Глава 2.
Связность графов

2.1. Маршруты, цепи, циклы

Маршруты

Определения. Конечная последовательность вершин и ребер

$$x_0, u_1, x_1, u_2, \dots x_{l-1}, u_l, x_l \ (l \geq 0), \quad x_i \in V,\ u_i \in E$$

в которой соседние ребра имеют общую вершину, называется маршрутом из вершины x_0 в вершину x_l, или маршрутом, соединяющим x_0 с x_l.

В случае $x_0 = x_l$ маршрут называется циклическим. Число l называется длиной маршрута. Маршрут не является частью графа, так как порядок его обхода существенен.

Понятие маршрута не зависит от ориентации ребер.

Цепи и циклы

Маршрут называется цепью, если все ребра в нем различны. Цепь называется простой, если все ее вершины различны.

Циклическая цепь называется циклом (следовательно, в цикле все ребра различны). Цикл называется простым, если все его вершины различны, кроме $x_0 = x_l$.

Лемма. Всякий маршрут графа содержит хотя бы одну простую цепь, соединяющую ту же пару вершин.

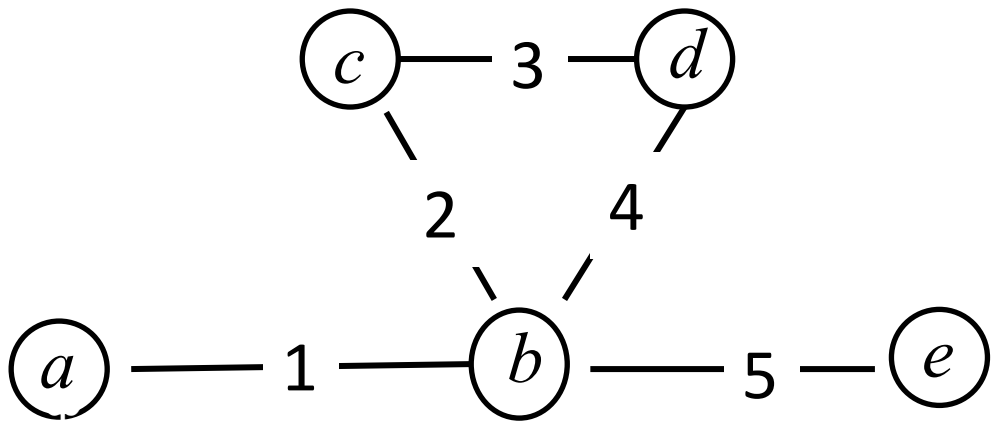
Доказательство

Если x_i -- первая из тех вершин маршрута, которая входит в него более одного раза, а x_j -- последняя из совпадающих с x_i вершин этого маршрута, то его можно заменить более коротким

$$x_0 u_1 x_1 \dots x_i u_{j+1} x_{j+1} u_{j+2} \dots x_{l-1} u_l x_l$$

Если в полученном маршруте есть еще повторяющиеся вер-шины, то снова заменяем его более коротким и так далее, пока не выделим маршрут без повторяющихся вершин.

Приме
р



Маршрут $a\ 1\ b\ 2\ c\ 3\ d\ 4\ b\ 5\ e$ содержит простую цепь $a\ 1\ b\ 5\ e$.

Следствие Всякий кратчайший маршрут между двумя заданными вершинами графа есть простая цепь.

Всякий цикл наименьшей длины при заданной вершине является простым.

Если маршрут рассматривать с учетом ориентации ребер (может быть и по звеньям), то получим соответствующие определения:

ориентированный маршрут (ормаршрут);

ориентированная цепь (орцепь) – путь;

простая орцепь – простой путь.

Задачи о маршрутах

В теории рассматривается ряд задач и алгоритмов определения свойств маршрутов:

существование маршрутов заданной длины;

достижимость вершин;

число маршрутов заданной длины;

компоненты связности и бисвязности;

кратчайшие цепи/пути;

кратчайшие цепи/пути на взвешенных графах.

Существование маршрутов

Рассматривается неориентированный (маршрут не предполагает ориентации) унограф (важен лишь факт смежности вершин).

Такой граф можно рассматривать как симметричное бинарное отношение и его можно задать матрицей смежности R, элементы r_{ij} которой – булевы. r_{ij} =1, если вершины смежны; из вершины x_i существует маршрут длины 1 в вершину x_j.

Элемент матрицы $r_{ij}^{(2)} = \bigvee_{k=1}^{n} r_{ik}^{(1)} \wedge r_{kj}$ определяет существование маршрута длины 2 между этими вершинами. Далее по индукции:

R^2

$$r_{ij}^{(l)} = \bigvee_{k=1}^{n} r_{ik}^{(l-1)} \wedge r_{kj} \qquad (*)$$

маршрут длины l из вершины x_i в вершину x_j существует, если

существует маршрут длины l-1 из вершины x_i в вершину x_k

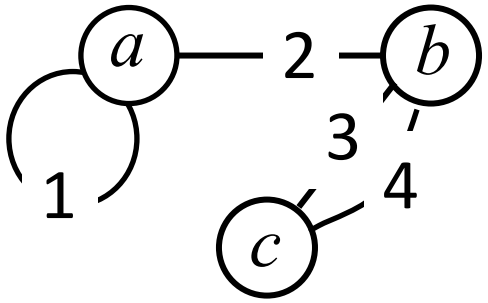
и вершины x_k и x_j

смежны.

Для маршрутов других видов задаются соответствующие матрицы смежности.

Нахождение маршрутов

Зададим граф общего вида тройками вида (x,v,y), где $x,y \in V,\ v \in E$.
Умножение отношений примет следующий вид: если существует маршрут из вершины x_i в вершину x_k длины l-1 и есть ребро $(x_k,\ v,\ x_j)$, то оно добавляется к маршруту длины l.



Длина 1	Длина 2	Длина 3	
$a1a$	$a1a1a$	$a1a1a1a$	
$a2b$	$a1a2b$	…	
$b2a$	$a2b2a$	$a1a2b3c$	…
$b3c$	$a2b3c$	$b2a1a2b$	
$b4c$	$a2b4c$	$b3c4b2a$	
$c3b$	$b2a1a$	…	
$c4b$	$b2a2b$	$c3b4c3b$	
	$b3c3b$	…	
	…		

Число маршрутов

Пусть задан граф общего вида с матрицей смежности R. Элемент r_{ij} определяет число ребер, соединяющих вершины x_i и x_j.

Возведем R в l – ю степень по правилам обычного матричного умножения и рассмотрим элемент $r_{ij}{}^{(l\}}$

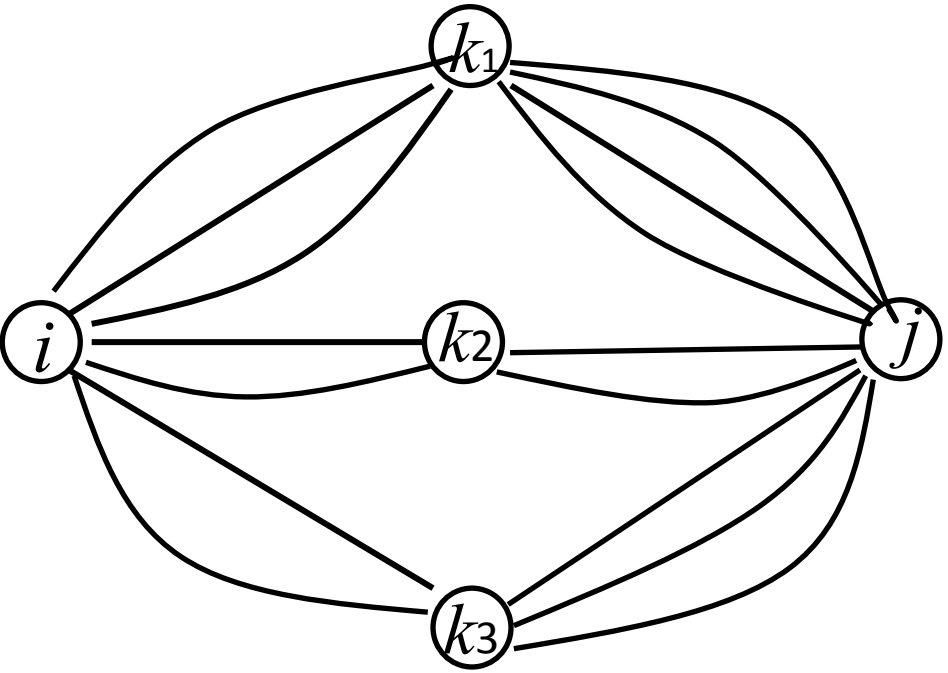
Теорема. Элемент $r_{ij}{}^{(l)}$ равен числу различных маршрутов длины l из вершины x_i в вершину x_j.

Доказательство индукцией по l.

$$r^{(l)}{}_{ij} = \sum_{k=1}^{n} r^{(l-1)}{}_{ik} r_{kj}$$

Число различных маршрутов длины 2 из вершины x_i в вершину x_j через вершину x_k есть

$$r_{ij}^{(2)} = r_{ik} r_{kj}$$

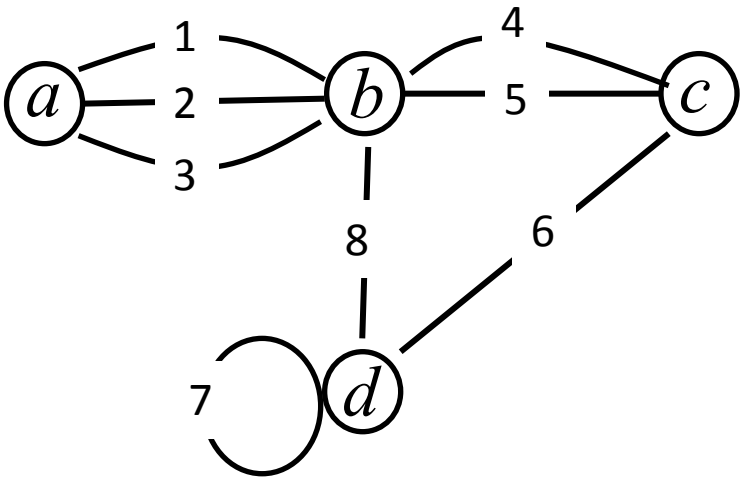


Просуммировав по всем k получим R^2 элемент матрицы

$$r^{(2)}{}_{ij} = \sum_{k=1}^{n} r^{(1)}{}_{ik} r_{kj}$$

Далее по индукции.

Здесь сложение и умножение – обычные арифметические операциии.



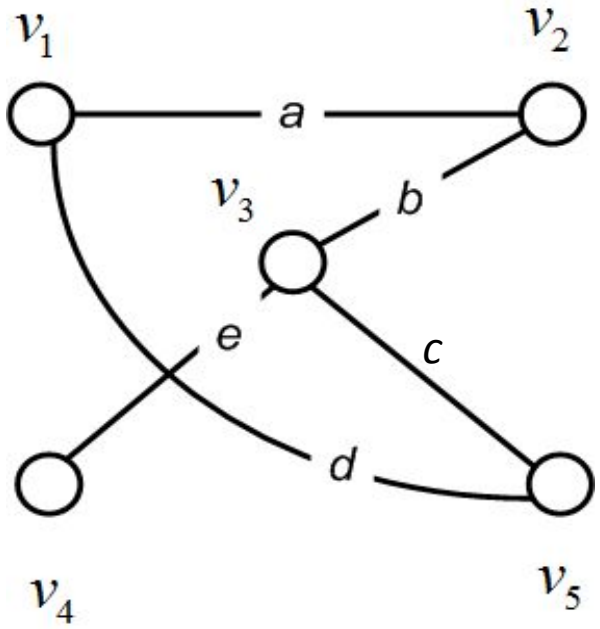
Приме
р

R	a	b	c	d
a	0	3	0	0
b	3	0	2	1
c	0	2	0	1
d	0	1	1	1

	a	b	c	d
a	9	0	6	3
b	0	14	1	3
c	6	1	5	3
d	3	3	3	3

	a	b	c	d
a	0	42	3	9
b	42	5	31	18
c	3	31	5	9
d	9	18	9	9

Для орграфов изменяется только матрица R.



R					
	0	1	0	0	1
	1	0	1	0	0
	0	1	0	1	1
	0	0	1	0	0
	1	0	1	0	0

	2	0	2	0	0
	0	2	0	1	2
	2	0	3	0	0
	0	1	0	1	1
	0	2	0	1	2

	0	4	0	2	4
	4	0	5	0	0
	0	5	0	3	5
	2	0	3	0	0
	4	0	5	0	0

	8	0	10	0	0
	0	9	0	5	9
	10	0	13	0	0
	0	5	0	3	5
	0	9	0	5	9

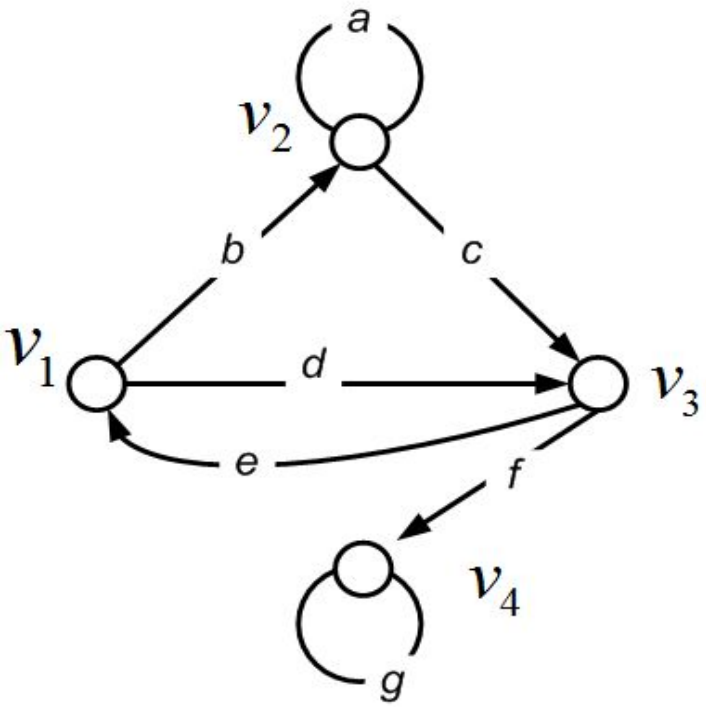
Достижимость

Если нас интересует только достижимость j — й вершины из i — й за l шагов, то есть наличие маршрута длины l, то определяем матрицу смежности над булевой алгеброй $\mathbf{B}\{0,1\}$.

При этом элемент $r_{ij}^{(l)}$ матрицы R^l будет равен 1, если существует хотя бы один маршрут длины l из вершины x_i в вершину x_j и 0, если не существует ни одного такого маршрута.

Ориентированные маршруты

Понятие маршрута можно обобщить на случай ориентированных графов. Ориентированная цепь (орцепь) часто называется путем , а ориентированный цикл – орциклом (*контуром*).



	0	1	1	0
	0	1	1	0
	1	0	0	1
	0	0	0	1

	1	1	1	1
	1	1	1	1
	0	1	1	1
	0	0	0	1

	1	1	1	1
	1	1	1	1
	1	1	1	1
	0	0	0	1

2.2. Компоненты связности

Компоненты и бикомпоненты

Определение. Вершины x, y графа G , между которыми существует простая цепь, называются *связными* (неотделенными).

Отношение *СВЯЗНОСТИ* (неотделенности) вершин
• рефлексивно (каждая вершина соединена с собой цепью нулевой длины);
• симметрично (цепь, записанная в обратном порядке – тоже цепь);
• транзитивно (если существует цепь из x в y, и цепь из y в z, то существует и цепь из x в z).

Таким образом, отношение связности (неотделенности) есть отношение эквивалентности на множестве вершин V; при этом V разбивается на классы попарно связных (неотделенных) вершин $V_1, V_2 \ldots V_{\text{К}}$.

Подграфы порожденные подмножествами V_i, не имеют общих вершин и ребер и называются *компонентами связности* (или просто *компонентами*).

Число их обозначается через $\kappa(G)$ (каппа).

Если $\kappa(G) = 1$, то граф называется *связным*. Другая крайность – *пустой* граф: для него $\kappa(G) = n$.

Отношение «быть в одной компоненте » для ребер – эквивалентность,
как и для вершин.

Для ориентированных графов

Вершина y *достижима* из вершины x, если существует путь из x в y (высказывание $D(x,y)$). Вершины x и y *взаимодостижимы* , если истинно высказывание $D(x,y) \wedge D(y,x)$.

Отношение взаимодостижимости рефлексивно, симметрично и транзитивно, т.е. является отношением эквивалентности.

Множество V вершин графа разбивается на классы взаимодостижимых вершин $V_1, V_2, \ldots, V_{\vec{k}}$.

Подграфы, порожденные этими подмножествами, называются *компонентами бисвязности*, или *бикомпонентами* графа G.

Число бикомпонент обозначается через $\breve{\kappa}(G)$.

Граф называется сильно связным, если он состоит из единственной
бикомпоненты.

Множество вершин, достижимых из вершины x, обозначим $Д(x)$

Теорема Вершины x и y взаимодостижимы тогда и только тогда, когда $Д(x) = Д(y)$.

$\Rightarrow$ Если x и y взаимодостижимы, то для любой вершины $z \in V$

ввиду транзитивности отношения взаимодостижимости, из $z \in Д(x)$ следует $z \in Д(y)$, а из $z \in Д(y)$ следует $z \in$ Поэтому $Д(x) = Д(y)$.

$\Leftarrow$ Пусть $Д(x) = Д(y)$. Так как $x \in Д(x)$ и $y \in Д(y)$, то из равенства $Д(x) = Д(y)$ следует, что $y \in Д(x)$ и $x \in Д(y)$, т.е. вершины x и y взаимодостижимы. □

Матрица смежности R над булевой алгеброй $\mathbf{B}\{0,1\}$ рефлексивна ($R = R \cup \Delta$, Δ – единичная матрица), т.к. любая вершина связна сама с собой и достижима сама из себя. Для неориентированного графа R симметрична.

Отношение неотделенности/достижимости – это транзитивное замыкание отношения смежности

$$\hat{R} = R \cup R^2 \cup \cdots .$$

Элемент матрицы $\hat{R}$ $\hat{r}_{ij} = 1,$ если существует цепь/путь из вершины x_i в вершину x_j. Для нахождения $\hat{R}$ применяется итеративная процедура.

Для первой итерации положим $R_1 := R \cup \Delta$. Элемент матрицы R_l

$$r_l[i,j] := r_{l-1}[i,j] \vee r_{l-1}[i,k] \wedge r_{l-1}[k,j] \quad (l - \text{индекс итерации})$$

На некоторой итерации $R_{l+1} = R_l$ и процесс останавливается; при этом $\hat{R} = R_l$. Алгоритм работает «на месте» — матрица R_{l+1} записывается на месте матрицы R.

$$R_0 = R;$$

$$R_1 = R_0 \cup R_0^2 = R \cup R^2;$$

$$R_2 = R_1 \cup R_1^2 = (R \cup R^2) \cup (R \cup R^2)^2 =$$

$$= R \cup R^2 \cup (R^2 \cup R^3 \cup R^3 \cup R^4) =$$

$$= R \cup R^2 \cup R^3 \cup R^4;$$

$$R_3 = R_2 \cup R_2^2 = (R \cup R^2 \cup R^3 \cup R^4) \cup (R \cup R^2 \cup R^3 \cup R^4)^2 =$$

$$= (R \cup R^2 \cup R^3 \cup R^4) \cup (R^2 \cup R^3 \cup R^4 \cup R^5 \cup R^3 \cup R^4 \cup R^5 \cup R^6 \cup$$

$$\cup R^4 \cup R^5 \cup R^6 \cup R^7 \cup R^5 \cup R^6 \cup R^7 \cup R^8) =$$

$$= R \cup R^2 \cup R^3 \cup R^4 \cup R^5 \cup R^6 \cup R^7 \cup R^8$$

$$R_4 = R_3 \cup R_3^2 = \cdots$$

Вершинам компоненты связности в матрице $\widehat{R}$ соответствует подмножество одинаковых строк. Отношение достижимости не является эквивалентностью – в общем случае оно не симметрично. Однако по теореме 2.2. вершинам бикомпоненты также соответствует подмножество одинаковых строк.

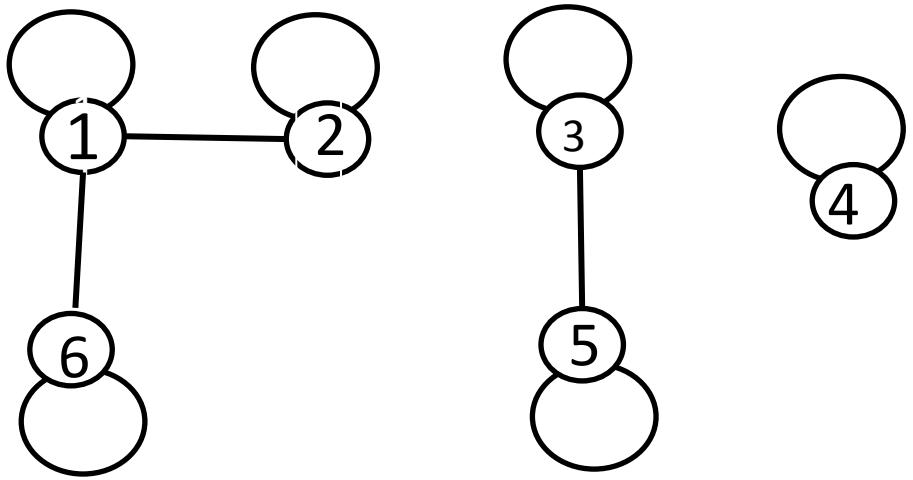
Пример (компоненты)

Фрагмент программы выглядит так:

```
for i=1 to n
 for j=1 to n
  for k=1 to n
    {r[i,j]:=r[i,j]∨ r[i,k] ∧
r[k,j];}
```

Пока матрица не перестанет изменяться

(**)



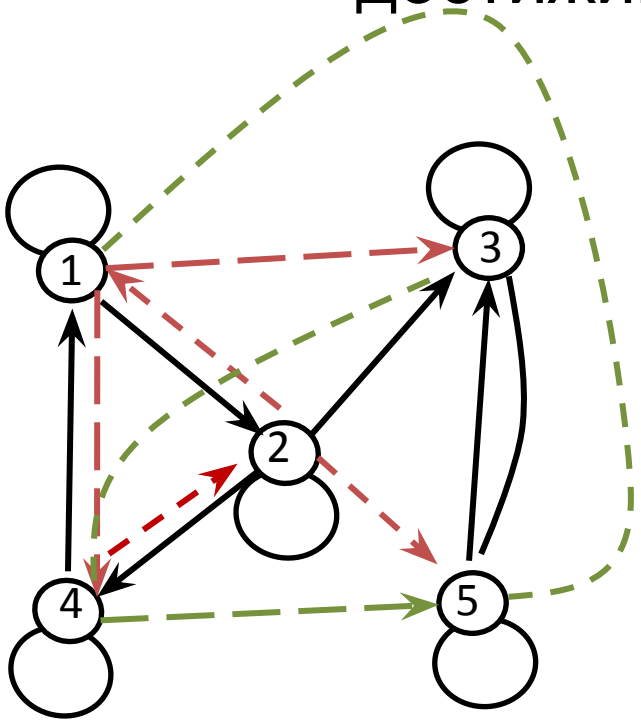
R	1	2	3	4	5	6
1	1	1	0	0	0	1
2	1	1	0	0	0	0
3	0	0	1	0	1	0
4	0	0	0	1	0	0
5	0	0	1	0	1	0
6	1	0	0	0	0	1

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

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

Компоненты связности образуют множества V_1={1,2,6}, V_2={3,5}, V_3={4}. Вершинам одной компоненты соответствуют одинаковые строки матрицы $R_{l+1} = \hat{R}$.

Пример (бикомпоненты, матрица достижимости)



R	1	2	3	4	5
1	1	1	0	0	0
2	0	1	1	1	0
3	0	0	1	0	1
4	1	0	0	1	0
5	0	0	1	0	1

$R2$	1	2	3	4	5
1	1	1	1	1	0
2	1	1	1	1	1
3	0	0	1	0	1
4	1	1	0	1	0
5	0	0	1	0	1

$R3$	1	2	3	4	5
1	1	1	1	1	1
2	1	1	1	1	1
3	0	0	1	0	1
4	1	1	1	1	1
5	0	0	1	0	1

$R4$	1	2	3	4	5
1	1	1	1	1	1
2	1	1	1	1	1
3	0	0	1	0	1
4	1	1	1	1	1
5	0	0	1	0	1

V_1={1,2,4},

V_2={3,5}

Сложность этого алгоритма $O(n^4)$. Сложность умножения матриц составляет $O(n^3)$ и число итераций имеет порядок n.

С. Уоршелл (1962 г.)предложил алгоритм нахождения транзитивного замыкания сложности $O(n^3)$. Экономия достигается за счет изменения порядка просмотра троек вершин в цикле (**).
На нулевой итерации R_0=R $(r_0[i,j] = r[i,j])$.

Фрагмент программы:

```
for k=1 to n
  for i=1 to n          (***)
   for j=1 to n
    {r[i,j]:=r[i,j] ∨ r[i,k]  ∧ r[k,j];}
```

На k-м шаге цикла по k ($k-$ индекс итерации)
$r_k[i,j] = r_{k-1}[i,j] \vee r_{k-1}[i,k] \wedge r_{k-1}[k,j]$
существует путь из вершины x_i в вершину x_j, проходящий через вершины с номерами, не превышающими k, только в следующих случаях.

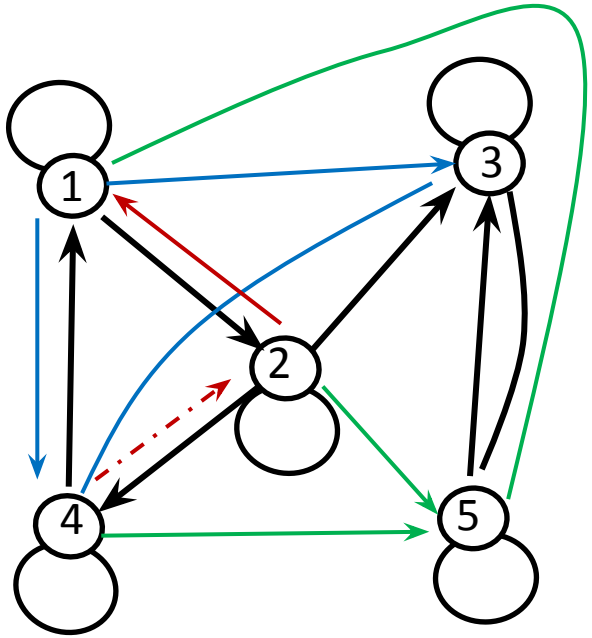
1. Уже существует путь из вершины x_i в x_j, который проходит через вершины с номерами не более k-1.
2. Существует путь из вершины x_i до вершины x_k, проходящий через вершины с номерами не более k-1, и путь от вершины x_k до вершины x_j, который также проходит через вершины с номерами не более k-1.

По завершению цикла по k определится существование путей между всеми парами вершин.

Ахо А., Хопкрофт Д., Ульман Д. (стр. 194-195)
Структуры данных и алгоритмы. : Пер. с англ. : Уч. пос. — М. : Издательский дом "Вильямс", 2000. — 384 с.

Алгоритм работает «на месте» -- матрица R_k записывается на месте матрицы R.

Пример (алгоритм Уоршалла)



R_1	1	2	3	4	5
1	1	1	0	0	0
2	0	1	1	1	0
3	0	0	1	0	1
4	1	1	0	1	0
5	0	0	1	0	1

R_2	1	2	3	4	5
1	1	1	1	1	0
2	0	1	1	1	0
3	0	0	1	0	1
4	1	1	1	1	0
5	0	0	1	0	1

R_3	1	2	3	4	5
1	1	1	1	1	1
2	0	1	1	1	1
3	0	0	1	0	1
4	1	1	1	1	1
5	0	0	1	0	1

R_4	1	2	3	4	5
1	1	1	1	1	1
2	1	1	1	1	1
3	0	0	1	0	1
4	1	1	1	1	1
5	0	0	1	0	1

$V_1=\{1,2,4\}, \quad V_2=\{3,5\}$

Warshall carried out research and development in operating systems, compiler design, language design, and operations research.
Known for Floyd–Warshall algorithm

https://en.wikipedia.org/wiki/Stephen_Warshall

Warshall Stephen (1935 – 2006)

2.3. Кратчайшие цепи

Волновой алгоритм

Пусть задан связный обыкновенный граф. Поставим задачу найти кратчайшую (с минимальным числом ребер) цепь, соединяющую вершины s и t.

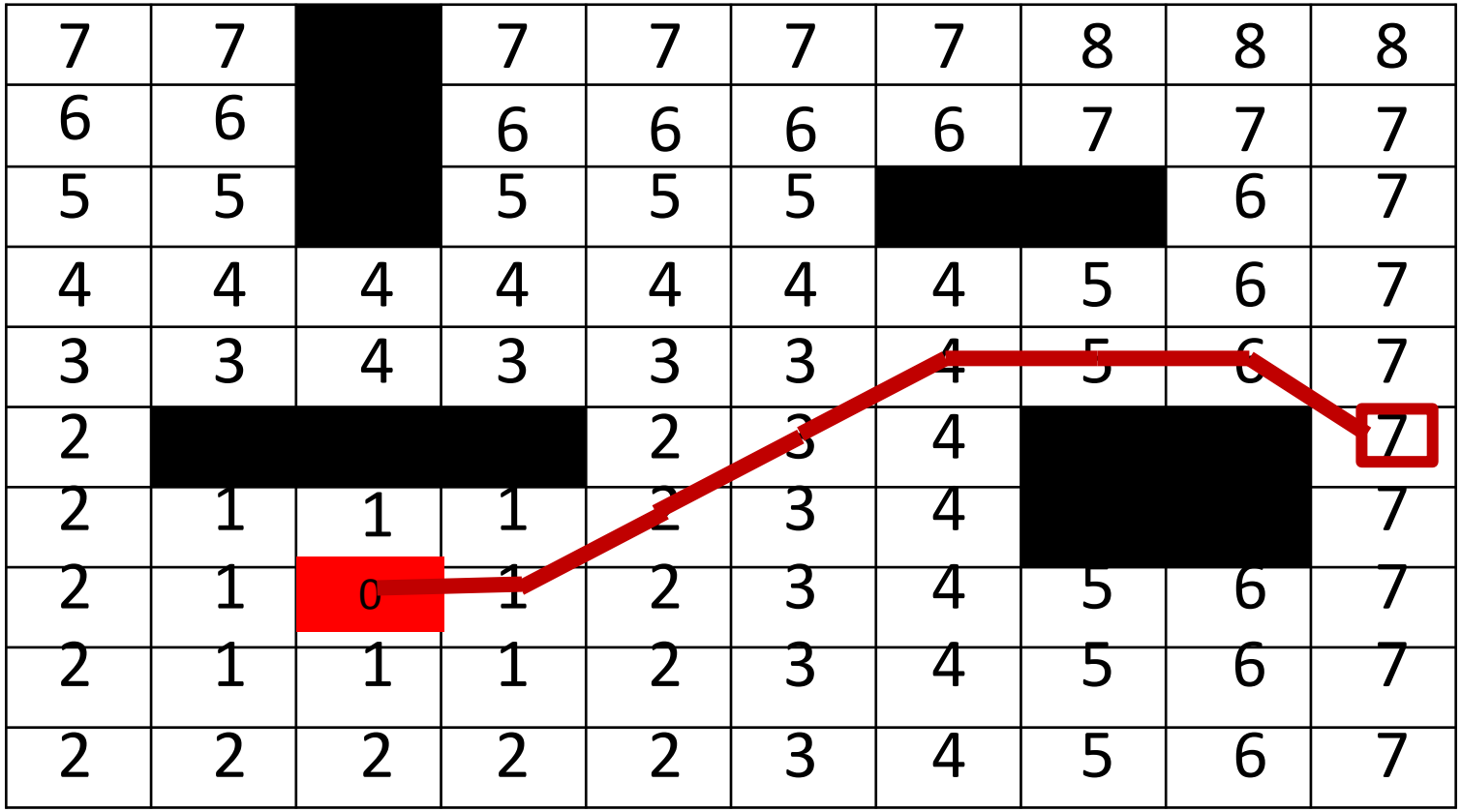
Описание алгоритма.

Шаг 1. Помечаем вершину s меткой 0.

Шаг 2. Меткой $k+1$ помечаем каждую вершину, которая еще не помечена и смежна хотя бы с одной вершиной, помеченной меткой k. Разметка прекращается, как только вершина t окажется помеченной некоторой меткой l.

Шаг 3. Сама кратчайшая цепь длины l отыскивается следующим образом. Начинаем с вершины t. За x_{l-1} берем любую вершину с меткой $l-1$ и смежную с t, а за u_l – любое ребро, соединяющее x_{l-1} с t. За x_{l-2} выбираем любую вершину, помеченную меткой $l-2$ и смежную с x_{l-1}, а за u_{l-1} – любое ребро, соединяющее x_{l-2} с x_{l-1} и так далее, пока не дойдем до вершины s.

Пример 2. Волновой алгоритм прокладки проводов на плате (алгоритм Ли)



Взвешенный граф

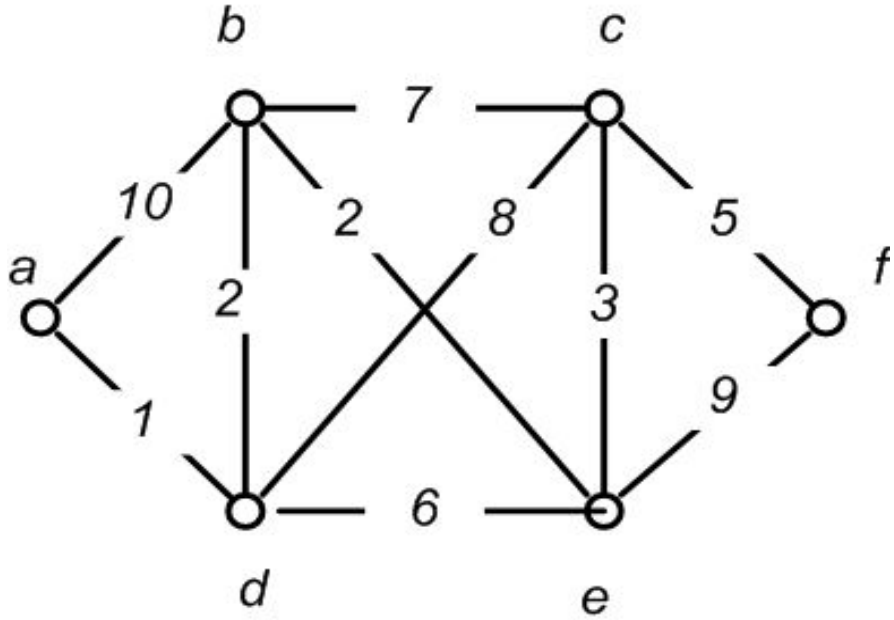
Пусть задан граф $G = (V, E)$ и отображение $f : E \to C$, ставящее в соответствие каждому ребру u, соединяющему пару вершин $\{x_i, x_j\}$, число c_{ij} – вес (длина) ребра. Таким образом задается *матрица весов* ребер $C = (c_{ij})$ размерности $n \times n$. Если ребро отсутствует, то $c_{ij} = \infty$.

Под длиной маршрута понимается сумма весов ребер, входящих в маршрут.

Требуется найти кратчайший по длине маршрут. Легко убедиться, что это простая цепь.

Пример

C	a	b	c	d	e	f
a	0	10		1		
b	10	0	7	2	2	
c		7	0	8	3	5
d	1	2	8	0	6	
e		2	3	6	0	9
f			5		9	0



Рассмотрим случай, когда имеется одна выделенная вершина (источник) s и требуется найти кратчайший маршрут до другой вершины.

$$s = a \rightarrow f \quad ?$$

Алгоритм Беллмана - Форда

Алгоритм заключается в последовательной разметке вершин графа. Метка при вершине x $L(x)=[l(x), p(x)]$, состоит из двух частей: числовой $l(x)$ и навигационной $p(x)$ – ссылкой на предшествующую вершину – метка предшестования.

Шаг 0. Инициализация.

Назначим вершине s *числовую* метку $l(s) = 0$. Для всех вершин x, отличных от s, назначим метки $l(x) = \infty$. Все навигационные метки при вершинах ссылаются на s: $p(x) = s$.

Шаг 1. Итерационный цикл разметки вершин.

Находим любое ребро (x, y), для которого $l(y) - l(x) > c(x, y)$ и обновляем числовую метку вершины y на $l'(y) = l(x) + c(x, y)$. Очевидно, метка $l'(y)$ может только уменьшиться: $l'(y) < l(y)$ при $y \neq s$. Обновленная навигационная метка при этом ссылается на предшествующую вершину x, вызвавшую данное обновление: $p'(y) = x$.

Цикл продолжается до тех пор пока метки не установятся, то есть не найдется больше ни одной вершины y, для которой можно уменьшить метку.

Шаг 2. Нахождение кратчайшей цепи.

Идем с конца - вершины $x_k = t$ к началу по меткам предшествования: $x_{k-1} = p(x_k), x_{k-2} = p(x_{k-1}), \ldots$ $x_0 = p(x_1) = s$.

Легко видеть, что при этом разность числовых меток соседних вершин в точности равна длине соединяющего их ребра: $l(x_k) - l(x_{k-1}) = c(x_{k-1}, x_k)$.

Теорема. Построенный маршрут кратчайший и его длина из s в t есть $l(t)$.

Доказательство. Пусть в размеченном с помощью алгоритма графе имеется **произвольный** маршрут $M\ [s = x_0, x_1 \ldots, x_{k-1}, x_k, \ldots, x_n = t]$. длины $c(s,t)$.

Тогда

$l(x_1) - 0 \leq c(x_0, x_1)$,

$l(x_2) - l(x_1) \leq c(x_1, x_2)$,

+

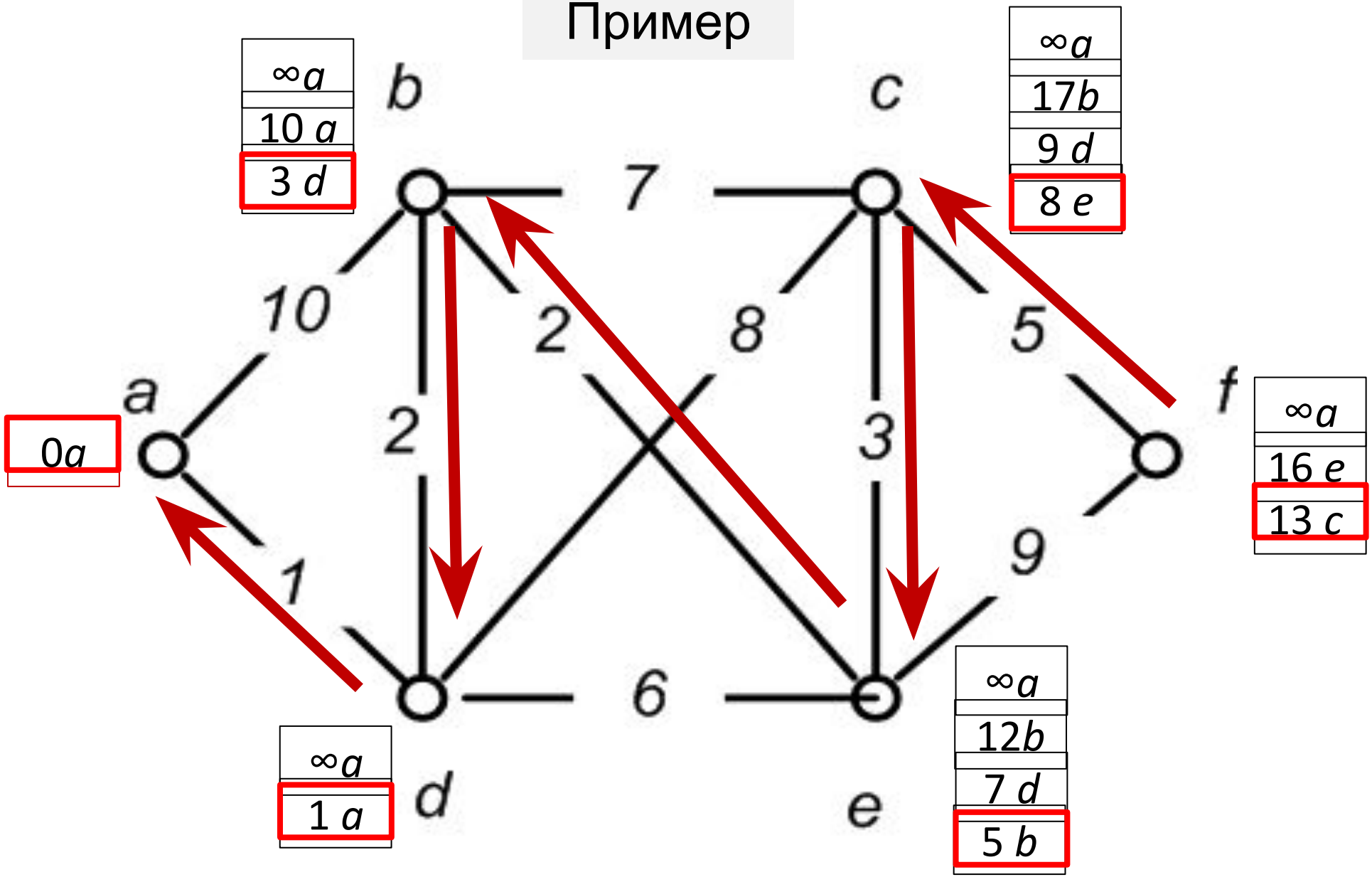
.....

$l(x_{n-1}) - l(x_{n-2}) \leq c(x_{n-2}, x_{n-1})$,

$l(t) - l(x_{n-1}) \leq c(x_n, x_{n-1})$

$$l(t) \leq c(s,t)$$

Пример



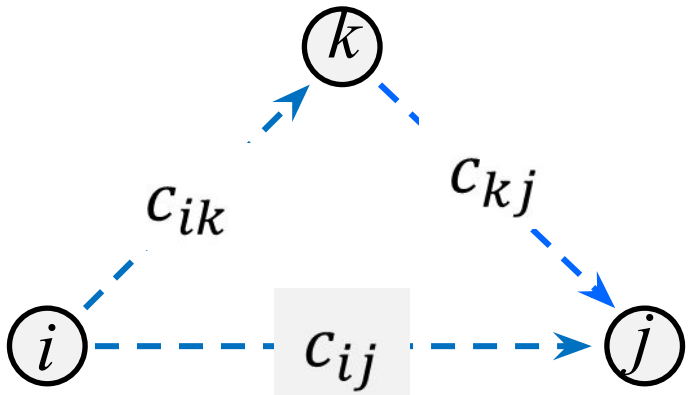
Замечание. Алгоритм находит кратчайшие цепи от одной исходной вершины (в данном случае a) до всех остальных.

Иллюстрация работы алгоритма Форда

Итерации	Ребро	*a*	*b*	*c*	*d*	*e*	*f*
0		*0a*	*∞a*	*∞a*	*∞a*	*∞a*	*∞a*
1	*ab*		*10a*				
2	*ad*				*1a*		
3	*bc*			*17b*			
4	*be*					*12b*	
5	*db*		*3d*				
6	*dc*			*9d*			
7	*de*					*7d*	
8	*ef*						*16e*
9	*eb*					*5b*	
10	*ec*			*8e*			
11	*cf*						*13c*

Если метка у вершины не меняется, клетка в следующей строке данного столбца не заполняется. Заключительные метки выделены красным цветом.

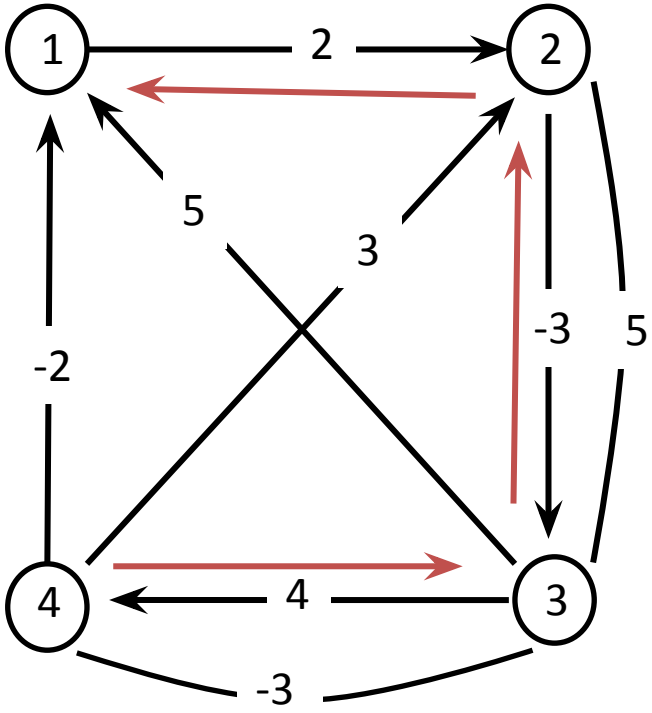
В таком представлении алгоритм Беллмана — Форда более пригоден
для «ручного» исполнения.
Пусть вершины пронумерованы $V=\{1,2,\ldots,n\}$ и начальная вершина
s =1. Метка вершины $L(x)=[l(x),p(x)]$, где $l(x)$ – метка расстояния вершины x от начальной, $p(x)$ – метка предшествования (навигационная). На шаге инициализации $i=s=1$, $l(1)=0$, $p(1)=1$; для остальных x $l(x)=\infty, p(x)=1$.



Перебираются тройки i,j,k:
$sum:=c_{ik}+c_{kj}$;
если $sum<c_{ij}$, то $\{c_{ij}:=sum;$
$\backslash^*$ $(c_{ij}:=\min[c_{ij}, c_{ik}+c_{kj}])$;
$p(j):=k\}$

Для нахождения кратчайших цепей/путей между всеми парами вер-шин нужно выполнить цикл по i.

Приме
р



0	2	∞	∞
0	2	-1	∞
0	2	-1	3

⊗

C	1	2	3	4
1	0	2	∞	∞
2	∞	0	-3	∞
3	5	5	0	4
4	-2	3	-3	0

=

0	2	-1	∞
0	2	-1	3

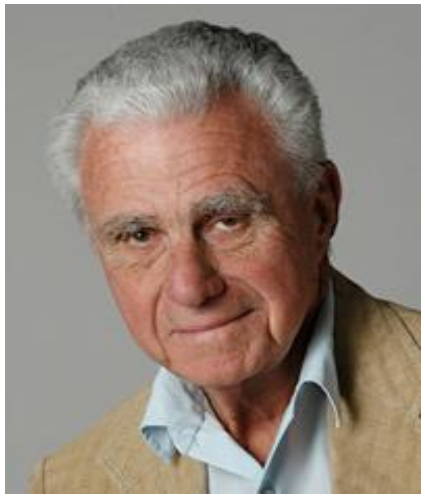
i:=1; {for q=1 to n {p[q]:=i}};

```
{for j=1 to n
 for k=1 to n
  { s:=c[i,k]+c[k,j];
    if s<c[i,j] then
    {c[i,j]:=s;
       p[j]:=k;}
  }
}
```

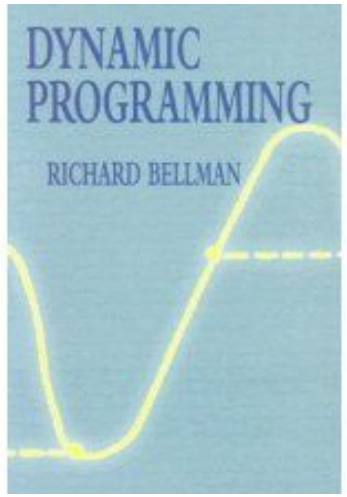
Пока метки не перестанут изменяться

1	2	3	4
0|1	2|1	∞| 1	∞|1
0|1	2|1	-1|2	∞ |1
0|1	2|1	-1|2	3|3
0|1	2|1	-1|2	3|3

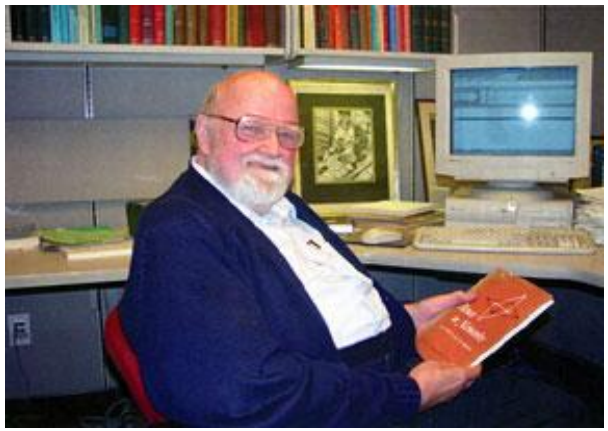
$$c[i,j]:=\min_{k}(c[i,j],c[i,k]+c[k,j])$$



БЕЛЛМАН, Ричард (1920- 1984) – «отец «динамического программирования» - американский математик. Опубликовал 619 статей и 39 книг. Один из наиболее цитируемых математиков в мире.
В расцвете творческой жизни после удаления опухоли на позвоночнике 11 лет был прикован к инвалидному креслу, но сохранил полную работоспособность
.



ФОРД, Лестер младший (1927 - 2017) – американский математик. Независимо от Беллмана в 1956 г. предложил алгоритм нахождения кратчайшего пути в графе, вместе с Фалкерсоном доказал теорему о наибольшем потоке в сети.



Алгоритм Дейкстры

Наиболее эффективный с точки зрения трудоемкости алгоритм разметки вершин предложил Дейкстра. Метки вершин делятся на две категории – временные L и постоянные L. Временная числовая метка дает верхнюю границу длины цепи от s до этой вершины. Эти метки постепенно уменьшаются с помощью итерационной процедуры, и на каждом шаге итерации одна из временных меток становится постоянной. Тогда метка уже не является верхней границей, а дает точную длину кратчайшей цепи от s к вершине с постоянной меткой.

Навигационные метки расставляются аналогично алгоритму Форда и обновляются вместе с числовыми..

Шаг 0. Инициализация.

Назначить вершине s метку $L(s) = [0, s]$ и считать ее постоянной. Вершины с постоянными метками считаются обработанными и к ним алгоритм не возвращается.

Для всех $x \neq s$ назначить метки $L(x) = [\infty, s]$ и считать эти метки временными.

Текущая вершина $p = s$.

Шаг 1. Обновление соседних временных меток из текущей вершины.

Взять текущую вершину p. Рассмотреть **все смежные с ней** вершины x с временными метками и, если $l(x) - l(p) > c(p, x)$, то обновить их: $L(x) = [l(p) + c(p, x), p]$.

Шаг 2. Превращение метки в постоянную.

а) Найти вершину x^*с наименьшей **временной** числовой меткой $l(x^*)$.

б) Превратить эту метку в постоянную.

в) Считать эту вершину текущей $p := x^*$.

г) Если есть еще временные метки, возврат на **шаг 1**.

д) Если все метки постоянные – переход на **шаг 3.**

Шаг 3. **Построение кратчайшей цепи.**

Точно так же, как в алгоритме Беллмана – Форда – от конца к началу по навигационным меткам $p(x)$.

При наличии всех постоянных меток доказательство оптимальности аналогично алгоритму Беллмана – Форда.

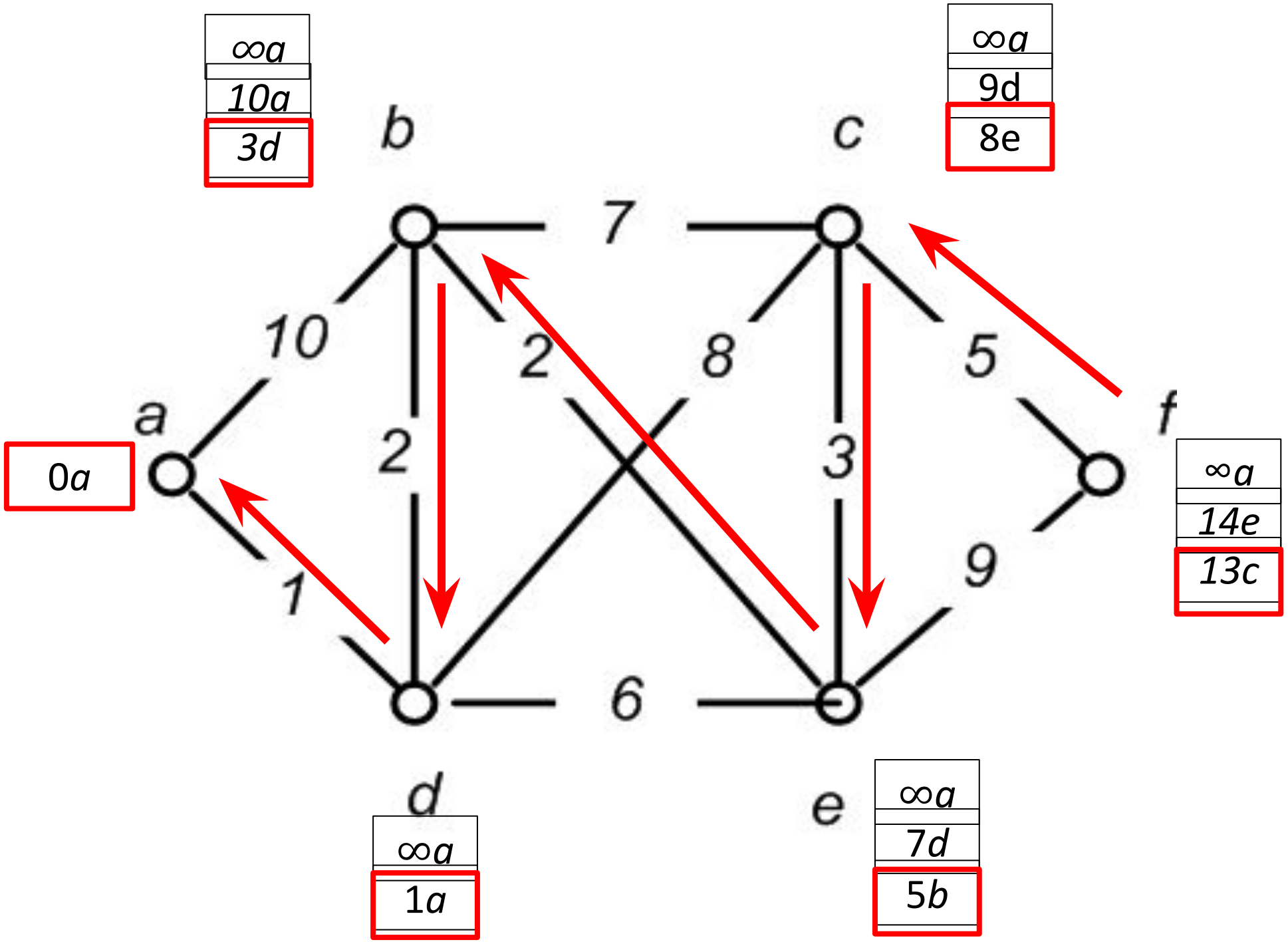

∞a
10a
3d
b
c
∞a
9d
8e
7
10
2
8
5
a
2
3
f
0a
1
9
6
d
∞a
1a
e
∞a
7d
5b
∞a
14e
13c

Иллюстрация работы алгоритма Дейкстры

Итера-ция		Ме-тим x	*a*	*b*	*c*	*d*	*e*	*f*
0			***0 a***	*∞a*	*∞a*	*∞a*	*∞a*	*∞a*
1	*a*	*b,d*		*10a*		***1a***		
2	*d*	*b,c,e*		***3d***	*9d*		*7d*	
3	*b*	*e*					***5b***	
4	*e*	*c,f*			***8e***			*14e*
5	*c*	*f*						***13c***

Теорема. Постоянная метка вершины x дает

длину кратчайшей цепи из s в x.

Доказательство. Предположим, что на некоторой итерации разметки вершин постоянные метки дают длины кратчайших (на первой итерации это очевидно).

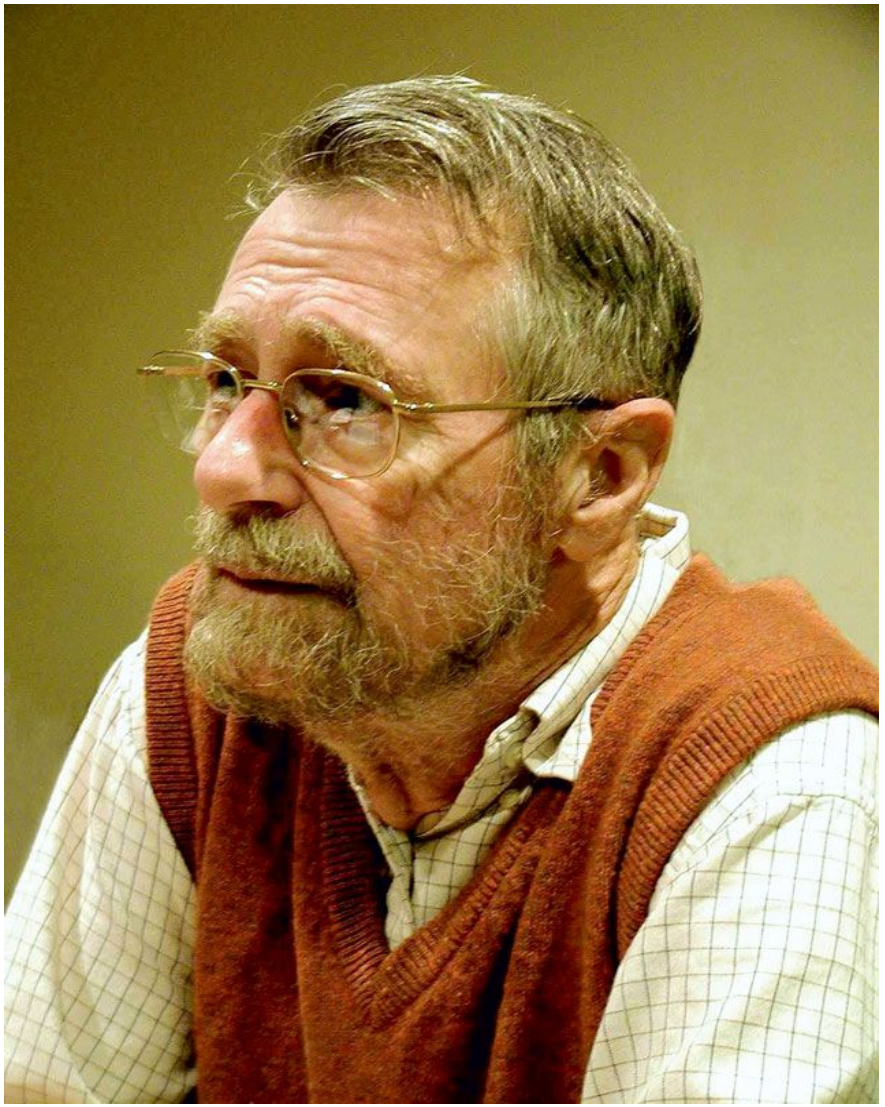
Пусть на следующей итерации вершина x^* получила постоянную метку. Предположим, что кратчайшая цепь из s в x^* проходит через вершину z с временной меткой, а вершина y (с постоянной меткой) предшествует z в этой цепи. По способу разметки $c(y, x^*) \geq c(y,z)+c(z, x^*)$ и $l(z) \geq l(x^*)$. Тогда длина цепи между z и x^* $c(z, x^*)$ должна быть отрицательной, что противоречит условию

Обобщения

1. Алгоритмы Беллмана – Форда и Дейкстры легко распространяются на случай ориентированных (частично ориентированных) графов, при этом каждое неориентированное ребро представляется двумя дугами, ориентированными в разные стороны. Более того, веса (длины) этих дуг могут быть различными.

2. В алгоритме Беллмана – Форда допускаются отрицательные веса ребер, такие постановки задачи иногда полезны в приложениях. Этим он принципиально отличается от алгоритма Дейкстры, где отрицательные веса невозможны. Но «всеядность» алгоритма Беллмана – Форда оплачивается его более высокой трудоемкостью.

ДЕЙКСТРА, Эдсгер (Edsger Dijkstra, 1930-2002).
Нидерландский учёный, труды которого оказали большое влияние на развитие информатики; один из авторов языка Алгол и концепции структурного программирования. По образованию физик-теоретик. Во второй половине 1950-х годов в поисках путей оптимизации разводки печатных плат разработал алгоритм поиска кратчайшего пути на графе.



Алгоритм Беллмана--Форда 2
(все пары вершин)

. Этот алгоритм является модификацией **Алгоритма 1** (стр. 32) нахождения транзитивного замыквния и обобщением алгоритма Беллмана–Форда на случай нахождения кратчайших цепей/путей из каждой вершины в каждую.
Вместо обычного произведения матриц $C = AB$

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

А. Шимбел предложил такую операцию умножения матриц:

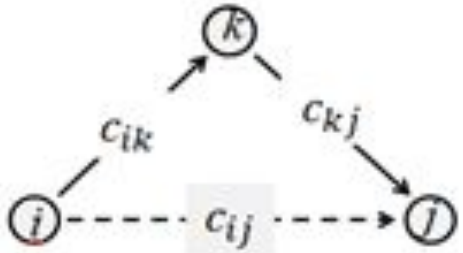
$$c_{ij} = \min_{k=1}^{n} (a_{ik} + b_{kj})$$

суммирование заменяется на min,
умножение на арифметическое сложение

Элемент, например, c_{24} матрицы C^2 будет

C	1	2	3	4
1	0	−2	3	−3
2	∞	0	2	∞
3	∞	∞	0	−3
4	4	5	5	0

$$c_{24} = \min_{k=1}^{n}(c_{21} + c_{14}, c_{22} + c_{24}, c_{23} + c_{34}, c_{24} + c_{44}) =$$
$$= \min(\infty, \infty, -1, \infty) = -1.$$



Это означает, что при при просмотре троек вершин с $i = 2, j = 4$ нашлась такая вершина $k = 3$, что путь через нее $c_{23} + c_{34} = -1$ на расстоянии 2 (по числу ребер) короче, чем путь на расстоянии 1 (по ребрам) . $c_{24} = \infty$

Описание алгоритма.

Инициализация. Для нулевой итерации положим $C_0 = C$.

Матрица предшествований $P_0 \Rightarrow$

```
for i=1 to n
for j=1 to n
p[i,j]:=i
```

Итерации. Алгоритм работает «на месте»: старые значения элементов матриц C и P заменяются на новые. На каждой итерации q матрица C_q получается как квадрат (по умножению Шимбела). матрицы, полученной на предыдущей итерации C_{q-1}.

Таким образом будет получена последовательность степеней матрицы весов от $C^1 = C^{2^0} = C_0 = C$ до C^{2^q}

Поэлементно на q-й итерации

$$c_{ij}^{(q)} := \min\left(c_{ij}^{(q-1)}, c_{ik}^{(q-1)} + c_{kj}^{(q-1)}\right)$$

элемент c_{ij} может разве что уменьшиться ($c_{ij}^{(q)} \leqslant c_{ij}^{(q-1)}$).

Если при этом $c_{ik}^{(q-1)} + c_{kj}^{(q-1)} < c_{ij}^{(q-1)}$, метку предшествования нужно заменить: $p_{ij} := p_{kj}$.

Фрагмент псевдокода итерации

```
1. for i=1 to n
2. for j=1 to n
3. for k=1 to n
4.  {s:=c[i,k]+c[k,j];
5.   if s<c[i,j] then
6.   {c[i,j]:=s;
7.    p[i,j]:=p[k,j];
9.   }
10  }
```

Это фрагмент текущей q-й итерации. В строках 4,5 вычисляется минимальное значение c_{ij}, в строках 6, 7 назначаются новые значения c_{ij} и p_{ij}.

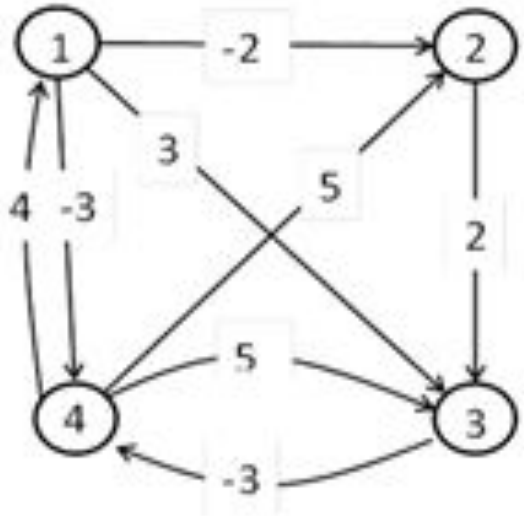
Правило остановки. На некоторой итерации $C_q = C_{q-1}$. Это означает, что на предыдущей итерации все кратчайшие цепи/пути найдены и следующие итерации будут только повторять уже полученные результаты. Поэтому на каждой итерации нужно отслеживать произошли ли изменения значений c_{ij} и p_{ij}. Если нет, то **выход**, иначе итерации продолжаются.

Нахождение кратчайших цепей/путей. Матрица предшествований P предложена Дж. Оттерманом для отслеживания кратчайщих цепей/путей по меткам предшествования.

При инициализации предполагается, что при поиске путей из i-й вершины она предшествует всем остальным вершинам -- i-я строка матрицы P заполняется метками i.

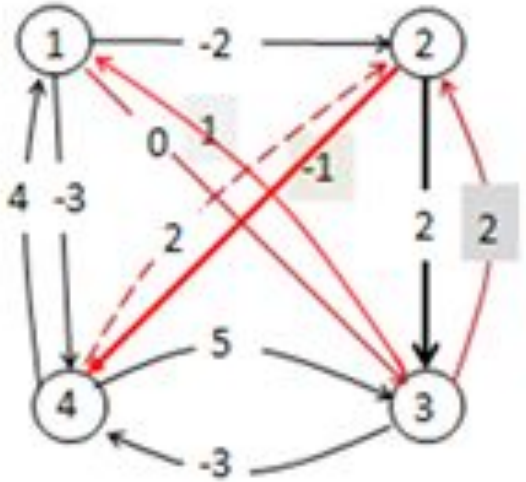
После окончания итераций и изменения меток кратчайший путь из вершины i вершину j отыскивается с конца в обратном порядке (начиная с вершины j, как в алгориме Беллмана--Форда 1)

Пример. Алгоритм Беллмана--Форда (все пары вершин)



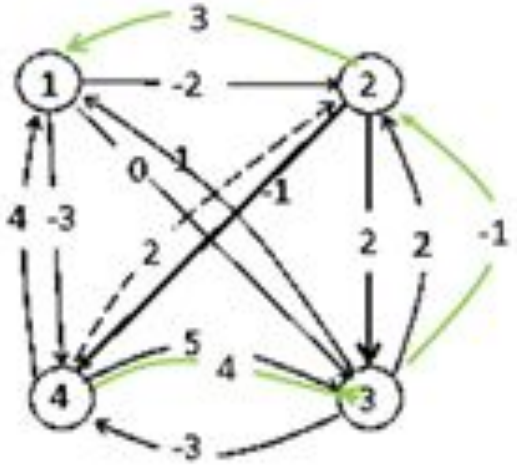
C_0	1	2	3	4
1	0	-2	3	-3
2	∞	0	2	∞
3	∞	∞	0	-3
4	4	5	5	0

P_0	1	2	3	4
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4



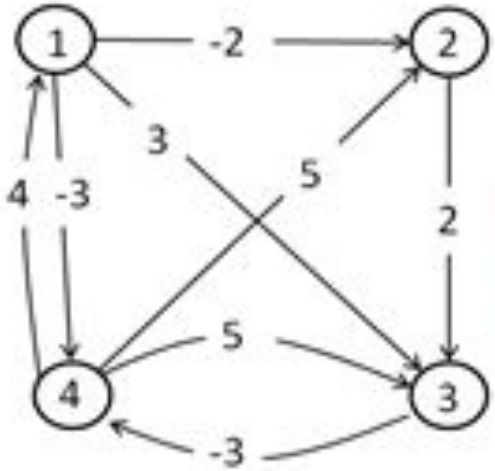
C_1	1	2	3	4
1	0	-2	0	-3
2	∞	0	2	-1
3	1	2	0	-3
4	4	2	5	0

P_1	1	2	3	4
1	1	1	2	1
2	2	2	2	3
3	4	4	3	3
4	4	1	4	4



C_2	1	2	3	4
1	0	-2	0	-3
2	3	0	2	-1
3	1	-1	0	-3
4	4	2	4	0

P_2	1	2	3	4
1	1	1	2	1
2	4	2	2	3
3	4	1	3	3
4	4	1	2	4



$C_3 = C_2$ и здесь не приводится.

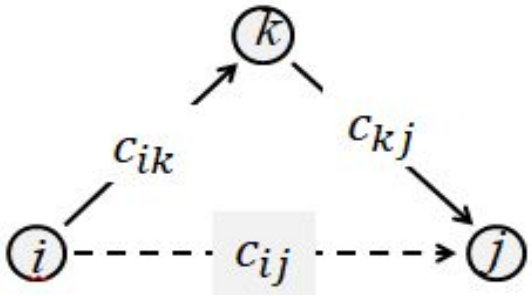
Найдем, например, кратчайший путь из вершины 3 в вершину 2 по меткам предшествования на исходном графе (в обратном порядке).

Выберем строку 3 в матрице P_2. Вершине 2 предшествует 1, 1-й – вершина 4, 4-й – вершина 3: 2-1-4-3. Сам путь: 3-4-1-2 длины $c_{32} = -1$

Элемент, например, c_{24} матрицы C^2 будет

$$c_{24} = \min_{k=1}^{n}(c_{21} + c_{14}, c_{22} + c_{24}, c_{23} + c_{34}, c_{24} + c_{44}) =$$

$$= \min(\infty, \infty, 1, \infty) = 1.$$



Это означает, что при при просмотре троек вершин с $i = 2, j = 4$ нашлась такая вершина $k = 3$, что путь через нее $c_{23} + c_{34} = 1$ на расстоянии 2 (по числу ребер) короче, чем путь на расстоянии 1 (по ребрам) . $c_{24} = \infty$

Алгоритм Флойда

Предыдущие алгоритмы (Беллмана – Форда и Дейкстры) давали возможность находить кратчайшие маршруты между одной начальной вершиной (источником) s и /или всеми другими вершинами графа.

Алгоритм Флойда (Флойда – Уоршалла), разработанный в 1962 г., позволяет найти длины кратчайших маршрутов между **всеми парами** вершинами.

Замечание. В алгоритме Флойда, так же как и в алгоритме Беллмана – Форда, нет ограничений на неотрицательность длин ребер, как будет показано в примере.

Пусть задан взвешенный *унграф* (ориентированный, неориентированный, смешанный) $G=(V,E)$ с матрицей весов (длин) $C=(c_{ij})_{n\times n}$, где $c_{ii}=0$; $c_{ij}=\infty$, если нет ребра (x_i, x_j);
навигационная матрица (матрица предшествований) $P=(p_{ij})_{n\times n}$;
C_k и P_k -- матрицы, полученные на k – й итерации.

Инициализация

$C_0:=C;$

P_0 ⇨

```
for i=1 to n
for j=1 to n
p[i,j]:=i;
```

Основной цикл

Алгоритм работает «на месте»: старые значения элементов матриц C и P заменяются на новые; по завершению цикла по k ($k=n$) C становится матрицей длин кратчайших цепей/путей, а P -- матрицей
самих цепей/путей.

На каждой итерации по k вычисляется C_k, элементы которой

$$c_{ij}^{(k)} := \min\left(c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)} \right)$$

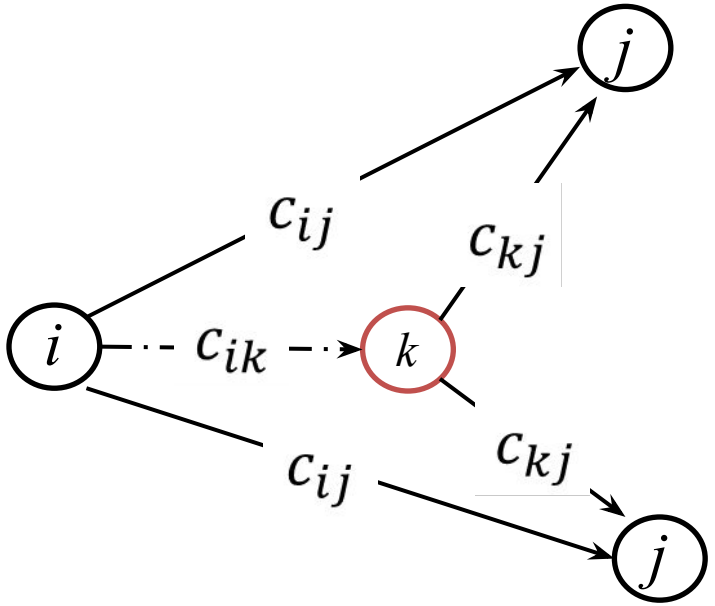
(Флойд1)

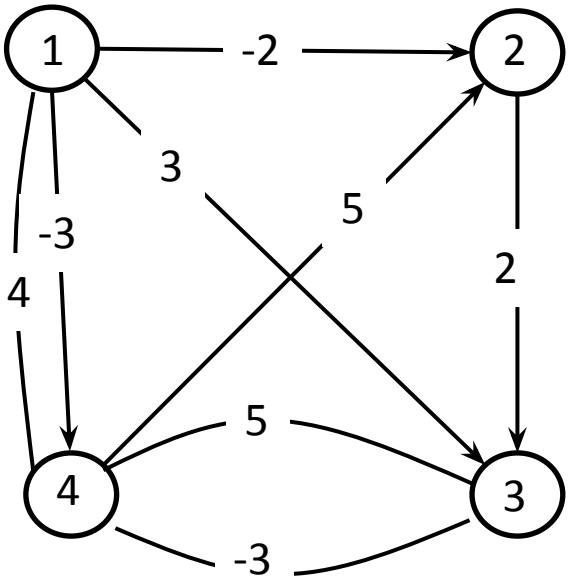
На k-й итерации вершина k включается в путь только тогда, когда новый путь короче старого. На n-й итерации все пути c_{ij} — кратчайшие.

Фрагмент программы:

```
for k=1 to n
for i=1 to n
for j=1 to n
  { s:=c[i,k]+c[k,j];
  if s<c[i,j] then
  {c[i,j]:=s; p[i,j]:=p[k,j];}
 }
```

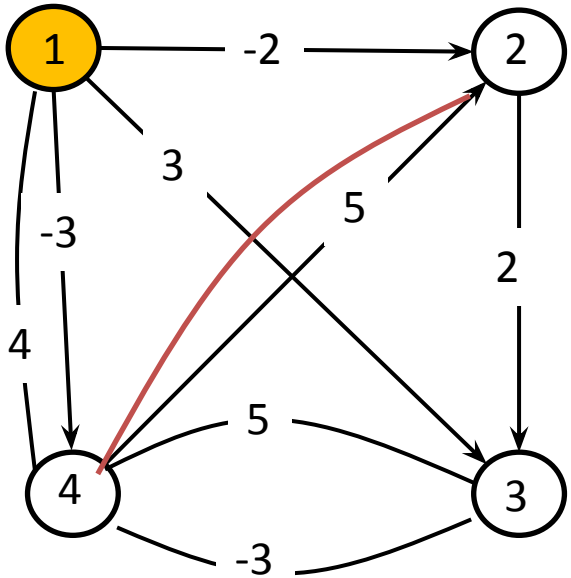
c[i,j]:=min(c[i,j],c[i,k]+c[k,j])
k





C_0	1	2	3	4
1	0	-2	3	-3
2		0	2	
3			0	-3
4	4	5	5	0

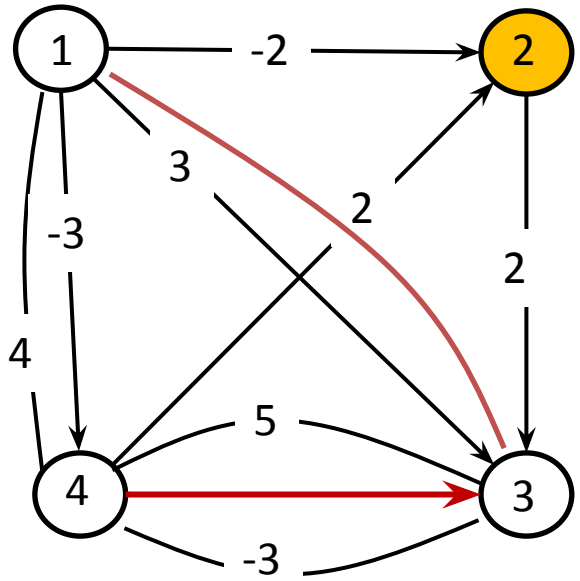
P_0	1	2	3	4
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4



	1	2	3	4
1	0	-2	3	-3
2		0	2	
3			0	-3
4	4	2	5	0

P_1	1	2	3	4
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	1	4	4

$p_{42} := p_{12} (= 1)$

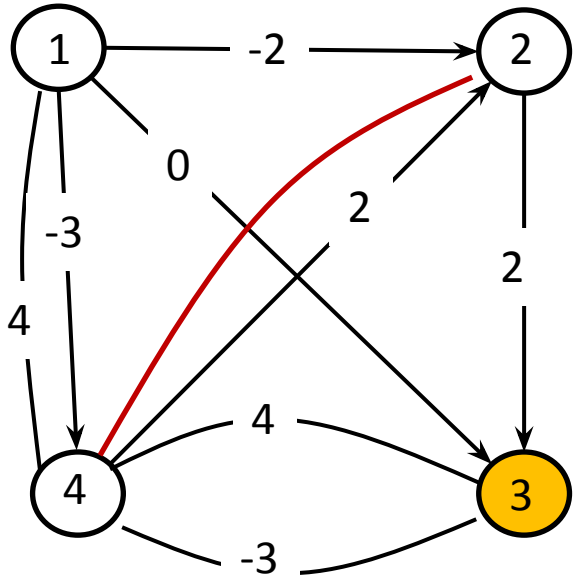


	1	2	3	4
1	0	-2	0	-3
2		0	2	
3			0	-3
4	4	2	4	0

P_2	1	2	3	4
1	1	1	2	1
2	2	2	2	2
3	3	3	3	3
4	4	1	2	4

$p_{13} := p_{23}(2)$

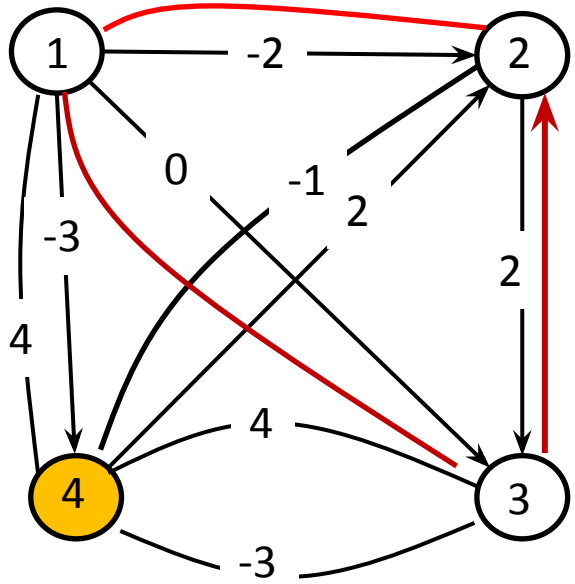
$p_{43} := p_{23}(2)$



	1	2	3	4
1	0	-2	0	-3
2		0	2	
3			0	-3
4	4	2	4	0

P_3	1	2	3	4
1	1	1	2	1
2	2	2	2	3
3	3	3	3	3
4	4	1	2	4

$p_{24} := p_{34}(3)$



	1	2	3	4
1	0	-2	0	-3
2		0	2	
3		-1	0	-3
4	4	2	4	0

P_4	1	2	3	4
1	1	1	2	1
2	4	2	2	3
3	4	1	3	3
4	4	1	2	4

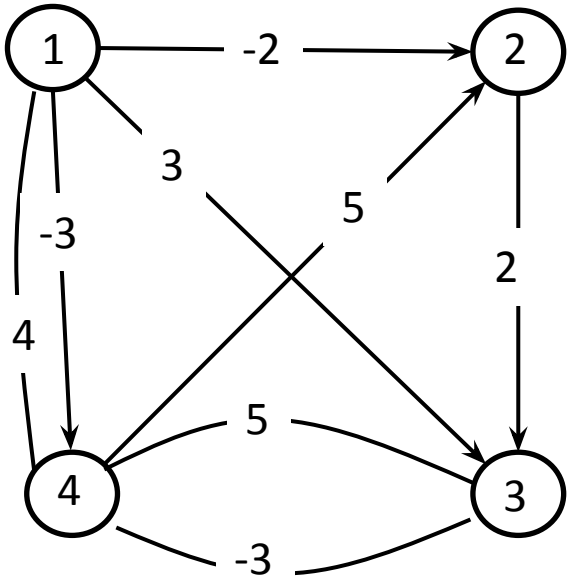
$p_{21} := p_{41}(4)$

$p_{31} := p_{41}(4)$

$p_{32} := p_{42}(1)$

Найдем, например, кратчайший путь из 2 в 1 по меткам предшествования на исходном графе (в обратном порядке). Выберем строку 2 в матрице P_4. Вершине 1 предшествует 4, 4-й – вершина 3, 3-й – вершина 2: 1-4-3-2.

Сам путь: 2-3-4-1 длины c_{21}=3.



P_4	1	2	3	4
2	4	2	2	3

Замечание. Алгоритмы Беллмана – Форда и Флойда работают корректно, если в графе нет циклов отрицательной длины (сумма длин которых отрицательна). В противном случае длина кратчайшей цепи/пути уменьшается до -∞ (программа зацикливает). Это обнаруживается,
когда некоторые диагональные элементы становятся отрицательными.

Запишем алгоритм [Флойд1] в общем виде

$$c_{ij}^{(k)} := \oplus\left(c_{ij}^{(k-1)}, c_{ik}^{(k-1)} \otimes c_{kj}^{(k-1)} \right)$$

Для кратчайшего пути $\oplus$ —это min, $\otimes$ — сложение (+).

Этот алгоритм применим и для нахождения цепи/пути наибольшей длины (критического пути).
Операция $\oplus$ заменяется на max, $\otimes$ на +, и в матрице весов $c_{ij}=-\infty$, если нет ребра (x_i, x_j).

Другой пример: найти цепь/путь максимальной надежности в сети связи. Надежность ребра ρ_{ij} — это вероятностная характеристика $0 \leq \rho_{ij} \leq 1$; надежность цепи определяется как произведение (обычное) надежностей ребер цепи.
Тогда $\oplus$ — операция max, а $\otimes$ — умножение.

ФЛОЙД, Роберт В (*Robert W Floyd*, 1936 – 2001) — американский учёный в области теории вычислительных систем. Лауреат премии Тьюринга. Роберт окончил школу в возрасте 14 лет, перепрыгнув три класса. Флойд не имел титула PhD.

В Стэнфорде Флойд тесно работал с Дональдом Кнутом, в том числе в качестве главного редактора серии его знаменитых книг «Искусство программирования».

Сравнение алгоритмов

www.youtube.com/watch?v=cSxnOm5aceA