**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: Потоки в сети

|  |  |  |
| --- | --- | --- |
| Студент гр. 9304 |  | Ковалёв П.Д. |
| Преподаватель |  | Шевская Н.В. |

Санкт-Петербург

2021

**Цель работы.**

Изучить алгоритм Форда-Фалкерсона поиска максимального потока в графе. Реализовать программу на Python, которая для заданного графа вычисляет максимальный для него поток.

**Задание.**

Вариант 1. Поиск в ширину. Поочерёдная обработка вершин текущего фронта, перебор вершин в алфавитном порядке.

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  
*v*0​ - исток  
*vn*​ - сток  
– ребро графа

– ребро графа

…

**Выходные данные:**  
​ - величина максимального потока  
– ребро графа ​​ с фактической величиной протекающего потока  
– 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).

**Выполнение работы.**

Сначала в функции *main()* производится считывание графа в словарь. После считывания следует сортировка словаря, сначала по ключам, а после по значениям таким образом, чтобы все вершины-ключи стояли в алфавитном порядке, а все вершины, с которыми вершина-ключ смежная (они являются элементами списка смежности, который является значением словаря) так же стояли в алфавитном порядке. Это необходимо для того, чтобы правильно выводить граф и правильно его обрабатывать при обходе в ширину.

После этого в функции *maximum\_flow()* происходит вычисление максимального потока. Сначала ищется путь в графе, представленный словарём, при помощи функции *bfs()*, после чего происходит обработка пути: функция *construct\_way()* конструирует из словаря пути строку вершин из которых состоит путь. После функция *find\_minimal\_capacity()* проходится по данному пути и находит минимальную пропускную способность ребер этого пути.

Следующий шаг алгоритма происходит в *maximum\_flow()*: запускается цикл обхода пути по графу и по всему пути сначала инвертируются ребра, после у пройденных ребер изменяется максимальная пропускная способность и фактический поток через них. Главный цикл функции останавливается, когда не получается найти путь из истока к стоку. После чего функция возвращает список, первый элемент которого – значение максимального потока, второй элемент – список ребер с фактическим числом потока. После программа выводит результаты в консоль.

**Описание структуры данных, представляющей граф.**

Граф представлен словарём, ключи которого – строки-вершины, значения по ключам – список смежных ребер вида [[, , , show(bool)],[ , , , show(bool)],…[,, , show(bool)]]. – строки вершины, с которыми смежная вершина-ключ, , – максимальная пропускная способность, – фактическое число потока, show(bool) – булевое значение, которое сообщает, было ли это ребро в наборе ребер, которое задал пользователь или нет. От этого зависит, нужно ли ребро выводить или нет.

**Оценка сложности алгоритма.**

Сложность данного алгоритма , т.к. поиск в ширину работает за , а инициализация алгоритма происходит за .

**Тестирование.**

Были написаны юнит-тесты к каждой функции программы, за исключением *main()*. Тесты рассматривали случаи, когда пути из истока в сток нету, а также обычные случаи, чтобы убедиться в корректности работы алгоритма. К каждой функции были написаны по 5 тестов. Код приведен в файле tests.py. Результаты тестирования представлены на рисунке 1.

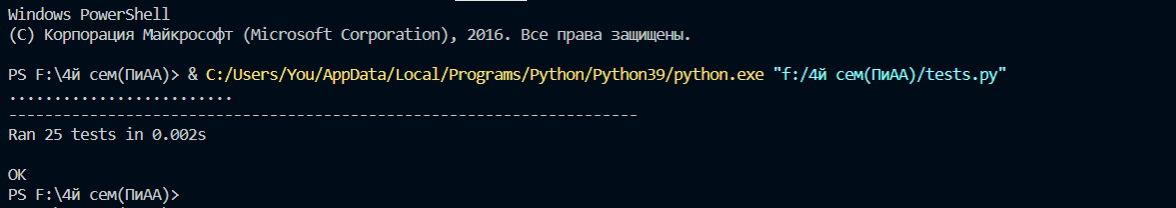


Рисунок 1 – Результаты юнит-тестирования

**Выводы.**

Изучили алгоритм Форда-Фалкерсона. Реализовали программу на Python, которая для заданного графа вычисляет максимальный для него поток.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

lr3.py:

import queue

def maximum\_flow(graph, startVertex, endVertex):

f = 0

while (True):

dway = bfs(graph, startVertex, endVertex)

if (dway is None):

break

way = construct\_way(dway, startVertex, endVertex)

cmin = find\_minimal\_capacity(dway, startVertex, endVertex)

f += cmin

for i in range(0, len(way) - 1):

for m in graph[way[i]]:

if m[0] is way[i + 1]:

if way[i+1] != endVertex:

for a in graph[way[i + 1]]:

if a[0] is way[i]:

a[1] += cmin

a[2] -= cmin

if a[3]:

a[3] = True

else:

a[3] = False

break

else:

graph.setdefault(way[i + 1], []).append([way[i], cmin, 0, False])

m[1] -= cmin

m[2] += cmin

res = [f]

for i in graph:

for m in graph[i]:

if(m[2] < 0):

res.append([i, m[0], 0, m[3]])

else:

res.append([i, m[0], m[2], m[3]])

return res

def find\_minimal\_capacity(dway, startVertex, endVertex):

tmpKey = endVertex

cmin = dway[tmpKey][1]

while dway[tmpKey][0] != startVertex:

tmpVal = dway[tmpKey][0]

if (cmin > dway[tmpVal][1]):

cmin = dway[tmpVal][1]

tmpKey = tmpVal

return cmin

def bfs(graph, startV, endV):

q = queue.Queue()

q.put(startV)

paths = {startV: []}

while (not q.empty()):

vertex = q.get()

if vertex in graph:

for v in graph[vertex]:

if (v[1] > 0 and v[0] not in paths):

paths[v[0]] = [vertex, v[1]]

if v[0] == endV:

return paths

q.put(v[0])

return None

def construct\_way(dict\_way, start, end):

way = ""

tmpKey = end

way += tmpKey

while dict\_way[tmpKey][0] != start:

tmpVal = dict\_way[tmpKey][0]

way += tmpVal

tmpKey = tmpVal

way += start

return way[::-1]

def sorter(diction):

key\_list = list(diction.keys())

key\_list.sort()

items\_list = []

for i in range(len(key\_list)):

items\_list.append(diction[key\_list[i]])

for i in items\_list:

i.sort(key=lambda x: x[0])

d = {}

for i in key\_list:

r = []

for m in items\_list:

if m == diction[i]:

r.append(m)

d[i] = r[0]

return d

def main():

n = int(input())

startVertex = input()

endVertex = input()

d = {}

m = []

while (n > 0):

s = input().split()

m.append(s)

d.setdefault(s[0], []).append([s[1], int(s[2]), 0, True])

n -= 1

d = sorter(d)

result = maximum\_flow(d, startVertex, endVertex)

flow = result[0]

l = []

for i in result:

if i != flow:

l.append(i)

l.sort(key=lambda x: x[0])

print(flow)

for i in range(0, len(l)):

if l[i][3]:

s = ""

for m in range(len(l[i])-1):

s += str(l[i][m]) + " "

num = len(s)

print(s[0:num - 1:1])

if \_\_name\_\_ == '\_\_main\_\_':

main()