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»
по теме: “Потоки в сетях”
Вариант 1

Студент гр. 9302 _____ Точилин. А.Е.

Преподаватель _____ Тутуева А. В.

Санкт-Петербург
2021

1. Постановка задачи

Найти максимальный поток в сети, используя алгоритм Форда-Фалкерсона.

2. Теоретическая часть

Сетью называется орграф без циклов с помеченными вершинами и дугами. Числа, которыми помечаются дуги сети, называются пропускными способностями дуг.

Примеры вершин сети: перекрёстки дорог, телефонные узлы, железнодорожные узлы, аэропорты, склады и т.д. Примеры дуг сети: дороги, трубы, телефонные и железнодорожные линии и т.д.

Сеть, у которой существует ровно один исток и один сток, называется транспортной сетью.

В теории оптимизации и теории графов, задача о максимальном потоке заключается в нахождении такого потока по транспортной сети. Сумма потоков из истока, сумма потоков в сток максимальна (что одно и то же).

3. 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.

Название	**Описание**
bool isContain(char* arr, char ch)	определяет наличие символа ch в массиве символов arr
struct set	структура для предварительной обработки, которая хранит уникальные имена вершин и их количество
void Ford::inputData(string fileName)	считывает данные из входного файла и составляет матрицу смежности
int Ford::MaxFlow()	поиск максимального потока
int Ford::dfs(int u, int Cmin, bool* visited)	обход в глубину

int min(int a, int b)	возвращает минимальный элемент из двух подаваемых

4. Оценка временной сложности

Название метода	**Оценка временной сложности**
inputData()	O(n)
MaxFlow()	$O(n^2)$

5. Описание реализованных unit-тестов

Название	**Назначение**
Test1	Проверка макс. потока с разными данными
Test2	Проверка макс. потока с разными данными
Test3	Проверка макс. потока с разными данными
Test4	Проверка макс. потока с разными данными

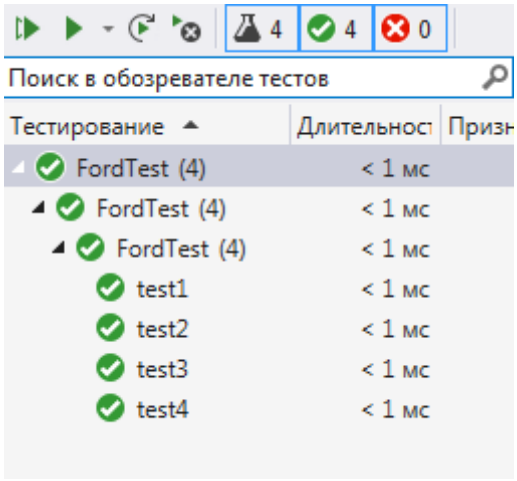

Рис. 1. Реализованные Unit-тесты.

6. Пример работы

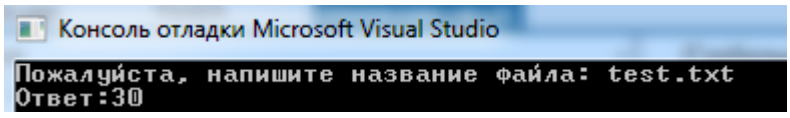

Рис. 2. Пример работы программы.

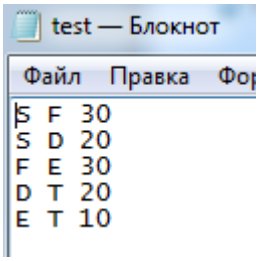

Рис. 3. Входные данные.

7. Листинг

Course.cpp

```
#include <iostream>
#include "Ford.h"
using namespace std;

int main()
{
    setlocale(LC_ALL, "RUS");
    cout << "Пожалуйста, напишите название файла: ";
    string filename;
    cin >> filename;
    Ford test;
    test.inputData(filename);
    int res = test.MaxFlow();
    cout << "Ответ:" << res;
    }
```

Ford.h

```
#ifndef FORD_H
#define FORD_H
#define infinity INT_MAX

#include <fstream>
#include <iostream>
#include <string>

using namespace std;

int min(int a, int b) {
	if (a < b)
		return a;
	else
		return b;
}

bool isContain(char* arr, char ch) {
	for (int i = 0; i < 26; i++) {
		if (arr[i] == ch)
			return true;
	}
	return false;
}

struct set {
	char* arr = new char[26];
	int size = 0;
};
```

```
class Ford
{
public:
	int Tops;
	int** graphMat;
	int from, to;
	int _stream;
	char* TopsName;

	int dfs(int u, int Cmin, bool* visited) {
		if (u == to)
			return Cmin;
		visited[u] = true;
		int delta;
		for (int v = 0; v < Tops; v++)
		{
			if (!visited[v] && (graphMat[u][v] > 0))
			{
				delta = dfs(v, min(Cmin, graphMat[u][v]), visited);
				if (delta > 0)
				{
					graphMat[u][v] -= delta;
					graphMat[v][u] += delta;
					return delta;
				}
			}
		}
		return 0;
	}


	int MaxFlow() {
		_stream = 0;
		int toAdd = 0;
		int buf1, buf2;
		bool* visited = new bool[Tops];
		do
		{
			for (int i = 0; i < Tops; i++)
				visited[i] = false;
			toAdd = dfs(from, infinity, visited);
			_stream += toAdd;
		} while (toAdd > 0);
		return _stream;
	};

	void inputData(string fileName) {
		string temp;
		fstream file;
		char firstNode, secondNode;
		int weight;
		int edge = 0;
		int CountTopName = 0;
		file.open(fileName);
		set _set;

		while (!file.eof()) {
			temp = "";
			getline(file, temp);
			if (!isContain(_set.arr, temp[0])) {
				_set.arr[_set.size] = temp[0];
				_set.size++;
```

```
		}
		if (!isContain(_set.arr, temp[2])) {
			_set.arr[_set.size] = temp[2];
			_set.size++;
		}
		edge++;
	}
	Tops = _set.size;

	file.close();
	file.open(fileName);

	TopsName = new char[Tops];
	int** arr_check = new int* [Tops];

	graphMat = arr_check;

	for (int i = 0; i < Tops; i++)
	{
		arr_check[i] = new int[Tops];
		graphMat[i] = arr_check[i];
		for (int j = 0; j < Tops; j++) {
			graphMat[i][j] = 0;
		}

	}

	int found1, found2;
	for (int i = 0; i < edge; i++)
	{
		file >> firstNode >> secondNode >> weight;

		found1 = -1;
		for (int j = 0; j < CountTopName; j++)
			if (TopsName[j] == firstNode)
				found1 = j;
		if (found1 == -1)
		{
			TopsName[CountTopName] = firstNode;
			found1 = CountTopName;
			CountTopName++;
		}

		found2 = -1;
		for (int j = 0; j < CountTopName; j++)
			if (TopsName[j] == secondNode)
				found2 = j;
		if (found2 == -1)
		{
			TopsName[CountTopName] = secondNode;
			found2 = CountTopName;
			CountTopName++;
		}
		graphMat[found1][found2] = weight;
	}
	file.close();
	for (int i = 0; i < Tops; i++)
	{
		if (TopsName[i] == 'A')
			from = i;
		else if (TopsName[i] == 'Z')
			to = i;
	}
```

```
        };
};

    #endif
```

FordTest.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "..\course\Ford.h"
#include "..\course\course.cpp"
#define TEST_CASE_DIRECTORY GetDirectoryName(__FILE__)

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace FordTest
{
	TEST_CLASS(FordTest)
	{
		string GetDirectoryName(string path) {
			const size_t last_slash_idx = path.rfind('\\');
			if (std::string::npos != last_slash_idx)
			{
				return path.substr(0, last_slash_idx + 1);
			}
			return "";
		}
	public:

		TEST_METHOD(test1)
		{
			string fileName = std::string(TEST_CASE_DIRECTORY) + "test1.txt";
			Ford test;
			test.inputData(fileName);
			Assert::AreEqual(test.MaxFlow(), 50);
		}
		TEST_METHOD(test2)
		{
			string fileName = std::string(TEST_CASE_DIRECTORY) + "test2.txt";
			Ford test;
			test.inputData(fileName);
			Assert::AreEqual(test.MaxFlow(), 20);
		}
		TEST_METHOD(test3)
		{
			string fileName = std::string(TEST_CASE_DIRECTORY) + "test3.txt";
			Ford test;
			test.inputData(fileName);
			Assert::AreEqual(test.MaxFlow(), 40);
		}
		TEST_METHOD(test4)
		{
			string fileName = std::string(TEST_CASE_DIRECTORY) + "test4.txt";
			Ford test;
			test.inputData(fileName);
			Assert::AreEqual(test.MaxFlow(), 50);
		}
	};
}
```