
potashnikov.com

КРАТЧАЙШИЕ ПУТИ В ГРАФЕ
ТОПОЛОГИЧЕСКАЯ СОРТИРОВКА

Лекция 17-18

План лекции

- Кратчайшие пути
  - Алгоритм Дейкстры
  - Алгоритм Беллмана-Форда
  - Алгоритм Флойда-Уоршелла
- Топологическая сортировка

Кратчайшие пути

Пусть $G = (V, E)$ – ориентированный граф и $w: E \longrightarrow R^+$ -- функция стоимости ребер G

Длиной ребра e называется значение $w(e)$

Длиной пути $p = (v_0, v_1, \ldots, v_k)$ называется сумма $w(p) = \sum w(v_{i-1}, v_i)$ длин ребер, входящих в p

Кратчайшие пути

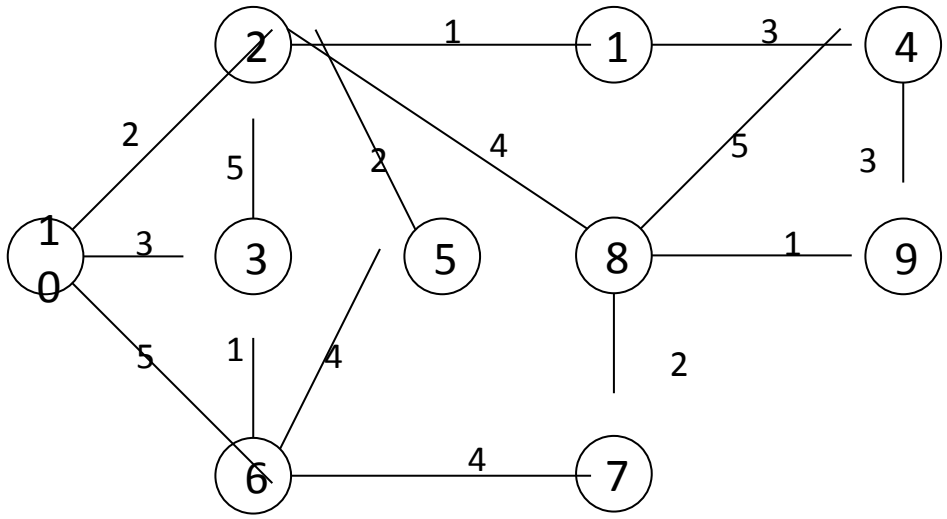
Длиной кратчайшего пути из u в v называется $\delta(u, v) = \min \{ w(p) \mid p = (u, \ldots, v)$ путь в графе $G \}$

– Считаем минимум пустого множества $\min \varnothing = \infty$

Кратчайшим путём из u в v называется путь из u в v, для которого $w(p) = \delta(u, v)$

– Кратчайших путей может быть несколько

Пример



Кратчайшие пути из вершины 10:

V	Длина	Путь через
1	3	2
2	2	
3	3	
4	6	2, 1
5	4	2
6	4	3
7	8	2, 8 или 3, 6
8	6	2
9	7	2, 8

Корректность определения и рёбра отрицательной длины

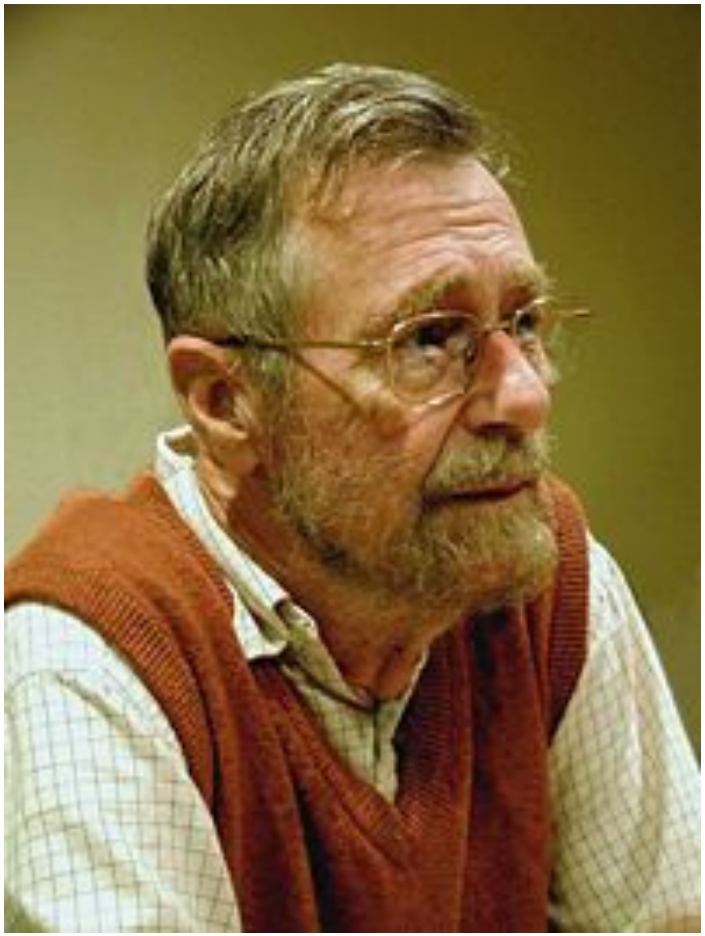
- Условие $w(e) >= 0$ можно заменить на менее строгое условие отсутствия циклов отрицательной длины

- Если циклов отрицательной длины нет, то длина кратчайшего пути определена корректно
  - Кратчайший путь не содержит циклов
  - Путей без циклов конечное число

- Если есть цикл отрицательной длины, то для любых вершин u, v из такого цикла $\delta(u, v) = -\infty$, но *кратчайшего* пути нет
  - длину любого пути из u в v можно уменьшить, обойдя цикл ещё один раз

Алгоритм Дейкстры -- схема

- Э́дсгер Ви́бе Де́йкстра 1930 – 2002
  - Edsger Wybe Dijkstra
- Dijkstra, E. W. (1959). "A note on two problems in connection with graphs". Numerische Mathematik 1: 269–271.

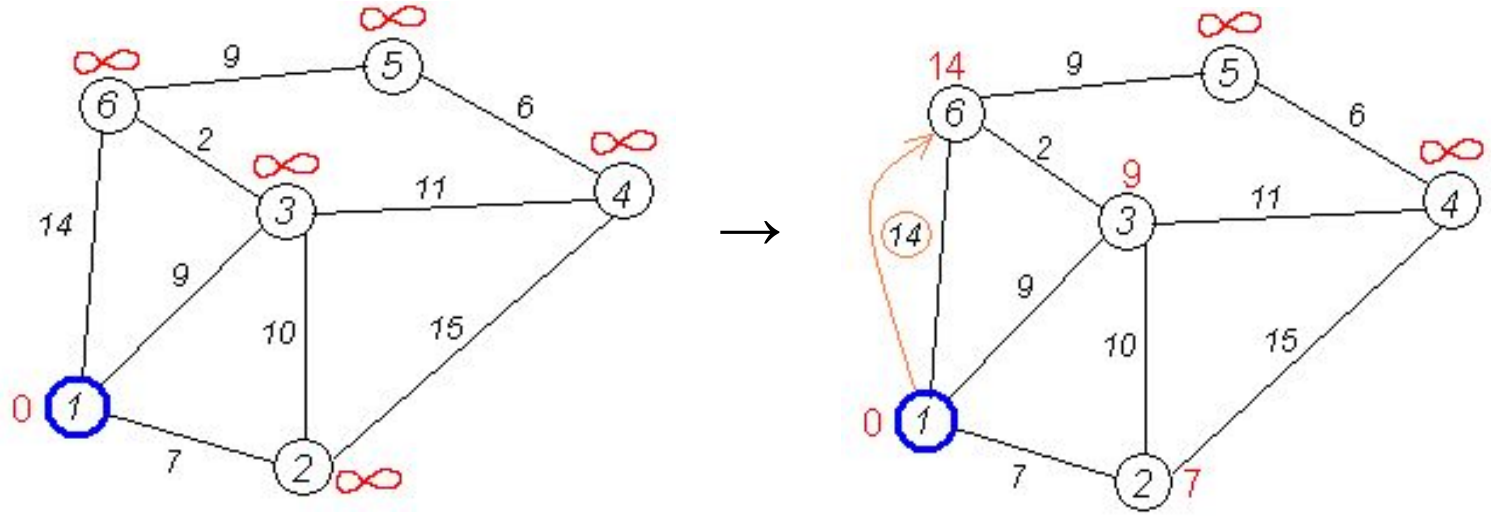


Алгоритм Дейкстры -- схема

- Вычисляем расстояния от вершины-источника до остальных вершин графа
- Строим множество S вершин графа, кратчайшие расстояния от которых до источника известны
- На каждом шаге
  - добавляем в S вершину v_{min}, которая ближе всего к источнику среди оставшихся вершин V \ S
  - обновляем расстояния от источника до соседей v_{min}

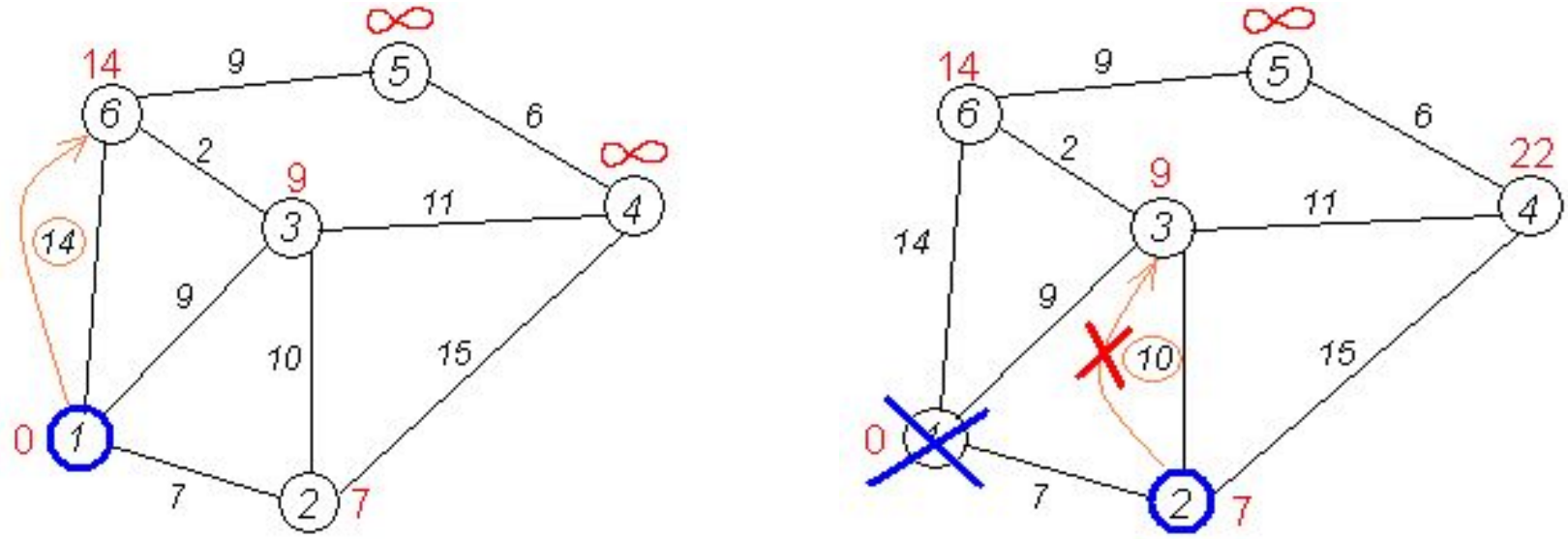
В каком алгоритме используется

Пример (Википедия) – шаг 1



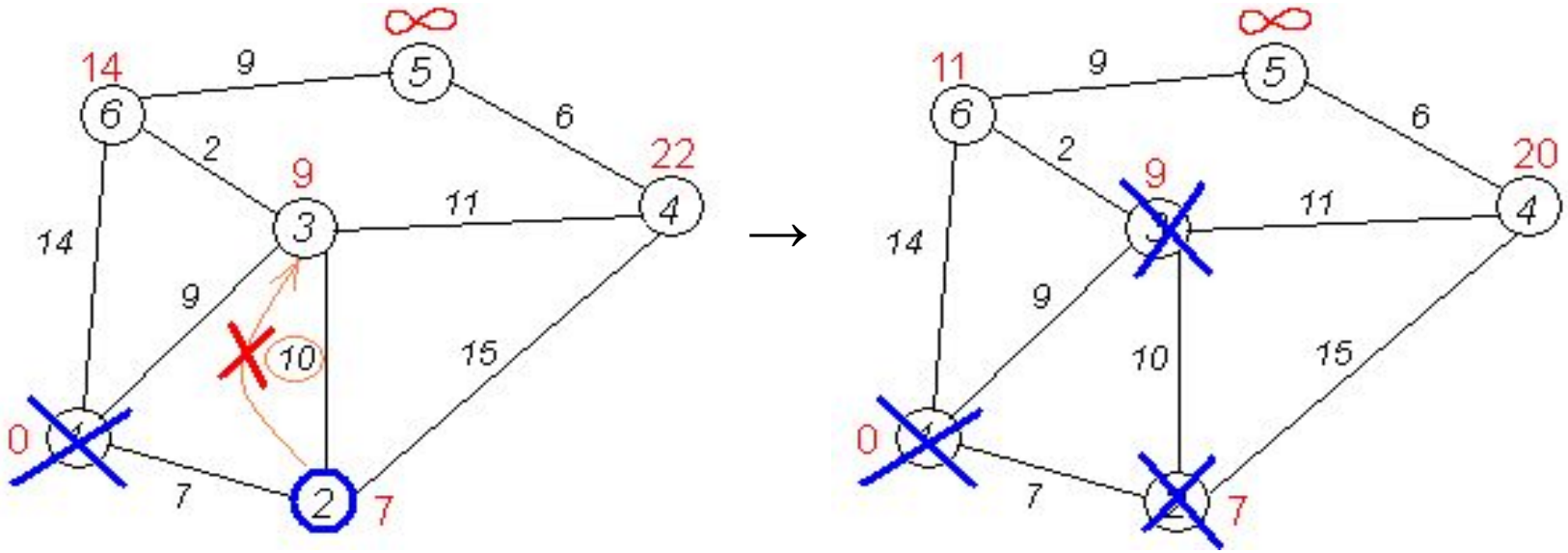
- S = ∅, d[] = { 0 , ∞ , ∞ , ∞ , ∞ , ∞ }
- ==> v_{min} = 1
- ==> S = {1}
- ==> d[] = { 0 , 7 , 9 , ∞ , ∞ , 14 }

Пример – шаг 2



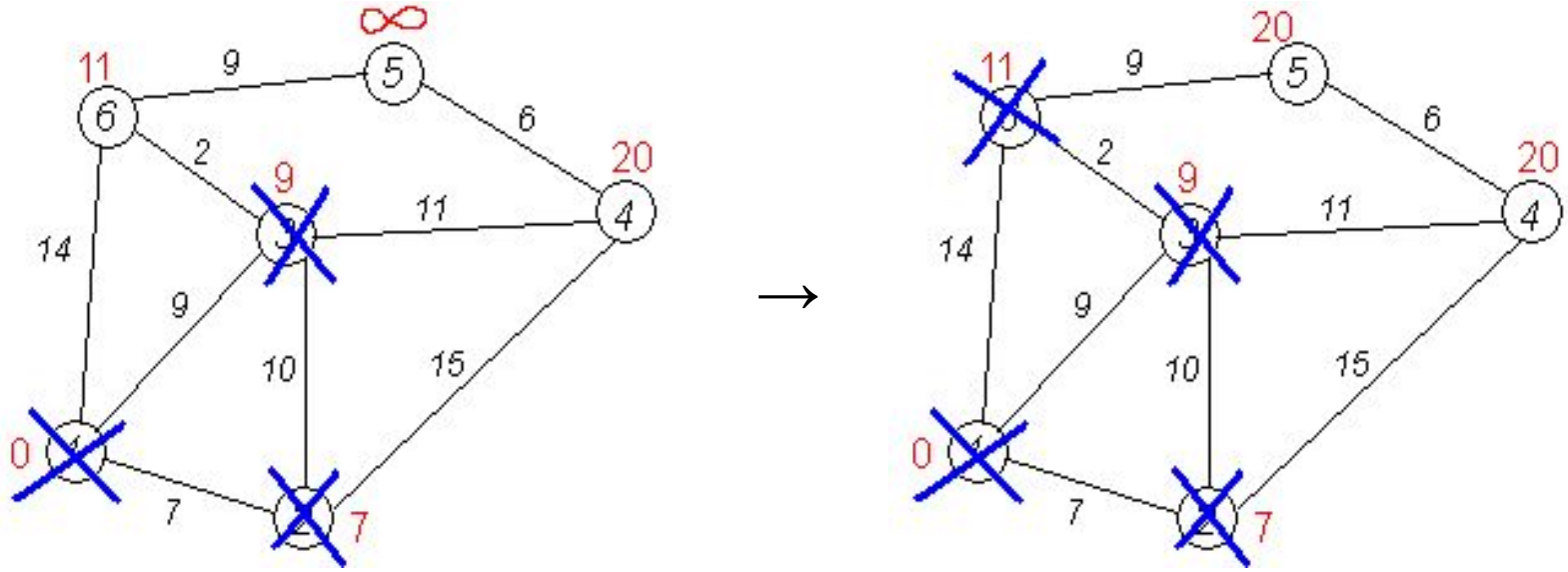
- S = {1}, d[] = { 0 , 7 , 9 , ∞ , ∞ , 14 }
- ==> v_{min} = 2
- ==> S = {1, 2}
- ==> d[] = { 0 , 7 , 9 , 22 , ∞ , 14 }

Пример – шаг 3



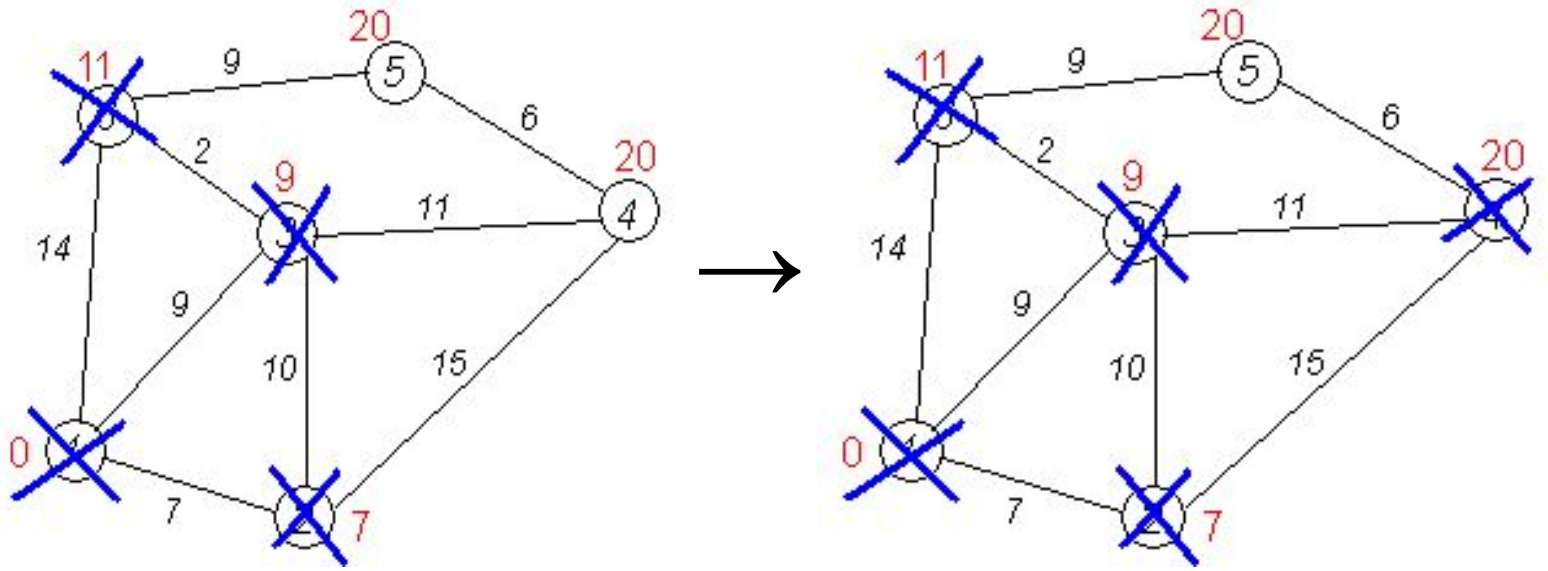
- S = {1, 2}, d[] = { 0 , 7 , 9 , 22 , ∞ , 14 }
- ==> v_{min} = 3
- ==> S = {1, 2, 3}
- ==> d[] = { 0 , 7 , 9 , 20 , ∞ , 11 }

Пример – шаг 4



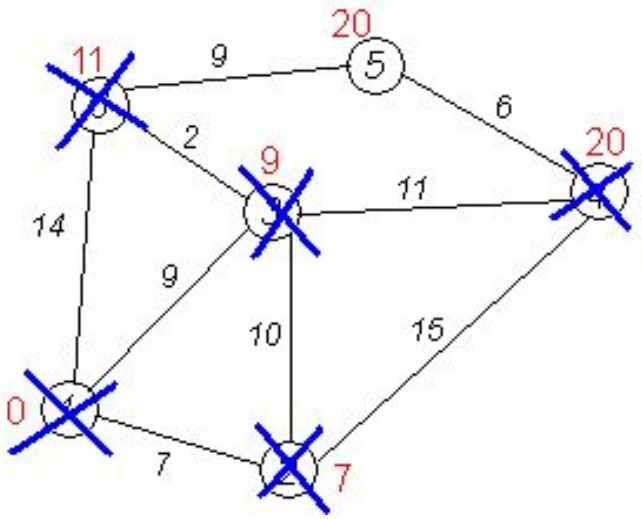
- S = {1, 2}, d[] = { 0 , 7 , 9 , 20 , ∞ , 11 }
- ==> v_{min} = 6
- ==> S = {1, 2, 3, 6}
- ==> d[] = { 0 , 7 , 9 , 20 , 20 , 11 }

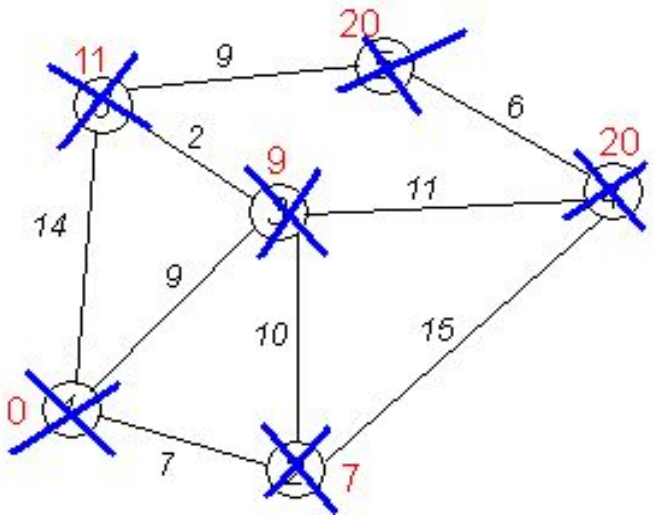
Пример – шаг 5



- S = {1, 2, 3, 6}, d[] = { 0 , 7 , 9 , 20 , 20 , 11 }
- ==> v_{min} = 4 (зависит от того, как мы ищем минимум)
- ==> S = {1, 2, 3, 4, 6}
- ==> d[] = { 0 , 7 , 9 , 20 , 20 , 11 }

Пример – шаг 6





- S = {1, 2, 3, 6}, d[] = { 0 , 7 , 9 , 20 , 20 , 11 }
- ==> v_{min} = 5
- ==> S = {1, 2, 3, 4, 5, 6}
- ==> d[] = { 0 , 7 , 9 , 20 , 20 , 11 }

Алгоритм Дейкстры

// S -- множество вершин v, для которых min
// расстояние δ(s,v) от источника s уже найдено
// d[v] – известная оценка сверху для δ(s,v)

- Пока S != V
  - Выбираем вершину u ∈ V \ S с наименьшим d[u]
  - Добавляем вершину u к множеству S
  - Обновляем d[v] для всех соседей u

Почему шаг 3 не нарушает корректность алгоритма?

Алгоритм Дейкстры

Dijkstra(граф G = (V,E), вершина s из V)

- S ← {s};
- D[s] ← 0;
- D[v] = w (s, v) для всех v != s;
- пока S ≠ V
  - выбрать вершину u ∈ V \ S, для которого D[u] принимает наименьшее значение;
  - добавить u к S;
  - для каждого соседа v вершины u
    - D[v] = min (D[v], D[u] + w(u, v));

Алгоритм Дейкстры C

```
// N -- число вершин в графе, s – источник
void sp(const int G[], int N, int s, int dmin[])
{
	int *blue = calloc(N, sizeof(*blue)), i;
	if (blue == 0) return;
	for (i = 0; i < N; ++i) dmin[i] = G[s*N+i];
	dmin[s] = 0; blue[s] = 1; // Красим вершины множества S в синий цвет
	for (i = 0; i < N; ++i) {
		int jmin = -1, j;
		for (j = 0; j < N; ++j)
			if (!blue[j] && (jmin == -1 || dmin[j] <= dmin[jmin]))
				jmin = j;
		if (jmin == -1) break; // Достижимые вершины кончились
		blue[jmin] = 1; // расстояние найдено
		for (j = 0; j < N; ++j) // Обновляем расстояния до соседей jmin
			if (!blue[j] && d[jmin]+G[j*N+jmin] < dmin[j])
				dmin[j] = dmin[jmin]+G[j*N+jmin];
	}
	free(blue);
}
```

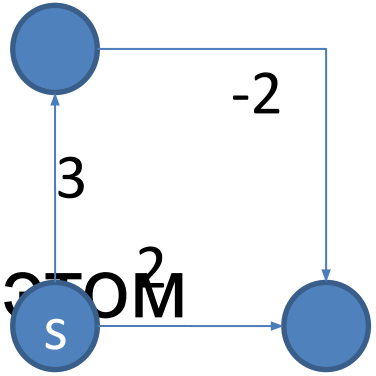
Сложность алгоритма Дейкстры по времени

- Сложность операции поиска min и операции обновления элемента зависит от типа dmin
  - Массив
  - Пирамида из пирамидальной сортировки
  - Фибоначчиева куча (Тарьян, Фредман 1984)
    - Обязательно знать оценки сложности поиска min и обновления элемента
    - Реализация операций и доказательство оценок -- по желанию

	Массив	Пирамида	Фибоначчиева куча
Поиск min	Один $O(N)$ Всего $O(N^2)$	Один $O(1)$ Всего $O(N)$	Один $O(\log N)$ Всего $O(N \log N)$
Обновление элемента	Один $O(1)$ Всего $O(M)$	Один $O(\log N)$ Всего $O(M \log N)$	Один $O(1)$ Всего $O(M)$
Всего	$O(N^2+M)$	$O(N+M \log N)$	$O(N \log N+M)$
Для простоты считаем, что присваивания, сравнения занимают единицу времени			

Алгоритм Беллмана-Форда

- Если есть ребера длины < 0, то алгоритм Дейкстры может вычислить $dmin[v_{min}] > \delta(s, v_{min})$
  - если есть вершина $v \in S$ и путь p отрицательной длины из v в v_{min} по вершинам $V \setminus S$ – см. рисунок



- Как вычислять кратчайшие пути в этом случае?

Алгоритм Беллмана-Форда

- Вычисление кратчайших путей в графе с ребрами и/или циклами отрицательной длины за O(|V| × |E|) операций

- Bellman, Richard (1958). "On a routing problem". Quarterly of Applied Mathematics 16: 87–90. MR 0102435.
- Ford, Lester Randolph, Jr.; Fulkerson, D. R. (1962). Flows in Networks. Princeton University Press.

Алгоритм Беллмана-Форда -- схема

- /* Вычисляем длины кратчайших путей от источника до вершин графа в порядке увеличения числа ребер в пути */
- dmin[s] = 0, dmin[v] = ∞ для v != s
- Для i = 1, …, |V|-1 // кратчайшем пути <= |V|-1 ребер
  - Для каждой вершины v
    - dnext[v] = min { dmin[v], dmin[u]+w(u,v) }
  - dmin = dnext
  - // обычно значение min{…} записывают сразу в dmin
- Если dmin[u]+w(u,v) < dmin[v] для одного из ребер (u, v), то G содержит цикл отрицательной длины

Алгоритм Беллмана-Форда C

```
int BellmanFord(const int G[], int N, int s, int dmin[])
{
   int i, v, u;
   for (i = 0; i < N; ++i) dmin[i] = G[s*N+i];
   dmin[s] = 0;
   for (i = 1; i < N; ++i)
      for (v = 0; v < N; ++v)
         for (u = 0; u < N; ++u)
            if (dmin[v] > dmin[u] + G[u*N+v])
               dmin[v] = dmin[u] + G[u*N+v];
   for (v = 0; v < N; ++v)
      for (u = 0; u < N; ++u)
         if (dmin[v]>dmin[u]+G[u*N+v]) return 0;
   return 1;
}
```

Алгоритм Флойда-Уоршелла

- Вычисление кратчайших расстояний между всеми парами вершин графа
- Warshall, Stephen (January 1962). "A theorem on Boolean matrices". Journal of the ACM 9 (1): 11–12
- Floyd, Robert W. (June 1962). "Algorithm 97: Shortest Path". Communications of the ACM 5 (6): 345

Алгоритм Флойда-Уоршелла -- схема

// Нумеруем вершины числами от 1 до N
// Вычисляем dmin[i, j] = кратчайшее расстояние от вершины i до
// вершины j
// Сложность по времени O(N^3)
// Вычисляем кратчайшие расстояния d[k, i, j] от вершины i до вершины j
// для путей с промежуточными вершинами из множества {1,…,k}

- Для i, j = 1, …, N d[0, i, j] =
  - 0, если i=j
  - w(i, j), если (i, j) ∈ E
  - ∞ , если (i, j) ∉ E
- Для k = 1, …, N
  - Для i, j = 1, …, N
    - d[k, i, j] = min(d[k-1, i, j], d[k-1, i, k] + d[k-1, k, j])
- // dmin[i, j] == d[N, i, j]

Как построить транзитивное замыкание G с помощью алгоритма Флойда-Уоршелла?
Т. з. G – это граф H = (V, Et)
Et = { (u, v) | в G есть путь из u в v }

Топологическая сортировка

- Топологической сортировкой ориентированного графа G = (V, E) называется нумерация T вершин V такая, что для каждой дуги (v, u) графа G верно T(v) < T(u)
- Если возможна топологическая сортировка G, то в G нет циклов
  - Почему?
- Если в G нет циклов, то возможна топологическая сортировка G
  - Алгоритм топологической сортировки

Алгоритм топологической сортировки

Вход

Ориентированный граф G = (V, E)

Выход

Топологическая сортировка G – нумерация T такая, что что для каждой дуги (v, u) графа G верно T(v) < T(u)

t = 1;

Для i = 1, …, N

Найти вершину v такую, что нет дуг входящих в v

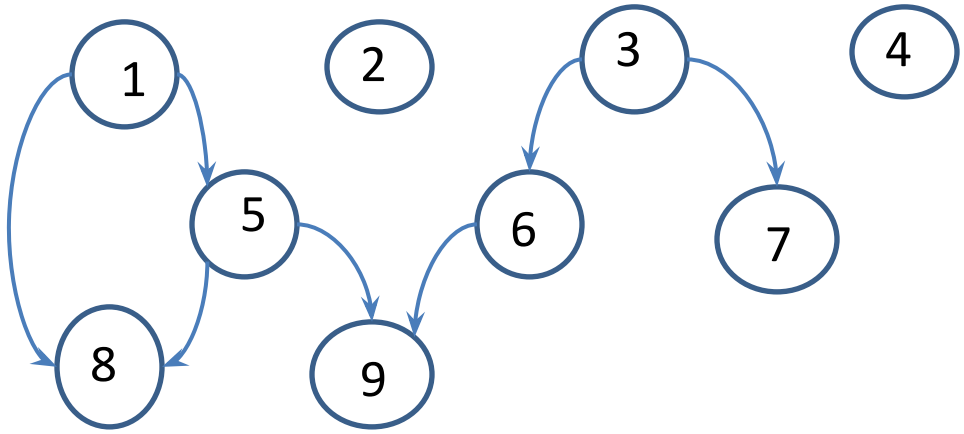
Если нет такой вершины, то граф содержит цикл – почему?

T[v] = t;

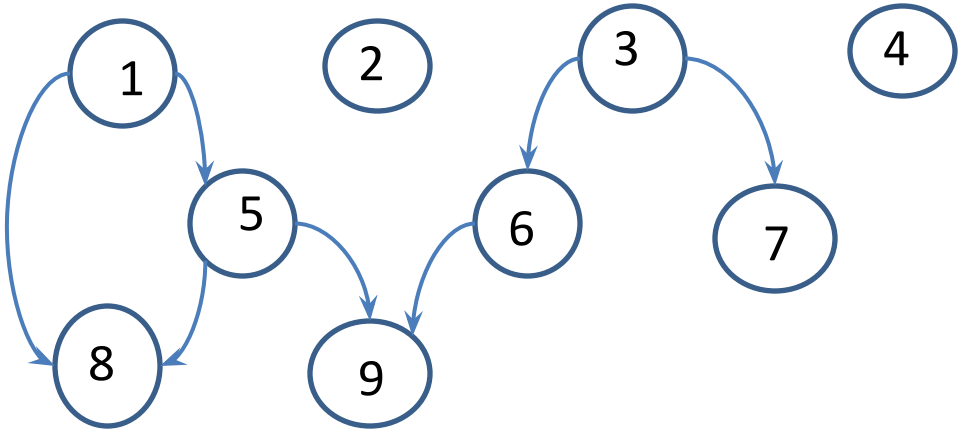
t = t+1;

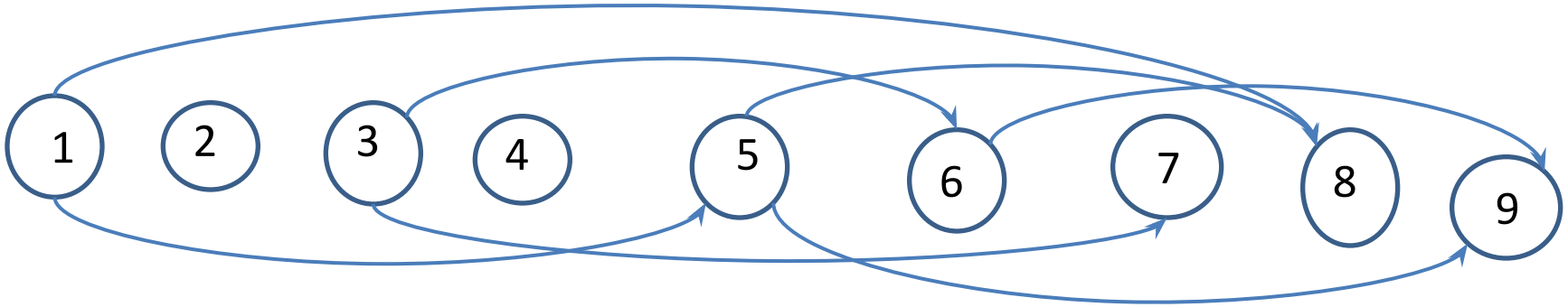
Удалить из E все дуги исходящие из v, удалить v из V

Топологическая сортировка -- пример

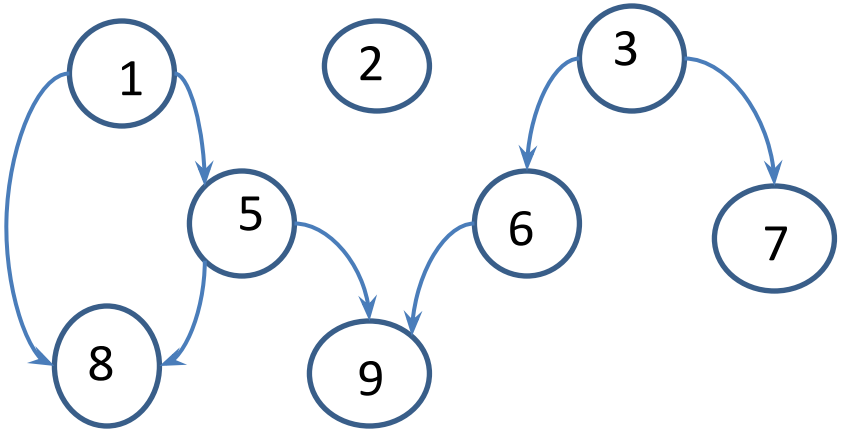


Топологическая сортировка -- пример





Топологическая сортировка с матрицей смежности

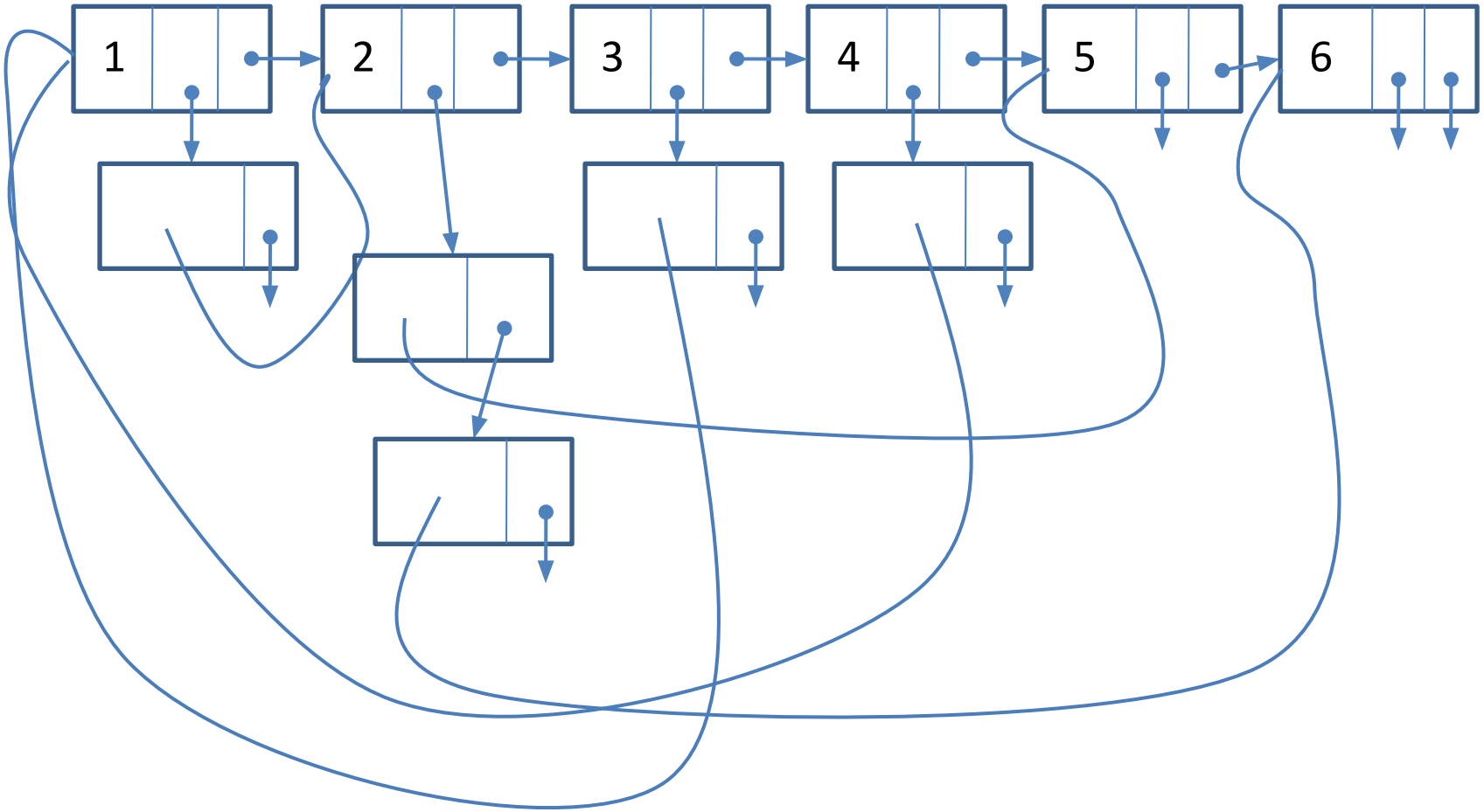


	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	1	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	1	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	1
6	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

1. Найти вершину, в которую не входит ни одна дуга (это нулевой столбец).
   Удалить все выходящие из нее дуги (обнулить соответствующую строку)
2. Пока не перебрали все вершины, повторять шаг 1.

Топологическая сортировка с иерархическими списками

1< 2; 3< 1; 4<1; 2< 5; 2< 6;



Топологическая сортировка – связь с частичным порядком

- Частичным порядком на множестве A называется отношение R на A такое, что
  1. верно a R b, верно b R c => верно a R c (транизтивность)
  2. не верно a R a (иррефлексивность)
- Следствие свойств (1) и (2)
  - если верно a R b, то не верно b R a (антисимметричность)

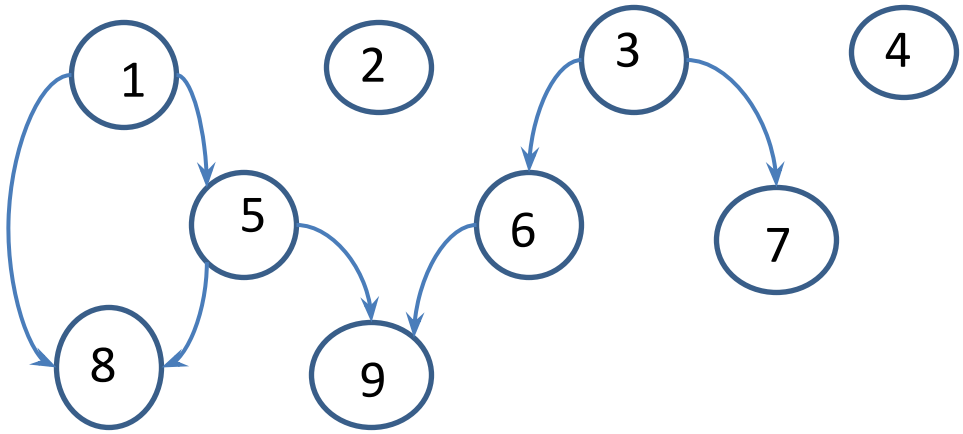
Топологическая сортировка – связь с частичным порядком

- Примеры частичных порядков
  - Зависимость по записи/чтению данных между операторами в программе без циклов
  - Зависимость курсов в учебной программе по излагаемому материалу
  - Зависимость строительных и т.п. работ

- Томас Хэрриот (англ. Thomas Harriot) (1560 год — 2 июля 1621 года) — английский астроном, математик, этнограф и переводчик
  - Придумал знаки для операций сравнения: «>» (больше) и «<» (меньше)
  - Впервые привез картофель в Великобританию и Ирландию

Топологическая сортировка – связь с частичным порядком

- Если G = (V, E) -- ациклический граф, то отношение R = E -- частичный порядок на множестве V
- Если отношение R частичный порядок на конечном множестве A, то G = (A, R) -- ациклический граф
- Пример
  - 1 < 8, 1 < 5, 5 < 8,
  - 5 < 9, 6 < 9, 3 < 6,
  - 3 < 7



Топологическая сортировка – связь с частичным порядком

- Линейный порядок R на множестве A -- это такой частичный порядок, что для любые a и b из A сравнимы
  - либо a R b, либо b R a, либо a = b

- Линейный порядок на конечном множестве $A = \{a_1, a_2, \ldots, a_n\}$ можно задать перестановкой $a_{p1}, a_{p2}, \ldots, a_{pn}$ такой, что a_{pi} R a_{pi+1}

Топологическая сортировка – связь с частичным порядком

- Частичный порядок R на множестве A вложен в линейный порядок R', если R' – это линейный порядок и $R \subseteq R'$, т. е. $a \; R \; b$ влечет $a \; R' \; b$ для всех a и b из A

- Алгоритм топологической сортировки вычисляет линейный порядок, в который вложен данный частичный порядок

Заключение

- Кратчайшие пути
  - Алгоритмы Дейкстры, Беллмана-Форда, Флойда-Уоршелла
- Топологическая сортировка
  - Алгоритм, связь с отношениями порядка

Транзитивное замыкание графа

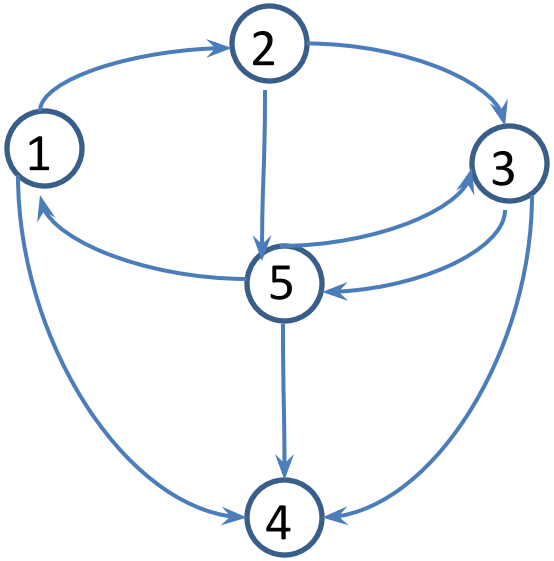
Пусть $G = (V, E)$ ориентированный граф.

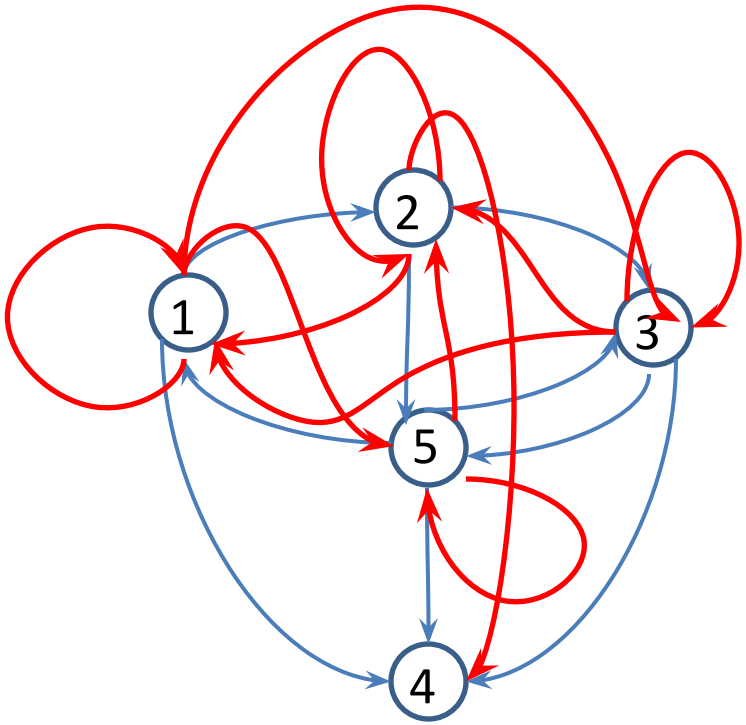
Транзитивным замыканием графа G называется граф $G' = (V, E')$, в котором из вершины v в вершину w идет ребро ⇔ существует путь (длины 0 или больше) из v в w в графе G.

E':

$(a, b) \in E \;\&\; (b, c) \in E \Rightarrow (a, b) \in E' \;\&\; (b, c) \in E' \;\&\; (a, c) \in E'$

Построение транзитивного замыкания графа. Пример





Обозначим через $t_{ij}^{(k)}$ наличие пути из вершины с номером i в вершину с номером j с промежуточными вершинами из множества {1, 2, …, k}. M – матрица смежностей графа G.

$$t_{ij}^{(k)} = \begin{cases} M[i, j], & \text{если } k = 0, \\ t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}), & \text{если } k \geq 1 \end{cases}$$

$T^{(n)}$ содержит искомое решение.

Алгоритм построения транзитивного замыкания графа

Tranzitive_Clusure(M, n)

{

$T^{(0)} \leftarrow M$;

for k ← 1 to n do

for i ←1 to n do

for j ← 1 to n do

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

return $T^{(n)}$;

}

Пусть $G = (V, E)$ – заданный граф.

Для каждой вершины $v \in V$ мы будем помнить ее предшественника.

Релаксация – постепенное уточнение верхней оценки на вес кратчайшего пути в заданную вершину.

Свойства оптимальности.

Лемма 1. Отрезки кратчайших путей являются кратчайшими: Если $p(v_1, v_2, \ldots, v_k)$ – кратчайший путь из v_1 в v_k и $1 \le i \le j \le k$, то $p_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ есть кратчайший путь из v_i в v_j

Следствие 1. Рассмотрим кратчайший путь p из s в v. Пусть (u,v) – последнее ребро этого пути. Тогда $\delta(s,v) = \delta(s,u) + w(u,v)$.

Следствие 2. Для любого ребра $(u,v) \in E$ справедливо $\delta(s,v) \le \delta(s,u) + w(u,v)$.

Техника релаксации

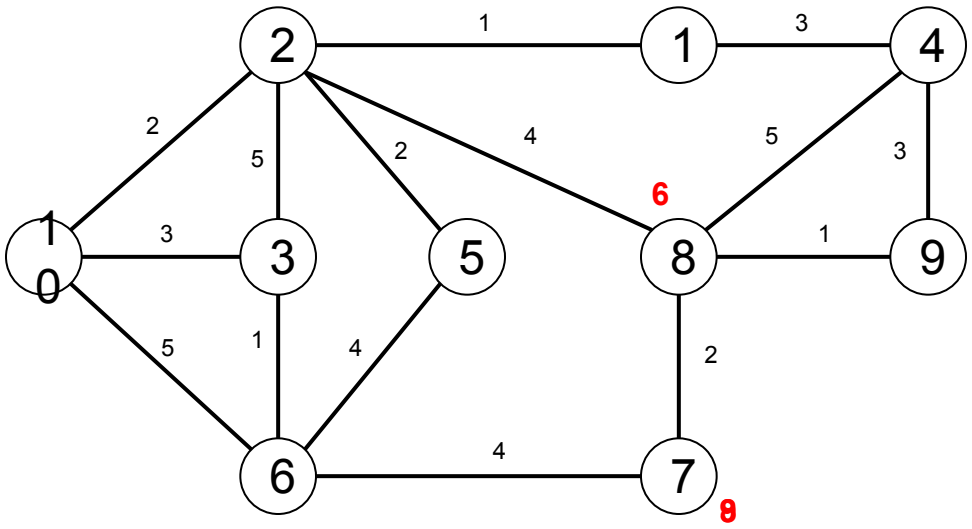
Для каждого ребра (u,v) храним $d[v]$ – верхнюю оценку кратчайшего пути из s в v.

```
Initialize (G,s){
     for (для ∀v ∈ V) {
         d[v] ← ∞
          П[v] ← NULL;
    }
  d[s] ← 0;
}
```

Релаксация ребра (*u*,*v*):
значение *d*[*v*] уменьшается до *d*[*v+w(u,v)*]
(если второе второе значение меньше первого)

```
Relax (u, v, w) {
  If (d[v] >  d[u] +w(u,v)){
   d[v] = d[ u] +w(u,v);
   Π[v] ← u;
  }
}
```

Релаксация ребра при поиске кратчайших путей.



Пусть уже найдены оценки кратчайших путей для вершин, соединенных красным ребром.

```
d[8] = 6;
d[7] = 9
```

Релаксация ребра (u, v):

```
if (d[u] + w(u,v) < d[v]) d[v] = d[u] +
w(u,v);
```

Релаксация ребра (7, 8):

```
9 + 2 > 6
```

Релаксация ребра (8, 7):

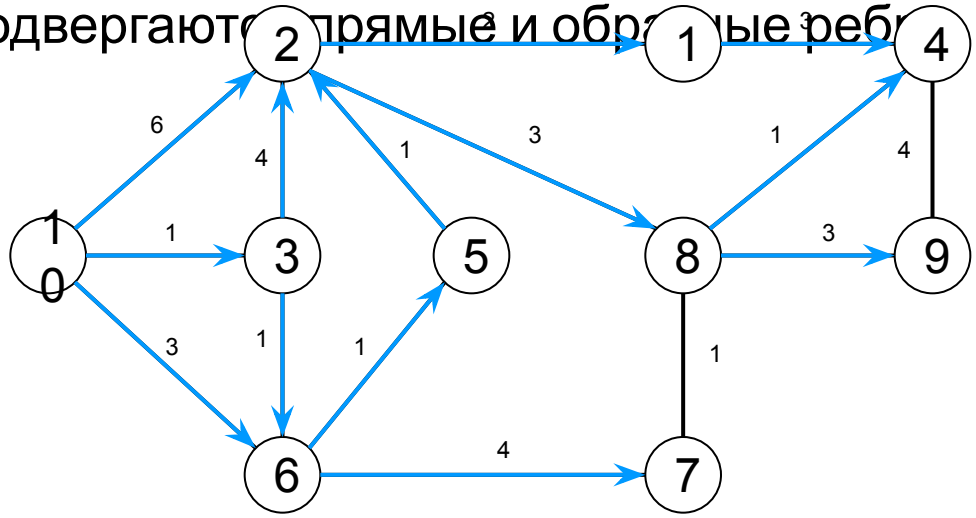
```
6 + 2 < 9   ⇒   d[7] = 8
```

Алгоритм Дейкстры

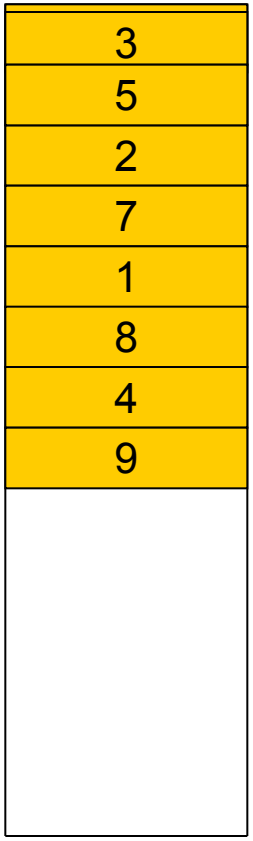
```
Dijkstra(G,w,s){
  Initialize(G,s);
  S ← ∅;
  Q ← V;                    //очередь с приоритетами
  While (Q ≠ ∅){
    u ← Exstract_min(Q); //выбрать ближайшую
    S ←  S U {u};
    for (для ∀v ∈ Adj[u])
      Relax ( u, v, w);
  }
}
```

Приме
р.

Реализация с использованием очереди с приоритетами. Приоритет – текущая
величина найденного расстояния от начальной вершины. Релаксации подвергаются прямые и обратные ребра



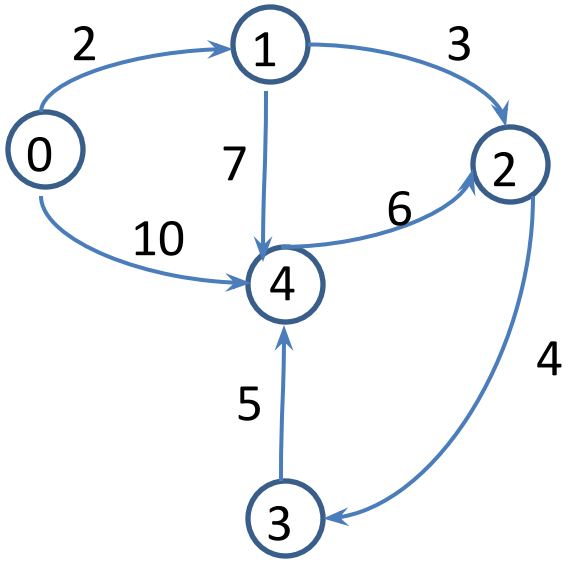
Очередь



n	1	2	3	4	5	6	7	8	9	10
π	2	5	10	8	6	3	6	2	8	
d	6	4	1	8	3	2	6	7	10	0

Реализация с дополнительным массивом - $O(n^2)$

Массив $D[v]$ содержит стоимость текущего кратчайшего пути из s в v.



Пример

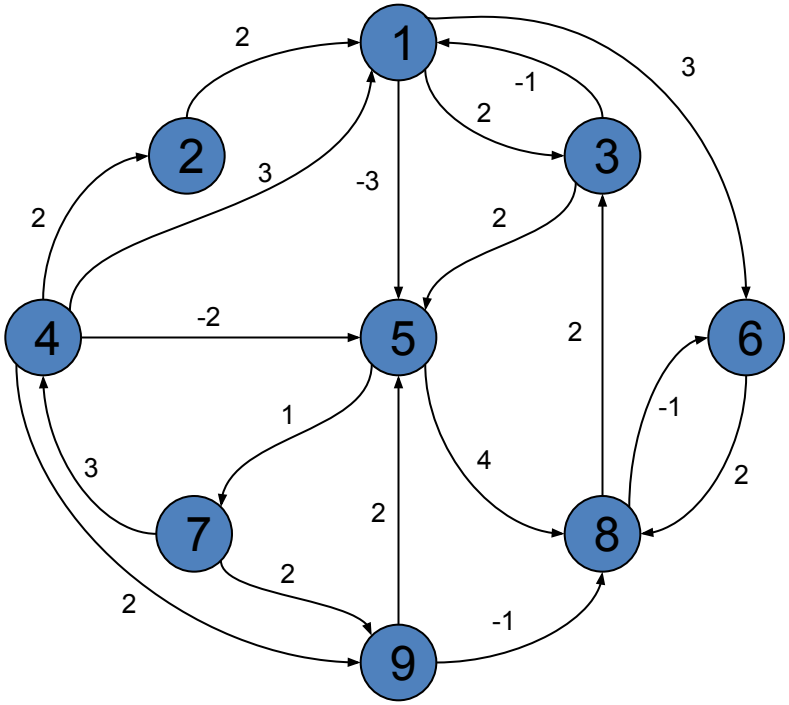
№	S	u	D[u]	D[1]	D[2]	D[3]	D[4]
0	{0}	-	-	2	+∞	+∞	10
1	{0, 1}	1	2	2	5	+∞	9
2	{0, 1, 2}	2	5	2	5	9	9
3	{0, 1, 2, 3}	3	9	2	5	9	9
4	{0, 1, 2, 3, 4}	4	9	2	5	9	9

Компьютерная сеть была названа ARPANET, все работы финансировались
за счёт Министерства обороны США. Затем сеть ARPANET начала активно
расти и развиваться, её начали использовать учёные из разных областей
науки. В 1973 году к сети были подключены первые иностранные
организации из Великобритании и Норвегии, сеть стала международной.
Стоимость пересылки электронного письма по сети ARPANET составляла
50 центов.
В 1984 году у сети ARPANET появился серьёзный соперник,
Национальный фонд науки США (NSF) основал обширную
Межуниверситетскую сеть NSFNet, которая имела гораздо бóльшую
пропускную способность (56 кбит/с), нежели ARPANET.
В 1990 году сеть ARPANET прекратила своё существование, полностью
проиграв конкуренцию NSFNet.

Кратчайшие пути в ориентированном графе

1. Если в ориентированном графе нет дуг с отрицательным весом, то алгоритм Дейкстры работает точно так же, как и в случае неориентированных графов.

2. Если в ориентированном графе нет циклов с отрицательным весом, то можно применить алгоритм Беллмана – Форда.



n	1	2	3	4	5	6	7	8	9
π	3	4	8		4	8	5	9	7
d	1	2	2	0	-2	-1	-1	0	1

И так далее…
В конце концов получится…

Floyd-Warshall(M, n) {

$D^{(0)} \leftarrow M$;

for k ← 1 to n do

for i ←1 to n do

for j ← 1 to n do

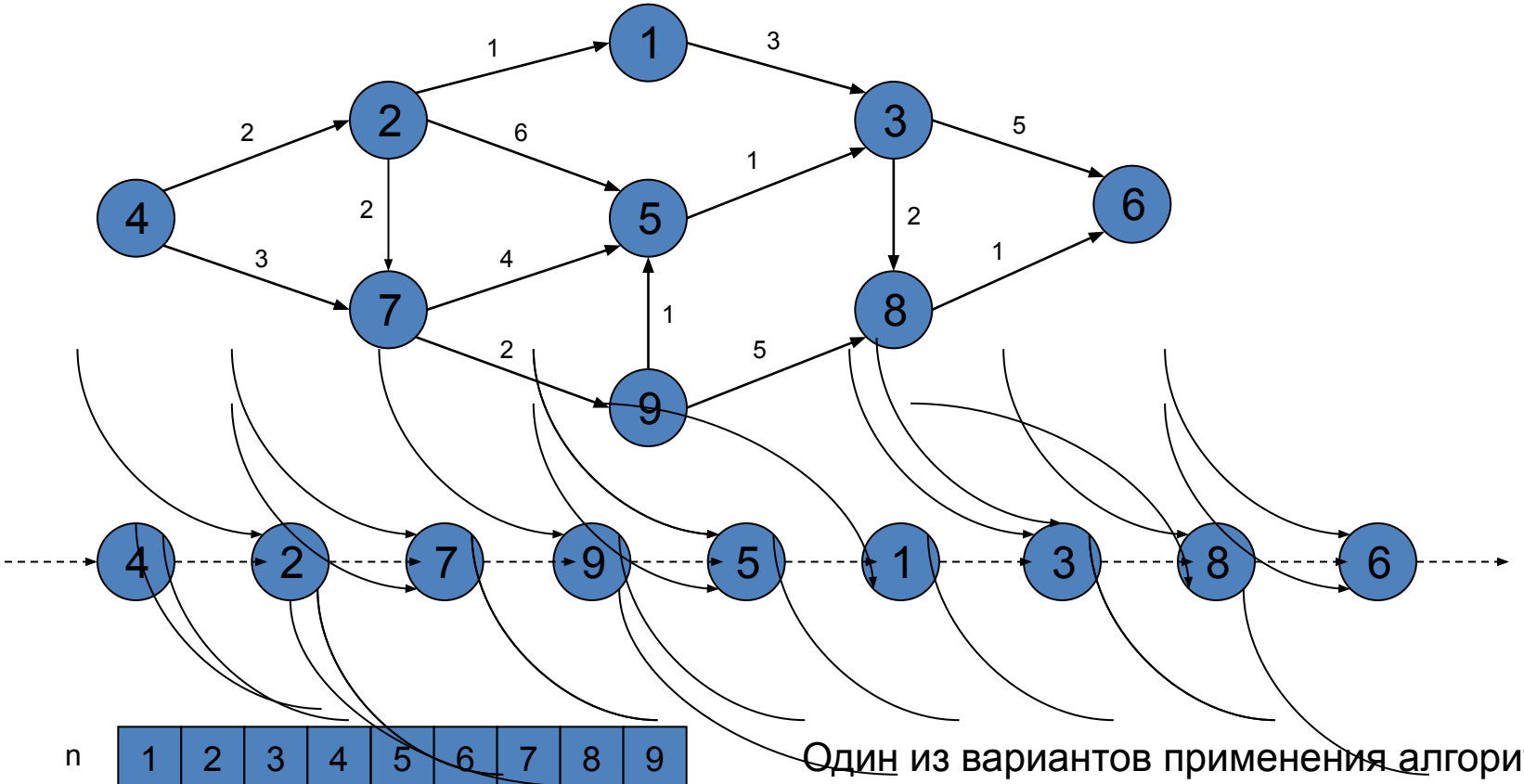
$d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$;

return $D^{(n)}$;

}

Кратчайшие пути в ориентированном графе

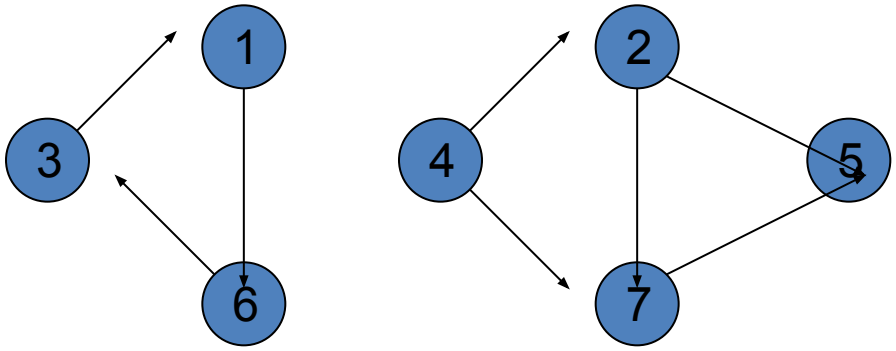
3. Если в ориентированном графе нет циклов, то можно провести топологическую сортировку вершин, после чего выполнить релаксацию исходящих дуг в порядке возрастания номеров вершин.



n	1	2	3	4	5	6	7	8	9
π	2	4	1		9	8	4	3	7
d	3	2	6	0	6	9	3	8	5

Один из вариантов применения алгоритма: нахождение критического пути.

Алгоритм «умножения матриц».



	1	2	3	4	5	6	7
1	0	0	0	0	0	1	0
2	0	0	0	0	1	0	1
3	1	0	0	0	0	0	0
4	0	1	0	0	0	0	1
5	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0
7	0	0	0	0	1	0	0

Пусть матрица $G^{(l)}$ представляет собой граф путей длиной l (то есть в матрице единица находится в ячейке (u,v), если в исходном графе существовал путь из u в v длиной не больше l).

Тогда матрица $G^{(1)}$ – это матрица смежности исходного графа G,
$G^{(n)}$ – матрица смежности его транзитивного замыкания (очевидно, что если в графе существует путь длины, большей *n*, то существует и путь, длины не большей *n*).

Алгоритм нахождения транзитивного замыкания: если удается вычислить $G^{(l+1)}$ по $G^{(l)}$, то можно, начав с матрицы G, за *n* шагов получить матрицу $G^{(n)}$.