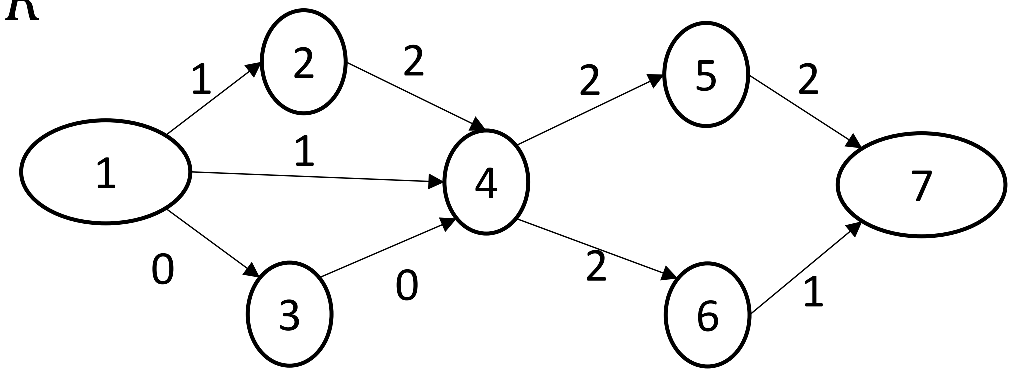
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Максимальный поток в сети и его приложения

https://github.com/larandaA/alg-ds-snippets

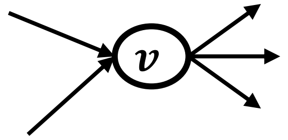
Дивергенция

$f\colon \mathrm{E} \to R$



Дивергенция функции $\boldsymbol{f}$ в вершине $\boldsymbol{v}$ (от лат. *divergere* — расхождение) определяется как разность сумм её значений на выходящих и входящих дугах:

$$div_f(v) = \sum_{e \in N^+(v)} f(e) - \sum_{e \in N^-(v)} f(e)$$



$e \in N^-(v)$ $e \in N^+(v)$

Г.М. Адельсон-Вельский, Е.А. Диниц, А.В. Карзанов. ***Потоковые алгоритмы.*** М.,Наука, 1975 г.

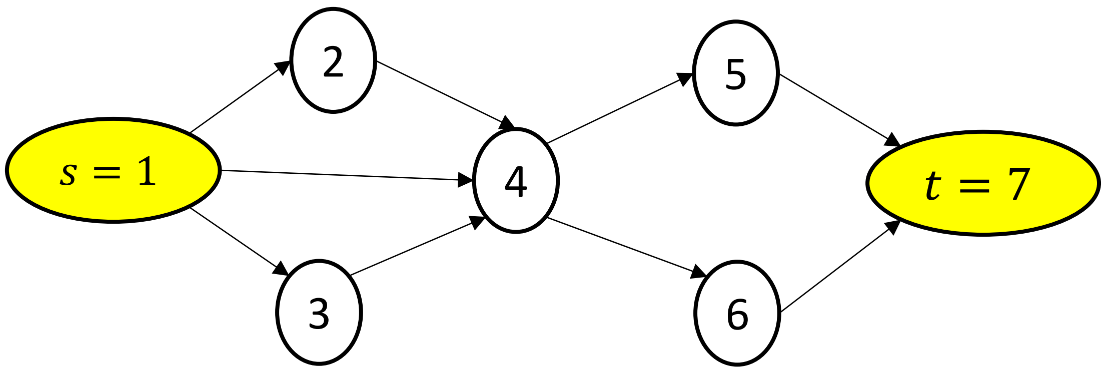
Сеть

Если в орграфе некоторые вершины отмечены, то он называется **сетью**, отмеченные вершины – **полюсами**, а остальные вершины – **внутренними**.

Если на дивергенцию в полюсе $\boldsymbol{s}$ наложено ограничение $div_f(s) \geq 0$, то он называется **источником**.

Если на дивергенцию в полюсе $\boldsymbol{t}$ наложено ограничение $div_f(t) \leq 0$, то он называется **стоком**.

В **классической задаче о максимальном потоке** имеются **два полюса** $\boldsymbol{s}$ и $\boldsymbol{t}$ и предполагается, что $\boldsymbol{s}$ – вершина сети, из которой дуги только выходят (**источник**), а $\boldsymbol{t}$ – вершина сети, в которую дуги только входят (**сток**).

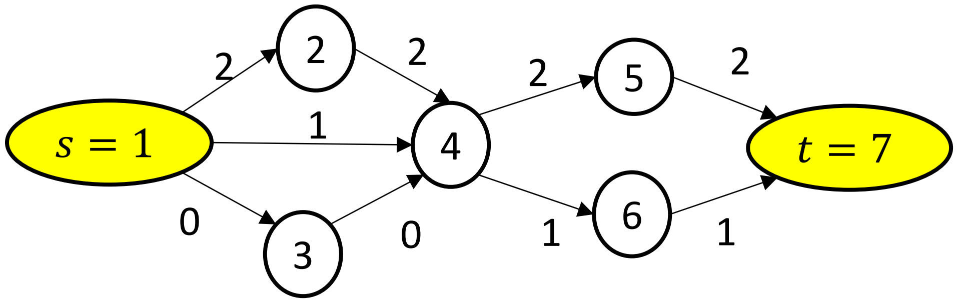


Потоком в сети

называется функция, дивергенция которой во внутренних вершинах равна 0 (для полюсов допускается дебаланс, т.е. ненулевая дивергенция).

Если $div_f(s) = div_f(t) = 0$, **поток f** называется **циркуляцией**.

Пусть f – поток. Сложим дивергенцию функции f во всех вершинах сети.



$$\sum_{v \in V} div_f(v) = div_f(s) + \sum_{w \in V \setminus \{s,t\}} div_f(w) + div_f(t) = \mathbf{0}$$

$$div_f(s) + div_f(t) = 0$$

Для потока f в сети с полюсами s и t следует: $div_f(s) = -div_f(t)$.

Эта дивергенция называется **мощностью потока** f:

$$\boldsymbol{M(f)} = div_f(s) = -div_f(t).$$

Сеть с ограничениями (потоковая сеть)

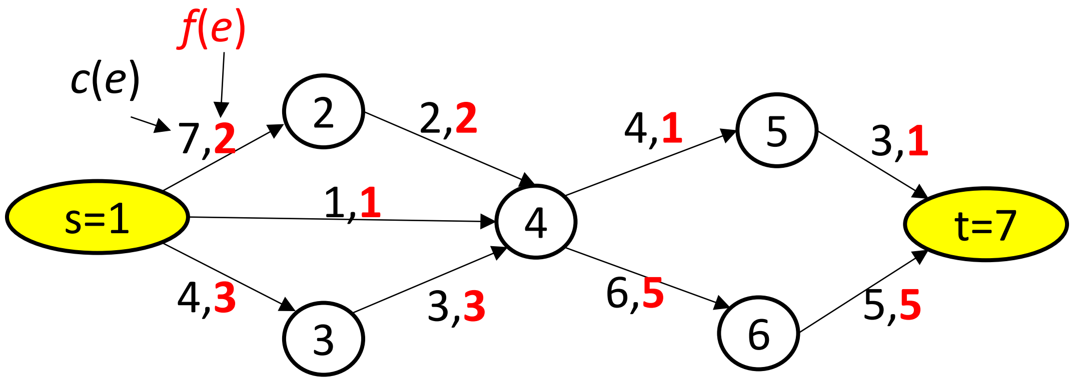
В дальнейшем на дугах сети задаются ограничения, которым должны удовлетворять все рассматриваемые потоки:

$$d(e) \leq f(e) \leq c(e)$$

в классической задаче о максимальном потоке все $d(e) = 0$

верхнее ограничение $c(e)$ *называется пропускной способностью дуги* e

$$0 \leq f(e) \leq c(e)$$



Классическая задача о максимальном потоке

Задан орграф в котором выделены две вершины: $\boldsymbol{s}$ (в которую нет входящих дуг) и $\boldsymbol{t}$ (из которой нет выходящих дуг). Каждой дуге орграфа приписана пропускная способность $c(e) \geq 0$. Таким образом, мы имеем некоторую потоковую (s,t)-сеть.

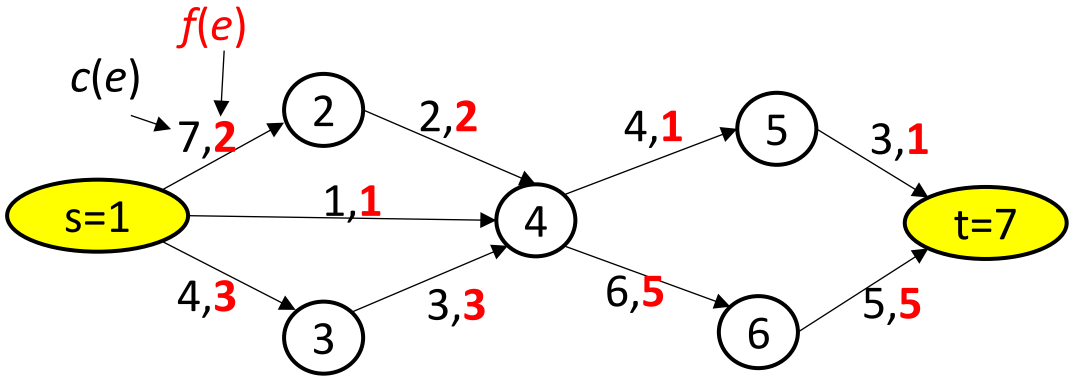
Требуется в (s,t)-сети найти поток $\boldsymbol{f}$ максимальной мощности, который удовлетворяет ограничениям:

$$0 \leq f(e) \leq c(e), \forall e \in E .$$

Мощность потока $\boldsymbol{f}$ равна количеству потока выходящего из вершины s:

$$M(f) = div_f(s) = -div_f(t).$$

Поток, мощность которого максимальна, называется **максимальным потоком**.



$M(f) = 6$

Замечания

1. В дальнейшем мы будем работать с **целочисленными потоками**, то есть все ограничения – целые числа.
2. Считаем, что любая внутренняя вершина сети лежит на некотором (s, t)-пути ($m \geq n - 1$).
3. Предполагаем, что в сети **нет кратных дуг**.

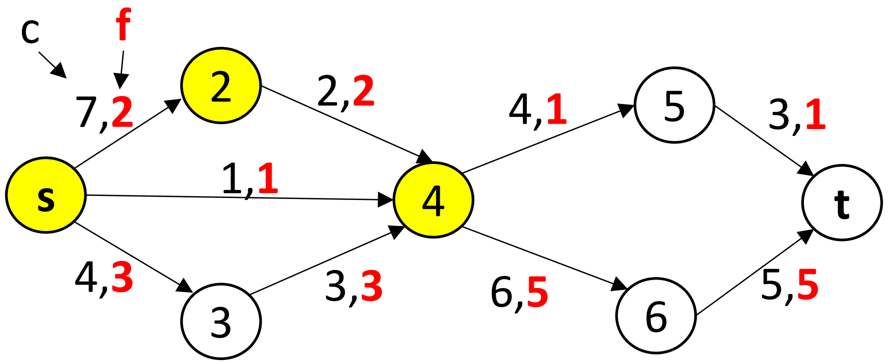
Максимальный поток и минимальный разрез

$R = (X, \overline{X}),\ X \subseteq V, \overline{X} = V \backslash X,\ s \in X,\ \mathrm{t} \in \overline{X}$

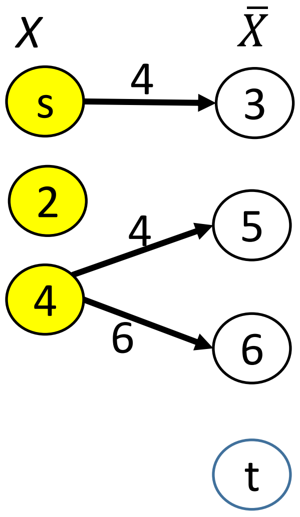
$E^+(R)$ – дуги, которые идут из X в $\overline{X}$

$E^-(R)$ – дуги, которые идут из $\overline{X}$ в X

остальные дуги сети – внутренние дуги разреза

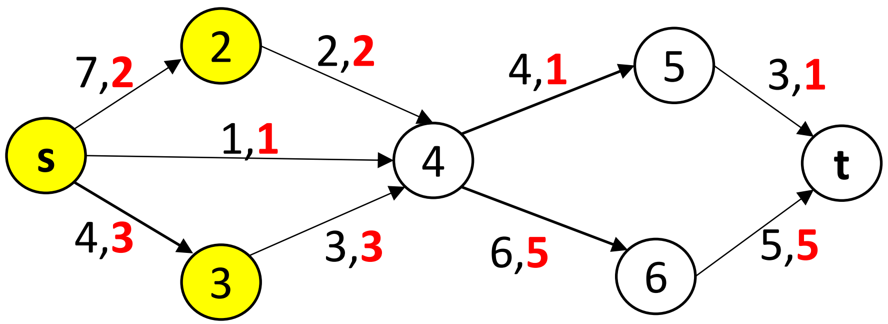


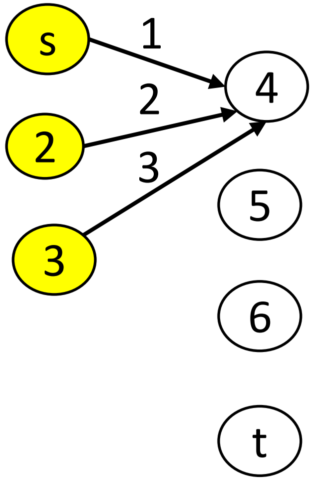
$$c(R) = \sum_{e \in E^+(R)} c(e) - \text{пропускная способность разреза}$$



$c(R) = 14$

Максимальный поток и минимальный разрез





$c(R) = 6$

$M(f) = 6$

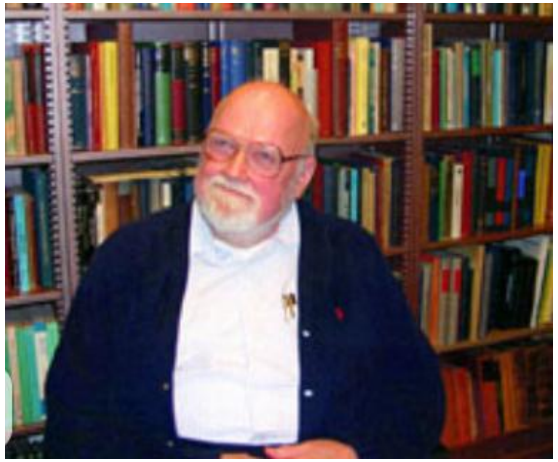
Разрез, пропускная способность которого минимальна, называется **минимальным разрезом.**

Утверждение

Для сети G **величина любого потока *f* не превосходит пропускной способности любого разреза *R*:** $M(f) \leq \mathrm{c}(R)$.

Значит, величина максимального потока не превосходит пропускной способности минимального разреза $M(f^{max}) \leq \mathrm{c}(R^{min})$. Поэтому, если для некоторого потока f справедливо, что $\mathrm{c}(R) = M(f)$, то это будет означать, что f – максимальный поток.

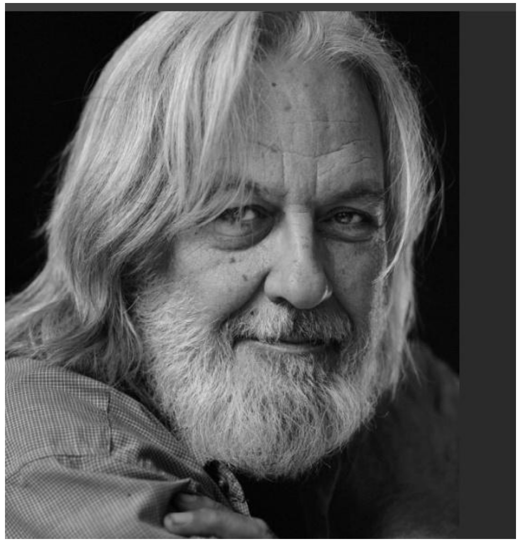
1955 год (**метод Форда-Фалкерсона**)



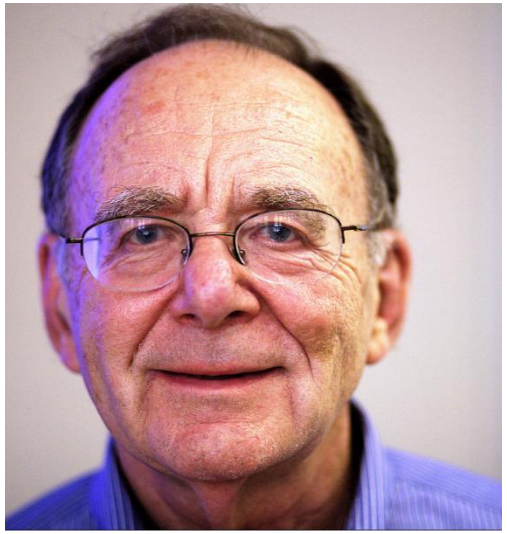
Лестер Рэндольф Форд младший
англ. *Lester Randolph Ford, Jr.*
1927 – 2017
США
Научная сфера - математик

Делберт Рей Фалкерсон
англ. *Delbert Ray Fulkerson*
1924 – 1976
США
Научная сфера - комбинаторика

1972 год (**алгоритм Эдмондса-Карпа**)

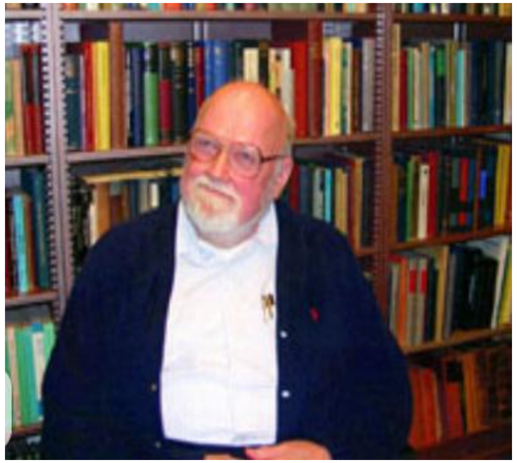


Джек Р. Эдмондс
англ. *Jack Edmonds*
1934
США
Научная сфера - комбинаторная оптимизация



Ричард Мэннинг Карп
англ. *Richard Manning Karp*
1935
США
Научная сфера – теория алгоритмов и биоинформатика

Метод **Форда-Фалкерсона**



Лестер Рэндольф Форд младший

Делберт Рей Фалкерсон

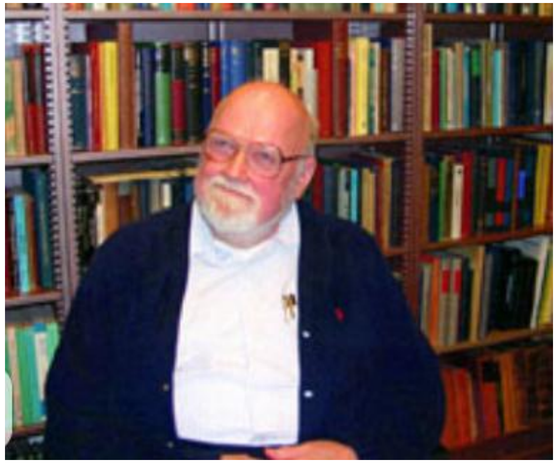
Теорема Форда – Фалкерсона

Пусть f некоторый поток в сети D, тогда следующие утверждения эквивалентны:

(1) f – максимальный поток;

(2) для потока f в сети остаточных пропускных способностей D_f нет увеличивающего (s, t)-пути

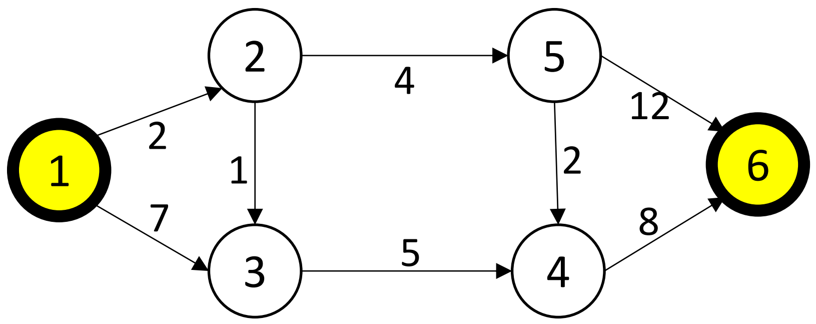
(3) $M(f) = c(R)$ для некоторого разреза R сети.



Лестер Рэндольф Форд младший
англ. *Lester Randolph Ford, Jr.*

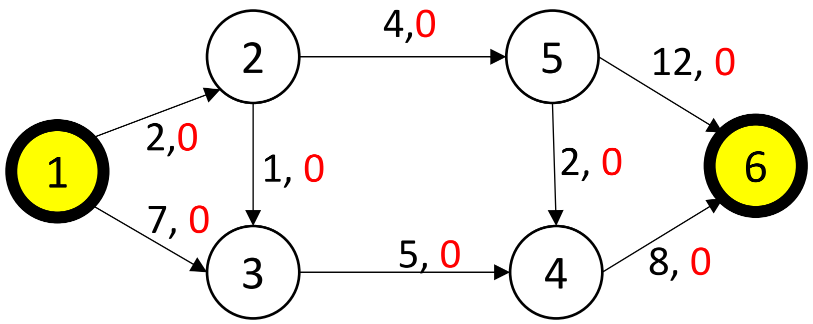
Делберт Рей Фалкерсон
англ. *Delbert Ray Fulkerson*

Найти максимальный поток в сети методом Форда-Фалкерсона

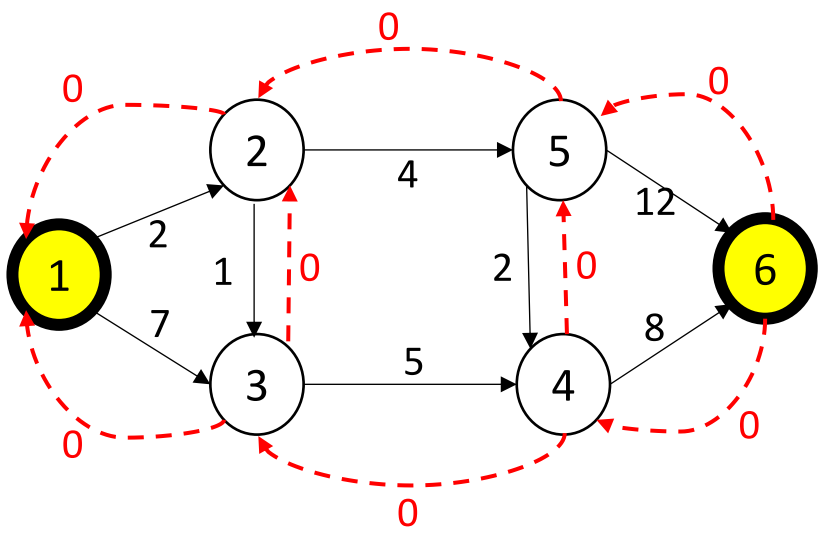


1-я итерация

$M(f) = 0$

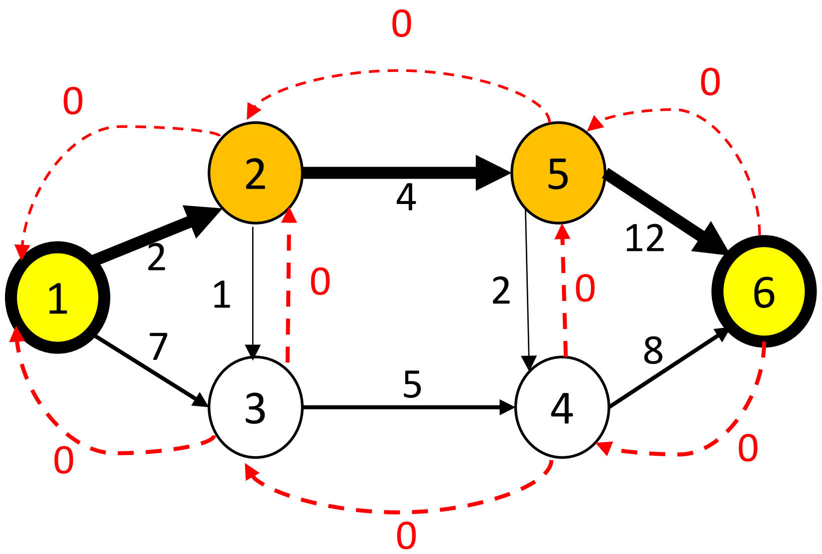


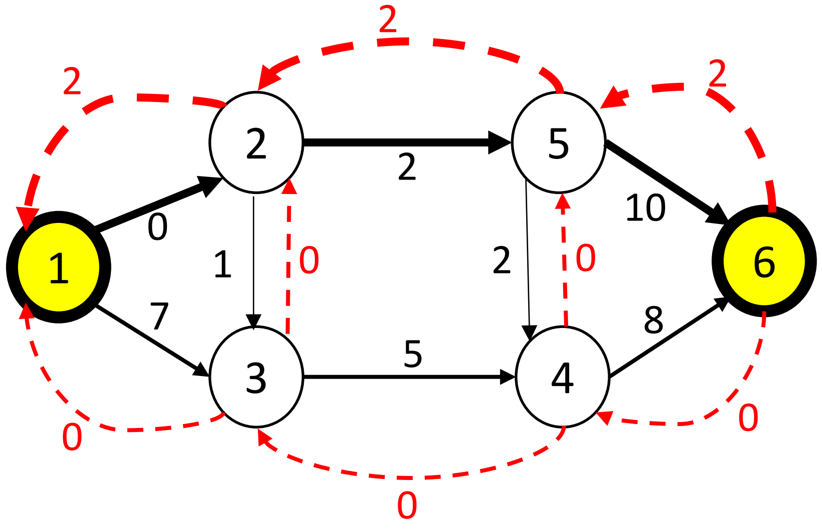
сеть остаточных пропускных способностей



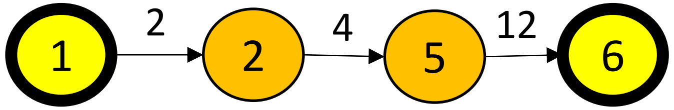
ФПМИ БГУ

1-я итерация (продолжение)





P:

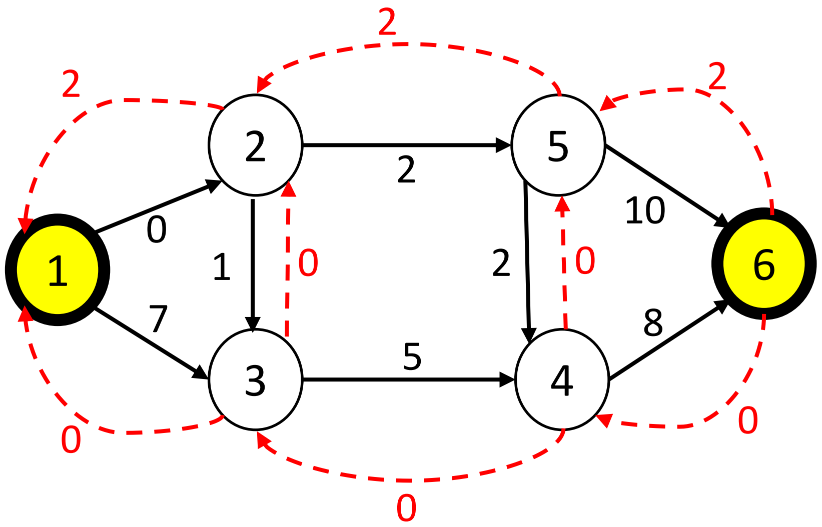


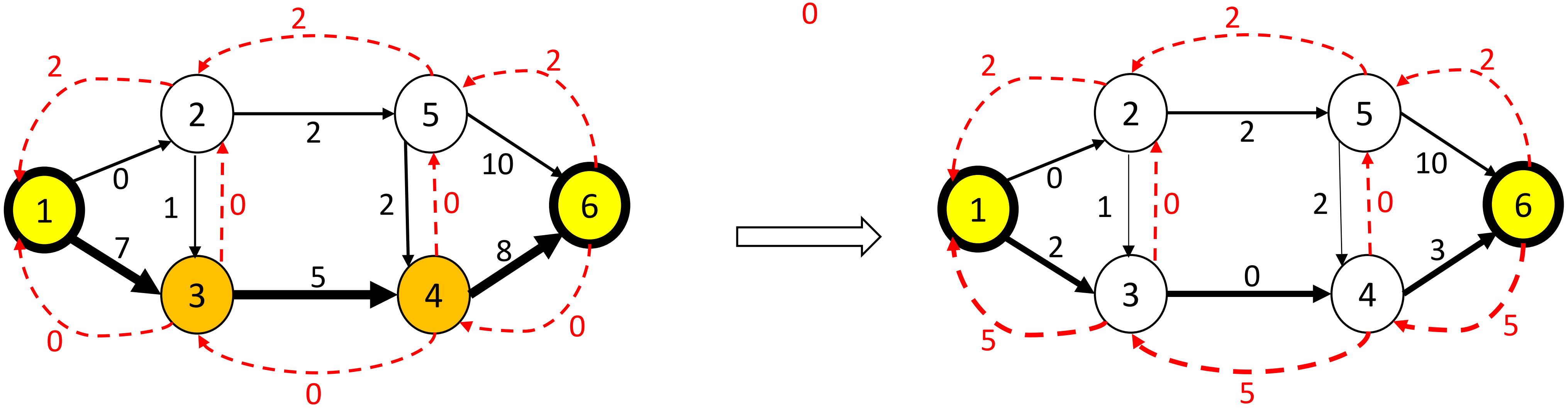
$$M(f) = M(f) + c_f^{\min} = 0 + 2 = 2$$

$c_f^{\min}$ =min {c′(1,2), c′(2,5), c′(5,6)}= min { 2, 4, 12}=2

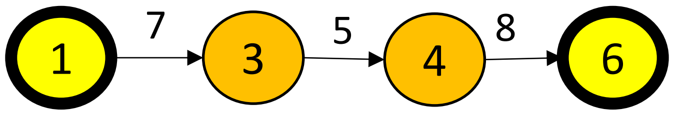
2-я итерация

$M(f) = 2$





P:

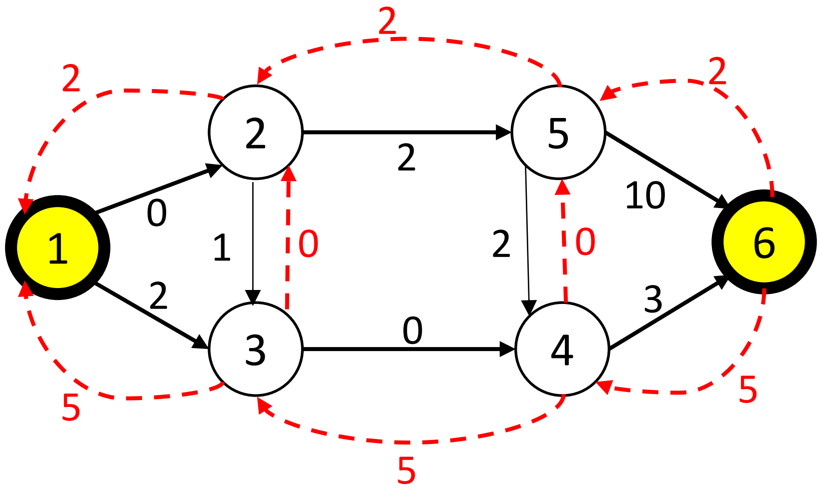


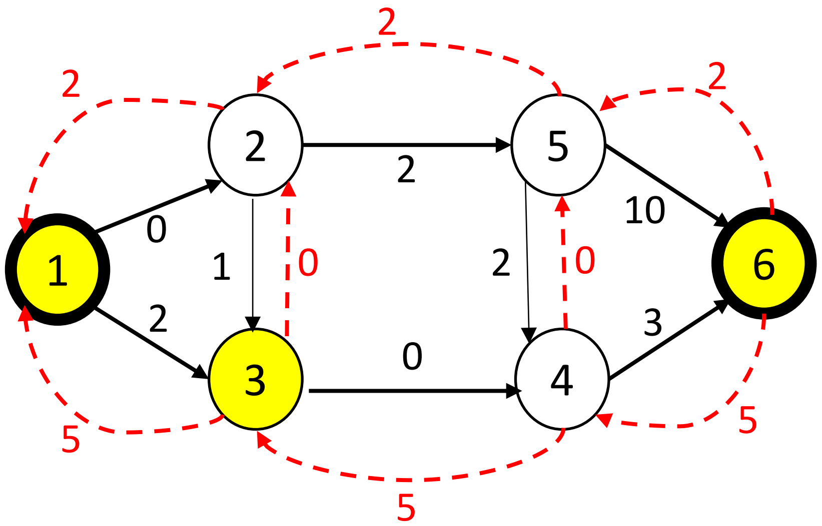
$c_f^{\min}$ =min {c′(1,3), c′(3,4), c′(4,6)}= min { 7, 5, 8}=5

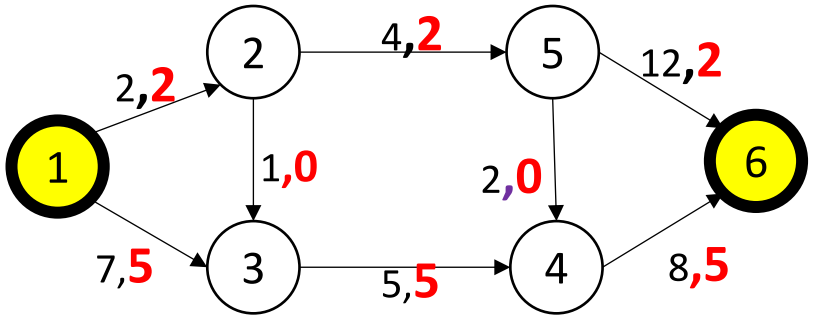
$M(f) = M(f) + c_f^{\min}$=2+5=7

3-я итерация

$$M(f) = 7$$





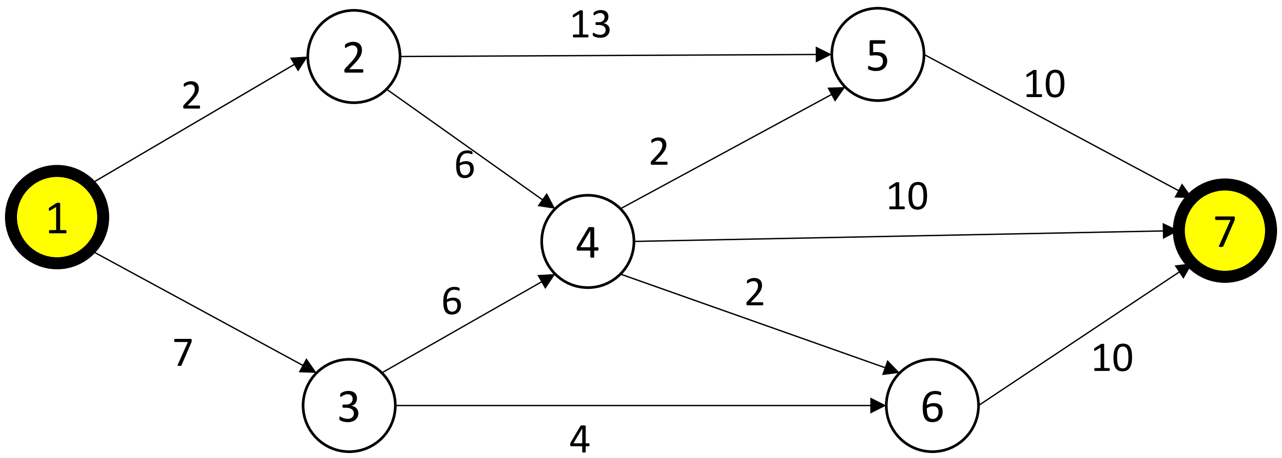


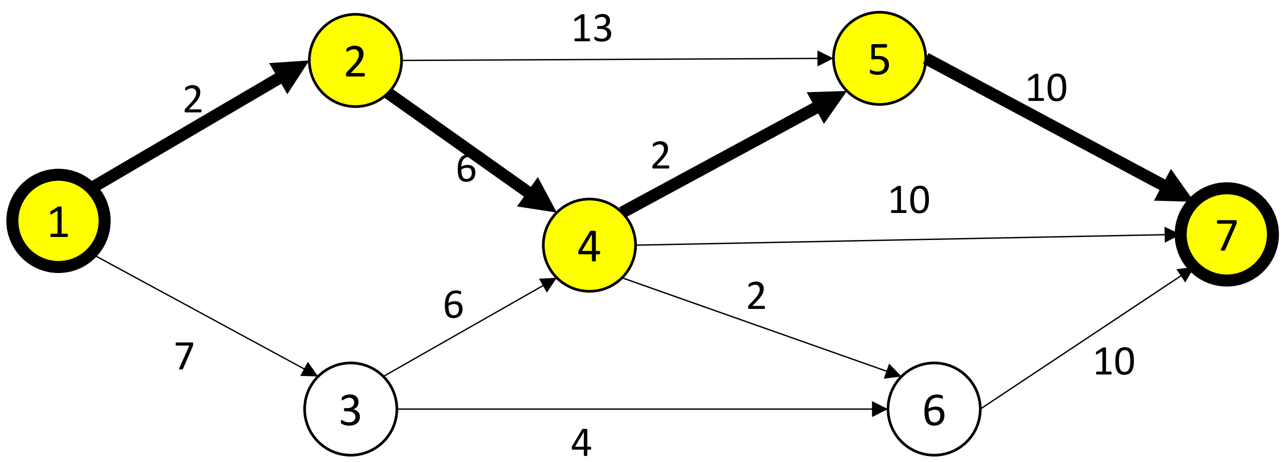
Если для текущего потока f в сети остаточных пропускных способностей D_f не существует увеличивающего (s,t)-пути, то величина потока f равна пропускной способности разреза $R = (X, \overline{X})$ (<u>в множество *X* берём вершину *s* и те вершины, до которых удалось дойти из вершины *s* на последней итерации метода Форда-Фалкерсона </u>).

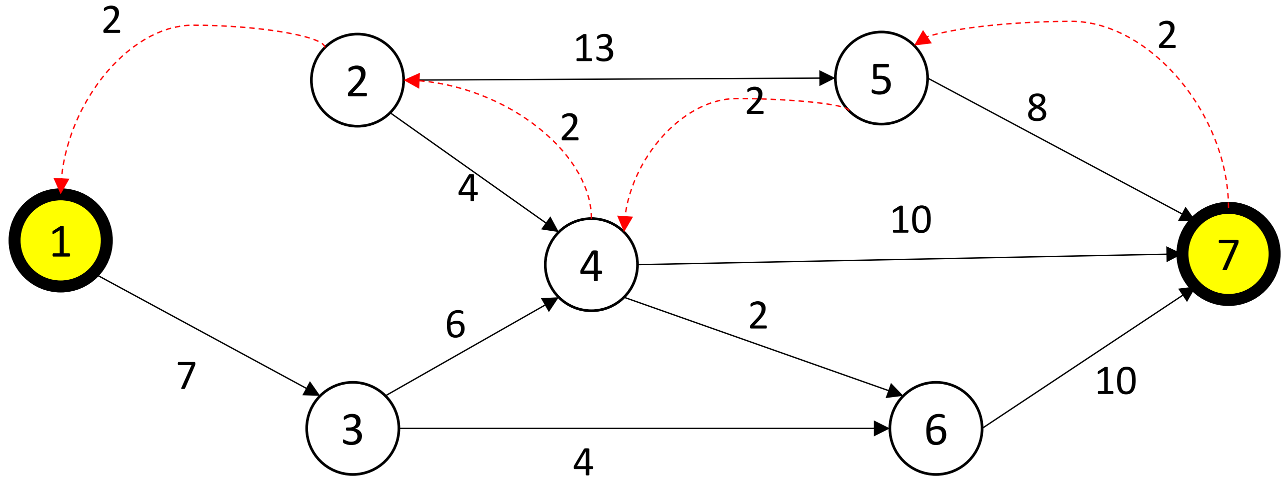
По теореме Форда-Фалкерсона текущий поток f - максимальный.

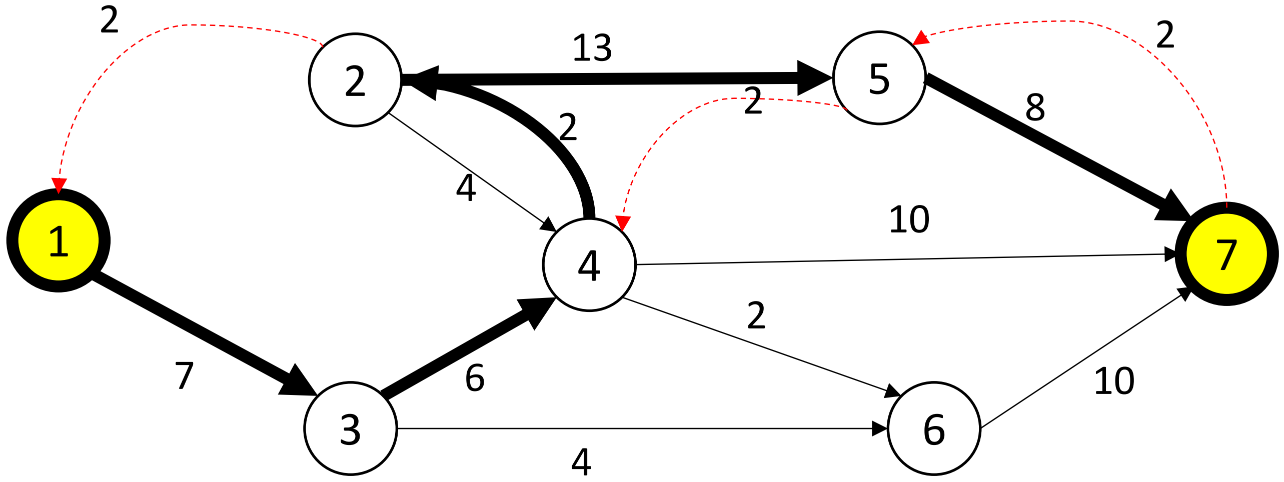
См. доказательство: «Сборник задач по теории алгоритмов : учеб.-метод. пособие» / В. М. Котов [и др.]. – Минск : БГУ, 2017. С. 26-30.

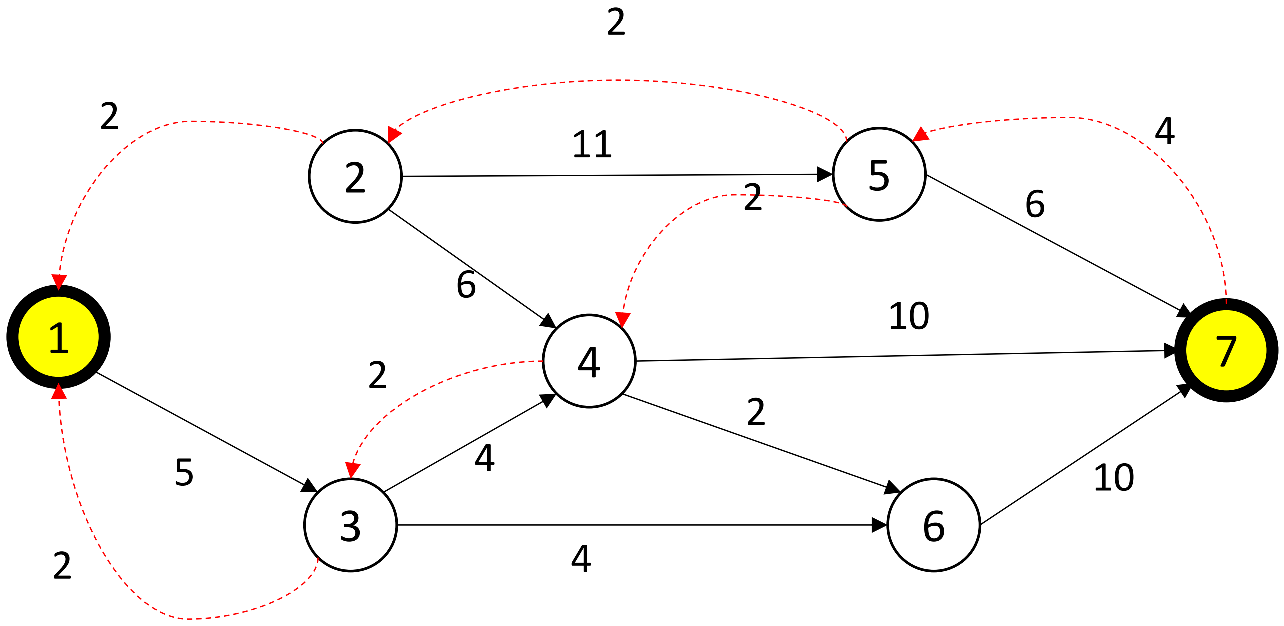
Пример 2 Найти максимальный поток в сети методом Форда-Фалкерсона (на итерация использовать dfs, при неоднозначности идти в вершину с меньшим номером).

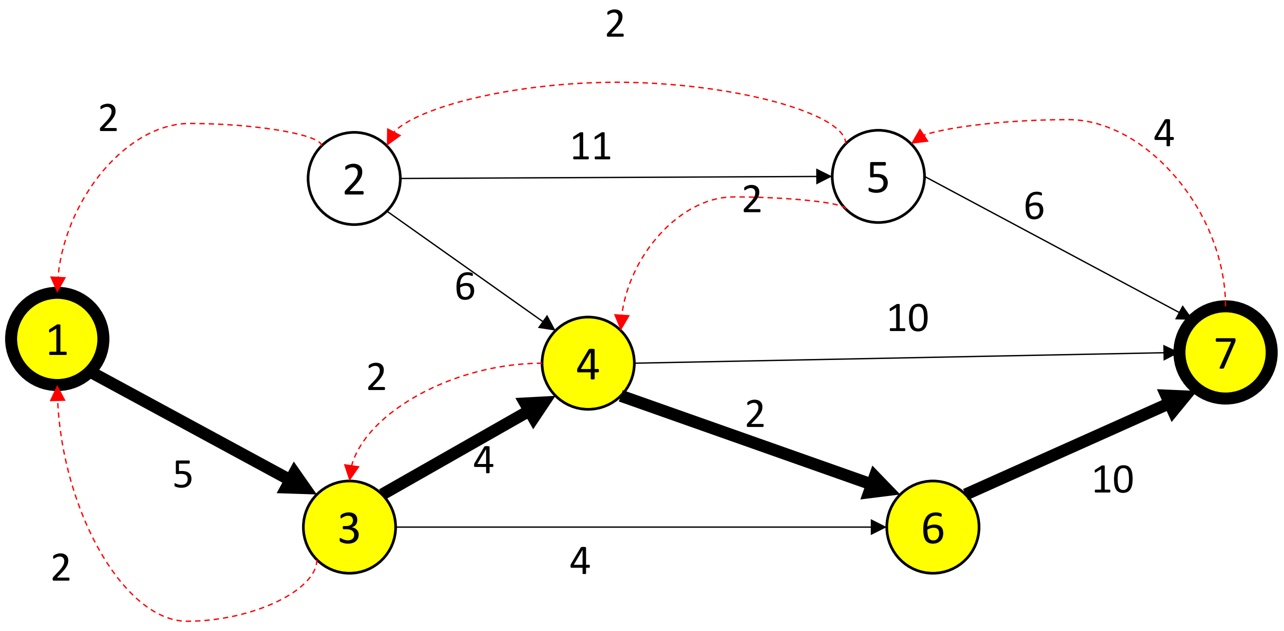


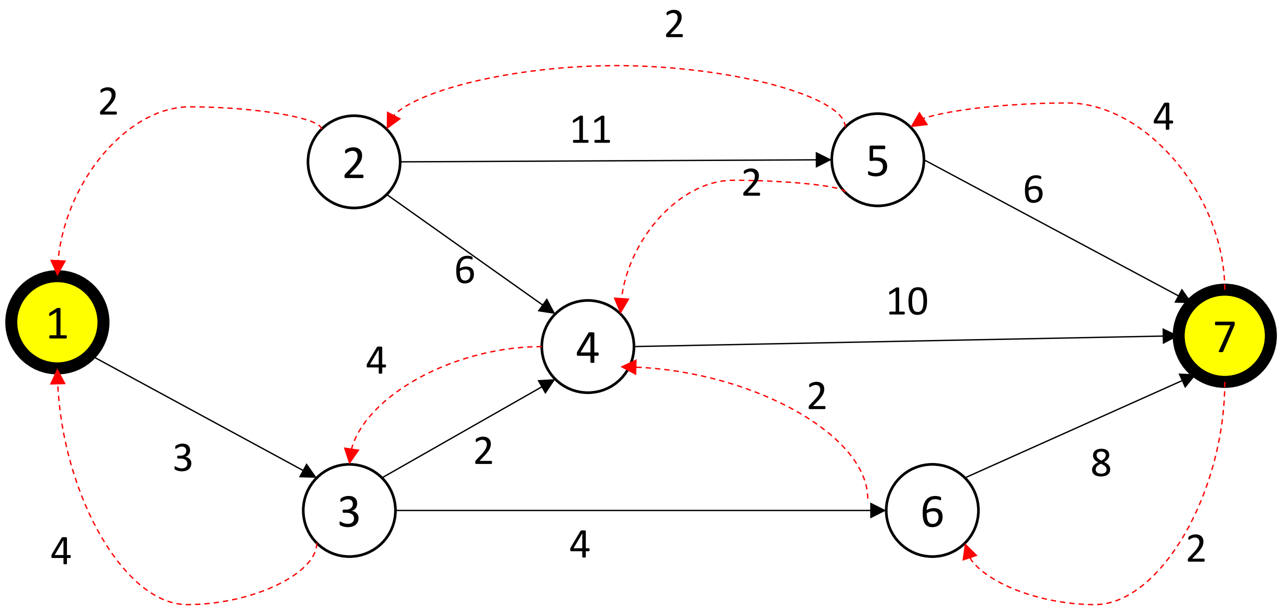

1
2
3
4
5
6
7
2
13
6
2
10
10
7
6
2
4
10

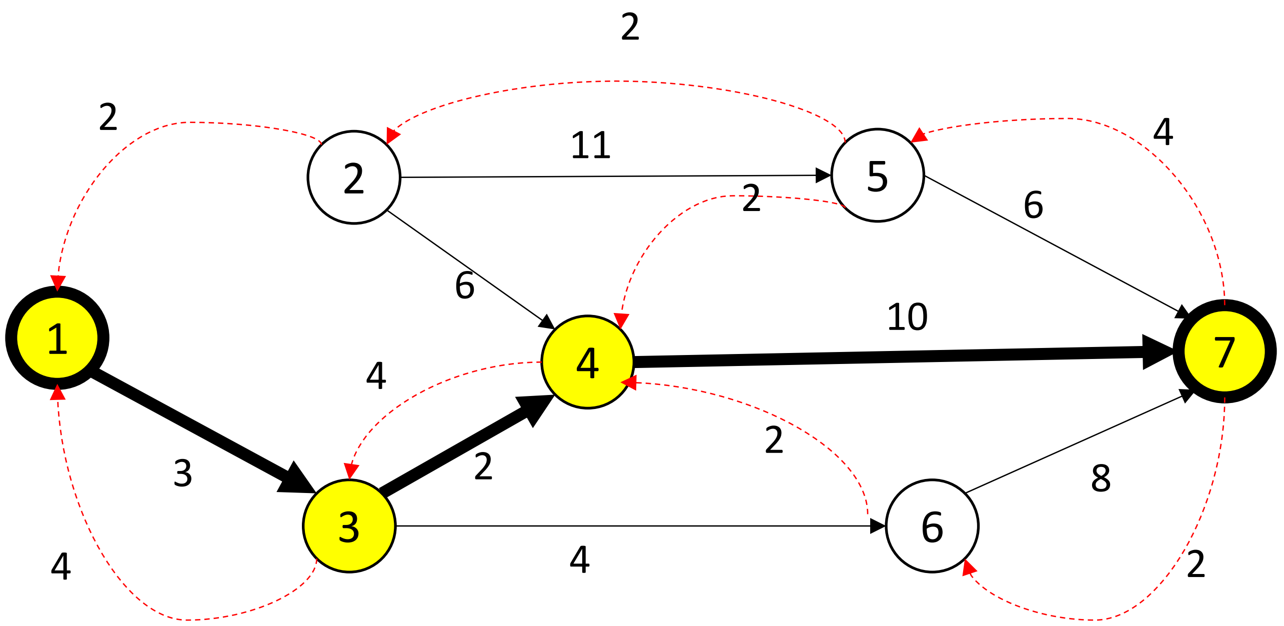

1
2
3
4
5
6
7
2
13
2
2
2
4
8
10
7
6
2
4
10

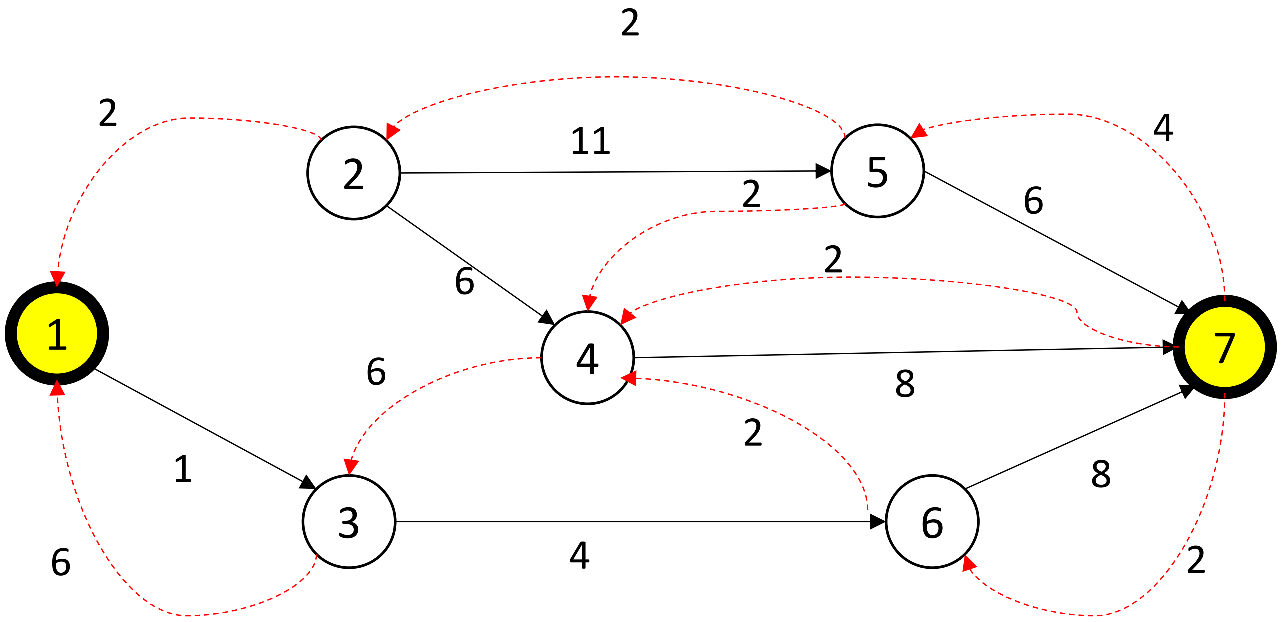

2
2
13
5
2
2
8
2
4
1
4
10
7
2
7
6
10
3
6
4

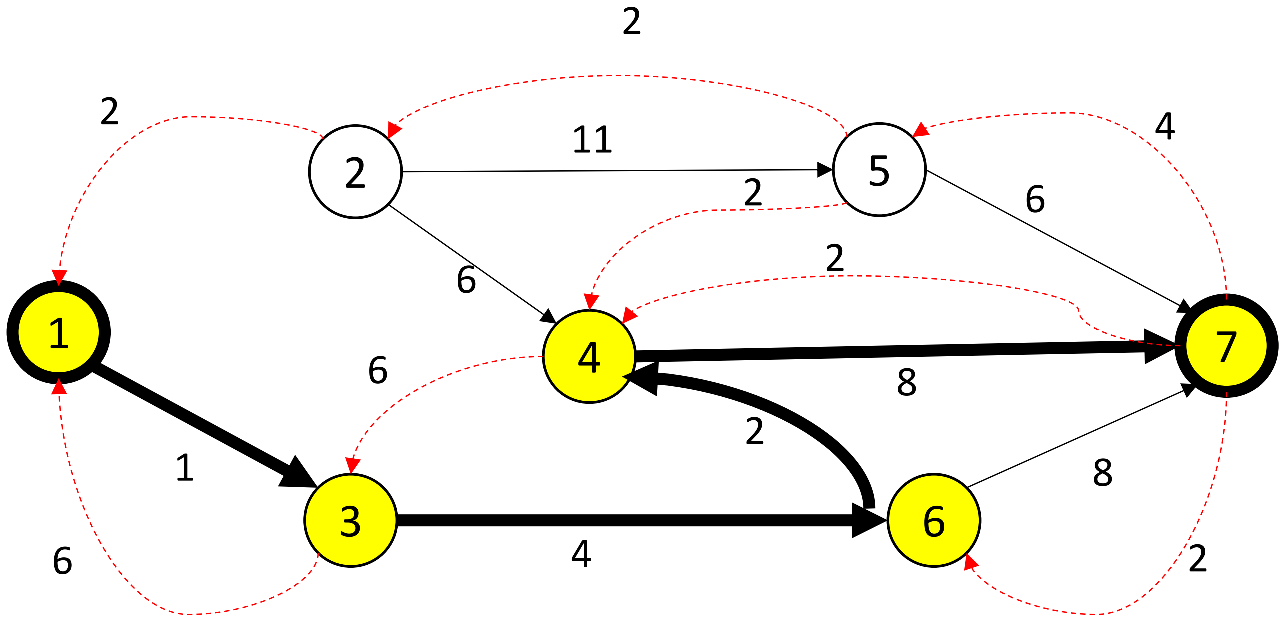

2
2
4
2
11
5
2
6
6
1
4
10
7
2
2
4
5
10
3
6
2
4

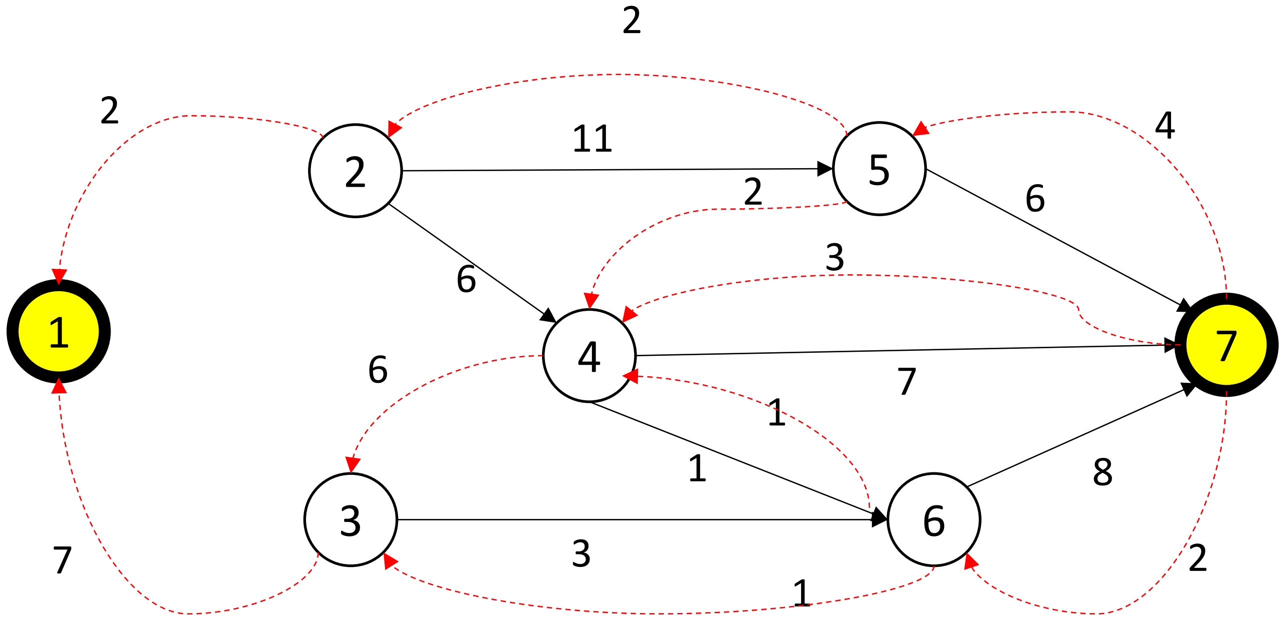

2
2
4
2
11
5
2
6
6
1
4
10
7
2
2
5
3
4
6
10
2
4

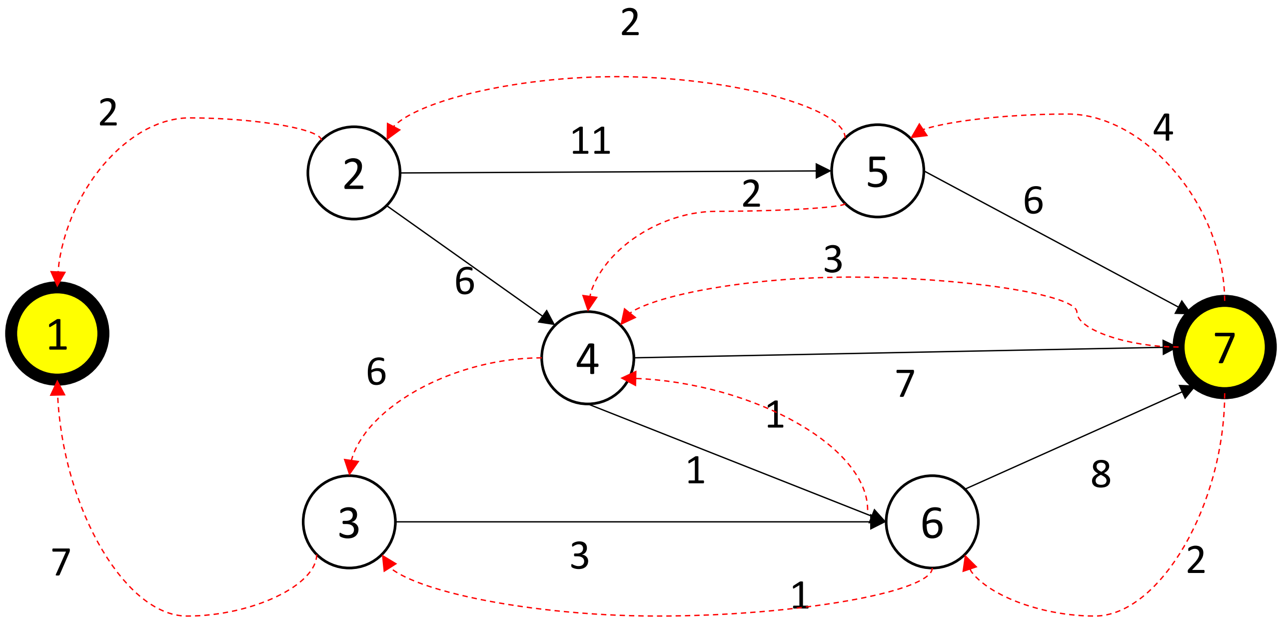

2
2
4
2
11
5
2
6
6
1
4
10
7
4
2
3
2
8
3
6
4
4
2


2
2
2
11
5
4
2
6
6
1
10
7
4
4
3
2
2
3
8
6
4
4
2

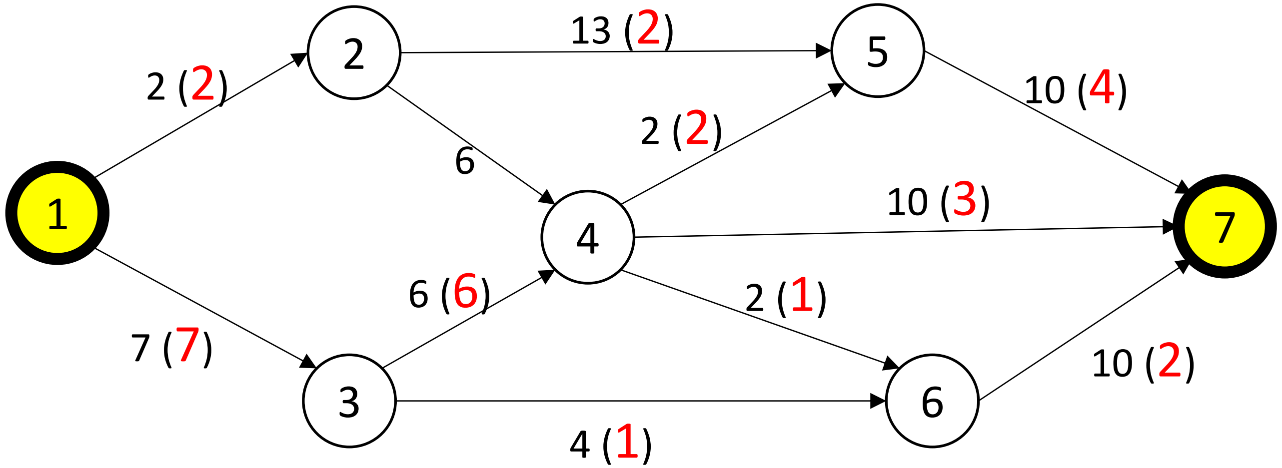

2
2
2
11
5
4
2
6
2
6
1
4
7
6
8
2
1
3
6
8
4
6
2


2
2
11
4
2
5
6
2
6
1
4
7
6
8
2
1
8
3
6
2
6
4


2
2
11
4
2
5
6
3
6
1
4
7
6
7
1
1
8
3
6
7
3
1
2

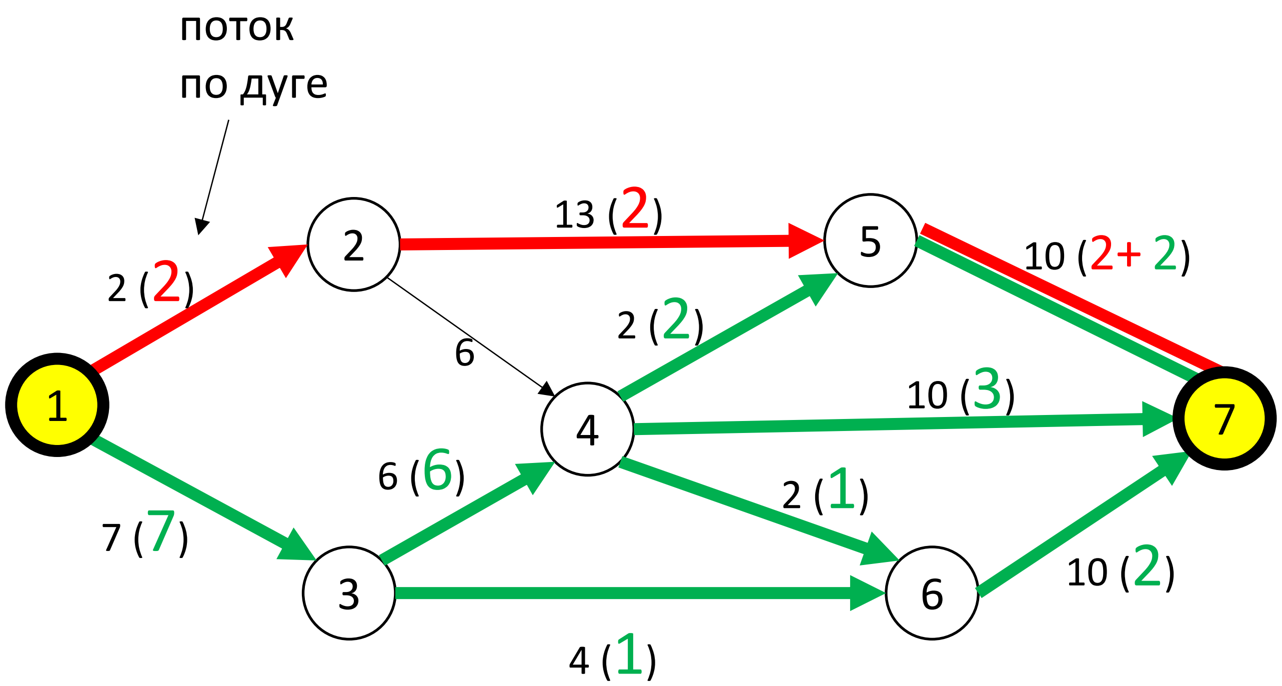


2
2
11
2
4
6
6
3
6
7
1
1
8
7
3
1
2
1 2 3 4 5 6 7



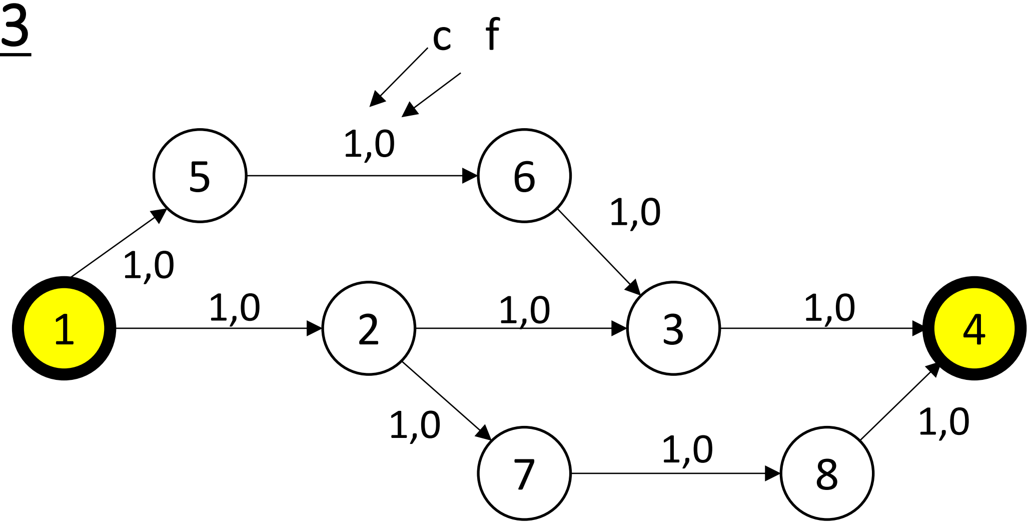
2 (2)
13 (2)
10 (4)
6
2 (2)
10 (3)
6 (6)
2 (1)
7 (7)
10 (2)
4 (1)
1 2 3 4 5 6 7

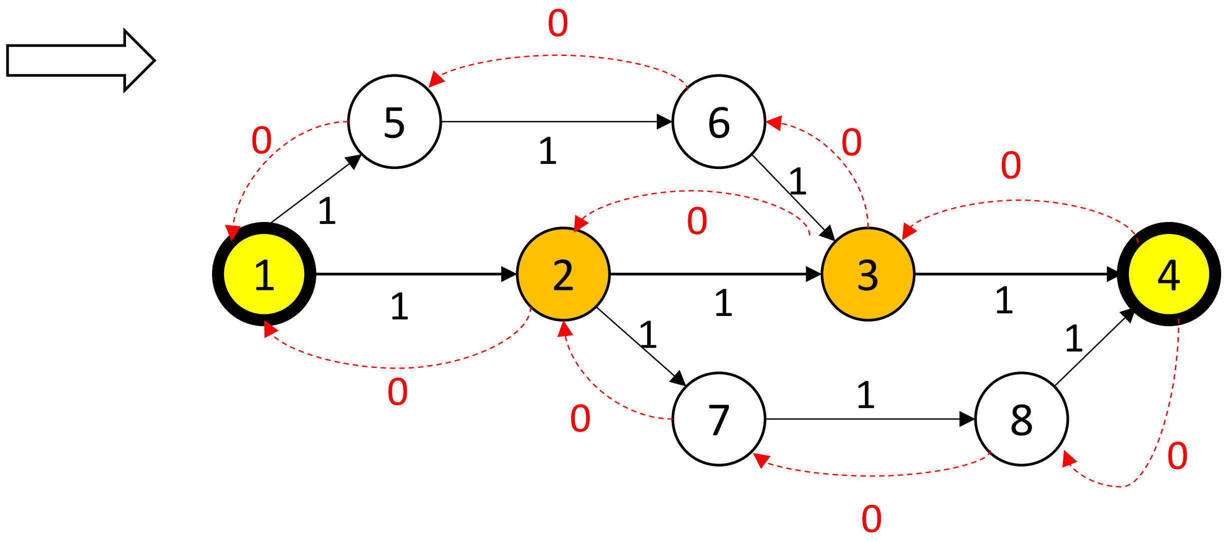
ФПМИ БГУ



Ответ: величина максимального потока равна 9.

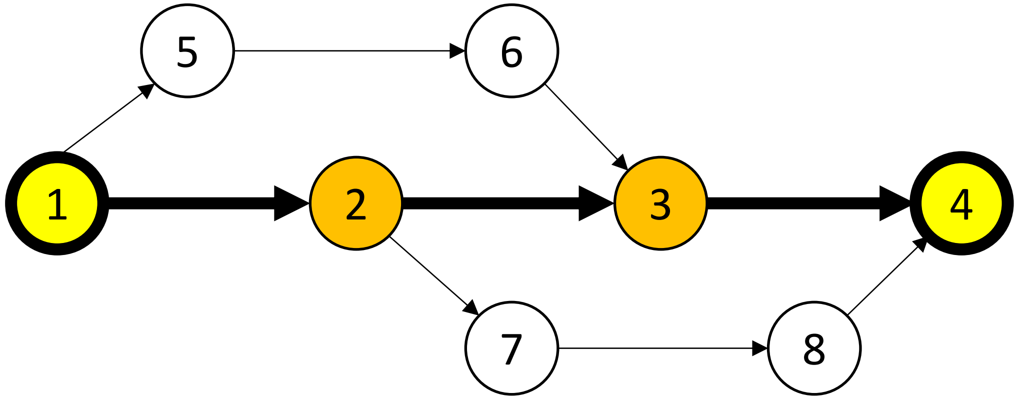
Пример 3

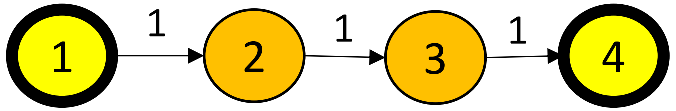


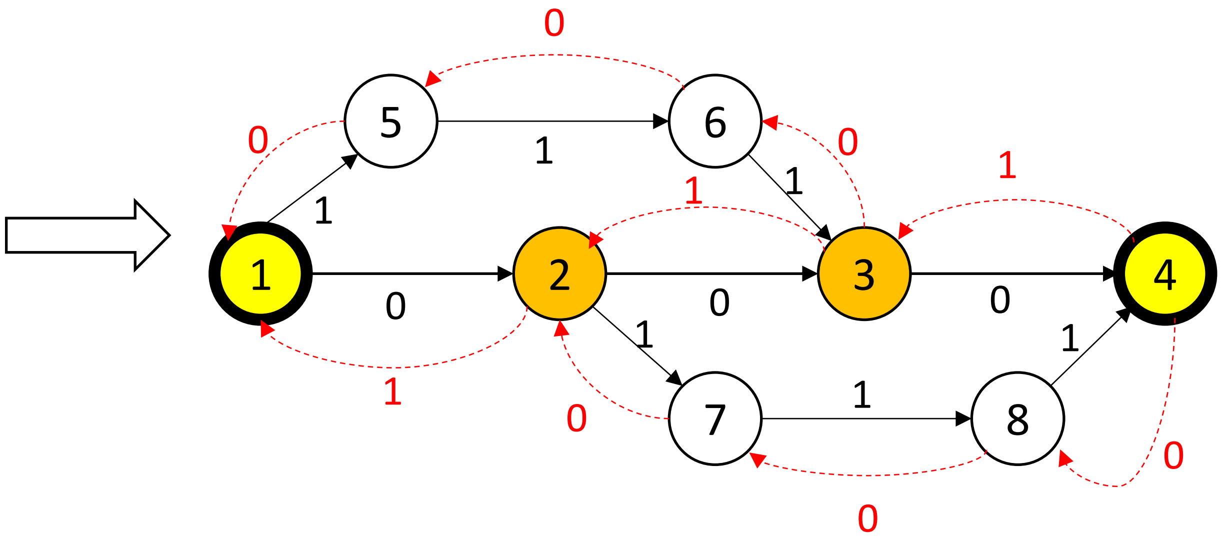


1-я итерация

$M(f) = 0$





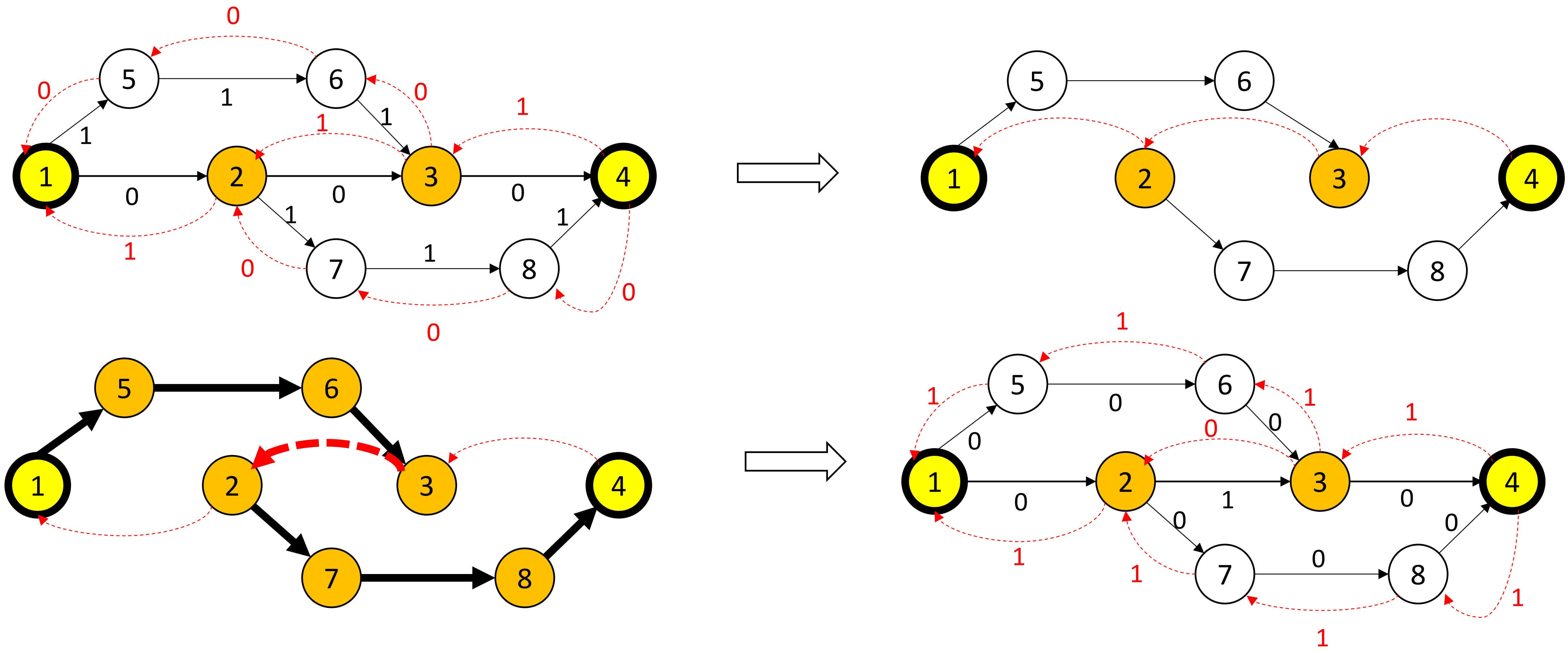


$c_f^{\min}$=1

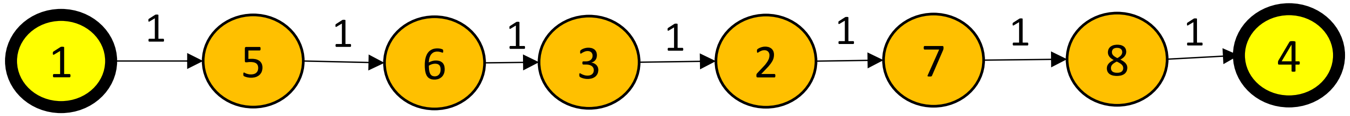
$M(f) = M(f) + c_f^{\min} = 0 + 1 = 1$

2-я итерация

$M(f) = 1$



$$M(f) = M(f) + c_f^{\min} = 1 + 1 = 2$$

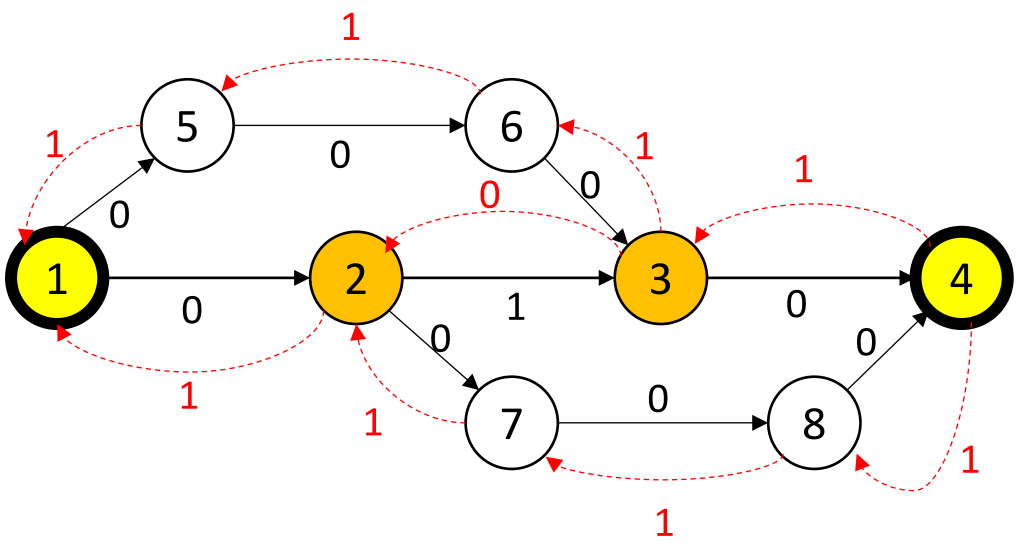


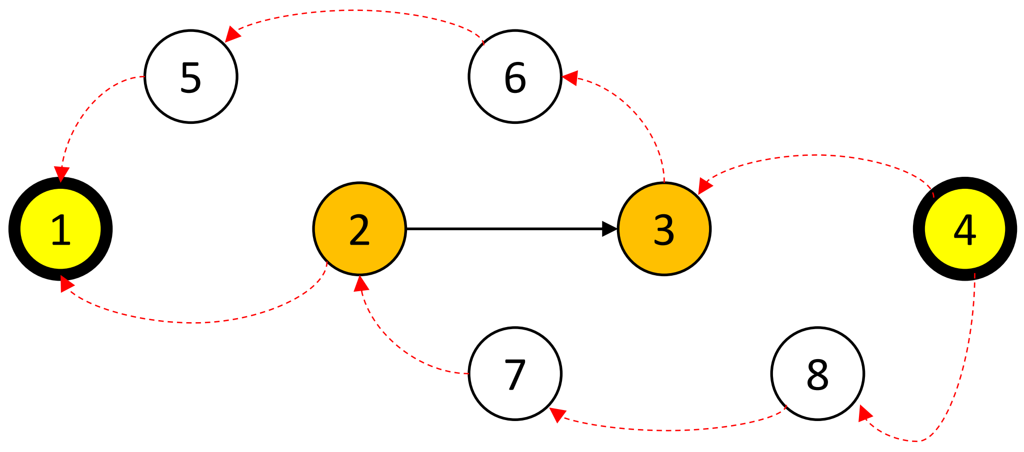
$c_f^{\min}=1$

ФПМИ БГУ

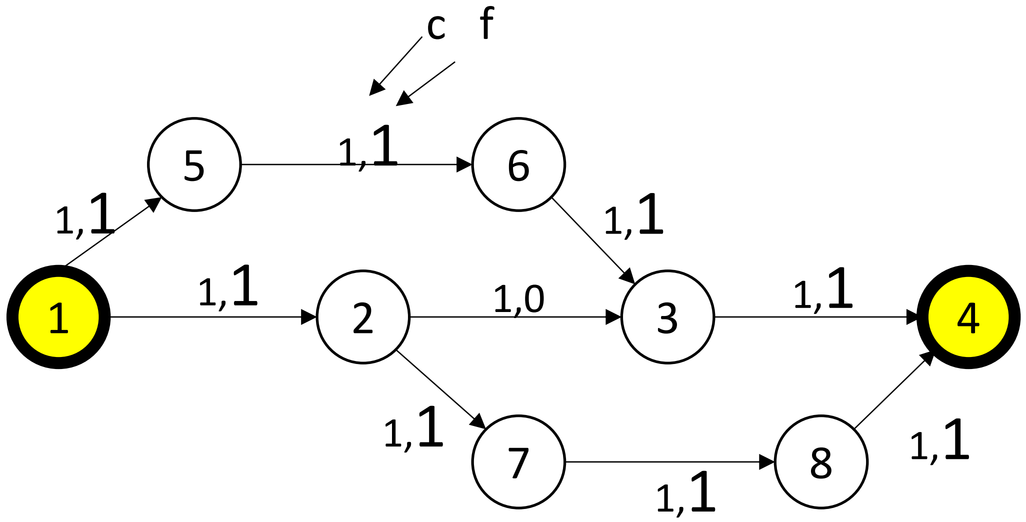
3-я итерация

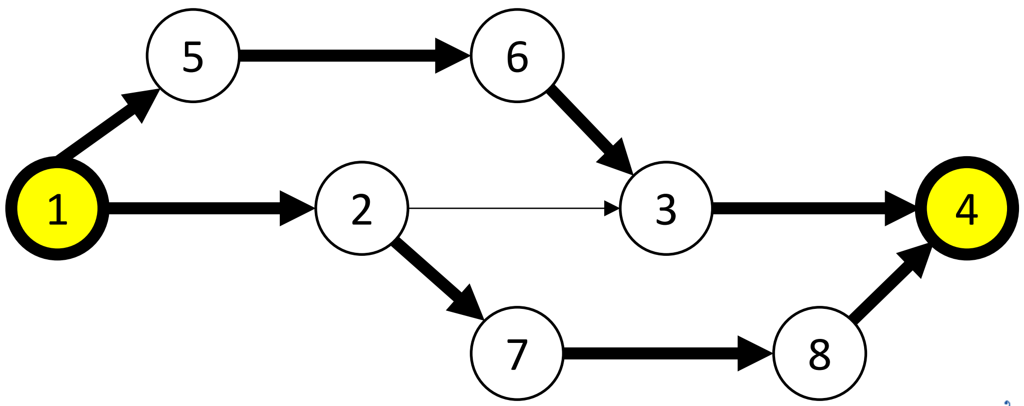
$M(f) = 2$





Восстановление потока:





Напомним наши предположения

1. В дальнейшем мы будем работать с **целочисленными потоками**, то есть все ограничения – целые числа.
2. Считаем, что любая внутренняя вершина сети лежит на некотором (s, t)-пути ($m \geq n - 1$).
3. Предполагаем, что в сети **нет кратных дуг**.

Метод Форда – Фалкерсона

Работает для сетей с целочисленными пропускными способностями дуг.

Время работы: $\mathbf{O}(\, M(f^{max}\,) \cdot ? \,)$, где

$M(f^{max}\,)$ – величина максимального потока

так как поток целочисленный, а в исходной сети (по сделанному ранее предположению) нет кратных дуг, то $M(f^{max}\,) \le \boldsymbol{c^{max}} \cdot \boldsymbol{n}$, где $\boldsymbol{c^{max}}$ – наибольшая из пропускных стоимостей дуг сети.

? – время поиска увеличивающего пути

для поиска увеличивающего пути воспользуемся поиском в глубину (DFS) - $\mathbf{O}(\boldsymbol{n} + \boldsymbol{m})$.

Если на итерациях метода Форда-Фалкерсона используется поиск в глубину, то можно выписать следующую оценку:

$$\mathbf{O}(\mathrm{c}^{\max} \cdot \boldsymbol{n} \cdot \boldsymbol{m})$$

псевдополиномиальный алгоритм

Алгоритми Эдмондса – Карпа

полиномиальный алгоритм

Время работы: $\mathbf{O}(\boldsymbol{n} \cdot \boldsymbol{m} \cdot (\boldsymbol{n} + \boldsymbol{m})) = \mathbf{O}(\boldsymbol{n} \cdot \boldsymbol{m}^2)$,

$\mathrm{O}(n + m)$ – время работы поиска в ширину;

$\mathrm{O}(n \cdot m)$ – число итераций алгоритма;

- Поиск увеличивающего пути: поиск в ширину (**BFS**).
- После каждой итерации алгоритма длина $\boldsymbol{dist(s, v)}$ (в дугах) наименьшего пути из источника $\boldsymbol{s}$ в вершину $\boldsymbol{v}$ монотонно не убывает. Так как длина (s,t)-пути не превосходит $(n-1)$, то конечная вершина t может изменять свою метку $dist(s, t)$ не более, чем n раз.
- Назовем $\boldsymbol{k}$**-этапом** совокупность итераций, на которых длина наименьшего пути сохраняется равной $\boldsymbol{k}$. Эти итерации идут подряд. На k-этапе после каждой итерации алгоритма из новой сети вычёркивается хотя бы одна дуга, построенного на этой итерации увеличивающего пути и она не может появиться вновь, так как не является обратной дугам следующих увеличивающих путей $\boldsymbol{k}$-этапа. Поэтому число итераций алгоритма на $\boldsymbol{k}$-этапе не превосходит $\boldsymbol{m}$.
- Получаем оценку на число итераций $\mathbf{O}(\boldsymbol{n} \cdot \boldsymbol{m})$.

Метод Форда – Фалкерсона

псевдополиномиальный алгоритм

работает для сетей с целочисленными пропускными способностями дуг

$$\mathbf{O}(\mathrm{c}^{\max} \cdot \boldsymbol{n} \cdot \boldsymbol{m})$$

Алгоритми Эдмондса – Карпа

полиномиальный алгоритм

$$\mathbf{O}(\boldsymbol{n} \cdot \boldsymbol{m}^{2})$$

Представление сети остаточных пропускных способностей на списках смежности

<u>списки смежности для исходной сети</u>

g

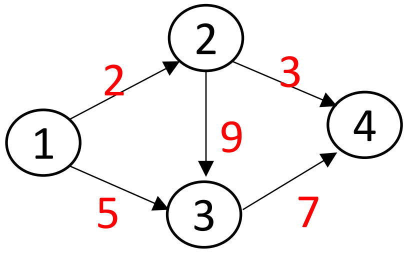
u c_{uv} v

1: [(2,2) , (5,3)]

2: [(9,3) , (3,4)]

3: [(7,4)]

4: [ ]



<u>списки дуг</u>

flow_edges

u c_{uv} f_{uv} v

0: (1,2,0, 2)

1: (2,0,0, 1)

2: (1,5,0, 3)

3: (3,0,0, 1)

4: (2,9,0, 3)

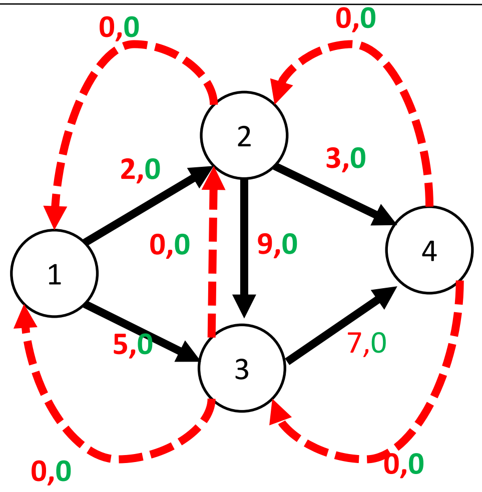
5: (3,0,0, 2)

6: (2,3,0, 4)

7: (4,0,0, 2)

8: (3,7,0, 4)

9: (4,0,0, 3)



остаточная пропускная способность:

$c'_{uv} = c_{uv} - f_{uv}$

<u>списки смежности для остаточной сети</u>

network

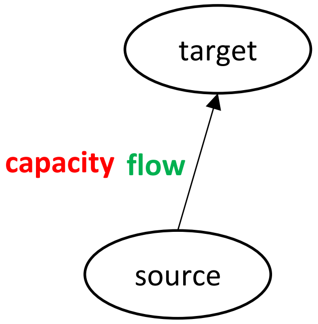
1: [0, 2]

2: [1, 4, 6]

3: [3, 5, 8]

u 4: [7, 9]

```python
@dataclass
class Edge:
    source: int
    target: int
    capacity: int
    flow: int
```




```python
network = [[] for v in range(n)]
flow_edges = []

def build_network():
    for v in range(n):
        for cu, u in g[v]:
            network[v].append(len(flow_edges))
            flow_edges.append(Edge(source=v, target=u, capacity=cu, flow=0))
            network[u].append(len(flow_edges))
            flow_edges.append(Edge(source=u, target=v, capacity=0, flow=0))
```

Псевдокод функций для работы с сетью остаточных пропускных способностей

- `build_network` для построения остаточной сети на основе исходного графа
- `source` для получения начальной вершины ребра в остаточной сети
- `target` для получения конечной вершины ребра в остаточной сети
- `available` для получения остаточной пропускной способности ребра
- `flow` для получения величины потока, пропущенного по ребру
- `edges` для получения исходящих из вершины ребер в остаточной сети
- `push` для увеличения потока вдоль ребра в остаточной сети

```
network = [[] for v in range(n)]
flow_edges = []

def build_network():
    for v in range(n):
        for cu, u in g[v]:
            network[v].append(len(flow_edges))
            flow_edges.append(Edge(source=v, target=u, capacity=cu
            network[u].append(len(flow_edges))
            flow_edges.append(Edge(source=u, target=v, capacity=0,
```

https://github.com/larandaA/alg-ds-snippets

```
def edges(v):
    return network[v]

def available(e):
    edge = flow_edges[e]
    return edge.capacity - edge.flow

def flow(e):
    edge = flow_edges[e]
    return edge.flow

def target(e):
    edge = flow_edges[e]
    return edge.target

def source(e):
    edge = flow_edges[e]
    return edge.source
```

```
def push(e, flow):
    edge = flow_edge[e]
    edge.flow += flow

    edge = flow_edge[e ^ 1]
    edge.flow -= flow
```

```
def ford_fulkerson(s, t):
    result_flow = 0

    while True:
        for v in range(n):
            visited[v] = False
            pred[v] = None

        find_path(s)
        if not visited[t]:
            break

        path = restore_path(t)
        flow = path_capacity(path)
        push_path(path, flow)
        result_flow += flow

    return result_flow

def max_flow(s, t):
    if s == t:
        return None
    build_network()
    return ford_fulkerson(s, t)
```

Общая схема метода Форда – Фалкерсона

dfs

```
def find_path(v):
    visited[v] = True
    for e in edges(v):
        u = target(e)
        if not visited[u] and available(e) > 0:
            pred[u] = e
            find_path(u)
```

bfs

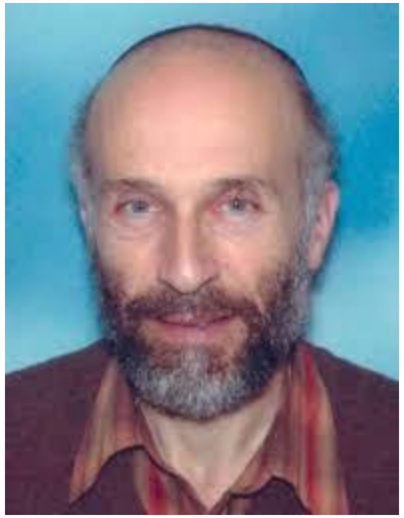
```
def find_path(s):
    q = queue()

    visited[s] = True
    q.enqueue(s)

    while not q.empty():
        v = q.dequeue()

        for e in edges(v):
            u = target(e)
            if not visited[u] and available(e) > 0:
                visited[u] = True
                pred[u] = e
                q.enqueue(u)
```

https://github.com/larandaA/alg-ds-snippets

1970 год

Алгоритм Диница

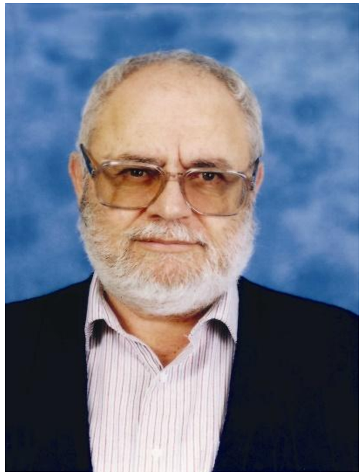
$$O(n^2 \cdot m)$$



Диниц Ефим Абрамович
Yefim Dinic
израильский (бывший советский) ученый

1979 год

Шимон **Ивен**
и его ученик Алон **Итаи**



Shimon **Even**

Alon **Itai**

доработали алгоритм, применяя идею *блокирующего потока* (англ. blocking flow) Александра **Карзанова** (1974 г.)

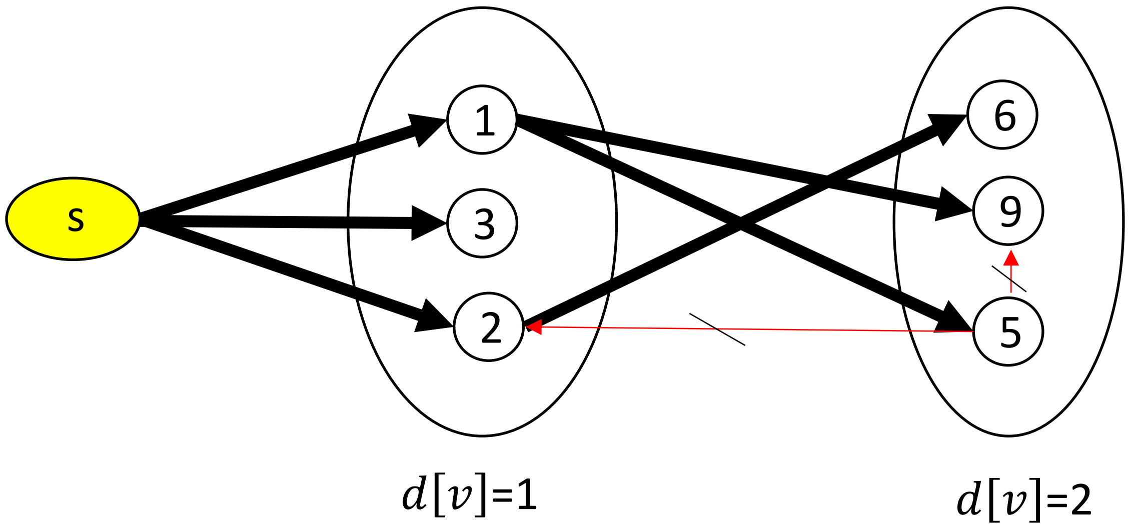
Alexander **Karzanov**

и переформулировали данный алгоритм в том виде, в котором он сейчас используется.

1972 год
Алгоритм Эдмонса-Карпа
$O(n \cdot m^2)$

1) На каждой итерации алгоритма построения максимального потока по сети остаточных пропускных способностей для текущего потока f строится **вспомогательная сеть** D_f.

Пусть $d[v]$ - длина (в дугах) наименьшего $(s, v) -$ пути в сети остаточных пропускных способностей для текущего потока f.
Тогда во вспомогательную сеть D_f войдут только те дуги (v, w) сети остаточных пропускных способностей, для которых $d[w]$= $d[v]$+1.
Сформировать метки $d[v]$ можно алгоритмом **BFS**, а во вспомогательную сеть войдут дуги, которые идут от вершины с меньшей меткой в вершину с большей меткой, т.е. исключаются дуги сети остаточных пропускных способностей между вершинами с одинаковыми метками, а также идущие из вершины с большей меткой в вершину с меньшей меткой.

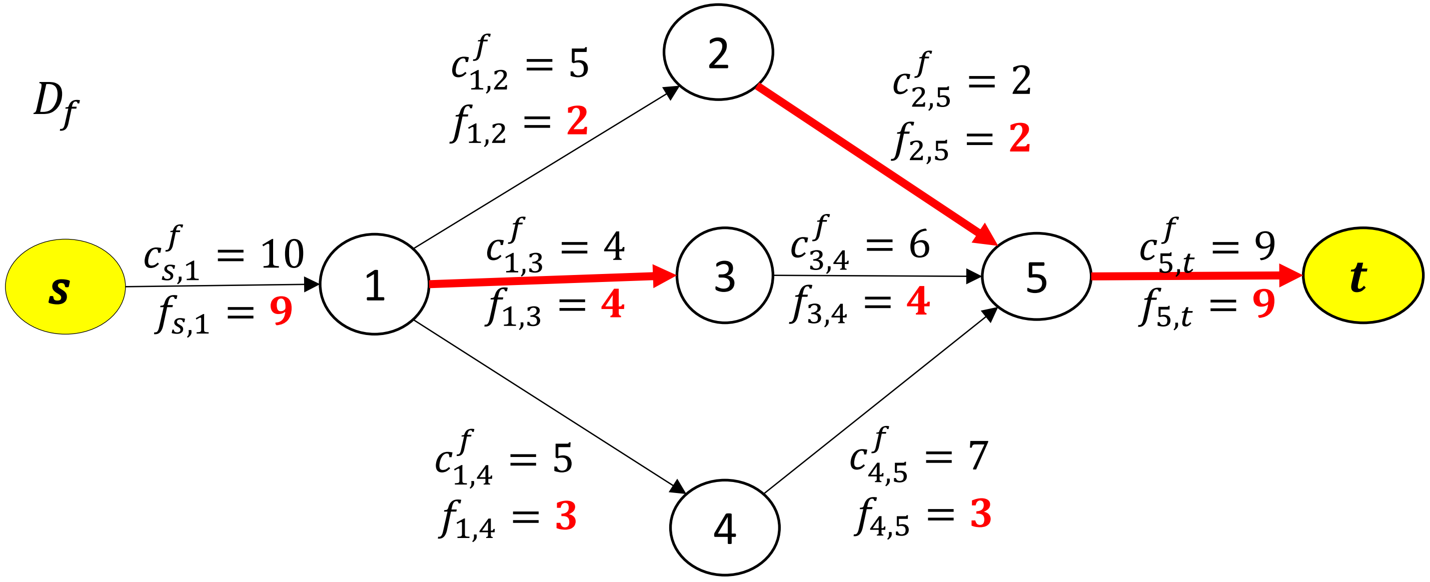


2) Во вспомогательной сети D_f находим **блокирующий поток** f_b – такой поток, что любой (s,t) – путь содержит не менее одной насыщенной потоком дуги.

Выполнить эти действия можно, используя, алгоритм поиска в глубину (**DFS**).

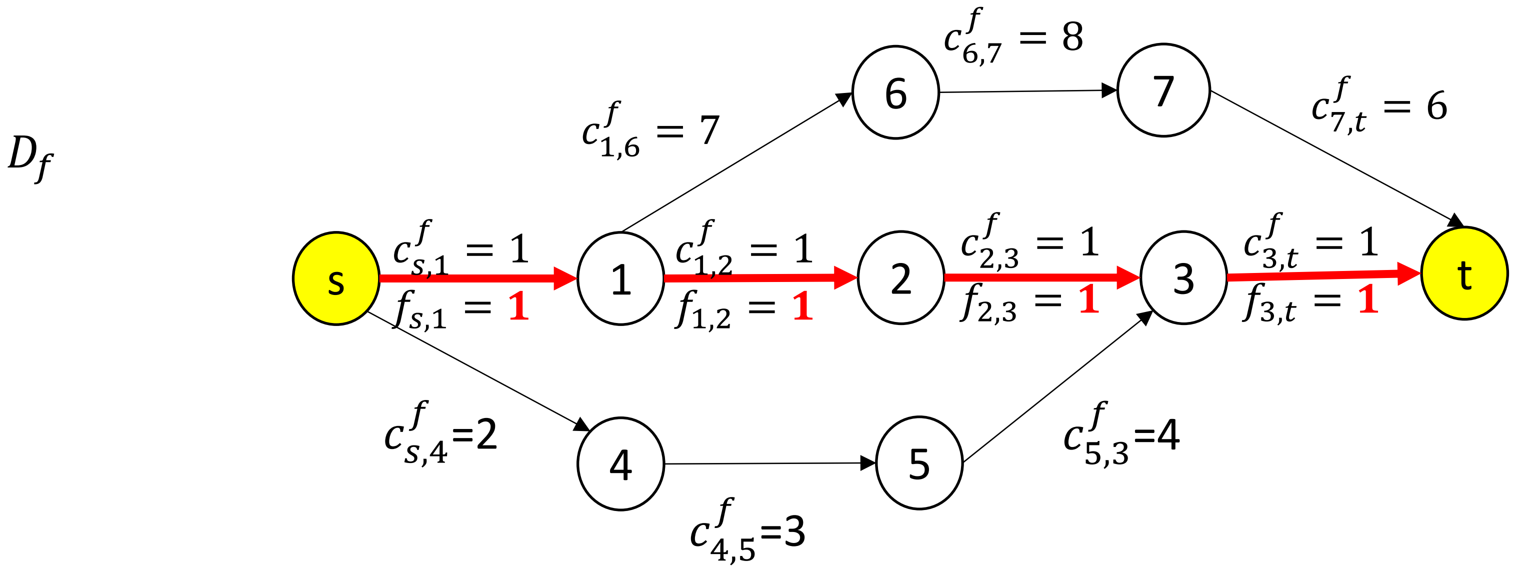
Алгоритм поиска в глубину запускается во вспомогательной сети D_f из вершины s до тех пор, пока существует увеличивающий (s,t) – путь (после того, как найден увеличивающий путь, корректируем вдоль этого пути остаточные пропускные способности c^f и снова запускаем DFS).

Для каждой вершины v сети D_f можно хранить список выходящих дуг. Если при исследовании некоторой дуги (v,w) по ней не был достигнут сток t, то дугу (v,w) нужно «заблокировать» и при последующих запусках поиска в глубину из вершины v исследовать выходящие дуги, которые следуют в списке за дугой (v,w), т.е. поддерживать для каждой вершины v указатель на первую незаблокированную дугу, выходящую из v.



$M(f_b) = 9$

Блокирующий поток f_b не обязательно будет максимальным.
Например, для вспомогательной сети максимальный поток равен 2, а найденный блокирующий поток f_b имеет величину, равную 1.



3) После того, как найден некоторый блокирующий поток f_b, добавляем его к текущему потоку, перестраиваем сеть остаточных пропускных способностей и повторяем алгоритм Диница.

Таким образом текущий поток увеличивается не вдоль одного (s, t) − пути, а сразу вдоль целого набора наименьших (s, t) −путей.

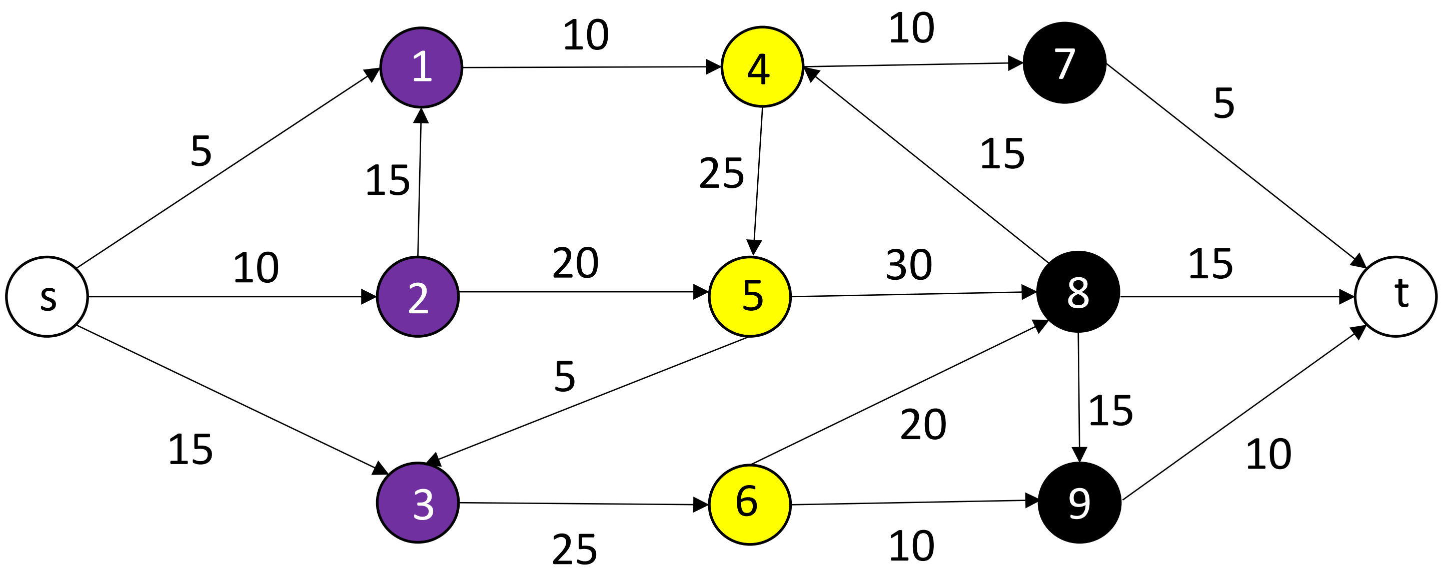
Если на некоторой итерации не существует блокирующего потока, то текущий поток по теореме Форда-Фалкерсона является максимальным.

Пример.

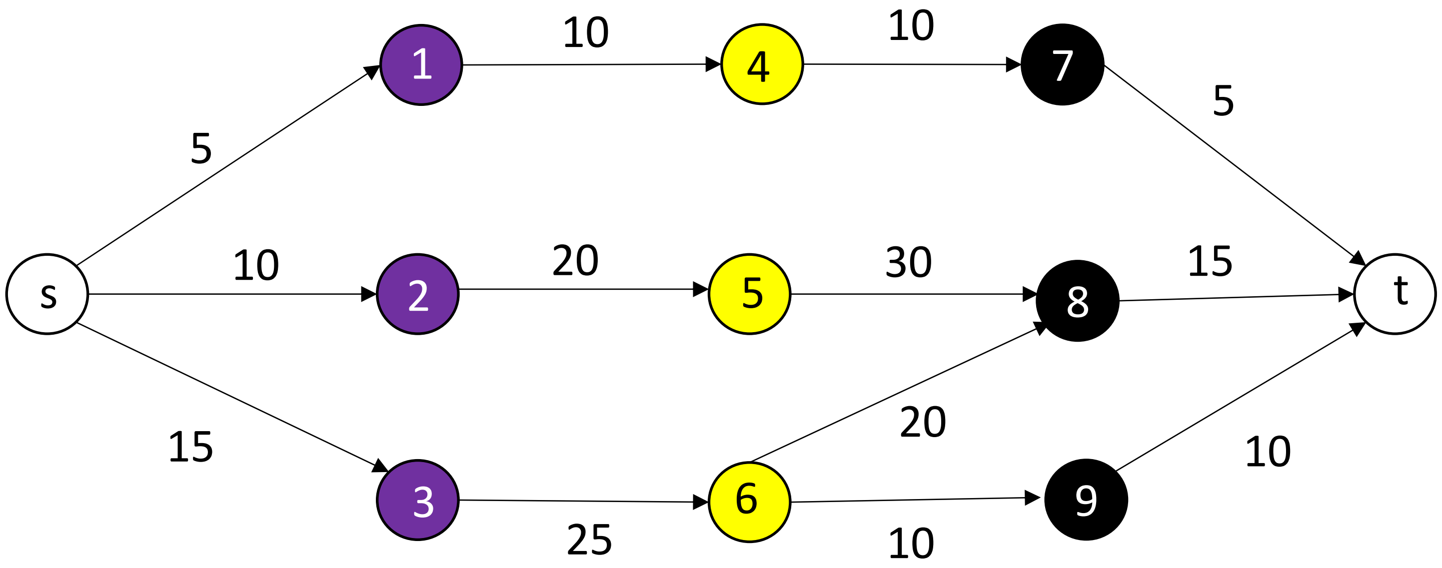
1-й слой

2-й слой

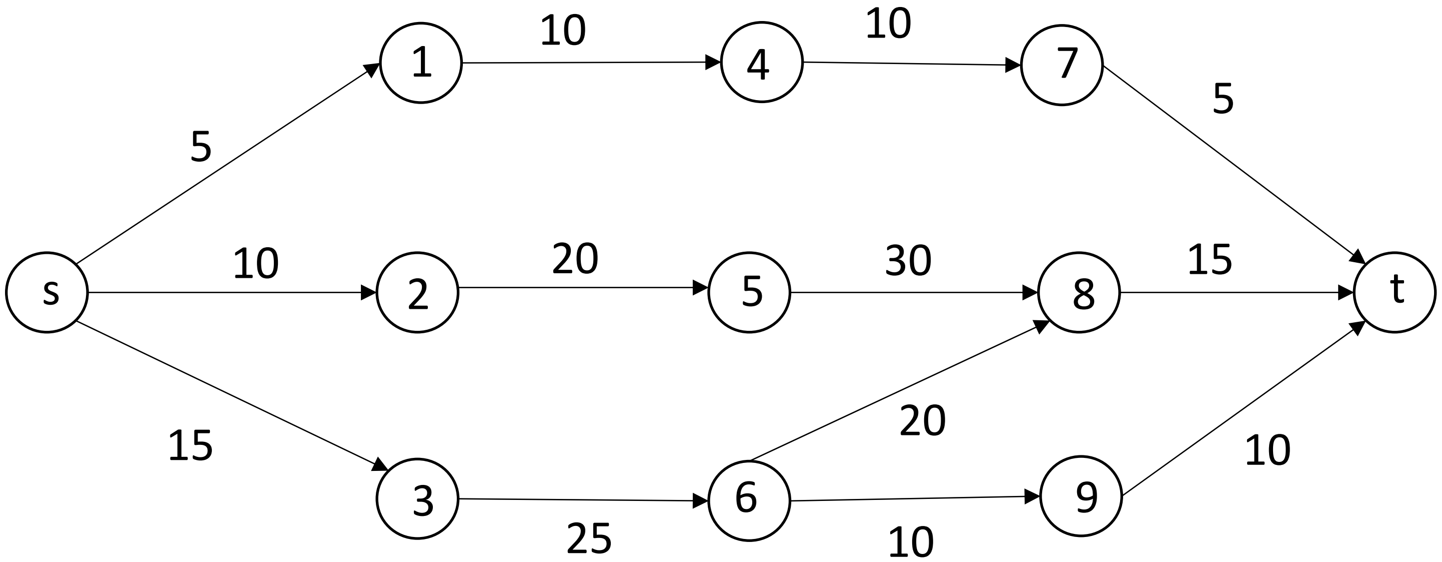
3-й слой

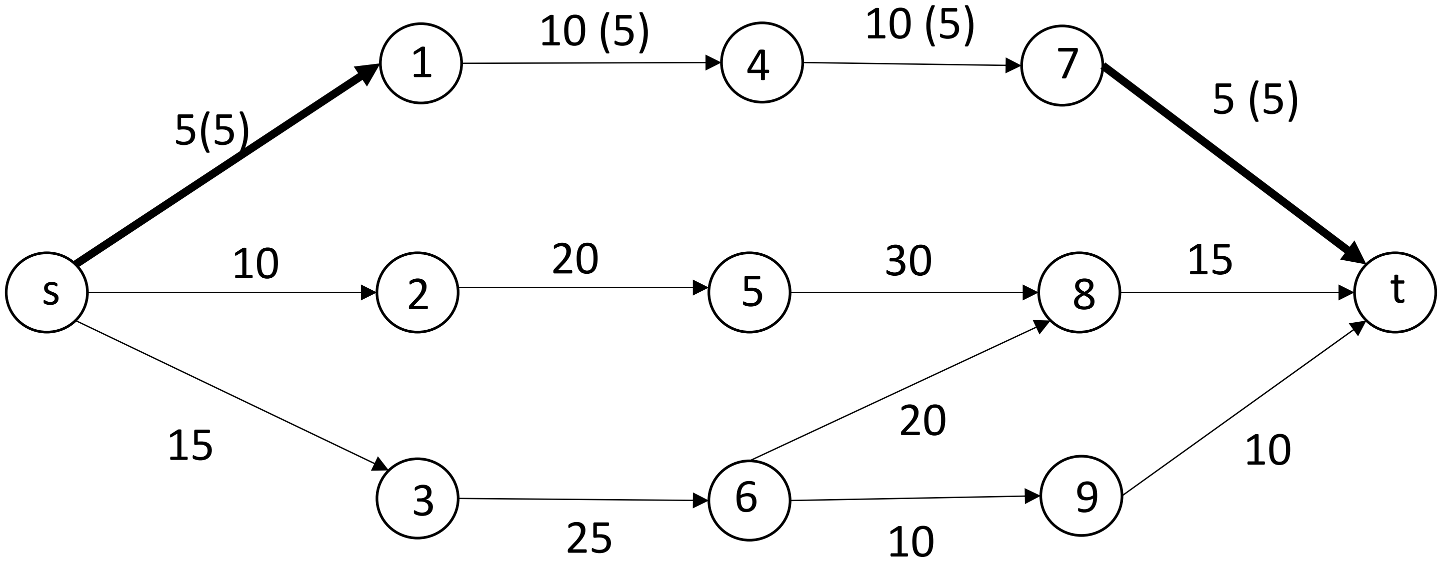


Вспомогательный граф
(слоистая сеть)

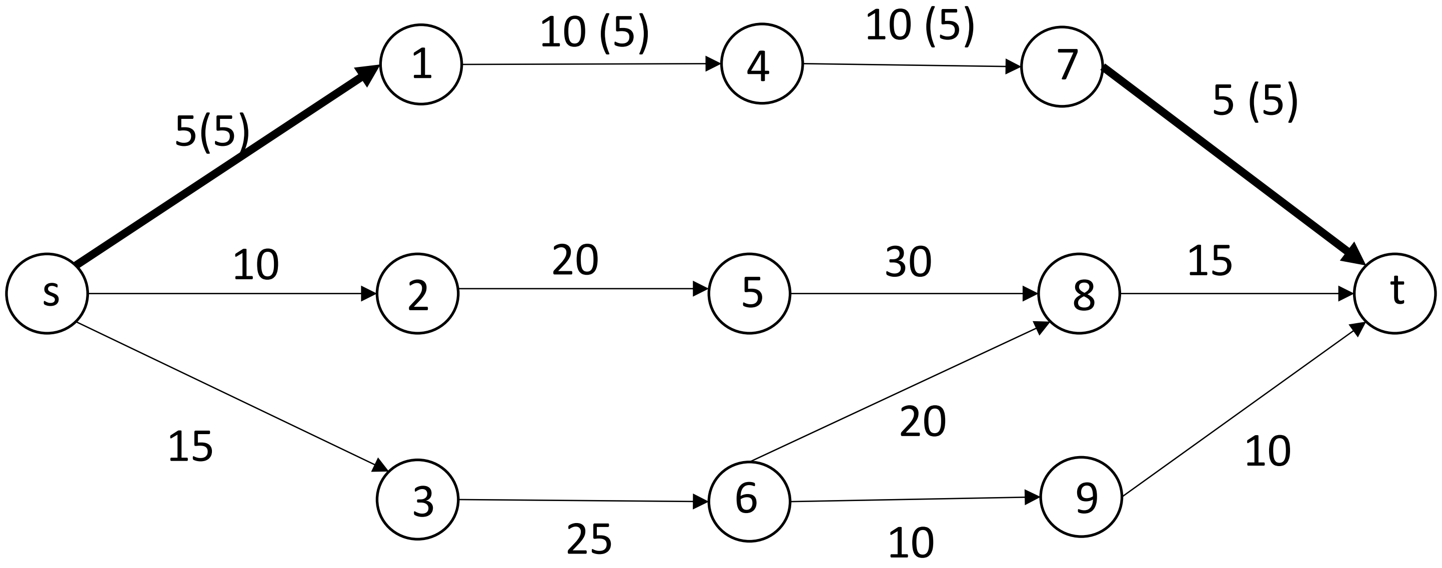


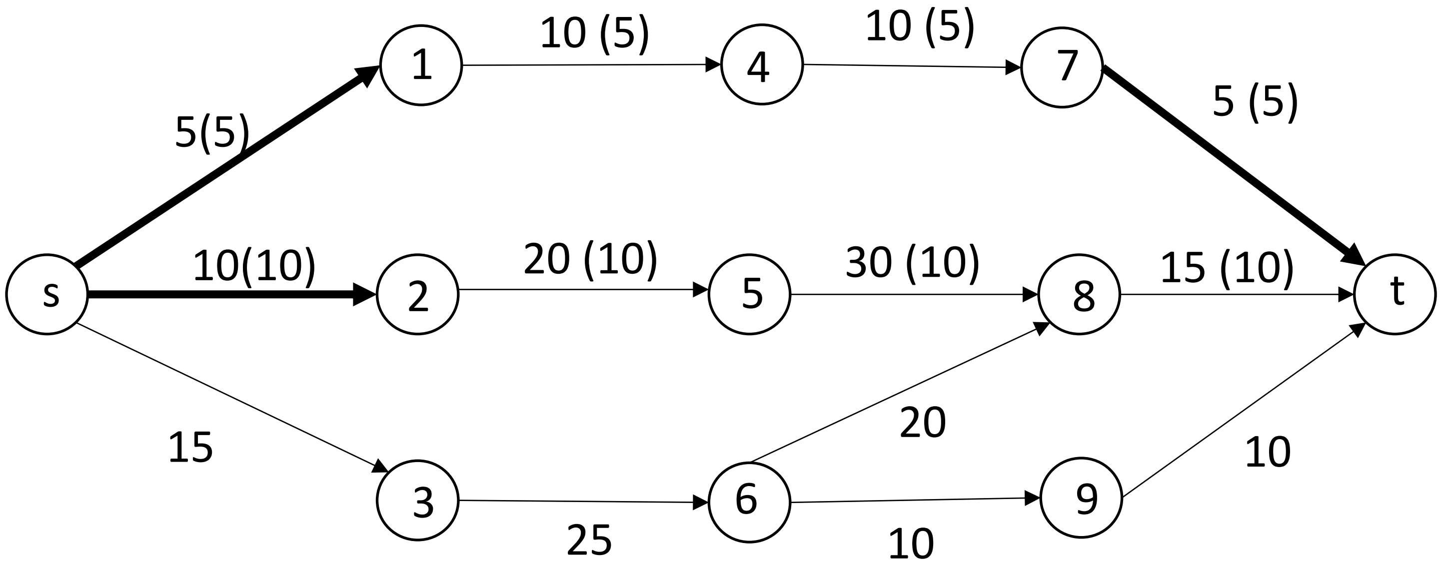
Блокирующий поток

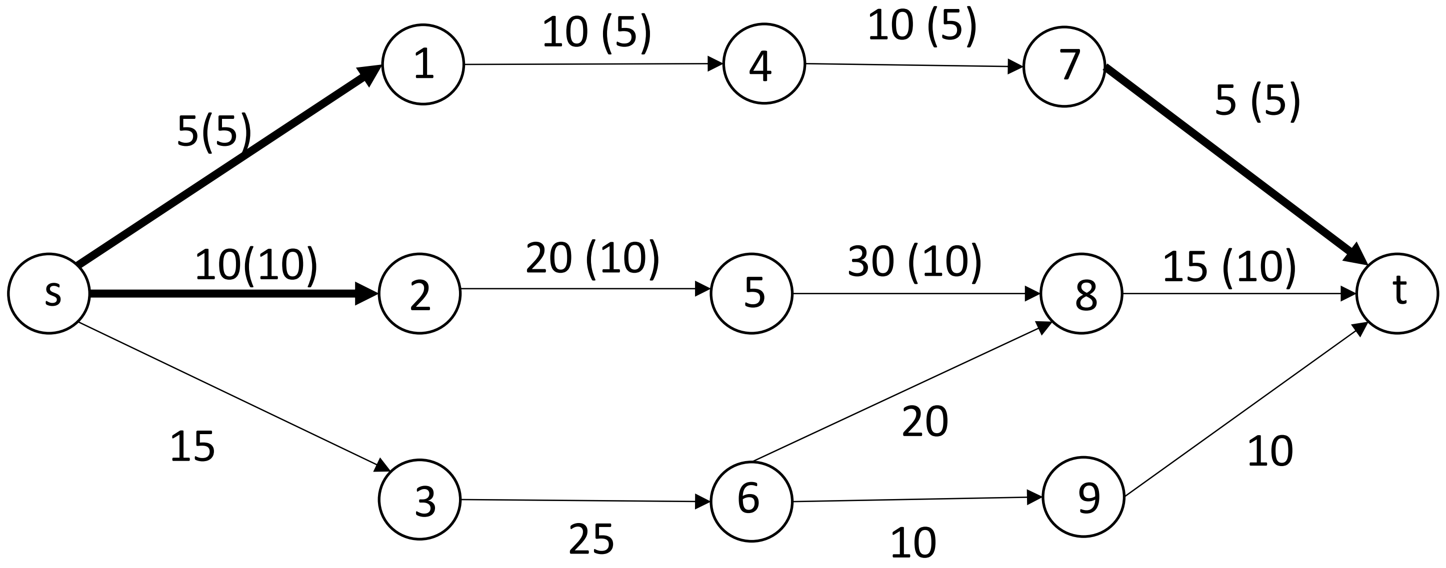


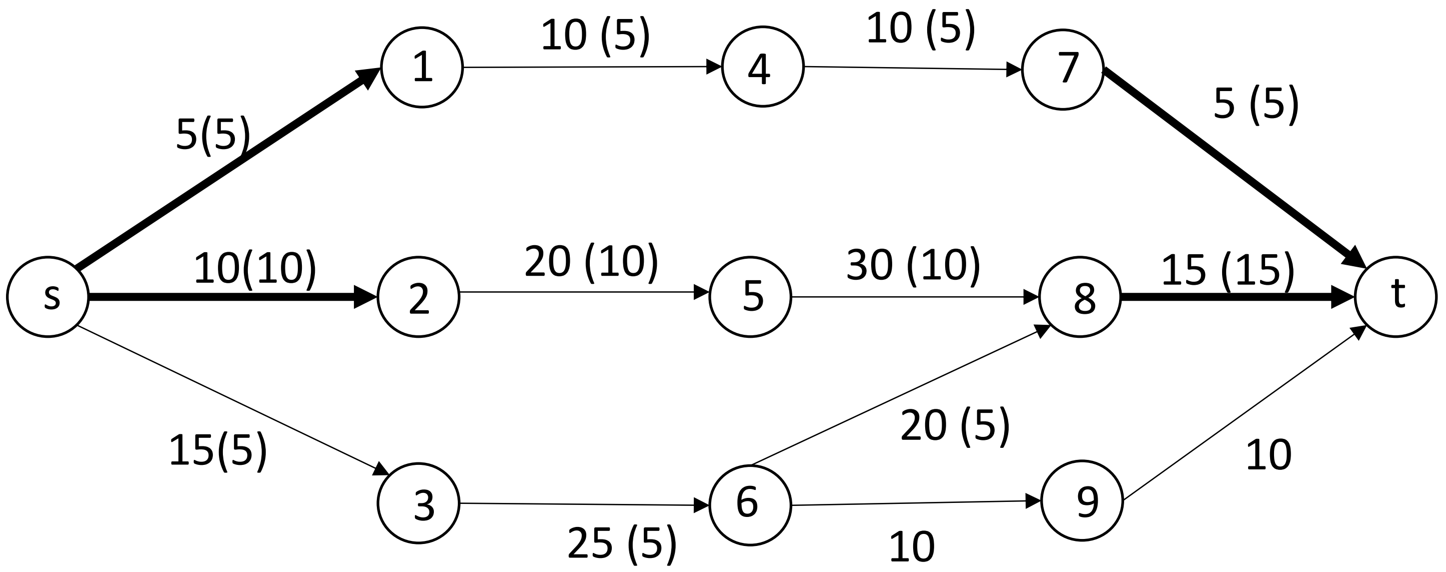


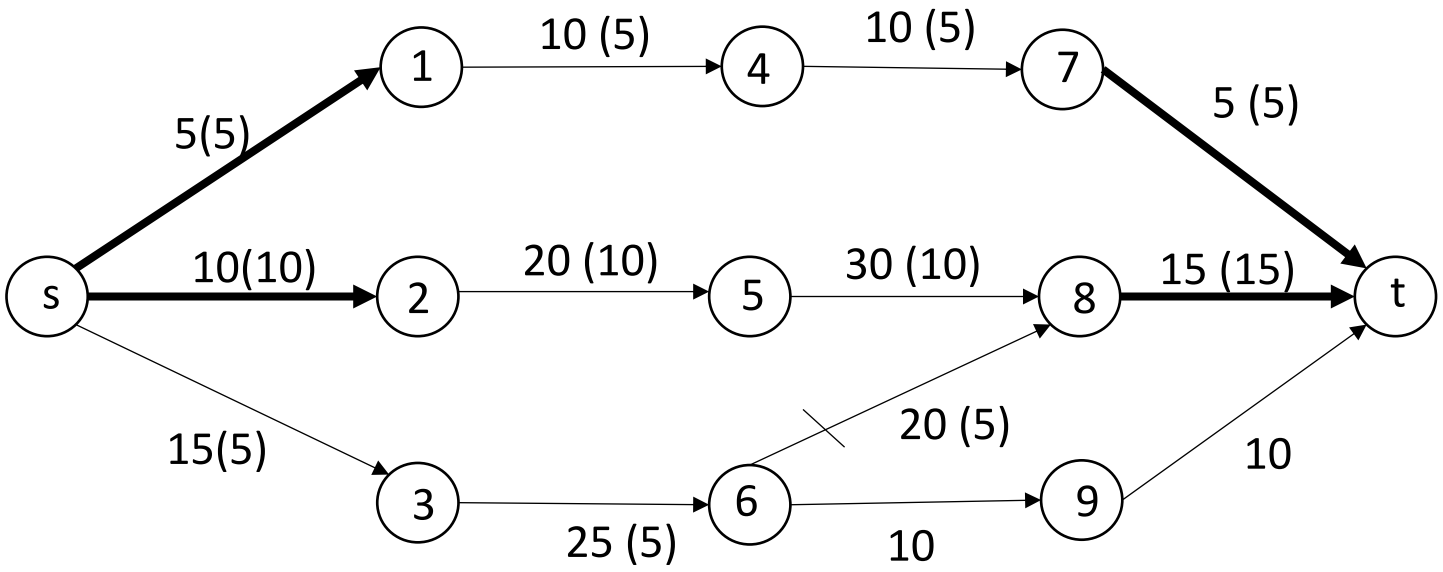
ФПМИ БГУ


s
1
2
3
4
5
6
7
8
9
t
5(5)
10
15
10 (5)
20
25
10 (5)
30
20
10
5 (5)
15
10

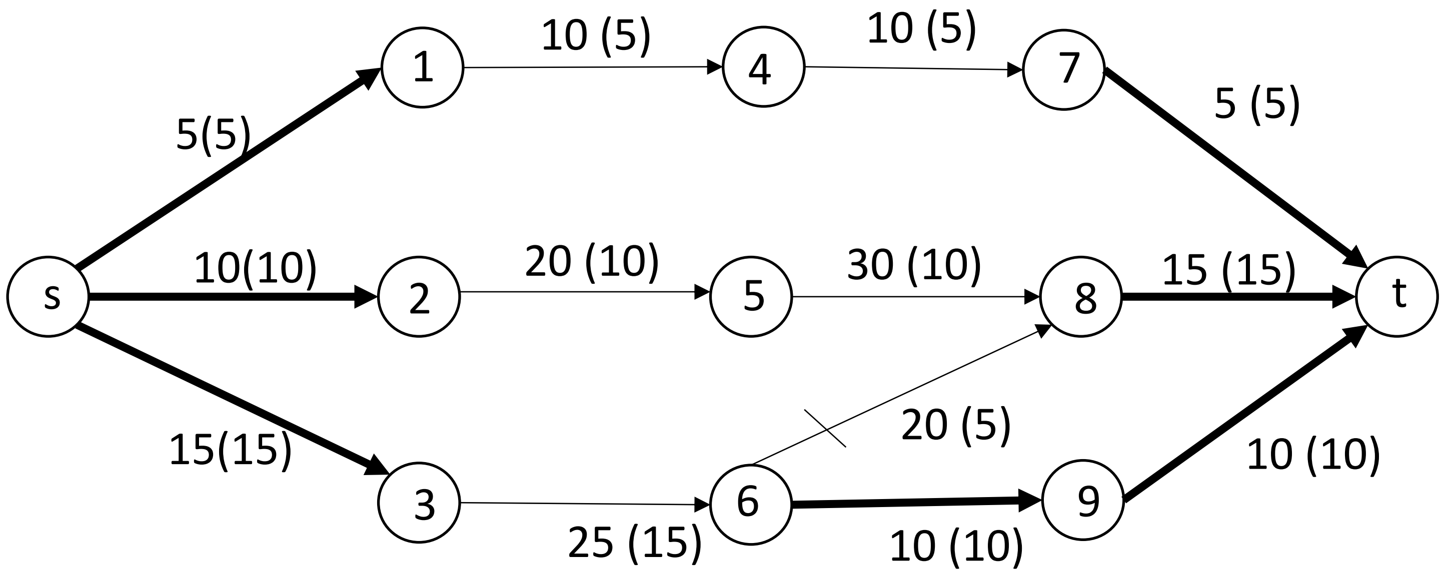

s
1
2
3
4
5
6
7
8
9
t
5(5)
10(10)
15
10 (5)
20 (10)
25
10 (5)
30 (10)
20
10
5 (5)
15 (10)
10


s
1
2
3
4
5
6
7
8
9
t
5(5)
10(10)
15
10 (5)
20 (10)
25
10 (5)
30 (10)
20
10
5 (5)
15 (10)
10

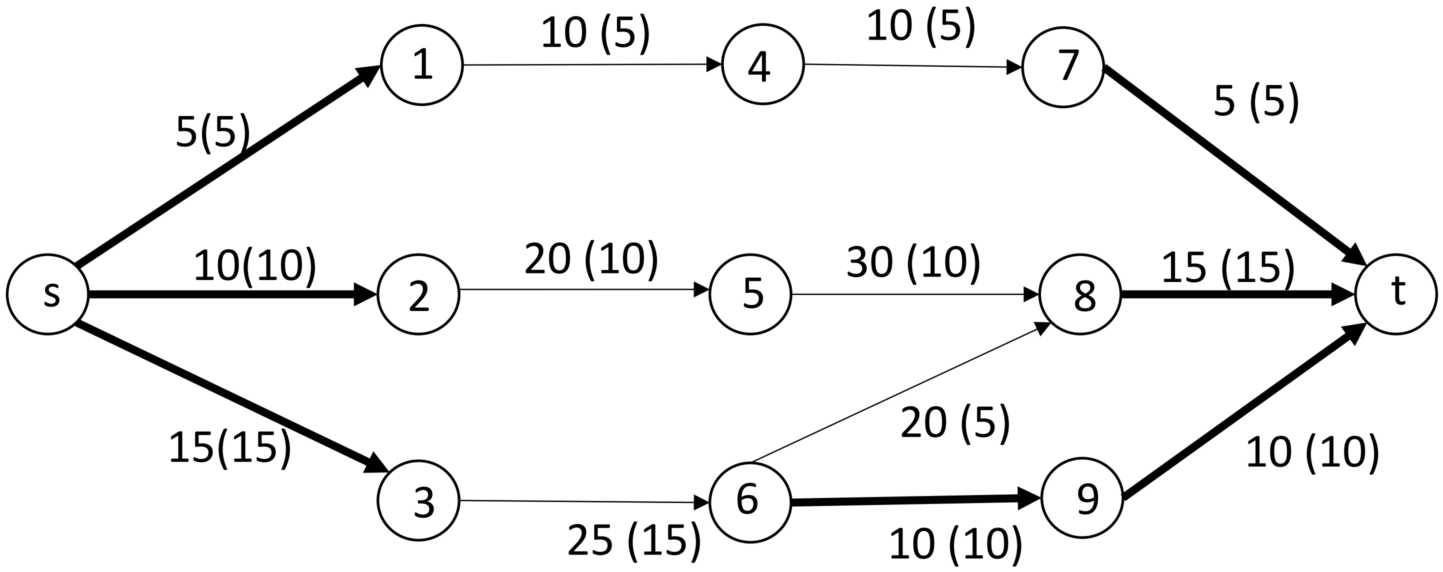

s
1
2
3
4
5
6
7
8
9
t
5(5)
10(10)
15(5)
10 (5)
20 (10)
25 (5)
10 (5)
30 (10)
20 (5)
10
5 (5)
15 (15)
10



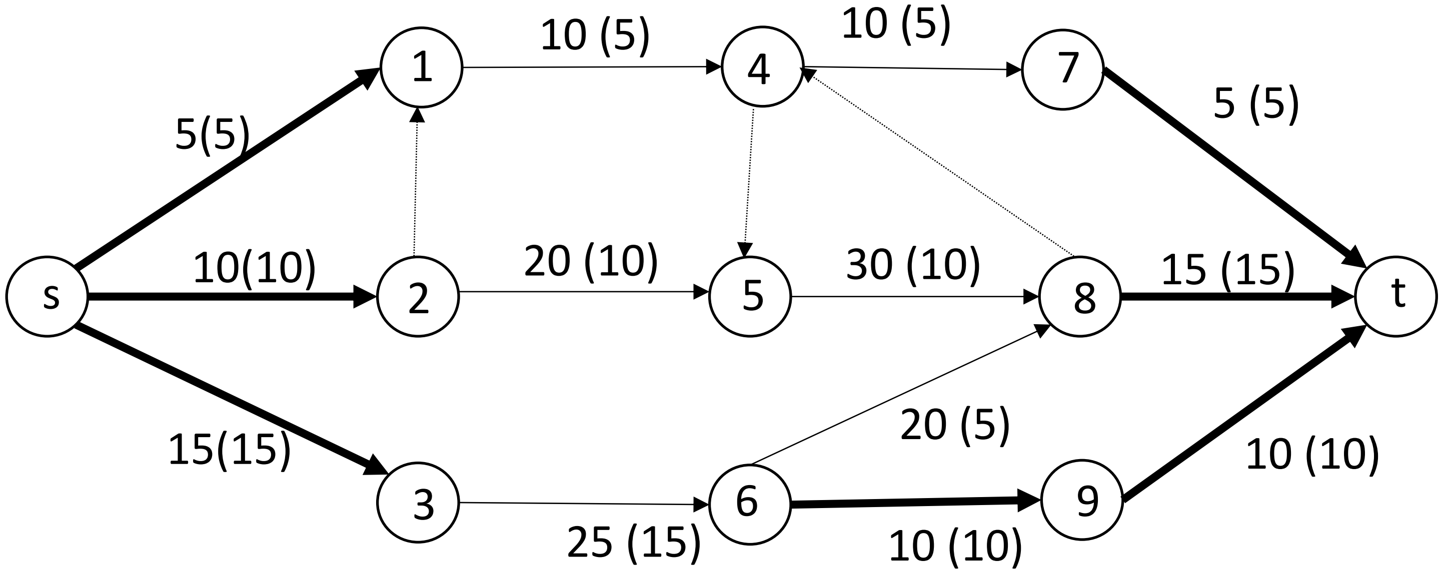
если при исследовании дуги (v, w) не был достигнут сток t, то дугу (v, w) блокируем, а при дальнейшем исследовании дуг, выходящих из вершины v, просматриваем список, начиная с дуги, следующей за (v, w)



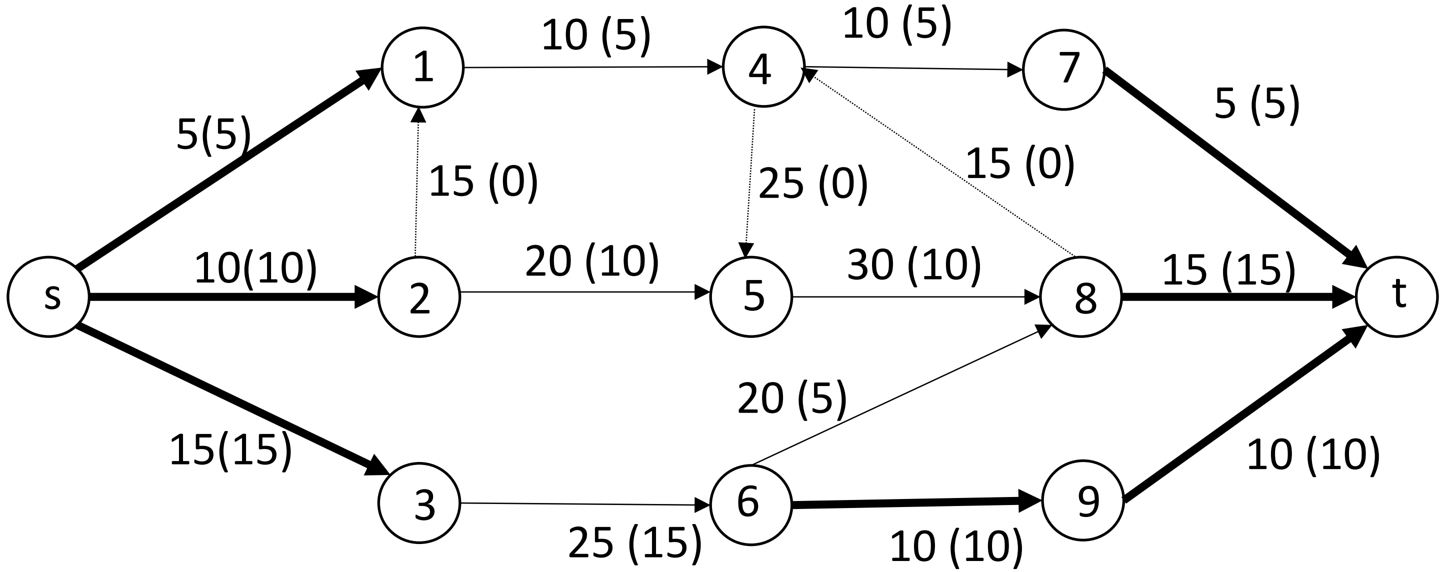
Блокирующий поток для слоистой сети на 1-ой фазе построен. Величина блокирующего потока – 5+10+15=30



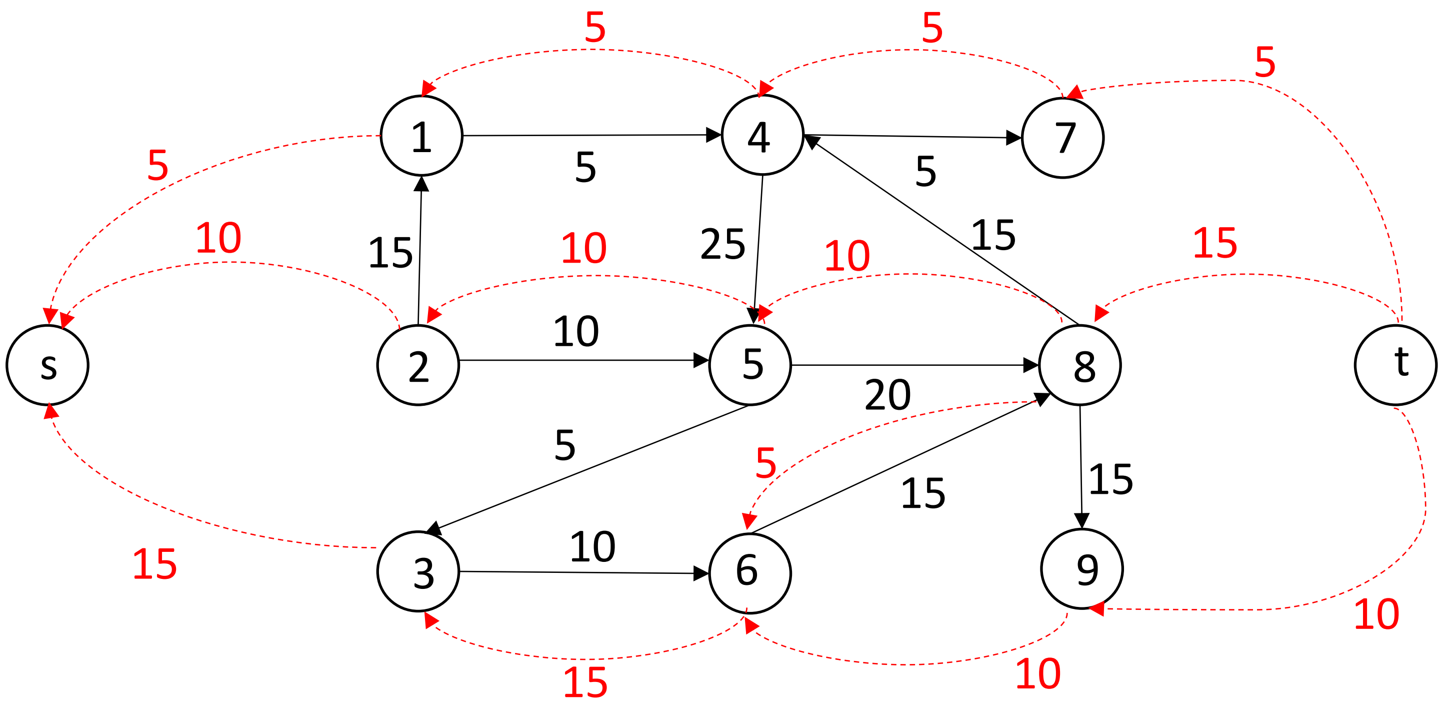
Блокирующий поток найден.
Величина блокирующего потока: 5+10+15=30



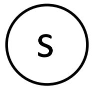
Корректируем по блокирующему потоку сеть остаточных пропускных способностей.



Сеть остаточных пропускных способностей



Слоистая сеть:



Так как (s,t) пути в слоистой сети не существует, то текущий поток является максимальным.

Время работы алгоритма Диница $-\ \mathbf{O}(\boldsymbol{m} \cdot \boldsymbol{n}^2)$.

Число итераций алгоритма Диница $- \mathbf{O}(\boldsymbol{n})$.

На каждой итерации время работы алгоритма построения блокирующего потока, в котором применяется процедура блокировки» дуг:

$$\mathrm{O}(m + m \cdot n) = \mathrm{O}(m \cdot n).$$

суммарное перемещение указателей в списках дуг, выходящих из вершин сети в результате выполнения процедуры «блокировки дуги»

так как каждый запуск DFS насыщает не менее 1 дуги, то число запусков поиска в глубину на каждой фазе не превосходит m;
так как длина наименьшего (s, t) −пути не превосходит n, то число «успешных» продвижений «вперед» dfs также не превосходит n (считаем продвижение успешным, если будет достигнут сток по данному направлению)

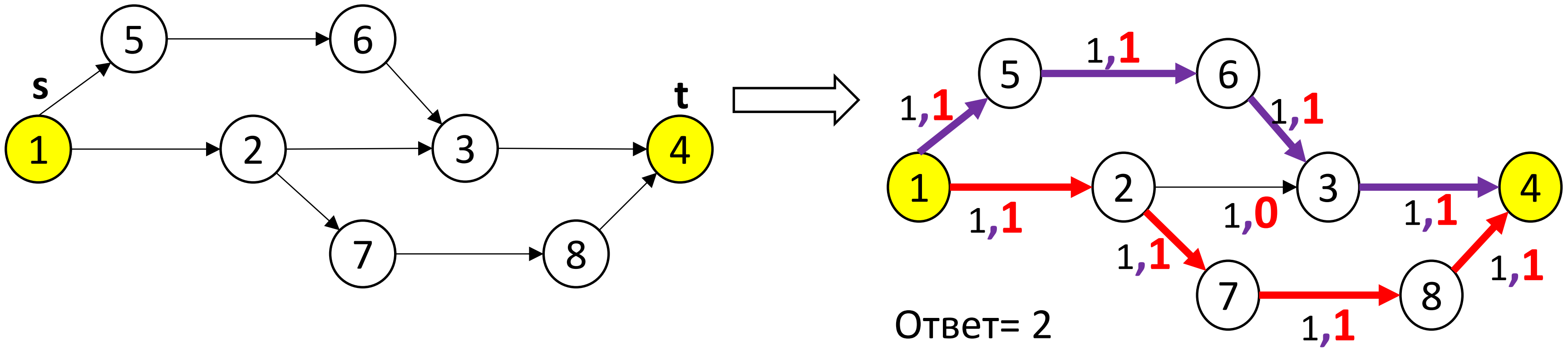
Приложения

Наибольшее число попарно различных путей

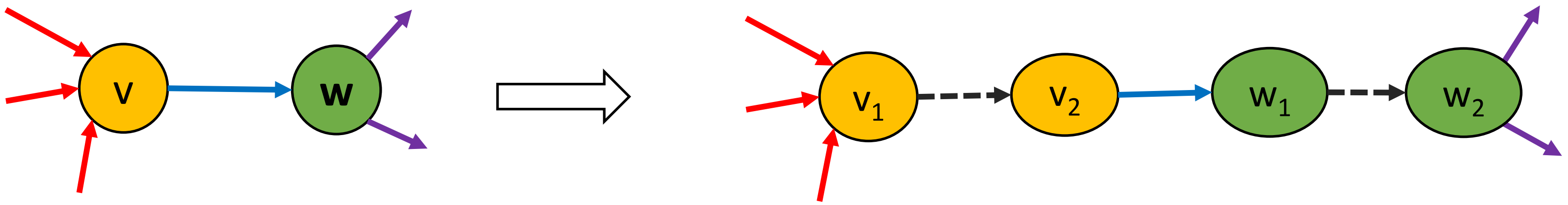
Наибольшее число (s,t)-путей, которые попарно не пересекаются

Задан ориентированый граф, в котором выделены две вершины **s** и **t**.

(1) Необходимо найти наибольшее число (**s,t**)-путей, которые попарно **не пересекаются по дугам**.



(2) Необходимо найти наибольшее число (**s,t**)-путей, которые попарно **не пересекаются по вершинам**.

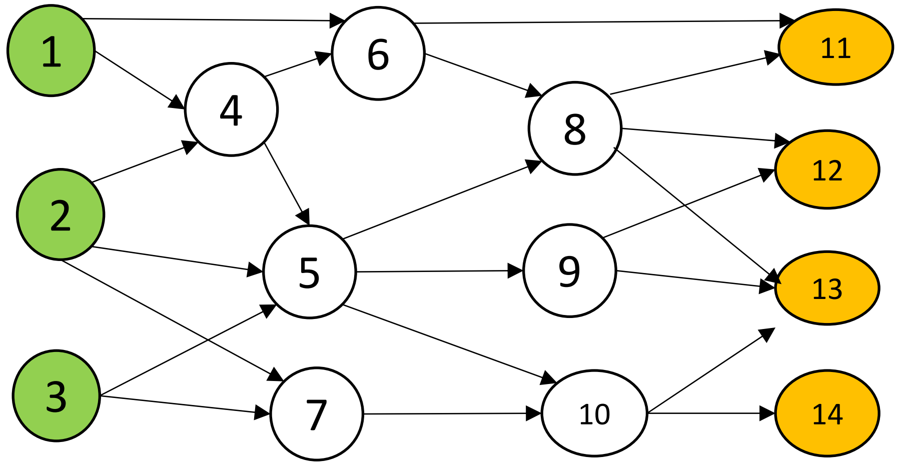


После преобразования решается задача нахождения наибольшего числа (**s,t**)-путей, которые попарно **не пересекаются по дугам** (пропускные способности всех дуг сети полагаются равными 1).

Время работы

O(m·n)

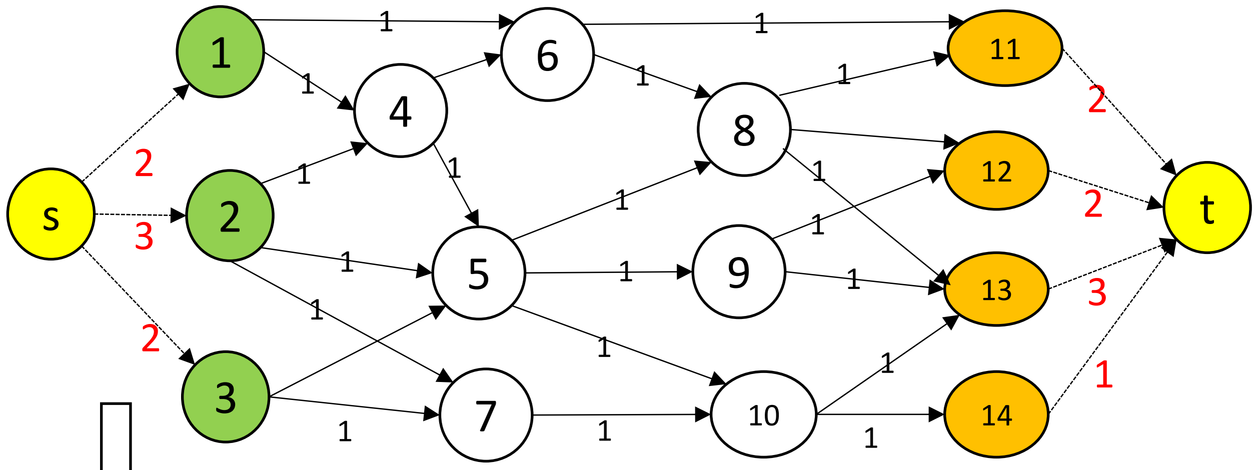
Задан ориентированый граф, в котором выделены два подмножества вершин: $\mathbf{V_1}$ и $\mathbf{V_2}$.
Необходимо найти наибольшее число путей, которые начинаются в $\mathbf{V_1}$ и заканчиваются в $\mathbf{V_2}$ и попарно не пересекаются по дугам.



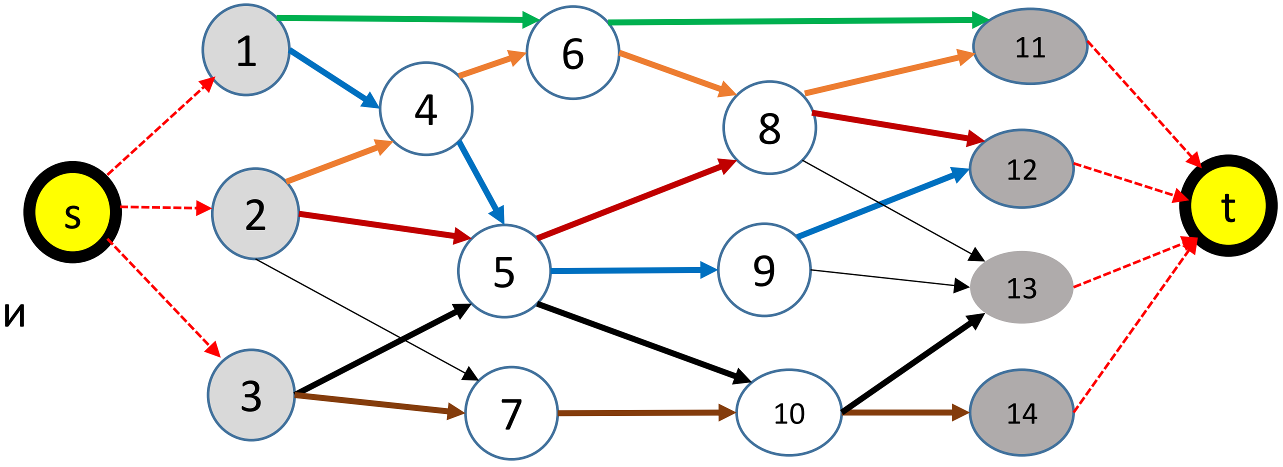
$V_1=\{1, 2, 3\}$

$V_2=\{11, 12, 13, 14\}$

строим сеть



находим максимальный поток



M(f)=6 - решение задачи

Время работы алгоритма:

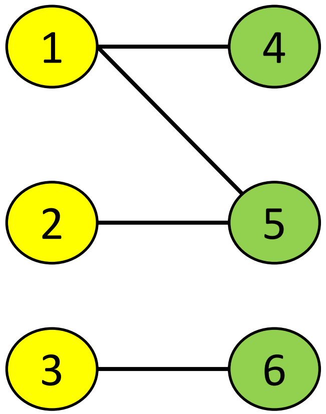
$$O(m^2)$$

Наибольшее паросочетание в двудольном графе

(*англ*. maximum matching)

Задан двудольный граф. Необходимо найти:

(1) наибольшее паросочетание;
(2) наибольшее паросочетание минимального веса (взвешенный граф).



Паросочетание это некоторое подмножество рёбер графа, в котором никакие два ребра не смежны.

maximum matching – наибольшее паросочетание

maximal matching – максимальное паросочетание

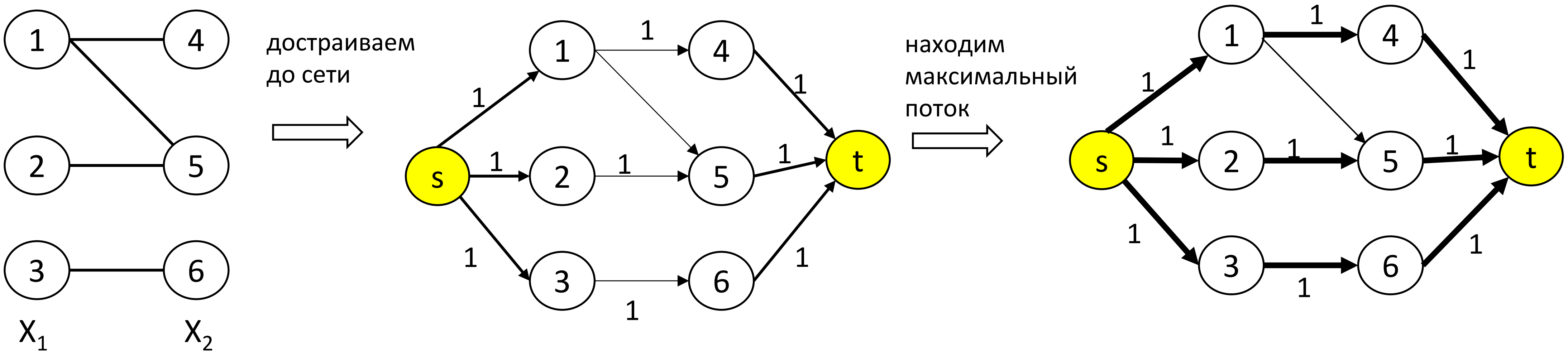
$M_1=\{\{1,5\}, \{3,6\}\}$	максимальное паросочетание	
$M_2=\{\{1,4\}, \{2,5\}\}$	паросочетание	
$M_3=\{\{1,4\}, \{2,5\}, \{3,6\}\}$	**наибольшее паросочетание**	совершенное паросочетание
$M_4=\{\{1,5\}, \{1,4\}\}$	НЕ паросочетание	

Наибольшее паросочетание в двудольном графе

Задан двудольный граф.

Известно разбиение на доли.

Необходимо найти **наибольшее паросочетание.**



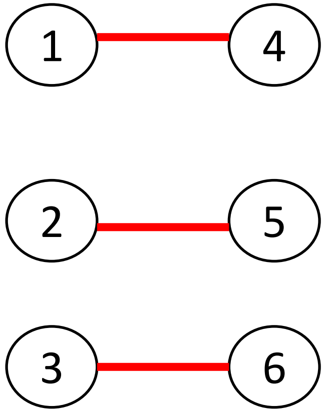
Время работы: $O(\boldsymbol{n} \cdot \boldsymbol{m})$

время работы bfs,dfs - $O(m)$

число итераций bfs - min {$|X_1|$, {$|X_2|$}=$O(n)$

рёбра двудольного графа, по которым поток равен 1, включаем в наибольшее паросочетание

M={{1,4}, {2,5}, {3,6}}



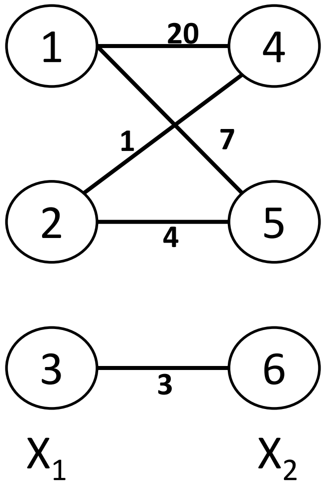
Теория алгоритмов: учеб.-метод. пособие / П. А. Иржавский и др.]. – Минск : БГУ, 2013. С. 77-105.

Наибольшее паросочетание минимального веса в двудольном графе

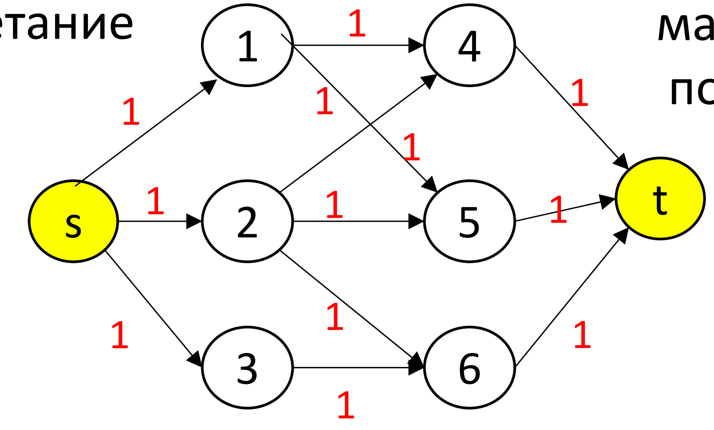
Задан двудольный граф. Каждому ребру которого приписан целочисленный вес $c(e) \geq 0$.

Известно разбиение вершин двудольного графа на доли.

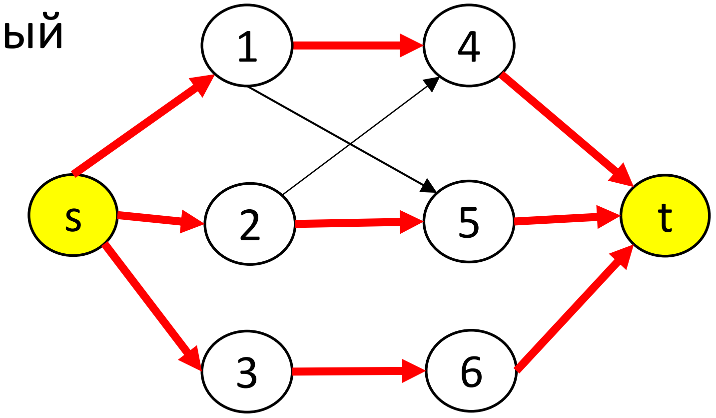
Необходимо найти **наибольшее паросочетание минимального веса.**

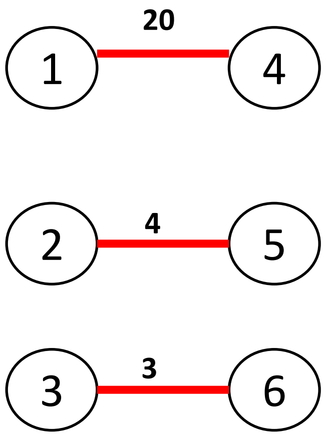


сначала найдём наибольшее паросочетание



строим максимальный поток

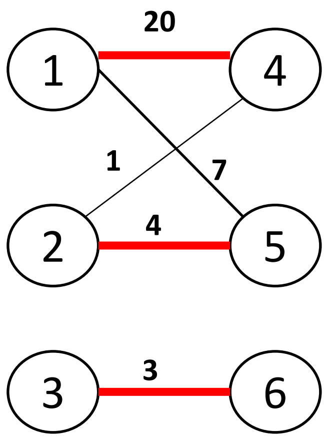




рёбра двудольного графа, по которым поток равен 1, включаем в наибольшее паросочетание

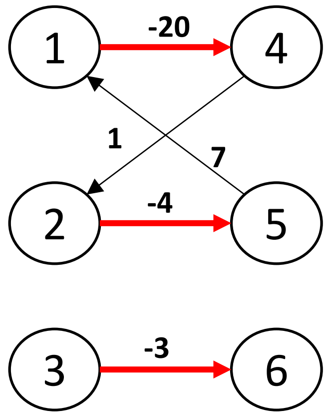
$$|M_1| = 3, \mathrm{c}(M_1) = 20 + 4 + 3 = 27$$

(*продолжение*) наибольшее паросочетание минимального веса

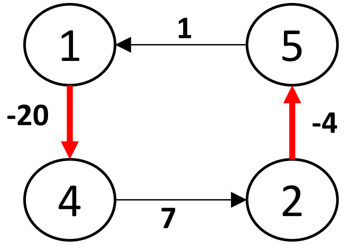


$|M_1| = 3$
$c(M_1) = 20 + 4 + 3 = 27$

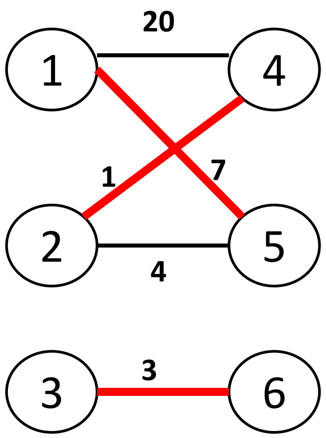
ориентируем
рёбра графа



контур отрицательного веса
$(= -16)$:
$1 \to 4 \to 2 \to 5 \to 1$

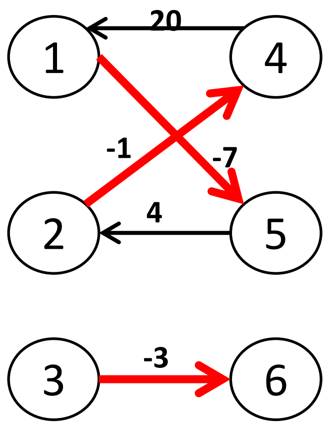


перестраиваем
паросочетание



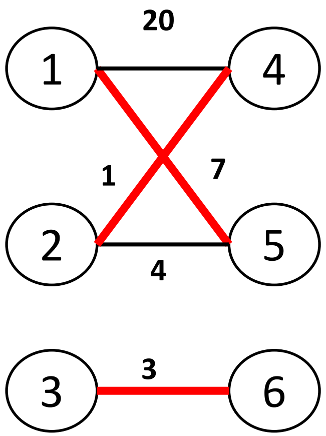
новое паросочетание
$|M_2| = 3$,
$c(M_2) = 1 + 7 + 3 = 11$

повторяем
процесс



контур отрицательного веса: НЕТ

$M_2 = \{\{1,5\},\{2,4\},\{3,6\}\}$ –
наибольшее паросочетание
минимального веса
$c(M_2) = 11$



Время работы алгоритма

$$O(c^{max} \cdot m \cdot n^2)$$

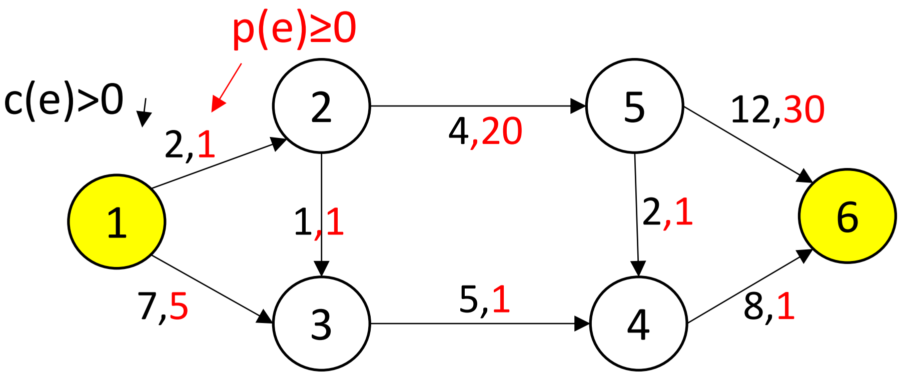
$$O(n \cdot m) + O(c^{max} \cdot n) \cdot (O(n \cdot m) + O(m))$$

поиск наибольшего паросочетания в двудольном графе;

максимальный вес наибольшего паросочетания (в паросочетании не может быть более, чем n рёбер), поэтому данной величиной можно оценить наибольшее число итераций поиска контура отрицательного веса;

поиск контура отрицательного веса алгоритмом Форда –Беллмана и перестройка наибольшего паросочетания вдоль контура отрицательного веса

Максимальный поток минимальной стоимости
(англ. max flow min cost)

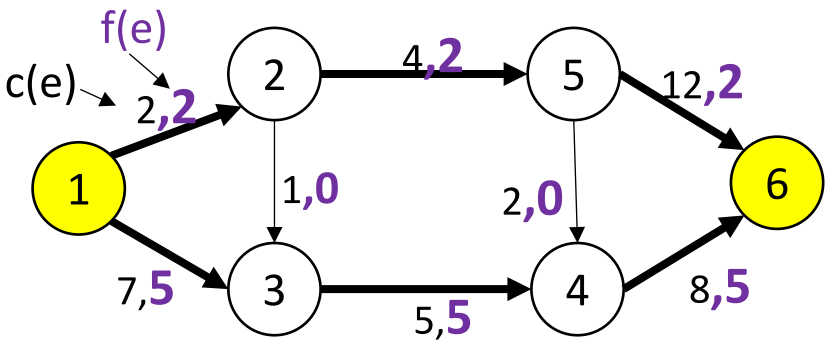


Стоимость потока f:

$$P(f) = \sum_{e \in E} f(e) \cdot p(e)$$

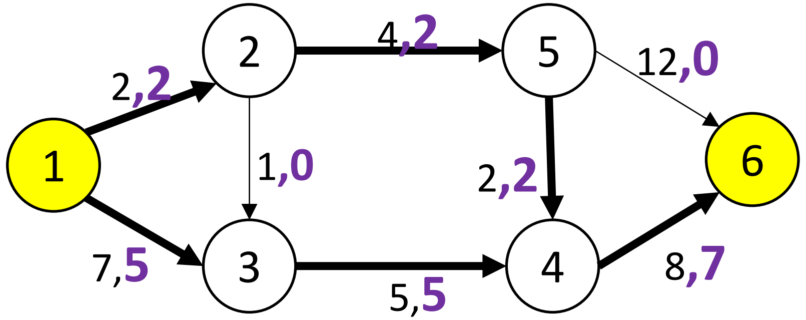
Максимальный поток, но среди всех максимальных потоков его удельная стоимость не является минимальной.

$M(f) = 7$



$$P(f) = f(1{,}2)\cdot p(1{,}2) + f(2{,}5)\cdot p(2{,}5) + f(5{,}6)\cdot p(5{,}6) + f(1{,}3)\cdot p(1{,}3) +$$
$$+ f(3{,}4)\cdot p(3{,}4) + f(4{,}6)\cdot p(4{,}6) = 2\cdot 1 + 2\cdot 20 + 2\cdot 30 + 5\cdot 5 + 5\cdot 1 + 5\cdot 1 = 137$$

$M(f) = 7$

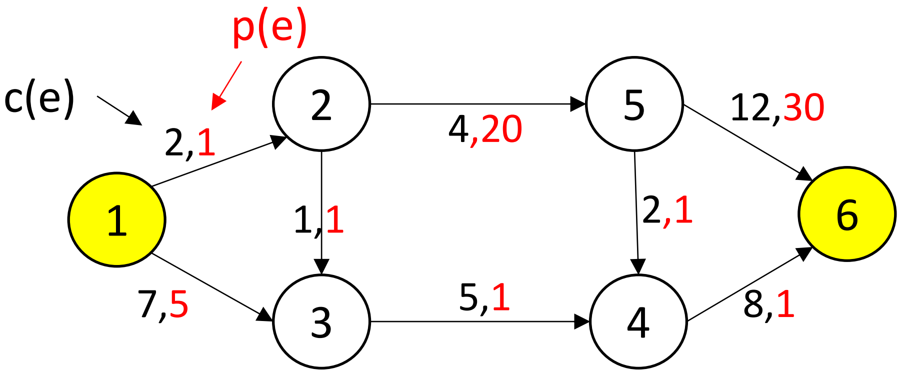


$$P(f) = f(1{,}2)\cdot p(1{,}2) + f(2{,}5)\cdot p(2{,}5) + f(5{,}4)\cdot p(5{,}4) + f(1{,}3)\cdot p(1{,}3) +$$
$$+ f(3{,}4)\cdot p(3{,}4) + f(4{,}6)\cdot p(4{,}6) = 2\cdot 1 + 2\cdot 20 + 2\cdot 1 + 5\cdot 5 + 5\cdot 1 + 7\cdot 1 = 81$$

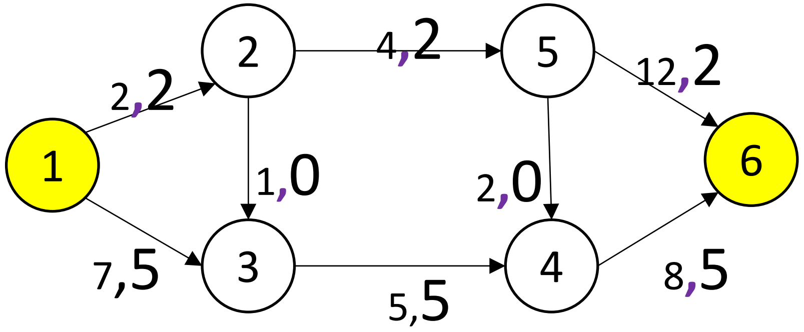
Максимальный поток минимальной стоимости

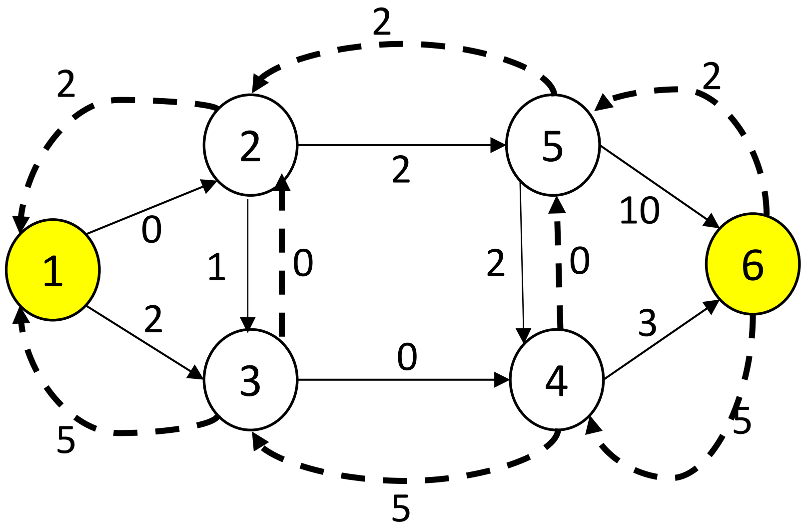
Метод устранения отрицательных циклов

исходная сеть



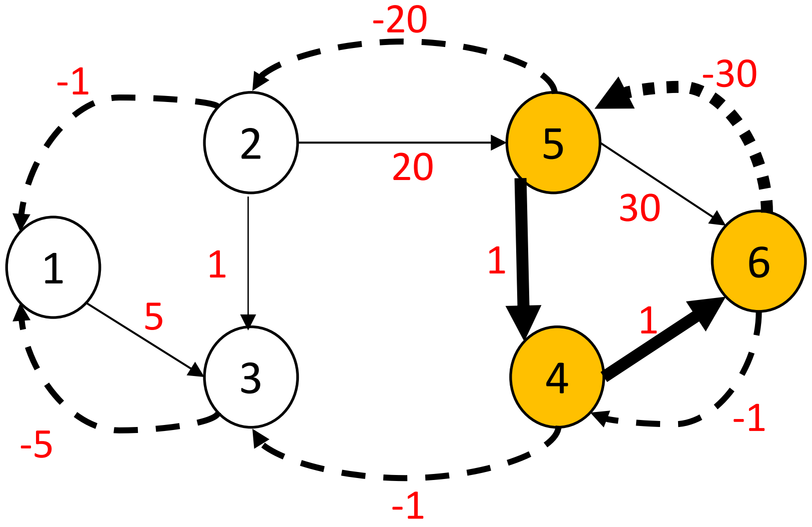
максимальный поток





сеть остаточных пропускных способностей на последней итерации алгоритма построения максимального потока

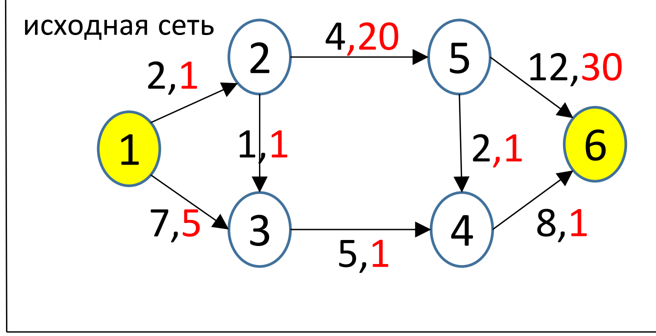
дуге *e* исходной сети **p(e)**

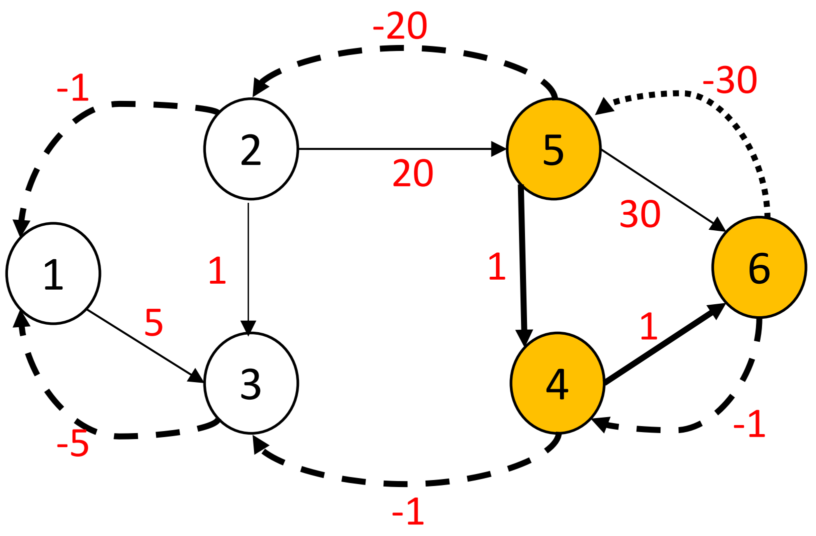


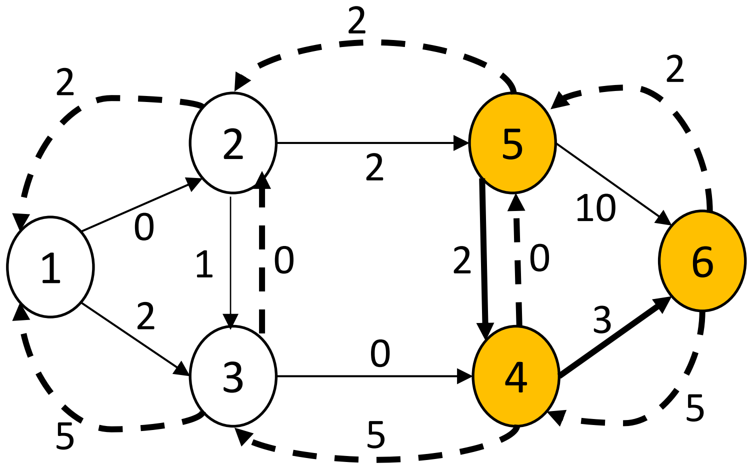
обратной дуге $\bar{e}$ ставим **−p(e)**

C={ (6,5),(5,4),(4,6)} – контур отрицательной стоимости **-28**; перераспределяя вдоль контура 1 единицу потока, получим поток той же величины, но стоимость которого меньше на **|p'(C)|** единиц, чем у текущего потока;

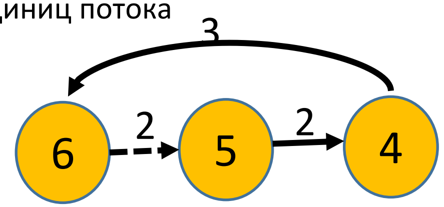
берём сеть остаточных пропускных способностей последней итерации алгоритма построения максимального потока и вдоль контура отрицательного веса перераспределяем поток

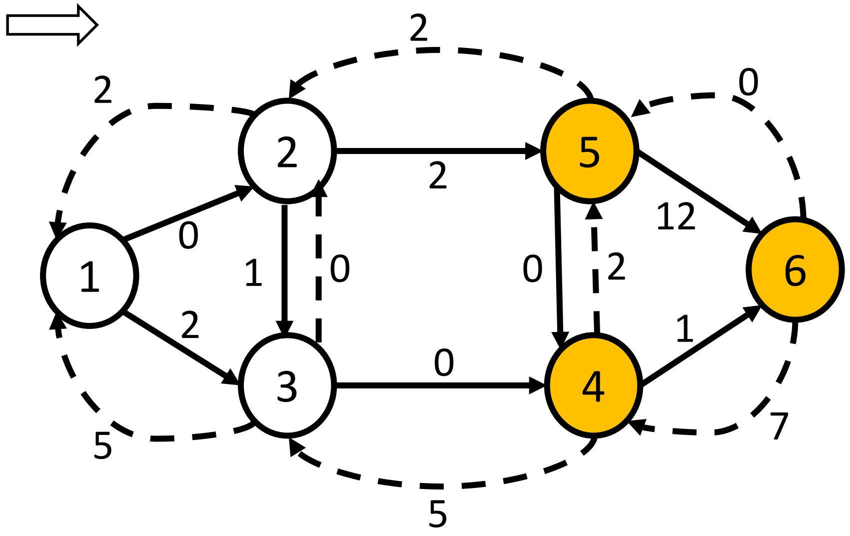




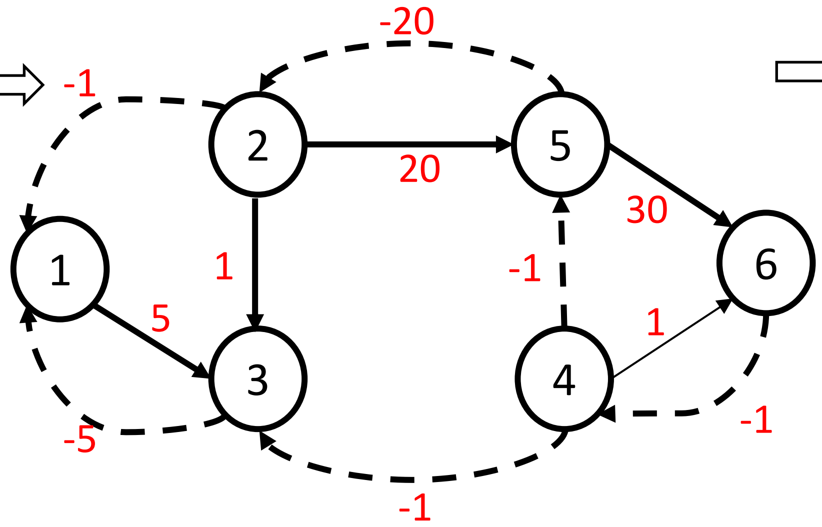


$c_f^{\min} = 2$ количество перераспределяемых единиц потока

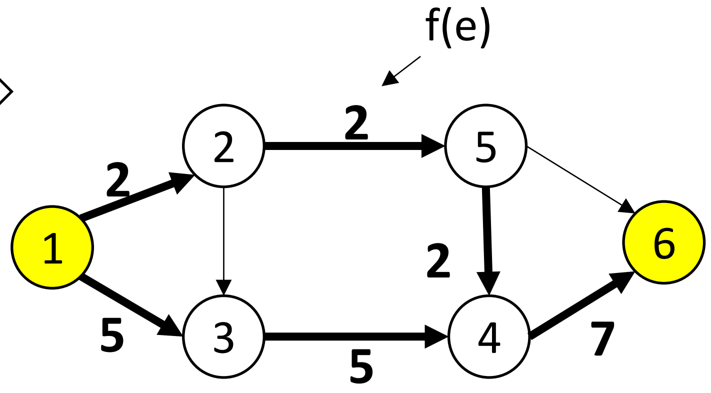




$P(f) = 137 - 2 \cdot 28$
$= 81$



нет контура отрицательной стоимости



$M(f) = 7$
$P(f) = 81$

Время работы: $O(\boldsymbol{c}^{max} \cdot \boldsymbol{p}^{max} \cdot \boldsymbol{m} \cdot \boldsymbol{n}^2 + \boldsymbol{n} \cdot \boldsymbol{m}^2)$

$O(\boldsymbol{n} \cdot \boldsymbol{m}^2)$ – поиск максимального потока, например, алгоритмом Эдмондса – Карпа

$+$

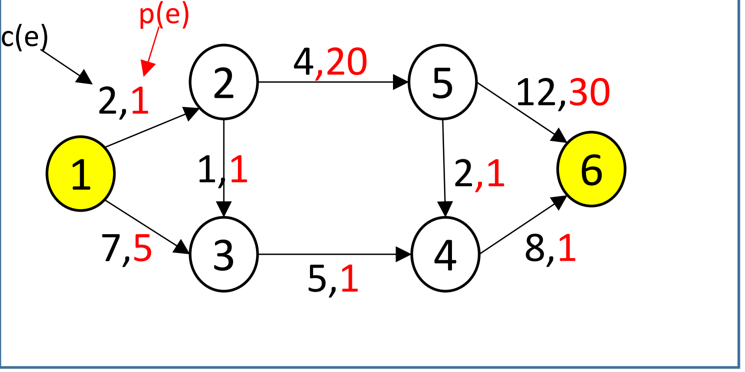
$O(\boldsymbol{c}^{max} \cdot \boldsymbol{n} \cdot \boldsymbol{p}^{max}) \cdot O(\boldsymbol{n} \cdot \boldsymbol{m})$

наибольшая возможная удельная стоимость максимального потока (по предположению сеть целочисленная и нет кратных дуг)

поиск циклов отрицательной удельной стоимости, например, алгоритмом Форда –Беллмана

Максимальный поток минимальной стоимости

Метод минимальных путей

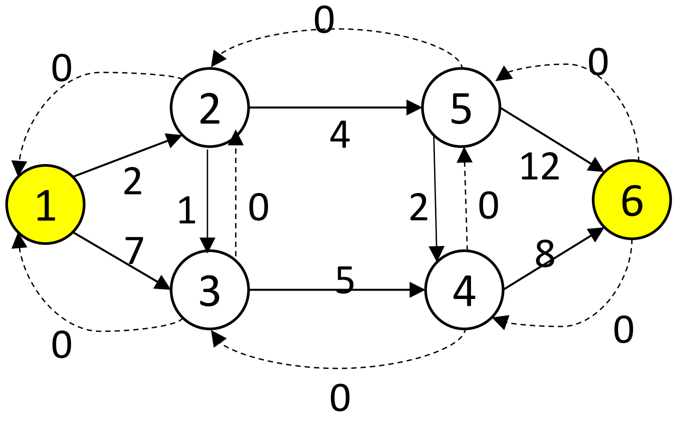


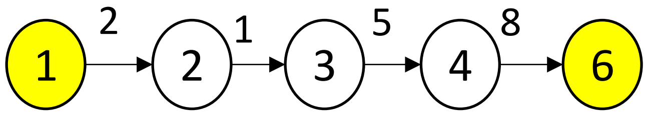
Исходная сеть

$M(f) = 0$
$P(f) = 0$

1-я итерация

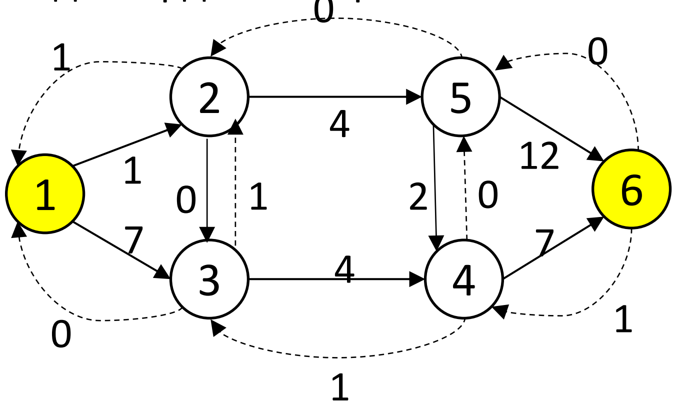
сеть остаточных
пропускных способностей



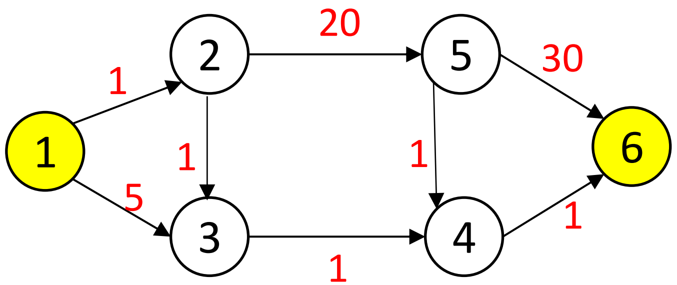


$c_f^{\min} = 1$

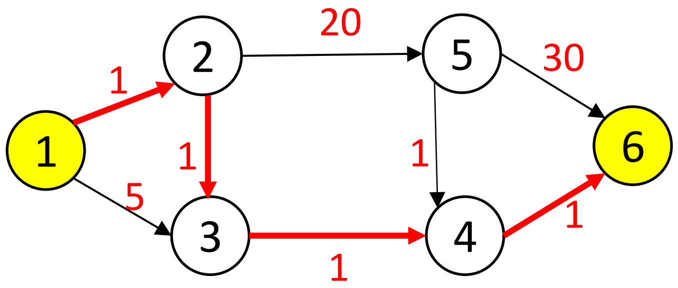
вдоль данного пути увеличиваем
текущий поток, как и на
итерациях
метода Форда-Фалкерсон



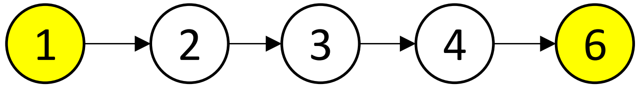
$M(f) = 1$
$P(f) = 4$

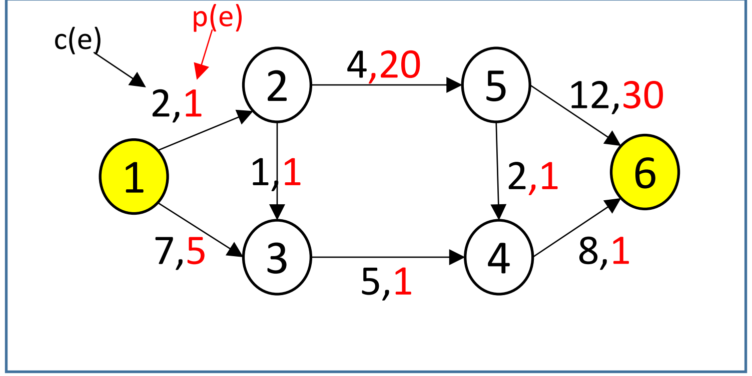


оставляем дуги, для которых остаточная
пропускная способность >0;
дуге e исходной сети приписываем **p(e)**,
а её обратной дуге приписываем **–p(e)**



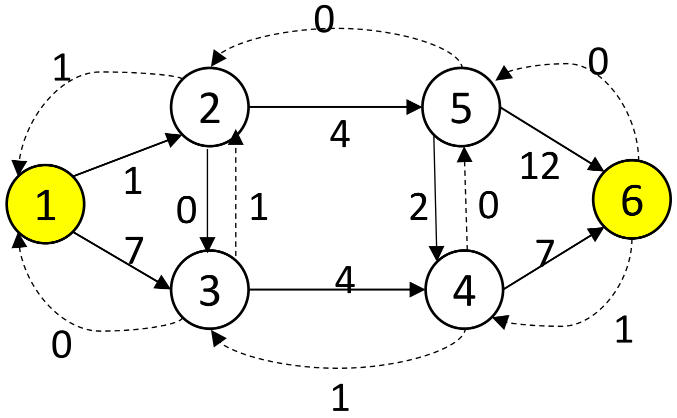
в сети могут быть дуги отрицательного
веса, но нет отрицательных контуров;
находим кратчайший по удельной
стоимости (1,6)-путь:



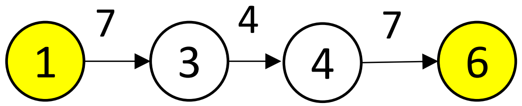


2-я итерация

сеть остаточных пропускных способностей

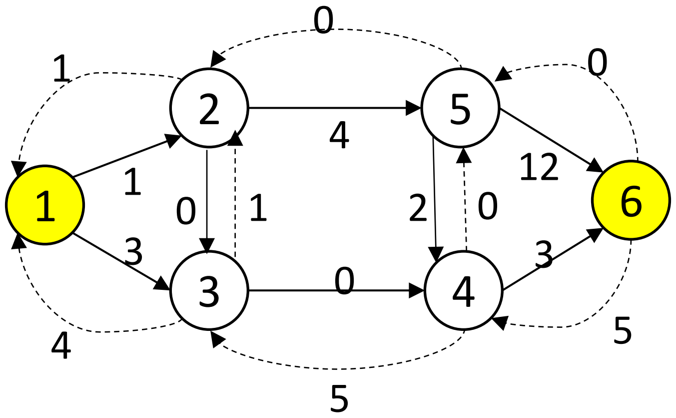


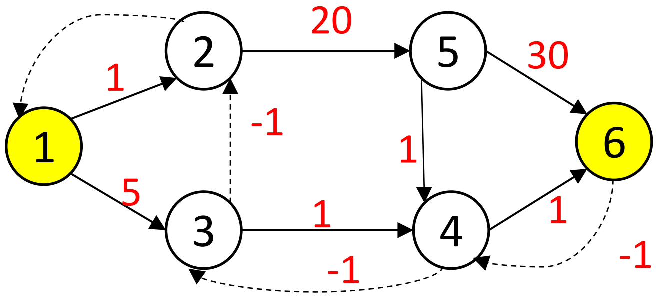
вдоль данного пути увеличиваем текущий поток



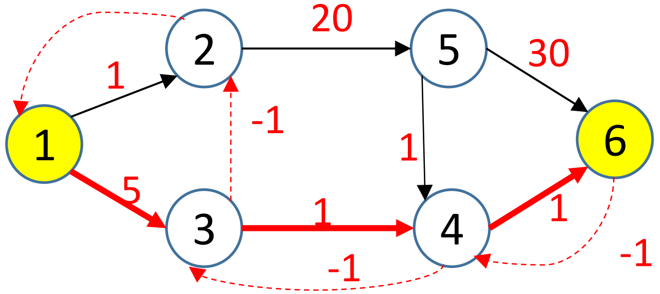
$c_f^{\min} = 4$

$M(f) = 5$
$P(f) = 32$

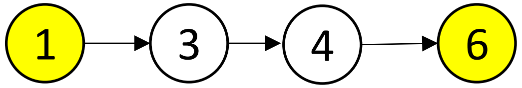




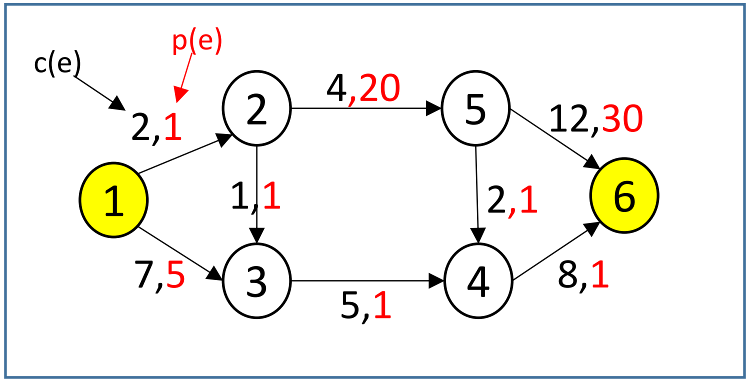
оставляем дуги, для которых остаточная пропускная способность >0;
дуге e исходной сети приписываем p(e),
а её обратной дуге приписываем –p(e)



находим кратчайший по удельной стоимости (1,6)-путь:

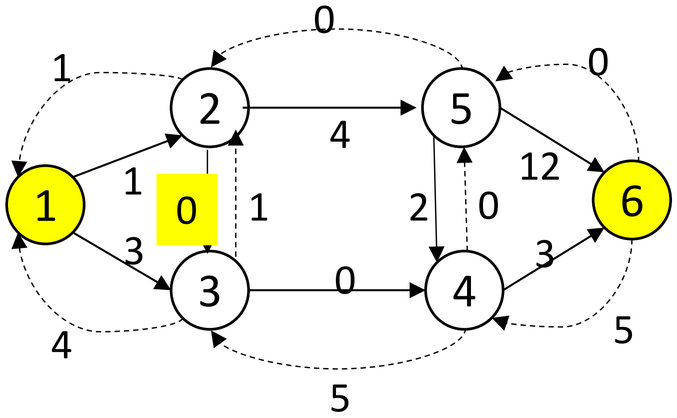


ФПМИ БГУ

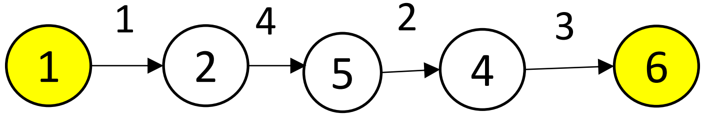


3-я итерация

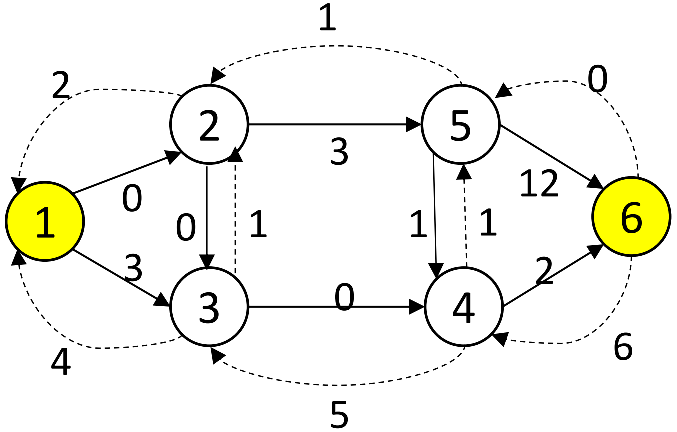
сеть остаточных пропускных способностей



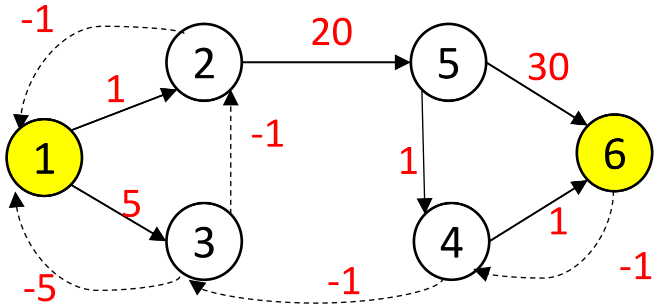
вдоль данного пути увеличиваем текущий поток



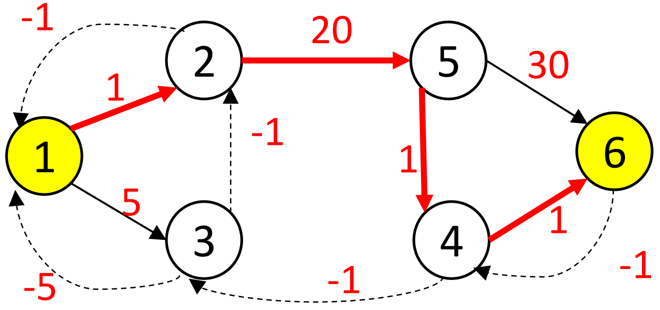
$$c_f^{\min} = 1$$



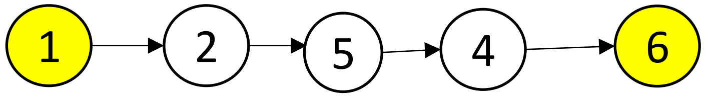
$$M(f) = 6$$
$$P(f) = 55$$

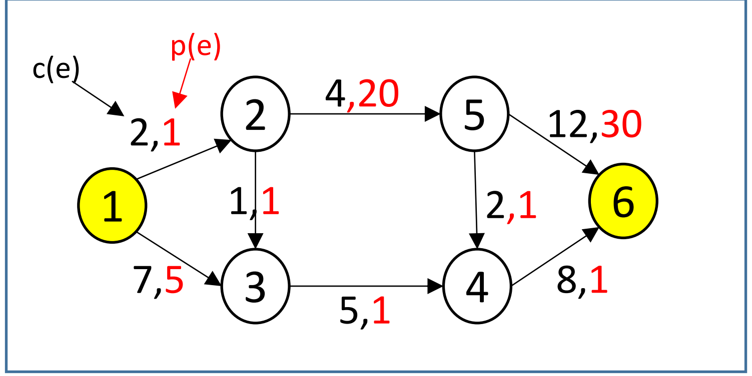


оставляем дуги, для которых остаточная пропускная способность >0;
дуге e исходной сети приписываем p(e),
а её обратной дуге приписываем –p(e)



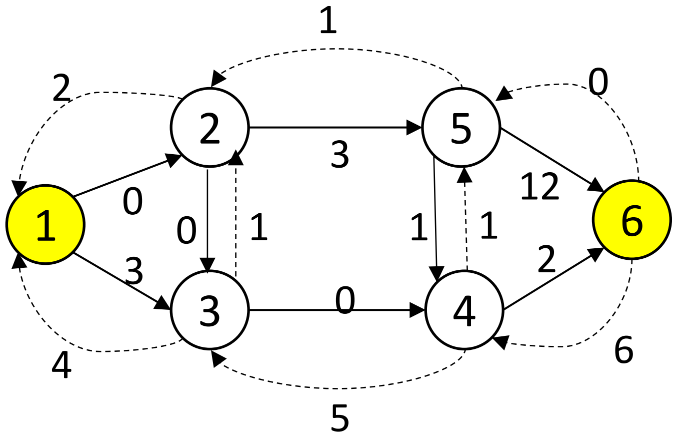
находим кратчайший по удельной стоимости (1,6)-путь:



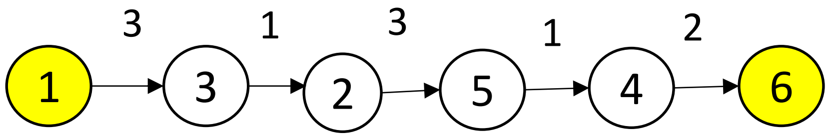


4-я итерация

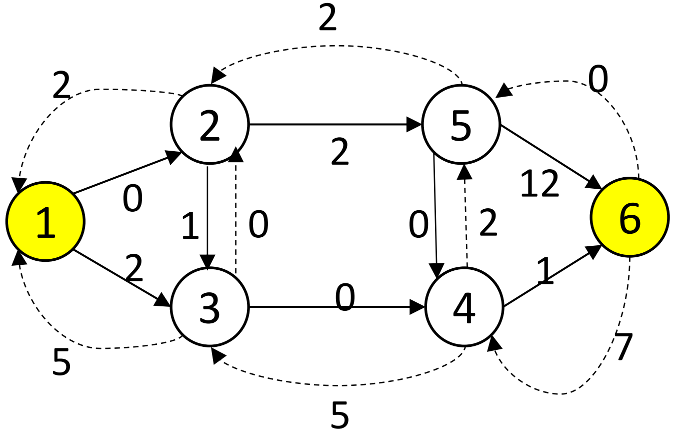
сеть остаточных
пропускных способностей



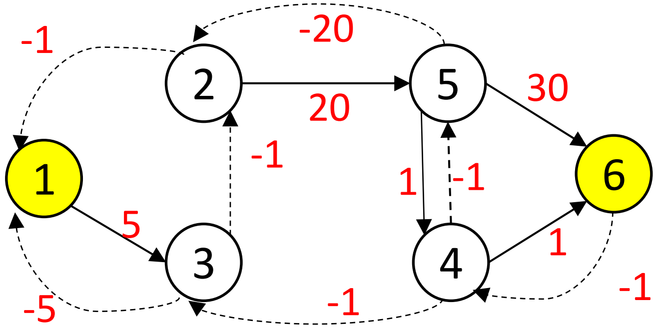
вдоль данного пути увеличиваем текущий поток



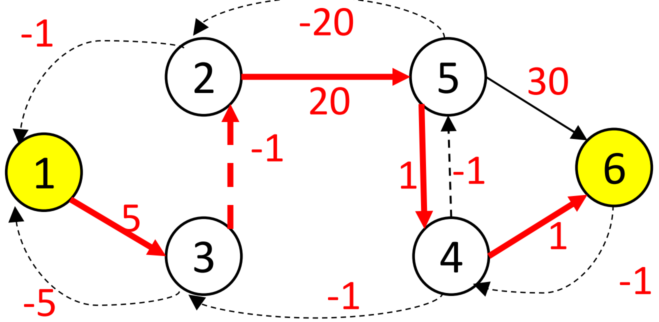
$c_f^{\min} = 1$



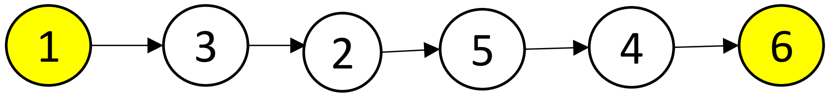
$M(f) = 7$
$P(f) = 81$



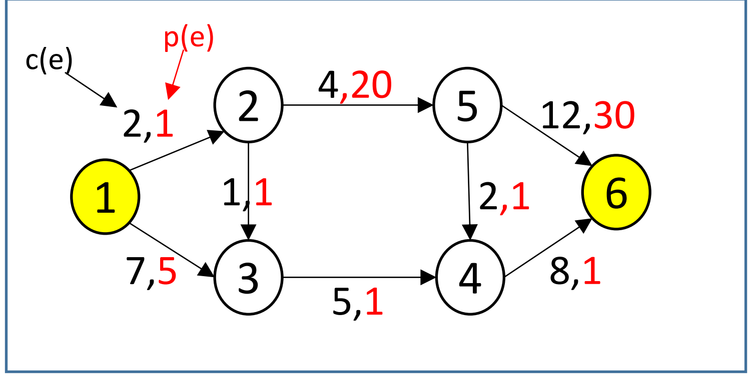
оставляем дуги, для которых остаточная пропускная способность >0;
дуге e исходной сети приписываем p(e),
а её обратной дуге приписываем –p(e)



находимим кратчайший по удельной стоимости (1,6)-путь:

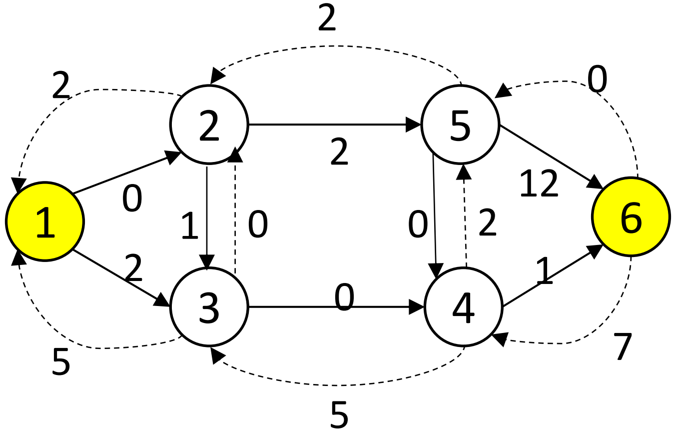


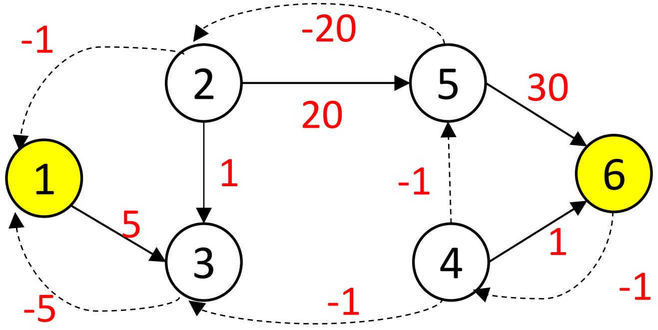
ФПМИ БГУ



5-я итерация

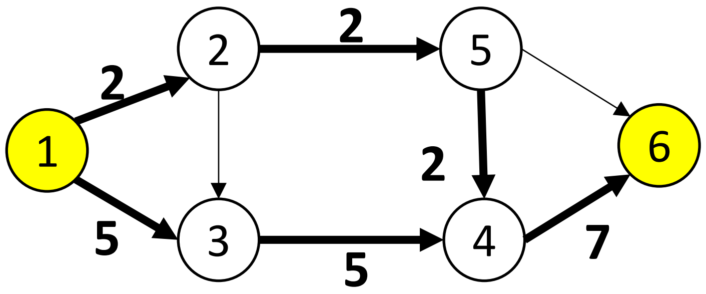
сеть остаточных
пропускных способностей





оставляем дуги, для которых остаточная пропускная способность >0;
дуге e исходной сети приписываем p(e),
а её обратной дуге приписываем –p(e)

НЕТ (1,6)-пути



$M(f) = 7$

$$P(f) = f(1,2)\cdot p(1,2) + f(2,5)\cdot p(2,5) + f(5,4)\cdot p(5,4) + f(1,3)\cdot p(1,3) + \\ + f(3,4)\cdot p(3,4) + f(4,6)\cdot p(4,6) = 2\cdot 1 + 2\cdot 20 + 2\cdot 1 + 5\cdot 5 + 5\cdot 1 + 7\cdot 1 \\ = 81$$

Время работы

$O\,(c^{max} \cdot m \cdot n^2)$	$O\,(p^{max} \cdot m^2 \cdot n^2)$
$O(c^{max} \cdot n) \quad \cdot \quad (O(n \cdot m) + O(m))$	$O(p^{max} \cdot n \cdot m) \quad \cdot \quad (O(n \cdot m) + O(m))$
так как сеть целочисленная, нет кратных дуг и на каждой итерации метода минимальных путей строится поток большей величины, то число итераций алгоритма ограничено сверху наибольшей возможной величиной потока конкретной индивидуальной задачи, т.е. $c^{max} \cdot n$.	число шагов запуска алгоритма нахождения кратчайшего пути (доказательство оценки выполняется по той же схеме, как и в алгоритме Эдмондса – Карпа)
время работы алгоритма Форда – Беллмана; модификация потока вдоль найденного увеличивающего пути.	время работы алгоритма Форда – Беллмана; модификация потока вдоль найденного увеличивающего пути.

Максимальный поток минимальной стоимости

Метод устранения отрицательных циклов

$$O(c^{max} \cdot p^{max} \cdot m \cdot n^2 + n \cdot m^2)$$

Метод минимальных путей

$$O(c^{max} \cdot m \cdot n^2)$$

$$O(p^{max} \cdot m^2 \cdot n^2)$$

оба алгоритма – псевдополиномиальные

Общие задачи в iRunner для закрепления навыков

[0.11 Максимальный поток в сети (простая версия)]
[0.12 Максимальный поток в сети (большие ограничения, по желанию)]

Курс лекций
по алгоритмам и структурам данных
з а в е р ш ё н ….

Никогда не останавливайтесь, расширяйте и углубляйте свои знания – это того стоит!