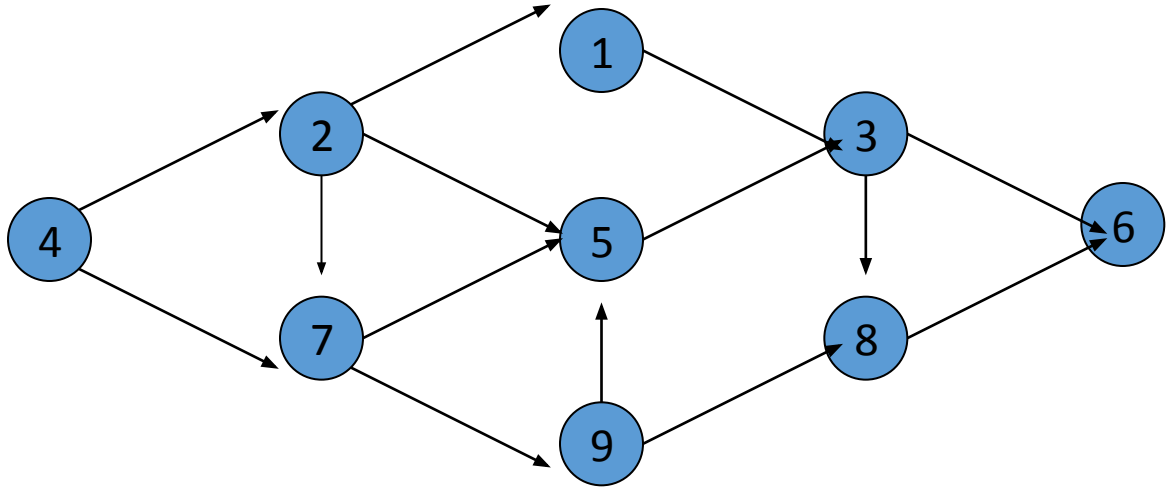
Графы

Топологическая сортировка

Топологическая сортировка вершин ориентированного графа без циклов.

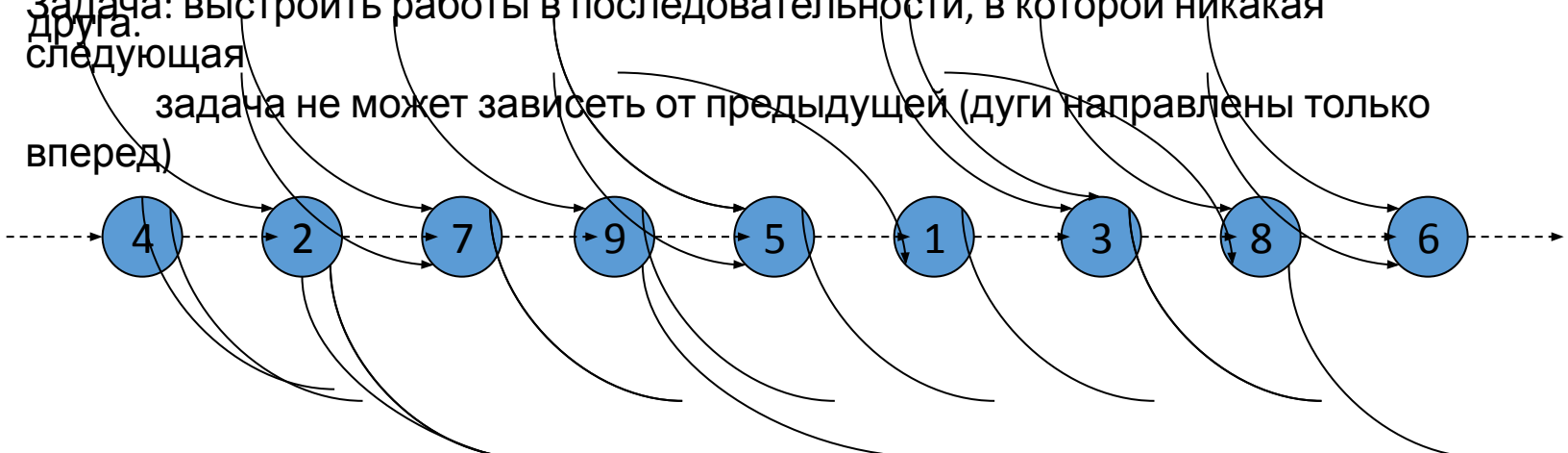
DAG – Directed Acyclic Graph – ориентированный граф без циклов



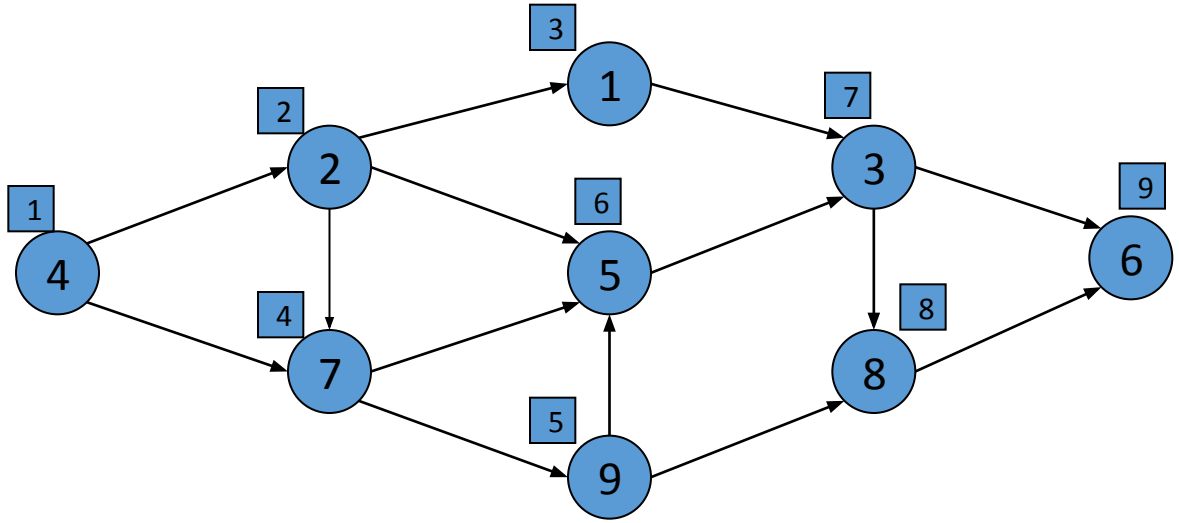
Интерпретация: вершины – элементарные работы;

дуги – зависимость работ друг от друга.

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.

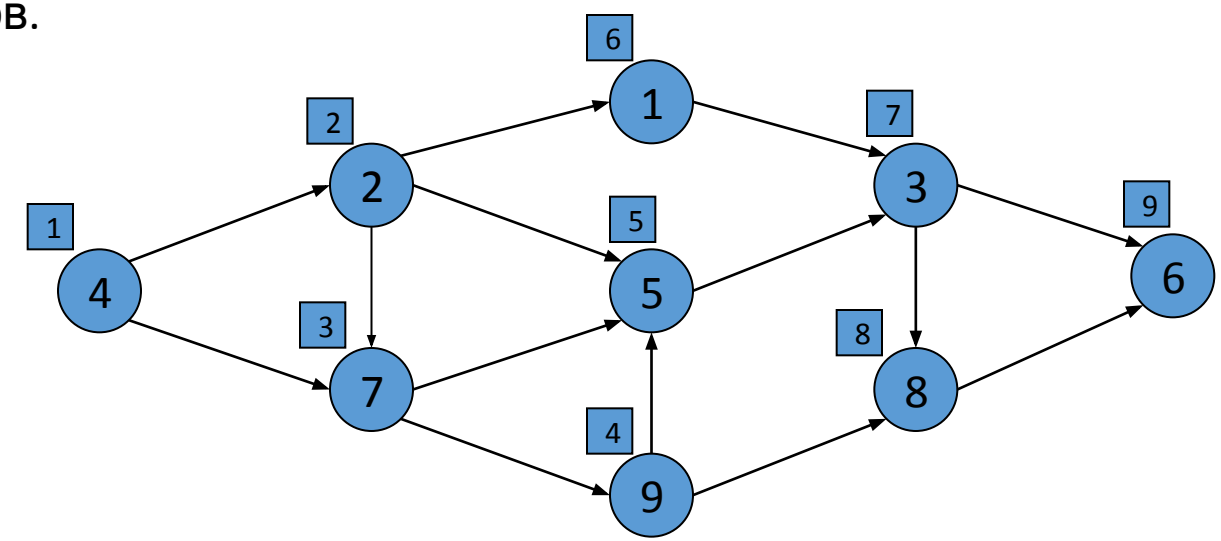
Оценка времени работы алгоритма.

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 удается 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 отличии 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 NxN, 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:

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

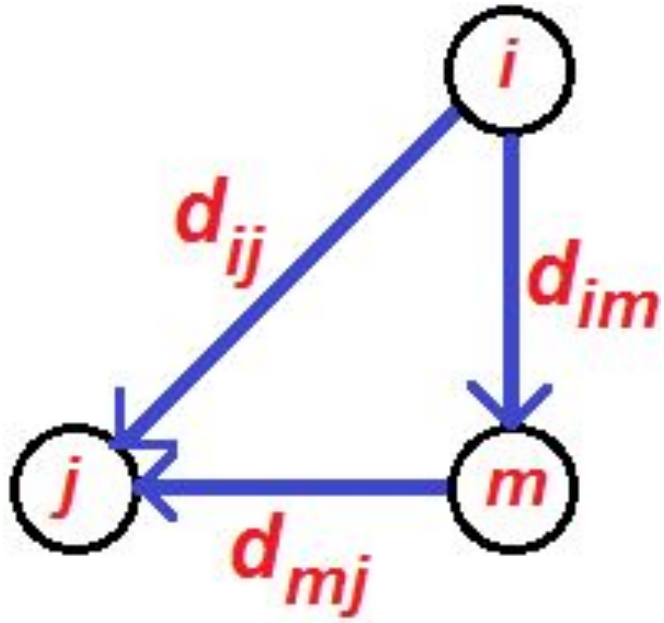
Процесс повторяется до тех пор, пока по матрице D^{N-1} не будет вычислена матрица D^N.

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}.

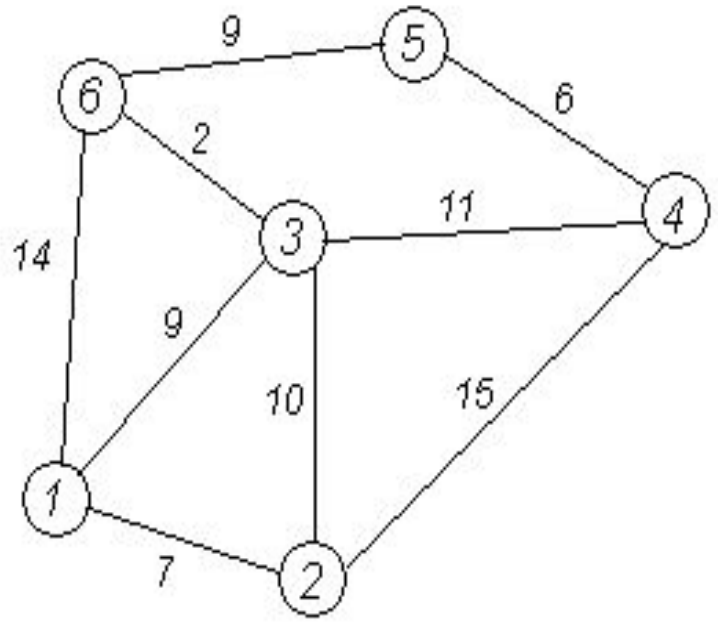
Если выполняется неравенство $d_{im} + d_{mj} < d_{ij}$, то целесообразно заменить путь $i\ j$ путём $i\ m + m\ j$



«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:

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

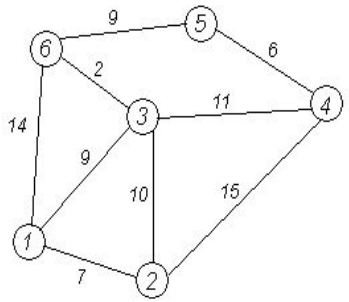
то делаем следующее:

a) в матрице D^m заменяем элемент d_{ij} матрицы D^{m-1} суммой $d_{im} + d_{mj}$;

b) в матрице S^m меняем элемент s_{ij} матрицы S^{m-1} на m;

c) 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

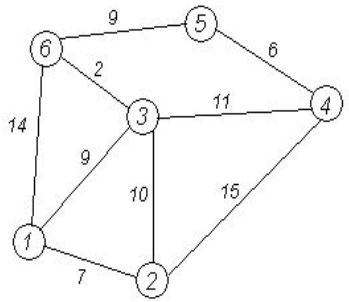
D^1

	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

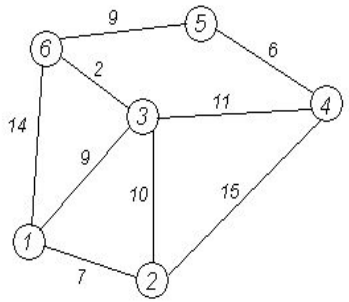
D^2

	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

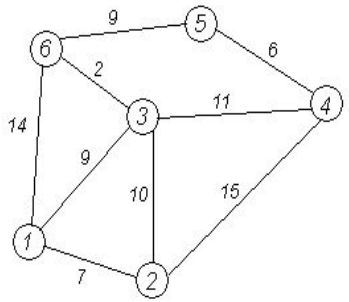
D^3

	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

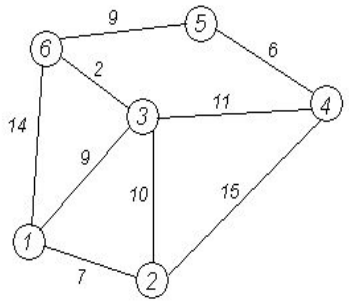
D^4

	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

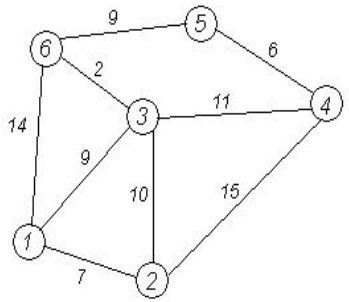
D^5

	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

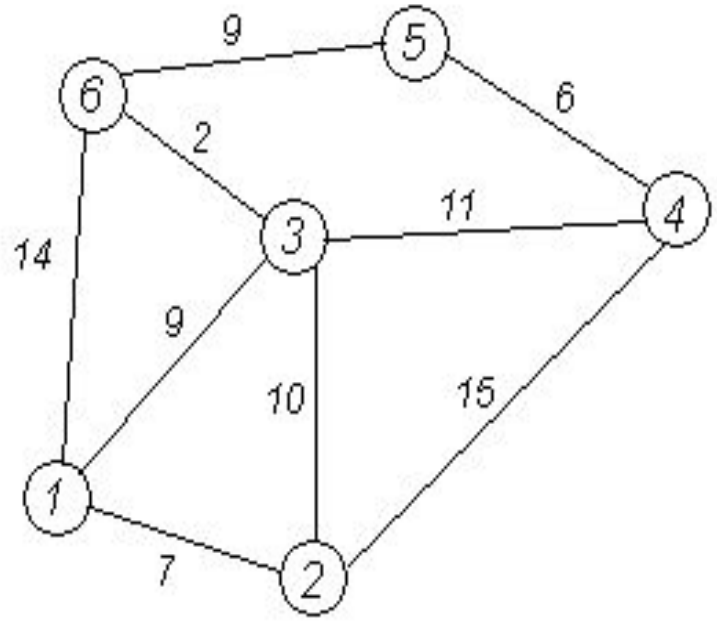
Последний этап.

После реализации ***N*** этапов алгоритма определение по матрицам ***Dn*** и ***Sn*** 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 -> j-> 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

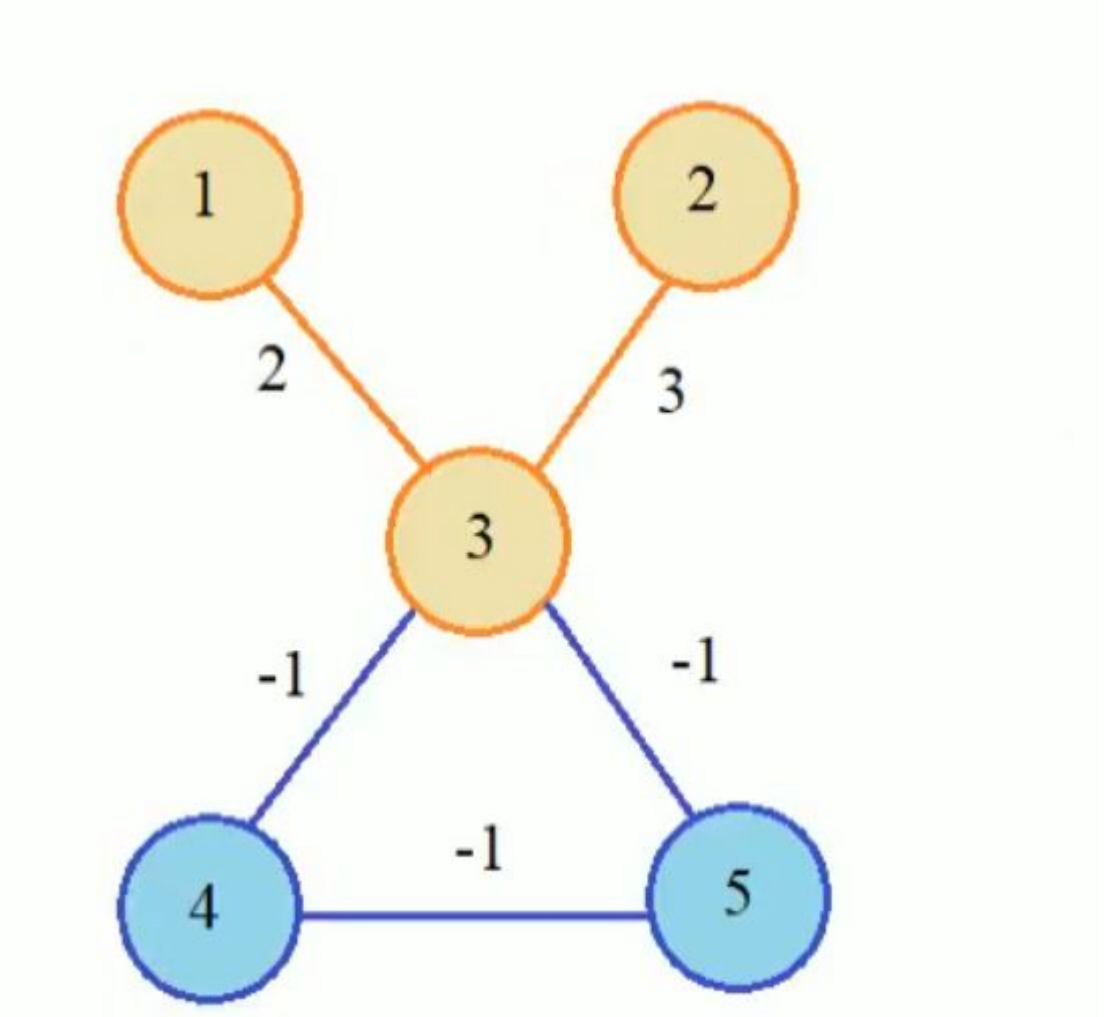
Путь: 5->6 ->3 -> 1

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



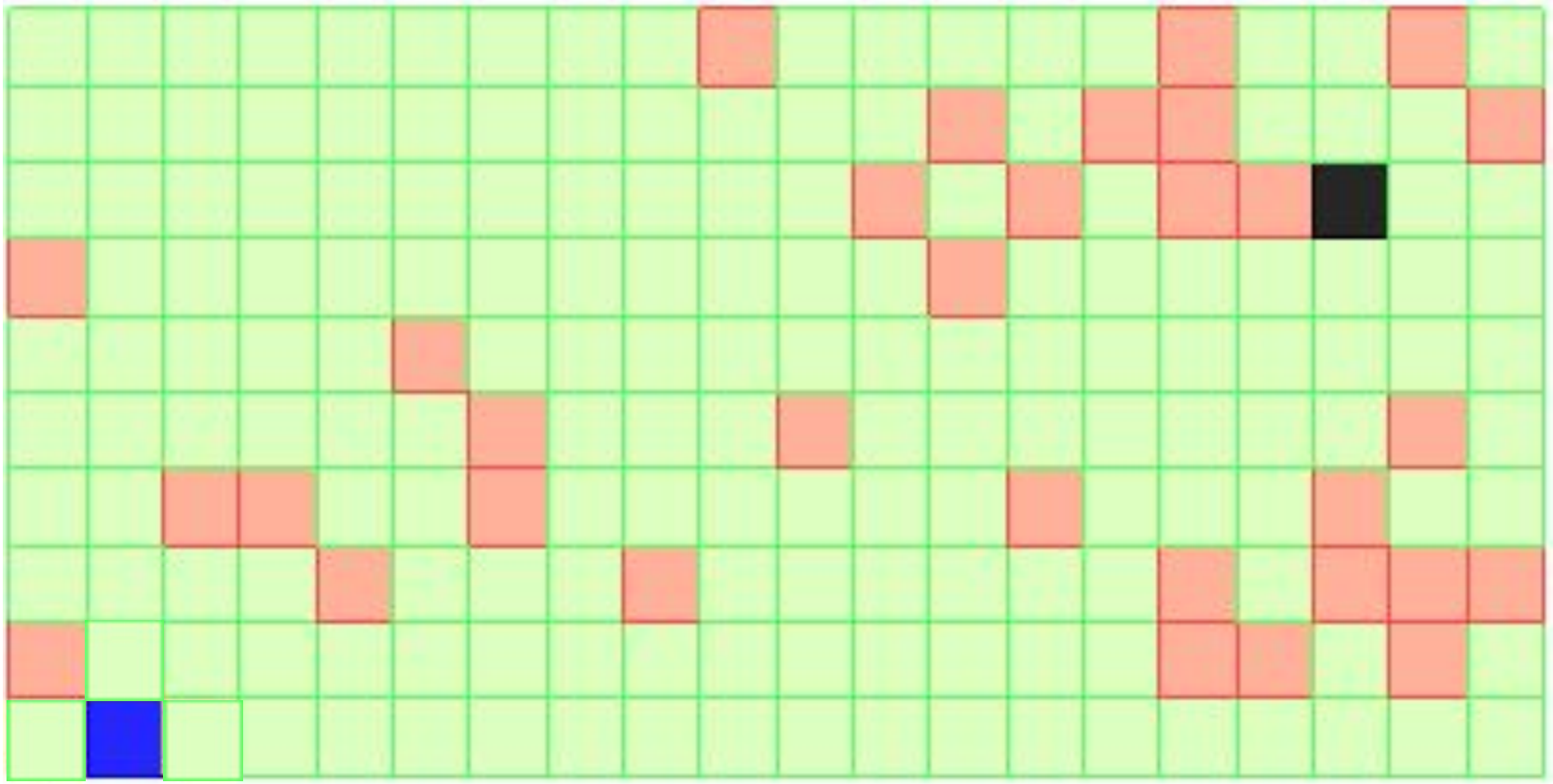
Объем памяти - ?

$V * V = V^2$

O-большое - ?

$O(V) = V^3$

Алгоритм Дейкстры
$O(V) = V^3$



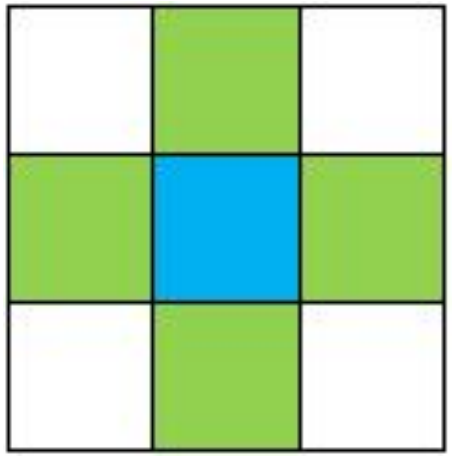
Волновой алгоритм (Алгоритм Ли)

Рассматривается алгоритм построения ортогонального пути.

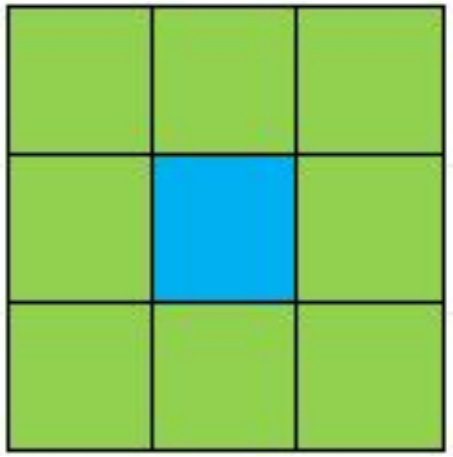
Алгоритм состоит из двух частей.

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

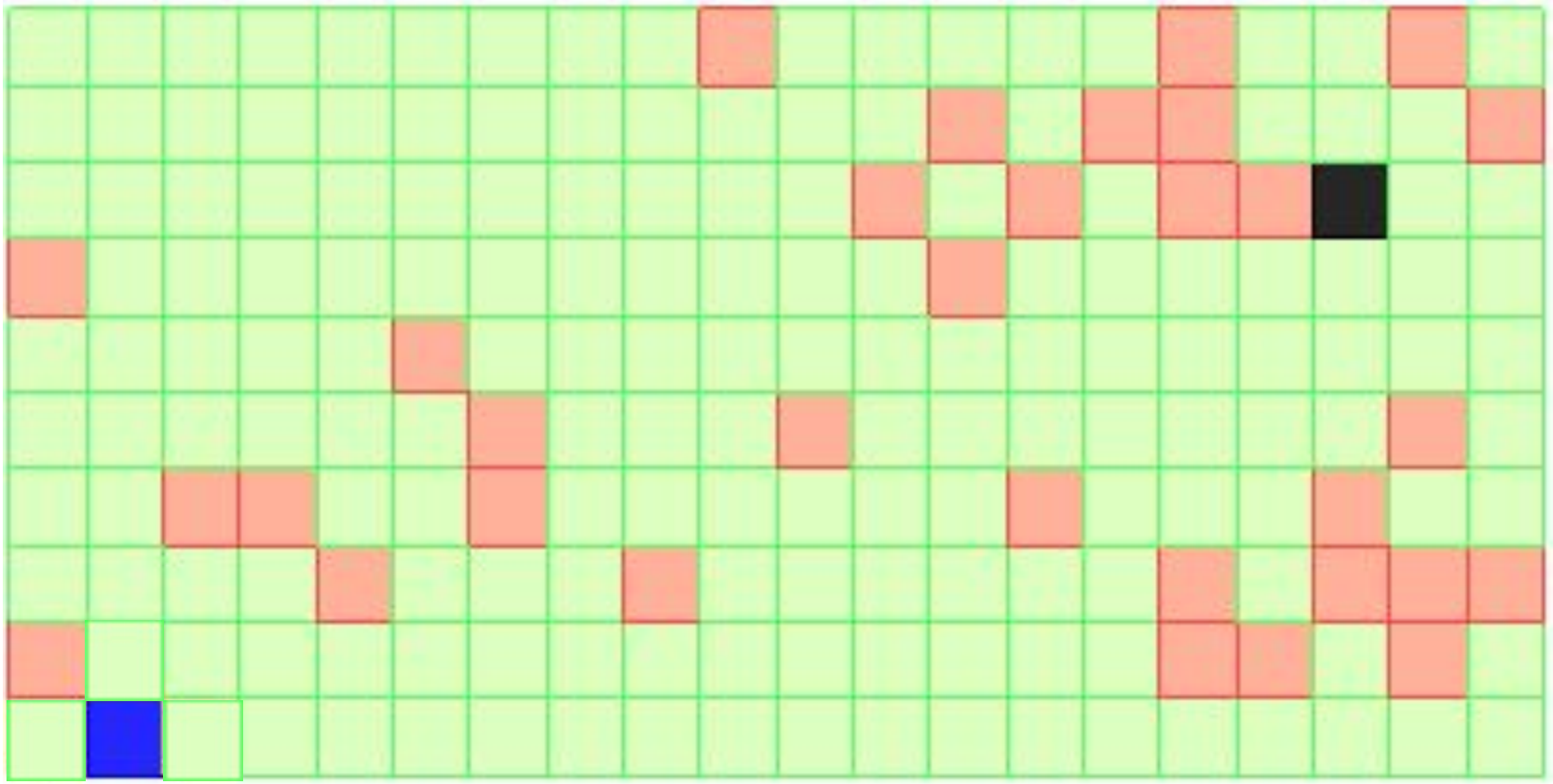
Волна, идущая от источника к приемнику, на каждом шаге первой части алгоритма пополняется свободными ячейками, которые, во-первых, еще не принадлежат волне, и, во-вторых, являются 4-соседями ячеек, попавших в волну на предыдущем шаге.



окрестность фон Неймана



окрестность Мура



Инициализация

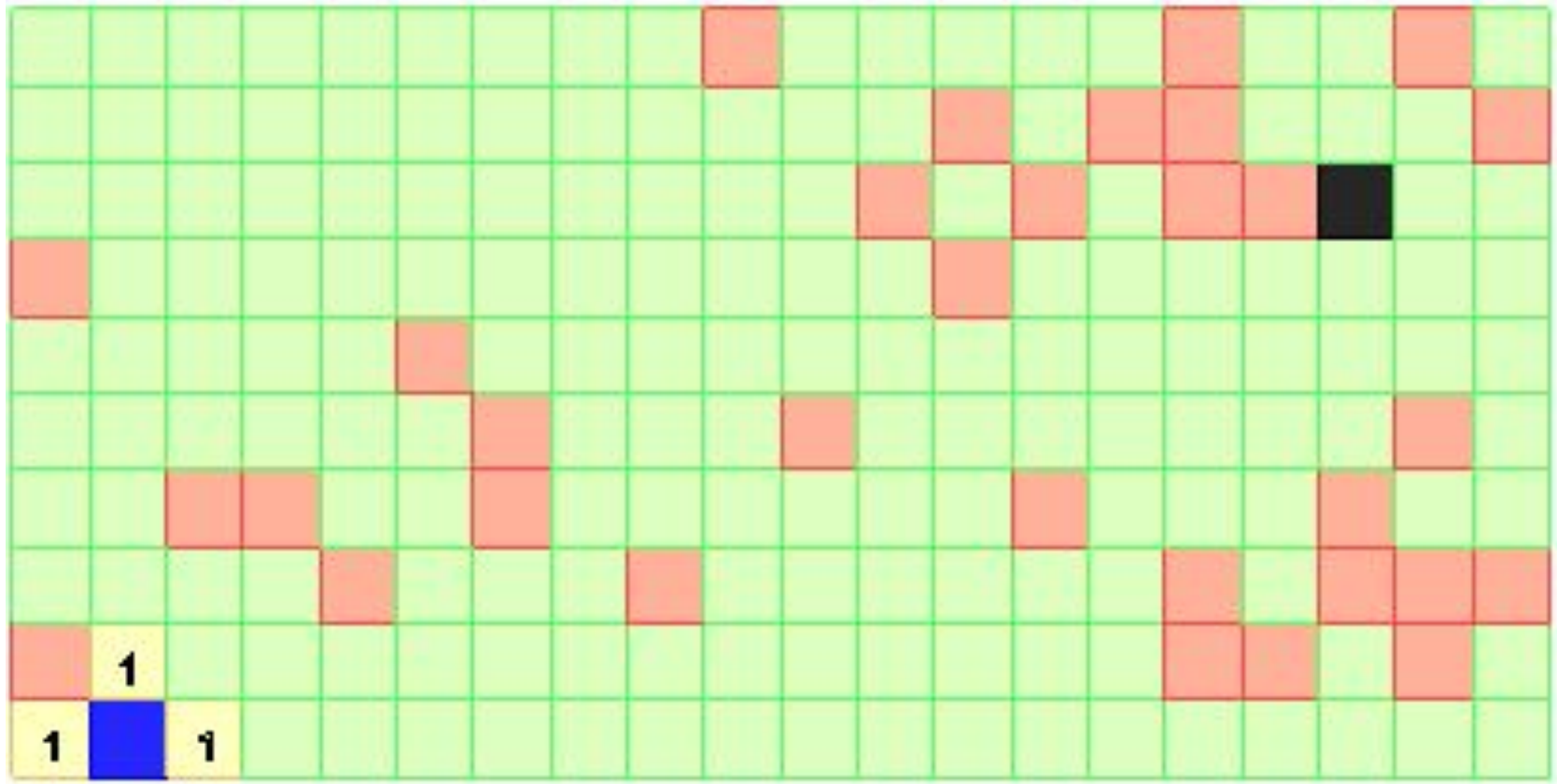
Пометить стартовую ячейку $d := 0$

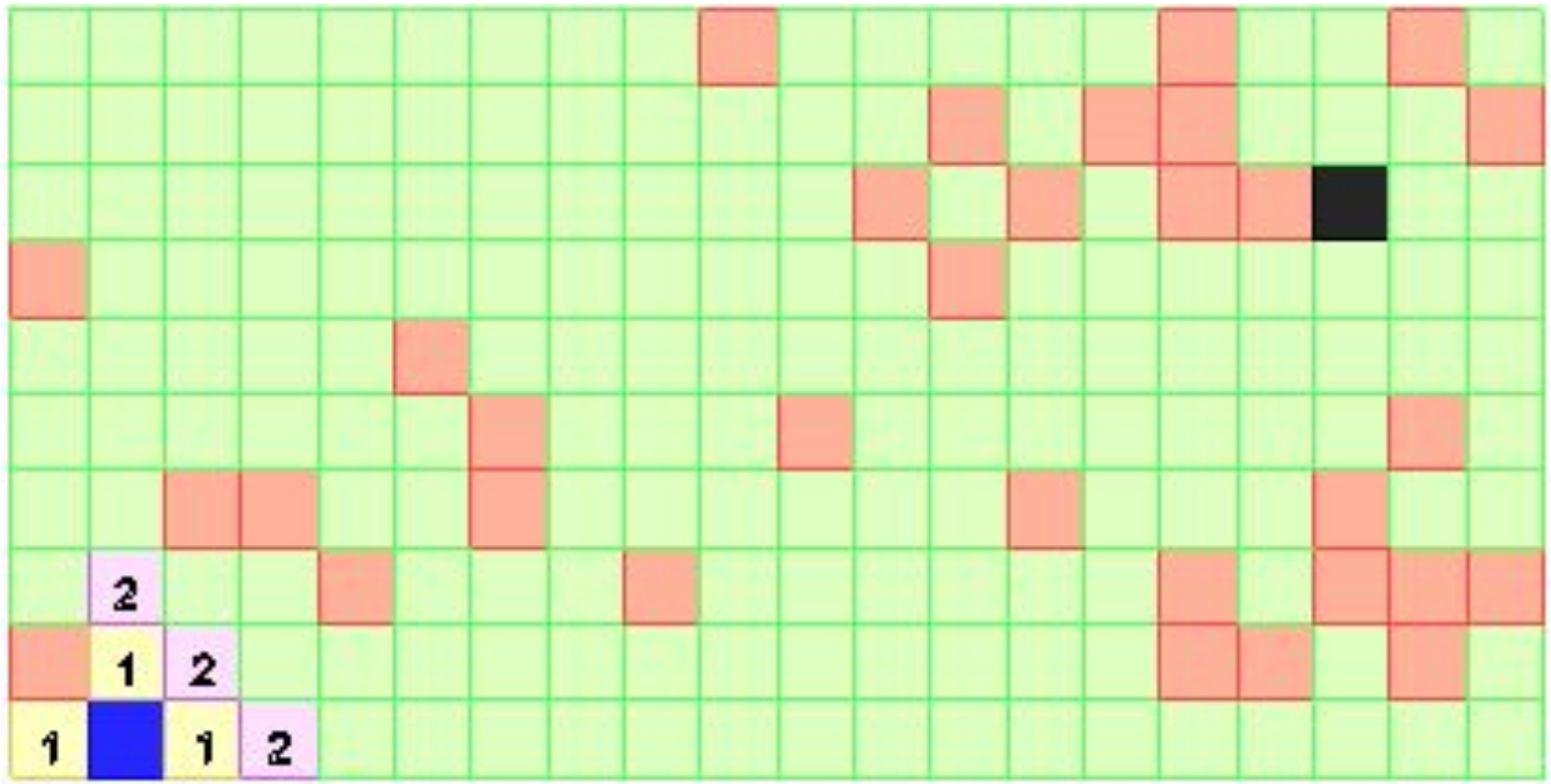
Распространение волны

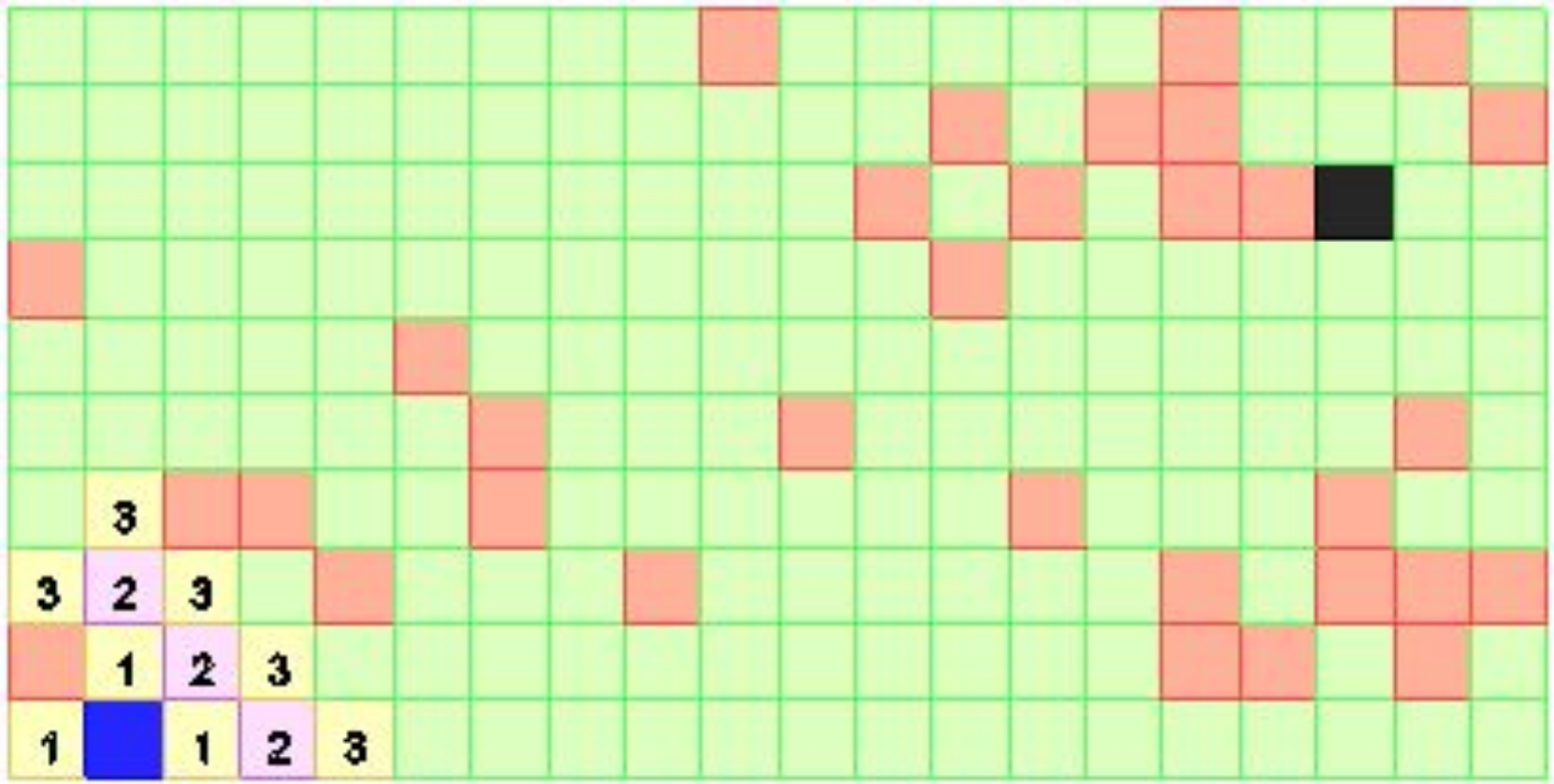
```
ЦИКЛ
    ДЛЯ каждой ячейки X, помеченной числом d
        пометить все соседние свободные непомеченные ячейки числом d + 1
    КЦ
    d := d + 1
ПОКА (финишная ячейка не помечена) И (есть возможность распространения волны)
```

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
```

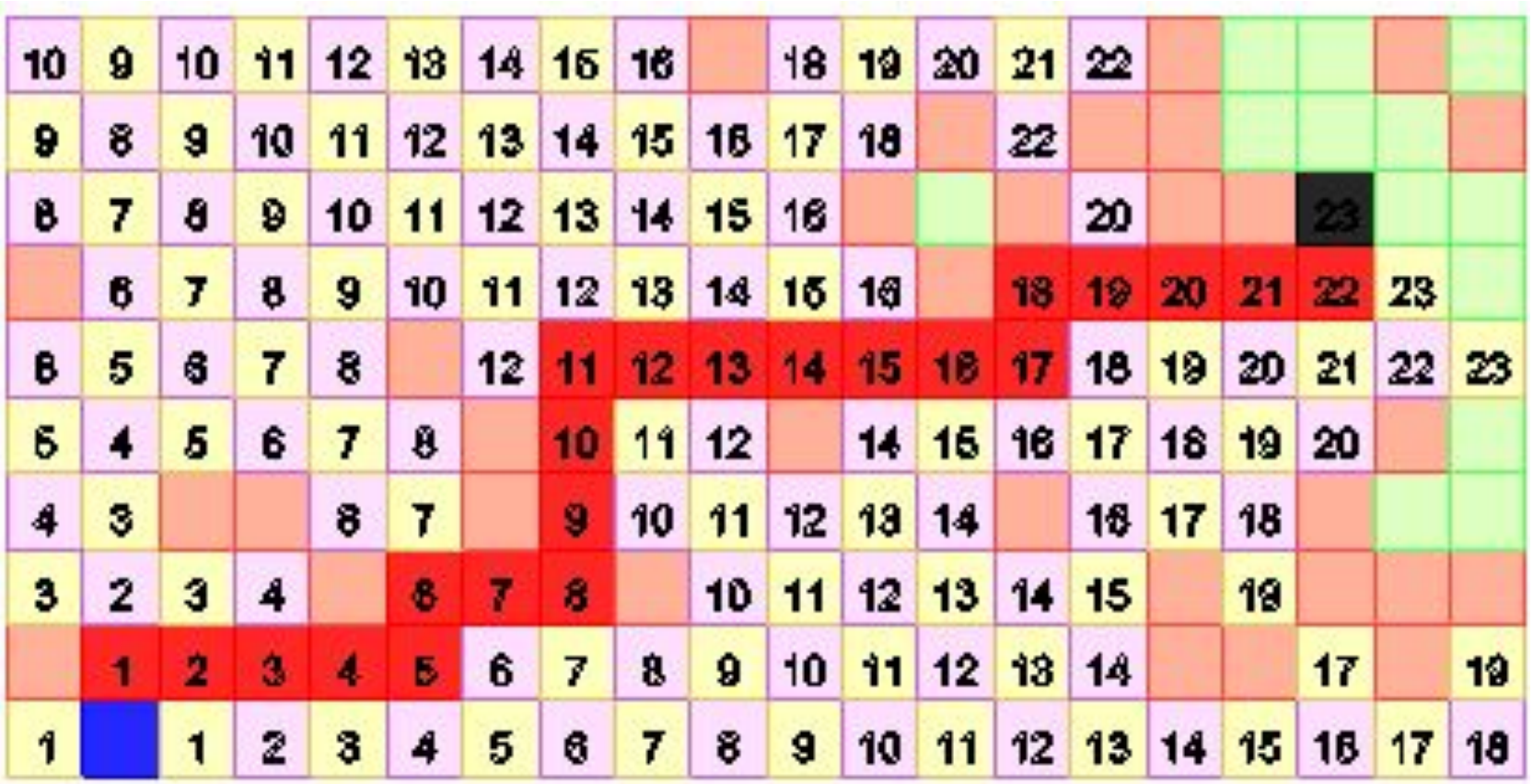

1
1
1


2
1 2
1 1 2


3
3 2 3
1 2 3
1 1 2 3


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 8 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

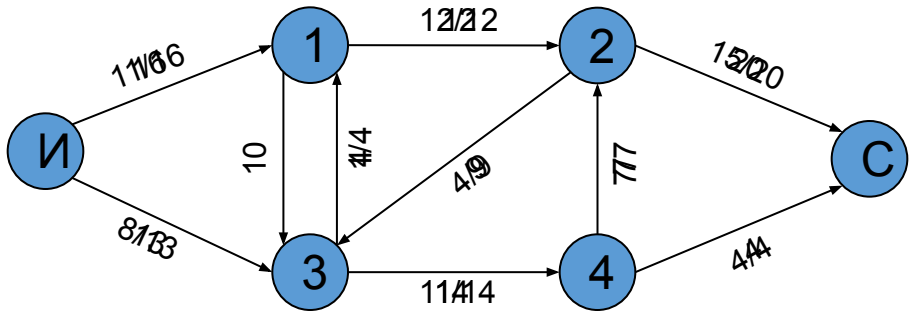
При обратном ходе в путь включается по одной ячейке каждого шага распространения волны. При выборе из двух ячеек приоритет имеет ячейка, обеспечивающая горизонтальное продвижение, что приводит к пути, показанному на рисунке



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 с (u, v).



Мы будем считать, что

a) в сети есть две выделенные вершины – исток (только исходящие дуги) и сток (только входящие дуги);

b) 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:

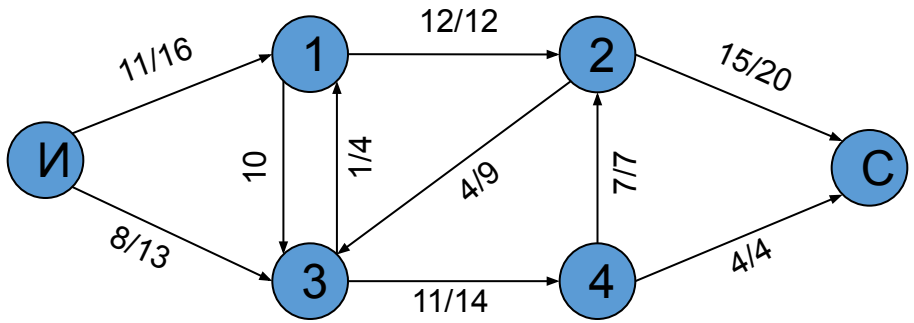
a) поток по ребру не может превышать пропускной способности ребра и всегда неотрицателен;

b) 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:



a) если по ребру (u,v) идет поток величиной *c*, то положим f (u, v) = c, f (v, u) = -c;

b) если есть два «встречных» потока c1 и c2 по ребрам (u, v) и (v, u) соответственно, то полагаем f (u, v) = c1-c2. На самом деле всегда можно считать, что на самом деле есть только один поток величиной |c1-c2|.

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»: $c_f(u, v) = c(u, v) – f(u, v)$

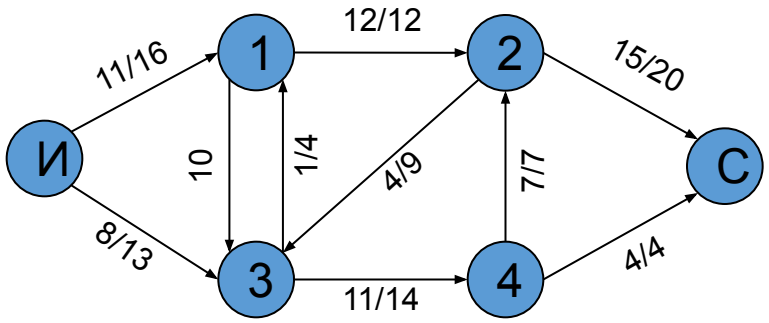
Например, для заданного потока $c_f(3, 4) = 3$, $c_f(1, 3) = 11$.

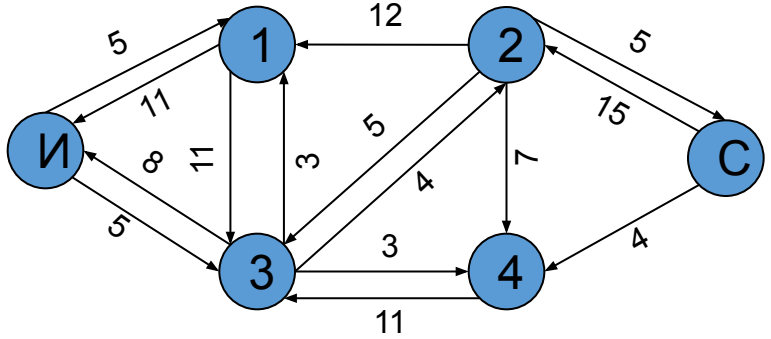
Метод Форда – Фалкерсона

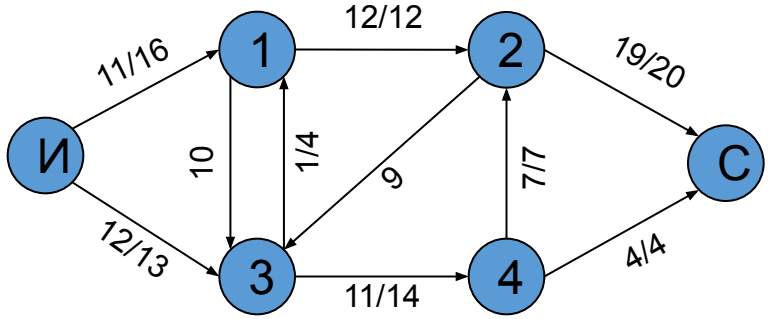
Будем искать *дополняющие пути* из истока в сток, по которому можно пропустить дополнительное количество вещества. Тогда схема алгоритма может быть записана следующим образом:

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

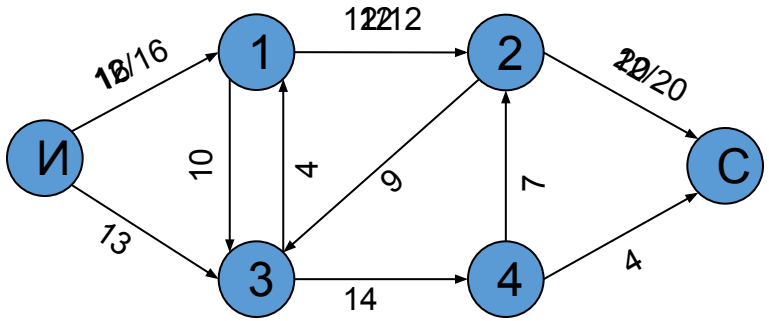
Для поиска дополняющих путей найдем все остаточные пропускные способности ребер сети и составим *остаточную сеть* из всех положительных величин:

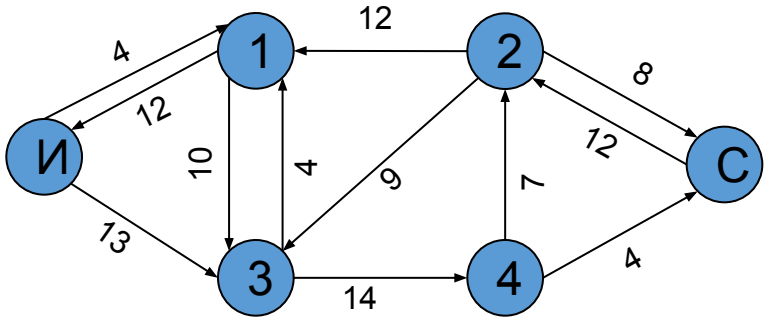


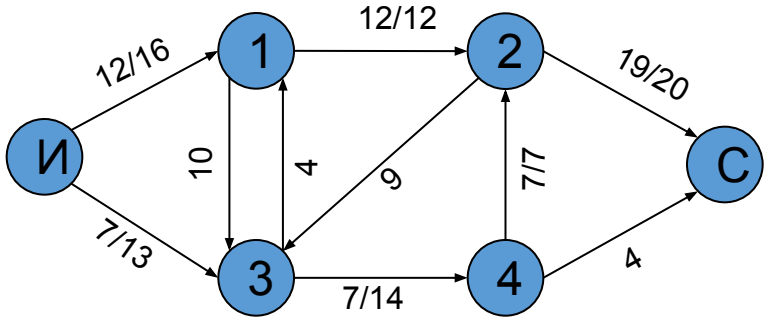


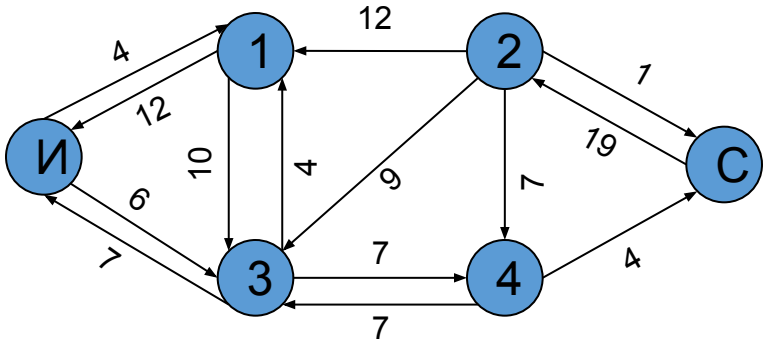


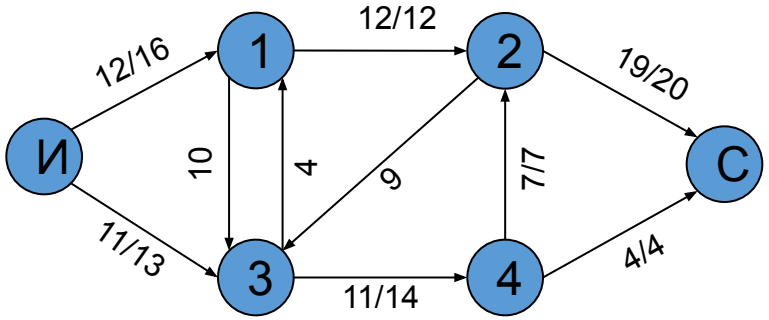
Пример реализации метода Форда – Фалкерсона

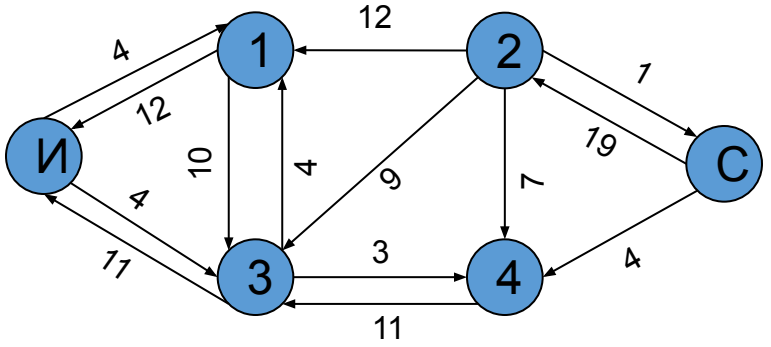






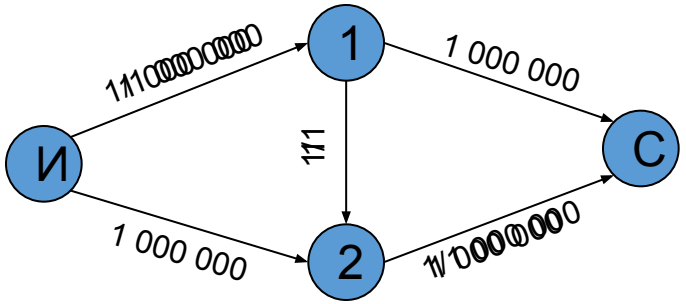


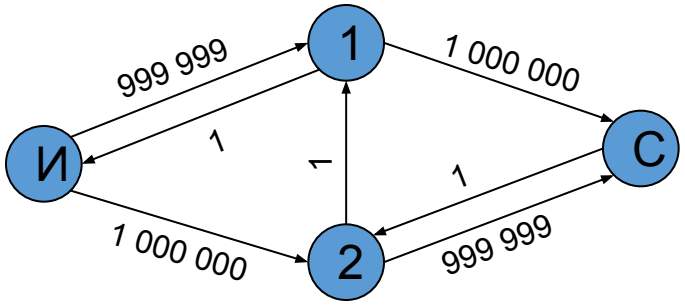


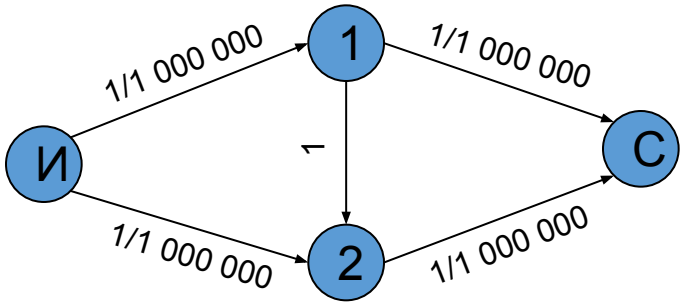


Выбор дополняющего пути в методе Форда – Фалкерсона

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







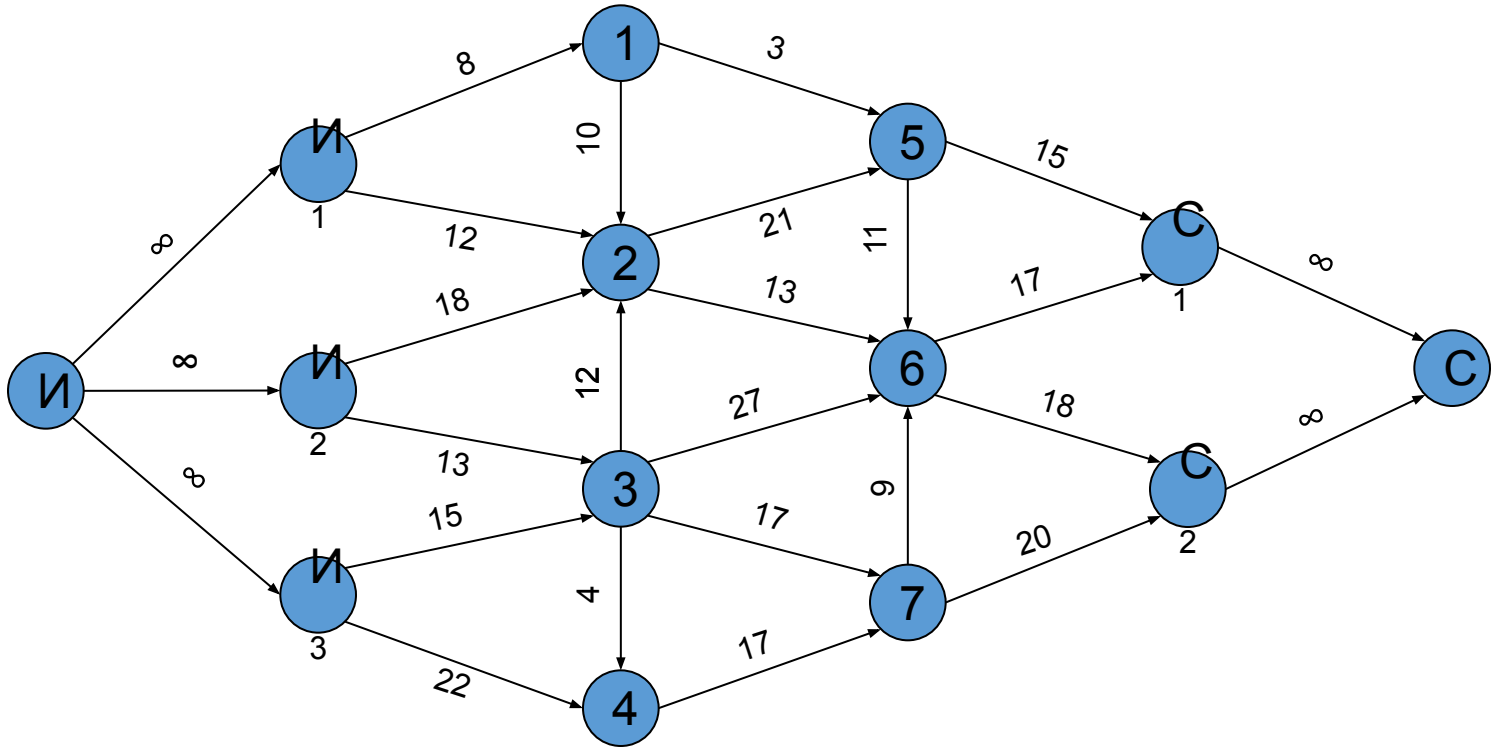
и так далее, всего 2 000 000 шагов…

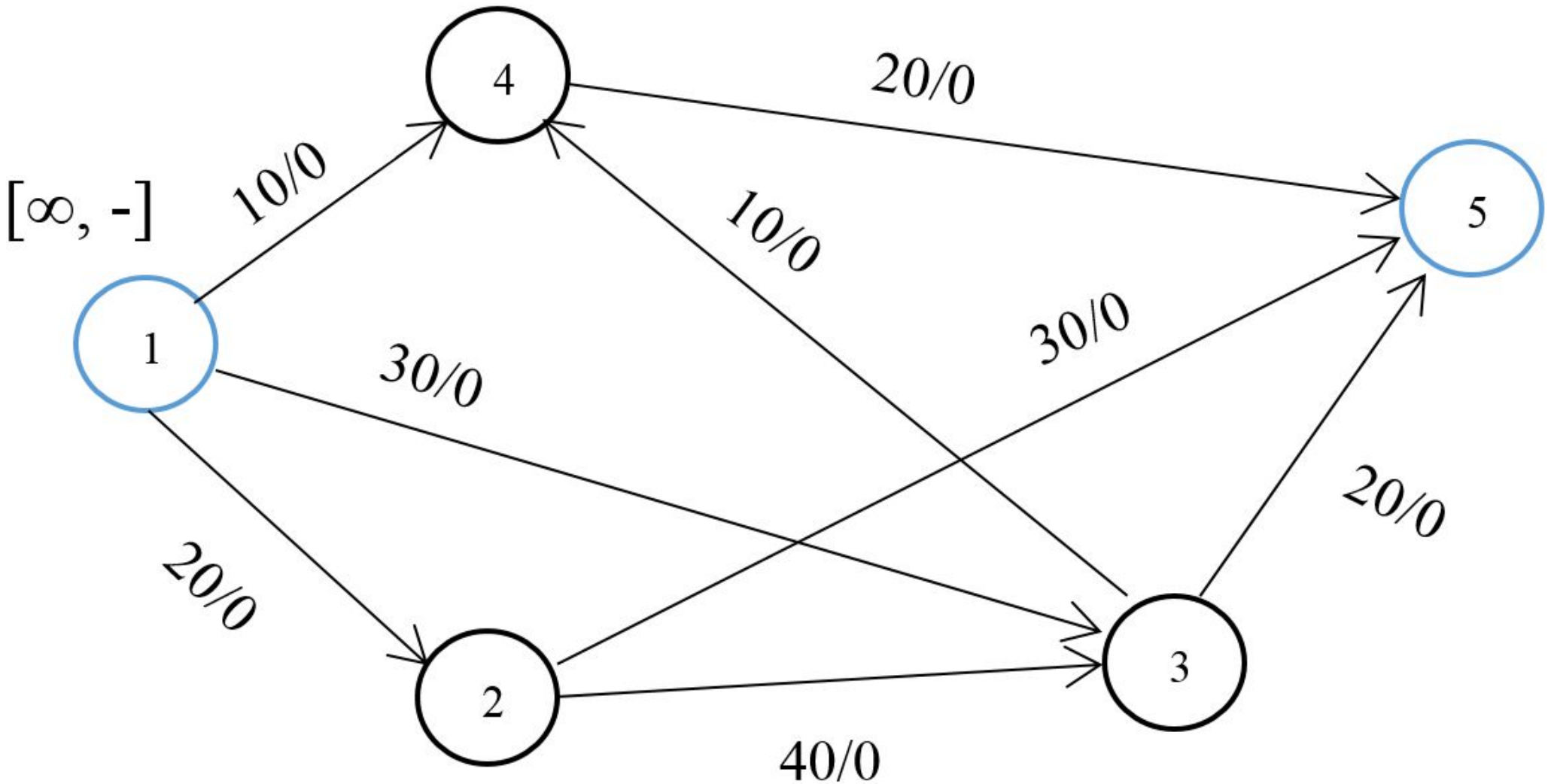
Можно попробовать искать кратчайший (по числу ребер) дополняющий путь между истоком и стоком. Например, с помощью поиска в ширину. Получающийся при этом алгоритм (реализация метода Форда – Фалкерсона) называется алгоритмом Эдмондса – Карпа).

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 $O(m^2n)$.

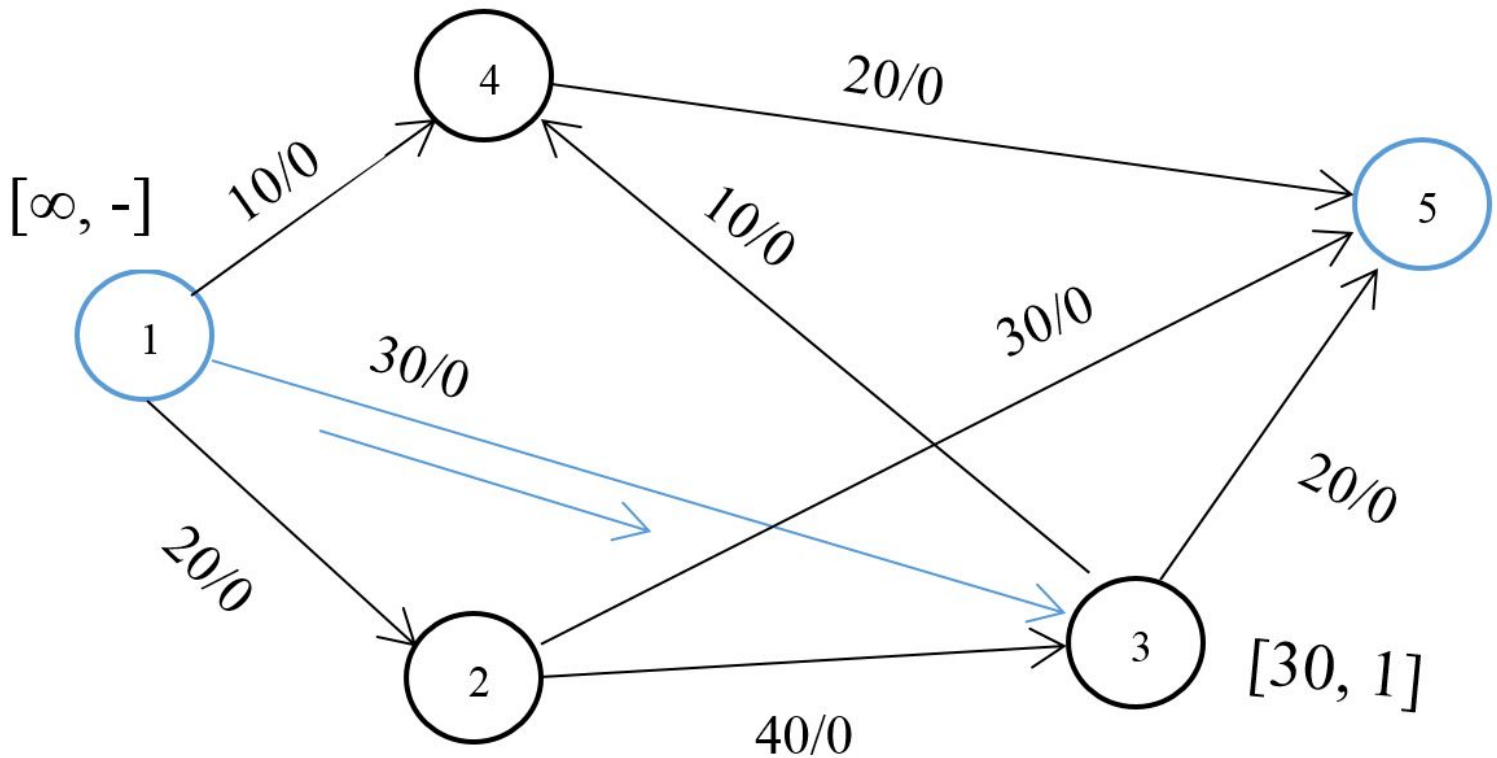
Сеть с несколькими истоками и стоками

Если в графе есть несколько истоков и/или несколько стоков, то задача нахождения максимального потока в такой сети сводится к задаче с одним истоком и одним стоком.

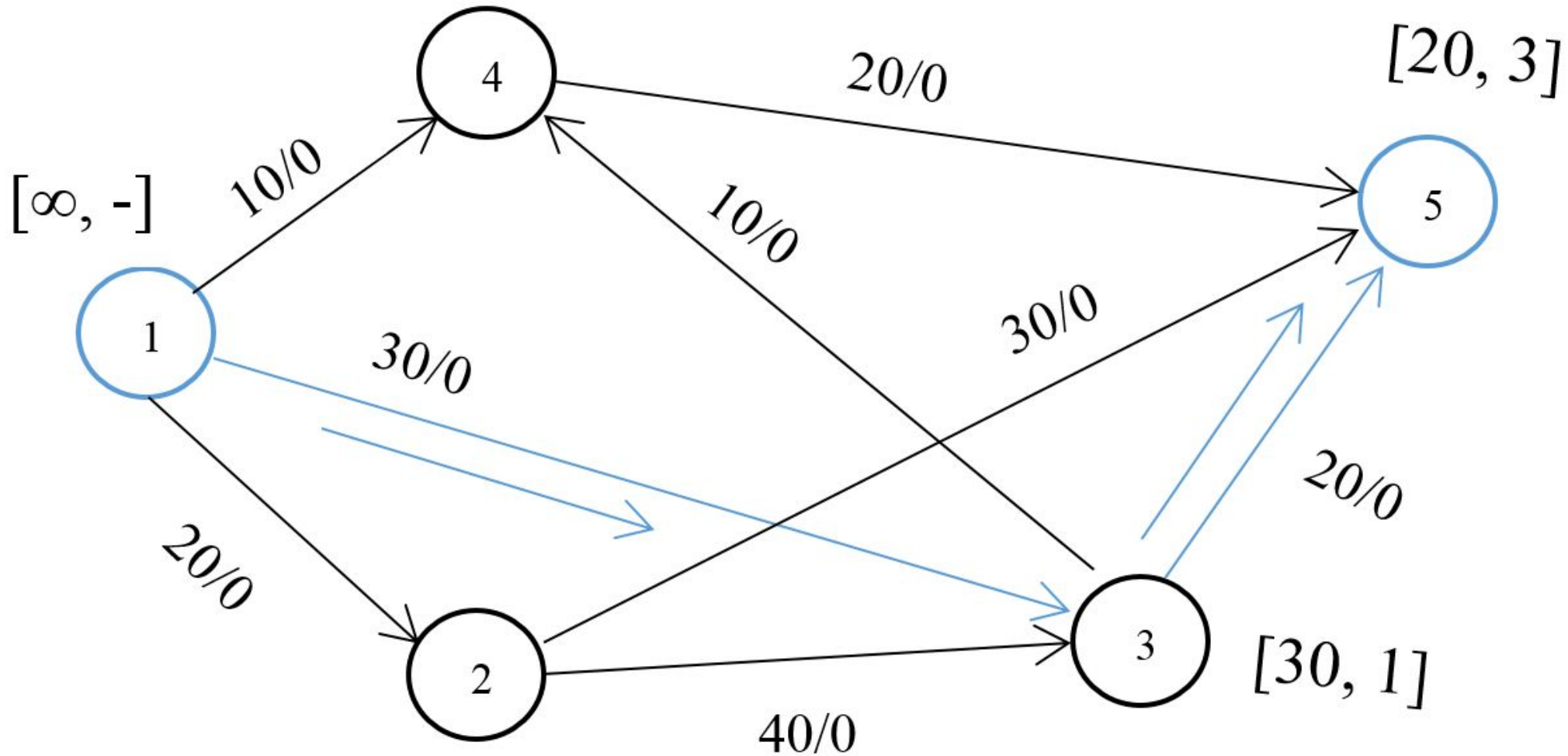



4
20/0
[∞, -]
10/0
10/0
5
1
30/0
30/0
20/0
20/0
2
3
40/0

- Шаг 1: Полагаем i = 1 и переходим ко второму шагу.
- Шаг 2: $S_1 = \{2, 3, 4\}$.
- Шаг 3: k = 3, т.к. $c_{13} = \max\{c_{12}, c_{13}, c_{14}\} = \max\{20, 30, 10\} = 30$.

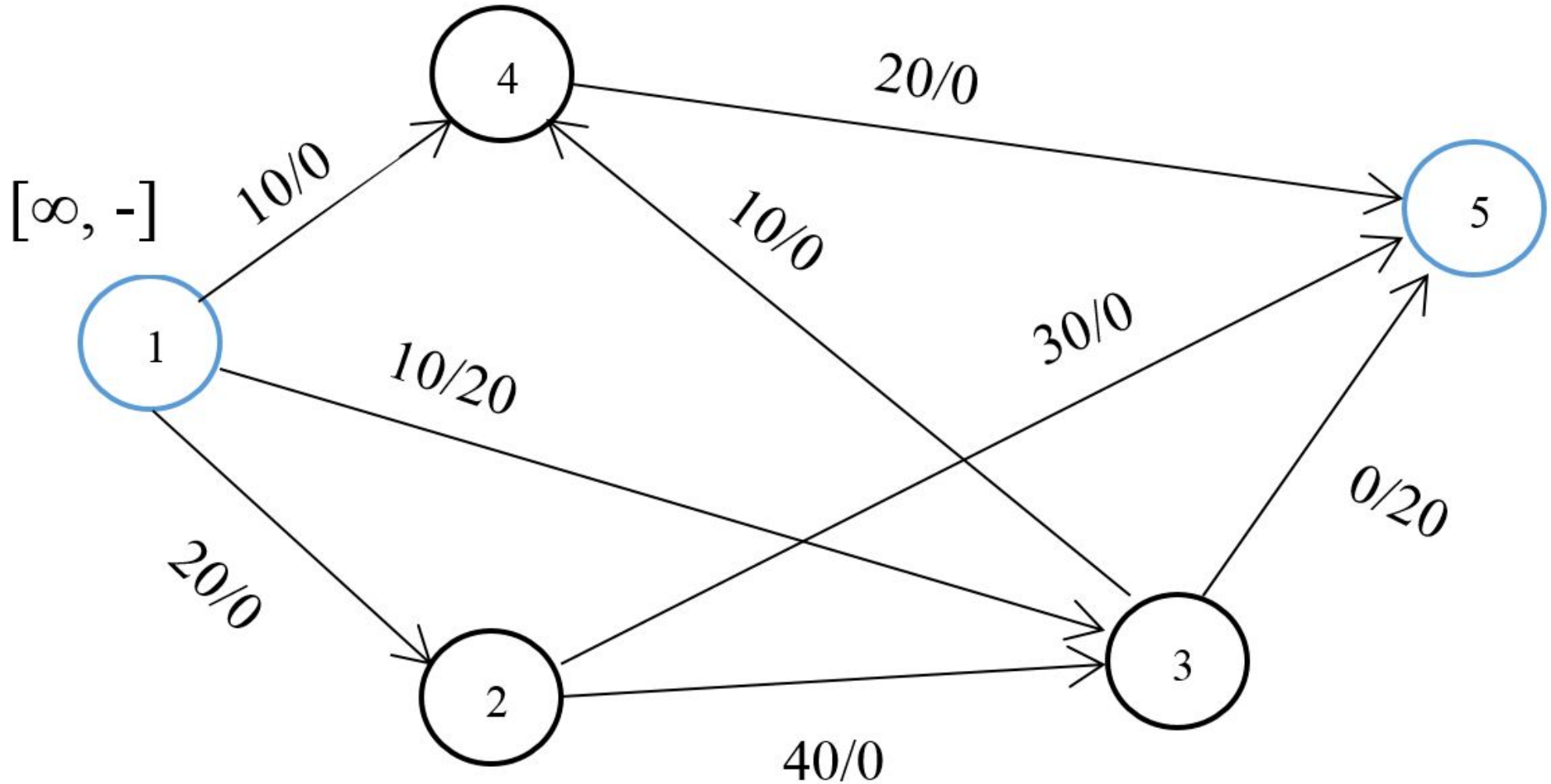


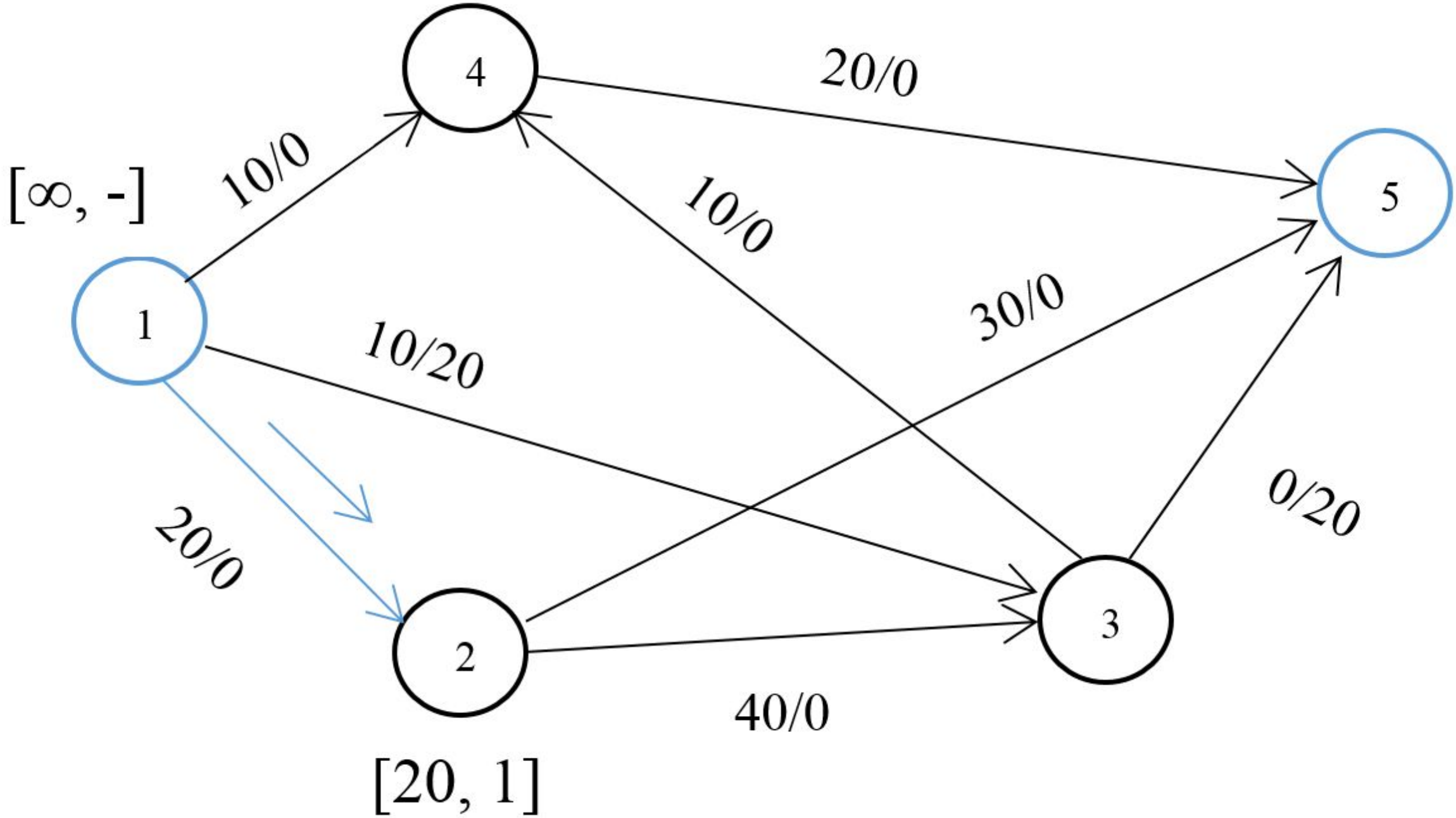
- Шаг 2: $S_3 = \{4, 5\}$.
- Шаг 3: k = 5, т.к. $c_{35} = \max\{c_{34}, c_{35}\} = \max\{10, 20\} = 20$.

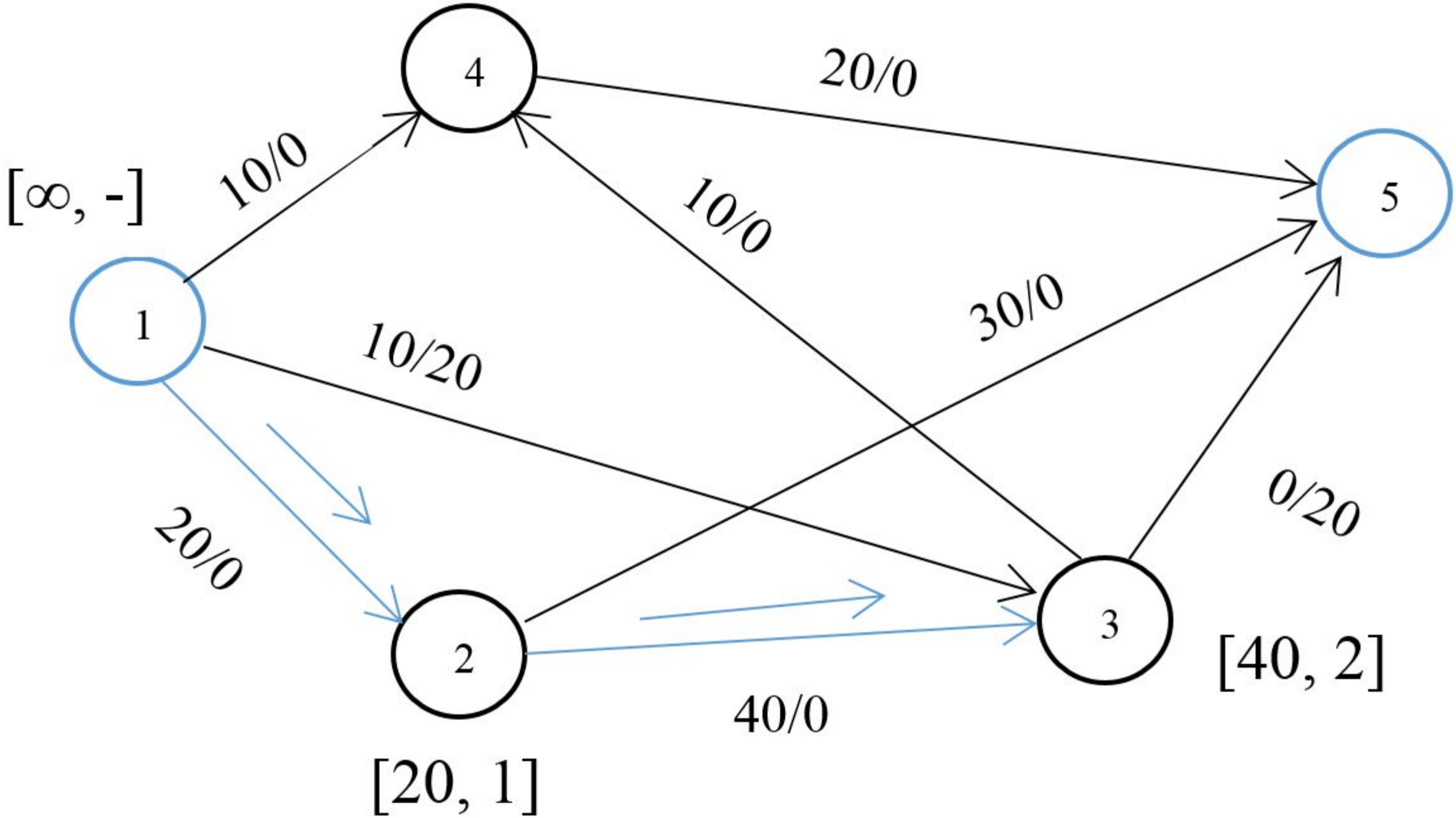


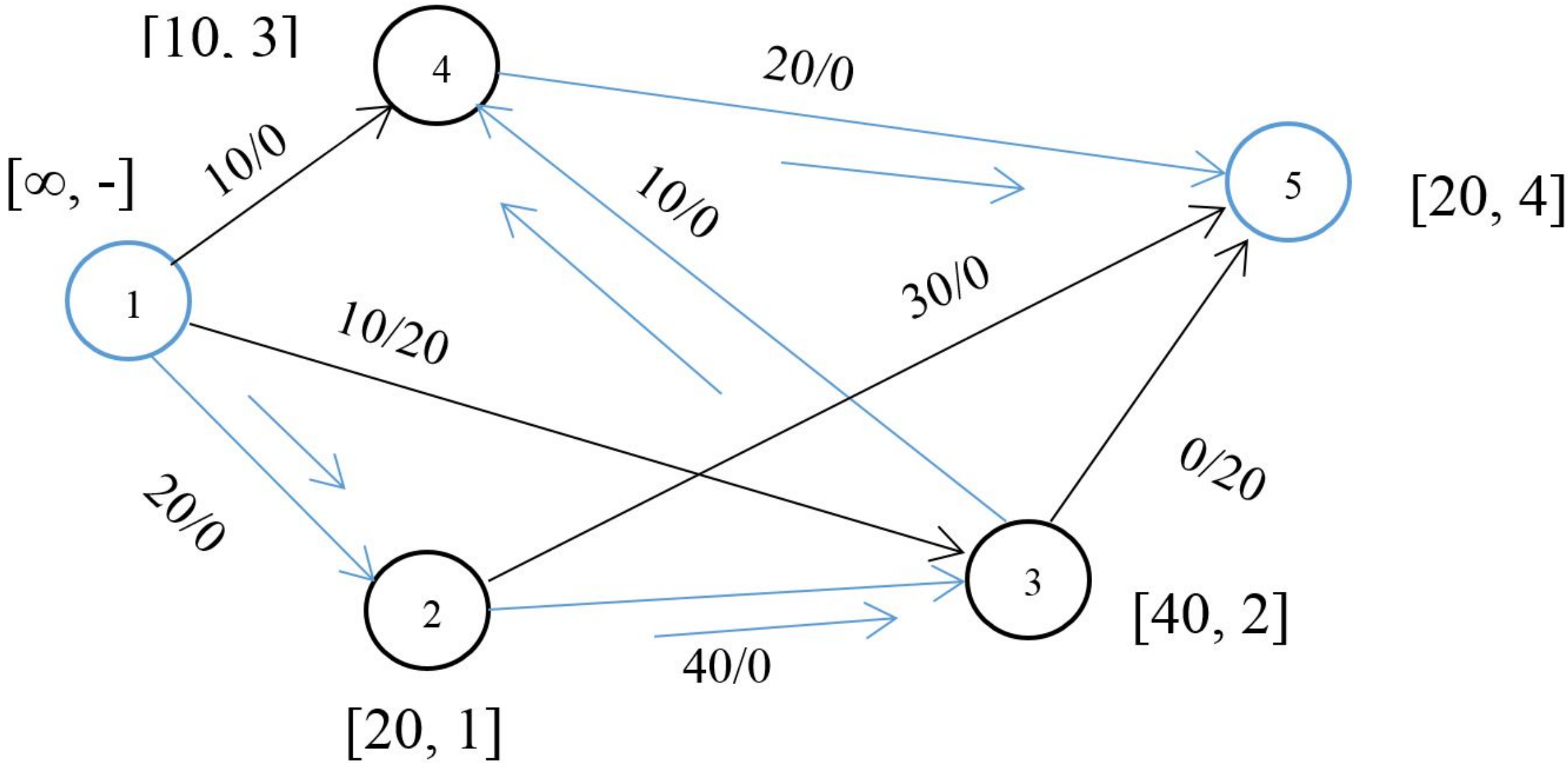
$\{c_{13}, c_{31}\} = \{30 - 20, 0 + 20\} = \{10\ ,\ 20\};$

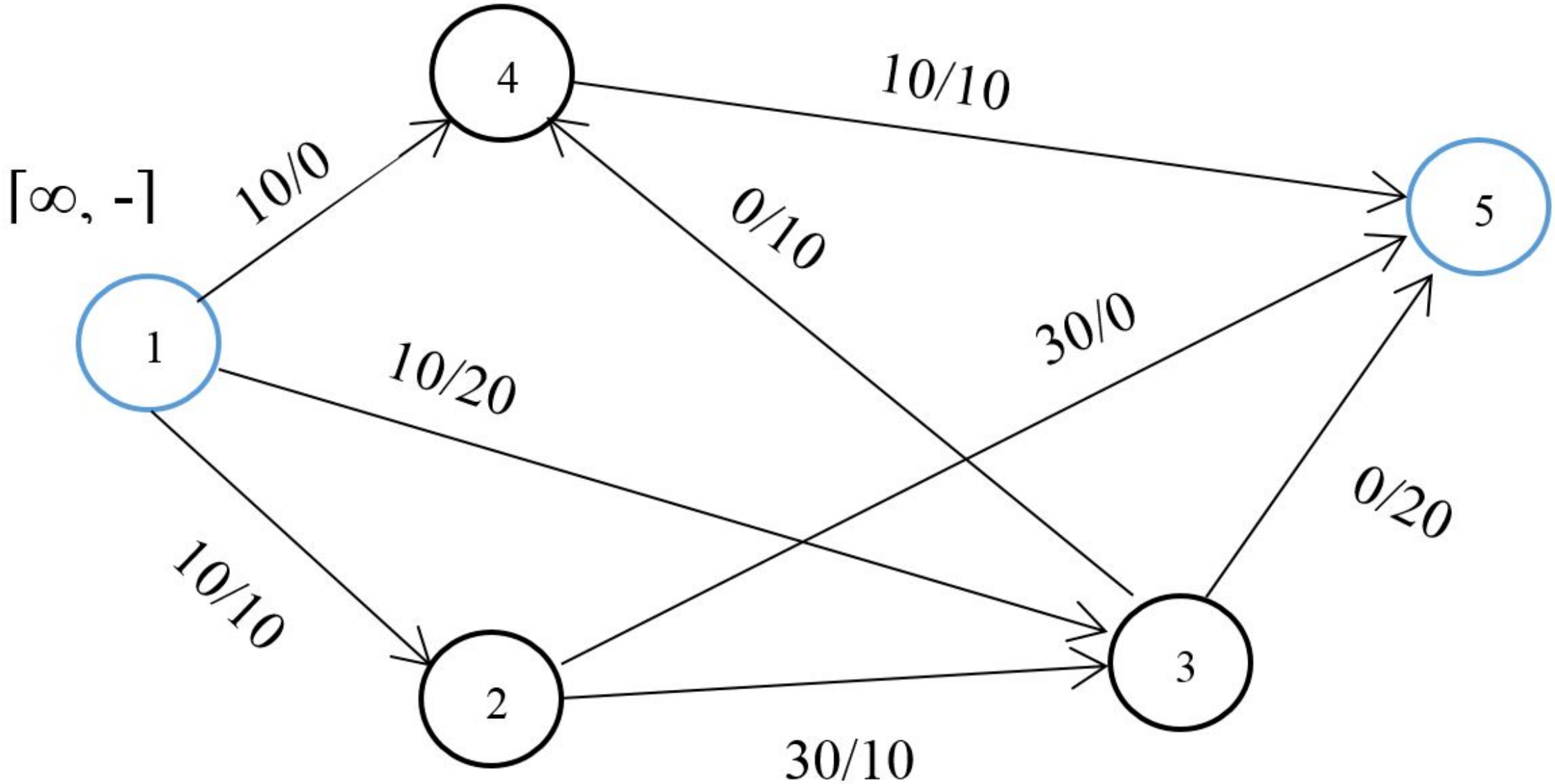
$\{c_{35}, c_{53}\} = \{20 - 20, 0 + 20\} = \{0\ ,\ 20\}.$

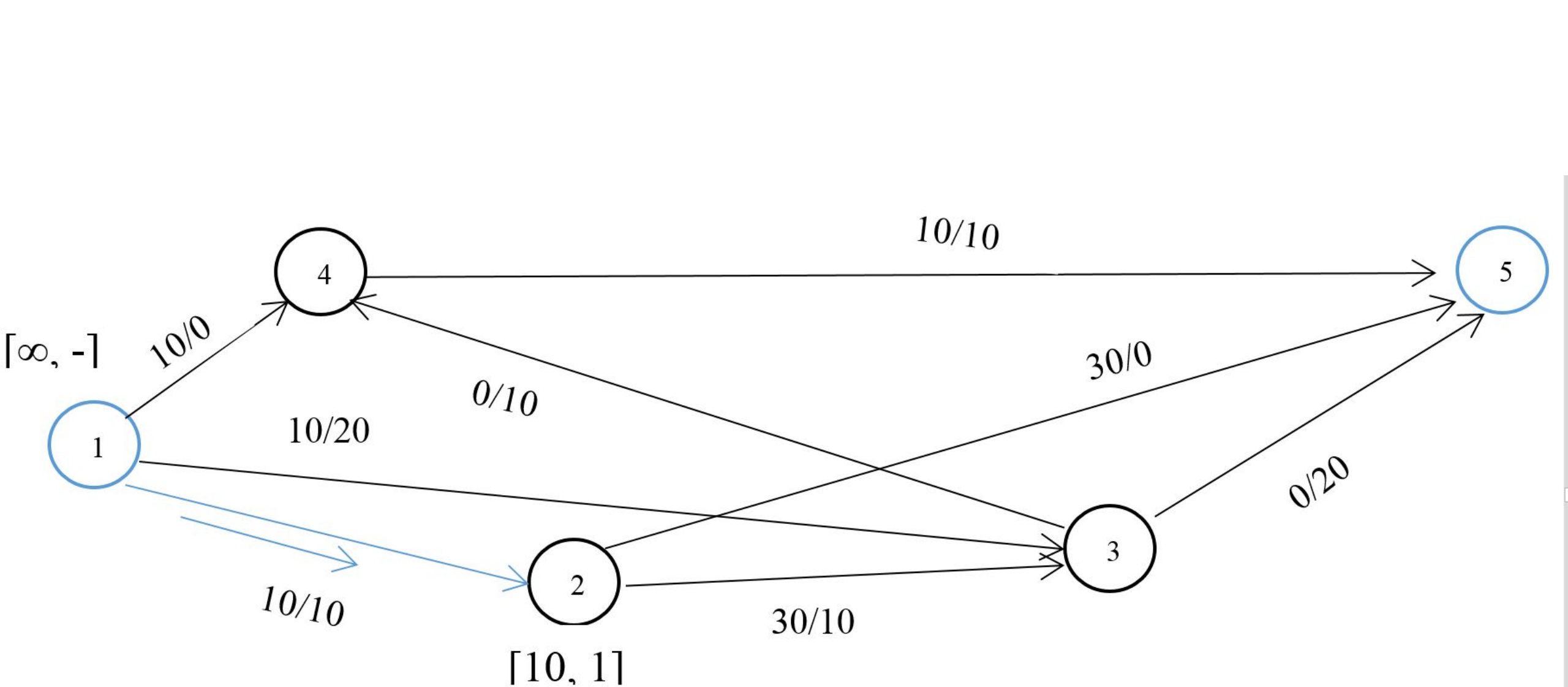


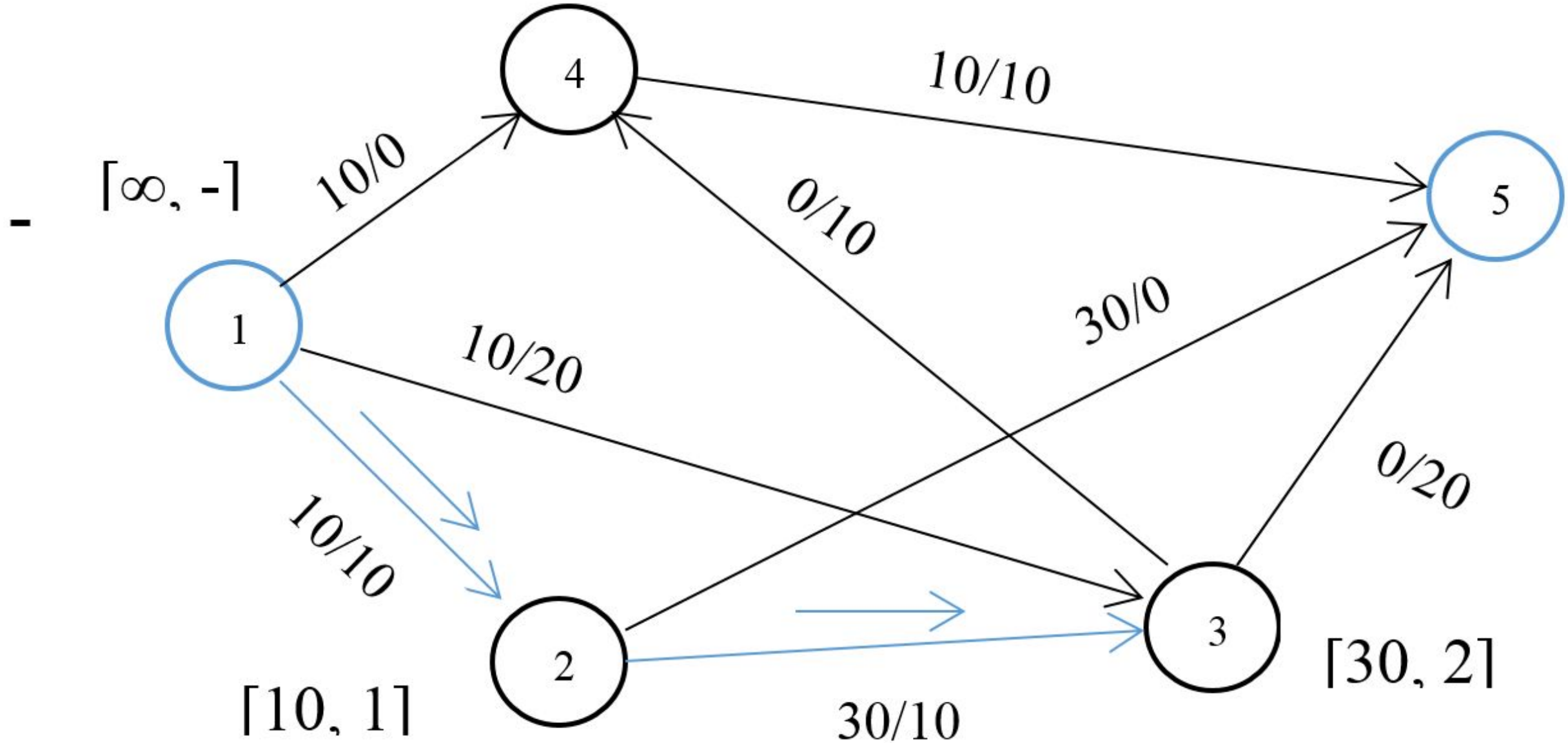

4
20/0
[∞, -]
10/0
5
10/0
1
30/0
10/20
0/20
20/0
2
3
40/0
[20, 1]

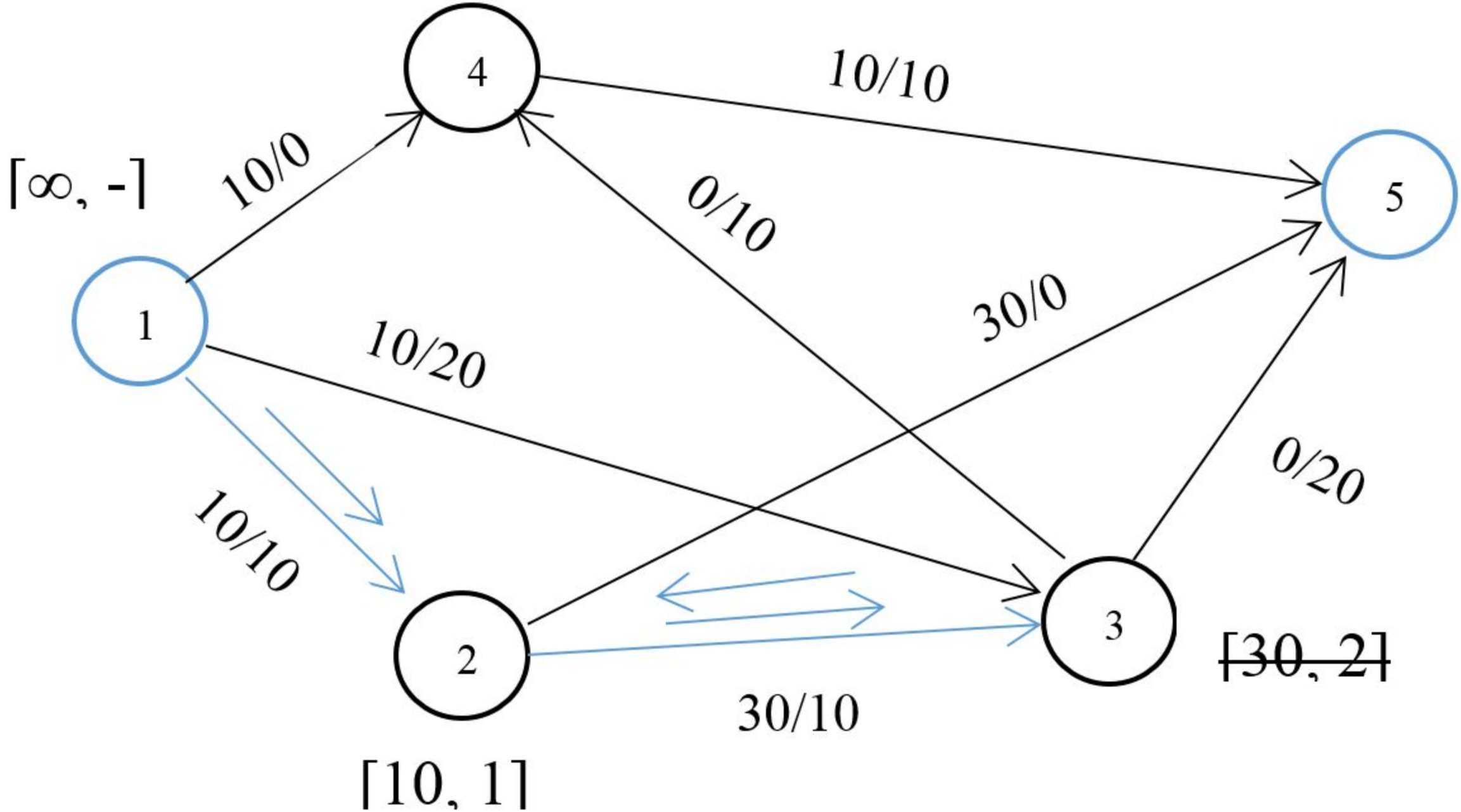

4
20/0
[∞, -]
10/0
5
10/0
1
30/0
10/20
0/20
20/0
2
3
[40, 2]
40/0
[20, 1]

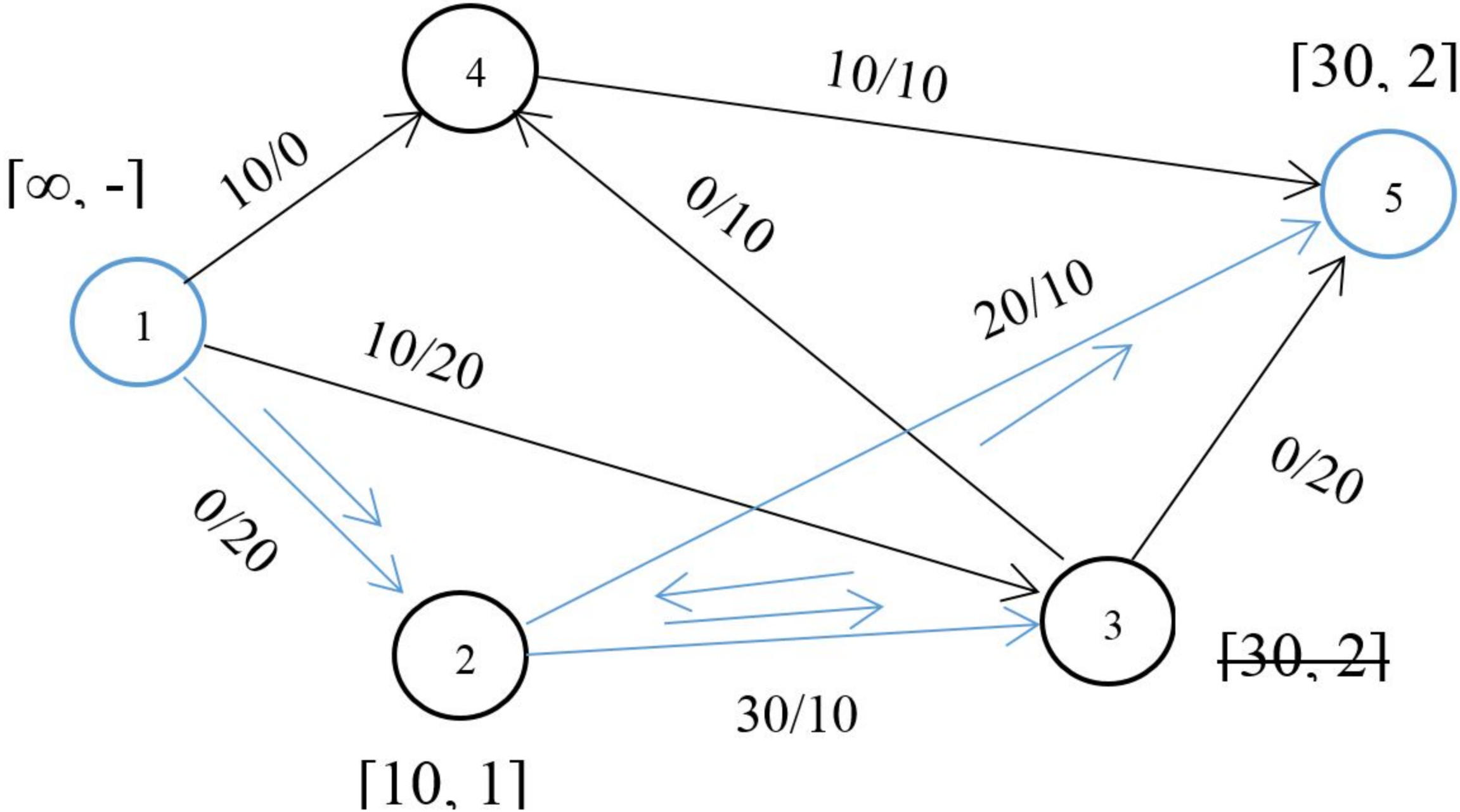

[10. 3]
4
20/0
[∞, -]
10/0
10/0
5
[20, 4]
1
30/0
10/20
0/20
20/0
3
2
[40, 2]
40/0
[20, 1]

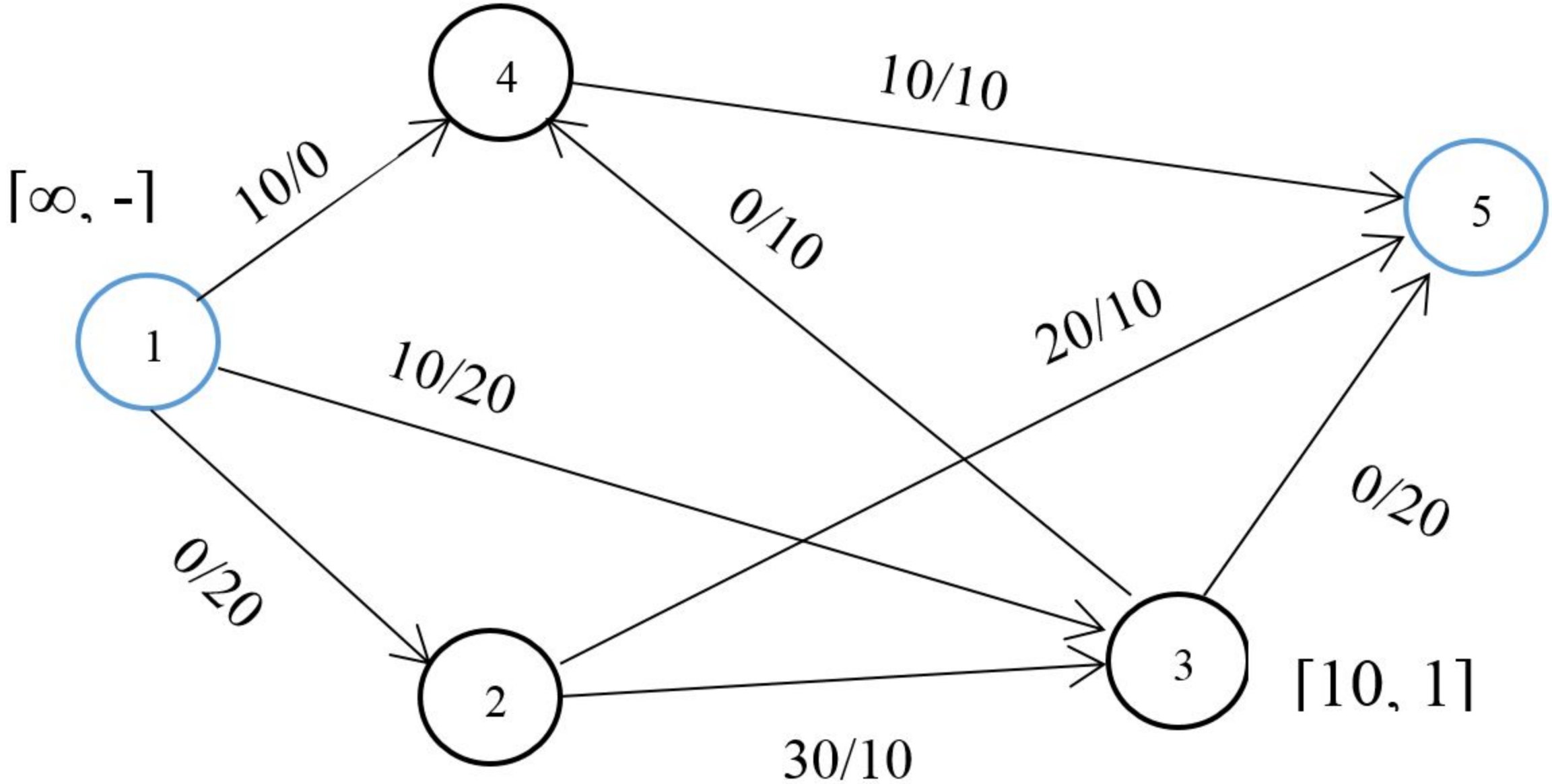

4
10/10
[∞, -]
10/0
5
0/10
1
30/0
10/20
0/20
10/10
3
2
30/10


4
10/10
5
[∞, -]
10/0
30/0
0/10
10/20
1
3
0/20
10/10
2
30/10
[10, 1]

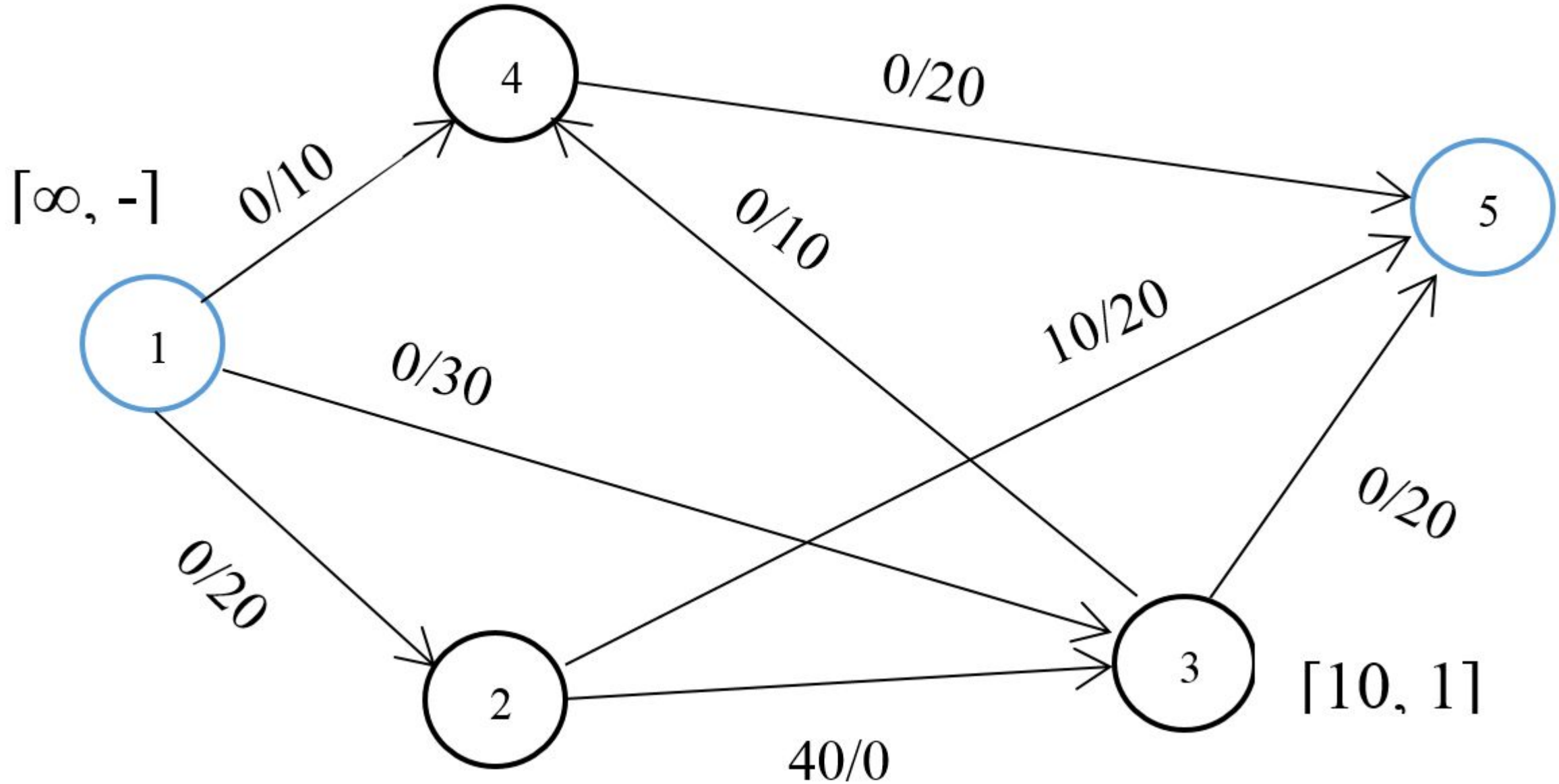

4
10/10
[∞, -]
10/0
5
0/10
1
30/0
10/20
0/20
10/10
3
2
[30, 2]
[10, 1]
30/10


4
10/10
[∞, -]
10/0
5
0/10
1
30/0
10/20
0/20
10/10
2
3
[30, 2]
30/10
[10, 1]


4
10/10
[30, 2]
[∞, -]
10/0
5
0/10
1
20/10
10/20
0/20
0/20
2
3
[30, 2]
30/10
[10, 1]


4
10/10
[∞, -]
10/0
5
0/10
1
20/10
10/20
0/20
0/20
2
3
[10, 1]
30/10

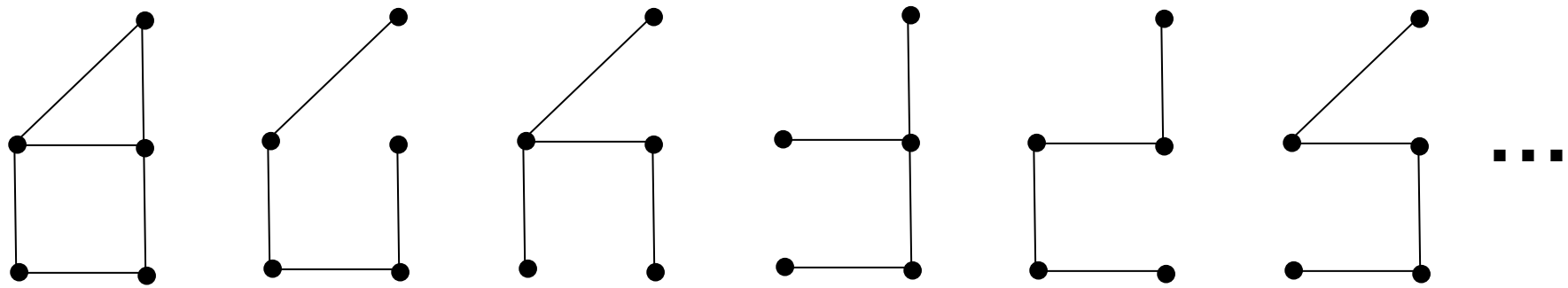
Пропустим итерацию 4 и 5



Минимальное остовное дерево (МОД)

Остовной связный подграф – подграф графа G, который содержит все его вершины и каждая вершина достижима из любой другой.

Остовное связное дерево – подграф, включающий вершины исходного графа G, не содержащего циклы, каждая вершина которого достижима из любой другой.



Цикломатическое число **γ** показывает, сколько ребер нужно удалить из графа, чтобы в нем не осталось циклов

$$\gamma = m - n + 1,$$

m - количество ребер

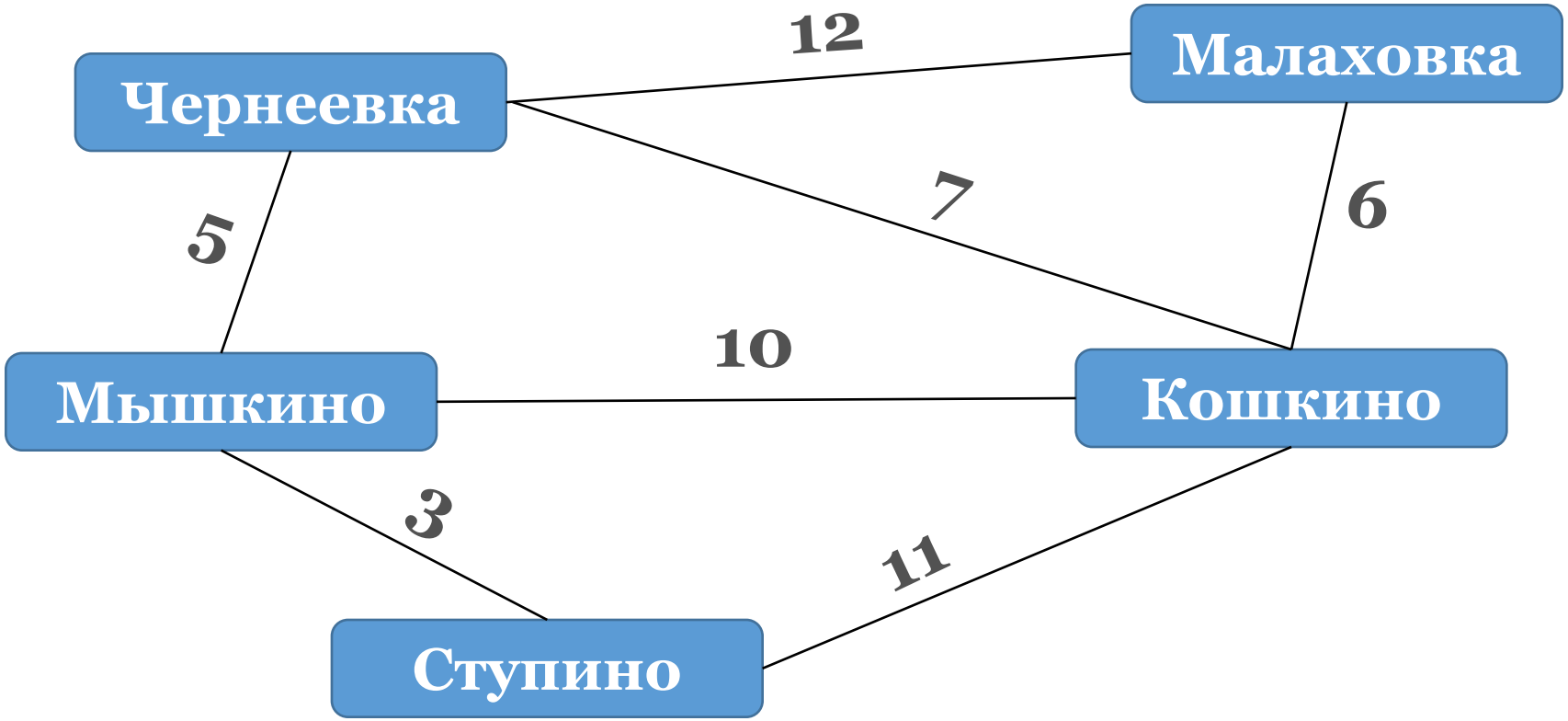
n - количество вершин

Задача 1

В некотором районе было решено провести газопровод между пятью деревнями. От Кошкино до Мышкино идти 10 км, от Мышкино до Ступино – 3 км, от Кошкино до Малаховки – 6 км, от Малаховки до Черняевки – 12 км, от Кошкино до Ступино – 11 км, от Мышкино до Чернеевки – 5 км, от Кошкино до Чернеевки – 7 км. Как необходимо провести трубу, чтобы она соединяла все пять деревень, и затраты при этом были минимальными?

Задача 1

Построим граф, моделирующий дороги, соединяющие деревни.



Задача 1

Задача сводится к построению остовного связного дерева минимального веса.

Рассчитаем цикломатическое число.

m (количество ребер) равно **7**

n (количество вершин) рано **5**

$$\gamma = 7 - 5 + 1 = 3$$

Т.е. три деревни напрямую соединены газовой трубой не будут.

Алгоритм Прима

- **Начало алгоритма: с произвольной вершины**
- **К текущему дереву присоединяется смежная вершина с кратчайшим ребром.**
- **Окончание алгоритма: либо все вершины подключены, либо невозможно подключить ни одно ребро.**

Алгоритм Прима

Пусть дан взвешенный неориентированный граф.

Для построения минимального остовного дерева необходимо:

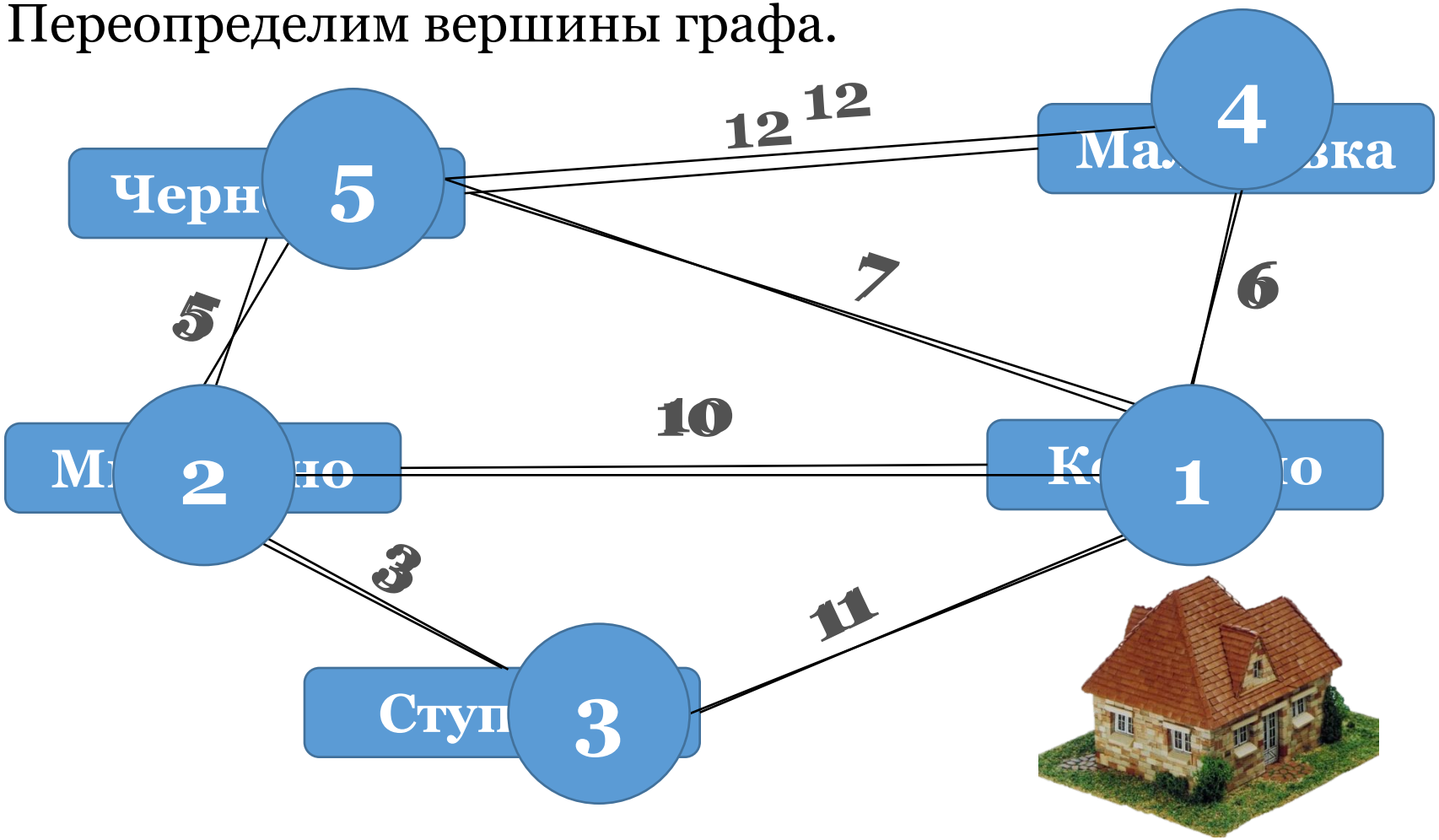
1. Представить граф в виде матрицы смежности
2. Найти в матрице наименьший элемент, соответствующий ребру, соединяющему i-ю и j-ю вершины графа
3. Вычеркнуть элементы i-й и j-й строки матрицы
4. Пометить i-й и j-й столбцы матрицы
5. В помеченных столбцах i и j найти наименьший элемент, отличный от уже найденного
6. Повторять пункты 3-5 до тех пор, пока не будут задействованы все вершины графа

(переходы по щелчку)

Задача 1

Решим задачу по алгоритму Прима.

Переопределим вершины графа.



Задача 1

Представим граф в виде матрицы смежности.

	1	2	3	4	5
1	0	10	11	6	7
2	10	0	3	0	5
3	11	3	0	0	0
4	6	0	0	0	12
5	7	5	0	12	0

Найдем минимальный элемент.

Он равен **3**

Задача 1

Вычеркнем 2-ю и 3-ю строки таблицы. А столбцы 2 и 3 выделим.

	1	2	3	4	5
1	0	10	11	6	7
2			**3**		
3					
4	6	0	0	0	12
5	7	**5**	0	12	0

Найдем минимальный элемент в выделенных столбцах.

Он равен **5**

Задача 1

Вычеркнем 5-ю строку таблицы. А столбец 5 выделим.

	1	2	3	4	5
1	0	10	11	6	7
2			3		
3					
4	6	0	0	0	12
5		5			

Найдем минимальный элемент в выделенных столбцах.

Он равен 7

Задача 1

Вычеркнем 1-ю строку таблицы. А столбец 1 выделим.

	1	2	3	4	5
1					7
2			3		
3					
4	6	0	0	0	12
5		5			

Найдем минимальный элемент в выделенных столбцах.

Он равен **6**

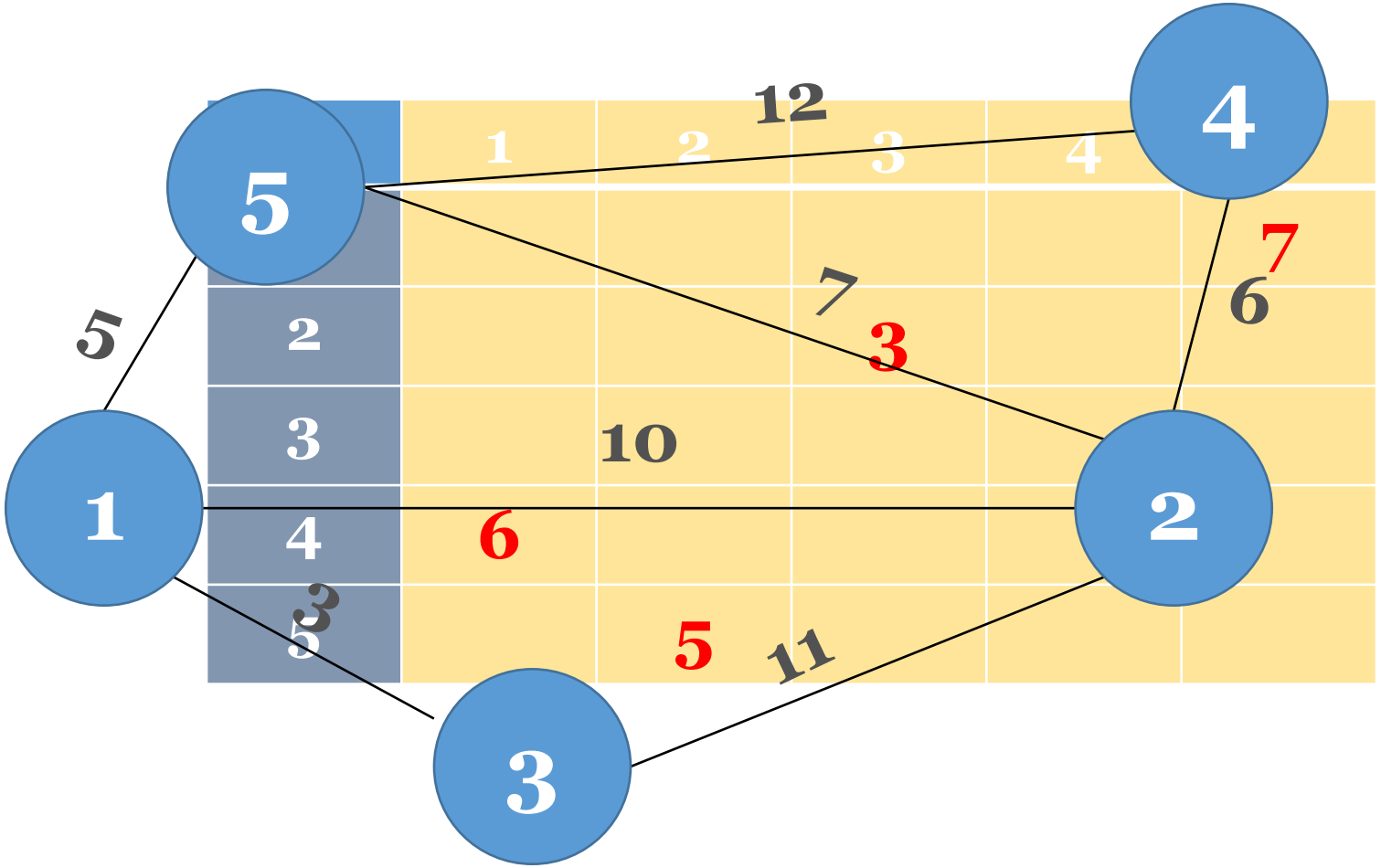
Задача 1

Вычеркнем 4-ю строку таблицы. А столбец 4 выделим.

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

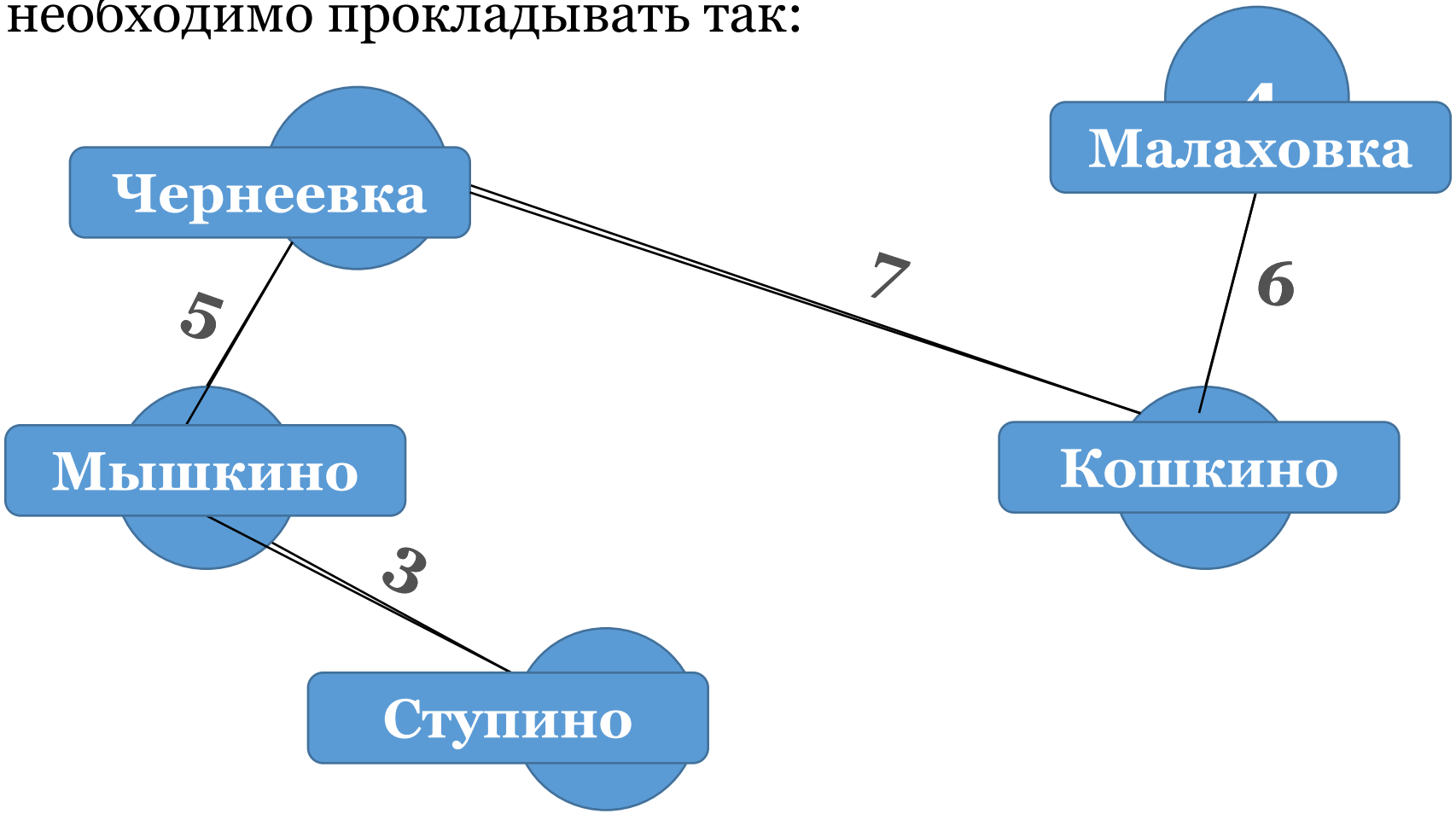
Задача 1

Получаем связное остовное дерево минимальное веса.



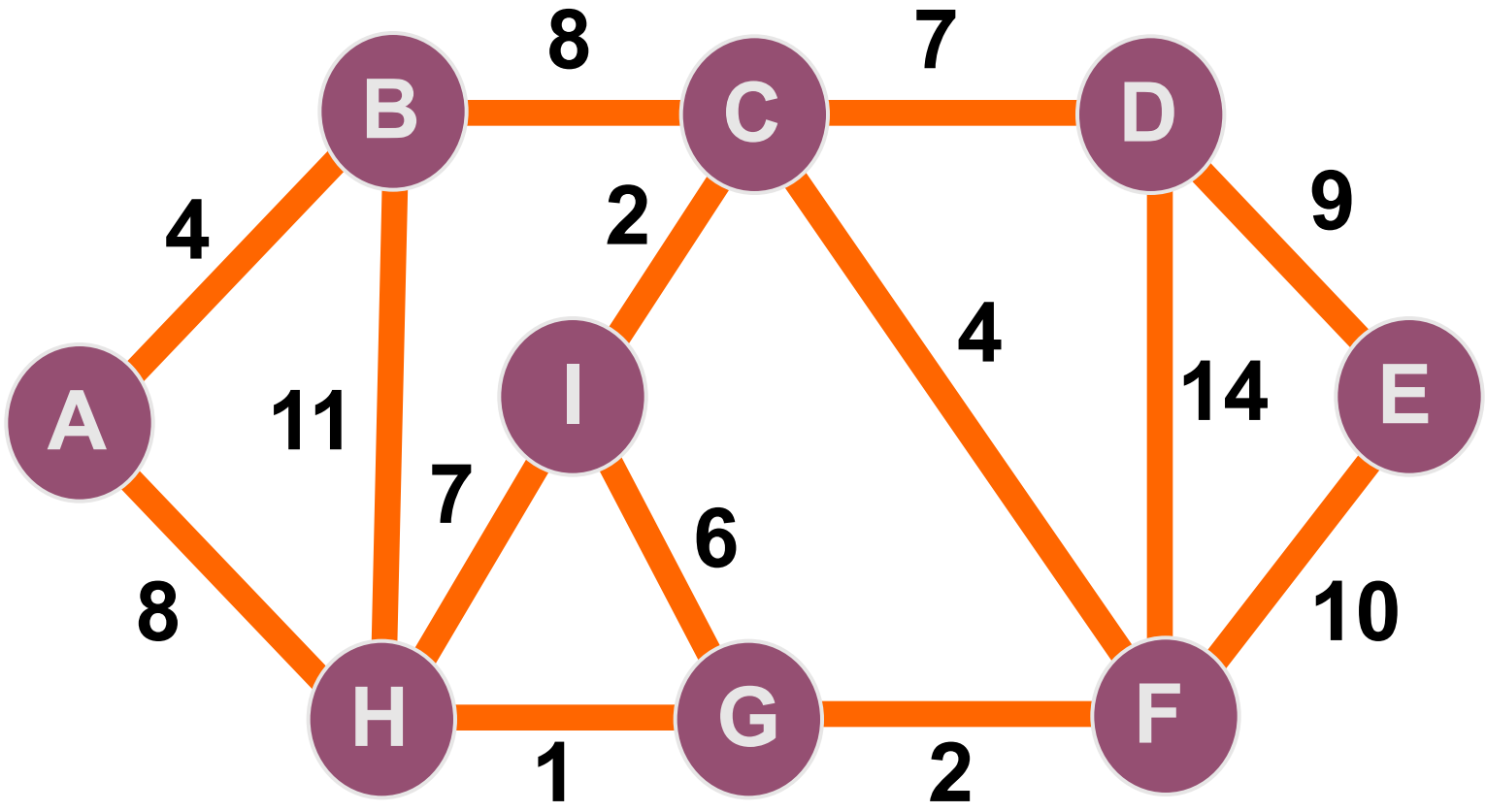
Задача 1

Ответ: газопровод с минимальными затратами необходимо прокладывать так:



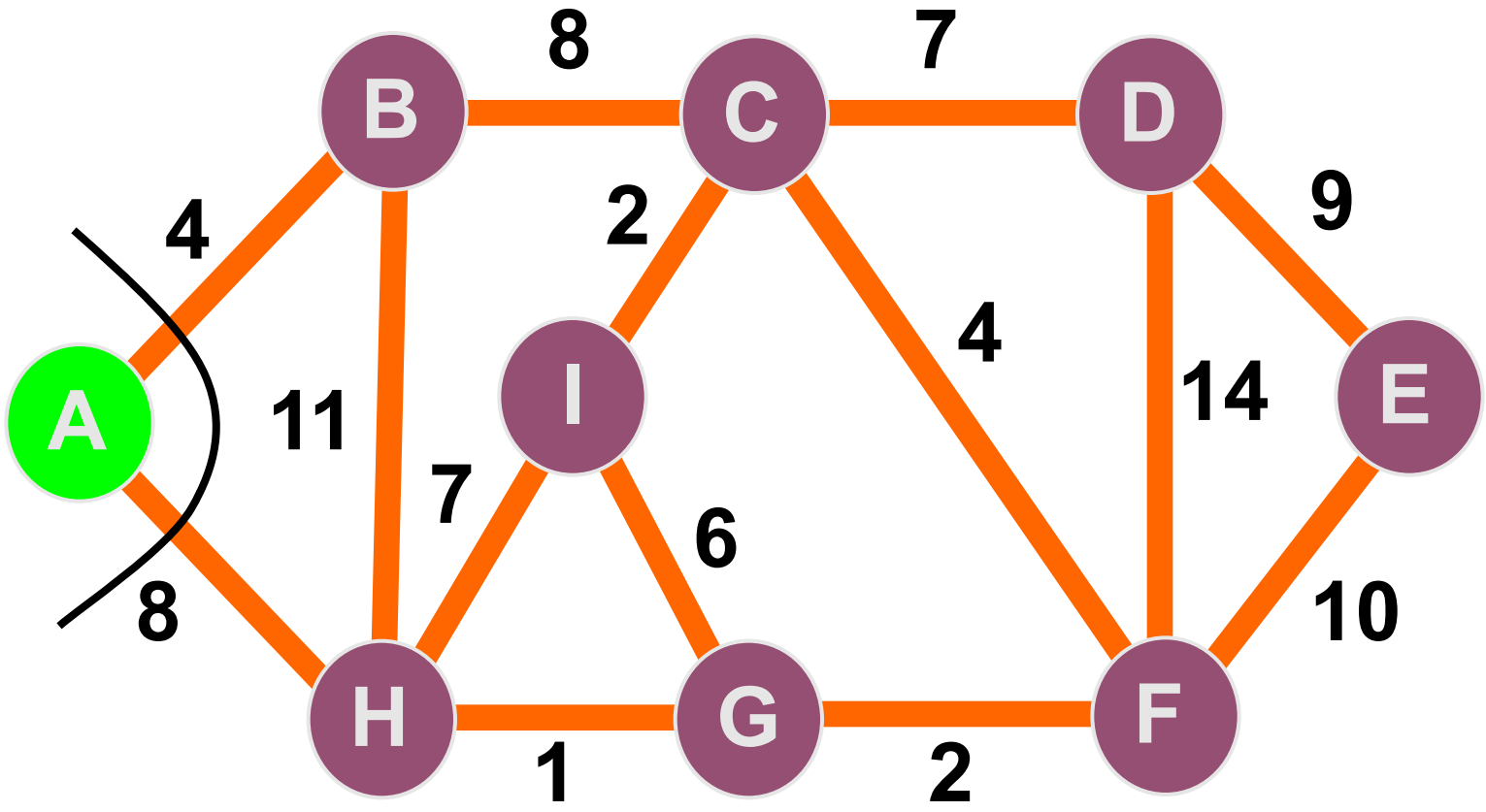
Протяженность газопровода – **21 км**.

Алгоритм Прима шаг 0



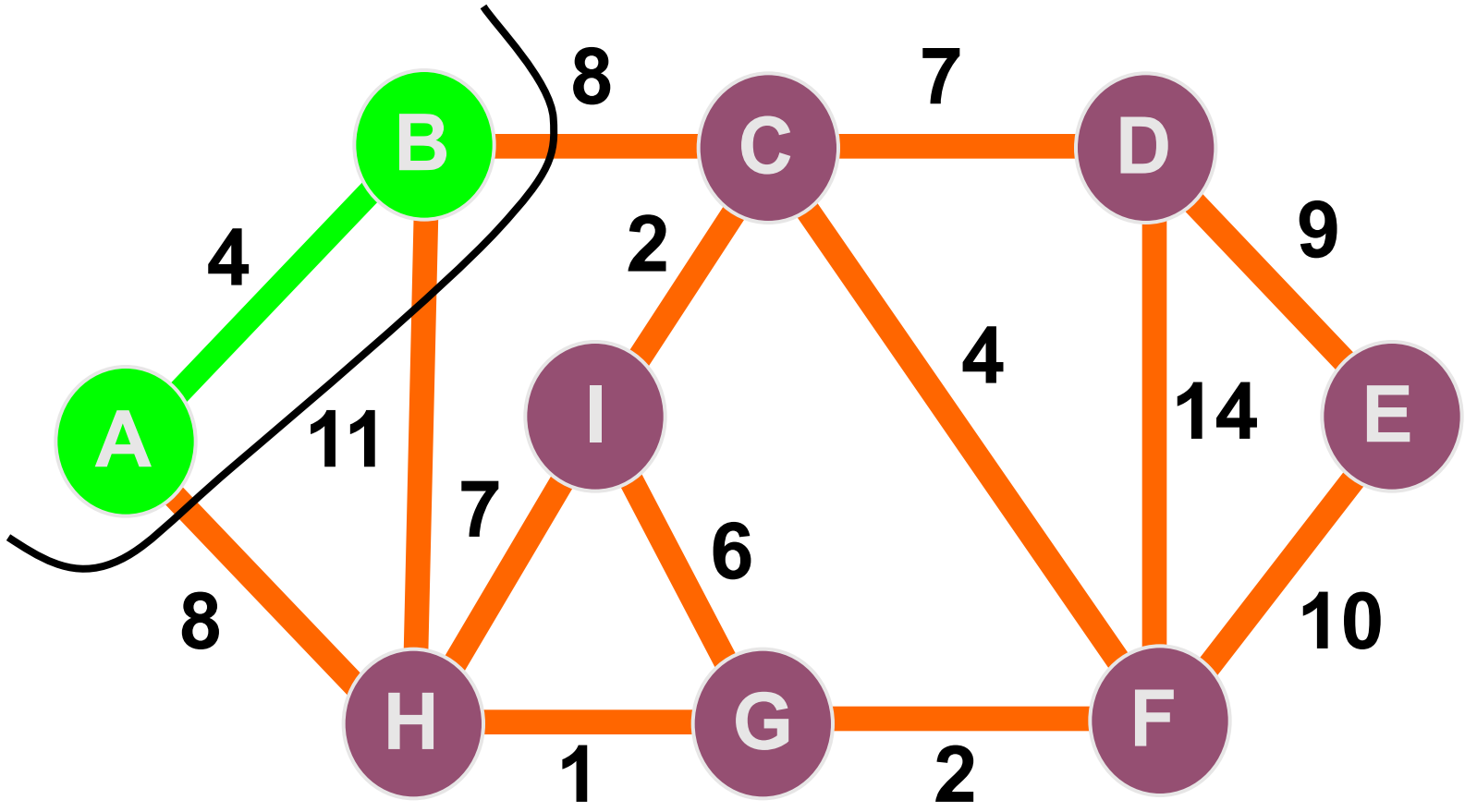
- **Суммарная длина дерева = 0**

Алгоритм Прима шаг 1



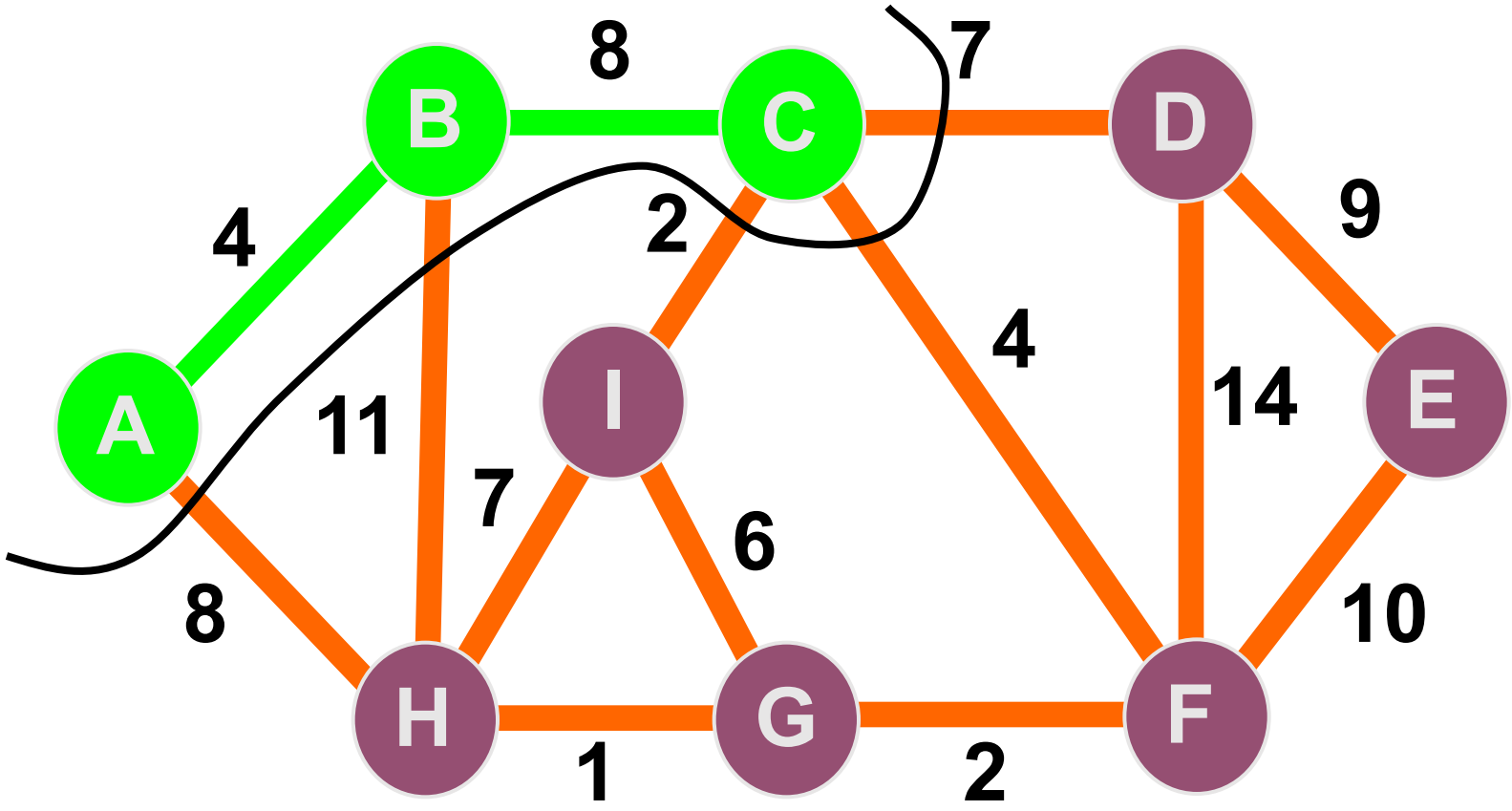
- **Суммарная длина дерева = 0**

Алгоритм Прима шаг 2



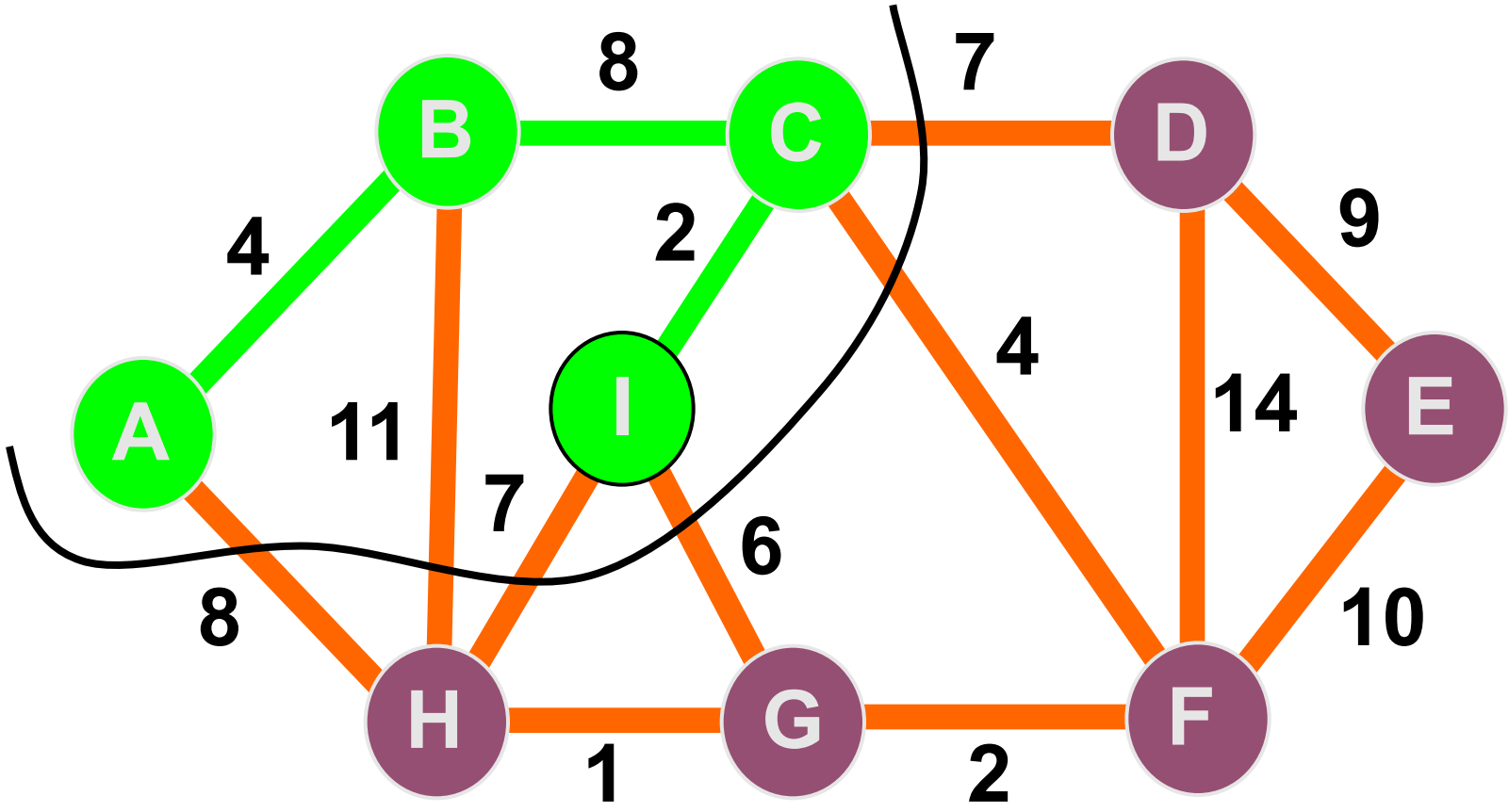
- **Суммарная длина дерева = 4**

Алгоритм Прима шаг 3



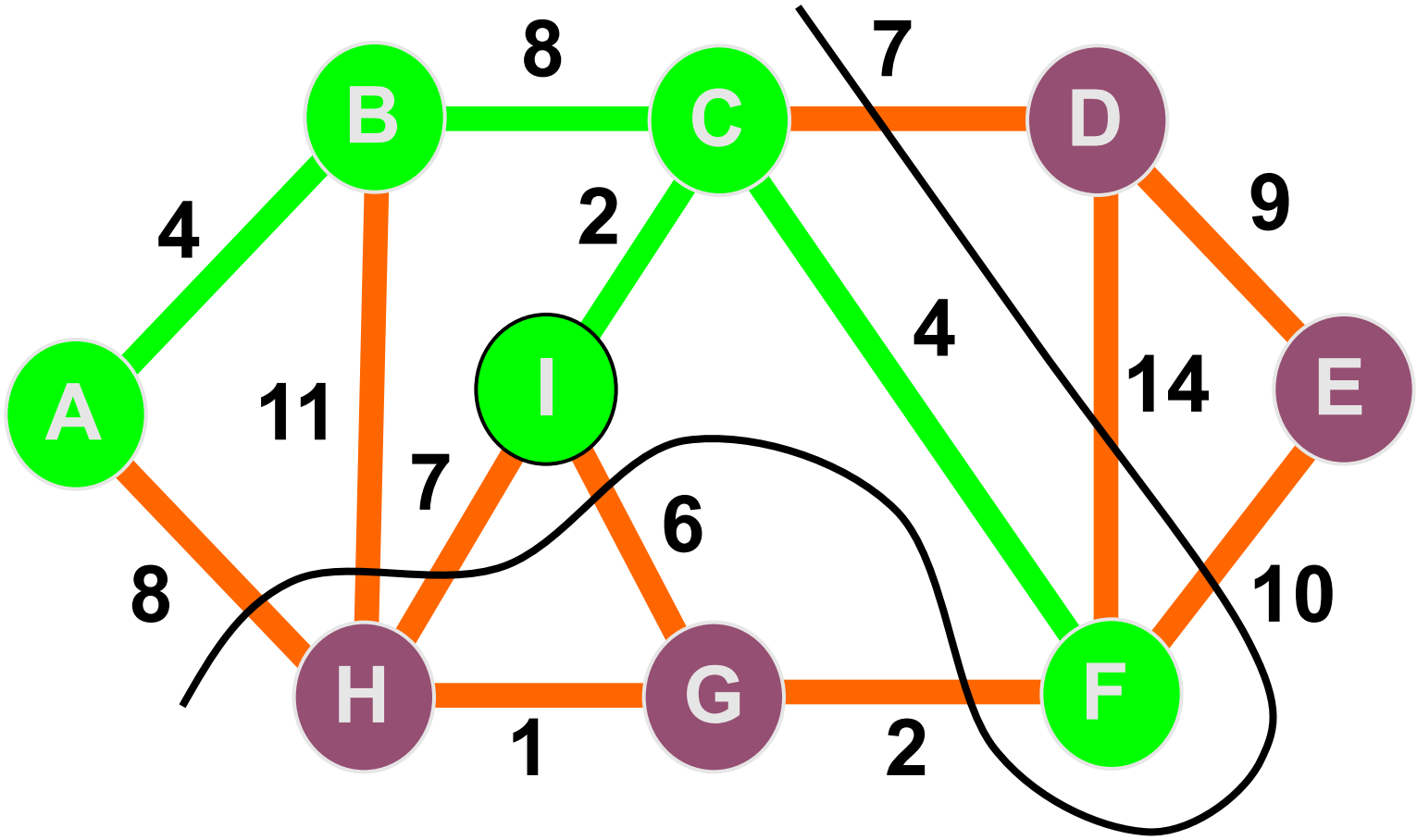
- **Суммарная длина дерева = 12**

Алгоритм Прима шаг 4



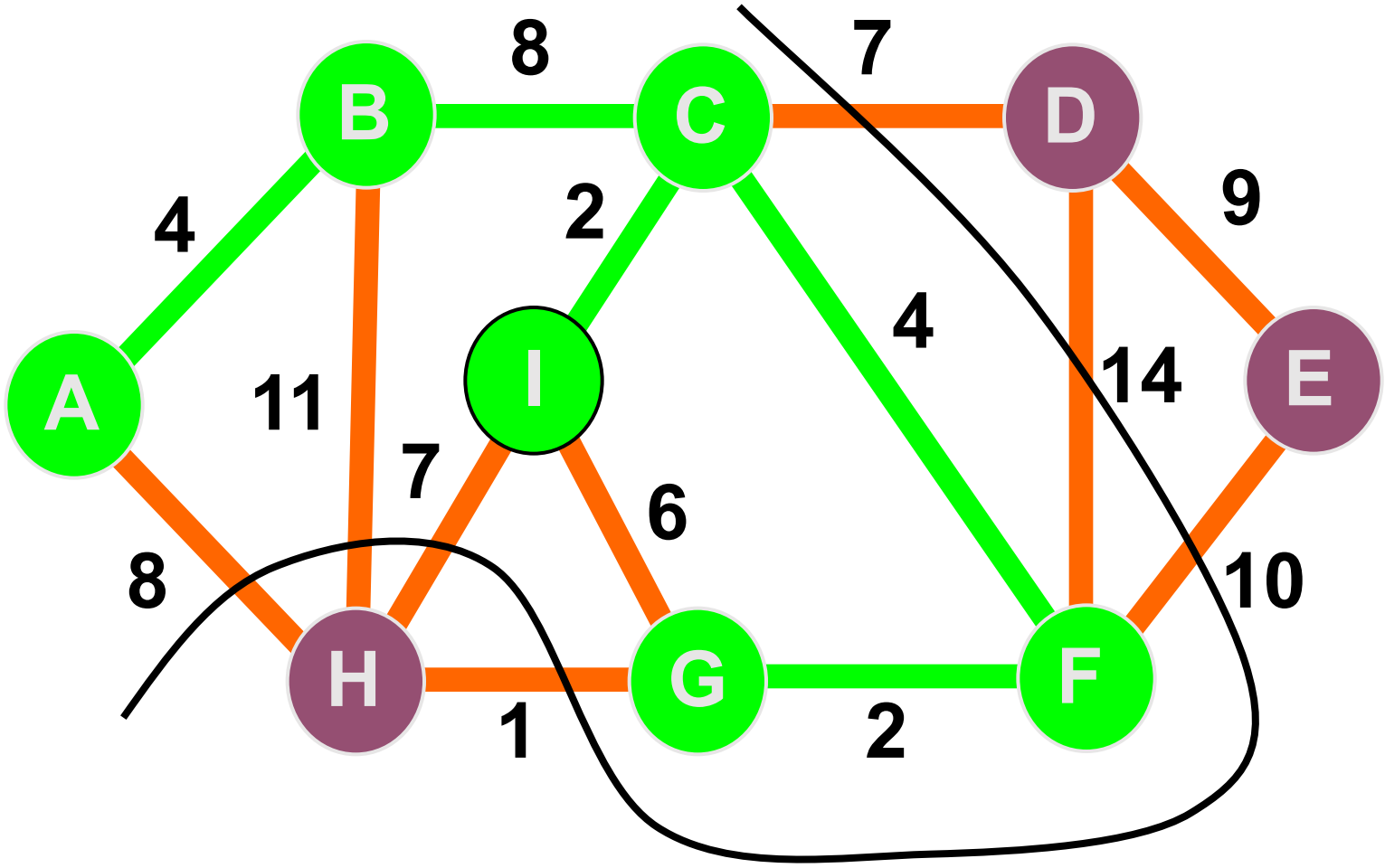
- **Суммарная длина дерева = 14**

Алгоритм Прима шаг 5



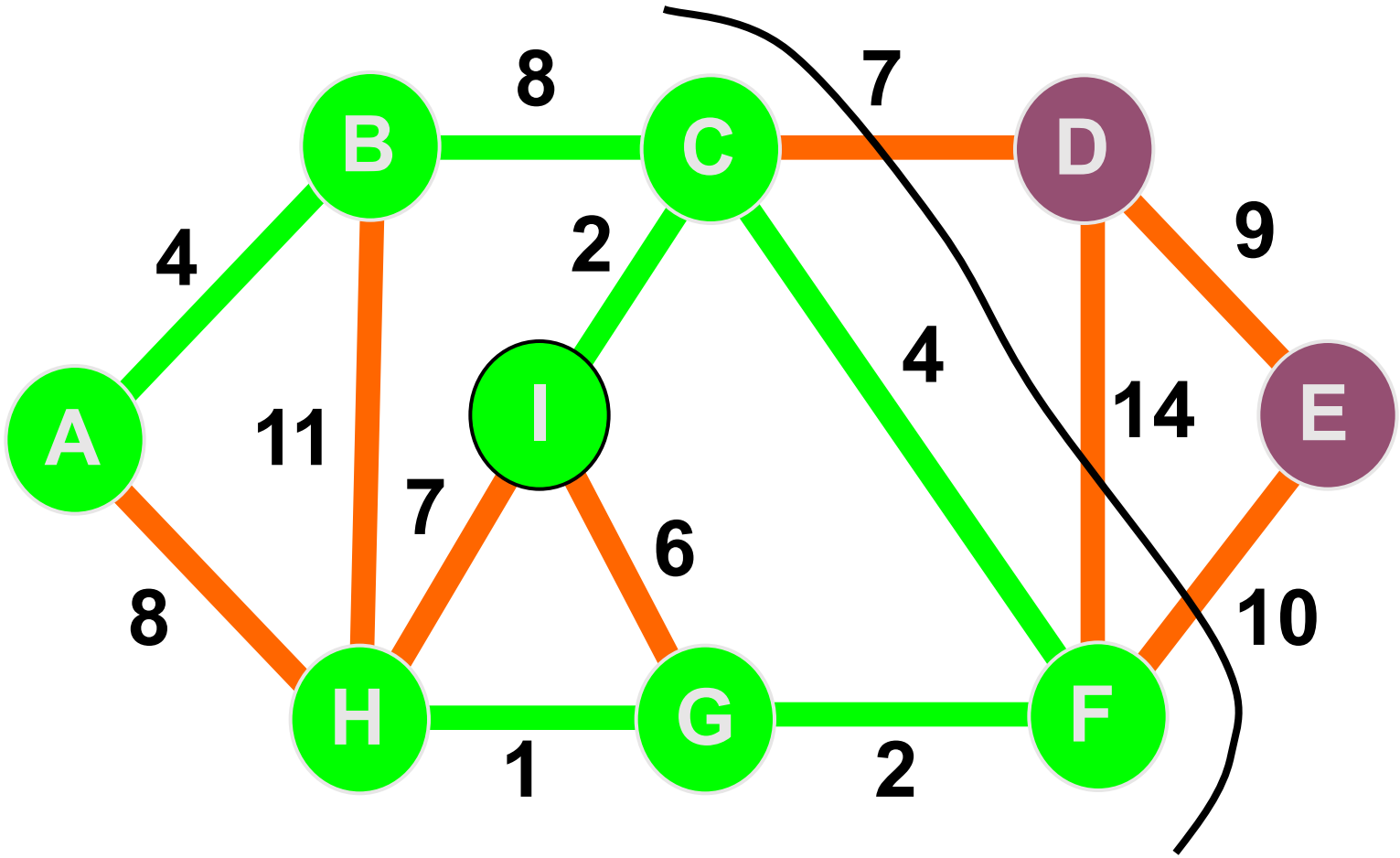
- **Суммарная длина дерева = 18**

Алгоритм Прима шаг 6



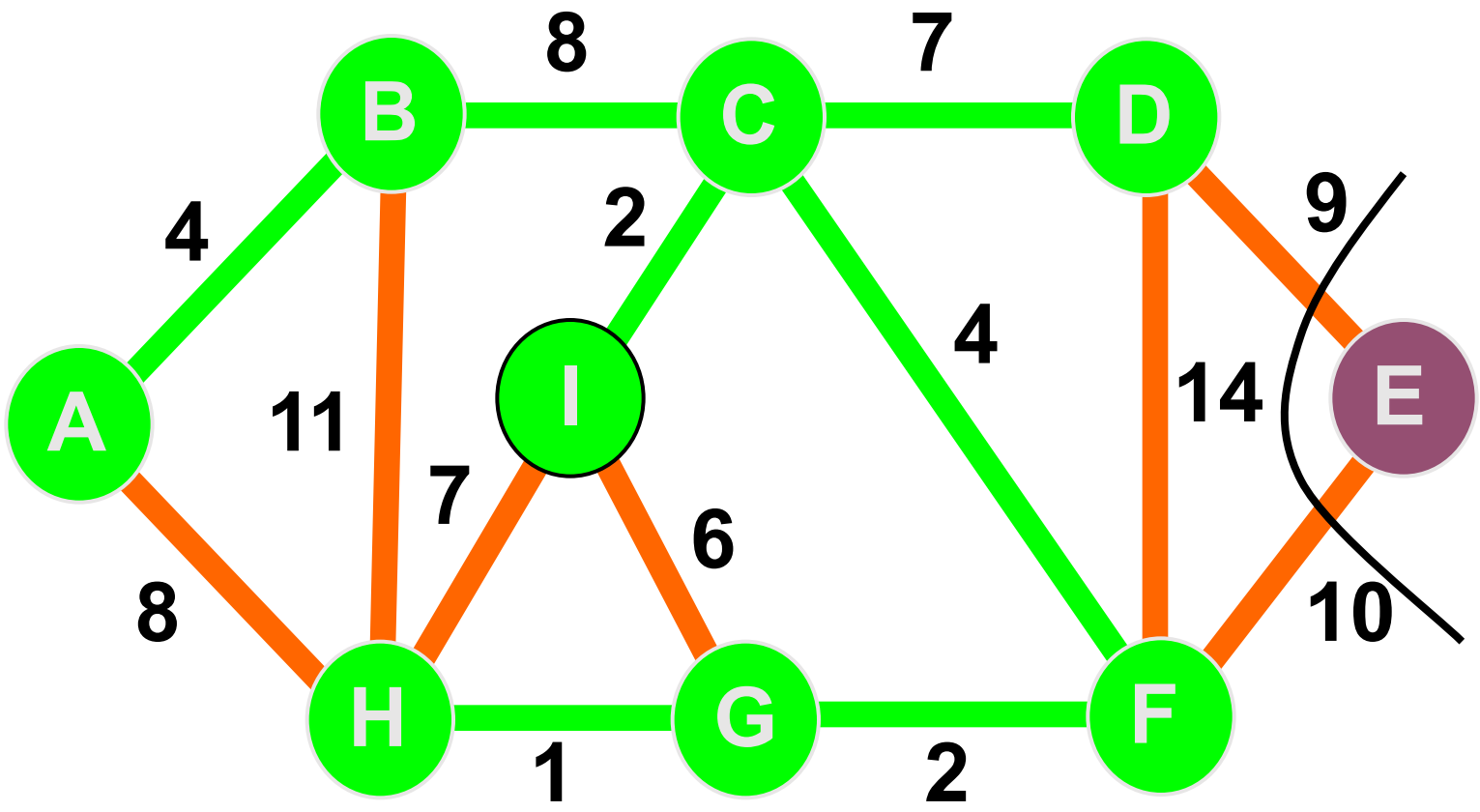
- **Суммарная длина дерева = 20**

Алгоритм Прима шаг 7



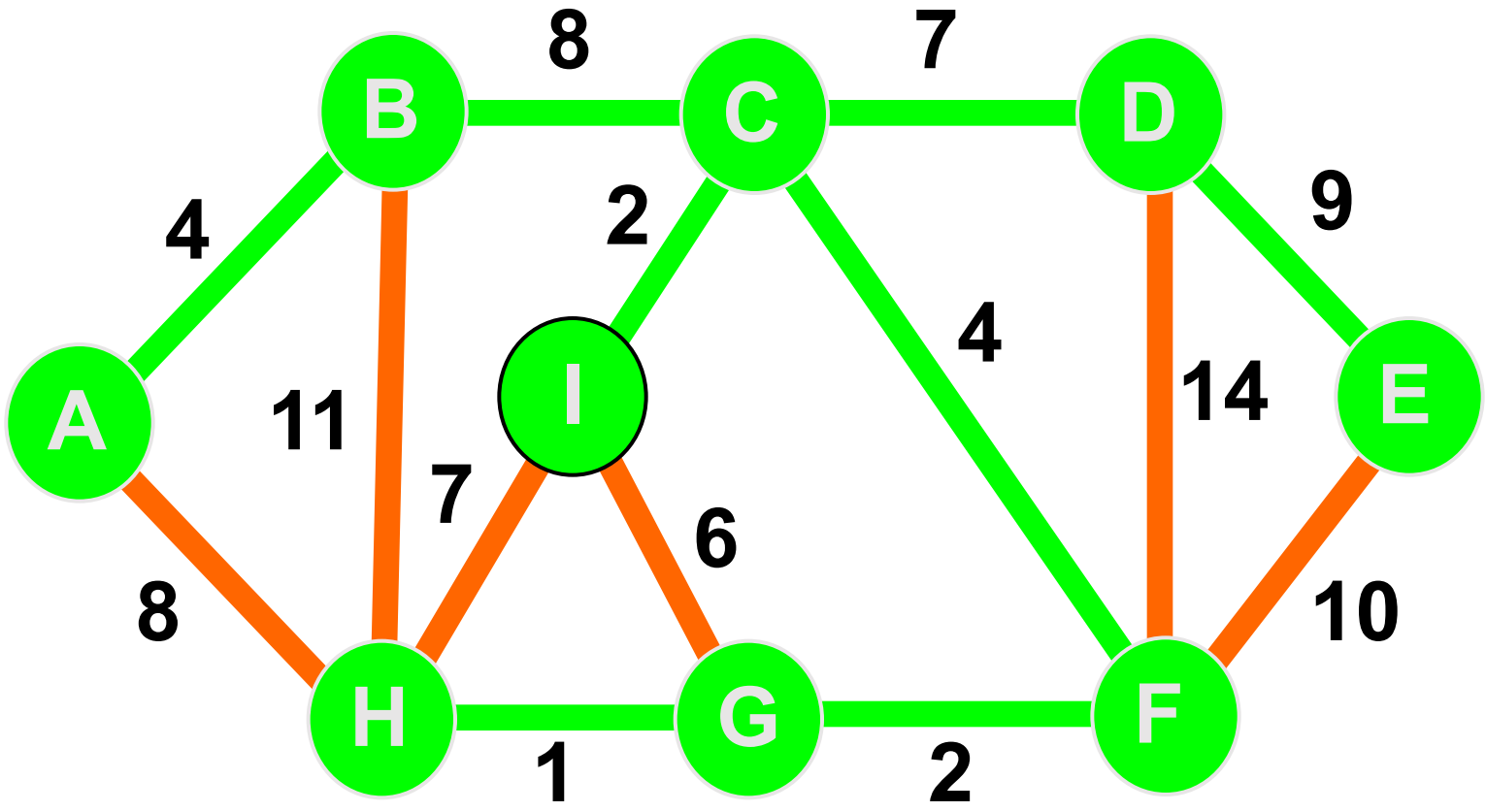
- **Суммарная длина дерева = 21**

Алгоритм Прима шаг 8



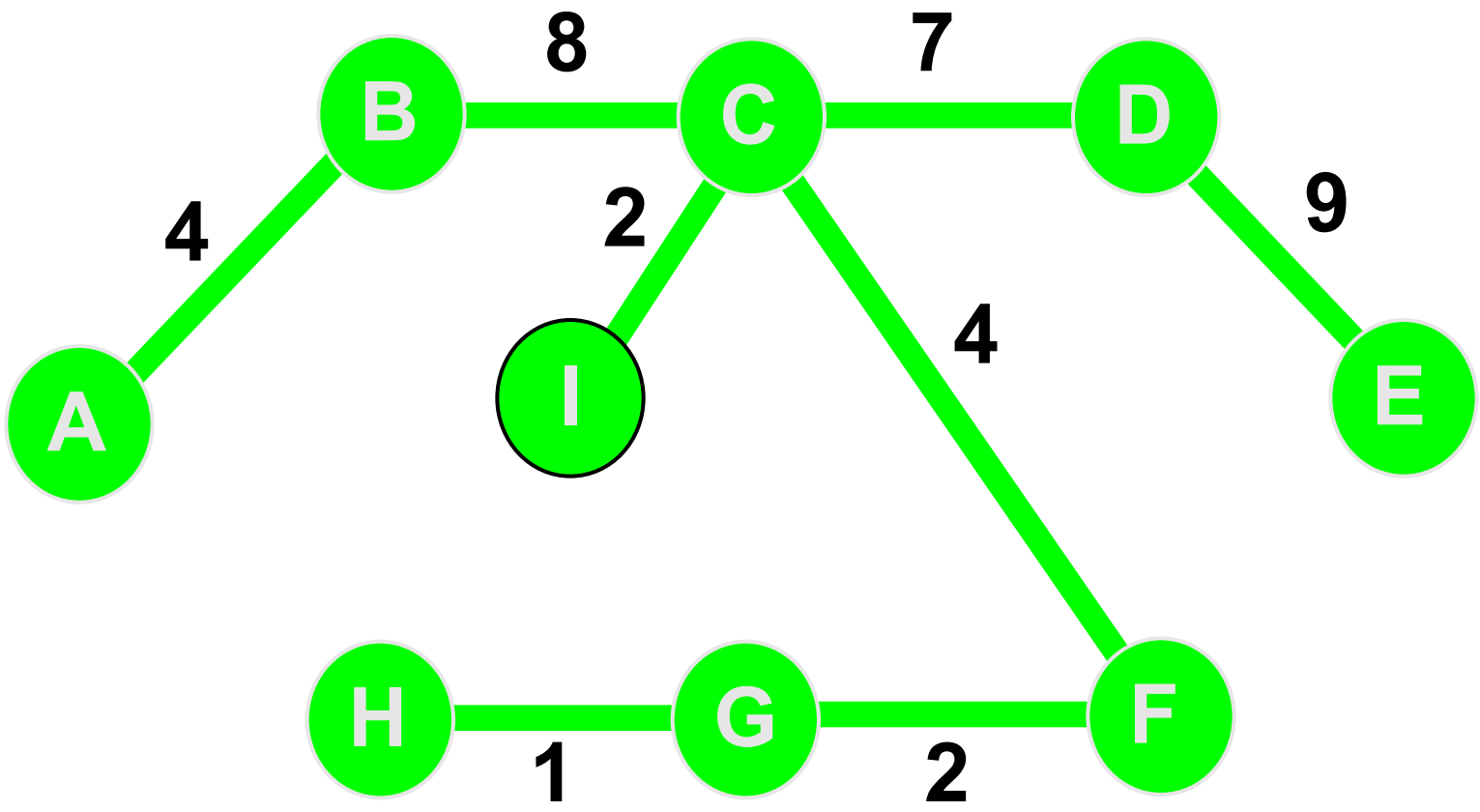
- **Суммарная длина дерева = 28**

Алгоритм Прима шаг 9



- **Суммарная длина дерева = 37**

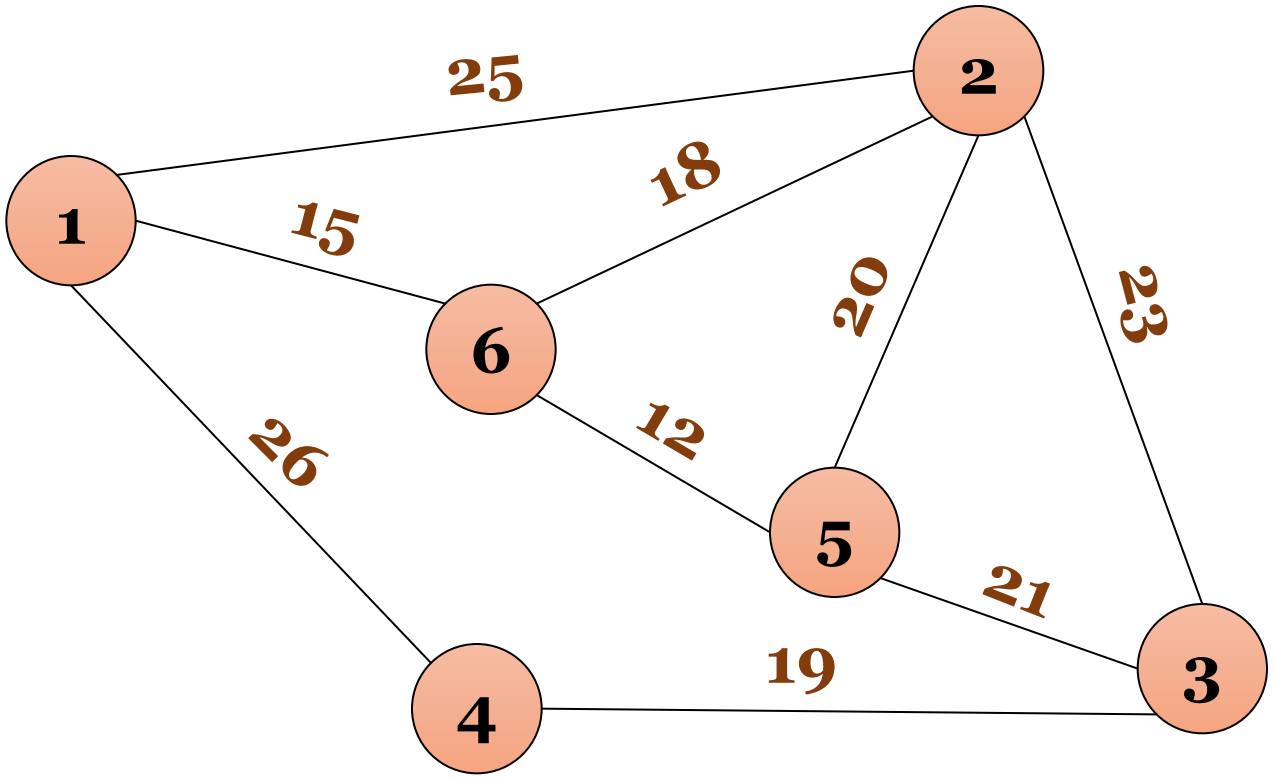
Алгоритм Прима шаг 10



- **Суммарная длина дерева = 37**

Задача 3

Даны города, часть которых соединена между собой дорогами. Необходимо проложить туристический маршрут минимальной длины, проходящий через все города.



Задача 3

Задача сводится к построению остовного связного дерева минимального веса.

Рассчитаем цикломатическое число.

m (количество ребер) равно **9**

n (количество вершин) рано **6**

$$\gamma = 9 - 6 + 1 = 4$$

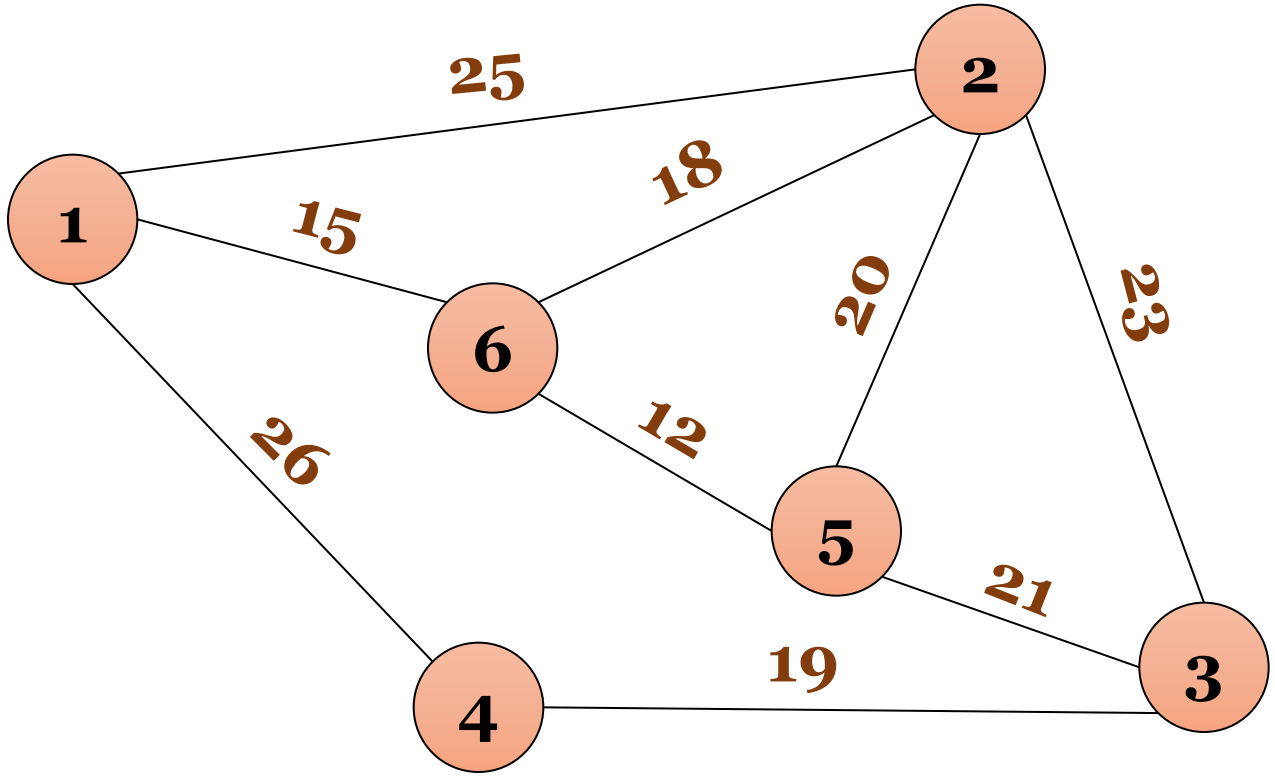
Т.е. четыре дороги, соединяющие города, не будут включены в туристический маршрут.

Алгоритм Крускала

1. Удалить все ребра и получить остовной подграф с изолированными вершинами.
2. Отсортировать ребра по возрастанию.
3. Ребра последовательно, по возрастанию их весов, включаются в остовное дерево. Возможны случаи:
   а) обе вершины включаемого ребра принадлежат одноэлементным подмножествам, тогда они объединяются в новое, связное подмножество;
   б) одна из вершин принадлежит связному подмножеству, другая нет, тогда включаем вторую в подмножество, которому принадлежит первая;
   в) обе вершины принадлежат разным связным подмножествам, тогда объединяем подмножества;
   г) обе вершины принадлежат одному связному подмножеству, тогда исключаем данное ребро.
4. Алгоритм завершается, когда все вершины будут объединены в одно множество.

Задача 3

Для определения туристического маршрута минимальной длины воспользуемся алгоритмом Крускала.

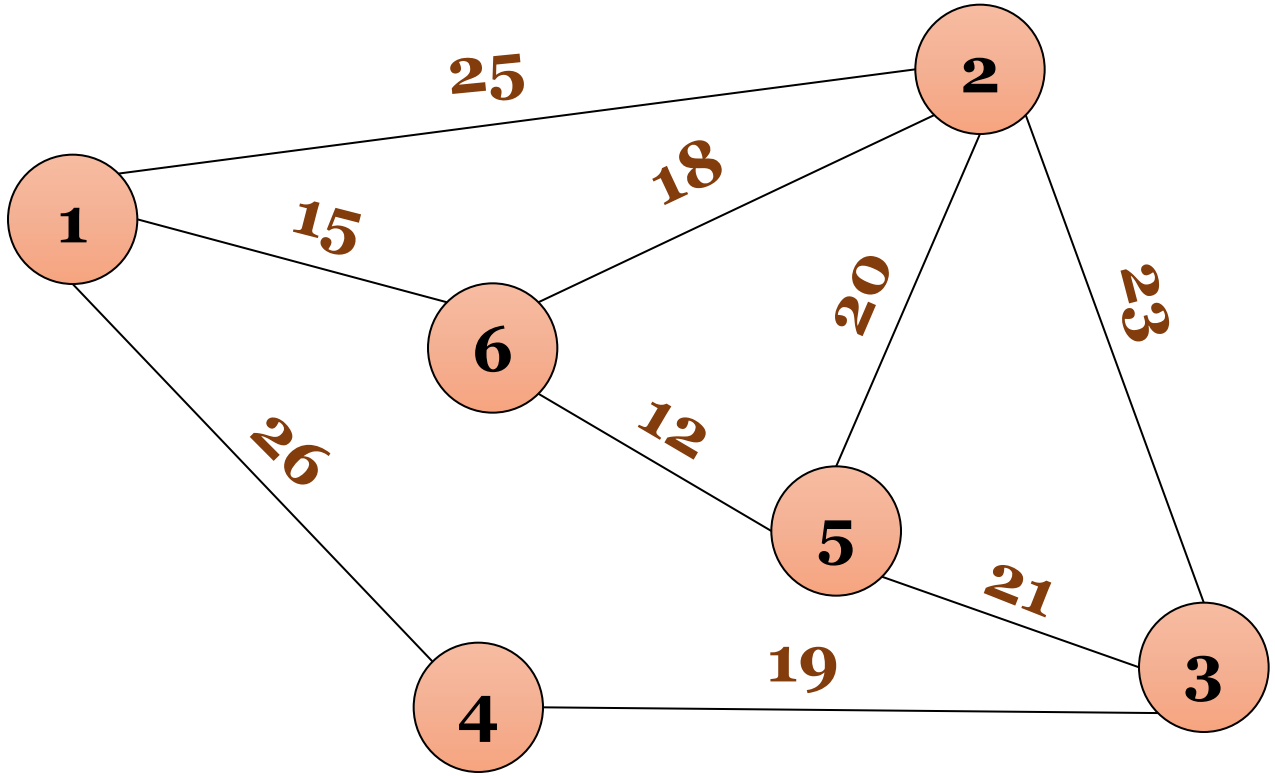


Задача 3

Шаг 1

Построим остовной подграф, содержащий только изолированные вершины.

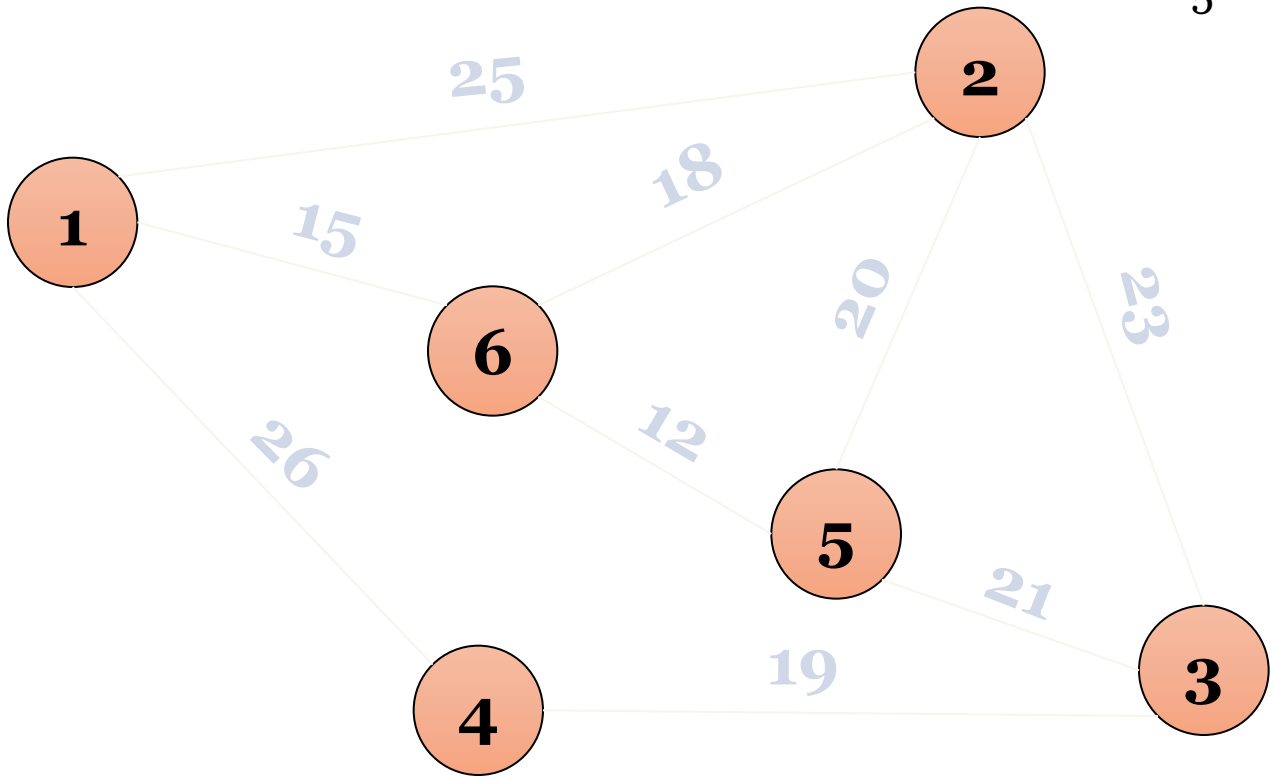
Получаем шесть одноэлементных подмножеств.



Задача 3

Шаг 2 Найдем ребро минимального веса и добавим его в остовной подграф.

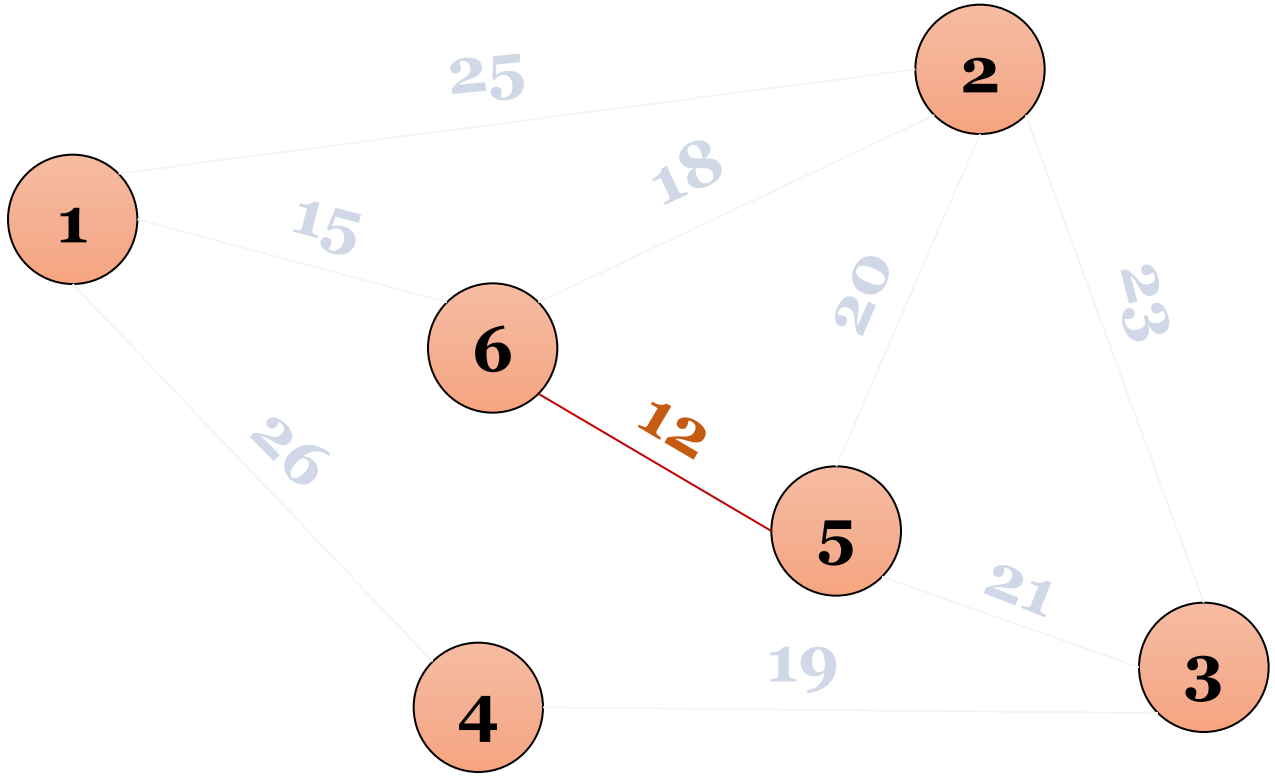
Образуется связное подмножество вершин $\{V_5, V_6\}$.



Задача 3

Шаг 3 Среди оставшихся ребер найдем ребро минимального веса и добавим его в остовной подграф.

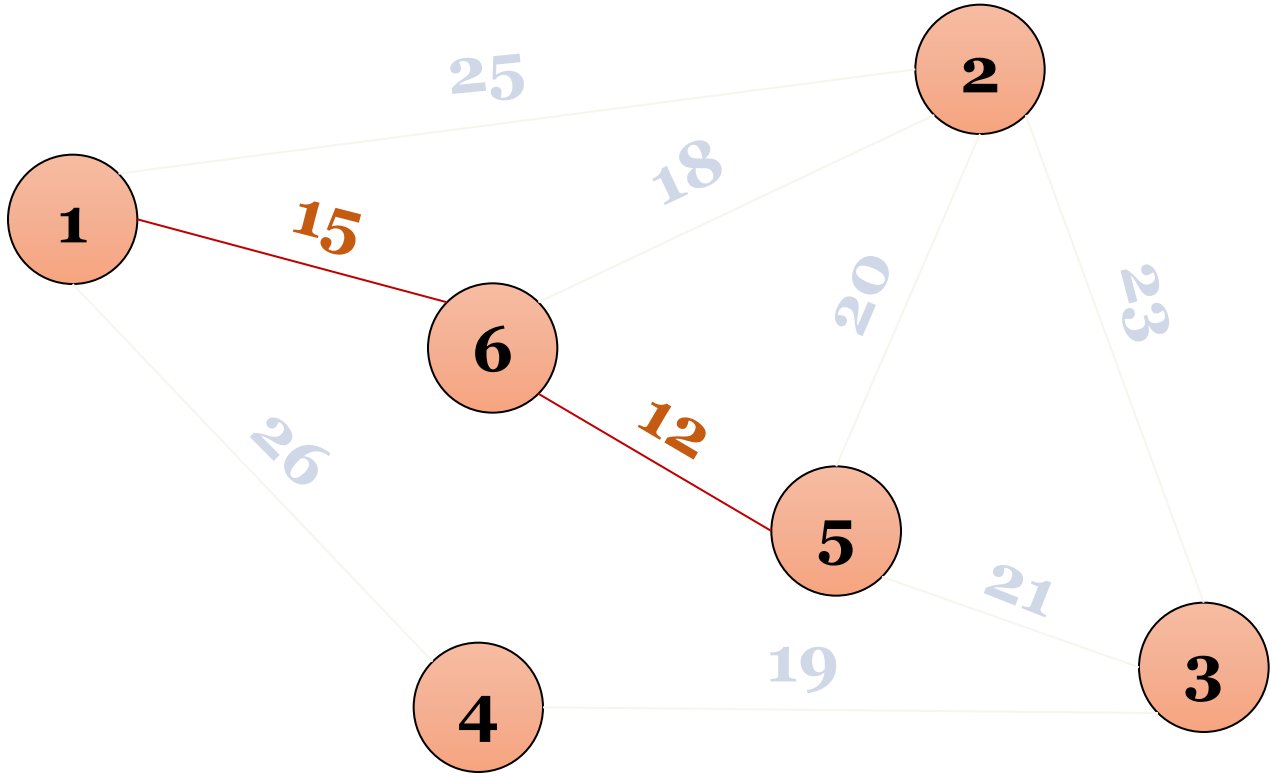
Добавляем в подмножество вершин еще одну $\{V_5, V_6, V_1\}$.



Задача 3

Шаг 4 Среди оставшихся ребер найдем ребро минимального веса и добавим его в остовной подграф.

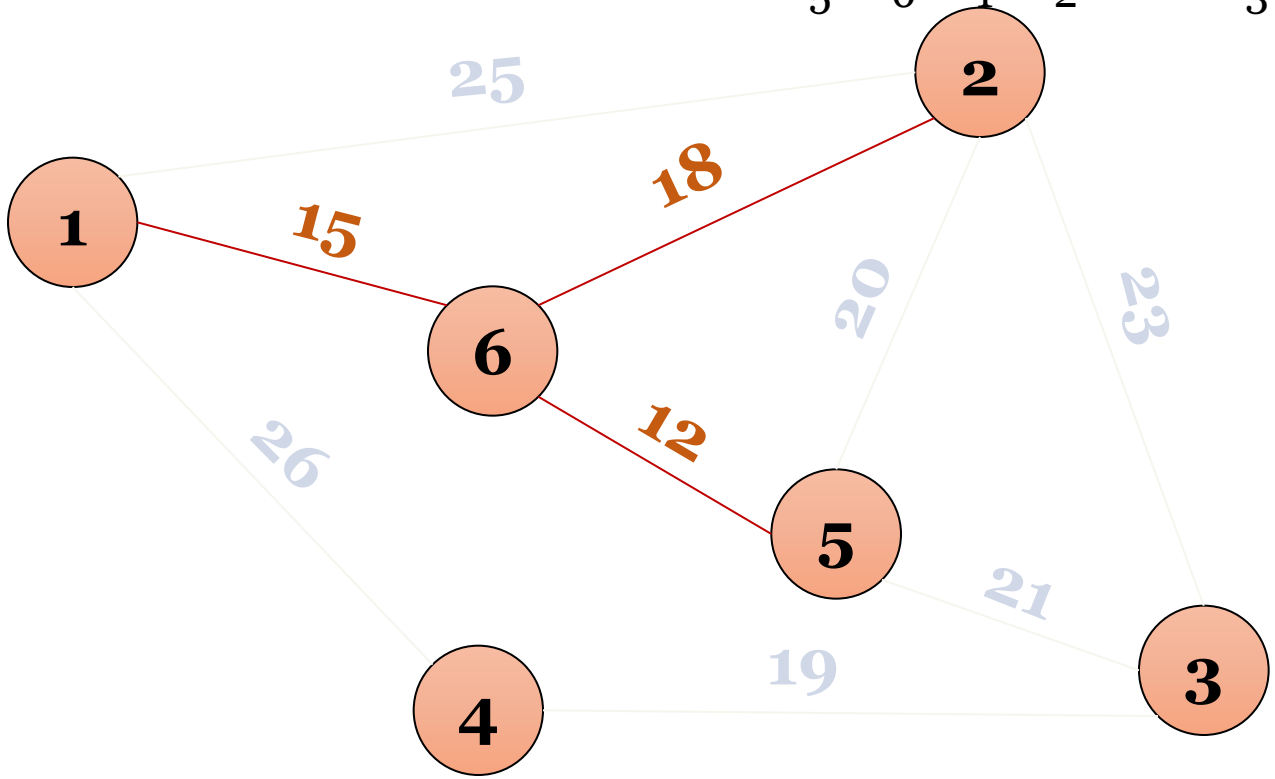
Добавляем в подмножество вершин еще одну $\{V_5, V_6, V_1, V_2\}$.



Задача 3

Шаг 5 Среди оставшихся ребер найдем ребро минимального веса и добавим его в остовной подграф.

Образуется два подмножества $\{V_5, V_6, V_1, V_2\}$ и $\{V_3, V_4\}$.

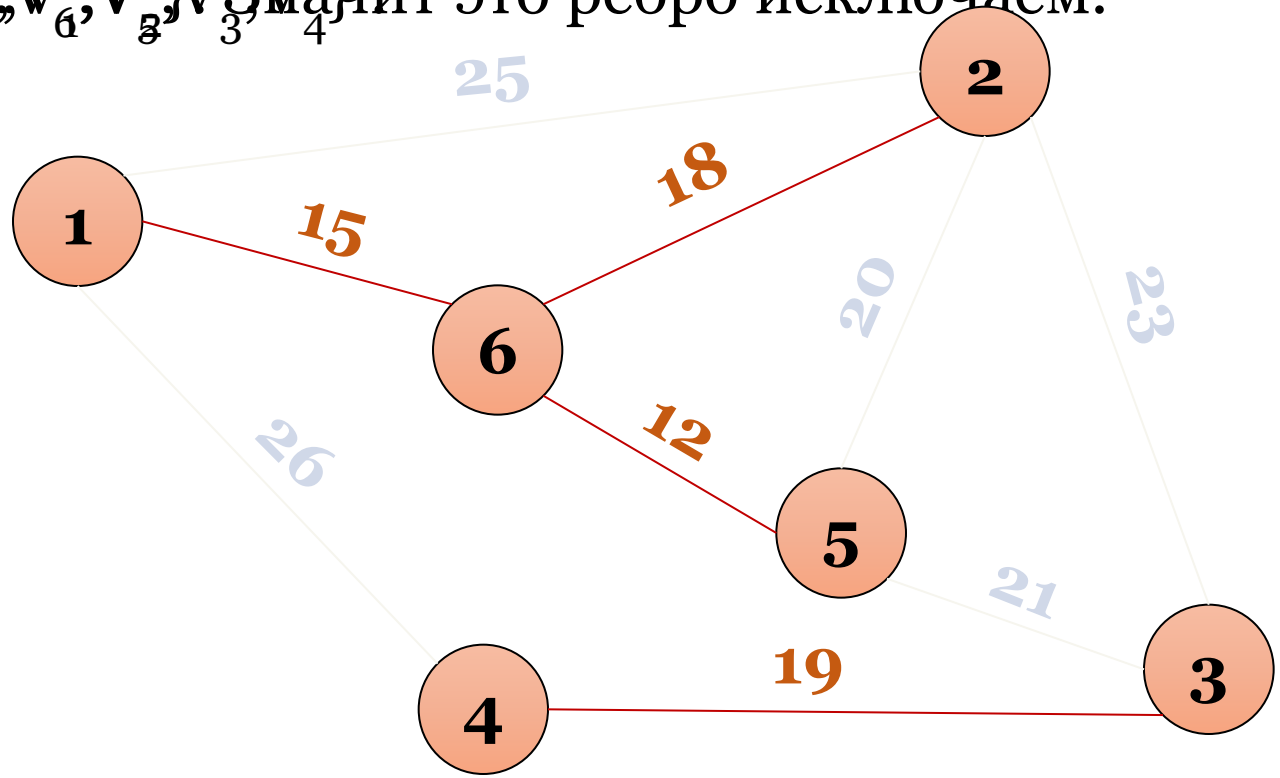


Задача 3

Шаг 6

Среди оставшихся ребер найдем ребро минимального веса и добавим его в остовной подграф.

Но обе вершины ребра принадлежат одному связному множеству $\{V_5, V_6, V_2, V_3\}$, значит это ребро исключаем.

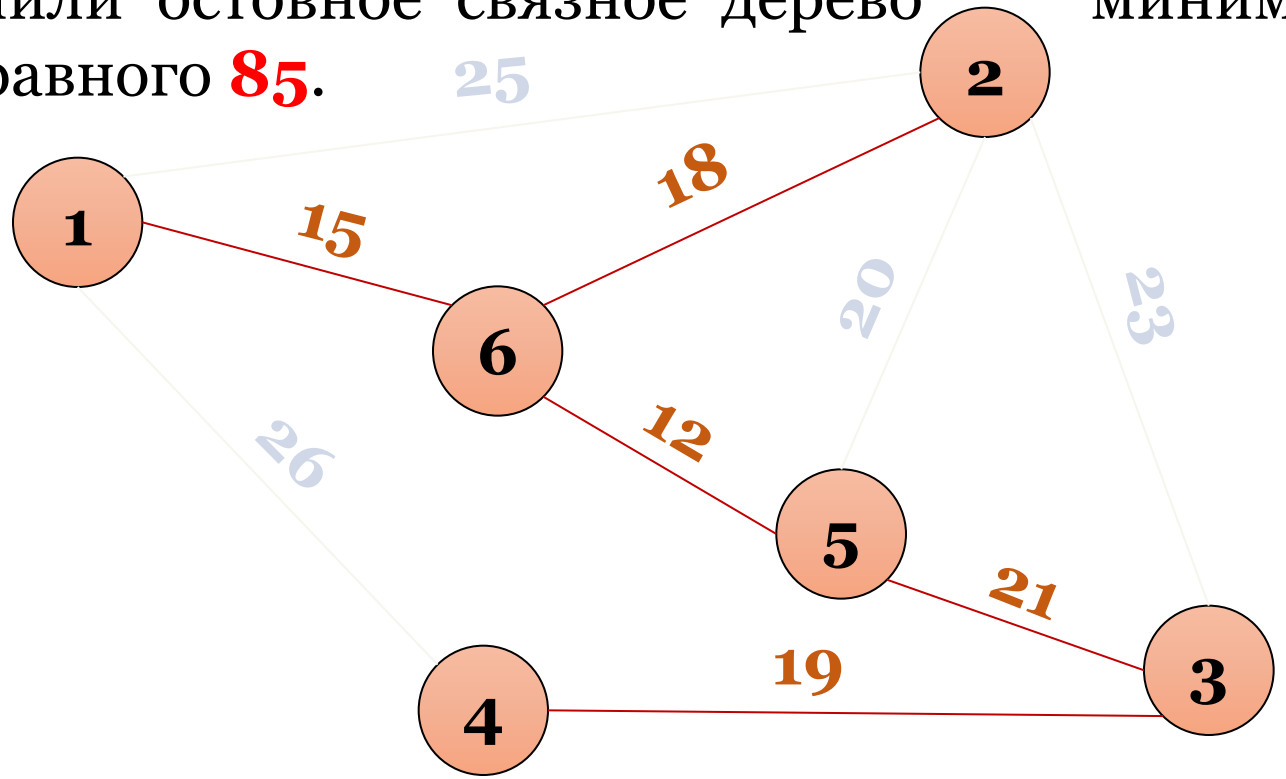


Задача 3

Итог

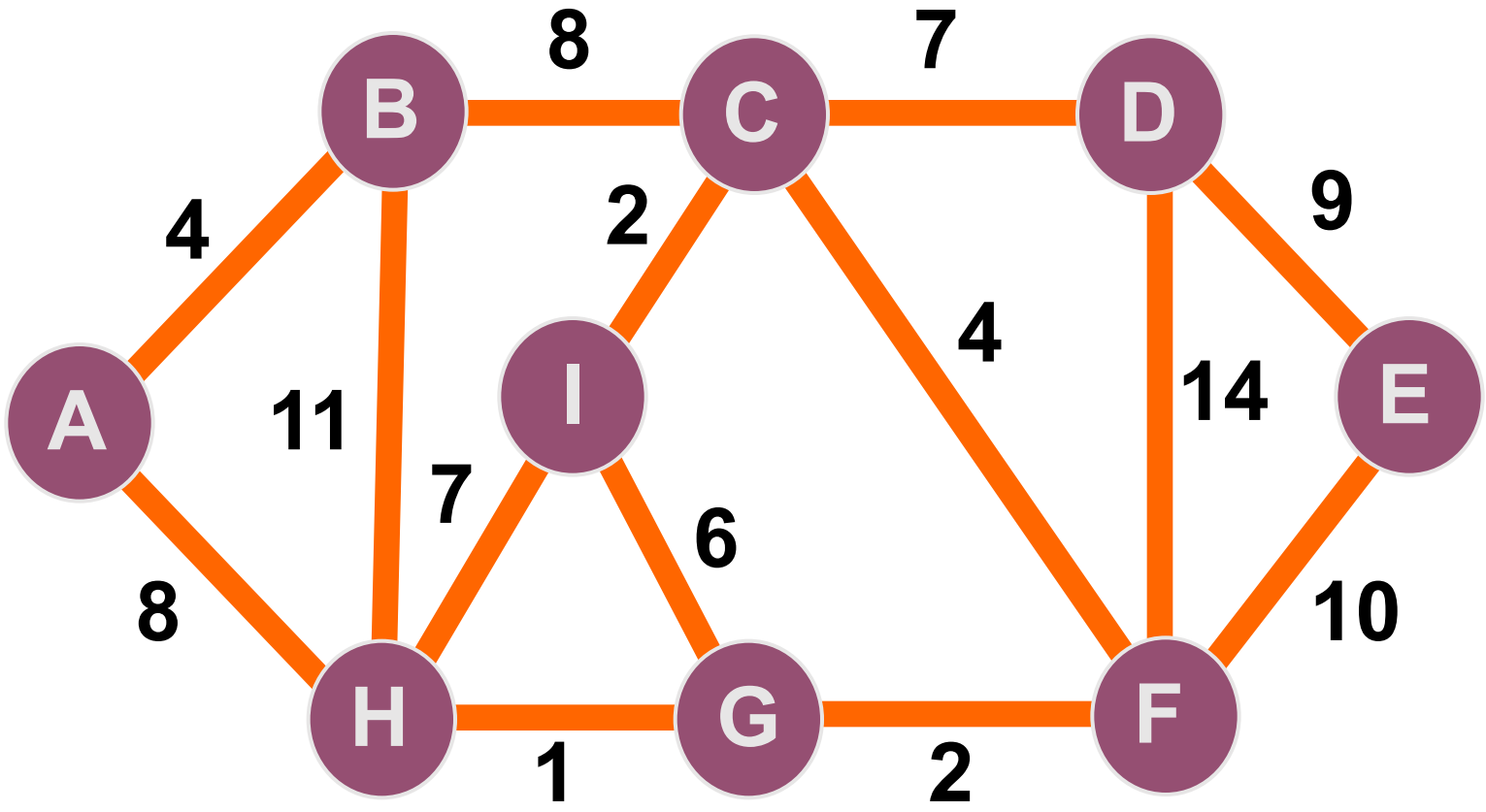
Остальные ребра включать в граф не надо, т.к. все их вершины уже принадлежат одному связному множеству.

Получили остовное связное дерево минимального веса, равного **85**.



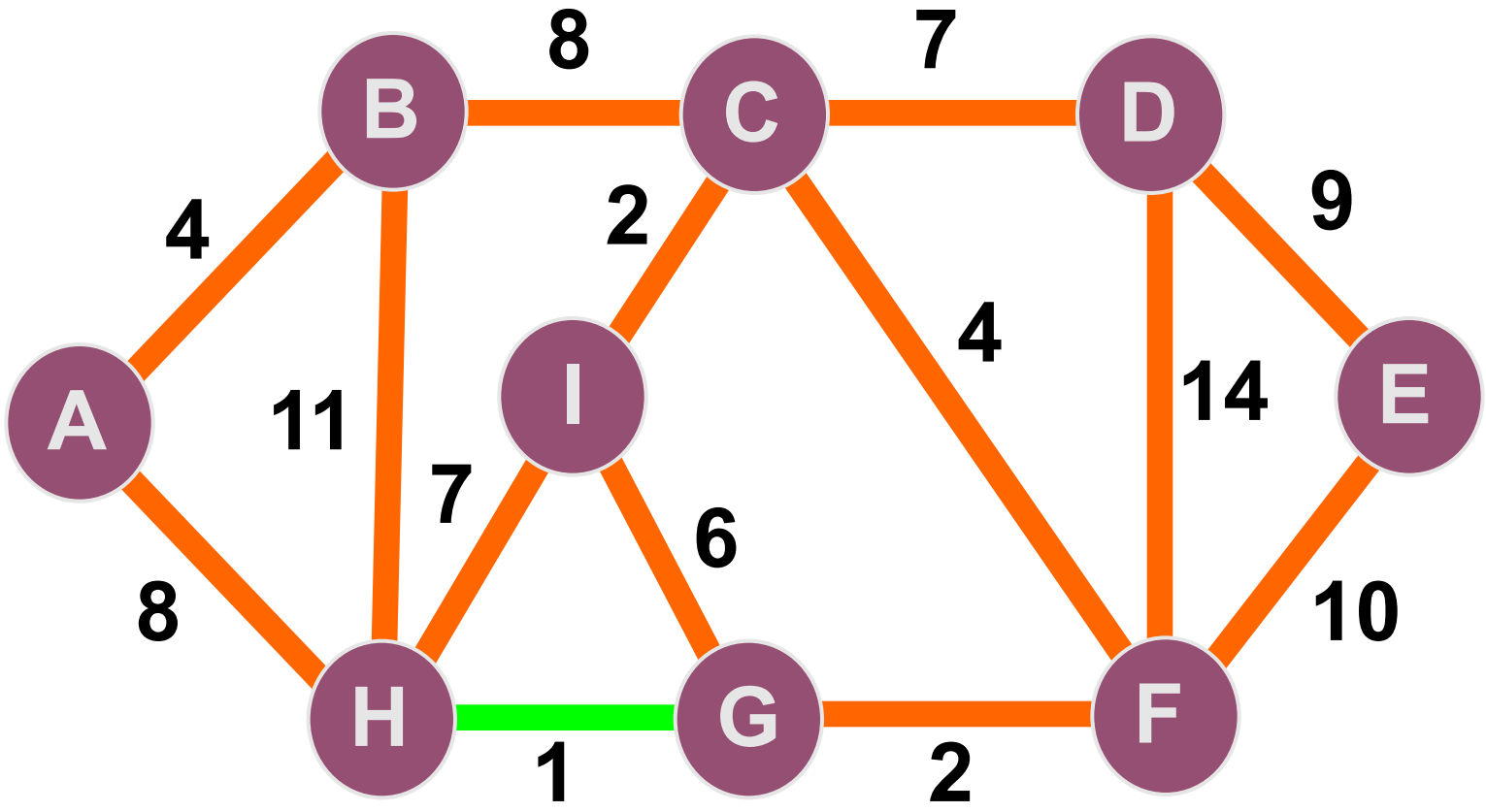
Пример 4

Алгоритм Краскала шаг 0



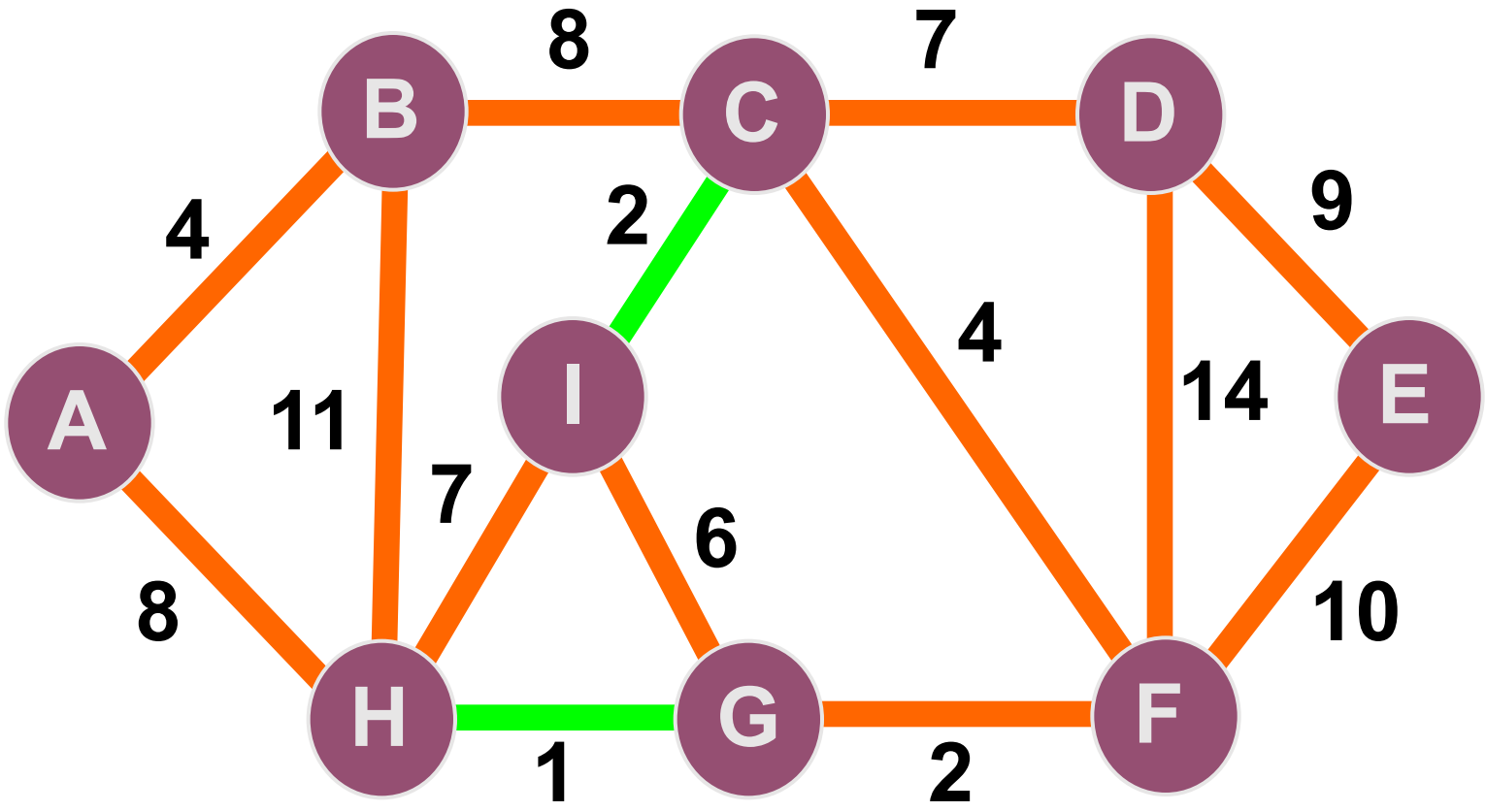
- **Суммарная длина деревьев = 0**

Алгоритм Краскала шаг 1



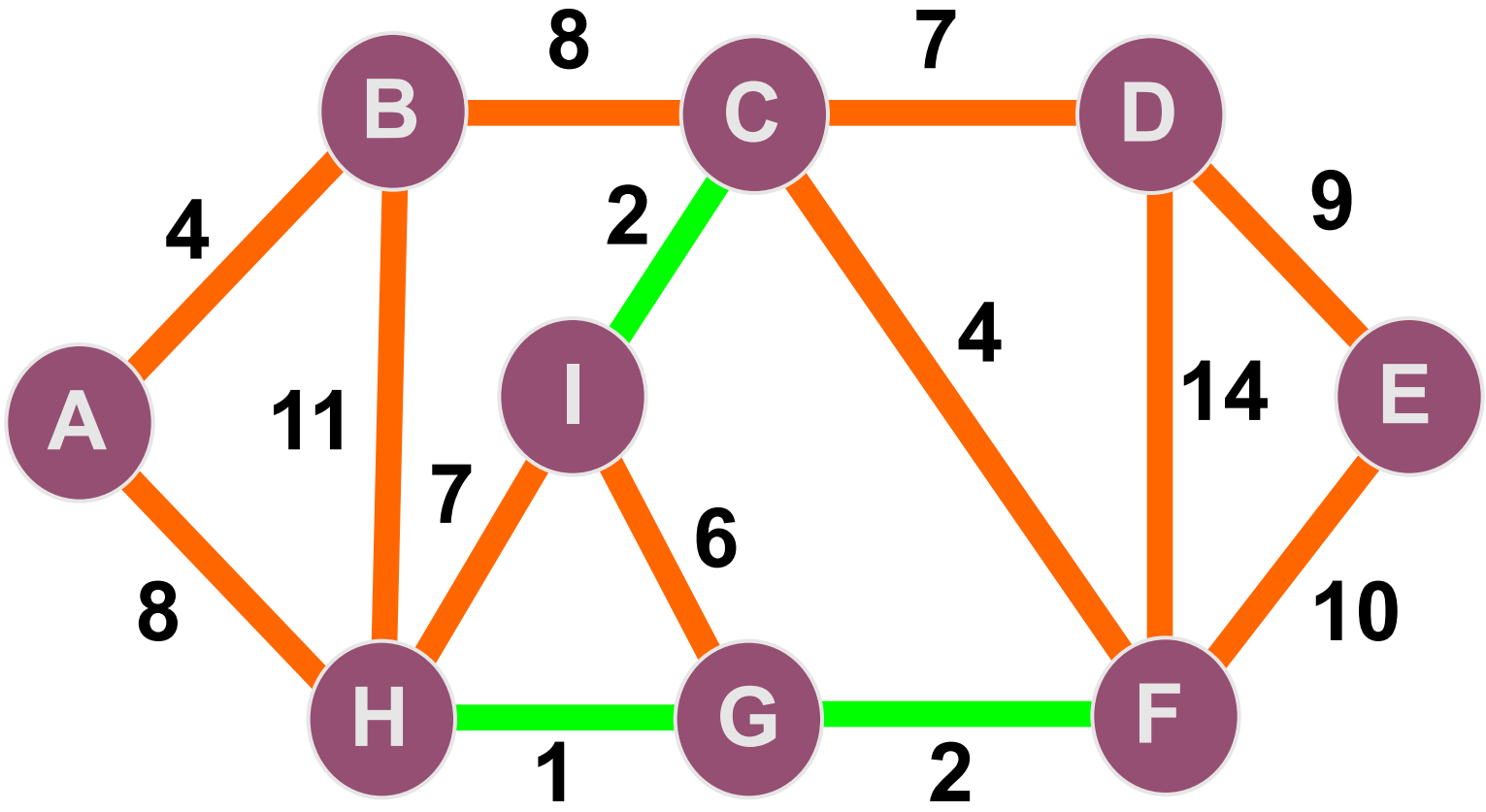
- **Суммарная длина деревьев = 1**

Алгоритм Краскала шаг 2



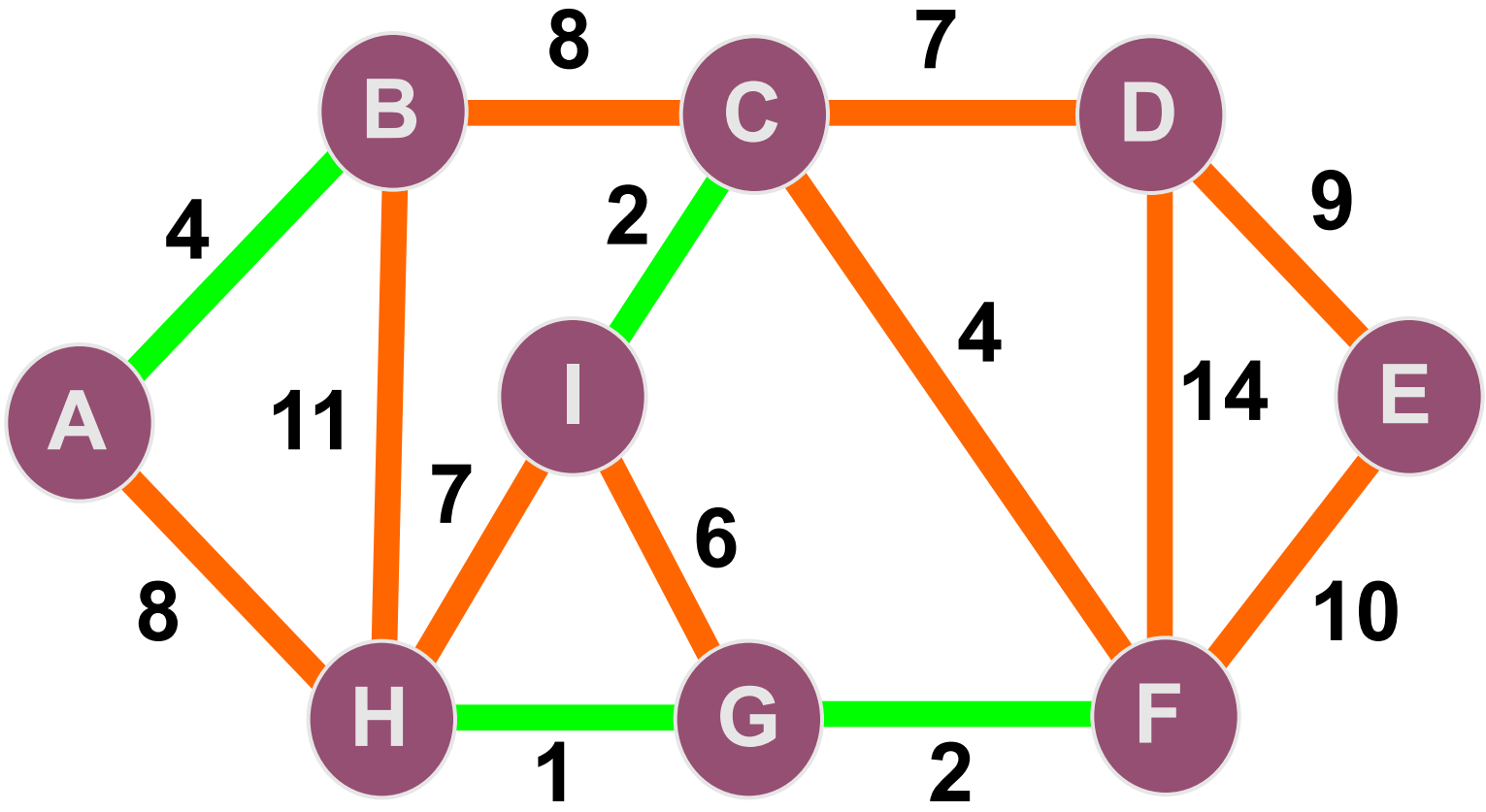
- **Суммарная длина деревьев = 3**

Алгоритм Краскала шаг 3



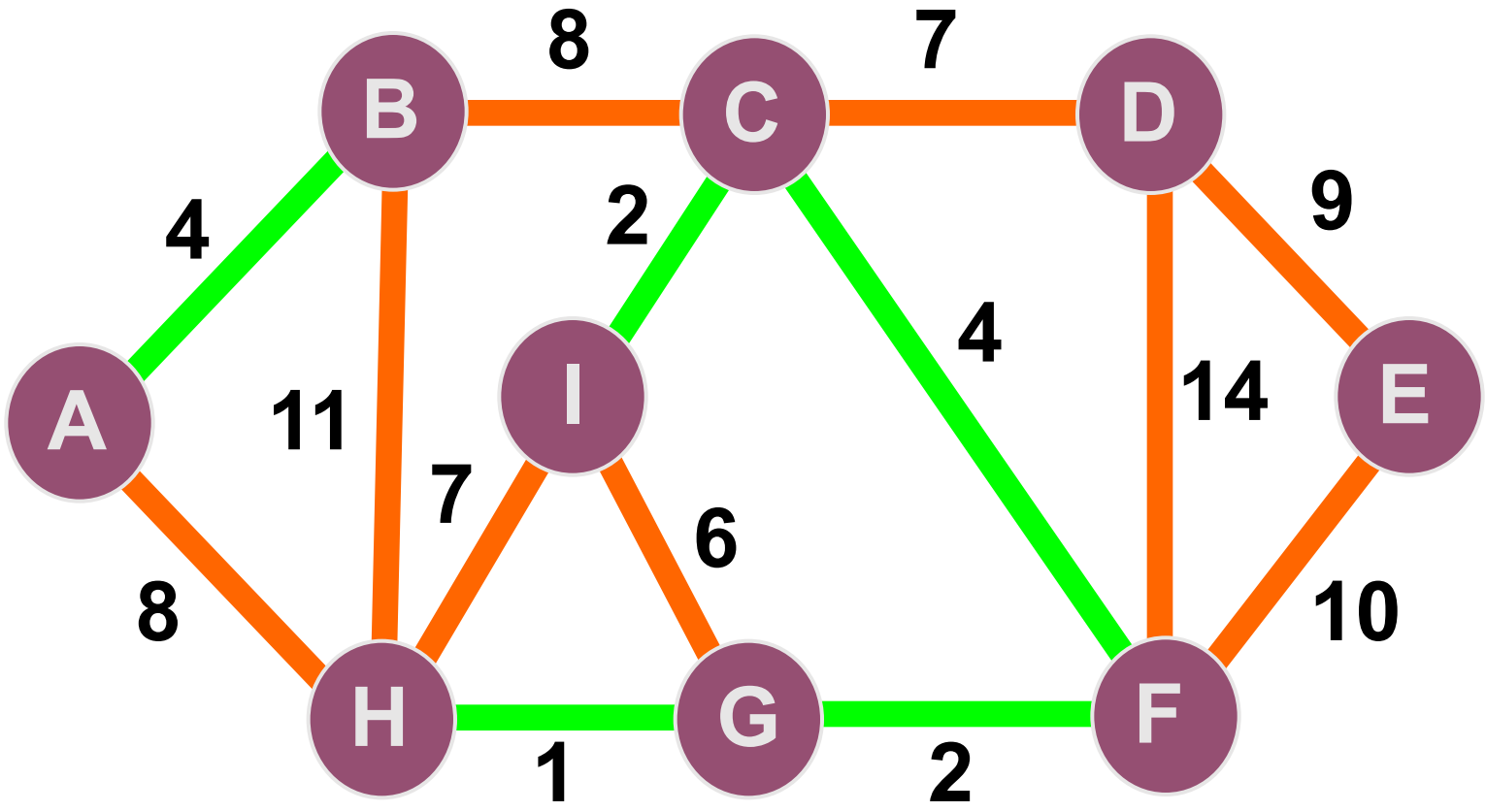
- **Суммарная длина деревьев = 5**

Алгоритм Краскала шаг 4



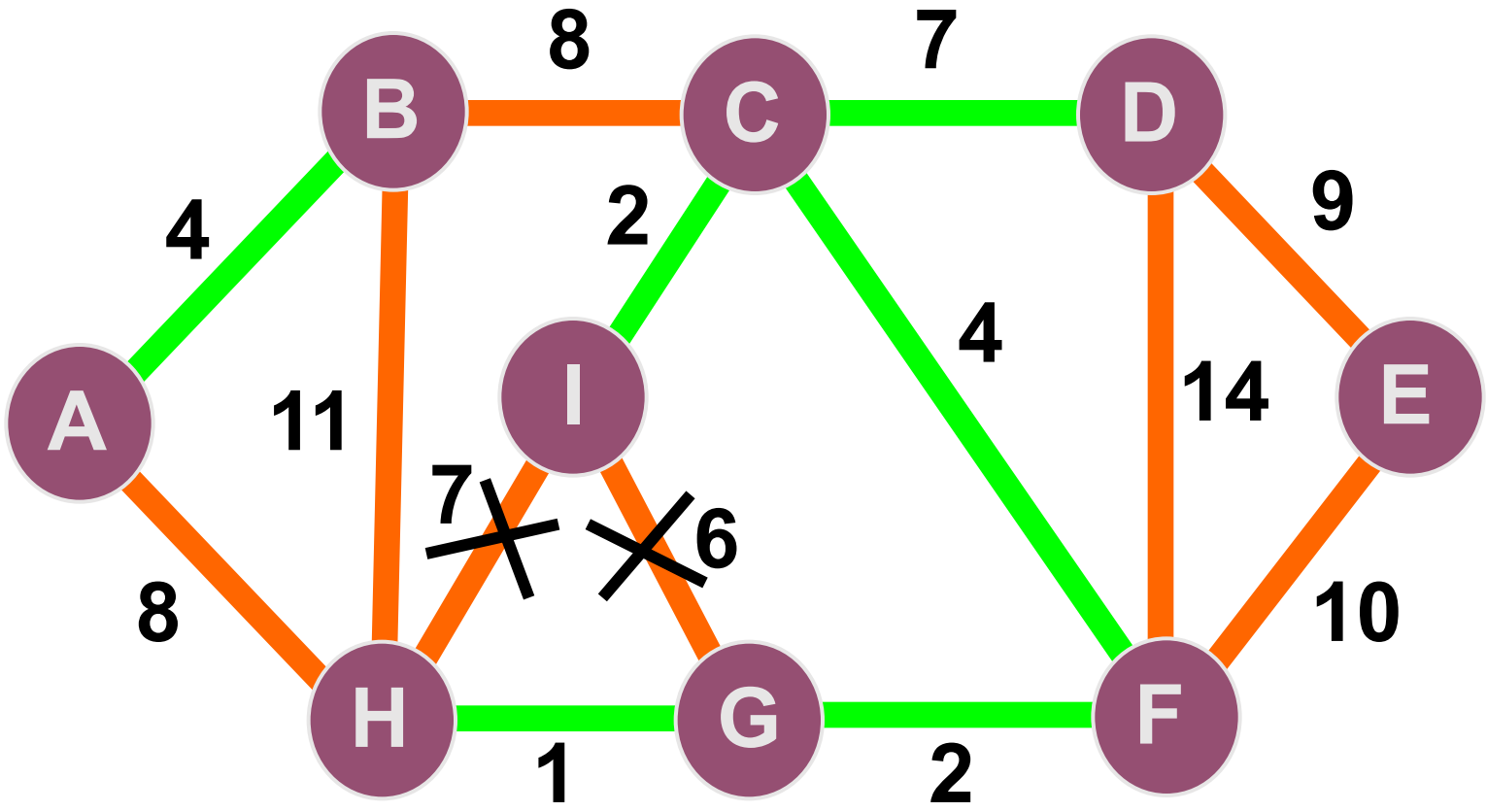
- **Суммарная длина деревьев = 9**

Алгоритм Краскала шаг 5



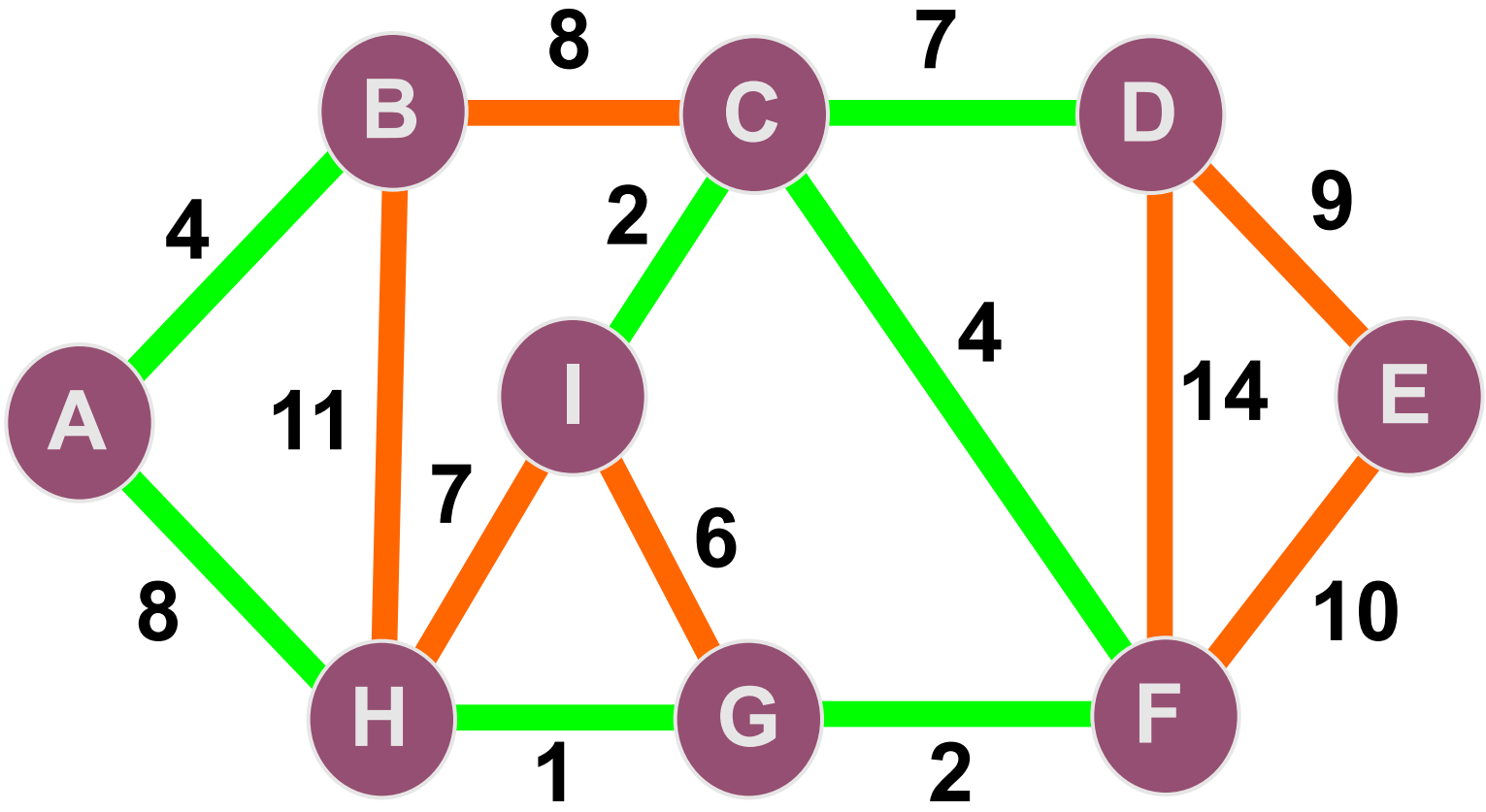
- **Суммарная длина деревьев = 13**

Алгоритм Краскала шаг 6



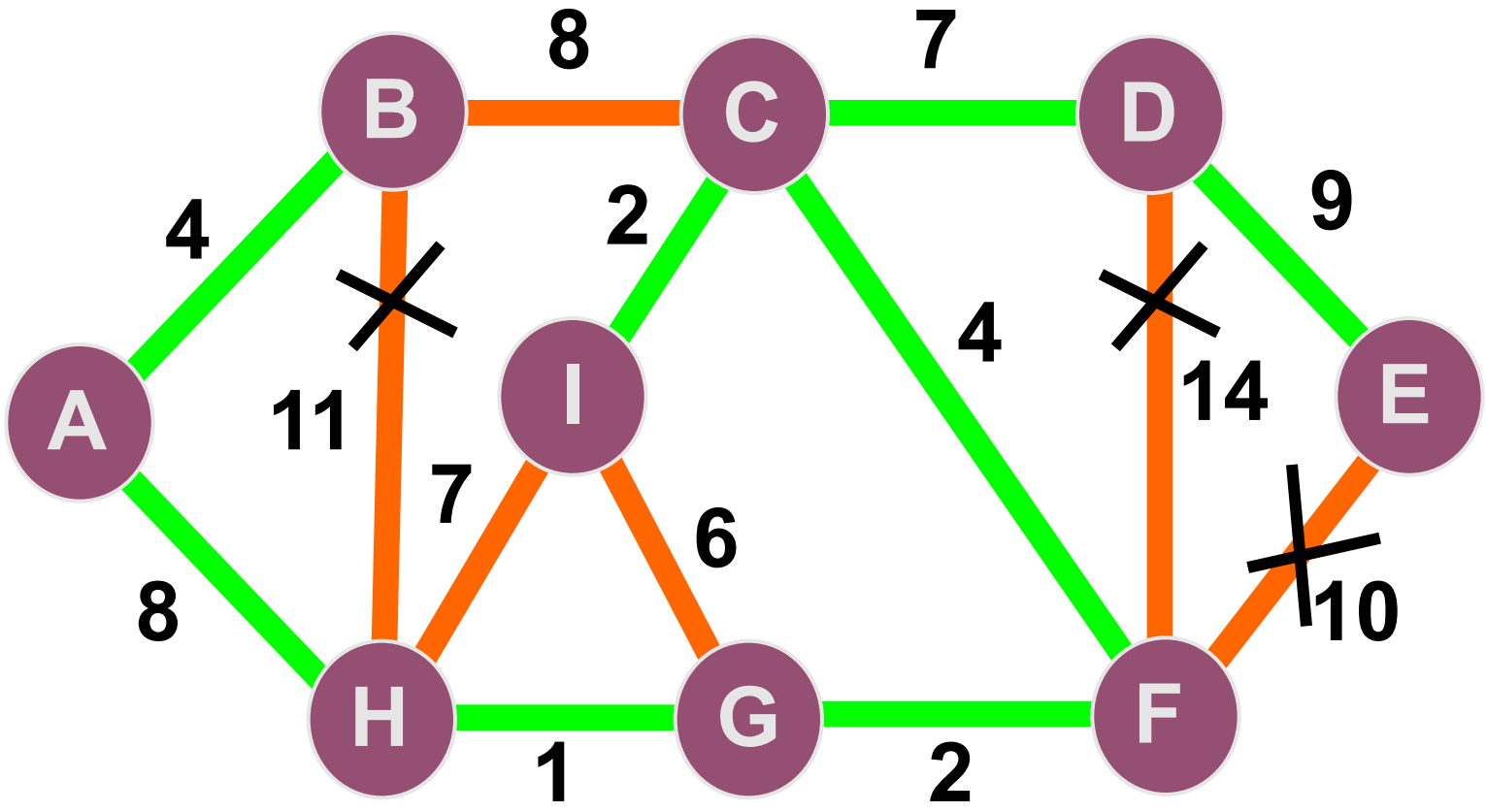
- **Суммарная длина деревьев = 20**

Алгоритм Краскала шаг 7



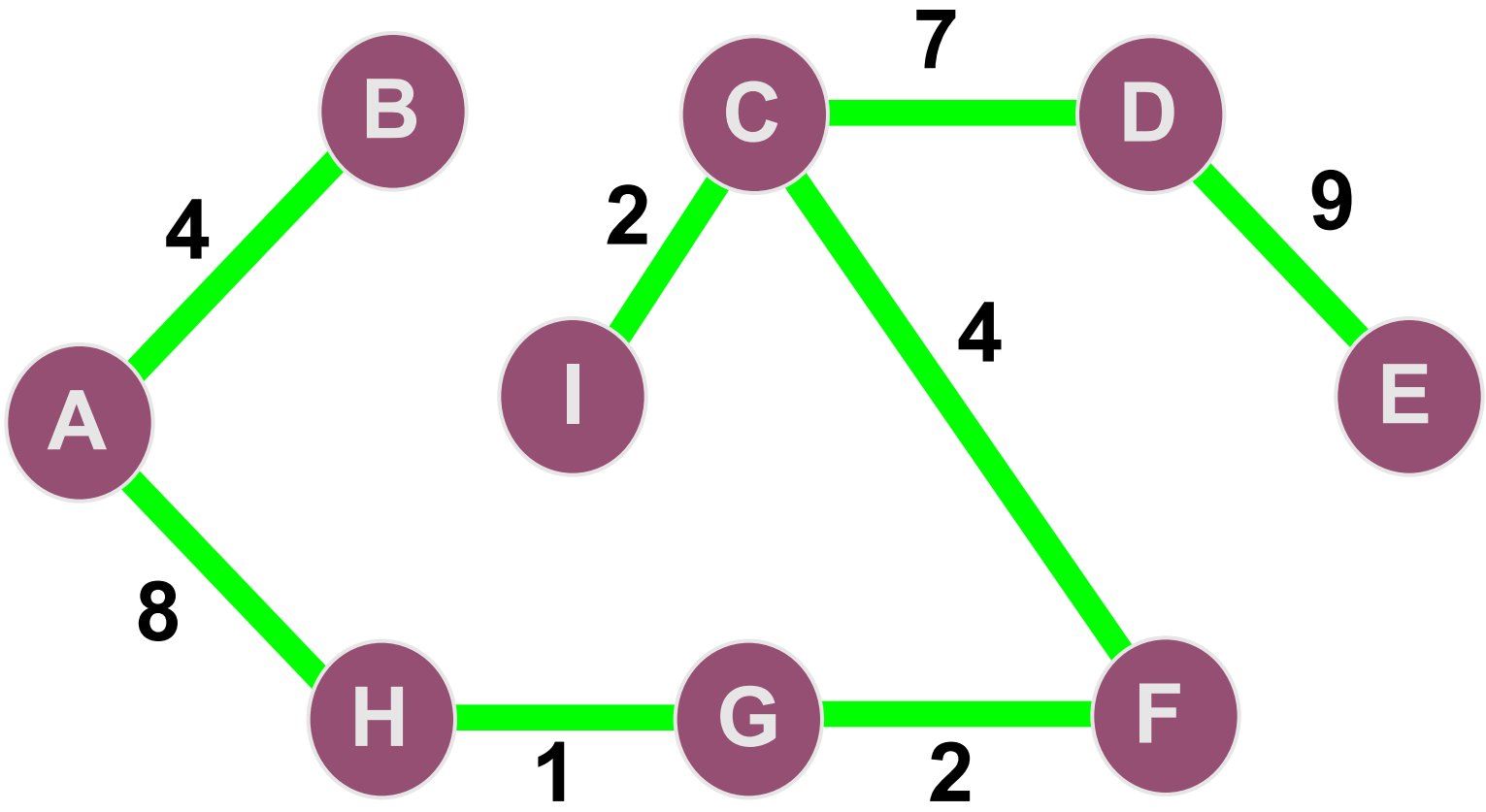
- **Суммарная длина деревьев = 28**

Алгоритм Краскала шаг 8



- **Суммарная длина деревьев = 37**

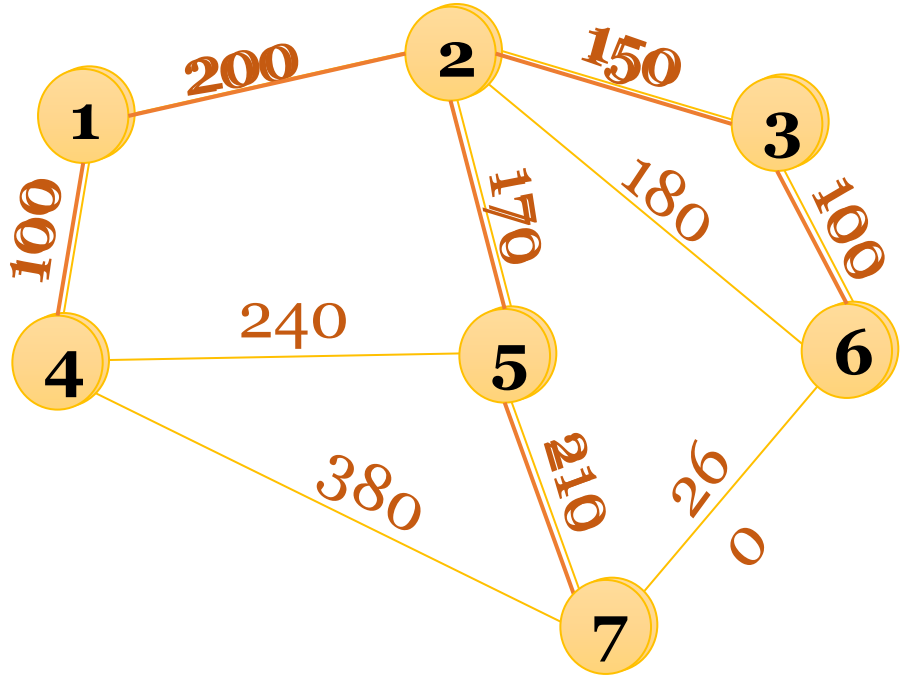
Алгоритм Краскала шаг 9



- **Суммарная длина деревьев = 37**

Задача 5

На строительном участке необходимо создать телефонную сеть, соединяющую все бытовки. Для того, чтобы телефонные линии не мешали строительству, их решили проводить вдоль дорог. Схема участка изображена на рисунке.



Каким образом провести телефонные линии, чтобы их общая длина была минимальной?

Ответ

Общая длина телефонной линии равна **930** метров