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. 8304 _____ Порывай П.А.

Преподаватель _____ Размочаева Н.В

Санкт-Петербург

2020

Цель работы.

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

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

...

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 2

Поиск в ширину. Обработка совокупности вершин текущего фронта как единого целого, ду́ги выбираются в порядке уменьшения остаточных пропускных способностей. При равенстве остаточных пропускных способностей выбирается та дуга, из начала которой выходит меньше дуг, при этом учитываются только ду́ги с положительными остаточными пропускными способностями, не ведущие в вершины текущего или прошлых фронтов.

Описание алгоритма

В начале алгоритм Форда-Фалкерсона заполняет значения функции f для всех ребер графа. Далее пока у нас в “остаточной сети” (изначально это сам граф) есть дополняющий путь то по Теореме о максимальном разрезе поток еще не полон и для каждого ребра дополняющего пути увеличиваем поток и перестраиваем “остаточную сеть” Сложность алгоритма O(E|f*|)

Описание основных структур данных и функций

struct vertex_len

Имя и длина до вершины в списке смежных вершин

map<char, char*> colour

Ассоциативный массив для определения цвета вершины

map<char, char> Pi

Ассоциативный массив для определения предшественника по ключю

set<char> Vertexes

Множество – вершины графа

queue<char> Q

Данная очередь используется в BFS

bool print_path(char* path, int* i_path, char s, char v)

Функция возвращает true, если нашли путь из s в v

void BFS(vector <vector <char> > Adj, char start);

Основной алгоритм, описанный выше

int c(char path[], vector <vector <vertex_len> > Adj)

Если путь из начальной вершины в конечную найден, то данный алгоритм находит минимальное ребро в этои пути

void Ford_Fulkerson(vector <vector <vertex_len> > Adj1, vector <vector<char>> Adj2, char s, char t, char** ak, int MaxN)

Функция, включающая основной алгоритм описанный выше

bool operator < (const vertex_len& dot1, const vertex_len& dot2)

Перегрузка оператора для функции qsort, которая позволяет сортировать по убыванию пропускной способности

Тестирование

В Таблице 1 не указаны промежуточные результаты, которые вывожу в программе.

Таблица 1 в формате(ввод/вывод)

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
4 a d a b 2 b d 1 a c 2 c d 11	3 a b 1 b d 1 a c 2 c d 2

1 a d a d 10	10 a d
5 a l a d 3 d m 10 m l 11 d k 7 k l 9	3 a d 3 d m 3 m l 3 d k 0 k l 0
8 a c a b 12 b k 4 a e 3 d e 4 c d 5 b c 6 k p 2 p c 12	8 a b 8 b k 2 a e 0 d e 0 c d 0 b c 6 k p 2 p c 2

Выводы.

В ходе выполнения работы были получены навыки работы с основными алгоритмами ”поиск в ширину”, алгоритм ”Форда-Фалкерсона”

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

```
#define _CRT_SECURE_NO_WARNINGS
#include<vector>
#include<set>
#include <iostream>
#include <iterator>
#include<map>
#include<queue>
#include<string>
#include<cstring>
#include<algorithm>
using namespace std;

struct vertex_len {
	char name;
	int len;

};
map<char, char*> colour;
map<char, int> d;//количество вершин до начальной для вершины по ключю
map<char, char>  Pi;//Предшественник для вершины по ключю



set<char> Vertexes;


queue<char> Q;
bool print_path(char* path, int* i_path, char s, char v);
void BFS(vector <vector <char> > Adj, char start);
int c(char path[], vector <vector <vertex_len> > Adj);




map < std::string, int > f;

char to_delete1;
char to_delete2;
int j_delete;
void Ford_Fulkerson(vector <vector <vertex_len> > Adj1, vector <vector<char>> Adj2, char s, char t, char** ak, int MaxN);


bool operator < (const vertex_len& dot1, const vertex_len& dot2)
{
	return dot1.len > dot2.len;
}
int main()
{
```

```
	setlocale(LC_ALL, "Russian");
	int MaxN = 0;
	char u, v;
	char start, end;
	std::cin >> MaxN;
	std::cin >> start;
	std::cin >> end;

	vector <vector <char> > Adj(100);//список смежных вершин
	vector <vector<vertex_len>> Adj1(100);//список смежных вершин с
расстояниями




	int MaxI = MaxN;

	int len;

	vertex_len v_l;

	char** aka = new char* [100];
	for (int i = 0; i < MaxN; i++)
		aka[i] = new char[3];

	int i = 0;
	int j;
	while (MaxN--) {

		cin >> u >> v >> len;
		j = 0;
		aka[i][j] = u;
		j++;
		aka[i][j] = v;
		j++;
		aka[i][j] = '\0';
		i++;
		if (Vertexes.find(u) == Vertexes.end())
			Vertexes.insert(u);

		if (Vertexes.find(v) == Vertexes.end())
			Vertexes.insert(v);

		v_l = { v,len };
		Adj1[u - 97].push_back(v_l);

	}

	std::cout << "\nДо сортировки смежных вершин\n";
	for (int i = 0; i < 100; i++) {

		if (Adj1[i].size() > 0) {
			char a = i + 97;
			std::cout << "\n" << a << ": ";
			for (int j = 0; j < Adj1[i].size(); j++)
```

```
                    std::cout   <<   Adj1[i][j].name   <<   "   "   <<
Adj1[i][j].len << " ";
          }
     }

     std::cout << "\nПосле сортировки смежных вершин\n";

     for (int i = 0; i < 100; i++) {

          if (Adj1[i].size() > 0) {
               sort(Adj1[i].begin(), Adj1[i].end());
               char a = i + 97;
               std::cout << "\n" << a << ": ";
               for (int j = 0; j < Adj1[i].size(); j++) {
                    std::cout   <<   Adj1[i][j].name   <<   "   "   <<
Adj1[i][j].len << " ";
                    Adj[i].push_back(Adj1[i][j].name);
               }
          }
     }
     std::cout << "\n";
     Ford_Fulkerson(Adj1, Adj, start, end, aka, MaxI);

}


void  Ford_Fulkerson(vector  <vector  <vertex_len>  >  Adj1,  vector
<vector<char>> Adj2, char s, char t, char** ak, int size1) {

     char** aka = new char* [100];
     for (int i = 0; i < 50; i++)
          aka[i] = new char[3];

     int y = 0;
     for (int i = 0; i < Adj1.size(); i++) {



          for (int j = 0; j < Adj1[i].size(); j++) {

               char u = i + 97;
               char v = Adj1[i][j].name;
               char qq[2] = { u,v };

               char* ver_s1 = new char[3];
               ver_s1[0] = u;
               ver_s1[1] = v;
               ver_s1[2] = '\0';

               std::string str = string(ver_s1);



               f[str] = 0;
               char* ver_s2 = new char[3];

               ver_s2[0] = v;
```

```
                ver_s2[1] = u;
                ver_s2[2] = '\0';
                char qq1[2] = { u,v };
                std::string str11 = string(ver_s2);

                f[str11] = 0;
           }

     }

     BFS(Adj2, s);

     char* path = new char[100];
     int i_path = 0;

     bool yes = print_path(path, &i_path, s, t);
     path[i_path] = '\0';

     while (yes) {

          int min = c(path, Adj1);//после определения минимального
пути надо перестроить граф
          for (int i = 0; i < strlen(path) - 1; i++)
{//перестраиваем граф


                char predok = path[i];
                char potomok = path[i + 1];

                int size = Adj1[predok - 97].size();
                for (int j = 0; j < size; j++) {
                     if (Adj1[predok - 97][j].name == potomok) {

                          Adj1[predok - 97][j].len -= min;

                          if (Adj1[predok - 97][j].len == 0)
{//удаляем ребро из остаточной сети
                               Adj1[predok - 97].erase(Adj1[predok
- 97].begin() + j);
                               Adj2[predok - 97].erase(Adj2[predok
- 97].begin() + j);
                               size--;
                          }

                          bool dobavili_obratny = false;

                          int ind_obratn;

                          for (int k = 0; k < Adj1[potomok -
97].size(); k++) {
                               if (Adj1[potomok - 97][k].name ==
predok) {
                                    dobavili_obratny = true;
                                    ind_obratn = k;
                                    break;
                               }
```

```
                        }

                        if (dobavili_obratny == false) {

                              vertex_len node = { predok,min };
                              Adj1[potomok - 97].push_back(node);
                              Adj2[potomok                          -
97].push_back(predok);
                        }
                        else if (dobavili_obratny == true) {
                              Adj1[potomok   -   97][ind_obratn].len
+= min;

                        }
                  }

            }

      }


      for (int i = 0; i < strlen(path) - 1; i++) {
            char predok = path[i];
            char potomok = path[i + 1];
            char* f_tuda = new char[3];
            f_tuda[0] = predok;
            f_tuda[1] = potomok;
            f_tuda[2] = '\0';

            char qq[2];
            qq[0] = predok;
            qq[1] = potomok;

            std::string strq = string(f_tuda);


            f[strq] = f[strq] + min;

            y++;

            char* f_obratno = new char[3];
            aka[y] = f_obratno;
            y++;
            f_obratno[0] = potomok;//
            f_obratno[1] = predok;
            f_obratno[2] = '\0';
            char qq2[2];
            qq2[0] = potomok;
            qq2[1] = predok;
            std::string str = string(f_obratno);
            f[str] = -f[strq];

      }
```

```
            BFS(Adj2, s);

            path = new char[100];
            i_path = 0;


            yes = print_path(path, &i_path, s, t);
            path[i_path] = '\0';

        }
        int it = 0;


        for (int m = 0; m < size1; m++) {
            string qq = string(ak[m]);
            if (qq[1] == t)
                it += f[qq];




        }
        std::cout << it << "\n";

        for (int m = 0; m < size1; m++) {
            string qq = string(ak[m]);



            std::cout << qq[0] << " " << qq[1] << " " << f[qq] << "\n";

        }

}//Как делать остаточную сеть?

int c(char* path, vector <vector <vertex_len> > Adj) {//нахождение минимального веса в пути
        //std::cout << "AA";
        int min = 99;

        for (int i = 0; i < strlen(path) - 1; i++) {

            char predok = path[i];
            char potomok = path[i + 1];
```

```
			//std::cout << predok;
			for (int j = 0; j < Adj[predok - 97].size(); j++) {
				//std::cout << "ye";
				if (Adj[predok - 97][j].name == potomok) {
					if (min > Adj[predok - 97][j].len) {//////////
						to_delete1 = predok;
						to_delete2 = Adj[predok - 97][j].name;

						j_delete = j;
						//здесь можно сохранять предка и потомка с минимальным значением
						min = Adj[predok - 97][j].len;


					}
				}

			}



		}

		return min;

}

void BFS(vector <vector <char> > Adj, char start) {//находит не для всех входных данных

		set<char>::iterator it;
		for (it = Vertexes.begin(); it != Vertexes.end(); it++) {

			char* col1 = new char[100];
			strcpy(col1, "белый");
			colour[*it] = col1;

			d[*it] = 99999;//значит пути нет(до начала)
			Pi[*it] = 0;//значит не записана предыдущая вершина


		}

		char* col2 = new char[100];
```

```
        strcpy(col2, "серый");
        colour[start] = col2;
        d[start] = 0;
        Pi[start] = 0;

        Q.push(start);

        while (Q.empty() == false) {

            char u = Q.front();


            for (int i = 0; i < Adj[u - 97].size(); i++) {

                if (strcmp(colour[Adj[u - 97][i]], "белый") == 0) {

                    char* col3 = new char[100];

                    strcpy(col3, "серый");


                    colour[Adj[u - 97][i]] = col3;
                    d[Adj[u - 97][i]] = d[u] + 1;
                    Pi[Adj[u - 97][i]] = u;
                    Q.push(Adj[u - 97][i]);
                }


            }

            Q.pop();

            char* col4 = new char[100];
            strcpy(col4, "черный");

            colour[u] = col4;


        }


}
```

```
bool print_path(char* path, int* i_path, char s, char v) {

	if (v == s) {

		path[*i_path] = v;
		*i_path = *i_path + 1;
		return true;

	}
	else if (Pi[v] == 0) {
		return false;
	}
	else {
		print_path(path, i_path, s, Pi[v]);
		path[*i_path] = v;
		*i_path = *i_path + 1;

	}
}
```