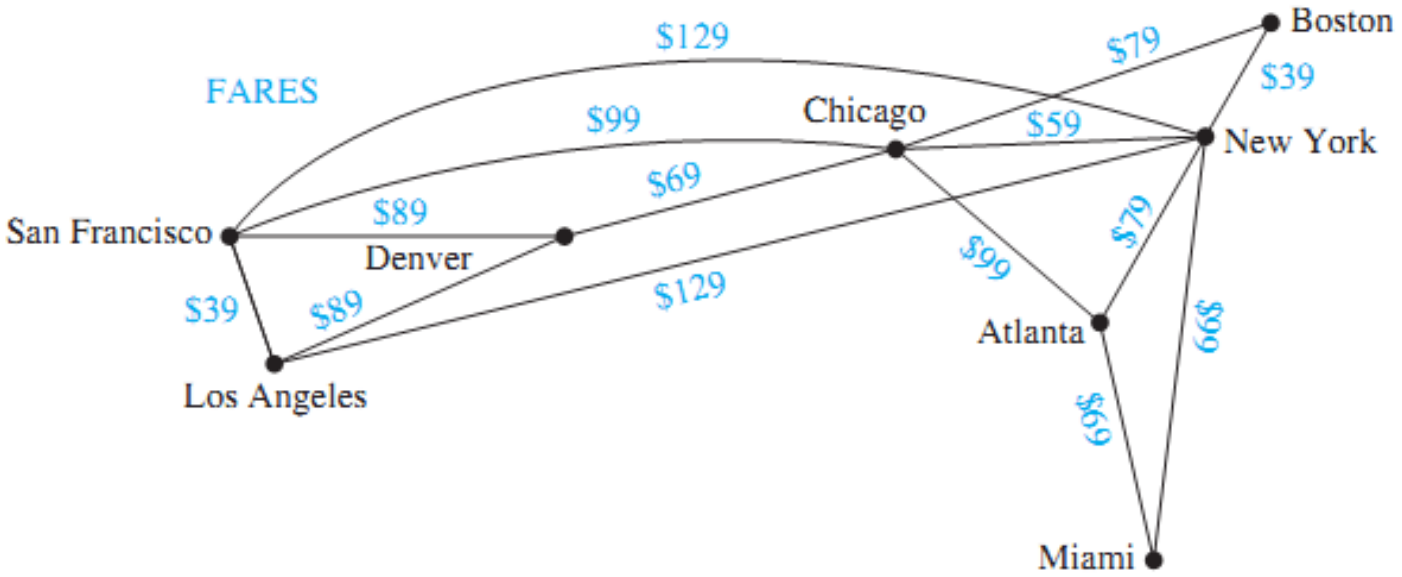
Расстояния между городами



Время полета между городами



Стоимость поездки между городами



Поиск кратчайшего пути во взвешенных графах

Взвешенные графы можно использовать для моделирования компьютерных сетей:

- расходы на связь (например, ежемесячная стоимость аренды телефонной линии),
- время отклика компьютеров по этим линиям,
- или расстояние между компьютерами,

все эти проблемы можно изучать с помощью взвешенных графов.

Поиск кратчайшего пути во взвешенных графах

В задаче о кратчайших путях нам дан взвешенный ориентированный граф $G = (V, E)$ с функцией веса

$w : E \to R$, отображающей ребра в вещественные веса.

Вес $w(p)$ пути $p = (v_0 , v_1 , \ldots, v_k)$ представляет собой сумму весов составляющих его ребер:

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

Поиск кратчайшего пути во взвешенных графах

Мы определяем вес кратчайшего пути $\delta(u, v)$ из вершины *u* *в* вершину *v* по формуле:

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise.} \end{cases}$$

Кратчайший путь от вершины *u* до вершины *v* тогда определяется как любой путь *p* с весом $w(p) = \delta(u, v)$.

Поиск кратчайшего пути во взвешенных графах

Веса ребер могут представлять собой значения, отличные от расстояний, такие как, например, время, стоимость, штрафы, потери или любую другую величину, которая линейно накапливается вдоль пути, и которую мы хотели бы минимизировать.

Поиск кратчайшего пути во взвешенных графах. Варианты

Рассмотрим задачу поиска кратчайших путей из одного источника:

дан граф $G = (V, E)$, мы хотим найти кратчайший путь из заданной исходной вершины $s \in V$

в каждую вершину $t \in V$.

Алгоритм для задачи с одним источником может решать множество других задач, включая следующие варианты.

Задача о кратчайших путях в заданный пункт назначения

Найти кратчайший путь в заданную вершину *t* из каждой вершины *v*.

Изменив направление каждого ребра в графе, мы можем свести эту задачу к задаче с одним источником.

Задача о кратчайшем пути для заданной пары вершин *u* и *v*

Найти кратчайший путь из *u в v* для пары вершин *u* и *v* .

Если мы решим задачу с одним источником и вершиной *u* в качестве исходной, мы решим и эту задачу.

Более того, все известные алгоритмы для этой задачи имеют такое же асимптотическое время выполнения в худшем случае, как и лучшие алгоритмы с одним источником.

Задача нахождения кратчайших путей для всех пар вершин

Найти кратчайший путь из *u в v* для каждой пары вершин *u* и *v*.

Хотя мы можем решить эту проблему, запустив алгоритм с одним источником gпо одному разу из каждой вершины, мы обычно можем решить ее быстрее. Эта проблема будет рассмотрена позже.

Подпути кратчайшего пути

Алгоритмы поиска кратчайших путей основаны на свойстве, что кратчайший путь между двумя вершинами содержит внутри себя другие кратчайшие пути.

Лемма 1 (Подпути кратчайших путей являются кратчайшими путями)

Дан взвешенный, ориентированный граф $G = (V, E)$ с функцией веса

$w : E \rightarrow R$,

пусть $p = < v_0 , v_1 , \ldots, v_k >$ — кратчайший путь из вершины v_0 в вершину v_k и для любых i и j таких, что $0 \leq i \leq j \leq k$, пусть

$p_{ij} = < v_i , v_{i+1} , \ldots, v_j >$ будет подпутем p из вершины v_i в вершину v_j .

Тогда p_{ij} — кратчайший путь из вершины v_i в вершину v_j .

Подпути кратчайшего пути

Доказательство. Если мы разложим путь p на $v_0 \overset{p0i}{\sim >} v_i \overset{pij}{\sim >} v_j$ $p^{jk} \sim > v_k$, тогда имеем, что

$w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$.

Теперь предположим, что существует путь p_{ij} из вершины v_i в вершину v_j с весом

$w(p_{ij}') < w(p_{ij})$.

Тогда, $v_0 \overset{p0i}{\sim >} v \overset{p'ij}{\sim >} v_j \overset{pjk}{\sim >} v_k$ — путь из v_0 в v_k , вес которого $w(p_{0i}) + w(p'_{ij}) + w(p_{jk})$ меньше $w(p)$, что противоречит предположению, о том, что p — кратчайший путь из v_0 в v_k .

Ребра с отрицательным весом

Некоторые примеры задачи поиска кратчайших путей из одного источника могут содержать ребра, веса которых отрицательны.

Если граф $G = (V, E)$ не содержит циклов с отрицательным весом, достижимых из источника s, тогда для всех $v \in V$, вес кратчайшего пути $\delta(s, v)$ остается четко определенным, даже если имеет отрицательное значение.

если граф содержит цикл с отрицательным весом, достижимый из s, веса кратчайшего пути не определены должным образом.

Никакой путь из s в вершину цикла не может быть кратчайшим путем — мы всегда можем найти путь с меньшим весом, следуя предложенному «кратчайшему» пути, а затем проходя по циклу с отрицательным весом.

Поэтому если из s в v есть цикл с отрицательным весом, мы определяем $\delta(s, v) = -\infty$.

На рисунке 1 показано влияние отрицательных весов и циклов с отрицательным весом на веса кратчайшего пути.

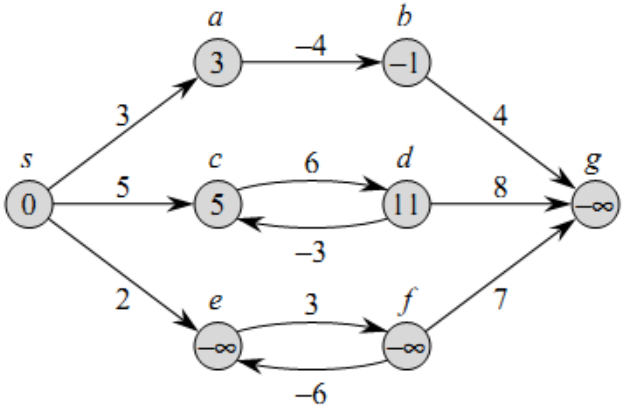
Отрицательные веса ребер в ориентированном графе

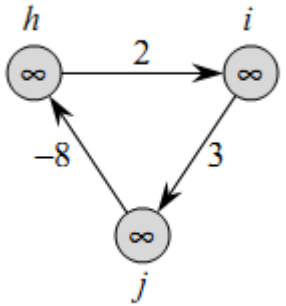
Существует только один путь из *s* в *a* (т.е. путь (*s, a*), имеем $\delta(s, a) = w(s, a) = 3$. Аналогично, существует только один путь из *s* в *b* , и поэтому δ (*s, b*) = *w* (*s, a*) + *w* (*a, b*) = 3 + (- 4) = -1.

Существует бесконечно много путей из *s* в *c* : (*s, c*), (*s, c, d, c*), (*s, c, d, c, d, c*) и так далее.

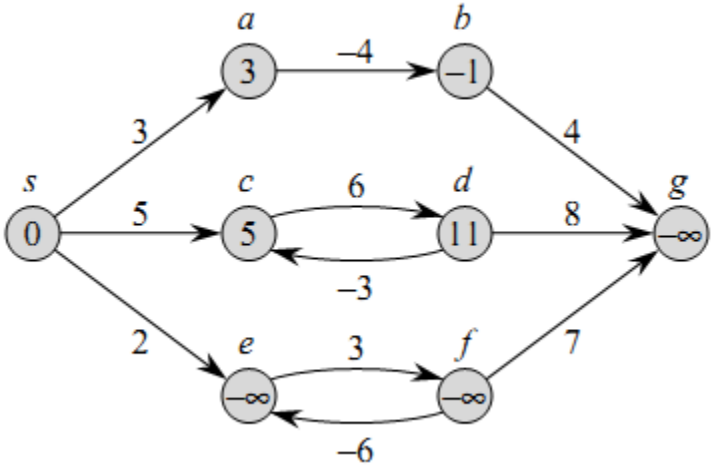
Поскольку цикл (*c, d, c*) имеет вес 6 + -3= 3 > 0, кратчайший путь от *s* до *c* — это (*s, c*) с весом $\delta(s, c) = w(s, c) = 5$.

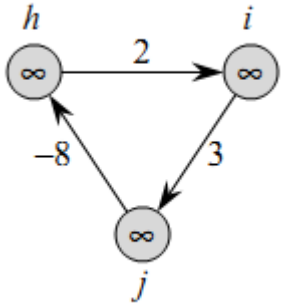
Аналогично, кратчайший путь из *s в d* — это (*s, c, d*), с весом δ (*s, d*) = *w*(*s, c*) + *w* (*c, d*) = 11.





Отрицательные веса ребер в ориентированном графе





Вершины *e* и *f* образуют цикл с отрицательным весом, достижимый из *s*, поэтому их веса кратчайшего пути равны - ∞ .
Вершина *g* достижима из вершины, вес кратчайшего пути которой равен - ∞ , она также имеет вес кратчайшего пути, равный - ∞ .
Такие вершины, как *h, i* и *j,* недостижимы из *s*, поэтому их веса кратчайшего пути равны ∞ , даже если они лежат на цикле с отрицательным весом.

Ребра отрицательного веса в ориентированном графе

Некоторые алгоритмы поиска кратчайших путей, такие как алгоритм Дейкстры, предполагают, что все веса ребер во входном графе неотрицательны .

Другие, такие как алгоритм Беллмана-Форда, допускают наличие ребер с отрицательным весом во входном графе и выдают правильный ответ, если из источника не достижимы циклы с отрицательным весом.

Обычно, если существует такой цикл с отрицательным весом, алгоритм может обнаружить его и сообщить о его существовании.

Циклы

Может ли кратчайший путь содержать цикл?

Как мы только что увидели, он не может содержать цикл с отрицательным весом.

Он также не может содержать цикл с положительным весом , поскольку удаление цикла из пути приводит к созданию пути с теми же начальными и конечными вершинами и меньшим весом.

То есть, если $p = (v_0 , v_1 , \ldots v_k)$ — это путь, а $c = (v_i , v_{i+1} , \ldots , v_j)$ — цикл с положительным весом на этом пути (так что $v_i = v_j$

и $w(c) > 0$), то путь $p' = (v_0 , v_1 , \ldots , v_i , v_{j+1} , v_{j+2} , \ldots, v_{ki})$ имеет вес

$w(p') = w(p) - w(c) < w(p)$, и поэтому p не может быть кратчайшим путем из v_0 в v_k .

Циклы

Остаются только циклы с нулевым весом .

Мы можем удалить цикл с нулевым весом из любого пути и создать другой путь с таким же весом.

Таким образом, если существует кратчайший путь из исходной вершины s в целевую вершину v, содержащий цикл с нулевым весом, то существует другой кратчайший путь из s в v без этого цикла.

Пока кратчайший путь имеет циклы с нулевым весом, мы можем многократно удалять эти циклы из пути, пока не получим кратчайший путь, не содержащий циклов.

Поэтому, без потери общности, можно предположить, что при поиске кратчайших путей они не имеют циклов, т. е. являются простыми путями.

Поскольку любой ациклический путь в графе $G = (V, E)$ содержит не более $|V|$ различных вершин, он также содержит не более $|V|$-1 ребер.

Таким образом, мы можем ограничить наше внимание кратчайшими путями, содержащими не более $|V|$-1 ребер .

Представление кратчайших путей

Часто нам требуется вычислить не только веса кратчайших путей, но и вершины этих кратчайших путей.

Дан граф $G = (V, E)$, будем сохранять для каждой вершины $v \in V$ ее предшественника $v.\pi$, это будет либо другая вершина, либо NIL.

Алгоритмы кратчайших путей устанавливают атрибуты π таким образом, чтобы цепочка предшественников, начинающихся в вершине v, проходила в обратном направлении по кратчайшему пути от s до v .

Однако в процессе выполнения алгоритма поиска кратчайших путей значения π могут не указывать на кратчайшие пути.

При поиске в ширину, мы уже рассмотрели подграф предшествования $G_\pi = (V_\pi, E_\pi)$, индуцированный значениями π .

Здесь мы снова определяем множество вершин V_π как множество вершин G с ненулевыми предшественниками, плюс источник s :

$$V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\} \ .$$

Представление кратчайших путей

Множество ориентированных ребер E_π - это множество ребер, индуцированных значениями π для вершин в V_π :

$$E_\pi = \{(v.\pi, v) \in E : v \in V_\pi - \{s\}\} \ .$$

Значения π, полученные с помощью алгоритмов поиска кратчайшего пути с одним источником, имеют свойство, что по завершении G_π — это «дерево кратчайших путей», то есть, корневое дерево, содержащее кратчайший путь от источника *s* до каждой вершины, достижимой из *s* . Дерево кратчайших путей похоже на дерево поиска в ширину, но оно содержит кратчайшие пути от источника, определенные в терминах весов ребер, а не количества ребер.

Представление кратчайших путей

Дерево кратчайших путей с корнем в s представляет собой ориентированный подграф $G' = (V', E')$ где

$V' \subseteq V$ и $E' \subseteq E$, такой, что

1. V' — множество вершин, достижимых из s в G ,
2. G' образует корневое дерево с корнем s, и
3. для всех $v \in V'$, единственный простой путь из s в v в графе G' является кратчайшим путем из s в v в графе G.

Кратчайшие пути не обязательно уникальны, как и деревья кратчайших путей.

Например, на следующем слайде показан взвешенный ориентированный граф и два дерева кратчайших путей с одним и тем же корнем.

Представление кратчайших путей

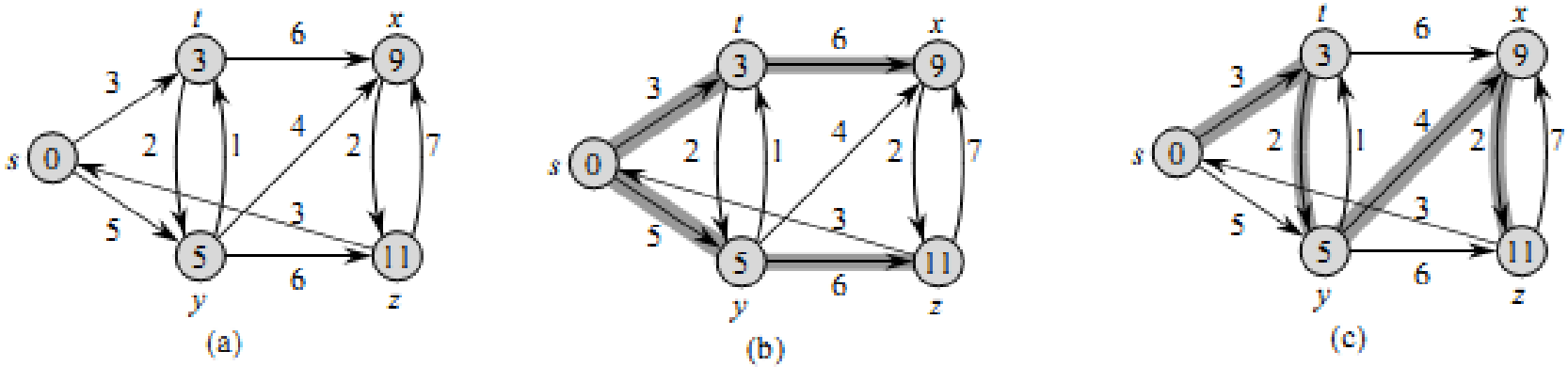


Рис. 2 а) Взвешенный ориентированный граф с весами кратчайших путей из источника *s* .
(b) Заштрихованные ребра образуют дерево кратчайших путей с корнем в источнике *s* .
(c) Другое дерево кратчайших путей с тем же корнем.

Инициализация

Для каждой вершины $v \in V$ будем поддерживать атрибут $v.d$, который является верхней границей веса кратчайшего пути из источника s в вершину v .

Будем называть $v.d$ оценкой длины кратчайшего пути .

Мы инициализируем оценки длины кратчайшего пути и предшественников следующим образом:

INITIALIZE-SINGLE-SOURCE(G, s)

1 **for** each vertex $v \in G.V$

2 $v.d = \infty$

3 $v.\pi$ = NIL

4 $s.d = 0$

После инициализации имеем $v.\pi$ = NIL для всех $v \in V$, $s.d = 0$, и $v.d = \infty$ для $v \in V$- {s} .

Релаксация

Процесс **релаксации** ребра (*u, v*) состоит из проверки того, можем ли мы улучшить кратчайший путь в вершину *v* , найденный на данный момент, пройдя через вершину *u*, и, если да, то обновления значений *v.d* и $v.\pi$.

Шаг релаксации может уменьшить значение оценки длины кратчайшего пути *v.d* и обновить предшествующий атрибут $v.\pi$.

Следующий код выполняет шаг релаксации на ребре (*u, v*) за время *O*(1):

RELAX(*u, v, w*)

1 **if** *v.d* > *u.d* + *w*(*u, v*)

2 *v.d* = *u.d* + *w*(*u, v*)

3 $v.\pi = u$

Релаксация

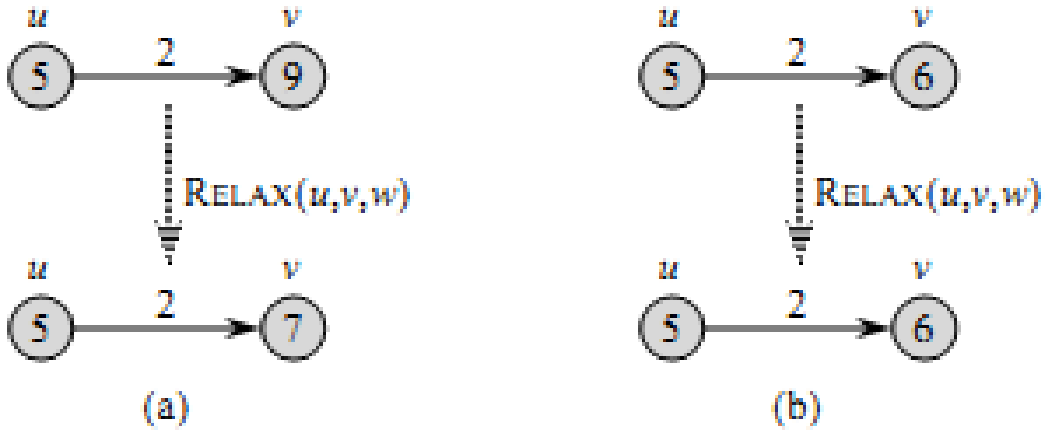


Рис. 3. Релаксация ребра (u, v) с весом $w(u, v) = 2$.
(а) Поскольку до релаксации $v.d > u.d + w(u, v)$, значение $v.d$ уменьшается.
(б) Здесь, $v.d \leq u.d + w(u, v)$ перед релаксацией ребра, и поэтому шаг релаксации оставляет $v.d$ неизменным.

Инициализация и релаксация в алгоритмах поиска кратчайшего пути

Каждый рассматриваемый алгоритм поиска кратчайшего пути с одним источником вызывает INITIALIZE-SINGLE-SOURCE, а затем многократно релаксирует ребра.

Более того, релаксация — единственный способ, с помощью которого изменяются оценки длины кратчайшего пути и предшественники.

Алгоритмы различаются тем, сколько раз они релаксируют каждое ребро и в каком порядке они релаксируют ребра.

Алгоритм Дейкстры релаксирует каждое ребро ровно один раз.

Беллмана-Форда релаксирует каждое ребро $|V|$- 1 раз.

Свойства кратчайших путей и релаксации

Чтобы доказать корректность алгоритмов, будем использовать несколько свойств кратчайших путей и релаксации.

Для справки: каждое указанное здесь свойство включает соответствующий номер леммы или следствия из Кормена (раздел 24.5).

Последние 5 из этих свойств, которые относятся к оценкам длин кратчайшего пути или подграфу предшествования, неявно предполагают, что граф инициализируется вызовом INITIALIZE-SINGLE-SOURCE(*G, s*), и что единственный способ, которым оценки кратчайшего пути и предшествующий подграф изменяются, — это некоторая последовательность шагов релаксации.

Свойства кратчайших путей и релаксации

Неравенство треугольника (лемма 24.10)
Для любого ребра $(u, v) \in E$, имеем $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Свойство верхней границы (лемма 24.11)
$v.d \geq \delta(s, u)$ для всех вершин $v \in V$, и как только $v.d$ достигает значения $\delta(s, v)$, оно никогда не меняется.

Свойство отсутствия пути (следствие 24.12)
Если нет пути из s в v , то мы всегда имеем $v.d = \delta(s, v) = \infty$.

Свойство сходимости (лемма 24.14)
Если $s \rightsquigarrow u \rightarrow v$ — кратчайший путь в G для некоторых $u, v \in V$, и если $u.d = \delta(s, u)$ в любой момент времени до релаксации ребра (u, v), то $v.d = \delta(s, v)$ в любой момент времени после этого.

Свойства кратчайших путей и релаксации

Свойство релаксации пути (лемма 24.15)

Если $p = < v_0, v_1, \ldots, v_k >$ — кратчайший путь из вершины $s = v_0$ в v_k, и ребра пути p релаксируются

в порядке $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, тогда $v_k.d = \delta(s, v_k)$.

Это свойство сохраняется независимо от любых других происходящих стадий релаксации, даже если

они перемешаны с релаксациями ребер пути p .

Свойство подграфа предшествования (лемма 24.17)

Как только $v.d = \delta(s, v)$ для всех $v \in V$, подграф предшествования представляет собой дерево кратчайших путей с корнем в вершине s .

Арифметика с бесконечностями.

Предположим, что для любого вещественного числа $a \neq \infty$,

$a + \infty = \infty + a = \infty$.

Кроме того, чтобы доказательства были верны при наличии циклов с отрицательным весом, будем считать, что для любого вещественного числа $a \neq \infty$, верно

$a + (- \infty) = (- \infty) + a = - \infty$.

Все алгоритмы этой лекции предполагают, что ориентированный граф G хранится в представлении списком смежности.

Кроме того, с каждым ребром хранится его вес, так что при обходе каждого списка смежности мы можем определить вес ребра за время $O(1)$ для каждого ребра.

Часть 2 Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда решает задачу поиска кратчайших путей из одного источника в общем случае, когда веса ребер могут быть отрицательными.

Для взвешенного ориентированного графа $G = (V, E)$ с источником s и функцией веса $w : E \to R$ алгоритм Беллмана-Форда возвращает логическое значение, указывающее, существует ли цикл с отрицательным весом, достижимый из источника.

Если такой цикл есть, алгоритм указывает, что решения не существует.

Если такого цикла нет, алгоритм выдает кратчайшие пути и их веса.

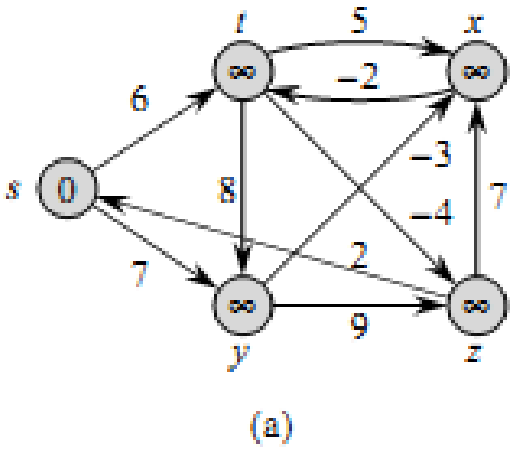
Алгоритм релаксирует ребра, постепенно уменьшая оценку $v.d$ веса кратчайшего пути из источника s в каждую вершину $v \in V$ до тех пор, пока не будет достигнут фактический вес кратчайшего пути $\delta(s, v)$.

Алгоритм возвращает ИСТИНА $\Leftrightarrow$ граф не содержит циклов с отрицательным весом, достижимых из источника.

Алгоритм Беллмана-Форда

```
BELLMAN-FORD(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 for i = 1 to |G.V| - 1
3       for each edge (u, v)∈G.E
4               RELAX(u, v, w)
5 for each edge (u, v) ∈ G.E
6       if v.d > u.d + w(u, v)
7               return FALSE
8 return TRUE
```

Пример. Инициализация

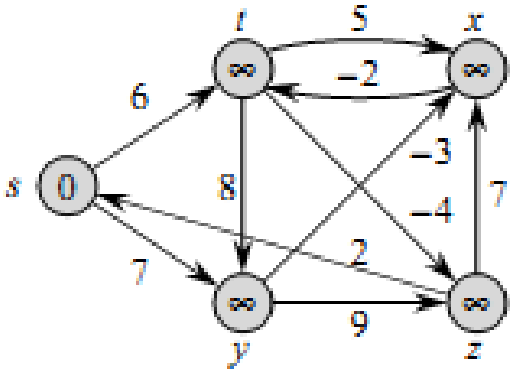


Источником является вершина s .
Значения d отображаются внутри вершин, а затененные ребра указывают на значения предшественников вершины: если ребро (u, v) затенено, то $v.\pi = u$.
В этом конкретном примере каждый проход осуществляет релаксацию ребер в следующем порядке: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y).

(a) Ситуация непосредственно перед первым проходом по ребрам.

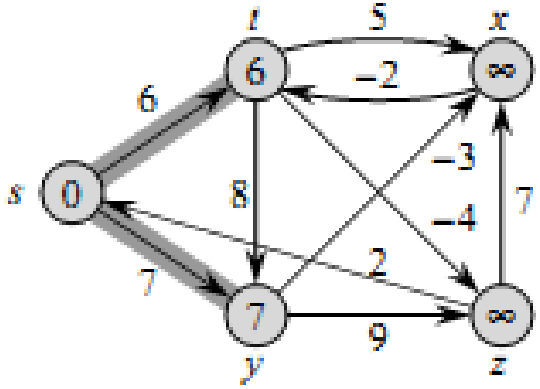
Пример, 1-я итерация

до:



(a)

После 1-й итерации :



(b)

Если $v.d > u.d + w(u, v)$

$v.d = u.d + w(u, v)$

$v.\pi = u$

(t, x): $t.d + 5 = \infty + 5 = \infty \Rightarrow x.d = \infty$ (без изменений)

(t, y): $t.d + 8 = \infty + 8 \Rightarrow y.d = \infty$ => без изменений

(t, z): $t.d - 4 = \infty - 4 \Rightarrow z.d = \infty$ => без изменений

(x, t): $x.d - 2 = \infty - 2 \Rightarrow t.d = \infty$ => без изменений

(y, x): $y.d - 3 = \infty - 3 \Rightarrow x.d = \infty$ => без изменений

(y, z): $y.d + 9 = \infty + 9 \Rightarrow z.d = \infty$ => без изменений

(z, x): $z.d + 7 = \infty + 7 \Rightarrow x.d = \infty$ => без изменений

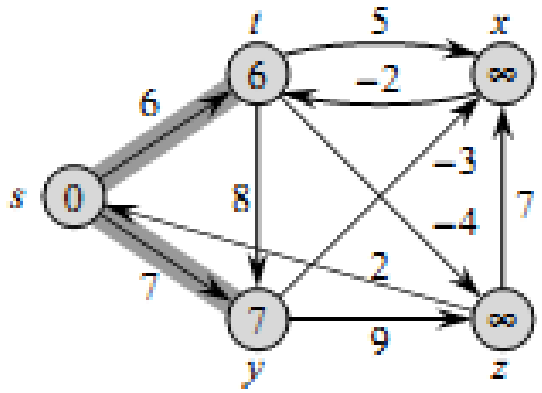
(z, s): $z.d + 2 = \infty + 2 > s.d = 0$ => без изменений

(s, t): $t.d = \infty$ $t.d > s.d + w(s, t) = 0 + 6 \Rightarrow t.d = 6$, $t.\pi = s$

(s, y): $y.d = \infty$ $y.d > s.d + w(s, y) = 0 + 7 \Rightarrow y.d = 7$, $y.\pi = s$

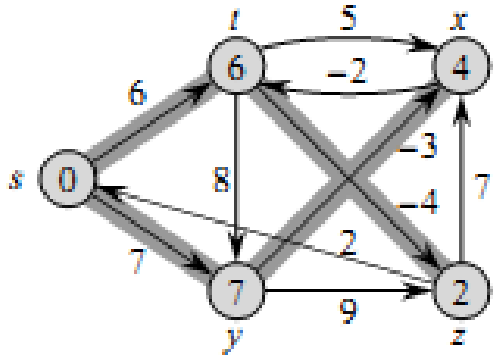
Пример, 2-я итерация:

До:



(b)

После 2-й итерации:

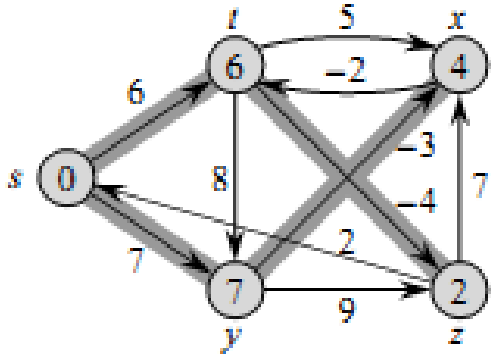


(c)

(*t*, *x*): *t.d* + *w*(*t*, *x*) = 6 + 5 < ∞ => *x.d* = 11, *x.*π = *t* ,
(*t*, *y*): *t.d* + *w*(*t*, *y*) = 6 + 8 > 7 => без изменений
(*t*, *z*): *t.d* + *w*(*t*, *z*) = 6 – 4 = 2 < ∞ => *z.d* = 2, *z.*π = *t*
(*x*, *t*): *x.d* + *w*(*x*, *t*) = 11 - 2 = 9 > 6 => без изменений,
(*y*, *x*): *y.d* + *w*(*y*, *x*) = 7 - 3 < 11 => *x.d* = 4, *x.*π = *y* ,
(*y*, *z*): *y.d* + *w*(*y*, *z*) = 7 + 9 > 2 => без изменений
(*z*, *x*): *z.d* + *w*(*z*, *x*) = 2 + 7 > 4 => без изменений
(*z*, *s*): *z.d* + *w*(*z*, *s*) = 2 + 2 > 0 => без изменений
(*s*, *t*): *s.d* + *w*(*s*, *t*) = 0 + 6 = 6 => без изменений
(*s*, *y*): *s.d* + *w*(*s*, *y*) = 0 + 7 = 7 => без изменений

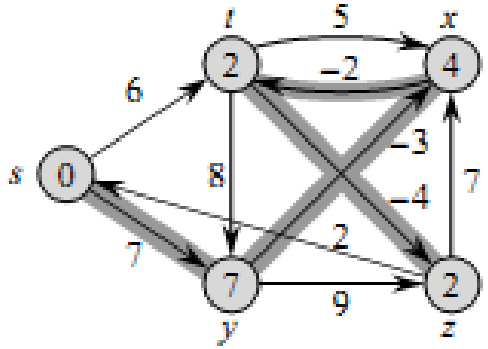
Пример, 3-я итерация:

До:



(c)

После 3- й итерации:

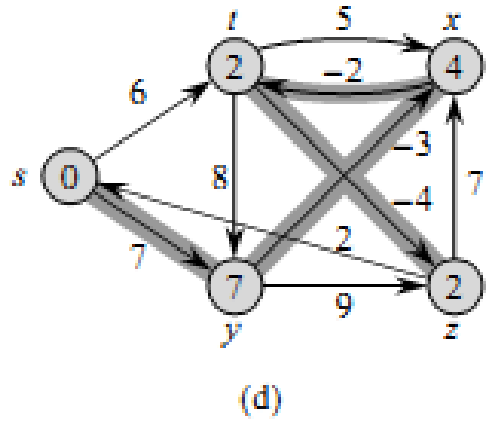


(d)

(*t*, *x*): *t.d* + *w*(*t*, *x*) = 6 + 5 > 4 => без изменений
(*t*, *y*): *t.d* + *w*(*t*, *y*) = 6 + 8 > 7 => без изменений
(*t*, *z*): *t.d* + *w*(*t*, *z*) = 6 – 4 = 2 => без изменений
(*x*, *t*): *x.d* + *w*(*x*, *t*) = 4 - 2 < 6 => *t.d* = 2, $t.\pi = x$
(*y*, *x*): *y.d* + *w*(*y*, *x*) = 7 - 3 = 4 => без изменений ,
(*y*, *z*): *y.d* + *w*(*y*, *z*) = 7 + 9 > 2 => без изменений
(*z*, *x*): *z.d* + *w*(*z*, *x*) = 2 + 7 > 4 => без изменений
(*z*, *s*): *z.d* + *w*(*z*, *s*) = 2 + 2 > 0 => без изменений
(*s*, *t*): *s.d* + *w*(*s*, *t*) = 0 + 6 = 6 => без изменений
(*s*, *y*): *s.d* + *w*(*s*, *y*) = 0 + 7 = 7 => без изменений

Пример, 4-я итерация:

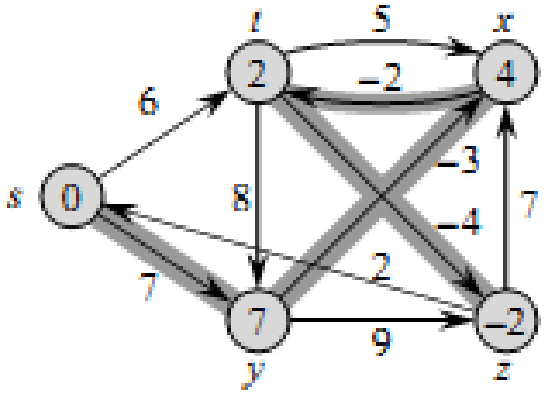
До:



(d)

(*t*, *x*): *t.d* + *w*(*t*, *x*) = 2 + 5 > 4 => без изменений
(*t*, *y*): *t.d* + *w*(*t*, *y*) = 2 + 8 > 7 => без изменений
(*t*, *z*): *t.d* + *w*(*t*, *z*) = 2 – 4 < 2 => *z.d* = -2, *z.*π = *t*
(*x*, *t*): *x.d* + *w*(*x*, *t*) = 4 - 2 = 2 => без изменений
(y, *x*): *y.d* + *w*(*y*, *x*) = 7 - 3 = 4 => без изменений
(*y*, *z*): *y*.d + *w*(*y*, *z*) = 7 + 9 > -2 => без изменений
(*z*, *x*): *z.d* + *w*(*z*, *x*) = -2 + 7=5 > 4 => без изменений
(*z*, *s*): *z.d* + *w*(*z*, *s*) = -2 + 2 = 0 => без изменений
(*s*, *t*): *s.d* + *w*(*s*, *t*) = 0 + 6 = 6 => без изменений
(*s*, *y*): *s.d* + *w*(*s*, *y*) = 0 + 7 = 7 => без изменений

После 4- й итерации:



(e)

Алгоритм Беллмана-Форда

На слайдах показано выполнение алгоритма Беллмана-Форда на графе с 5 вершинами.

После инициализации значениq $v.d$ и $v.\pi$ всех вершин в строке 1, алгоритм делает

$|V|-1 = 4$ прохода по ребрам графа.

Каждый проход представляет собой 1 итерацию цикла **for** строк 2–4 и состоит из однократной релаксации каждого ребра графа.

После выполнения $|V|-1$ проходов строки 5–8 проверяют наличие цикла с отрицательным весом и возвращают соответствующее логическое значение.

Алгоритм Беллмана-Форда выполняется за время $O(VE)$, поскольку инициализация в строке 1 занимает $\Theta(V)$ времени, каждый из проходов

$|V|-1$ по ребрам в строках 2–4 занимает $\Theta(E)$ времени, а цикл for в строках 5–7 занимает $O(E)$ времени.

Чтобы доказать корректность алгоритма Беллмана-Форда, мы начнем с демонстрации того, что если в графе нет циклов с отрицательным весом, алгоритм вычисляет правильные веса кратчайшего пути для всех вершин, достижимых из источника.

Свойства алгоритма Беллмана-Форда

Лемма 2. Пусть $G = (V, E)$ — взвешенный ориентированный граф с источником s и функцией веса $w : E \to R$, и предположим, что G не содержит циклов с отрицательным весом, достижимых из s.

Тогда, после $|V| - 1$ итераций цикла **for** строк 2–4 BELLMAN-FORD, мы имеем $v.d = \delta(s, v)$ для всех вершин v, достижимых из s.

Доказательство Мы докажем лемму, апеллируя к свойству релаксации пути. Рассмотрим любую вершину v, достижимую из s, и пусть

$p = \langle v_0, v_1, \ldots, v_k \rangle$, где $v_0 = s$ и $v_k = v$, — любой кратчайший путь из s в v.

Поскольку кратчайшие пути просты, p имеет не более $|V| - 1$ ребер, и поэтому $k \leq |V| - 1$.

Каждая из $|V| - 1$ итераций цикла **for** строк 2–4 релаксирует все $|E|$ ребра.

Среди ребер, релаксированных на i- й итерации, для $i = 1, 2, \ldots, k$, находится ребро (v_{i-1}, v_i).

Следовательно, по свойству релаксации пути $v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$.

Свойства алгоритма Беллмана-Форда

Следствие 3. Пусть $G = (V, E)$ — взвешенный ориентированный граф с исходной вершиной s и функцией веса $w : E \to R$, и предположим, что G не содержит циклов с отрицательным весом, достижимых из s .

Тогда для каждой вершины $v \in V$ существует путь из s в $v \Leftrightarrow$ BELLMAN-FORD завершается с $v.d < \infty$ при запуске на G.

Корректность алгоритма Беллмана-Форда

Теорема 4. Алгоритм BELLMAN-FORD запущен на взвешенном ориентированном графе $G = (V, E)$ с источником s и функцией веса $w : E \to R$.

Если G не содержит циклов с отрицательным весом, достижимых из s , то алгоритм возвращает TRUE, мы имеем $v.d = \delta(s, v)$ для всех вершин $v \in V$, а подграф предшествования G является деревом кратчайших путей с корнем в s.

Если G содержит цикл с отрицательным весом, достижимый из s, то алгоритм возвращает FALSE.

Корректность алгоритма Беллмана-Форда

Доказательство. Предположим, что граф *G не* содержит циклов с отрицательным весом, достижимых из источника *s.*

Сначала мы докажем утверждение, что в конце $v.d = \delta(s, v)$ для всех вершин $v \in V$.

а) Если вершина *v* достижима из *s*, то лемма 2 доказывает это утверждение.

б) Если *v* недостижима из *s* , то утверждение следует из свойства отсутствия пути.

Таким образом, утверждение доказано.

в) Свойство подграфа предшествования вместе с утверждением подразумевает, что G_π является деревом кратчайших путей.

г) Теперь мы используем все эти факты, чтобы показать, что BELLMAN-FORD возвращает TRUE.

В конце работы алгоритма мы имеем для всех ребер $(u, v) \in E$,

$v.d = \delta(s, v)$

$\leq \delta(s, u) + w(u, v)$ (по неравенству треугольника)

$= u.d + w(u, v)$.

и поэтому ни один из тестов в строке 6 не заставляет BELLMAN-FORD возвращать FALSE. Следовательно, он возвращает TRUE.

Корректность алгоритма Беллмана-Форда

е) Теперь предположим, что граф G содержит цикл с отрицательным весом, который достижим из источника s.

Пусть этот цикл будет $c = < v_0 , v_1 , \ldots, v_k >$, где $v_0 = v_k$. Тогда,

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0 . \qquad (1)$$

Предположим для доказательства противоречия, что алгоритм Беллмана-Форда возвращает значение TRUE.

Чтобы получилось TRUE, надо иметь на каждом ребре

$v_i.d \leq v_{i-1}.d + w(v_{i-1} , v_i)$ для $i = 1, 2, \ldots, k$.

Корректность алгоритма Беллмана-Форда

Просуммируем эти неравенства по ребрам цикла *c:*

$$\begin{aligned}\sum_{i=1}^{k} v_i.d &\leq \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i)) \\ &= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)\ .\end{aligned}$$

Поскольку $v_0 = v_k$, каждая вершина цикла *c* появляется ровно один раз в каждой из сумм

$$\sum_{i=1}^{k} v_i.d \qquad \sum_{i=1}^{k} v_{i-1}.d,$$

Поэтому

$$\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d\ .$$

Корректность алгоритма Беллмана-Форда

Более того, по следствию 3, $v_i.d$ конечно для $i = 1, 2, \ldots, k$.

Таким образом,

$$0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i) ,$$

что противоречит неравенству (1).

Стало быть, алгоритм Беллмана-Форда возвращает значение TRUE, если граф G не содержит циклов с отрицательным весом, достижимых из источника, и FALSE в противном случае.

Часть 3. Алгоритм Дейкстры

Алгоритм Дейкстры

Алгоритм Дейкстры решает задачу поиска кратчайших путей из одного источника на взвешенном, ориентированном графе $G = (V, E)$ в случае, когда все веса ребер неотрицательны.

Поэтому далее мы предполагаем, что $w(u, v) \geq 0$ для каждого ребра $(u, v) \in E$.

Как мы увидим, при хорошей реализации время работы алгоритма Дейкстры меньше, чем у алгоритма Беллмана-Форда.

Алгоритм Дейкстры поддерживает множество вершин S, чьи окончательные веса кратчайшего пути от источника s уже определены.

Алгоритм на каждой итерации выбирает вершину $u \in V - S$ с минимальной оценкой кратчайшего пути, добавляет u к S и релаксирует все ребра, исходящие из u .

Q -очередь с минимальным приоритетом вершин, ключами которых являются их значения d .

Алгоритм Дейкстры

DIJKSTRA(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 $S = \varnothing$;
3 $Q = G.V$
4 **while** $Q \neq \varnothing$;
5 u = EXTRACT-MIN(Q)
6 $S = S \cup \{u\}$
7 **for** each vertex $v \in G.Adj[u]$
8 RELAX(u, v, w)

Алгоритм Дейкстры

Алгоритм Дейкстры релаксирует ребра, как показано на слайде 52.

Строка 1 инициализирует значения атрибутов d и π обычным способом, а строка 2 инициализирует множество S пустым множеством.

Алгоритм сохраняет инвариант, что $Q = V - S$ в начале каждой итерации цикла **while** строк 4–8.

Строка 3 инициализирует очередь с минимальным приоритетом Q , так чтобы она содержала все вершины в V ; поскольку $S = \varnothing$; в это время инвариант становится истинным после строки 3.

Каждый раз в цикле **while** строк 4–8 строка 5 извлекает вершину u из $Q = V - S$, а строка 6 добавляет ее к множеству S, тем самым сохраняя инвариант.

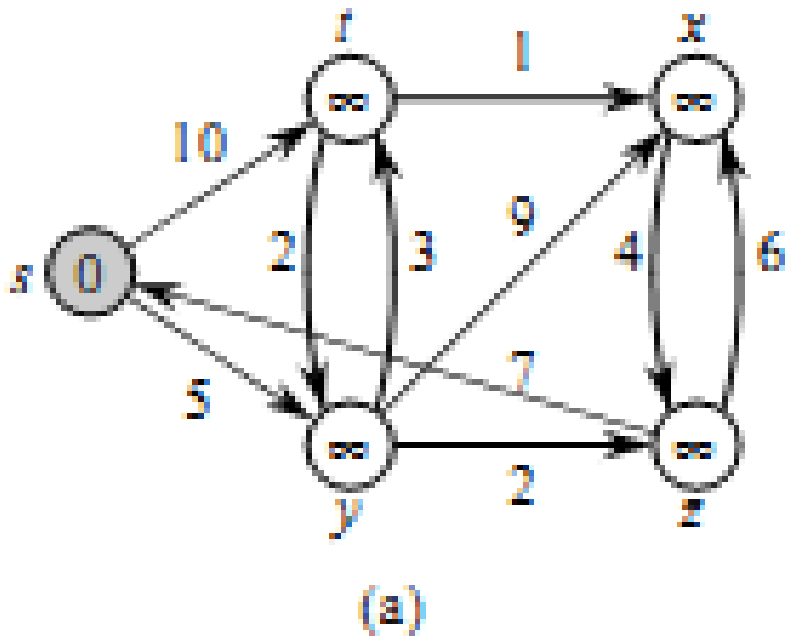
(В первый раз в этом цикле $u = s$.)

вершина u имеет наименьшую оценку длины кратчайшего пути среди всех вершин в $V - S$.

Затем строки 7–8 релаксируют каждое ребро (u, v), выходящее из u, тем самым обновляя оценку $v.d$ и предшественника $v.\pi$, если мы можем улучшить длину кратчайшего пути в v, найденную до сих пор, пройдя через u .

Обратите внимание, что алгоритм никогда не вставляет вершины в Q после строки 3 и что каждая вершина извлекается из Q и добавляется в S ровно один раз, так что цикл **while** строк 4–8 повторяется ровно $|V|$ раз.

Алгоритм Дейкстры. Пример

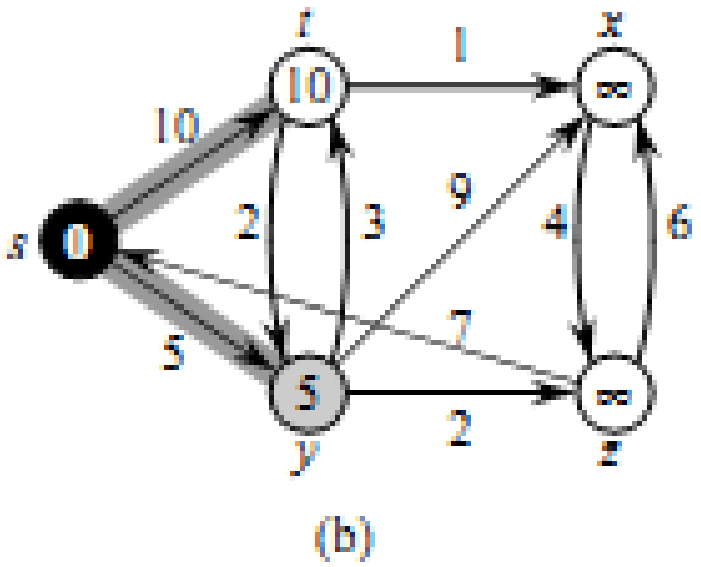


(a)

После инициализации:

$V = \varnothing$

Алгоритм Дейкстры. Пример



(b)

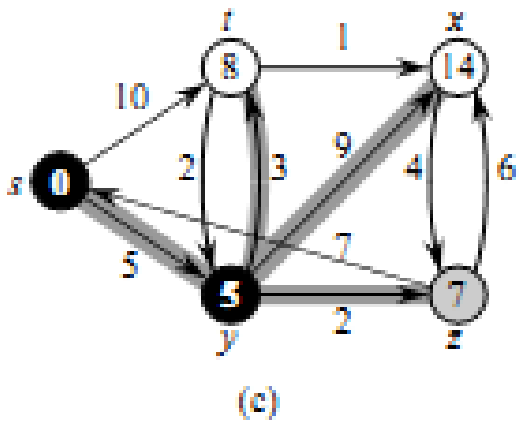
1-я итерация:

$S = \{ s \}$

$\{ s, y \}$: $0 + 5 < \infty \Rightarrow y.d = 5$, $y.\pi = s$

$\{ s, t \}$: $0 + 10 < \infty \Rightarrow t.d = 10$, $t.\pi = s$

Алгоритм Дейкстры. Пример



(c)

2-я итерация:

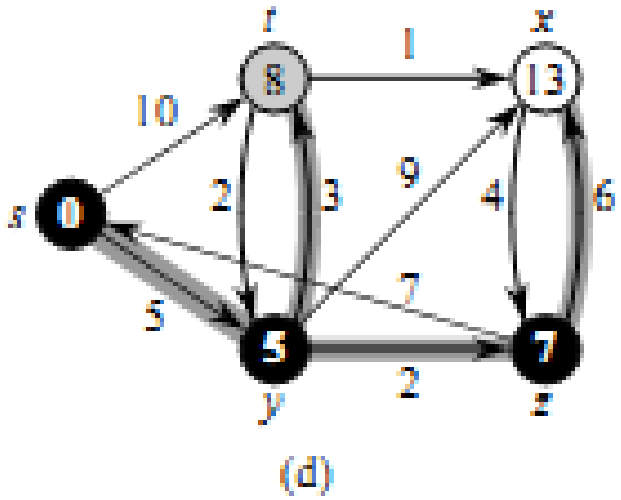
$S = \{ s, y \}$

$\{ y, t \}$: $5 + 3 < 10 \Rightarrow t.d = 8, t.\pi = y$

$\{ y, x \}$: $5 + 9 < \infty \Rightarrow x.d = 14, x.\pi = y$

$\{y, z\}$: $5 + 2 < \infty \Rightarrow z.d = 7$, $z.\pi = y$

Алгоритм Дейкстры. Пример



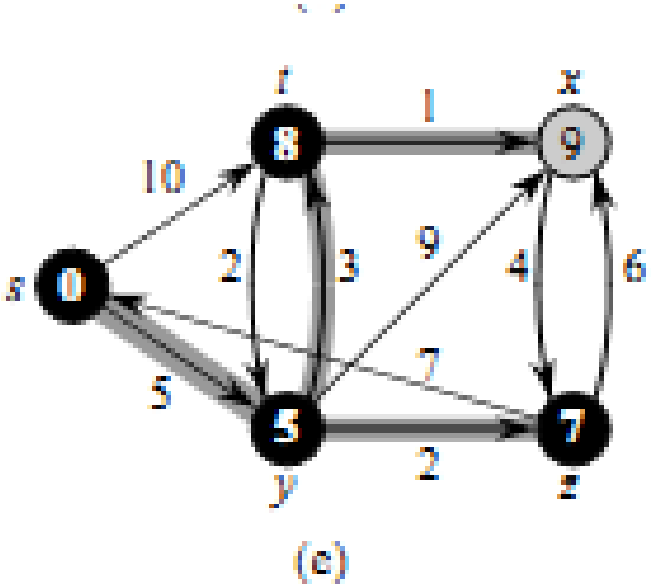
(d)

3-я итерация:

$S = \{ s, y, z \}$

$\{ z, x \}$: $7 + 6 < 13 \Rightarrow x.d = 13$, $x.\pi = z$

Алгоритм Дейкстры. Пример

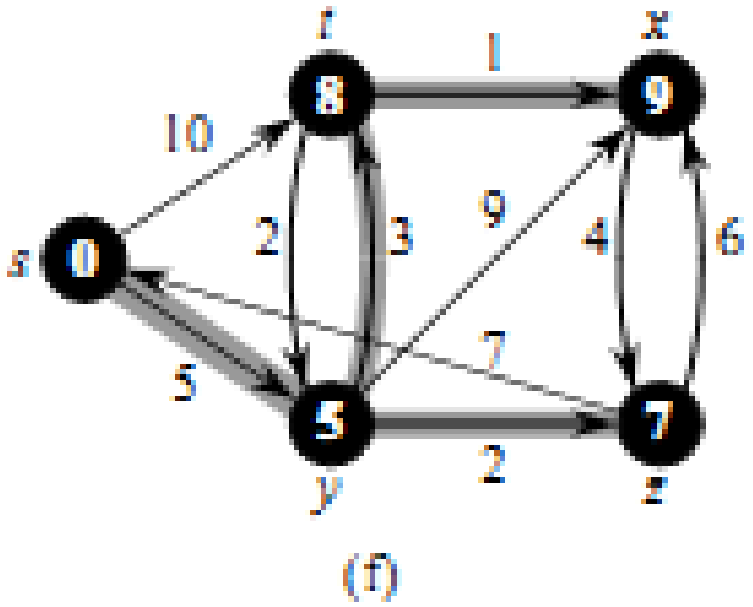


4-я итерация:

$S = \{ s, y, z, m \}$

$\{ t, x \}$: $8 + 1 < 13 \Rightarrow x.d = 9, x.\pi = t$

Алгоритм Дейкстры. Пример



(f)

5-я итерация:

$S = \{ s, y, z, t, x \}$

Алгоритм Дейкстры

Поскольку алгоритм Дейкстры всегда выбирает «самую легкую» или «ближайшую» вершину в $V - S$ для добавления к множеству S, мы говорим, что он использует жадную стратегию .

Жадные стратегии не всегда дают оптимальные результаты в целом, но, как показывают следующая теорема и ее следствие, алгоритм Дейкстры действительно вычисляет кратчайшие пути.

Главное — показать, что каждый раз, когда он добавляет вершину u к множеству S, мы имеем

$u.d = \delta(s, u)$.

Корректность алгоритма Дейкстры

Теорема 5 (Корректность алгоритма Дейкстры)

Алгоритм Дейкстры, запущенный на взвешенном ориентированном графе $G = (V, E)$ с неотрицательной функцией веса w и источником s, завершается с

$u.d = \delta(s, u)$ для всех вершин $u \in V$.

Корректность алгоритма Дейкстры

Доказательство. Мы используем следующий инвариант цикла:

В начале каждой итерации цикла **while** строк 4–8, $v.d = \delta(s, v)$ для каждой вершины $v \in S$.

Достаточно показать для каждой вершины $u \in V$, мы имеем $u.d = \delta(s, u)$ в момент, когда вершина u добавляется к множеству S .

Как только мы покажем, что $u.d = \delta(s, u)$, мы полагаемся на свойство верхней границы, чтобы показать, что равенство сохраняется в любое время после этого.

Корректность алгоритма Дейкстры

Инициализация : Первоначально $S = \varnothing$, и поэтому инвариант тривиально верен.

Поддержание инварианта: Мы хотим показать, что на каждой итерации $u.d = \delta(s, u)$ для вершины, добавленной к множеству S .

Для доказательства от противного предположим, что u будет первой вершиной, для которой $u.d \neq \delta(s, u)$ при добавлении к множеству S .

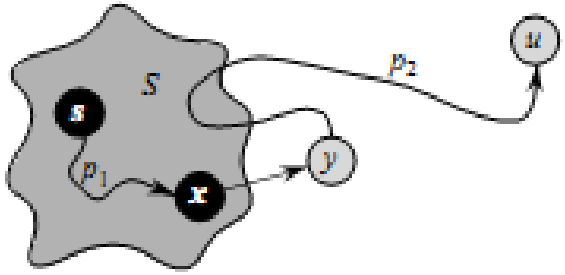
Мы сосредоточим наше внимание на ситуации в начале итерации цикла **while**, в которой u добавляется к S, и получим противоречие, что $u.d = \delta(s, u)$ в этот момент, исследуя кратчайший путь от s до u .

1. $u \neq s$, потому что s — первая вершина, добавленная к множеству S, и $s.d = \delta(s, s) = 0$ в этот момент.
2. Поскольку $u \neq s$, можно утверждать, что $S \neq \varnothing$ непосредственно перед добавлением u к S .
3. Должен быть какой-то путь из s в u , иначе бы $u.d = \delta(s, u) = \infty$ по свойству отсутствия пути, что нарушило бы наше предположение о том, что $u.d \neq \delta(s, u)$.

Корректность алгоритма Дейкстры

- Поскольку существует по крайней мере один путь, существует и кратчайший путь *p* *из s в u* .
- Перед добавлением вершины *u* к множеству *S* путь *p* соединяет вершину во множестве *S*, а именно вершину *s*, с вершиной в *V - S* ,

  а именно с вершиной *u*.



Рассмотрим первую вершину *y* вдоль пути *p* такую, что *y*∈ *V - S* , и пусть *x* ∈ *S* будет предшественником *y вдоль пути p* .
Таким образом, мы можем разложить путь *p* от источника *s* до вершины *u* в
c ^{p1} ~> *x - > y* ^{p2} ~ > *y* .
(Любой из путей p_1 или p_2 может не иметь ребер.)

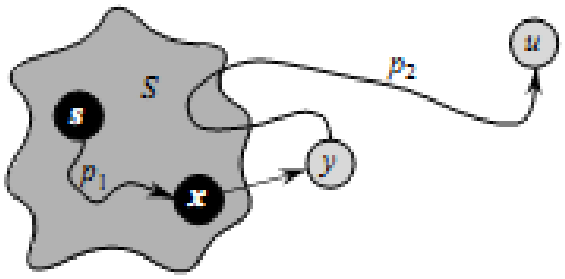
Корректность алгоритма Дейкстры

Множество S не пусто непосредственно перед добавлением к нему вершины u .

y — первая вершина на пути, которая не входит в S, а $x \in S$ непосредственно предшествует y .

Вершины x и y различны, но может быть $s = x$ или $y = u$.

Путь p_2 может повторно войти в множество S , а может и не войти .



Корректность алгоритма Дейкстры

Мы утверждаем, что $y.d = \delta(s, y)$, когда u добавляется к S .

Чтобы доказать это утверждение, заметим, что $x \in S$.

Тогда, поскольку мы выбрали u как первую вершину, для которой $u.d \neq \delta(s, u)$ при добавлении к S, мы имели $x.d = \delta(s, x)$ при добавлении x к S .

Ребро (x, y) было релаксировано в это время, и утверждение $y.d = \delta(s, y)$, следует из свойства сходимости.

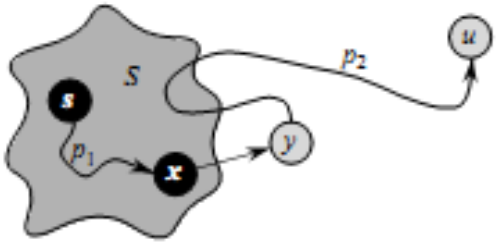
Теперь мы можем получить противоречие и доказать, что $u.d = \delta(s, u)$.

Поскольку y появляется раньше u на кратчайшем пути от s до u , а все веса ребер неотрицательны (особенно на пути p_2), мы имеем $\delta(s, y) \leq \delta(s, u)$, и таким образом

$y.d = \delta(s, y)$

$\leq \delta(s, u)$ (2)

$\leq u.d$ (по свойству верхней границы).

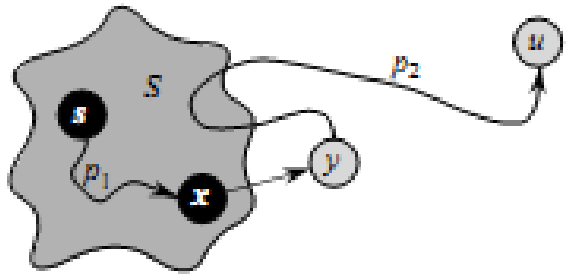


Но поскольку обе вершины *u* и *y* находились в *V - S* , когда *u* была выбрана в строке 5, мы имеем $u.d \leq \delta(s, u)$.

Таким образом, два неравенства в (2) на самом деле являются равенствами, что дает $y.d = \delta(s, y) = \delta(s, u) = u.d$.

Следовательно, $u.d = \delta(s, u)$, что противоречит нашему выбору *u* .

Мы приходим к выводу, что $u.d = \delta(s, u)$, когда *u* добавляется к *S* , и что это равенство сохраняется во все последующие моменты времени.



Корректность алгоритма Дейкстры

Завершение : В момент завершения работы алгоритма $Q = \varnothing$, что вместе с нашим предыдущим инвариантом $Q = V - S$ подразумевает, что $S = V$.

Таким образом, $u.d = \delta(s, u)$ для всех вершин $u \in V$.

Следствие 6

Если мы запустим алгоритм Дейкстры на взвешенном ориентированном графе $G = (V, E)$ с неотрицательной функцией веса w и источником s , то при завершении подграф предшествования G_π — это дерево кратчайших путей с корнем в s .

Доказательство следует непосредственно из теоремы 5 и свойства подграфа предшествования .

Сложность алгоритма Дейкстры

Теперь мы можем оценить вычислительную сложность алгоритма Дейкстры (с точки зрения операций сложения и сравнения).

Алгоритм использует не более $|V|$ итераций , где $|V|$ — количество вершин в графе, поскольку на каждой итерации к выделенному множеству S добавляется одна вершина.

Оценим количество операций, используемых для каждой итерации.

Мы можем идентифицировать вершину, не лежащую в S, с наименьшей оценкой $v.d$, используя не более $|V| - 1$ сравнений.

Затем мы используем одно сложение и одно сравнение для обновления оценки длины пути для каждой вершины, не входящей в S.

Из этого следует, что на каждой итерации используется не более чем

$2(|V| - 1)$ операций, поскольку на каждой итерации необходимо обновить не более $n - 1$ оценок.

Поскольку используется не более $|V|$ итераций, каждая из которых использует не более $2(|V| - 1)$ операций, получаем Теорему 7.

Сложность алгоритма Дейкстры

ТЕОРЕМА 7 Алгоритм Дейкстры использует $O(|V|^2)$ операций (сложений и сравнений) для решения задачи поиска кратчайших путей из одного источника в ориентированном взвешенном графе, имеющем $|V|$ вершин.

- Ваши вопросы?
- Контакты лектора:
  apanovich_09@mail.ru