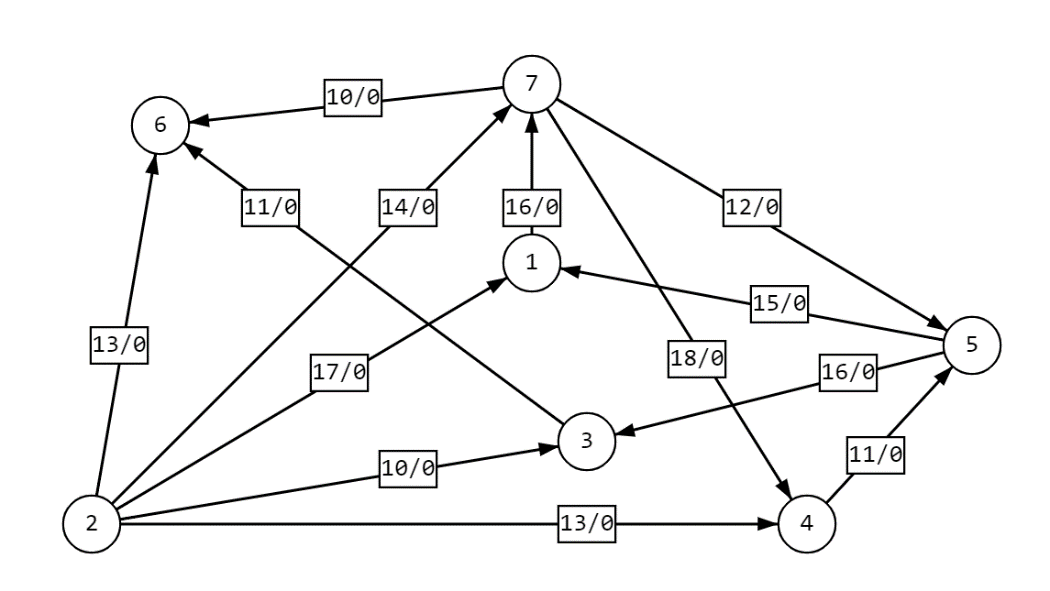
**Лабораторная работа**

**Вариант № 4**

**Задача о максимальном потоке. Алгоритм Форда-Фалкерсона**

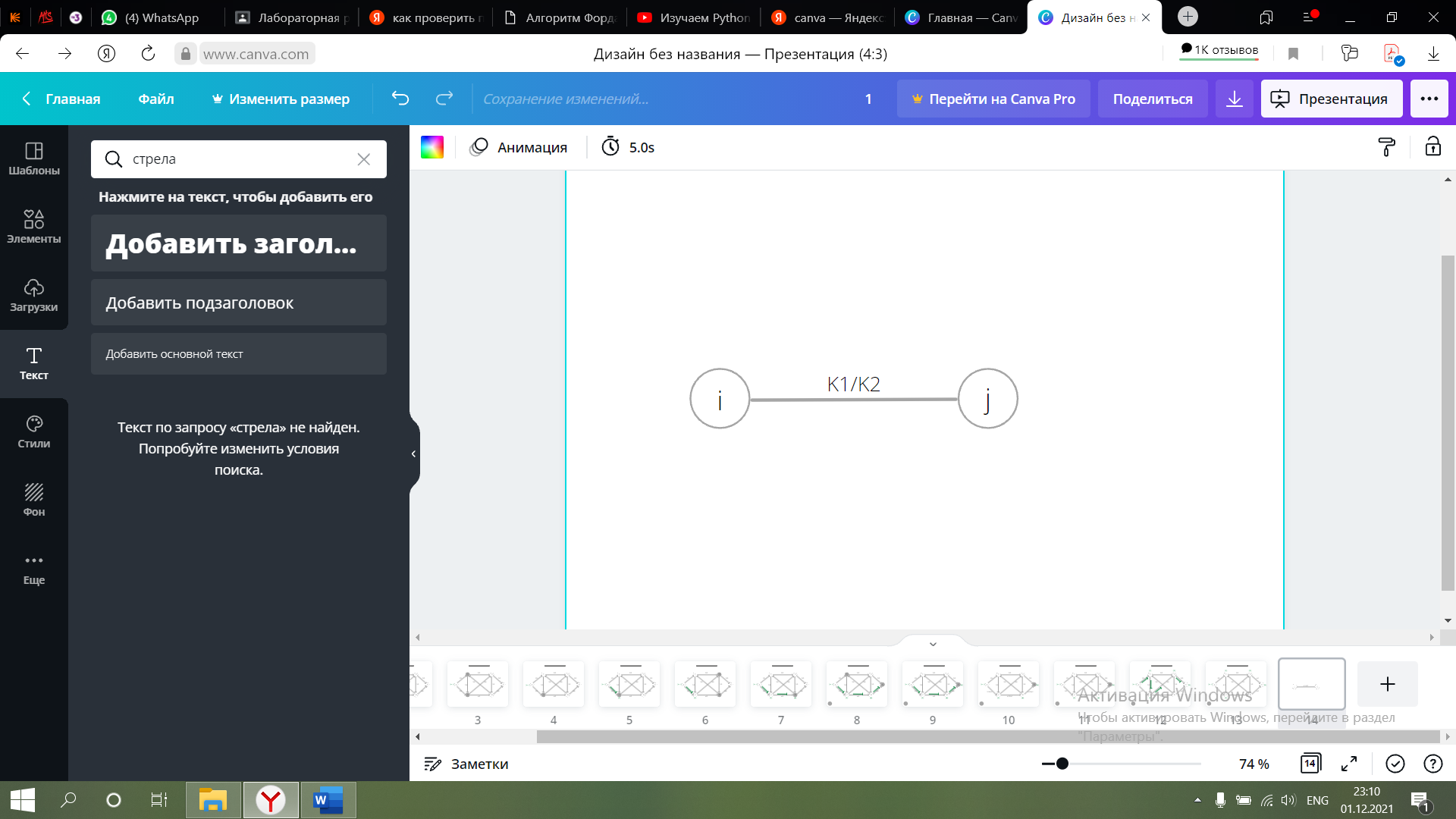
1. Составляю граф на основе индивидуального задания (7;13); 10-18.



1. Реализуем алгоритм Форда-Фалкерсона.

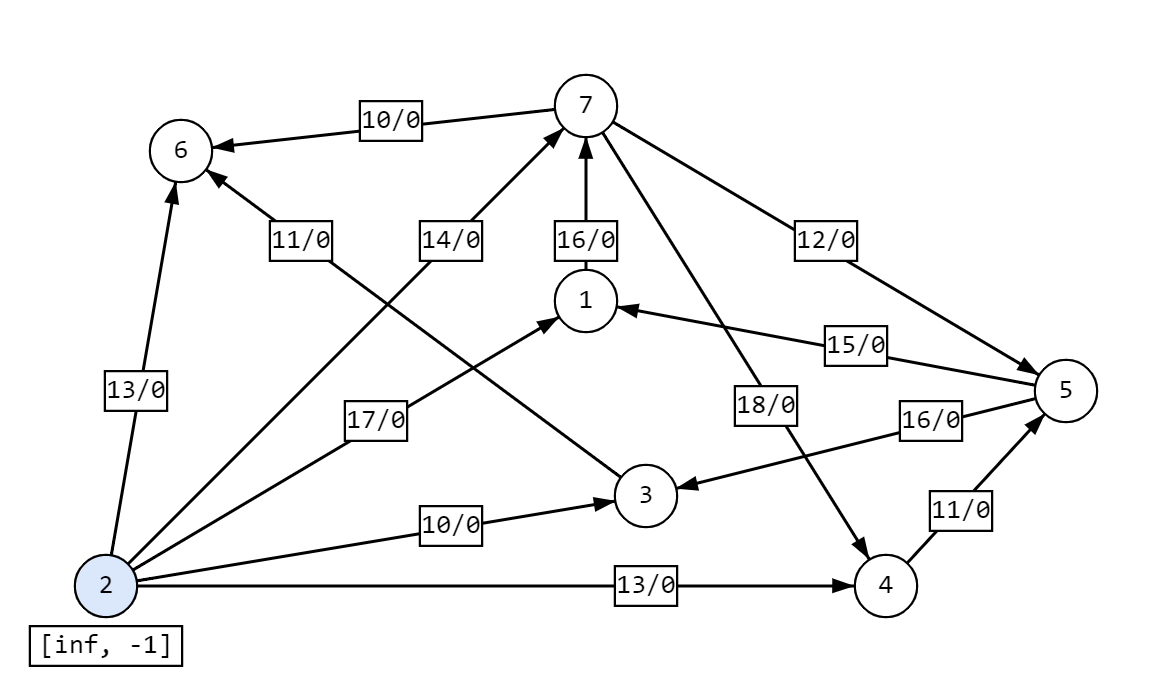
s-источник, t-сток.

Изначально через рёбра не протекает никакой поток, поэтому значения =0.

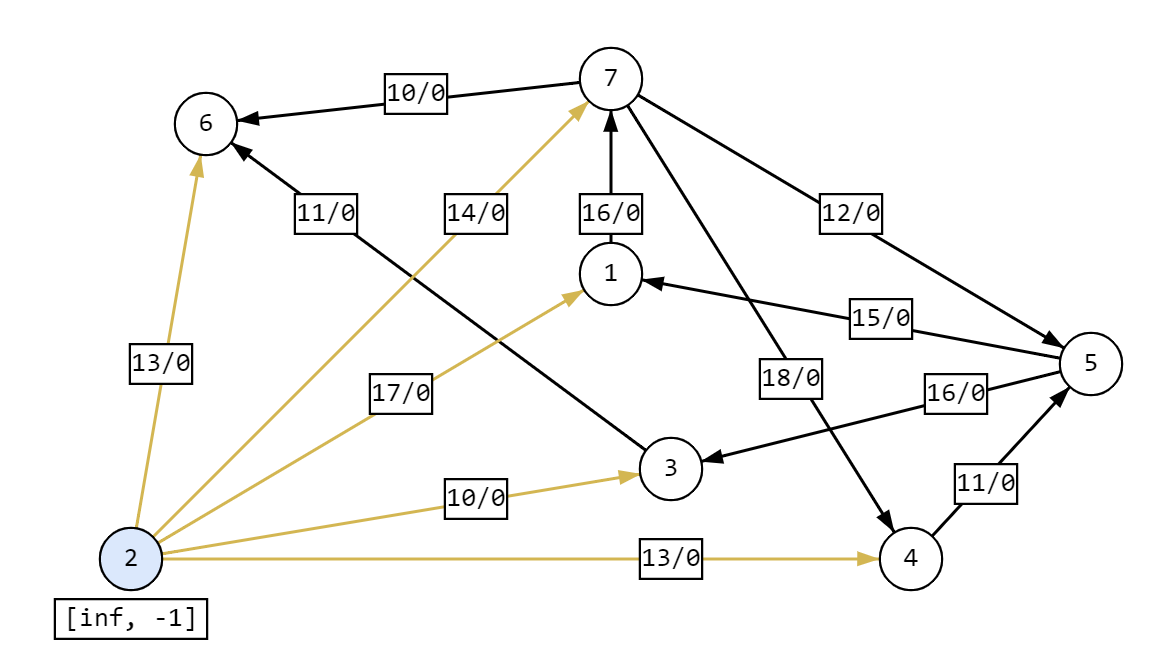


Решение:

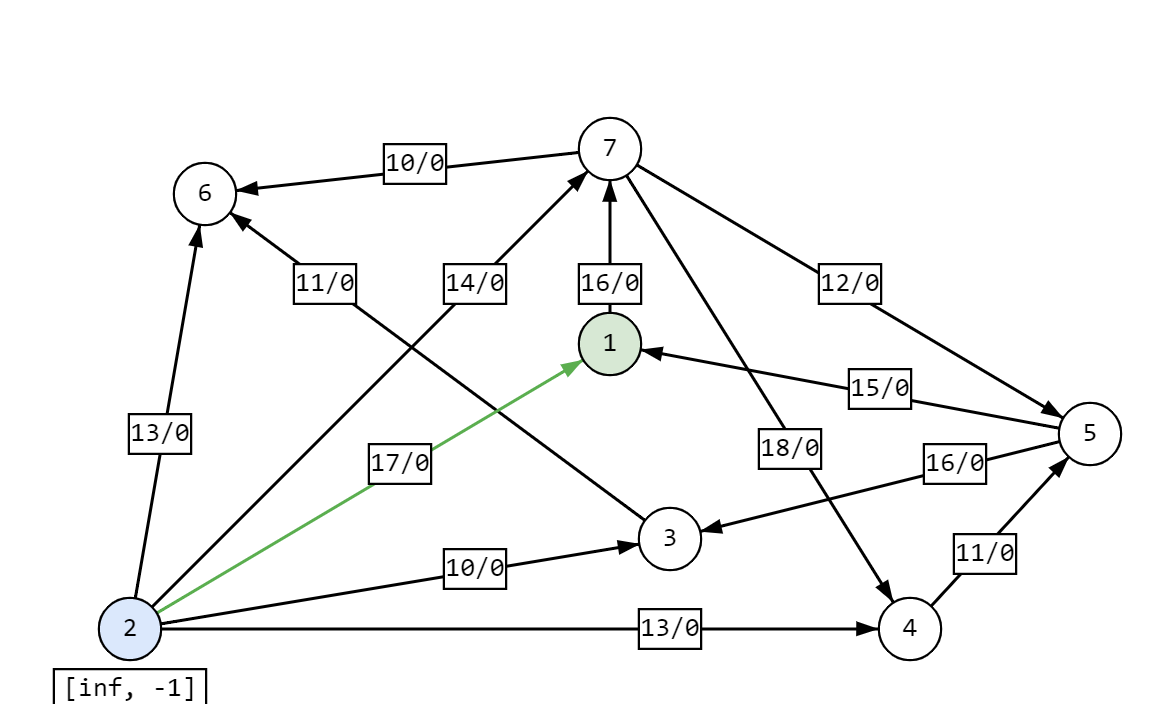
2.1. Начинаем движение от истока т.е. от вершины 2, помечаем 2, как исток.



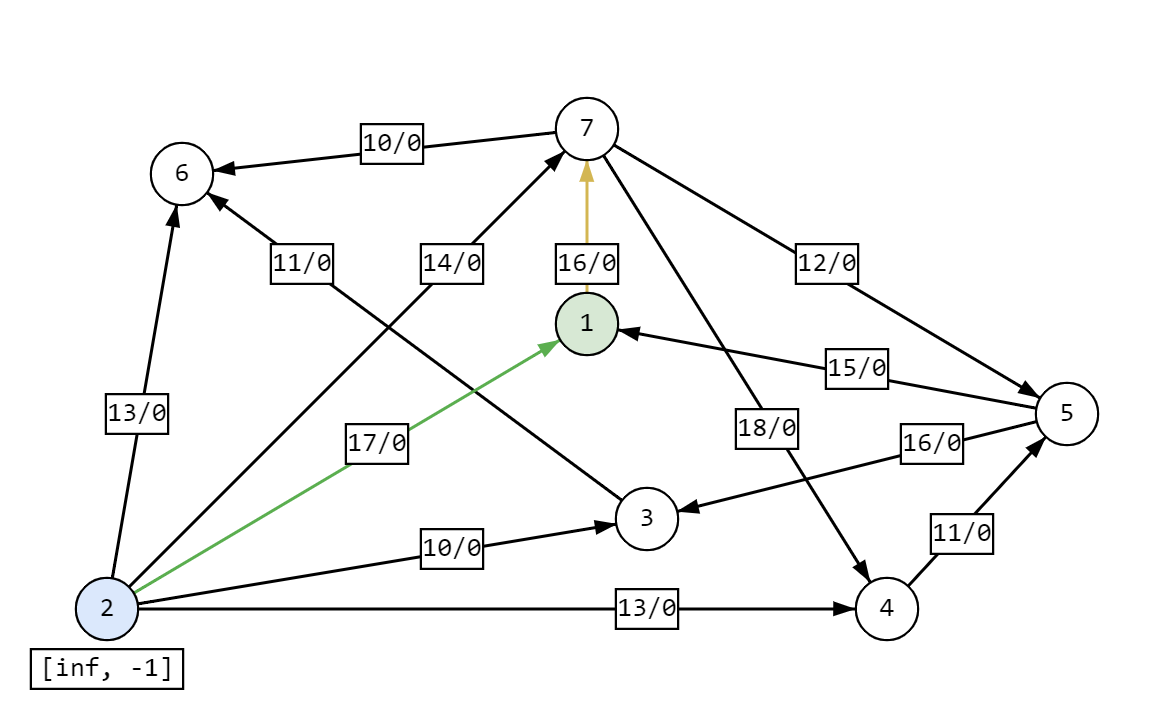
2.2 Находим множество вершин смежных с вершиной 2.



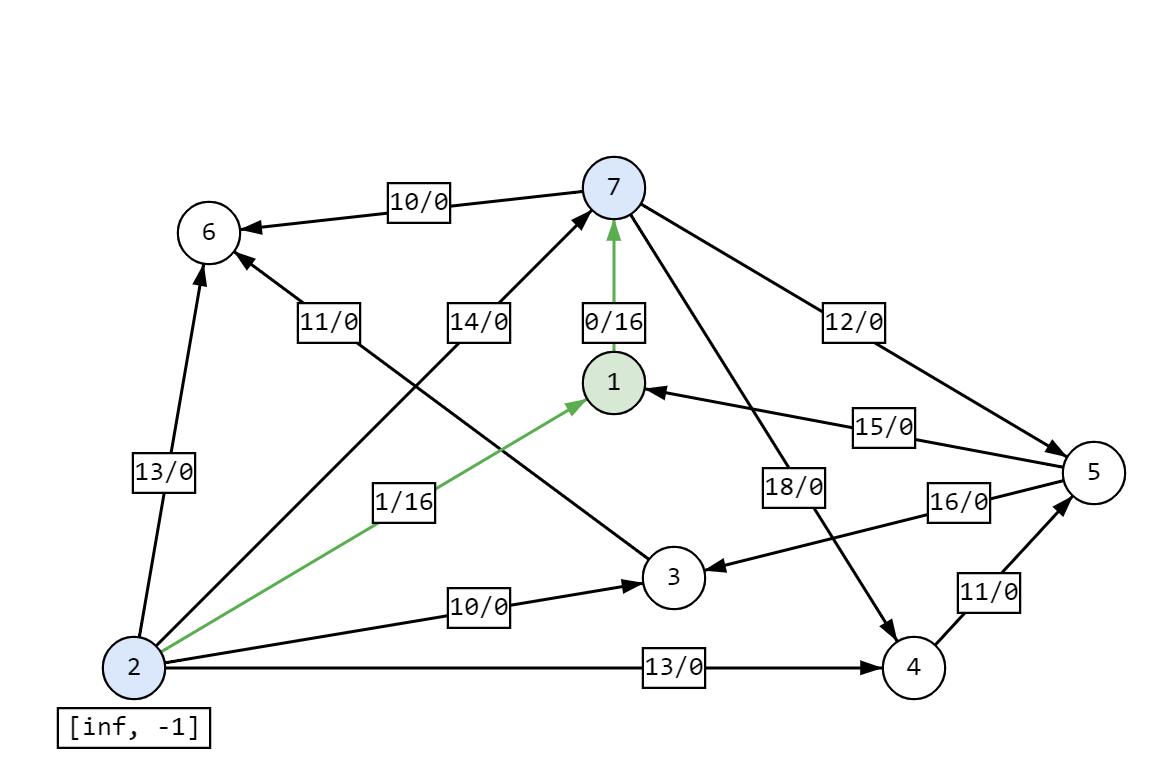
2.3 Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 2 и 1, пропускная способность равна 17. Помечаем вершину 1, т.к. она не является стоком.



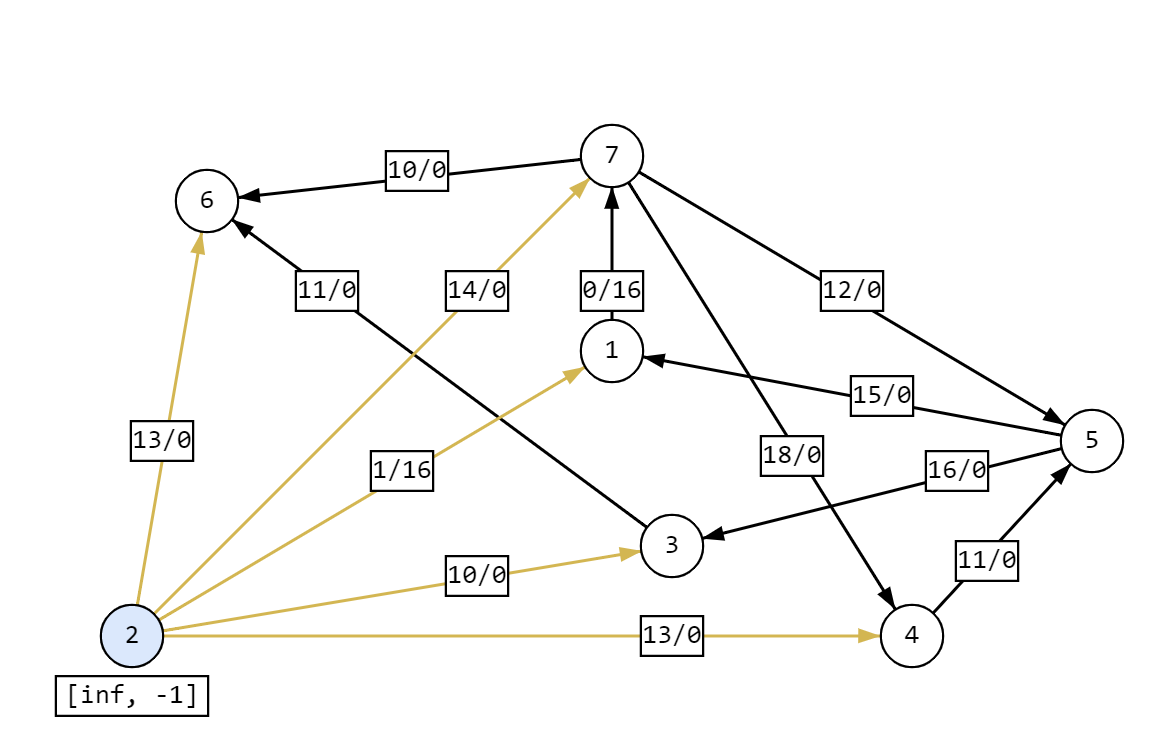
2.4 Находим множество вершин смежных с вершиной 1.



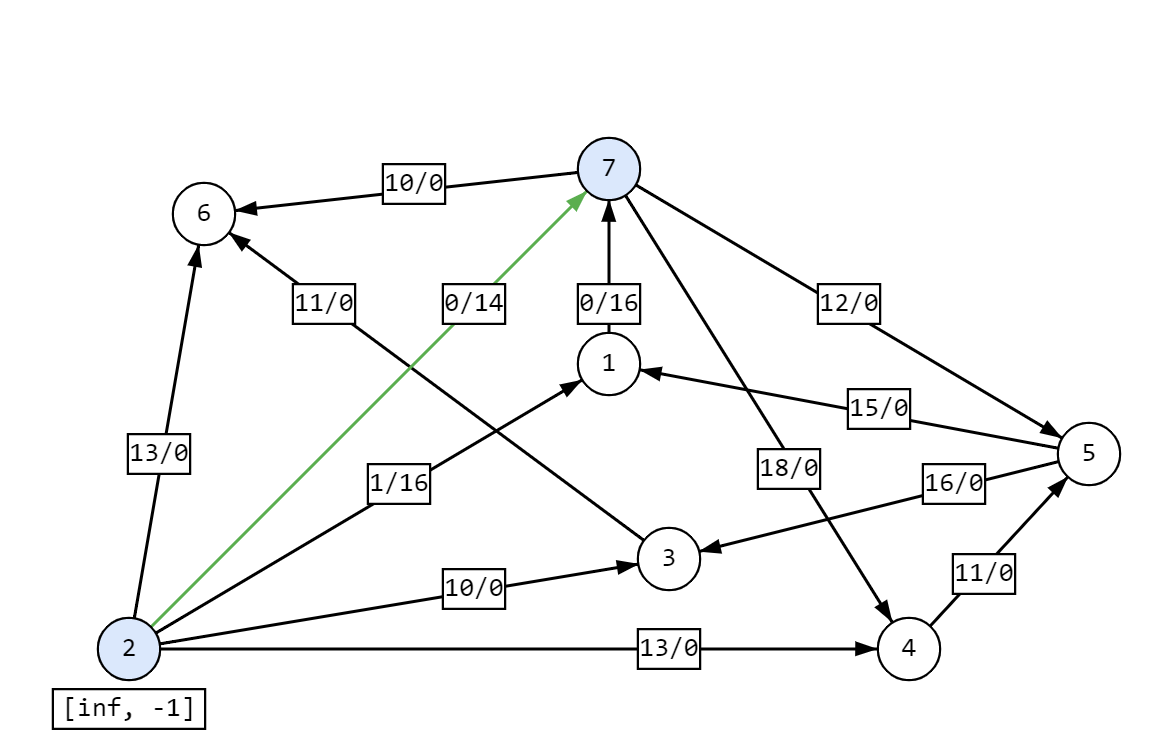
2.5 Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 1 и 7, пропускная способность равна 16. Вершина 7 является стоком => путь 2 -> 1 -> 7, min(17, 16) = 16. Изменяем веса дуг.



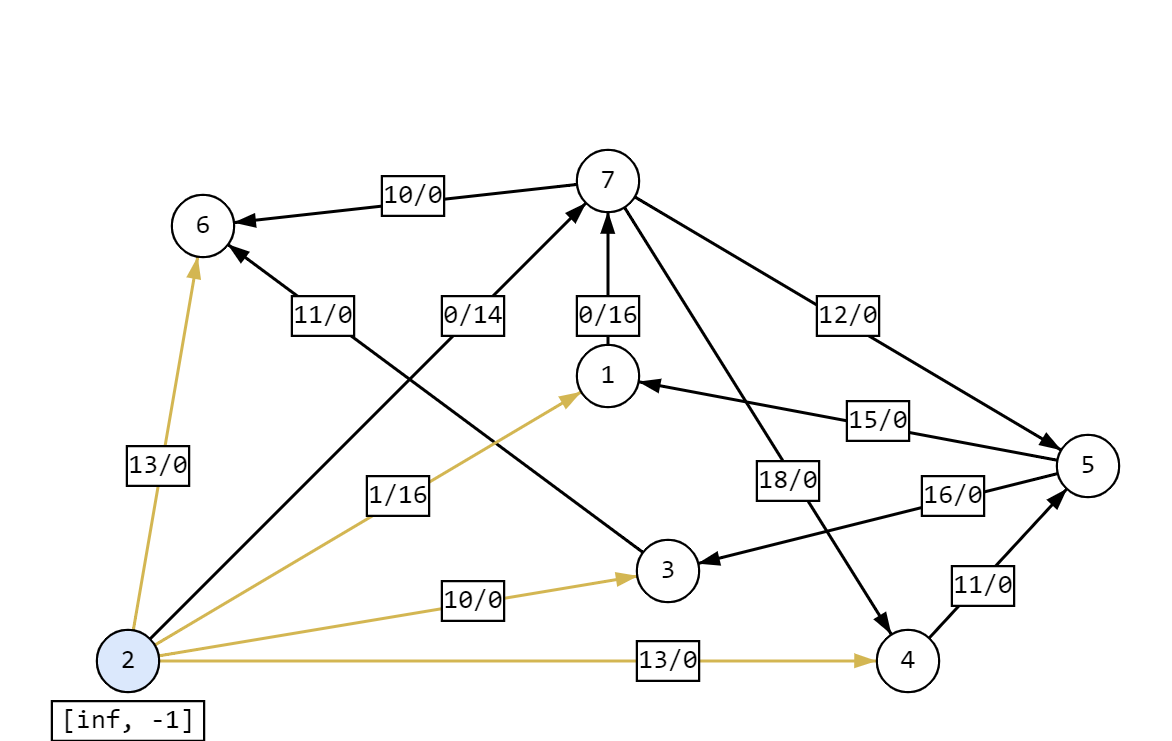
2.6 Находим множество вершин смежных с вершиной истока.



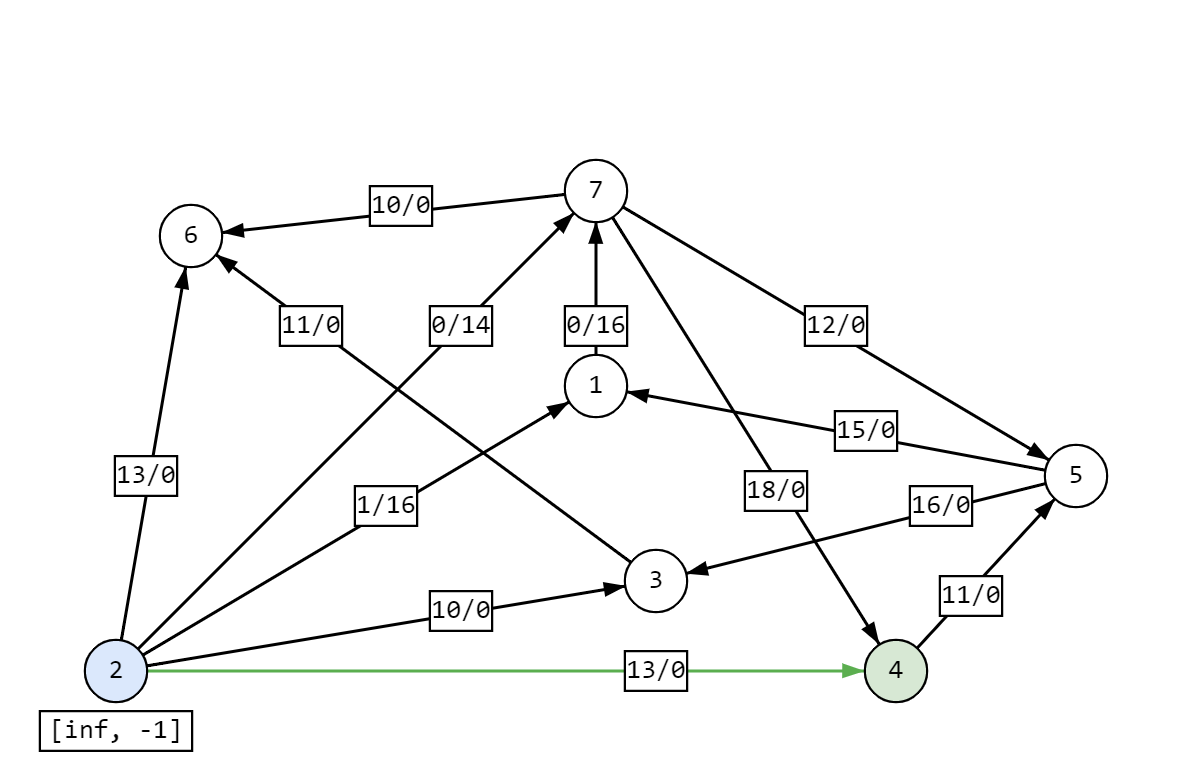
2.7 Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 2 и 7, пропускная способность равна 14. Вершина 7 является стоком => путь 2 -> 7, min(14) = 14. Изменяем веса дуг.



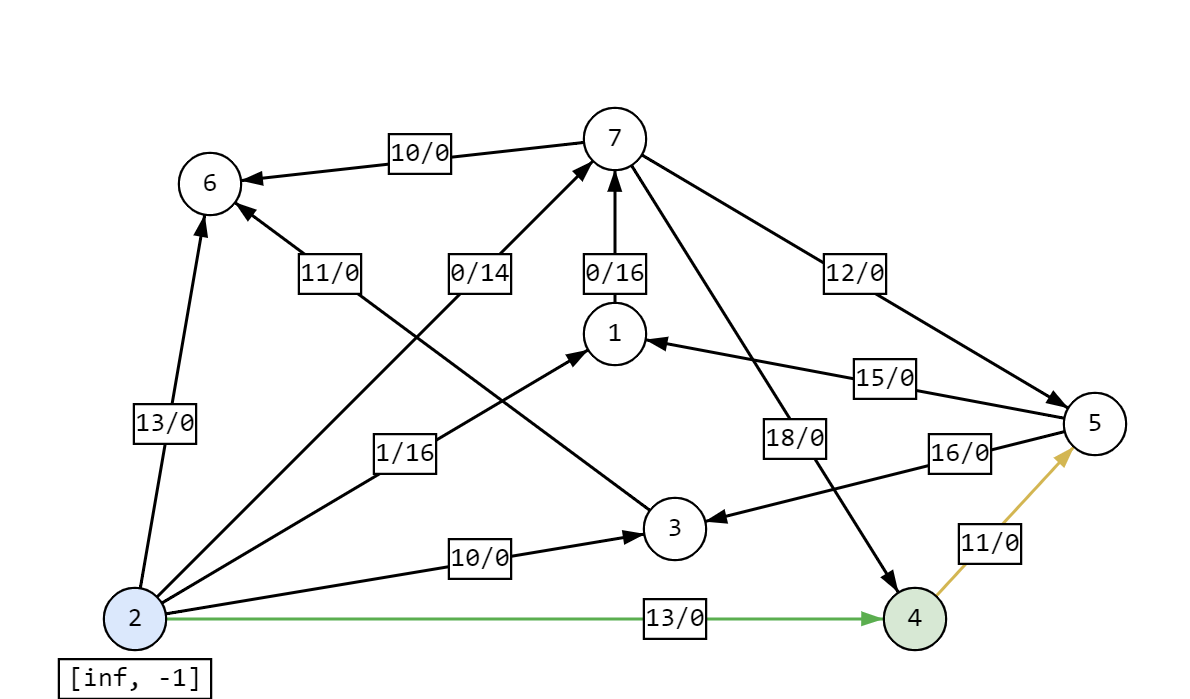
2.8 Находим множество вершин смежных с вершиной истока.



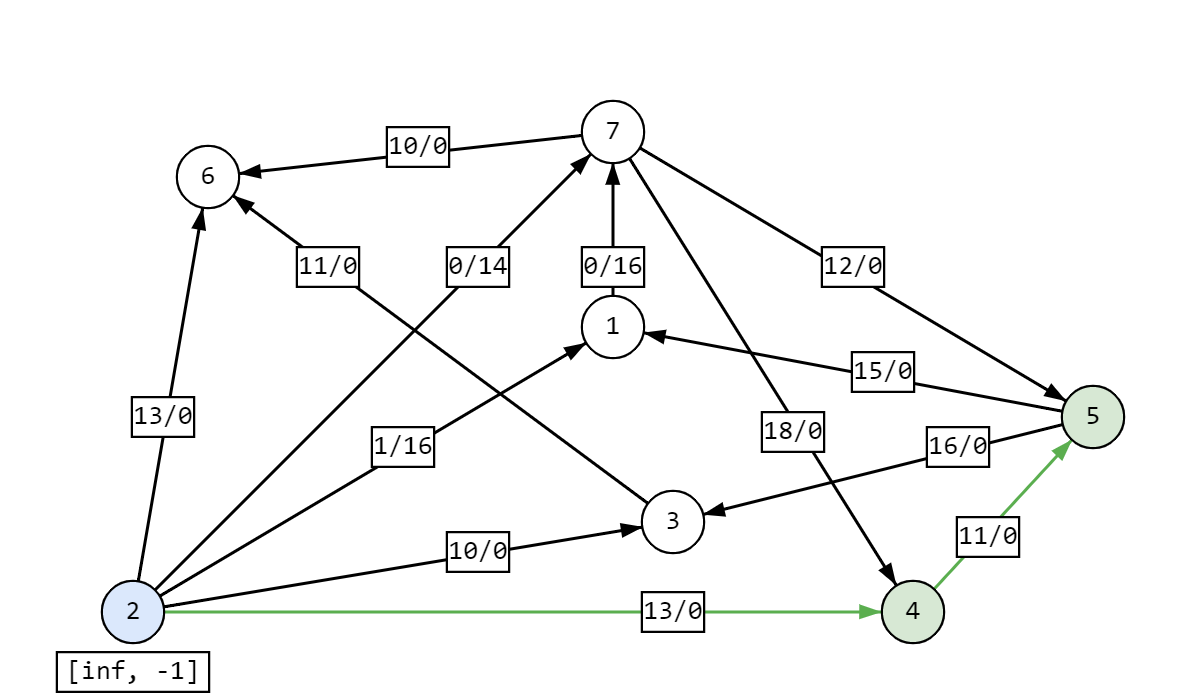
2.9 Находим такую вершину, чтобы было наибольшим, из них ту, номер которой наименьший. В нашем случае, это ребро, соединяющее вершины 2 и 4, пропускная способность равна 13. Помечаем вершину 4, т.к. она не является стоком.



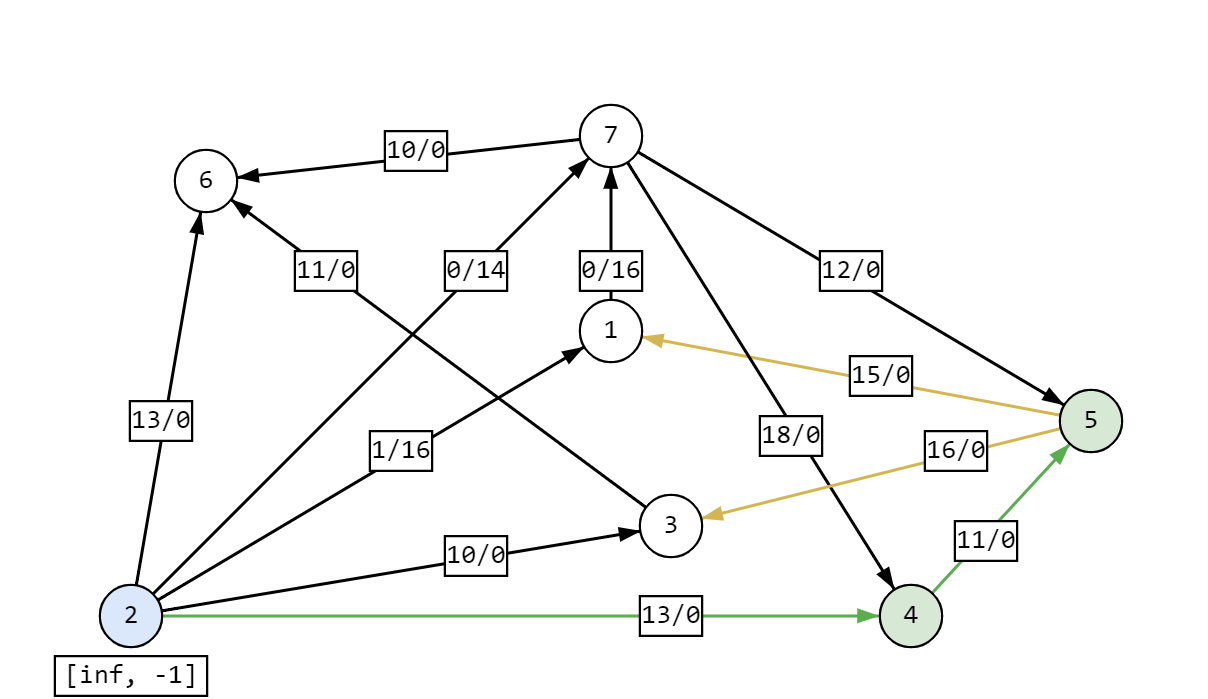
2.10 Находим множество вершин смежных с вершиной 4.



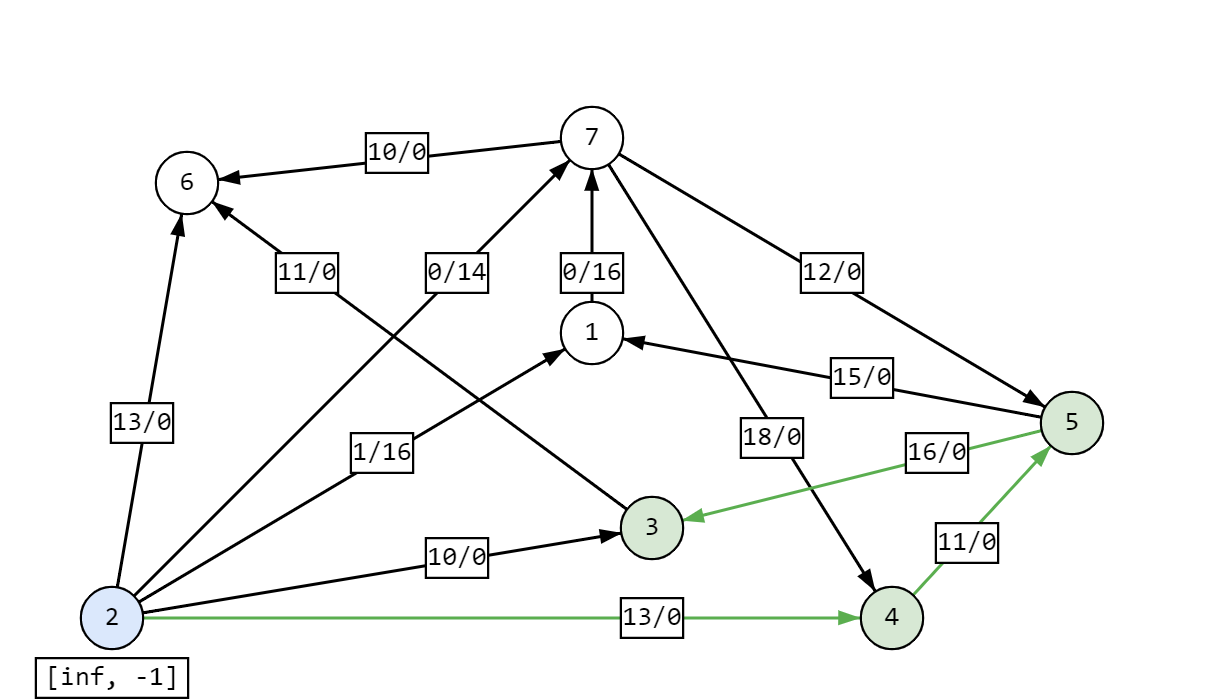
2.11 Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 4 и 5, пропускная способность равна 11. Помечаем вершину 5, т.к. она не является стоком.



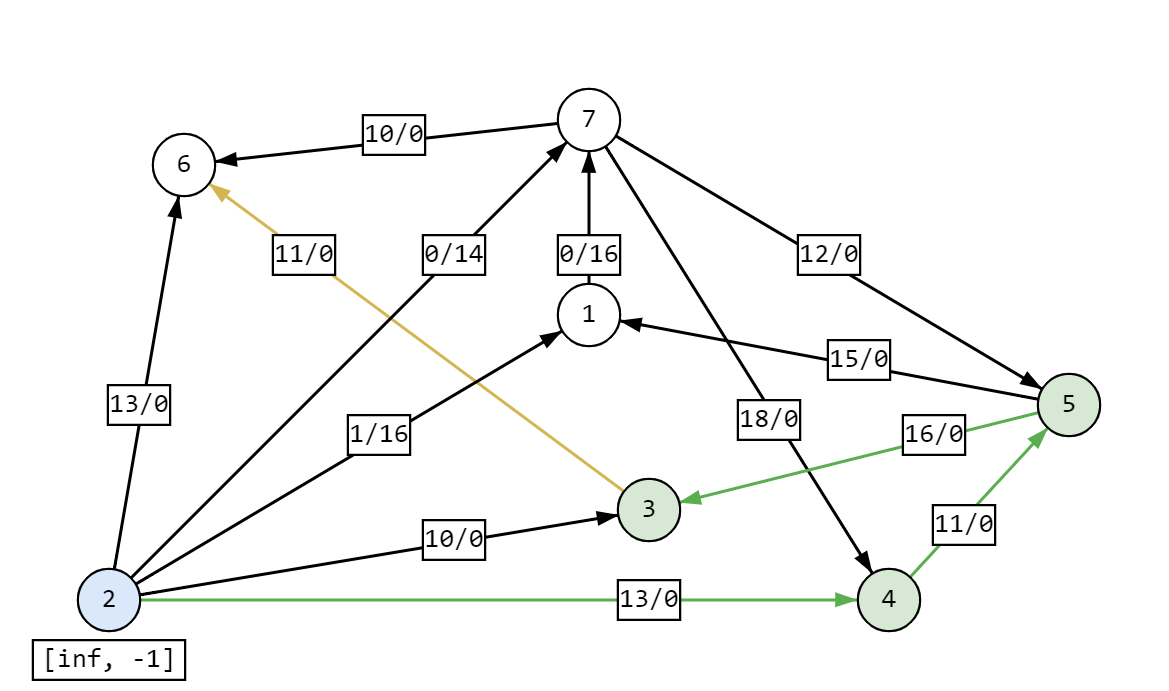
2.12 Находим множество вершин смежных с вершиной 5.



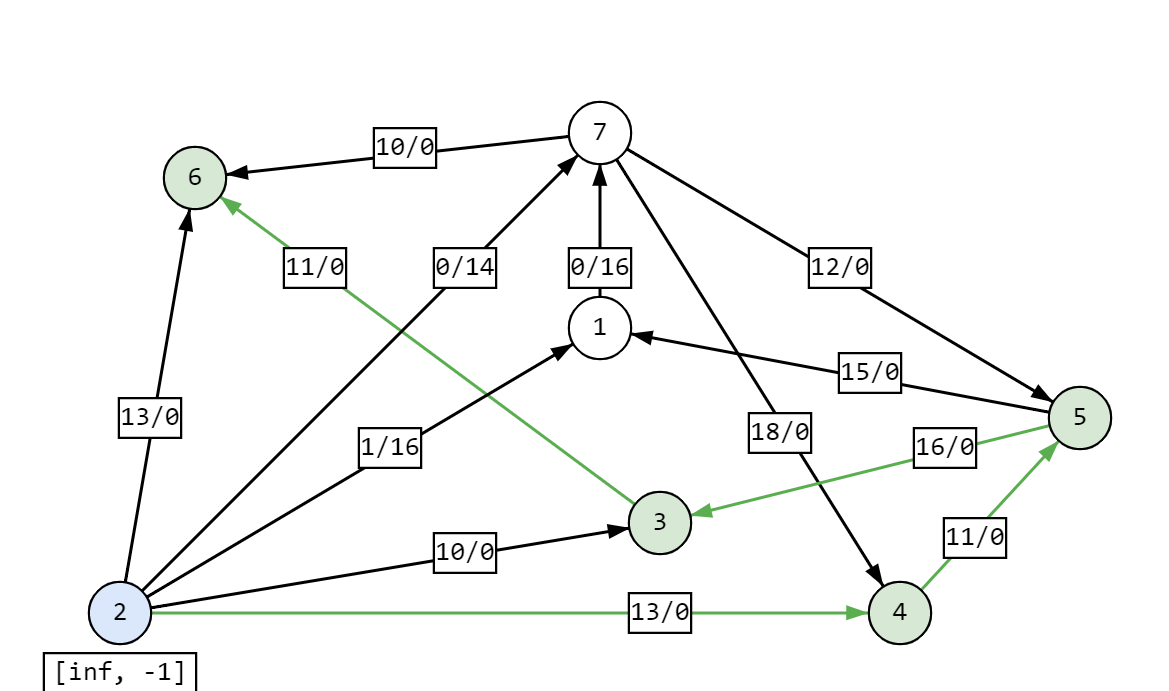
2.13 Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 5 и 3, пропускная способность равна 16. Помечаем вершину 3, т.к. она не является стоком.



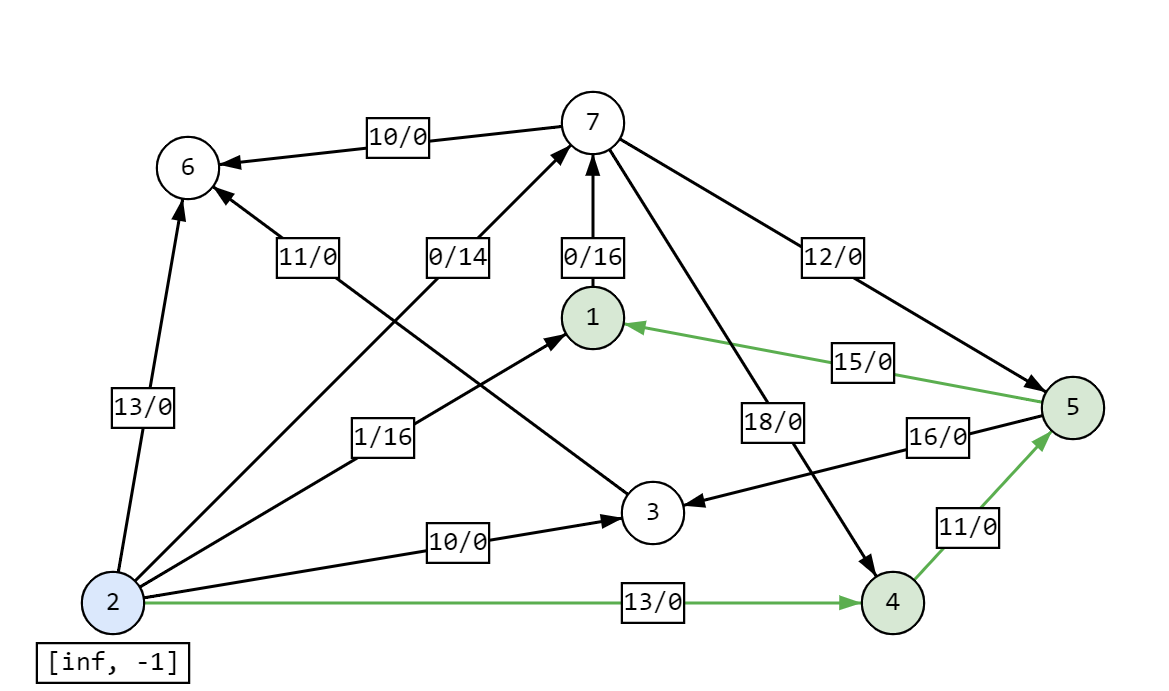
2.14 Находим множество вершин смежных с вершиной 3.



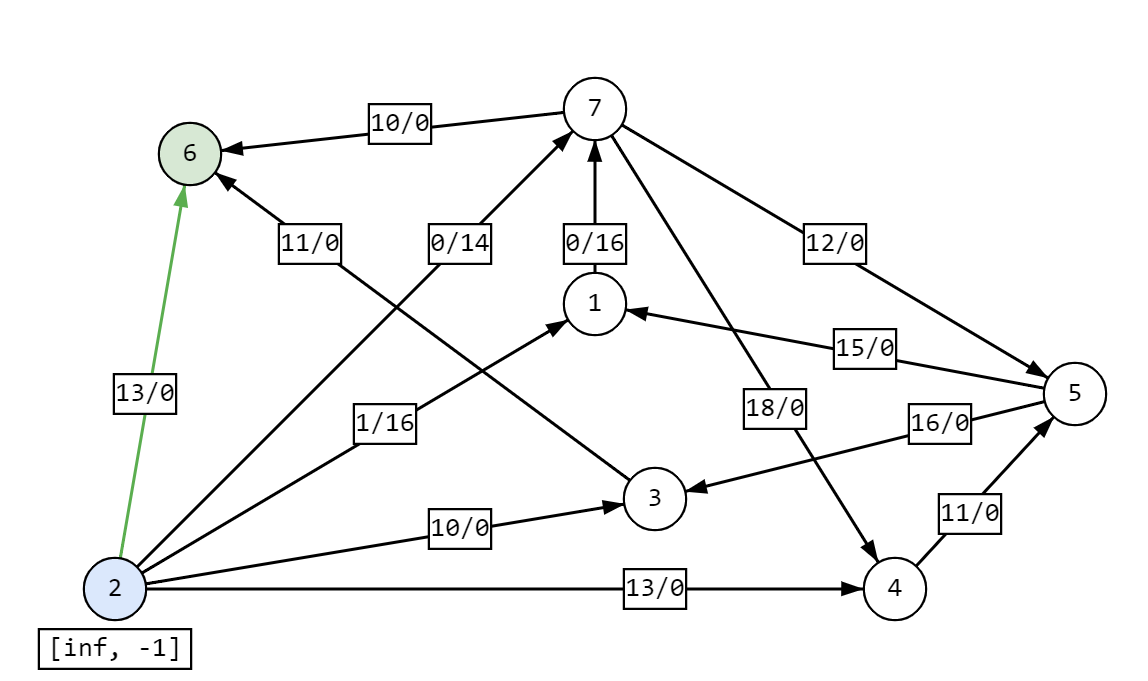
2.15 Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 3 и 6, пропускная способность равна 11. Помечаем вершину 6, т.к. она не является стоком.



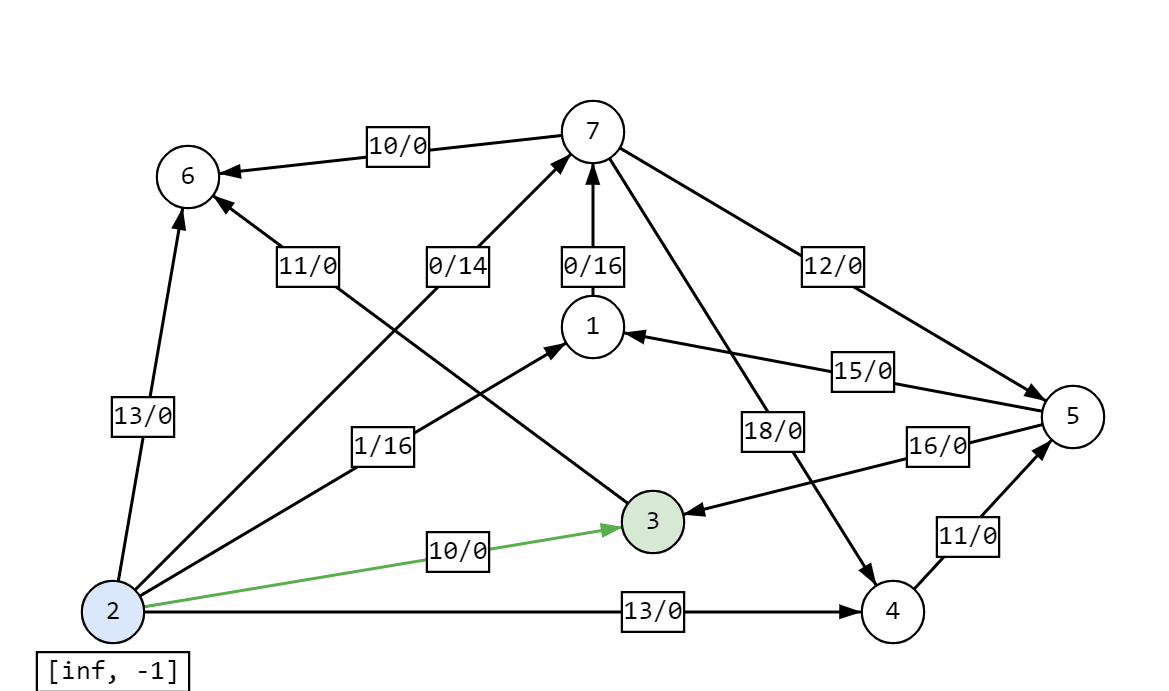
2.16 Из вершины 6 невозможно попасть ни в одну вершину => возвращаемся в предыдущую вершину. Из вершины 3 невозможно никуда попасть, кроме вершины 6 => возвращаемся в предыдущую вершину. Из вершины 5 возможно попасть только в вершину 1, если не учитывать вершину 3 => идем в вершину 1.



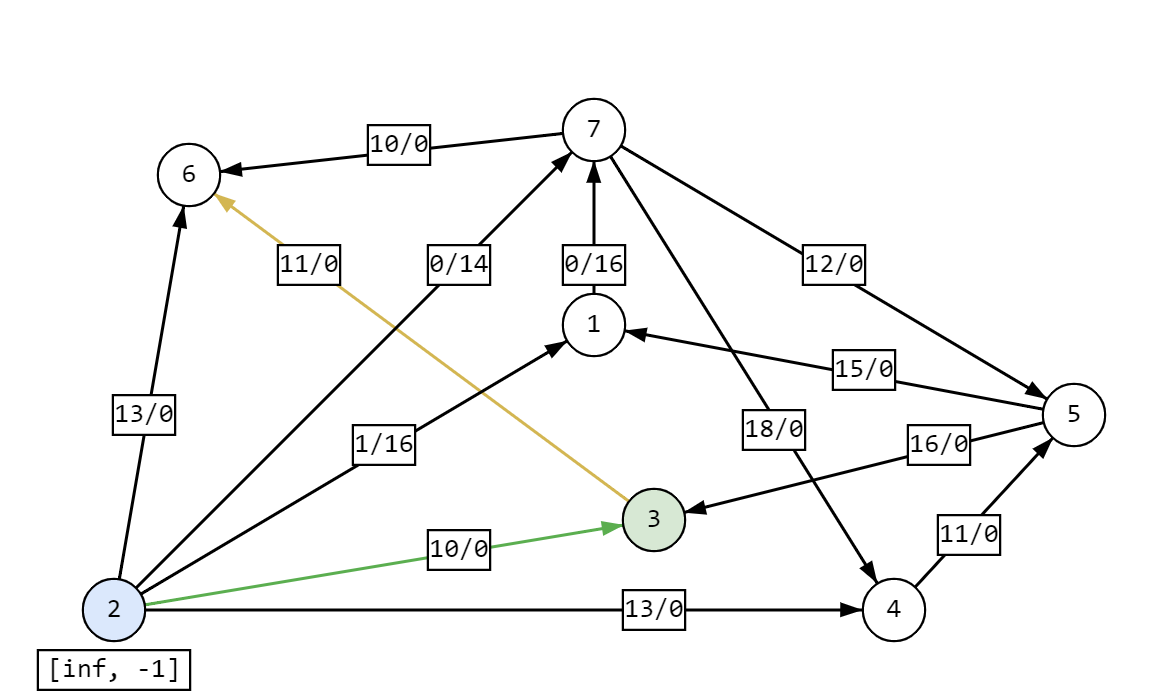
2.17 Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 1 и 2, но в вершине 2 мы уже были => возвращаемся на вершину назад. Из вершины 5 никуда невозможно пойти => возвращаемся на вершину назад. Из вершины 4 никуда невозможно попасть => возвращаемся в вершину 2. Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 2 и 6, пропускная способность равна 13. Помечаем вершину 6, т.к. она не является стоком.



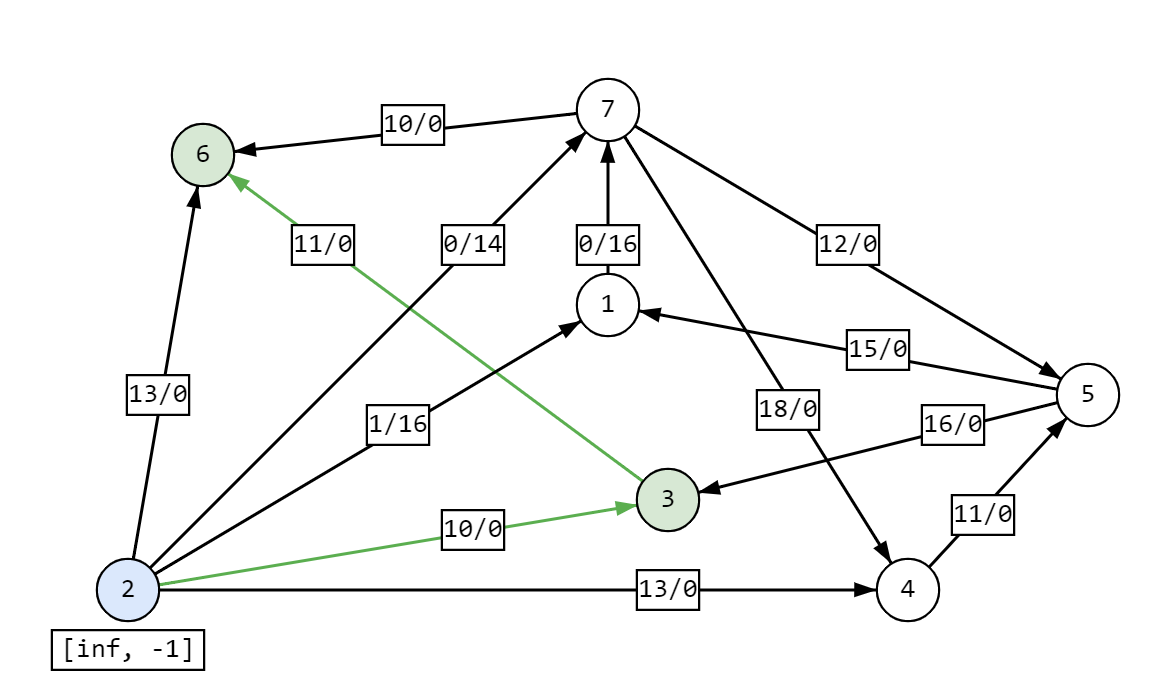
2.18 Находим множество вершин смежных с вершиной 6, таких вершин нет => возвращаемся на вершину назад. Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 2 и 3, пропускная способность равна 10. Помечаем вершину 3, т.к. она не является стоком.



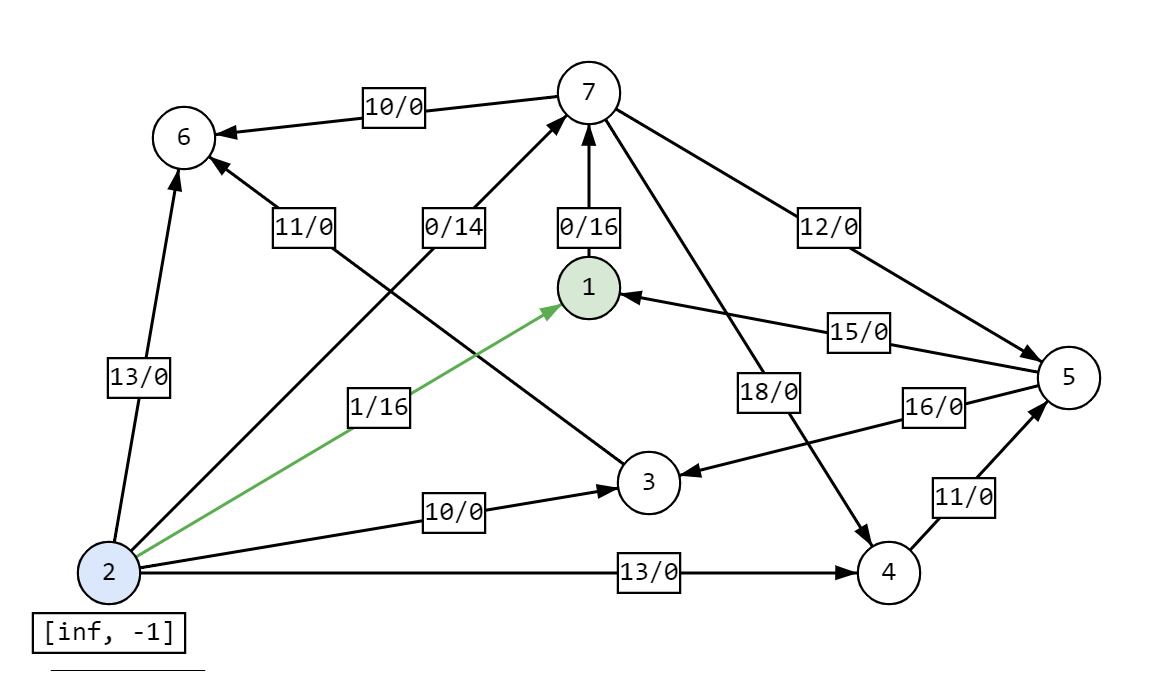
2.19 Находим множество вершин смежных с вершиной 3.



2.20 Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 3 и 6, пропускная способность равна 11. Помечаем вершину 6, т.к. она не является стоком.



2.16 Из вершины 6 невозможно попасть ни в одну вершину => возвращаемся в предыдущую вершину. Из вершины 3 невозможно никуда попасть, кроме вершины 6 => возвращаемся в предыдущую вершину. Находим такую вершину, чтобы было наибольшим. В нашем случае, это ребро, соединяющее вершины 2 и 1, пропускная способность равна 1. Помечаем вершину 1, т.к. она не является стоком.



2.13 Из вершины 1 возможно попасть только в вершину 5, из вершины 5 возможно попасть только в вершину 3, из вершины 3 возможно попасть только в вершину 6, а из вершины 6 невозможно пойти никуда => из вершины 1 нельзя попасть в сток => возвращаемся в вершину 2, а из нее невозможно никуда попасть => работа программы завершена, вычисляем максимальный поток.

*Максимальный поток равен сумме найденных: 14+16=30*

1. Коды программ, реализующий решение задачи о максимальном потоке. Алгоритм Форда-Фалкерсона. (код составлен на основе кода, написанного на лекции)

import math

def get\_max\_vertex(k, V, S):

m = 0 # наименьшее допустимое значение

v = -1

for i, w in enumerate(V[k]):

if i in S:

continue

if w[2] == 1: # движение по стрелке

if m < w[0]:

m = w[0]

v = i

else: # движение против стрелки

if m < w[1]:

m = w[1]

v = i

return v

def get\_max\_flow(T):

w = [x[0] for x in T]

return min(\*w)

def updateV(V, T, f):

for t in T:

if t[1] == -1: # это исток

continue

sgn = V[t[2]][t[1]][2] # направление движения

# меняем веса в таблице для (i,j) и (j,i)

V[t[1]][t[2]][0] -= f \* sgn

V[t[1]][t[2]][1] += f \* sgn

V[t[2]][t[1]][0] -= f \* sgn

V[t[2]][t[1]][1] += f \* sgn

V = [[[0,0,1], [17,0,-1], [0,0,1], [0,0,1], [15,0,-1], [0,0,1], [16, 0, 1]],

[[17,0,1], [0,0,1], [10,0,1], [13,0,1], [0,0,1], [13,0,1], [14,0,1]],

[[0,0,1], [10,0,-1], [0,0,1], [0,0,1], [16,0,-1], [11,0,1], [0,0,1]],

[[0,0,1], [13,0,-1], [0,0,1], [0,0,1], [11,0,1], [0,0,1], [18,0,-1]],

[[15,0,1], [0,0,1], [16,0,1], [11,0,-1], [0,0,1], [0,0,1], [12,0,-1]],

[[0,0,1], [13,0,-1], [11,0,-1], [0,0,1], [0,0,1], [0,0,1], [10,0,-1]],

[[16,0,-1], [14,0,-1], [0,0,1], [18,0,1], [12,0,1], [10,0,1], [0,0,1]]

]

N = len(V) # число вершин в графе

init = 0 # вершина истока (нумерация с нуля)

end = 5 # вершина стока

Tinit = (math.inf, -1, init) # первая метка маршруто (a, from, vertex)

f = [] # максимальные потоки найденных маршрутов

j = init

while j != -1:

k = init # стартовая вершина (нумерация с нуля)

T = [Tinit] # метки маршрута

S = {init} # множество просмотренных вершин

while k != end: # пока не дошли до стока

j = get\_max\_vertex(k, V, S) # выбираем вершину с наибольшей пропускной способностью

if j == -1: # если следующих вершин нет

if k == init: # и мы на истоке, то

break # завершаем поиск маршрутов

else: # иначе, переходим к предыдущей вершине

k = T.pop()[2]

continue

c = V[k][j][0] if V[k][j][2] == 1 else V[k][j][1] # определяем текущий поток

T.append((c, j, k)) # добавляем метку маршрута

S.add(j) # запоминаем вершину как просмотренную

if j == end: # если дошди до стока

f.append(get\_max\_flow(T)) # находим максимальную пропускную способность маршрута

updateV(V, T, f[-1]) # обновляем веса дуг

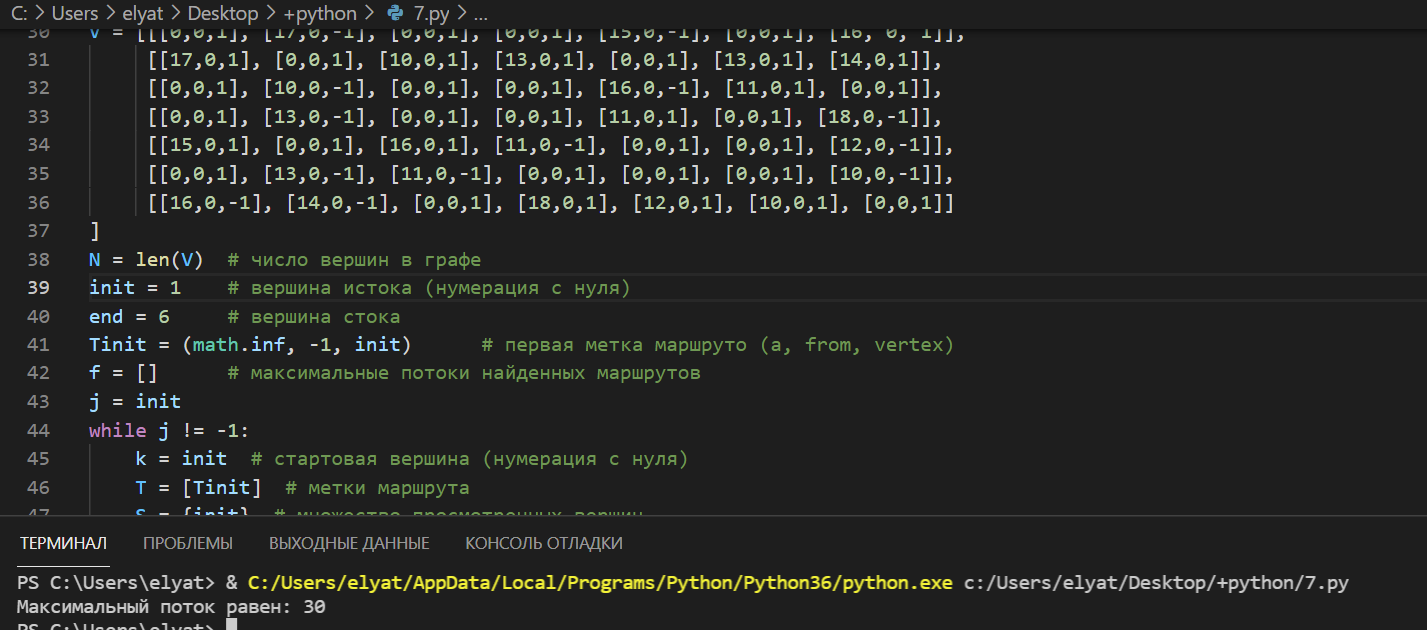
break

k = j

F = sum(f)

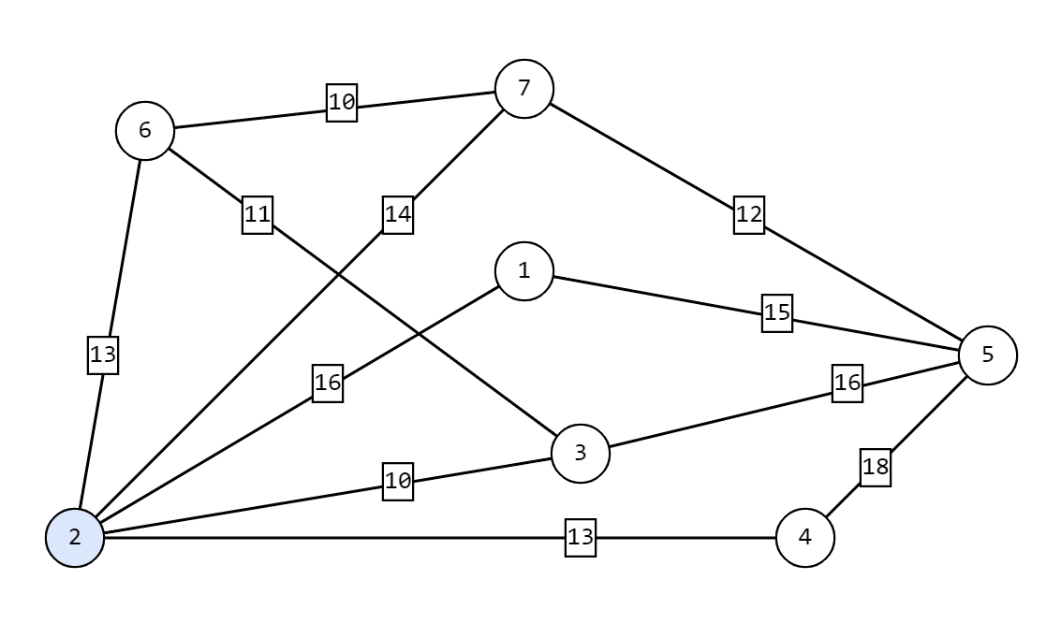
print(f"Максимальный поток равен: {F}")

1. Скриншот с результатом работы программы, реализующей алгоритм Форда-Фалкерсона.



**Задача о кратчайшем пути. Алгоритма Дейкстры.**

1. Составляю граф на основе индивидуального задания (7;11); 10-18.



1. Реализую алгоритм Дейкстры.

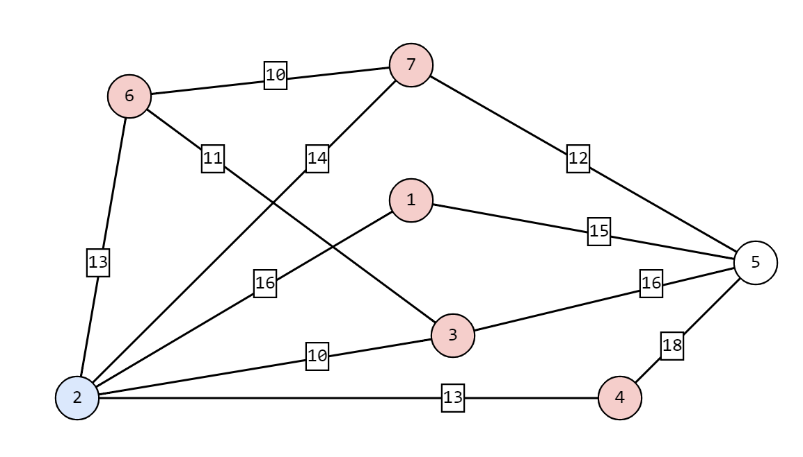
2.1 Устанавливаем расстояние Т[i] до всех остальных вершин = ∞

2.2 Полагаем T[2]=0

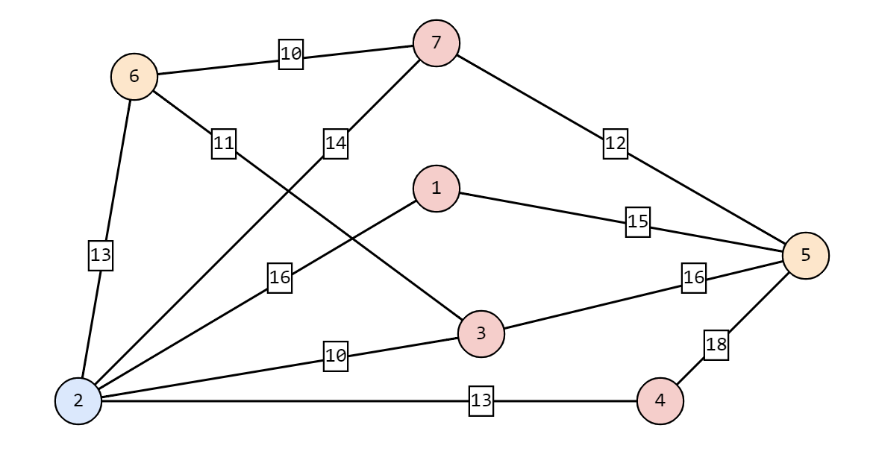
2.3. Запускаю цикл просмотра всех вершин

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |

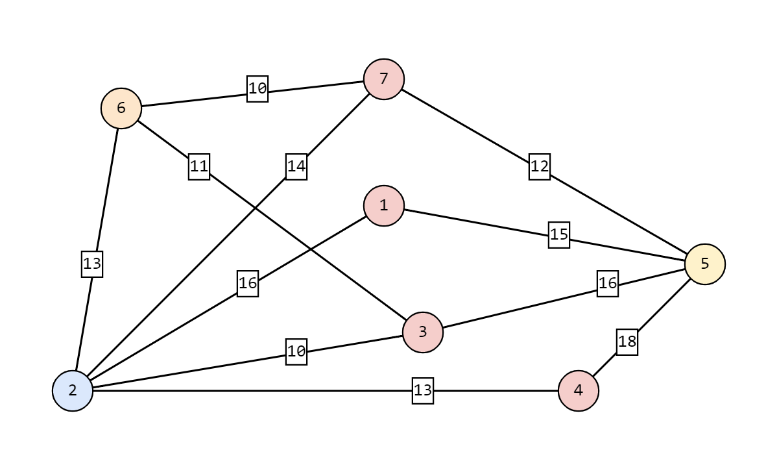
2.4. Перебираю все вершины, связанные с вершиной v, если вершина не просмотрена, т.е. не в множестве S, то формирую для нее вес. Если новый вес вершины меньше того, что уже содержится для нее, то перезаписываю его в T[j] выбираю вершину с наименьшим весом из еще не просмотренных. Продолжаю алгоритм, пока все вершины не просмотрены.



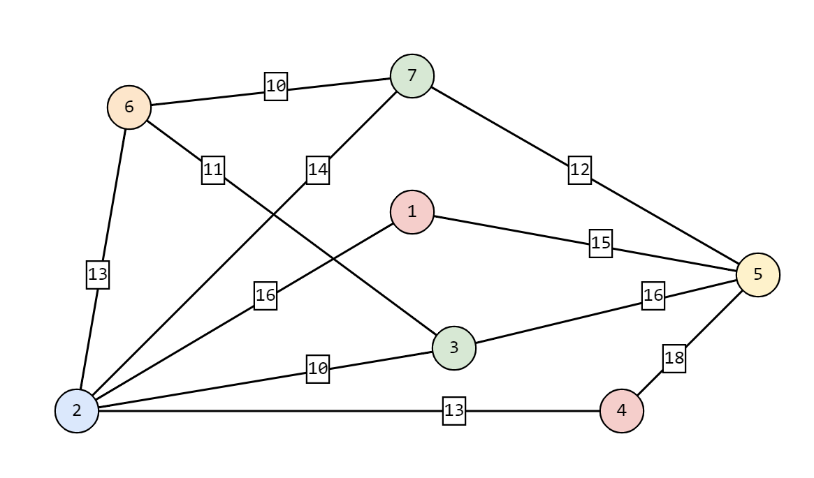
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 16 | 0 | 10 | 13 | ∞ | 13 | 14 |



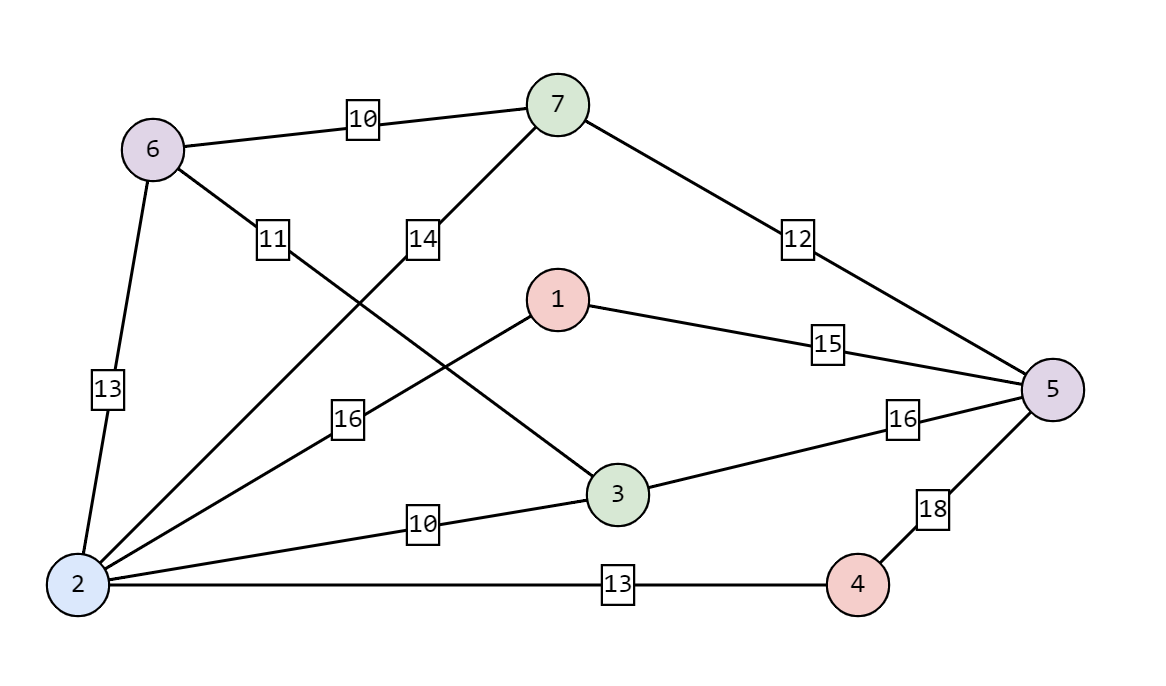
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 16 | 0 | 10 | 13 | ∞ | 13 | 14 |
| 3 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |



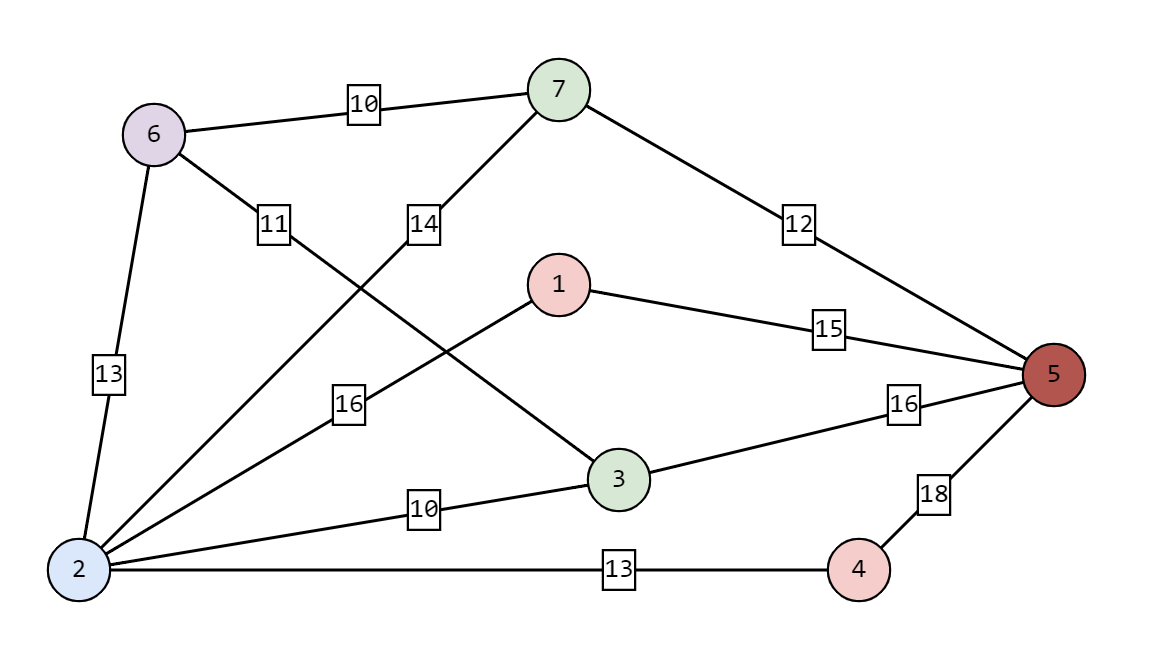
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 16 | 0 | 10 | 13 | ∞ | 13 | 14 |
| 3 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |
| 4 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |



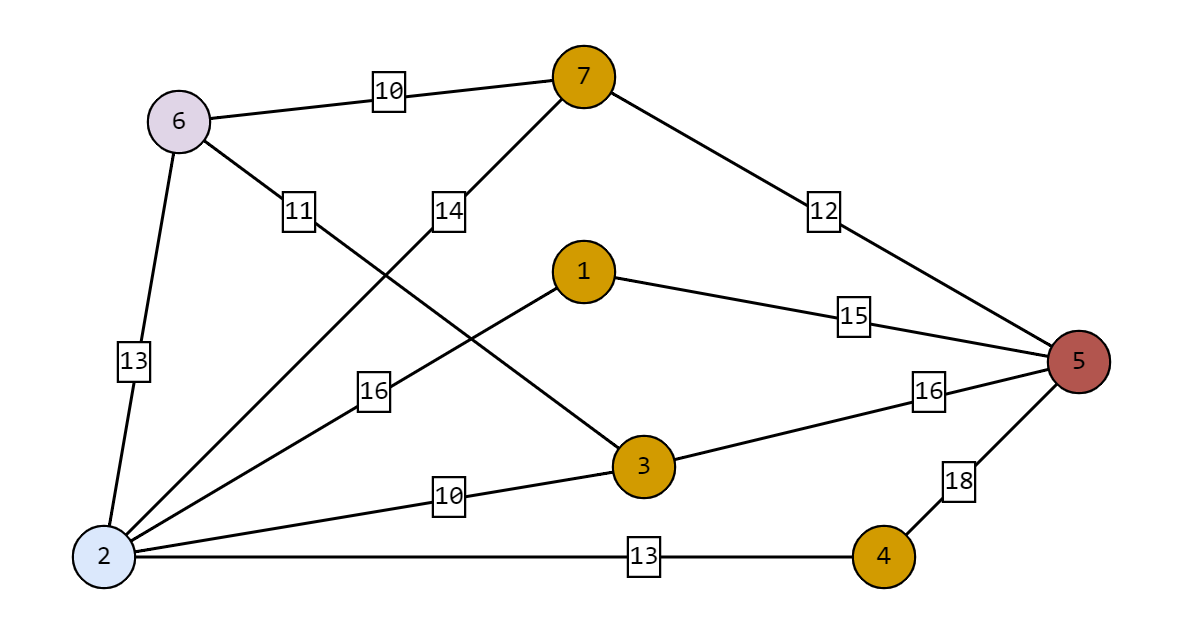
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 16 | 0 | 10 | 13 | ∞ | 13 | 14 |
| 3 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |
| 4 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |
| 5 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |

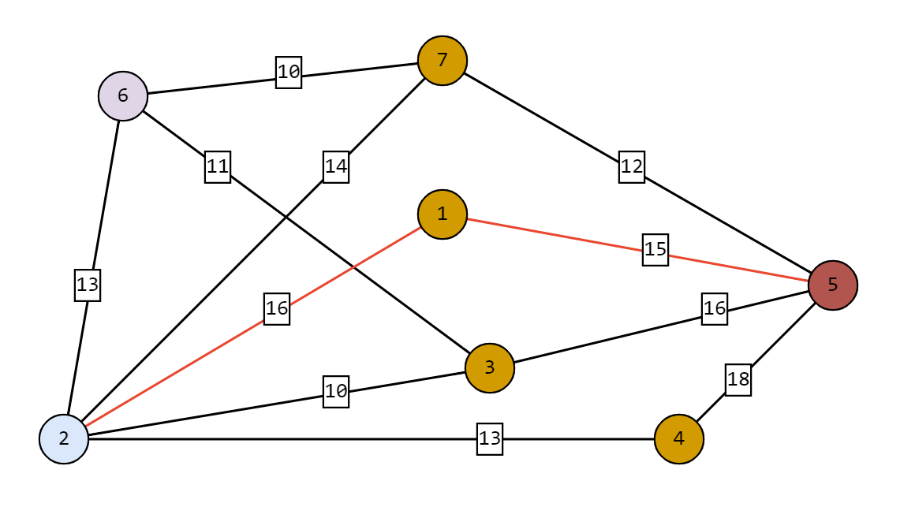


|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 16 | 0 | 10 | 13 | ∞ | 13 | 14 |
| 3 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |
| 4 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |
| 5 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |
| 6 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 16 | 0 | 10 | 13 | ∞ | 13 | 14 |
| 3 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |
| 4 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |
| 5 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |
| 6 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |
| 7 | 16 | 0 | 10 | 13 | 26 | 13 | 14 |





Получаю *кратчайший путь* (выделен красным цветом), он *равен 26*.

1. Коды программ, реализующий решение задачи о минимальном пути. Алгоритм Дейкстры.

import math

def get\_link\_v(v,D):

for i, weight in enumerate(D[v]):

if weight>0:

yield i

def arg\_min(T, S):

amin = -1

m = math.inf # максимальное значение

for i, t in enumerate(T):

if t < m and i not in S:

m = t

amin = i

return amin

D = ((0, 16, 0, 0, 15, 0, 0),

(16, 0, 10, 13, 0, 13, 14),

(0, 10, 0, 0, 16, 11, 0),

(0, 13, 0, 0, 18, 0, 0),

(15, 0, 16, 18, 0, 0, 12),

(0, 13, 11, 0, 0, 0, 10),

(0, 14, 0, 0, 12, 10, 0))

N = len(D) # число вершин в графе

T = [math.inf]\*N # последняя строка таблицы

v = 3 # стартовая вершина (нумерация с нуля)

S = {v} # просмотренные вершины

T[v] = 0 # нулевой вес для стартовой вершины

while v != -1: # цикл, пока не просмотрим все вершины

for j in get\_link\_v(v, D): # перебираем все связанные вершины с вершиной v

if j not in S: # если вершина еще не просмотрена

w = T[v] + D[v][j]

if w < T[j]:

T[j] = w

v = arg\_min(T, S) # выбираем следующий узел с наименьшим весом

if v >= 0: # выбрана очередная вершина

S.add(v) # добавляем новую вершину в рассмотрение

print(T)

1. Скриншот с результатом работы программы, реализующей алгоритм Форда-Фалкерсона.

Изображение выглядит как текст

Автоматически созданное описание