МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»

Тема: Потоки в сети

Студент гр. 8382 _____ Чирков С.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучение работы алгоритма Форда-Фалкерсона для нахождения максимального потока в сети.

Задание.

Найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро, используя алгоритм Форда-Фалкерсона.

Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа – пропускной способности (веса). Пример входных и выходных данных представлен на рисунке 1.

Входные данные:
N - количество ориентированных рёбер графа
v_0 - исток
v_n - сток
v_i v_j ω_{ij} - ребро графа
v_i v_j ω_{ij} - ребро графа
...

Выходные данные:
P_{max} - величина максимального потока
v_i v_j ω_{ij} - ребро графа с фактической величиной протекающего потока
v_i v_j ω_{ij} - ребро графа с фактической величиной протекающего потока
...

Рисунок 1. Входные и выходные данные

В ответе выходные рёбра отсортируйте в лексикографическом порядке по первой вершине, потом по второй (в ответе должны присутствовать все указанные входные рёбра, даже если поток в них равен 0).

Sample input.

```
7
a
f
a b 7
a c 6
b d 6
c f 9
d e 3
d f 4
e c 2
```

Sample output.

```
12
a b 6
a c 6
b d 6
c f 8
d e 2
d f 4
e c 2
```

Вариант дополнительного задания.

Вар. 6. Поиск не в глубину и не в ширину, а по правилу: каждый раз выполняется переход по дуге, соединяющей вершины, имена которых в алфавите ближе всего друг к другу. Если таких дуг несколько, то выбрать ту, имя конца которой в алфавите ближайшее к началу алфавита.

Описание алгоритма

Остаточная сеть — это граф с множеством ребер с положительной остаточной пропускной способностью. В остаточной сети может быть путь из u в v, даже если его нет в исходном графе (если в исходной сети есть путь (v, u) с положительным потоком).

Дополняющий путь — это путь в остаточной сети от истока до стока. Идея алгоритма заключается в том, чтобы запускать поиск по правилу (каждый раз выполняется переход по дуге, соединяющей вершины, имена которых в алфавите ближе всего друг к другу) в остаточной сети до тех пор, пока возможно найти новый путь от истока до стока. Вначале алгоритма остаточная сеть — это исходный граф. Алгоритм ищет дополняющий путь в остаточной сети по следующему алгоритму:

• Находим все смежные вершины к текущей рассматриваемой

• Переходим к вершине, имя которой ближе в алфавите.

• Повторяем предыдущие шаги для новой рассматриваемой вершины (алгоритм итеративный)

• Продолжаем, пока не дойдем до стока.

Если путь был найден, то остаточная сеть перестраивается, а к максимальному потоку прибавляется величина максимальной пропускной способности дополняющего пути.

Если путь от истока к стоку не был получен, то максимальный поток найден и алгоритм завершает свою работу.

Очевидно, что максимальный поток в сети является суммой всех пропускных способностей дополняющих путей.

Описание функций и структур данных.

graph = {} – словарь, с помощью которого хранится сеть.

edges = [] – список изначальных данных о ребрах сети, с помощью которого строится ответ.

В программе используются встроенные функции языка.

Тестирование

Ввод	Вывод
4 a c a b 2 b a 2 b c 2 c b 2	2 a b 2 b a 0 b c 2 c b 0
5 a e a b 2 b a 2 a c 1 b d 3 d e 1	1 a b 1 a c 0 b a 0 b d 1 d e 1
7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2

16 a e a b 15 b a 15 a d 5 d a 5 a c 10 c a 10 b c 8 c b 8 c d 10 d c 10 c e 13 e c 13 b e 7 e b 7 d e 12 e d 12	30 a b 15 a c 10 a d 5 b a 0 b c 8 b e 7 c a 0 c b 0 c d 5 c e 13 d a 0 d c 0 d e 10 e b 0 e c 0 e d 0
2 k k k e 2 e k 1	0 e k 0 k e 0

Тестирование с промежуточными результатами

```
введите количество ребер
11
введите исток и сток (через enter)
a
x
введите ребра с величиной протекающего потока
a b 5
a c 5
a d 5
a z 10
b c 5
c d 8
d z 15
z x 50
f y 12
c f 12
y z 11


{'a': [['b', 5], ['c', 5], ['d', 5], ['z', 10]], 'b': [['a', 0], ['c', 5]], 'c': [['b', 0], ['d', 8],
['a', 0], ['f', 12]], 'd': [['c', 0], ['a', 0], ['z', 15]], 'z': [['y', 0], ['x', 50], ['d', 0], ['a',
0]], 'x': [['z', 0]], 'f': [['c', 0], ['y', 12]], 'y': [['z', 11], ['f', 0]]} - остаточный путь

построение пути

рассматриваемые рёбра [['b', 5], ['c', 5], ['d', 5], ['z', 10]] в вершине a

в путь добавлена вершина a с потоком 5

рассматриваемые рёбра [['b', 5], ['a', 0], ['c', 5], ['c', 5], ['d', 5], ['z', 10]] в вершине b

в путь добавлена вершина b с потоком 5

рассматриваемые рёбра [['c', 5], ['c', 5], ['b', 5], ['b', 0], ['d', 5], ['d', 8], ['a', 0], ['a', 0],
['f', 12], ['z', 10]] в вершине c

в путь добавлена вершина c с потоком 5

рассматриваемые рёбра [['d', 5], ['d', 8], ['c', 5], ['c', 5], ['c', 0], ['b', 5], ['b', 0], ['f', 12]
, ['a', 0], ['a', 0], ['a', 0], ['z', 10], ['z', 15]] в вершине d

в путь добавлена вершина d с потоком 12

рассматриваемые рёбра [['f', 12], ['d', 5], ['d', 8], ['c', 5], ['c', 5], ['c', 0], ['c', 0], ['b', 5]
, ['b', 0], ['a', 0], ['a', 0], ['a', 0], ['y', 12], ['z', 10], ['z', 15]] в вершине f

в путь добавлена вершина f с потоком 12

рассматриваемые рёбра [['y', 12], ['z', 10], ['z', 15], ['z', 11], ['f', 12], ['f', 0], ['d', 5], ['d'
, 8], ['c', 5], ['c', 5], ['c', 0], ['c', 0], ['b', 5], ['b', 0], ['a', 0], ['a', 0], ['a', 0]] в верш
```

```
ине y

в путь добавлена вершина y с потоком 10

рассматриваемые рёбра [['z', 10], ['z', 15], ['z', 11], ['y', 12], ['y', 0], ['x', 50], ['f', 12], ['f
', 0], ['d', 5], ['d', 8], ['d', 0], ['c', 5], ['c', 5], ['c', 0], ['c', 0], ['b', 5], ['b', 0], ['a',
0], ['a', 0], ['a', 0], ['a', 0]] в вершине z

в путь добавлена вершина z с потоком 50

из пути удалены лишние вершины

abcfyzx

в остаточном пути изменились рёбра из пути на 5

{'a': [['b', 0], ['c', 5], ['d', 5], ['z', 10]], 'b': [['a', 5], ['c', 0]], 'c': [['b', 5], ['d', 8],
['a', 0], ['f', 7]], 'd': [['c', 0], ['a', 0], ['z', 15]], 'z': [['y', 5], ['x', 45], ['d', 0], ['a',
0]], 'x': [['z', 5]], 'f': [['c', 5], ['y', 7]], 'y': [['z', 6], ['f', 5]]} - остаточный путь

построение пути

рассматриваемые рёбра [['b', 0], ['c', 5], ['d', 5], ['z', 10]] в вершине a

в путь добавлена вершина a с потоком 5

рассматриваемые рёбра [['c', 5], ['b', 0], ['b', 5], ['d', 5], ['d', 8], ['a', 0], ['f', 7], ['z', 10]
] в вершине c

в путь добавлена вершина c с потоком 5

рассматриваемые рёбра [['b', 0], ['b', 5], ['a', 0], ['a', 5], ['c', 5], ['c', 0], ['d', 5], ['d', 8],
['f', 7], ['z', 10]] в вершине b

в путь добавлена вершина b с потоком 5

рассматриваемые рёбра [['d', 5], ['d', 8], ['c', 5], ['c', 0], ['c', 0], ['b', 0], ['b', 5], ['f', 7],
['a', 0], ['a', 5], ['a', 0], ['z', 10], ['z', 15]] в вершине d

в путь добавлена вершина d с потоком 7

рассматриваемые рёбра [['f', 7], ['d', 5], ['d', 8], ['c', 5], ['c', 0], ['c', 0], ['c', 5], ['b', 0],
['b', 5], ['a', 0], ['a', 5], ['a', 0], ['y', 7], ['z', 10], ['z', 15]] в вершине f

в путь добавлена вершина f с потоком 7

рассматриваемые рёбра [['y', 7], ['z', 10], ['z', 15], ['z', 6], ['f', 7], ['f', 5], ['d', 5], ['d', 8
], ['c', 5], ['c', 0], ['c', 0], ['c', 5], ['b', 0], ['b', 5], ['a', 0], ['a', 5], ['a', 0]] в вершине
y
```

```
в путь добавлена вершина y с потоком 10

рассматриваемые рёбра [['z', 10], ['z', 15], ['z', 6], ['y', 7], ['y', 5], ['x', 45], ['f', 7], ['f',
5], ['d', 5], ['d', 8], ['d', 0], ['c', 5], ['c', 0], ['c', 0], ['c', 5], ['b', 0], ['b', 5], ['a', 0]
, ['a', 5], ['a', 0], ['a', 0]] в вершине z

в путь добавлена вершина z с потоком 45

из пути удалены лишние вершины

acfyzx

в остаточном пути изменились рёбра из пути на 5

{'a': [['b', 0], ['c', 0], ['d', 5], ['z', 10]], 'b': [['a', 5], ['c', 0]], 'c': [['b', 5], ['d', 8],
['a', 5], ['f', 2]], 'd': [['c', 0], ['a', 0], ['z', 15]], 'z': [['y', 10], ['x', 40], ['d', 0], ['a',
0]], 'x': [['z', 10]], 'f': [['c', 10], ['y', 2]], 'y': [['z', 1], ['f', 10]]} - остаточный путь

построение пути

рассматриваемые рёбра [['b', 0], ['c', 0], ['d', 5], ['z', 10]] в вершине a

в путь добавлена вершина a с потоком 5

рассматриваемые рёбра [['d', 5], ['c', 0], ['c', 0], ['b', 0], ['a', 0], ['z', 10], ['z', 15]] в верши
не d

в путь добавлена вершина d с потоком 10

рассматриваемые рёбра [['z', 10], ['z', 15], ['y', 10], ['x', 40], ['d', 5], ['d', 0], ['c', 0], ['c',
0], ['b', 0], ['a', 0], ['a', 0]] в вершине z

в путь добавлена вершина z с потоком 10

рассматриваемые рёбра [['y', 10], ['x', 40], ['z', 10], ['z', 15], ['z', 1], ['f', 10], ['d', 5], ['d'
, 0], ['c', 0], ['c', 0], ['b', 0], ['a', 0], ['a', 0]] в вершине y

в путь добавлена вершина y с потоком 40

из пути удалены лишние вершины

adzx

в остаточном пути изменились рёбра из пути на 5

{'a': [['b', 0], ['c', 0], ['d', 0], ['z', 10]], 'b': [['a', 5], ['c', 0]], 'c': [['b', 5], ['d', 8],
['a', 5], ['f', 2]], 'd': [['c', 0], ['a', 5], ['z', 10]], 'z': [['y', 10], ['x', 35], ['d', 5], ['a',
0]], 'x': [['z', 15]], 'f': [['c', 10], ['y', 2]], 'y': [['z', 1], ['f', 10]]} - остаточный путь

построение пути
```

Сложность алгоритма

E – множество ребер графа.

V – множество вершин графа.

F – величина максимальной пропускной способности графа.

По времени.

На каждом шаге мы ищем путь от стока к истоку, поиском с модификацией: каждый раз выполняется переход по дуге, имеющей максимальную остаточную пропускную способность.

Так как просматривать ребра нужно в определенном порядке, для этого все ребра вершины сортируются, на это приходится тратить |E| * log(|E|) операций. Помимо этого, алгоритм представляет собой обычный поиск в глубину, поэтому поиск нового дополняющего пути в сети происходит за O(|E| * log|E| * |V|).

В худшем случае, на каждом шаге мы будем находить дополняющий путь с пропускной способностью 1, тогда получим сложность по времени O(F* |E| * log|E| * |V|).

По памяти.

Сложность по памяти O(|E|).

Выводы.

В ходе лабораторной работы была изучена работа алгоритма поиска максимального потока в сети - метод Форда-Фалкерсона, способы хранения графа и остаточной сети и сложности по времени и памяти.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД.

```
print("введите количество ребер")
n=int(input())
graph={}
print("введите исток и сток (через enter)")
stream = [input(),input()]
edges=[]
print("введите ребра с величиной протекающего потока")
while n>0:                  #добавление ребер в словарь (с учетом обратных)
 x = input().split(' ')
 edges.append(x)
 if graph.get(x[0],1)==1:
  graph[x[0]]=[]
  graph[x[0]].append([x[1],int(x[2])])
 else:
  flag=0
  for item in graph.get(x[0]):#перезаписывание существующего ребра
     if item[0]==x[1]:
        item[1]=int(x[2])
        flag=1
  if flag==0:
     graph[x[0]].append([x[1],int(x[2])])
 if graph.get(x[1],1)==1:
  graph[x[1]]=[]
  graph[x[1]].append([x[0],0])#учет обратного ребра
 else:
  flag=0
  for item in graph.get(x[1]):
     if item[0]==x[0]:
        flag=1
  if flag==0:
     graph[x[1]].append([x[0],0])
 n-=1
ans=0
pathexist=1
while pathexist and stream[0]!=stream[1]:
   path=[]
   q=[]
   flags={}
   maxstream=999999
   for key in graph.keys(): #инициализация словаря посещенных ребер
      flags[key]=0
   curr=stream[0]
   for gkey, value in graph.items():
      value.sort(key=lambda x: (abs(ord(gkey)-ord(x[0])),abs(ord('a')-ord(x[0])))) #поиск пути по правилу
варианта]
   print()
   print(graph,'- остаточный путь')
   print()
   print('построение пути')
   print()
   while curr!=stream[1]:
      found=0
      for item in graph[curr]:          #обход ребер
        q.append(item)
        q.sort(key=lambda x: (abs(ord(curr)-ord(x[0])),abs(ord('a')-ord(x[0]))))
      print('рассматриваемые рёбра',q,'в вершине',curr)
      print()
```

```
            for item in q:
                if flags[item[0]]!=1 and item[1]>0:
                    flags[curr]=1
                    path.append(curr)
                    print('в путь добавлена вершина',curr,'с потоком',item[1])
                    print()
                    curr=item[0]
                    flags[curr]=1
                    found=1        #обновление максимальной величины потока текущего шага
                    break
            if curr!=stream[0] and found==0:       #шаг назад, если некуда идти
                flags[curr]=1
                curr=path[-1]
                path.pop()
            elif found==0:                    #если некуда идти в истоке
                pathexist=0
                maxstream=0
                break
        path.append(stream[1])
        path2=[]
        path2.append(curr)
        curr=stream[1]
        for i in reversed(range(len(path)-1)):
          for item in graph[path[i]]:
            if item[0]==curr and item[1]>0:
              if item[1]<maxstream:
                maxstream=item[1]
              curr = path[i]
              path2.append(curr)
              break
        ans+=maxstream
        if path!=list(reversed(path2)):
            path=list(reversed(path2))
            print('из пути удалены лишние вершины')
            print()
        if len(path)>1:
          for x in path:
            print(x,end='')
          print()
        for i in range(len(path)-1):    #цикл изменения протекающего потока
            for item in graph[path[i]]:
                if item[0]==path[i+1]:
                    item[1]-=maxstream
            for item in graph[path[i+1]]:
                if item[0]==path[i]:
                    item[1]+=maxstream
        print()
        print('в остаточном пути изменились рёбра из пути на',maxstream)
    print('путей больше нет')
    print()
    for item in edges:                #получение фактических величин протекающего потока в ребре с помощью
исходной и конечной сети
        for vertice in graph.get(item[0]):
            if vertice[0]==item[1]:
                item[2]=int(item[2])-vertice[1]
    print(ans)
    edges.sort(key=lambda x: (x[0],x[1]))#отсортированный вывод
    for item in edges:
        if item[2]>=0:
```

```
    print(item[0],item[1],item[2])
else:
    print(item[0],item[1],0)
```