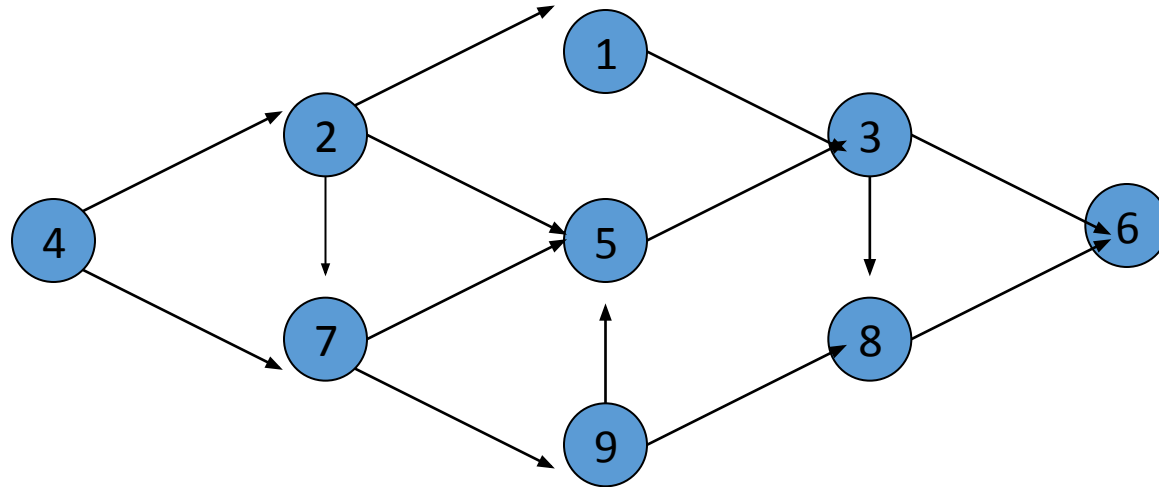


Графы

Топологическая сортировка

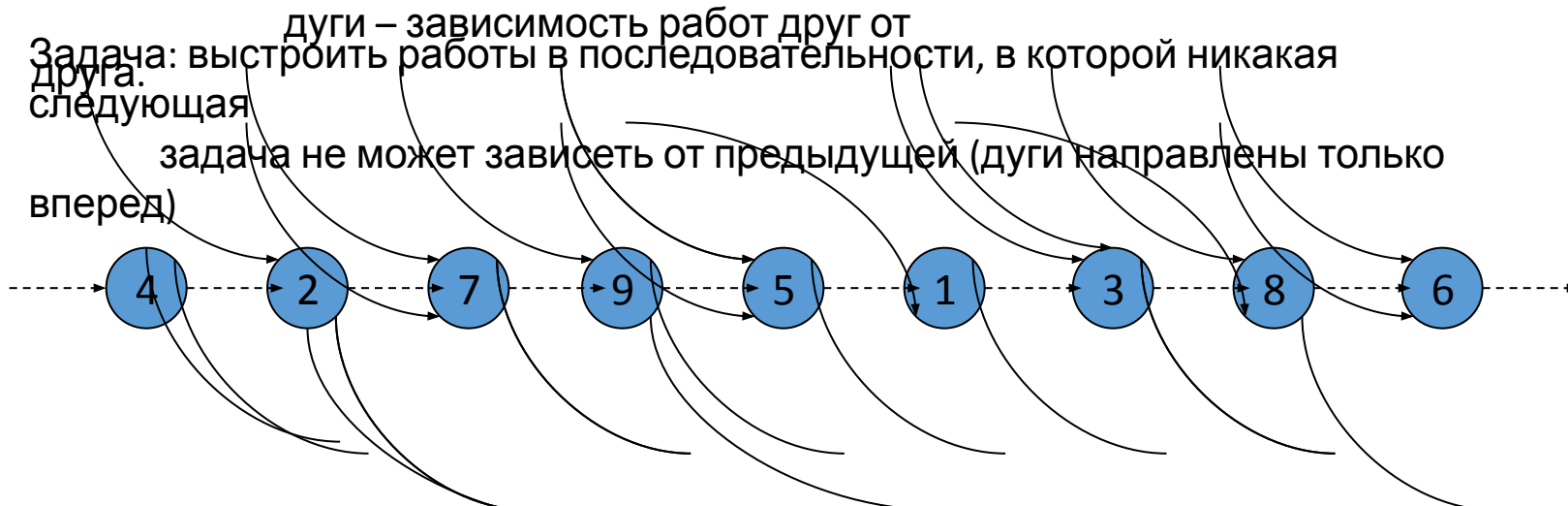
Топологическая сортировка вершин ориентированного графа без ЦИКЛОВ.

DAG – Directed Acyclic Graph – ориентированный граф без ЦИКЛОВ

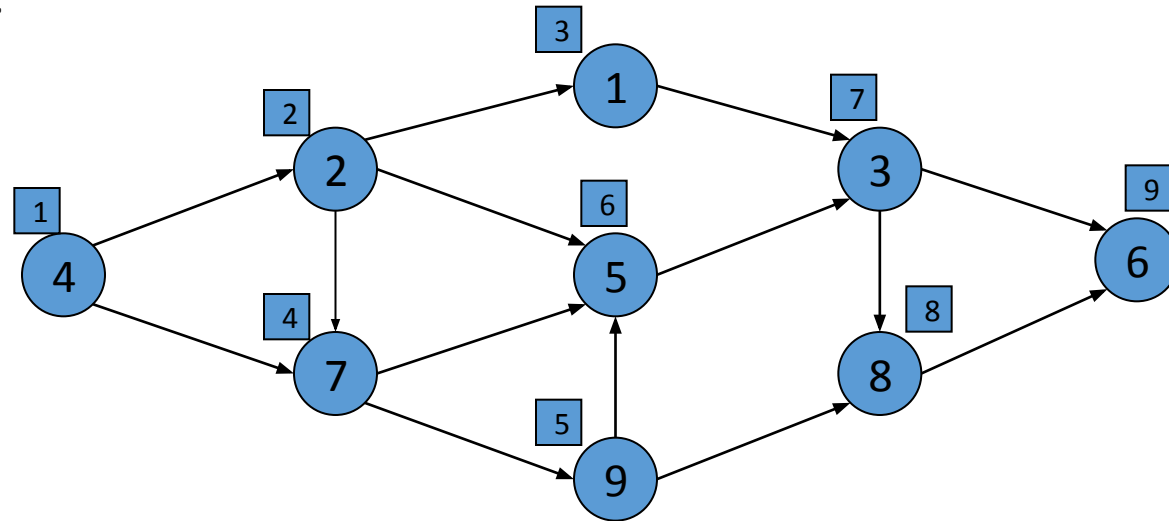


Интерпретация: вершины – элементарные работы;

Задача: выстроить работы в последовательности, в которой никакая следующая задача не может зависеть от предыдущей (дуги направлены только вперед).



Топологическая сортировка вершин ориентированного графа без циклов.



«Наивный» алгоритм нумерации

вершин:

1. Находим какую-либо вершину, в которую не входят дуги, нумеруем ее.
2. Помечаем дуги, выходящие из помеченной вершины, как «не существующие».
3. Повторяем шаги (1) и (2), пока не будут занумерованы все вершины.

Оценка времени работы

алгоритма

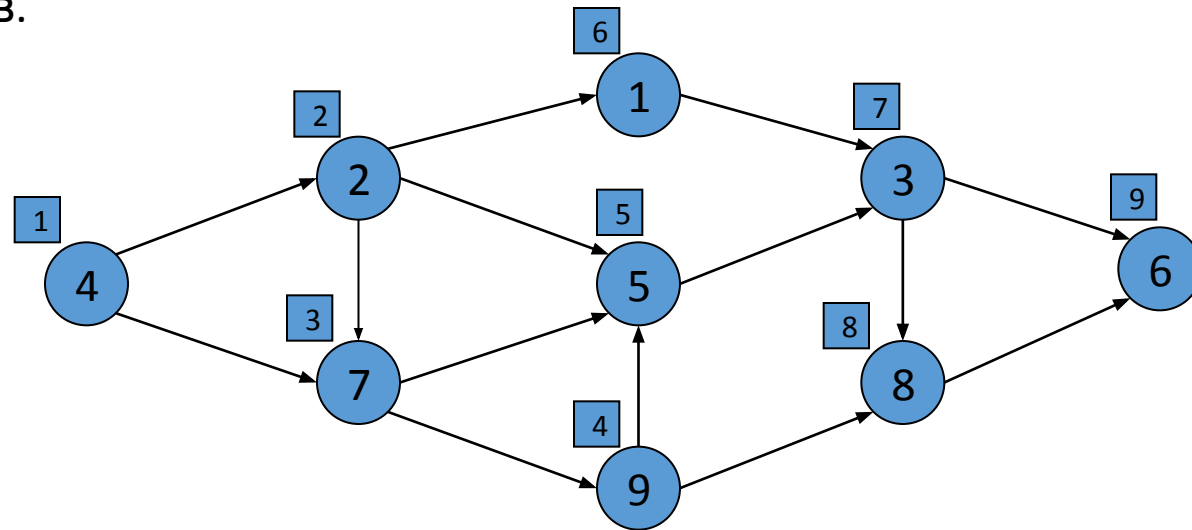
- | | |
|--|-----------------|
| 1. Построение очереди с приоритетами из вершин графа | $n \log n$ |
| 2. Выборка вершин из очереди | $n \log n$ |
| 3. Коррекция очереди при пометке дуг | $m \log n$ |
| Итого: | $(m+2n) \log n$ |

Проверка ацикличности

графа

Граф содержит цикл, если на шаге (1) не удастся найти вершину, в которую не входят дуги.

Топологическая сортировка вершин ориентированного графа без циклов.



«Эффективный» алгоритм нумерации

вершин:

1. Производим обход графа с помощью рекурсивной процедуры обхода, начиная с произвольной вершины.
2. Нумеруем каждую вершину при «прохождении ее назад» максимальным из номеров (то есть нумерация происходит в порядке убывания номеров).
3. Повторяем шаги (1) и (2), пока не останется непройденных вершин.

Оценка времени работы алгоритма = время обхода =

$(m+n)$.

Проверка ацикличности

графа.

Граф содержит цикл, если при проходе по «обратной дуге» попадаем в еще непомеченную («синюю») вершину.

Алгоритм Флойда - Уоршелла

Алгоритм Флойда - Уоршелла



Роберт Флойд

Разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом

В отличие от алгоритма Дейкстры, который позволяет построить ориентированное **дерево кратчайших путей от некоторой вершины**, метод Флойда позволяет найти длины **всех кратчайших путей в графе**.

Конечно эта задача может быть решена и многократным применением алгоритма Дейкстры (каждый раз последовательно выбираем вершину от первой до N-ной, пока не получим кратчайшие пути от всех вершин графа), однако реализация подобной процедуры требует значительных

Обозначения

Перенумеруем вершины графа целыми числами от 1 до N .

Обозначим через $d_{i,j}^m$ длину кратчайшего пути из вершины i в вершину j , который в качестве промежуточных может содержать только первые m вершин графа (промежуточной вершиной пути является любая принадлежащая ему вершина, не совпадающая с его начальной или конечной вершинами).

Если между вершинами i и j не существует ни одного пути указанного типа, то условно будем считать, что $d_{i,j}^m = \infty$.

Величина $d_{i,j}^0$, представляет длину кратчайшего пути из вершины i в вершину j , не имеющего промежуточных вершин, т. е. длину кратчайшей дуги, соединяющей i с j (если такие дуги присутствуют в графе).

Обозначения

Обозначим через D^m матрицу размера $N \times N$, элемент (i,j) которой совпадает с $d_{i,j}^m$.

Если в исходном графе нам известна длина каждой дуги, то мы можем сформировать матрицу D^0 , которая в алгоритме Флойда выступает в качестве исходной.

Вначале из этой матрицы вычисляется матрица D^1 . Затем по матрице D^1 вычисляется матрица D^2 и т. д. по формуле:

$$d_{i,j}^m = \min\{d_{i,m}^{m-1} + d_{m,j}^{m-1}; d_{i,j}^{m-1}\}$$

Процесс повторяется до тех пор, пока по матрице D^{N-1} не будет вычислена матрица D^N .

Суть алгоритма Флойда

заключается в проверке того, не окажется ли путь из вершины i в вершину j короче, если будет проходить через некоторую промежуточную

вершину m .

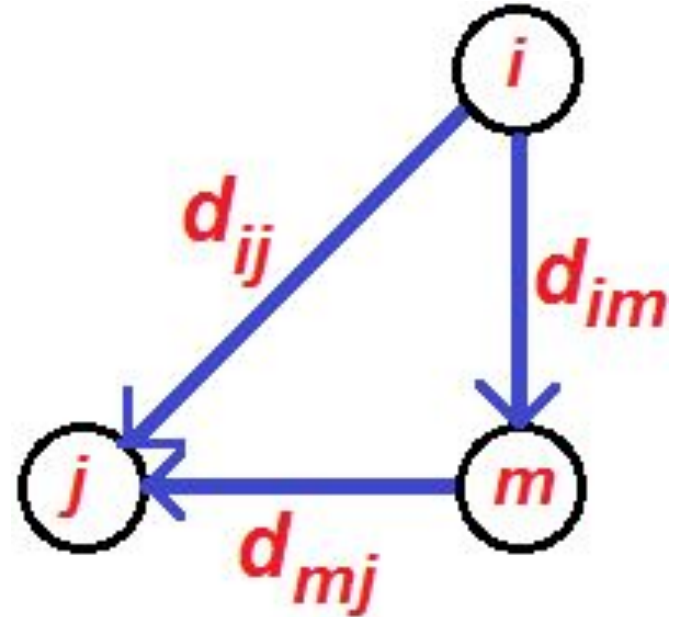
Пусть есть три вершины – i , j, m – и расстояния между

ними: d_{ij} , d_{im} , d_{mj} .

Если выполняется

неравенство $d_{im} + d_{mj} < d_{ij}$, то целесообразно заменить

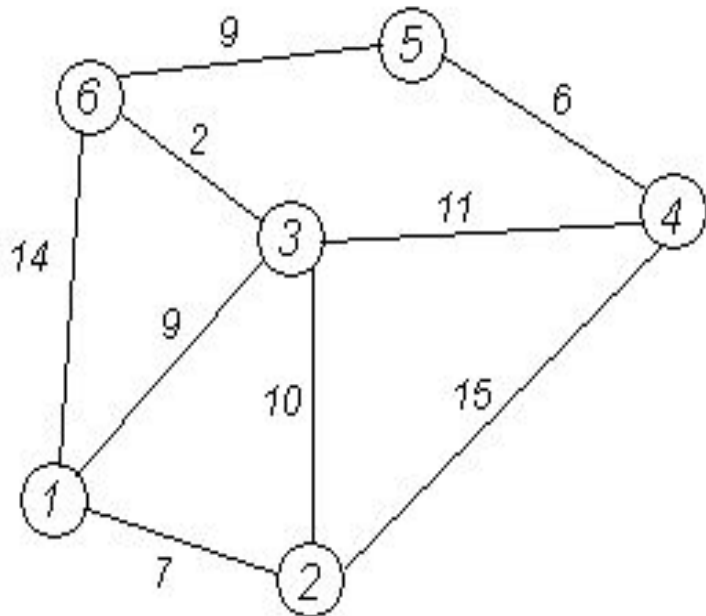
путь ij путём $im + mj$



«Треугольный оператор»

Алгоритм Флойда на примере

Этап 1. Перенумеровать вершины графа от 1 до N, определить матрицу D^0 , каждый элемент $d_{i,j}^0$ которой есть длина кратчайшей дуги между вершинами i и j . Если такой дуги нет, положить значение элемента $d_{i,j}^0 = \infty$. Значения диагональных элементов $d_{i,i} = 0$.



D^0	1	2	3	4	5	6
1	0	7	9	∞	∞	14
2	7	0	10	15	∞	∞
3	9	10	0	11	∞	2
4	∞	15	11	0	6	∞
5	∞	∞	∞	6	0	9
6	14	∞	2	∞	9	0

Алгоритм Флойда на примере

Матрицу S^0 , в которой будем запоминать последовательность вершин в кратчайшем пути, заполним так: $s_{ij} = j$

D^0		1	2	3	4	5	6
	1	0	7	9	∞	∞	14
	2	7	0	10	15	∞	∞
	3	9	10	0	11	∞	2
	4	∞	15	11	0	6	∞
	5	∞	∞	∞	6	0	9
	6	14	∞	2	∞	9	0

S^0		1	2	3	4	5	6
	1	1	2	3	4	5	6
	2	1	2	3	4	5	6
	3	1	2	3	4	5	6
	4	1	2	3	4	5	6
	5	1	2	3	4	5	6
	6	1	2	3	4	5	6

Алгоритм Флойда на примере

Основной этап

Задаём строку m и столбец m как *ведущую строку* и *ведущий столбец*.

Для всех элементов d_{ij} матрицы D^{m-1} рассматриваем возможность применения *треугольного оператора*.

Если выполняется неравенство:

$$d_{im} + d_{mj} < d_{ij} \quad (i \neq j, j \neq m, i \neq m),$$

то делаем следующее:

- в матрице D^m заменяем элемент d_{ij} матрицы D^{m-1} суммой $d_{im} + d_{mj}$;
- в матрице S^m меняем элемент s_{ij} матрицы S^{m-1} на m ;
- полагаем $m=m+1$, повторяем основной этап, пока $m < N$

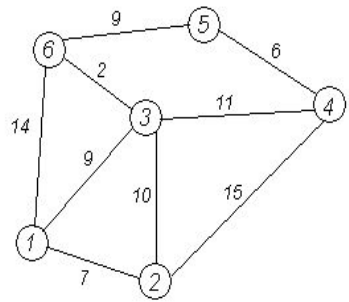
Алгоритм Флойда на примере

D^0

		1	2	3	4	5	6
1	0	7	9	∞	∞	14	
2	7	0	10	15	∞	∞	
3	9	10	0	11	∞	2	
4	∞	15	11	0	6	∞	
5	∞	∞	∞	6	0	9	
6	14	∞	2	∞	9	0	

S^0

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



D^1

		1	2	3	4	5	6
1	0	7	9	∞	∞	14	
2	7	0	10	15	∞	21	
3	9	10	0	11	∞	2	
4	∞	15	11	0	6	∞	
5	∞	∞	∞	6	0	9	
6	14	21	2	∞	9	0	

S^1

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

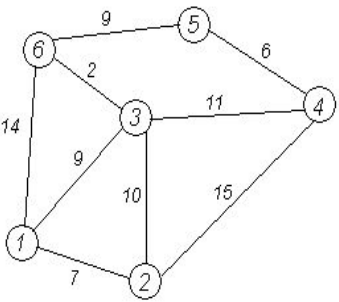
Алгоритм Флойда на примере

D^1

	1	2	3	4	5	6
1	0	7	9	∞	∞	14
2	7	0	10	15	∞	21
3	9	10	0	11	∞	2
4	∞	15	11	0	6	∞
5	∞	∞	∞	6	0	9
6	14	21	2	∞	9	0

S^1

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



D^2

	1	2	3	4	5	6
1	0	7	9	22	∞	14
2	7	0	10	15	∞	21
3	9	10	0	11	∞	2
4	22	15	11	0	6	∞
5	∞	∞	∞	6	0	9
6	14	21	2	36	9	0

S^2

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

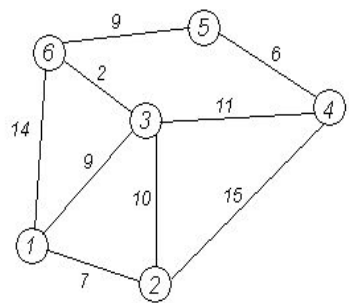
Алгоритм Флойда на примере

D^2

		1	2	3	4	5	6
1	0	7	9	22	∞	14	
2	7	0	10	15	∞	21	
3	9	10	0	11	∞	2	
4	22	15	11	0	6	∞	
5	∞	∞	∞	6	0	9	
6	14	21	2	36	9	0	

S^2

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



D^3

		1	2	3	4	5	6
1	0	7	9	20	∞	11	
2	7	0	10	15	∞	12	
3	9	10	0	11	∞	2	
4	20	15	11	0	6	13	
5	∞	∞	∞	6	0	9	
6	11	12	2	13	9	0	

S^3

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

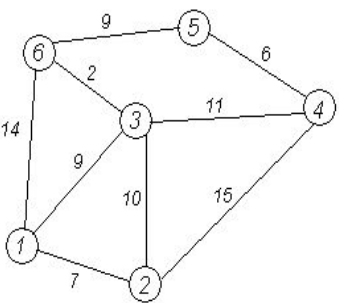
Алгоритм Флойда на примере

D^3

		1	2	3	4	5	6
1	0	7	9	20	∞	11	
2	7	0	10	15	∞	12	
3	9	10	0	11	∞	2	
4	20	15	11	0	6	13	
5	∞	∞	∞	6	0	9	
6	11	12	2	13	9	0	

S^3

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



D^4

		1	2	3	4	5	6
1	0	7	9	20	26	11	
2	7	0	10	15	21	12	
3	9	10	0	11	17	2	
4	20	15	11	0	6	13	
5	26	21	17	6	0	9	
6	11	12	2	13	9	0	

S^4

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

Алгоритм Флойда на примере

D^4

		1	2	3	4	5	6
1	0	7	9	20	26	11	
2	7	0	10	15	21	12	
3	9	10	0	11	17	2	
4	20	15	11	0	6	13	
5	26	21	17	6	0	9	
6	11	12	2	13	9	0	

S^4

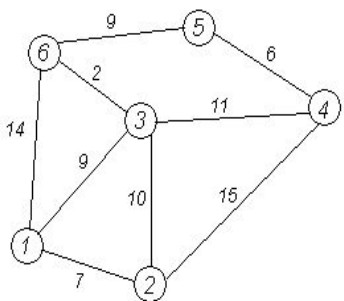
		1	2	3	4	5	6
1	1	2	3	3	4	3	
2	1	2	3	4	4	3	
3	1	2	3	4	4	6	
4	3	2	3	4	5	3	
5	4	4	4	4	5	6	
6	3	3	3	3	5	6	

D^5

		1	2	3	4	5	6
1	0	7	9	20	26	11	
2	7	0	10	15	21	12	
3	9	10	0	11	17	2	
4	20	15	11	0	6	13	
5	26	21	17	6	0	9	
6	11	12	2	13	9	0	

S^5

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



Алгоритм Флойда на примере

D^5

		1	2	3	4	5	6
1	0	7	9	20	26	11	
2	7	0	10	15	21	12	
3	9	10	0	11	17	2	
4	20	15	11	0	6	13	
5	26	21	17	6	0	9	
6	11	12	2	13	9	0	

S^5

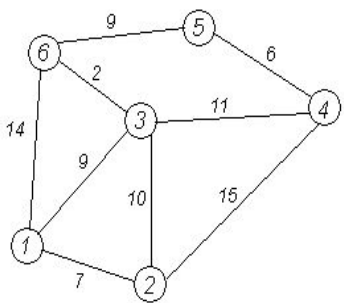
		1	2	3	4	5	6
1	1	2	3	3	4	3	
2	1	2	3	4	4	3	
3	1	2	3	4	4	6	
4	3	2	3	4	5	3	
5	4	4	4	4	5	6	
6	3	3	3	3	5	6	

D^6

		1	2	3	4	5	6
1	0	7	9	20	20	11	
2	7	0	10	15	21	12	
3	9	10	0	11	11	2	
4	20	15	11	0	6	13	
5	20	21	11	6	0	9	
6	11	12	2	13	9	0	

S^6

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



Алгоритм Флойда на примере

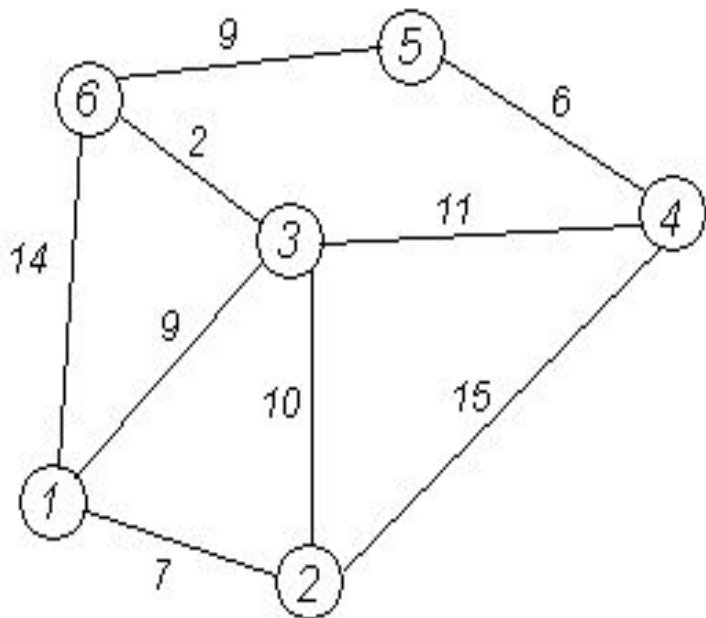
Последний этап.

После реализации N этапов алгоритма определение по матрицам D^n и S^n кратчайшего пути между узлами i и j выполняется по следующим правилам:

1. Кратчайшее расстояние между узлами i и j равно элементу d_{ij} в матрице D^n .
2. Промежуточные узлы пути от узла i к узлу j определяем по матрице S^n . Пусть $s_{ij} = k$, тогда имеем путь $i \rightarrow j \rightarrow k$.

Если далее $s_{ik} = k$ и $s_{kj} = j$, то считаем, что весь путь определён. В противном случае повторяем процедуру для путей от узла i к узлу k и от узла k к узлу j .

Алгоритм Флойда на примере



$$d_{25} = 21$$

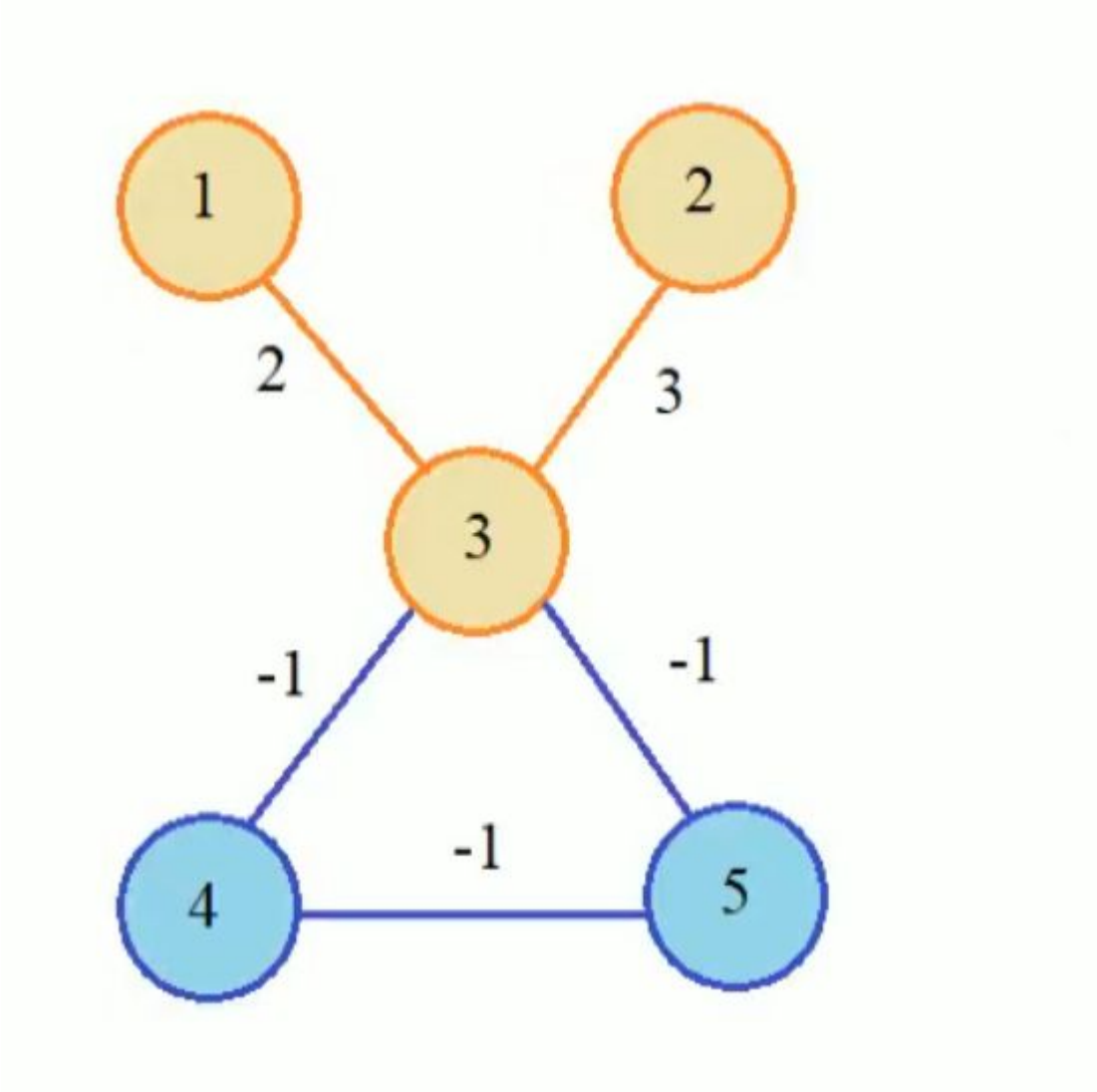
Путь: 2->4->5

$$d_{51} = 20$$

Путь: 5->6->3->1

D^6		1	2	3	4	5	6
	1	0	7	9	20	20	11
	2	7	0	10	15	21	12
	3	9	10	0	11	11	2
	4	20	15	11	0	6	13
	5	20	21	11	6	0	9
	6	11	12	2	13	9	0

S^6		1	2	3	4	5	6
	1	1	2	3	3	6	3
	2	1	2	3	4	4	3
	3	1	2	3	4	6	6
	4	3	2	3	4	5	3
	5	6	4	6	4	5	6
	6	3	3	3	3	5	6



Объем памяти -
?

$$V * V = V^2$$

O-большое - ?

$$O(V) = V^3$$

Алгоритм Дейкстры
 $O(V) = V^3$

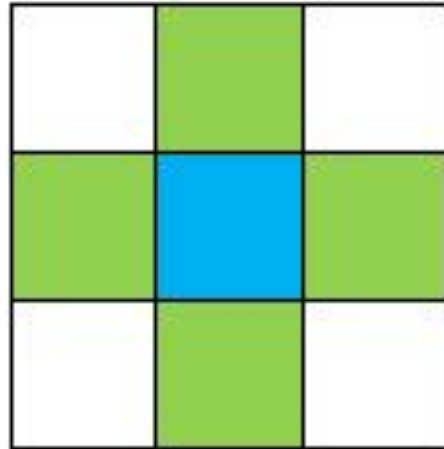
Волновой алгоритм (Алгоритм Ли)

Рассматривается алгоритм построения ортогонального пути.

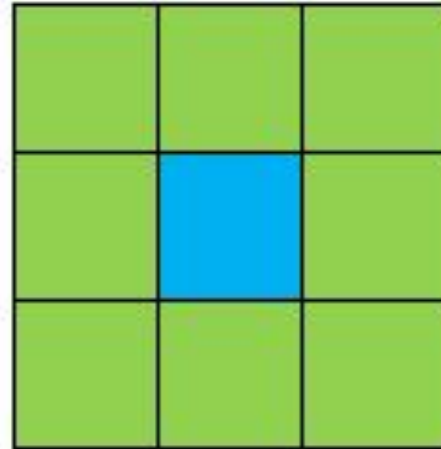
Алгоритм состоит из двух частей.

- 1) В первой от источника к приемнику распространяется волна.
- 2) Во второй выполняется обратный ход, в процессе которого из ячеек волны формируется путь.

Волна, идущая от источника к приемнику, на каждом шаге первой части алгоритма пополняется свободными ячейками, которые, во-первых, еще не принадлежат волне, и, во-вторых, являются 4-соседями ячеек, попавших в волну на предыдущем шаге.



окрестность фон
Неймана



окрестность
Мура

Инициализация

Пометить стартовую ячейку $d := 0$

Распространение волны

ЦИКЛ

ДЛЯ каждой ячейки X , помеченной числом d

пометить все соседние свободные непомеченные ячейки числом $d + 1$

КЦ

$d := d + 1$

ПОКА (финишная ячейка не помечена) **И** (есть возможность распространения волны)

Восстановление пути

ЕСЛИ финишная ячейка помечена

ТО

перейти в финишную ячейку

ЦИКЛ

выбрать среди соседних ячейку, помеченную числом на 1 меньше числа в текущей ячейке

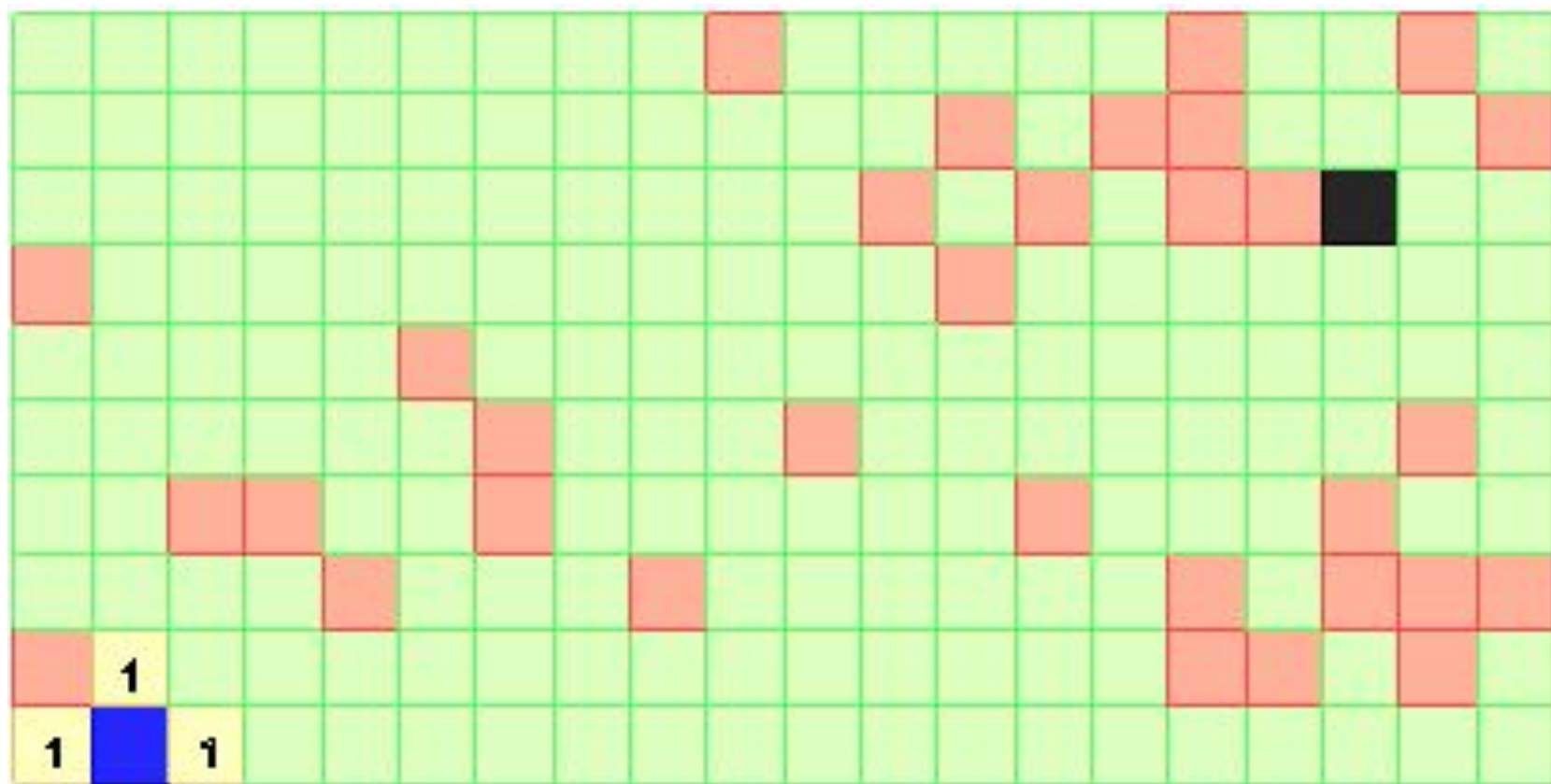
перейти в выбранную ячейку и добавить её к пути

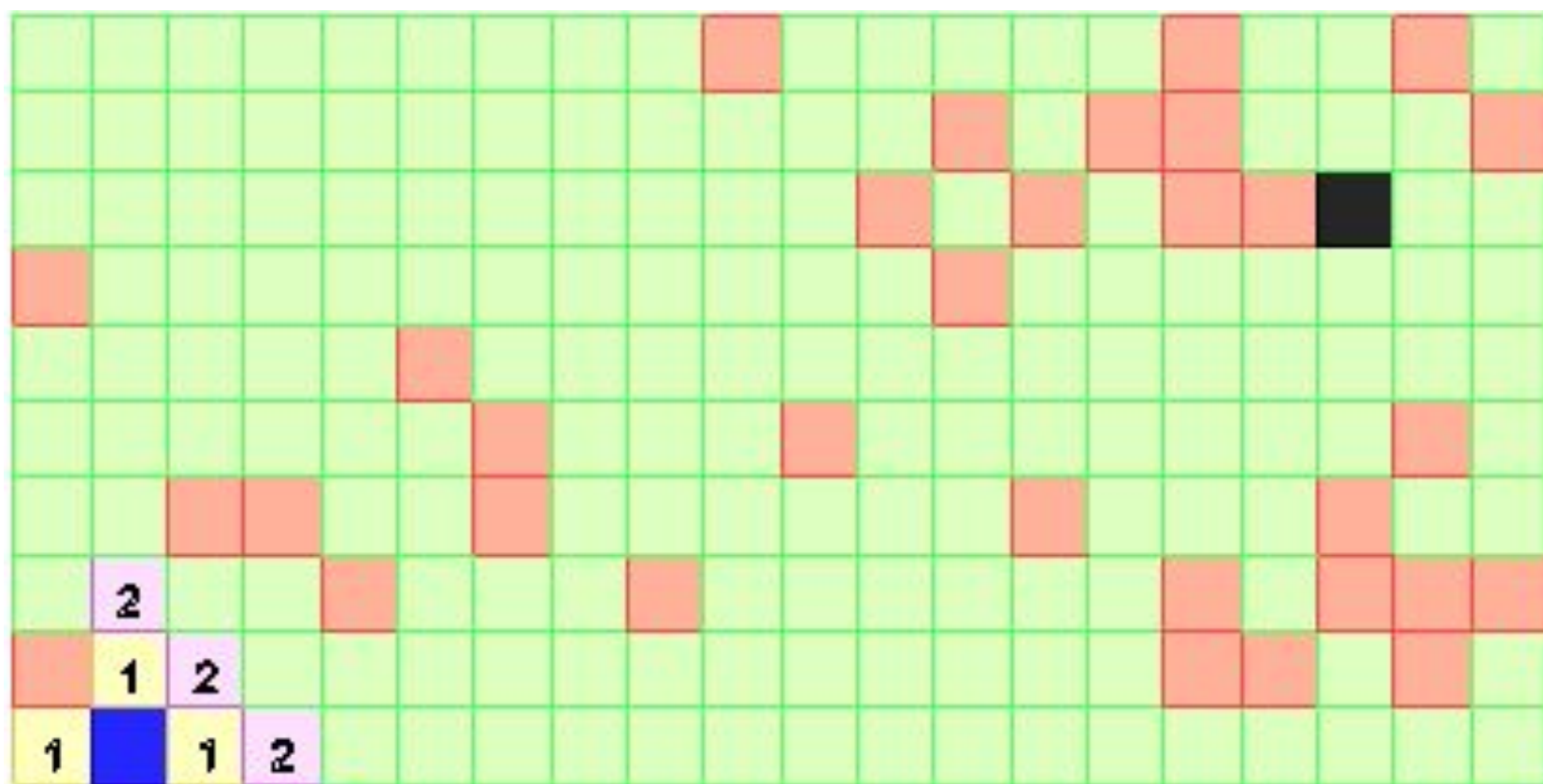
ПОКА текущая ячейка – не стартовая

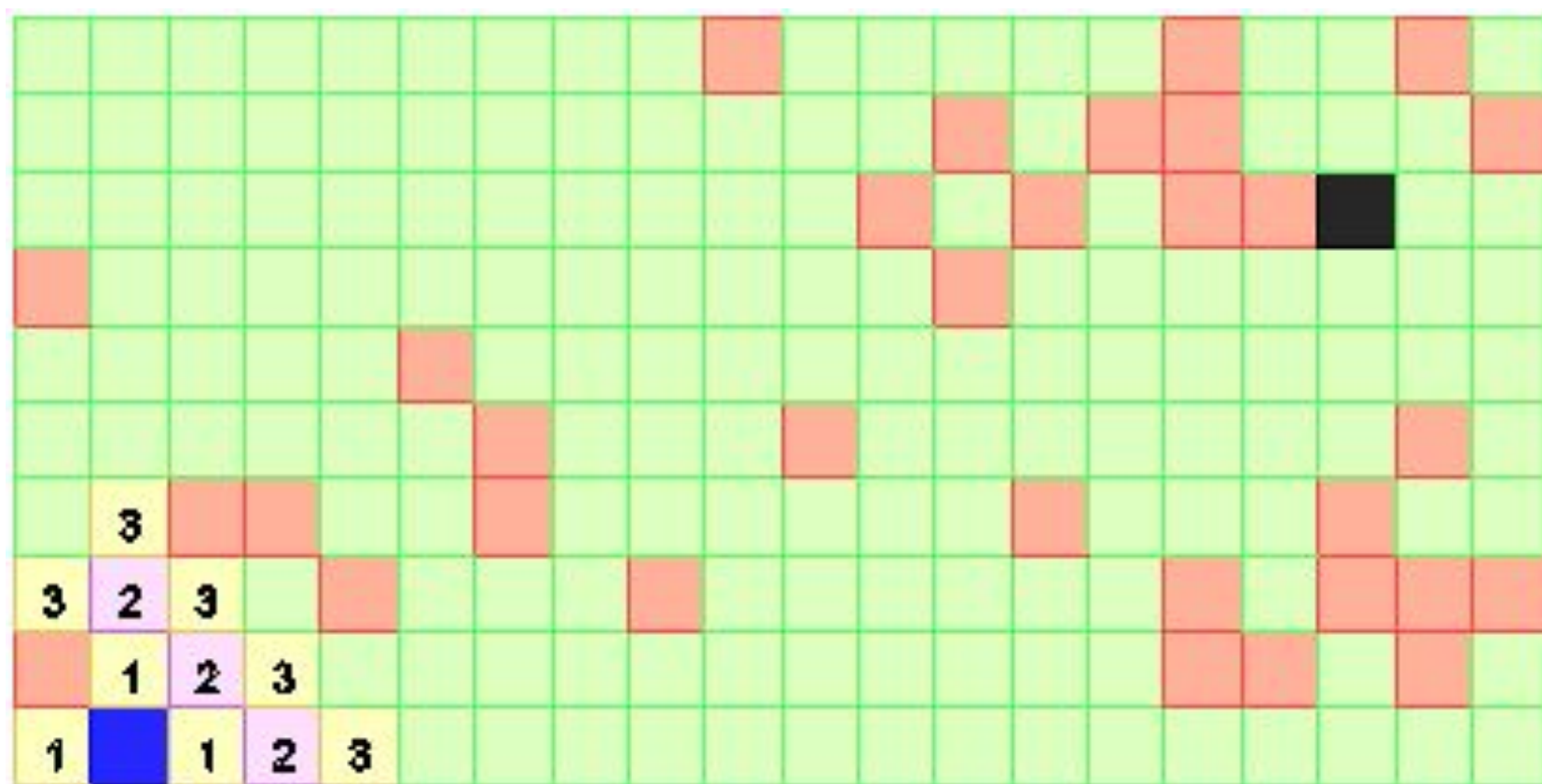
ВОЗВРАТ путь найден

ИНАЧЕ

ВОЗВРАТ путь не найден







10	9	10	11	12	13	14	15	16		18	19	20	21	22					
9	8	9	10	11	12	13	14	15	16	17	18		22						
8	7	8	9	10	11	12	13	14	15	16			20			23			
	6	7	8	9	10	11	12	13	14	15	16		18	19	20	21	22	23	
8	5	6	7	8		12	11	12	13	14	15	16	17	18	19	20	21	22	23
6	4	5	6	7	8		10	11	12		14	15	16	17	18	19	20		
4	3			8	7		9	10	11	12	13	14		16	17	18			
3	2	3	4		8	7	8		10	11	12	13	14	15		16			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14			17		19
1		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

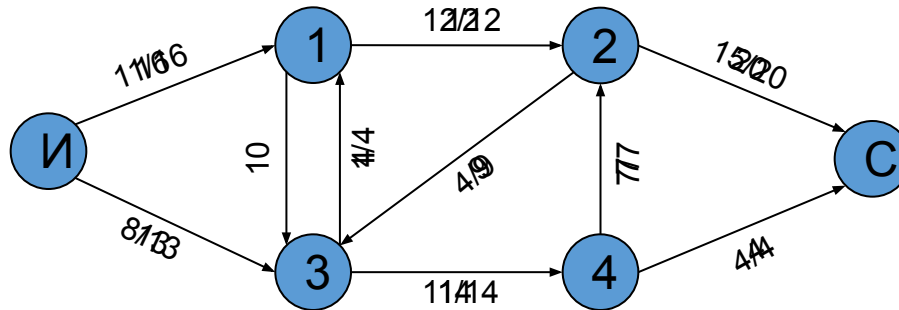
При обратном ходе в путь включается по одной ячейке каждого шага распространения волны. При выборе из двух ячеек приоритет имеет ячейка, обеспечивающая горизонтальное продвижение, что приводит к пути, показанному на рисунке

10	9	10	11	12	13	14	15	16		18	19	20	21	22					
9	8	9	10	11	12	13	14	15	16	17	18		22						
8	7	8	9	10	11	12	13	14	15	16				20			23		
	6	7	8	9	10	11	12	13	14	15	16		18	19	20	21	22	23	
6	5	6	7	8		12	11	12	13	14	15	16	17	18	19	20	21	22	23
5	4	5	6	7	8		10	11	12		14	15	16	17	18	19	20		
4	3			8	7		9	10	11	12	13	14		16	17	18			
3	2	3	4		6	7	8		10	11	12	13	14	15		18			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14			17		19
1		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Алгоритм Форда – Фалкерсона

Потоки в сетях.

Сеть – это ориентированный нагруженный граф, в котором нагрузка имеет интерпретацию «пропускная способность» (разумеется, положительная; можно считать, что отсутствующее ребро соответствует нулевой нагрузке). Будем обозначать эту нагрузку $c(u, v)$.



Мы будем считать, что

- в сети есть две выделенные вершины – исток (только исходящие дуги) и сток (только входящие дуги);
- любая вершина лежит на каком-нибудь пути из истока в сток (нет «бесполезных» вершин).

Поток в сети – это задание некоторой дополнительной нагрузки на ребра.

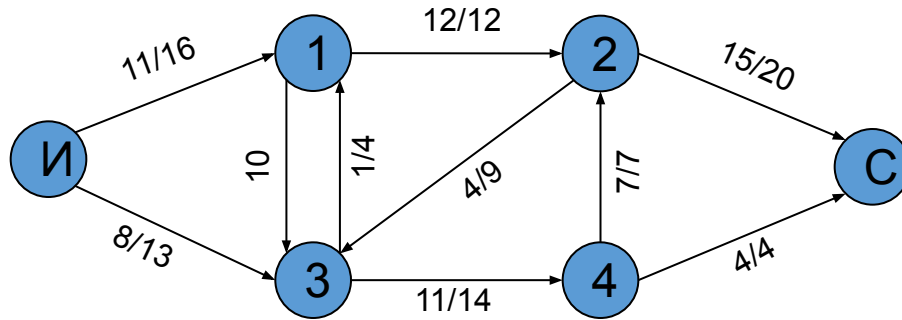
Свойства потока:

- поток по ребру не может превышать пропускной способности ребра и всегда неотрицателен;
- в любую вершину (кроме истока и стока) количество втекающей жидкости равно количеству вытекающей.

Величина потока – это сумма исходящего потока из истока. Очевидно, она равна сумме входящего потока в сток. Основная задача – найти максимальный поток в сети с заданной пропускной способностью.

Потоки в сетях.

Пусть задана сеть и поток в ней. Свяжем с потоком функцию $f(u, v)$, обладающую следующими свойствами:



- если по ребру (u, v) идет поток величиной c , то положим $f(u, v) = c$, $f(v, u) = -c$;
- если есть два «встречных» потока c_1 и c_2 по ребрам (u, v) и (v, u) соответственно, то полагаем $f(u, v) = c_1 - c_2$. На самом деле всегда можно считать, что на самом деле есть только один поток величиной $|c_1 - c_2|$.

На приведенной выше картинке: $f(И, 3) = 8$; $f(3, 2) = -4$; $f(1, 3) = -1$.

Для каждого ребра (при заданном потоке f) определим также его «остаточную пропускную способность»: $c_f(u, v) = c(u, v) - f(u, v)$

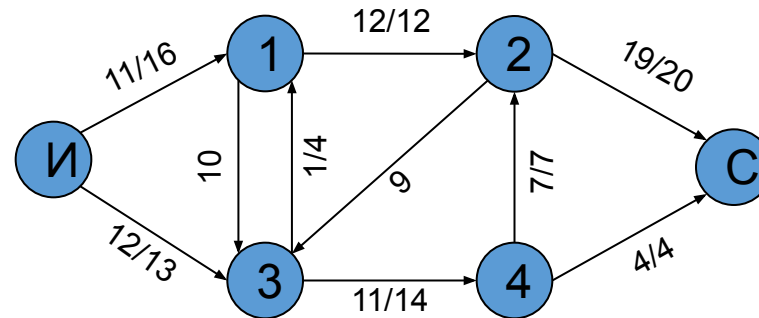
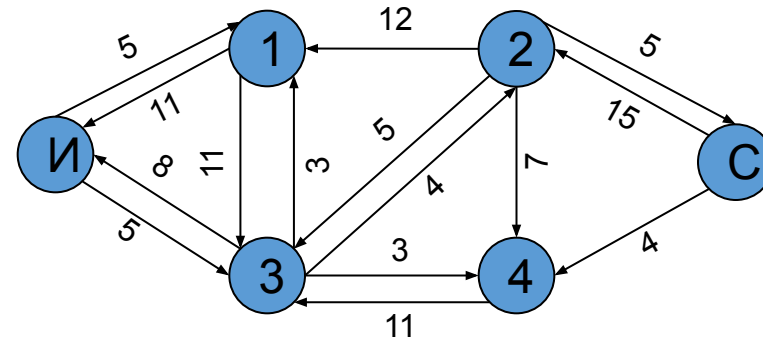
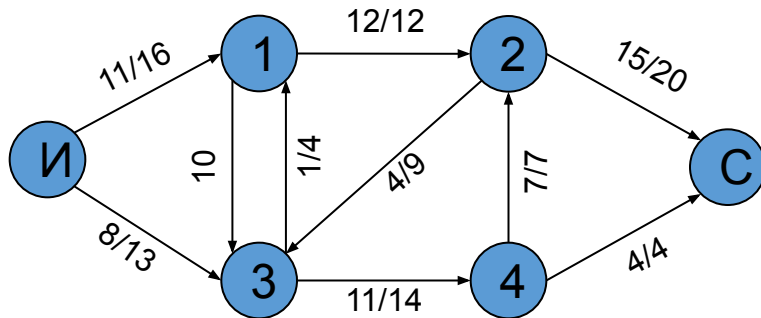
Например, для заданного потока $c_f(3, 4) = 3$, $c_f(1, 3) = 11$.

Метод Форда – Фалкерсона

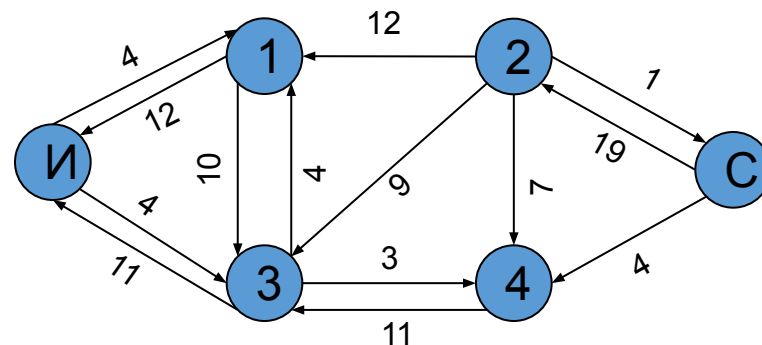
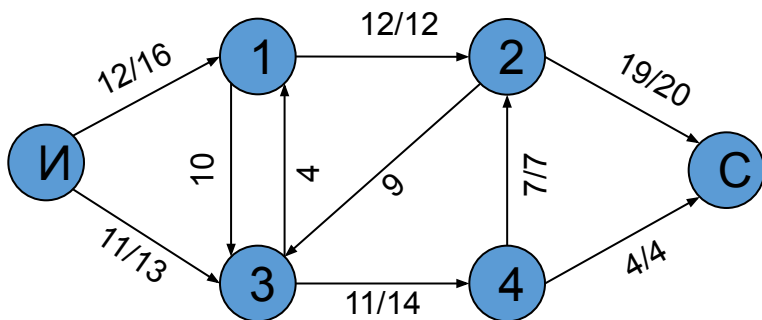
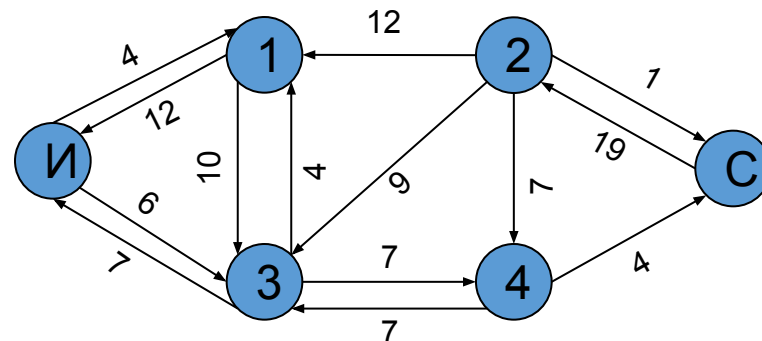
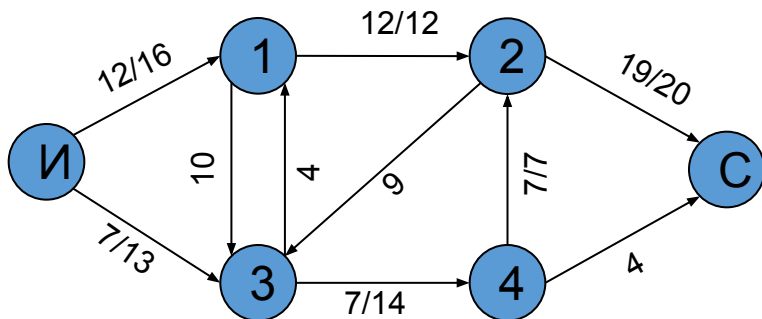
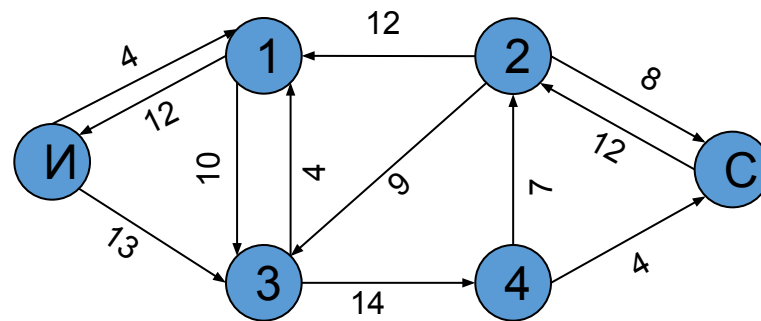
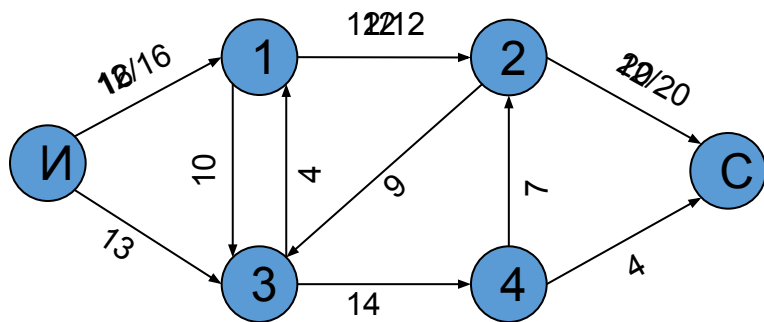
Будем искать *дополняющие пути* из истока в сток, по которому можно пропустить дополнительное количество вещества. Тогда схема алгоритма может быть записана следующим образом:

```
f = 0;  
while (существует дополняющий путь p) {  
    дополнить f вдоль p;  
}
```

Для поиска дополняющих путей найдем все остаточные пропускные способности ребер сети и составим *остаточную сеть* из всех положительных величин:

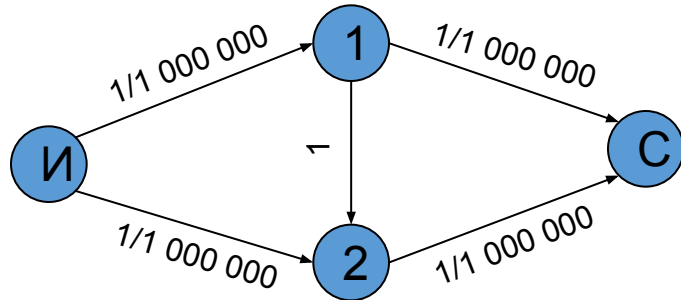
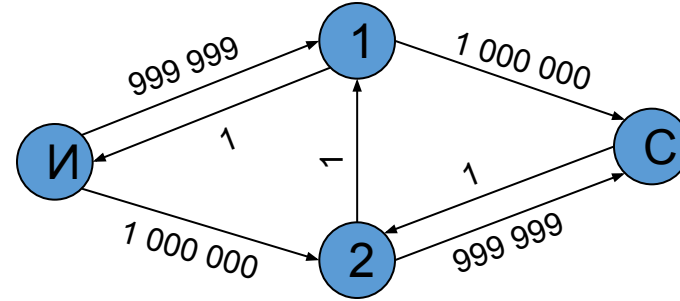
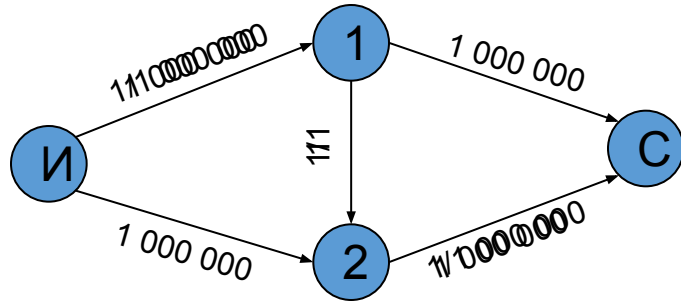


Пример реализации метода Форда – Фалкерсона



Выбор дополняющего пути в методе Форда – Фалкерсона

Если выбор дополняющего пути производится не очень удачно, то процедура поиска максимального потока может затянуться.



и так далее, всего 2 000 000 шагов...

Можно попробовать искать кратчайший (по числу ребер) дополняющий путь между истоком и стоком. Например, с помощью поиска в ширину. Получающийся при этом алгоритм (реализация метода Форда – Фалкерсона) называется алгоритмом Эдмондса – Карпа).

Можно показать, что в алгоритме Эдмондса – Карпа число шагов ограничено сверху числом $2nm$, где m – число ребер в сети, а n – число вершин. Поскольку поиск в ширину, изменение потоков вдоль ребер и построение остаточной сети требуют времени $O(m)$, то общее время работы алгоритма можно оценить как $O(m^2n)$.