**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

# **по лабораторной работе №3**

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

**Тема: «алгоритм Форда-Фалкерсона»**

**Вариант 4м**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7382 |  | Гиззатов А.С. |
| Преподаватель |  | Фирсов М.А. |

**Санкт-Петербург**

**2019**

**Задание.**

Разработайте программу, которая решает задачу построения пути в  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  
vivjωij - ребро графа  
vivjωij - ребро графа  
...

Выходные данные:  
Pmax - величина максимального потока  
vivjωij - ребро графа с фактической величиной протекающего потока  
vivjωij - 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).

**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

Вар. 4м. Матрица смежности. Поиск пути по правилу: каждый раз   
выполняется переход по ребру, соединяющему вершины, имена которых в   
алфавите ближе всего друг к другу.

**Пояснение задания.**

На вход программе подается граф. Программа находит максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро.

**Описание алгоритмов.**

Изначально величине потока присваивается значение 0 для всех. Затем величина потока увеличивается посредством поиска увеличивающего пути (путь от источника *s* к стоку *t*, вдоль которого можно послать больший поток). Процесс повторяется, пока можно найти увеличивающий путь.

**Описание структур**

Класс Graph.

Класс Compare.

Очередь с приоритетом std::priority\_queue.

Контейнеры std::set. std::vector, std::map.

Структура данных std::pair.

**Описание функций**

В программе использовалось класс Graph и его 11 методов:

* **void add\_way(char from, char to, double length)**

Добавление пути в матрицу.

* **int next\_way(char start, char end)**

нахождение потока через поданый путь

* **int get\_index(char el)**

получение индекса вершины в матрице смежности

* **void print\_matr()**

метод печати матрицы

* **int get\_cur\_stream(char start, char end)**

Метод нахождения максимально возможного потока через данное ребро

* **void find\_neighbour(queue &q,std::set<char> &visited, char el,char end)**

Метод для нахождения инцидентных вершин, которые не были посещены и добавления их в очередь

* **void recount\_stream(std::map<char,char> &way, int stream, char start, char end)**

Метод для пересчета потока проходящего через ребра

* **int min\_stream(std::map<char, char> &way,char start, char end)**

Метод нахождения минимального потока через данное ребро

* **void find\_stream(char start, char end)**

Метод для нахождения максимального потока в графе

* + **void print\_streams()**

Метод для печати реальных потов в изначальных ребрах

* + - * + **void print\_q(queue q)**

Метод для печати очереди с приоритетом.

Также был использован класс Compare с 1 перегруженной операцией:

* **bool operator()(const std::pair<char, char> &a,const std::pair<char, char> &b) const**

Операция для сравнения 2 вершин имена которых в   
алфавите ближе всего друг к другу.

**Тестирование.**

Рис.1 - Тестовый граф.

|  |  |
| --- | --- |
| Вводимые данные | Результат: |
| 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 |
| 7  a  f  a b 7  a c 6  b d 6  c f 9  d e 3  d f 4  e c 0 | 10  a b 4  a c 6  b d 4  c f 6  d e 0  d f 4  e c 0 |
| 7  a  f  a b 7  a c 6  b d 6  c f 5  d e 3  d f 4  e c 10 | 9  a b 4  a c 5  b d 4  c f 5  d e 0  d f 4  e c 0 |
| 7  a  f  a b 2  a c 9  b d 4  c f 3  d e 4  d f 5  e c 1 | 5  a b 2  a c 3  b d 2  c f 3  d e 0  d f 2  e c 0 |

**Вывод.**

В процессе выполнения лабораторной работы были изучены алгоритм Форда-Фалкерсона.

**Приложение 1.**

**Код программы.**

#include <iostream>

#include <vector>

#include <map>

#include <queue>

#include <set>

class Compare{ //Класс-компоратор для сортировки пар в очереди с приоритетом

public:

bool operator()(const std::pair<char, char> &a,const std::pair<char, char> &b) const

{

return abs(a.first-a.second)>abs(b.first-b.second);

}

};

typedef std::priority\_queue <std::pair<char, char>,

std::vector < std::pair<char, char>>,

Compare> queue; //объявление очереди с приоритетом

std::set<std::pair<char,char>> input; //объявление котейнера для ребер, поданных на вход

class Graph{

std::vector<std::vector<std::pair<int,int>>> matr; //объявление матрица смежности графа

std::vector<char> v;

int next\_way(char start, char end){ //нахождение потока через поданый путь

queue q;

std::set<char> visited; //список посещенных вершин

std::map<char,char> way; //текущий путь

char cur=start;

visited.insert(cur);

find\_neighbour(q,visited,start, end);

while(!q.empty()){

cur=q.top().second;

visited.insert(cur);

way[cur]=q.top().first;

q.pop();

if(cur==end){

int min=min\_stream(way,start, end); //находим поток

recount\_stream(way, min, start, end); //пересчитываем проходящий поток

return min;

}

find\_neighbour(q,visited,cur, end);

}

return 0;

}

public:

Graph(){}

void add\_way(char start, char end, int stream){ //добавления ребра в граф

int fr,t;

fr=get\_index(start);

if(fr==-1)

v.push\_back(start);

t=get\_index(end);

if(t==-1)

v.push\_back(end);

if(matr.size()<v.size()){

matr.resize(v.size());

for(int i=0;i<matr.size();i++){

matr[i].resize(matr.size());

}

}

matr[get\_index(start)][get\_index(end)]={stream,0};

}

int get\_index(char el){ //получение индекса вершины

for(int i=0;i<v.size();i++){

if(v[i]==el)

return i;

}

return -1;

}

void print\_matr(){ //печать матрицы

std::cout<<" ";

for(int i=0;i<v.size();i++)

std::cout<<v[i]<<" ";

std::cout<<std::endl;

for(int i=0;i<matr.size();i++){

std::cout<<v[i]<<" ";

for(int j=0;j<matr[i].size();j++)

std::cout<<matr[i][j].first<<"/"<<matr[i][j].second<<' ';

std::cout<<std::endl;

}

}

int get\_cur\_stream(char start, char end){ //максимально возможный поток через данное ребро

if(get\_index(start)==-1 || get\_index(end)==-1)

return -1;

return matr[get\_index(start)][get\_index(end)].first-matr[get\_index(start)][get\_index(end)].second;

}

void find\_neighbour(queue &q,std::set<char> &visited, char el,char end){ //нахождение инцидентных вершин, которые не были посещены и добавляет их в очередь

if(get\_cur\_stream(el,end)>0){

while(!q.empty()){

q.pop();

}

q.push({el,end});

return;

}

for(int k=0;k<v.size();k++){

if(get\_cur\_stream(el,v[k])<=0 || visited.find(v[k])!=visited.end())

continue;

q.push({el,v[k]});

visited.insert(v[k]);

}

}

void recount\_stream(std::map<char,char> &way, int stream, char start, char end){ //пересчет потока проходящего через ребра

char cur=end;

while(cur!=start){

matr[get\_index(way[cur])][get\_index(cur)].second+=stream;

matr[get\_index(cur)][get\_index(way[cur])].second-=stream;

cur=way[cur];

}

}

int min\_stream(std::map<char, char> &way,char start, char end){ //поиск минимального потока в пути

int min=get\_cur\_stream(way[end],end);

char cur=way[end];

std::string str;

str.push\_back(cur);

str.push\_back(end);

while(cur!=start){

if(get\_cur\_stream(way[cur],cur)<min)

min=get\_cur\_stream(way[cur],cur);

cur=way[cur];

str.insert(str.begin(),cur);

}

std::cout<<"Путь: "<<str<<std::endl

<<"Поток пути: "<<min<<std::endl << std::endl;

return min;

}

void find\_stream(char start, char end){ //поиск максимального потока в графе

int max\_gr\_stream=0,min\_way\_stream=0;

min\_way\_stream=next\_way(start, end);

while(min\_way\_stream>0){

max\_gr\_stream+=min\_way\_stream; //увеличиваем максимальный поток на величину потока пути

min\_way\_stream=next\_way(start, end);

}

std::cout<<max\_gr\_stream<<std::endl;

print\_streams();

}

void print\_streams(){ //печать реальных потоков в изначальных ребрах

for(auto k:input){

if(matr[get\_index(k.first)][get\_index(k.second)].second>=0)

std::cout<<k.first<<' '<<k.second<<' '<<matr[get\_index(k.first)][get\_index(k.second)].second<<std::endl;

else

std::cout<<k.first<<' '<<k.second<<' '<<0<<std::endl;

}

}

void print\_q(queue q){ //печать очереди с приоритетом

std::cout<<"Ребра в очереди:\n";

while(!q.empty()){

std::cout<<q.top().first<<"->"<<q.top().second<<std::endl;

q.pop();

}

std::cout<<std::endl;

}

};

int main(){

Graph a;

int n,stream;

char start,end,first,second;

std::cin>>n>>start>>end;

for(int i=0;i<n;i++){

std::cin>>first>>second>>stream;

input.insert(std::pair<char,char>(first,second));

a.add\_way(first,second,stream);

}

a.find\_stream(start,end);

return 0;

}