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).

Входные данные:

N - количество ориентированных рёбер графа

v0 - исток

vn - сток

vi vj ωij - ребро графа

vi vj ωij - ребро графа

...

Выходные данные:

vi vj ωij - ребро графа с фактической величиной протекающего потока

vi vj ωij - ребро графа с фактической величиной протекающего потока

…

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).

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 в глубину (в индивидуализации по соответствующему правилу) 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	12 a b 6 a c 6 b d 6 c f 8 d e 2

c f 9 d e 3 d f 4 e c 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

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
print("вывод сети, для наглядности поиска пути")
while pathexist:
   print(graph)
   path=[]
   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
   while curr!=stream[1]:
      found=0
      for item in graph[curr]:            #обход ребер
         if flags[item[0]]!=1 and item[1]>0:
            flags[curr]=1
            path.append(curr)
            curr=item[0]
            found=1
            if item[1]<maxstream:
               maxstream=item[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])
      ans+=maxstream
      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
    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)
```