**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по курсовой работе**

**по дисциплине «Алгоритмы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9301 |  | Русанова К.В. |
| Преподаватель |  | Тутуева А. В. |

Санкт-Петербург

2021

### Постановка задачи

Входные данные: в текстовом файле в первой строке записано количество вершин графа и количество рёбер. В последующих строках в формате V1, V1, P, где V1, V2 направленная дуга транспортной сети, а P – ее пропускная способность. Исток всегда обозначен как S, сток – как T

Пример файла для сети с изображения выше:

5 8

S O 3

S P 3

O Q 3

O P 2

P R 2

Q R 4

Q T 2

R T 3

Выводимые данные: Найти максимальный поток в сети используя алгоритм Форда — Фалкерсона.

**Описание реализуемых алгоритмов и используемых структур**

Хранение графа осуществляется с помощью матрицы смежности. Обходом в глубину происходит поиск пути, после этого ребра графа меняются в зависимости от минимального потока на этом пути.

### Оценка временной сложности

Табл.

|  |  |
| --- | --- |
| **Название метода** | **Сложность** |
| readList() | O(E) |
| MaxFlow () | O(E\*V2) |

### Реализованные unit-тесты

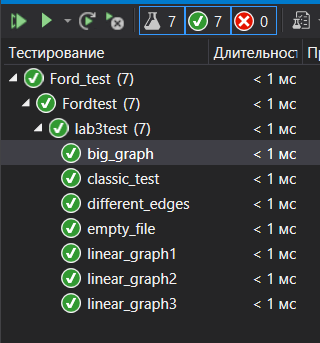


Рис. 1 — Пройденные тесты

* empty\_file — Проверка ошибки при пустом файле
* classic\_test — Проверка теста с википедии
* different\_edges — Как и предыдущий, но рёбра графа имеют разные веса
* linear\_graph1 — Три вершины соединённые двумя рёбрами, имеющими равные веса
* linear\_graph2 — Как и предыдущий но первое ребро имеет больший вес
* linear\_graph3 — Как и предыдущий но первое ребро имеет меньший вес
* big\_graph — Проверка на большом графе из ИДЗ

### Пример работы

На рисунке 2 показан пример выводимых данных и итогового файлов.

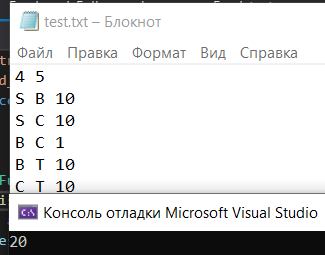


Рис. 2 — Демонстрация работы

Лист. 1 — Ford\_and\_Fulkerson\_lab.cpp

#include <iostream>

#include "Ford\_and\_Fulkerson.h"

using namespace std;

int main()

{

Ford\_and\_Fulkerson alg;

alg.readList("test.txt");

int res = alg.MaxFlow();

cout << res;

}

### Листинг

Лист. 1 — Ford\_and\_Fulkerson.h

#pragma once

#include <iostream>

using namespace std;

class Ford\_and\_Fulkerson

{

private:

int Vertices;

int\*\* graph\_matrix;

int from, to;

int resultStream;

char\* VerticesName;

int dfs(int, int, bool\*);

public:

void readList(string);

int MaxFlow();

};

Лист. 3 — Ford\_and\_Fulkerson.cpp

#include "Ford\_and\_Fulkerson.h"

#include <fstream>

void Ford\_and\_Fulkerson::readList(string fileName) {

fstream file;

char firstName, secondName;

int weight;

int edge;

int CountVerticesName = 0;

file.open(fileName, ios::in);

/\*read count vertices and edge\*/

file >> Vertices >> edge;

if (file.eof() || Vertices == 0) throw out\_of\_range("Graph is empty");

/\*memory allocation\*/

VerticesName = new char[Vertices];

int\*\* arr\_check;

arr\_check = (int\*\*)malloc(sizeof(int\*) \* Vertices);

if (!arr\_check) {

throw std::out\_of\_range("Allocation error");

return;

}

else {

graph\_matrix = arr\_check;

}

for (size\_t i = 0; i < Vertices; i++)

{

arr\_check[i] = (int\*)malloc(Vertices \* sizeof(int));

if (!arr\_check[i]) { throw std::out\_of\_range("Allocation error"); return; }

else graph\_matrix[i] = arr\_check[i];

for (size\_t j = 0; j < Vertices; j++) {

graph\_matrix[i][j] = 0;

}

}

/\*read file\*/

int found1, found2;

for (size\_t i = 0; i < edge; i++)

{

file >> firstName >> secondName >> weight;

found1 = -1;

for (size\_t j = 0; j < CountVerticesName; j++)

if (VerticesName[j] == firstName)

found1 = j;

if (found1 == -1)

{

VerticesName[CountVerticesName] = firstName;

found1 = CountVerticesName;

CountVerticesName++;

}

found2 = -1;

for (size\_t j = 0; j < CountVerticesName; j++)

if (VerticesName[j] == secondName)

found2 = j;

if (found2 == -1)

{

VerticesName[CountVerticesName] = secondName;

found2 = CountVerticesName;

CountVerticesName++;

}

graph\_matrix[found1][found2] = weight; //save in matrix

}

for (size\_t i = 0; i < Vertices; i++) //search for source and sink

{

if (VerticesName[i] == 'S')

from = i;

else if (VerticesName[i] = 'T')

to = i;

}

}

int Ford\_and\_Fulkerson::MaxFlow() {

resultStream = 0;

int toAdd = 0;

int bufVe, bufPrice;

bool\* visited = new bool[Vertices];

do

{

for (size\_t i = 0; i < Vertices; i++)

visited[i] = false;

toAdd = dfs(from, INT\_MAX, visited);

resultStream += toAdd;

} while (toAdd>0); //while there is something to add

return resultStream;

}

int min(int a, int b) {

if (a < b)

return a;

else

return b;

}

int Ford\_and\_Fulkerson::dfs(int u, int Cmin,bool\* visited) {

if (u == to)

return Cmin;

visited[u] = true;

int delta;

for (size\_t v = 0; v < Vertices; v++)

{

if (!visited[v] && (graph\_matrix[u][v] > 0))

{

delta = dfs(v, min(Cmin, graph\_matrix[u][v]), visited);

if (delta > 0) //change the network

{

graph\_matrix[u][v] -= delta;

graph\_matrix[v][u] += delta;

return delta;

}

}

}

return 0;

}