**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»**

**Тема: Реализация алгоритма Форда-Фалкерсона**

**Вариант 1**

Студентка гр. 8302 \_\_\_\_\_\_\_\_\_\_\_\_\_ Колмакова С.П.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ Тутуева А.В.

Санкт-Петербург

2020

Цель работы

Реализовать алгоритм Форда-Фалкерсона для поиска максимального потока в сети.

Описание алгоритма решения

1.Необходимо проверить файл на корректность входных данных. Далее считать данные из файла, применить к ним реализованный алгоритм Форда-Фалкерсона.

Входные данные:

* Файл формата:

A/B/C…/Z A/B/C…/Z NUM

. . .

A/B/C…/Z A/B/C…/Z NUM

NUM – целое, неотрицательное число

Выходные данные:

* Сообщение Max flow is MAX\_FLOW

MAX\_FLOW – результат работы нахождения максимального потока с помощью алгоритма Форда-Фалкерсона.

2. Я создала пользовательский класс Flows. Класс представляет сеть в виде графа и содержит метод – алгоритм Форда-Фалкерсона. Класс содержит в себе поля:

* numOfVertex – число ребер
* numOfEdge – число верши
* sourceVertex – источник сети
* destinationVertex – сток сети
* capacity – одномерный динамический массив пропускных способностей
* onEnd - одномерный динамический массив
* nextEdge - одномерный динамический массив вершин идущих после вершины по номеру которой обратились
* firstEdge - одномерный динамический массив начал рёбер
* visited – одномерный динамический массив для проверки посещённых

вершин

* INF – бесконечность – реализована значением INT\_MAX величина необходимая для корректноой работы алгоритма

Методы класса:

* Конструктор – Вызывает метод CheckInput, инициализирует поля класса, использует метод addEdge. Сложность инициализации O(V), V – количество вершин. Ввиду того, что количество полей класса равно const временная сложность инициализации определяется как O(V). Сложность методов CheckInput и addEdge описана ниже.
* Деструктор – удаляет динамические поля класса, работает за O(const) ввиду использования стандартной функции языка delete.
* *max\_flow* – поиск максимального потока в сети. Работает за O(MAX\_FLOW\*E), обусловлена сходимостью алгоритма только для целочисленных потоков. В случае с нецелочисленными данными сложность алгоритма является бесконечной, не сходясь к правильному решению. В нашем случае проверка ввода не пропустит нецелочисленные данные на вход алгоритма. Метод использует метод *findFlow*.
* *addEdge* – метод, используемый конструктором – присваивает полям класса значения, работает за O(const) ввиду обращения к конкретным индексам конкретных полей.
* *findFlow* –Находит увеличивающий путь, пропускает через него максимальный поток, а также работает с остаточной сетью в соответсвии с алгоритмом. Работает за О(E), E – количество ребер графа.
* *CheckInput* – проверяет корректность входных данных на соответствие шаблону. Кидает исключение в случае ошибки. Также подсчитывает количество вершин в графе. Временная сложность проверки O(n), n – количество строк в файле. Временная сложность подсчета количества вершин V – количество вершин в графе O(V\*log(V)) – обусловлена использованием ассоциативного массива, реализованного на основе красно-черного дерева для хранения считаных данных (O(log(V)) – сложность вставки и поиска в красно- черном дереве). Таким образом, в худшем случае, алгоритму придется применить операции поиска по 2 раза и вставки по 1 разу, что даст сложность (O(V\*(2log(V)+log(V))) = O(V\*log(V)). Альтернативная временная сложность – O(n\*log(n)), n – количество строк во входном файле. Следует отметить, что худший случай для этого алгоритма – граф, вырожденный в “ломанную” линию.

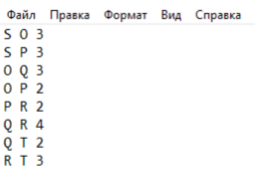
3. Использование ассоциативного массива для проверки ввода обусловлено вычислительной сложностью алгоритма. Ввиду того, что ассоциативный массив реализован на основе красно-черного дерева операции вставки и поиска имеют логарифмическую вычислительную сложность, что ускоряет быстродействие программы на больших объемах данных. Для сравнения: операция поиска в массиве/списке имеет сложность O(n), а в красно-черном дереве O(log n).

4. На вход программе поступает файл, после чего он проверяется на корректность входных данных. Далее запускается алгоритм Форда-Фалкерсона.

Пример работы программы

**Пример 1**

Входные данные:

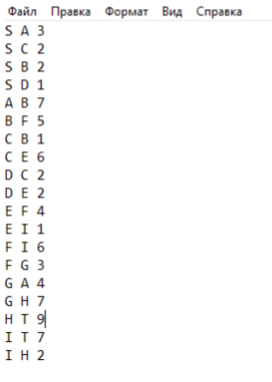


Выходные данные:



**Пример 2**

Входные данные:

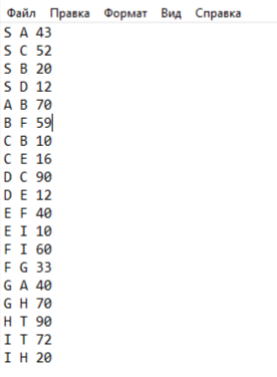


Выходные данные:



**Пример 3**

Входные данные:

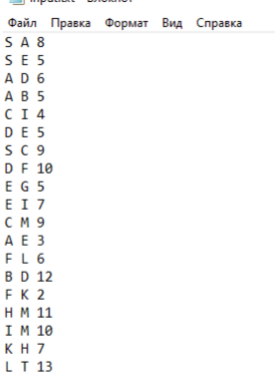


Выходные данные:



**Пример 4**

Входные данные:

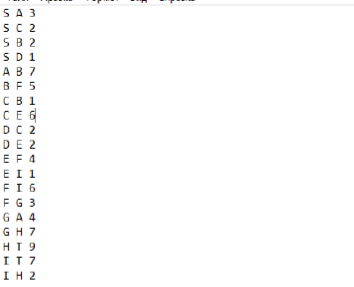
****

Выходные данные:



**Пример 5**

Входные данные:



Выходные данные:



Описание реализованных unit-тестов

Юнит тесты проверяют как работу кода на корректных данных, так и обработку исключений.

#include "pch.h"

#include "CppUnitTest.h"

#include"../Ford\_Fulkerson/Flows.h"

#include<fstream>

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestForFordFulkerson

{

TEST\_CLASS(UnitTestForFordFulkerson)

{

public:

TEST\_METHOD(TestErrorFirstInputCharacter) {

try {

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input2.txt");

Flows Flows(input);

}

catch (exception& ex) {

Assert::AreEqual(ex.what(), "Error entering the first character in the string or missing a space after it. Check the correctness of the input in the file and correct these errors in the line under the number: 2");

}

}

TEST\_METHOD(TestCorrectInput)

{

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input1.txt");

Flows Flows(input);

Assert::AreEqual(Flows.max\_flow(), 5);

}

TEST\_METHOD(TestErrorSecondInputCharacter) {

try {

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input3.txt");

Flows Flows(input);

}

catch (std::exception& ex) {

Assert::AreEqual(ex.what(), "Error entering the second character in the string or missing a space after it. Check the correctness of the input in the file and correct these errors in the line under the number: 2");

}

}

TEST\_METHOD(TestErrorNumberInput) {

try {

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input4.txt");

Flows Flows(input);

}

catch (exception& ex) {

Assert::AreEqual(ex.what(), "Error entering the third character (bandwidth) in the string or the presence of a space after it.Please note that the bandwidth cannot be negative. Check that you entered the file correctly and correct these errors in the line number: 2");

}

}

TEST\_METHOD(TestEmptyInput) {

try {

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input5.txt");

Flows Flows(input);

}

catch (exception& ex) {

Assert::AreEqual(ex.what(), "A data-entry error. Check the correctness of the input in the file and correct these errors in the line under the number: 2");

}

}

TEST\_METHOD(TestCorrectInput\_6vert)

{

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input6.txt");

Flows Flows(input);

Assert::AreEqual(Flows.max\_flow(), 25);

}

TEST\_METHOD(TestCorrectInput\_2vert)

{

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input7.txt");

Flows Flows(input);

Assert::AreEqual(Flows.max\_flow(), 20);

}

TEST\_METHOD(TestErrorVertexPathToItself) {

try

{

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input8.txt");

Flows Flows(input);

}

catch (exception& ex) {

Assert::AreEqual(ex.what(), "The path from the vertex to itself is impossible in the string under the number: 2");

}

}

TEST\_METHOD(TestCorrect\_20vert)

{

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input9.txt");

Flows Flows(input);

Assert::AreEqual(Flows.max\_flow(), 19);

}

TEST\_METHOD(TestErrorNoSource) {

try

{

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input10.txt");

Flows Flows(input);

}

catch (exception& ex) {

Assert::AreEqual(ex.what(), "Source is missing");

}

}

TEST\_METHOD(TestErrorNoSink) {

try

{

ifstream input("C:\\Users\\sveta\\Desktop\\kr1\\UnitTestForFordFulkerson\\Input11.txt");

Flows Flows(input);

}

catch (exception& ex) {

Assert::AreEqual(ex.what(), "Sink is missing");

}

}

TEST\_METHOD(TestMethod\_MinFunctionMin) {

int a = 1;

int b = 2;

int m = min(a, b);

Assert::AreEqual(m, 1);

}

TEST\_METHOD(TestMethod\_Exception\_MinFunctionEqual) {

int a = 1;

int b = 1;

int m = min(a, b);

Assert::AreEqual(m, 1);

}

};

}

Листинг

**Ford\_Fulkerson.cpp**

#include <iostream>

#include<fstream>

#include "Map.h"

#include"Flows.h"

using namespace std;

int main() {

ifstream input("input.txt");

Flows\* flow=new Flows(input);

// Выводим максимальный поток

int max = flow->max\_flow();

input.close();

delete flow;

cout <<"Max flow: "<< max << endl;

return 0;

}

**Flows.h**

#pragma once

#include<fstream>

#include<string>

#include "Map.h"

#include "Used\_function.h"

class Flows {

public:

~Flows() {

if (this != NULL) {

delete[] capacity;

delete[] onEnd;

delete[] nextEdge;

delete[] firstEdge;

delete[] visited;

}

}

Flows(ifstream& file) {

Map<char, int>\* Map\_from\_char\_to\_number = new Map<char, int>();

int edgeCount = 0;

numOfEdge = 0;

numOfVertex = 0;

CheckInput(file, Map\_from\_char\_to\_number);

visited = new int[numOfVertex];

firstEdge = new int[numOfVertex \* 2];

capacity = new int[numOfEdge \* 2];

onEnd = new int[numOfEdge \* 2];

nextEdge = new int[numOfEdge \* 2];

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

firstEdge[i] = -1;

visited[i] = 0;

}

file.clear();

file.seekg(ios::beg);

while (!file.eof()) {

string s1;

int vert1, vert2, cap;

getline(file, s1);

vert1 = Map\_from\_char\_to\_number->find(s1[0]);

vert2 = Map\_from\_char\_to\_number->find(s1[2]);

cap = stoi(s1.substr(4));

addEdge(vert1, vert2, edgeCount, cap);

}

Map\_from\_char\_to\_number->clear();

}

int max\_flow() {

int maxFlow = 0;

int iterationResult = 0;

while ((iterationResult = findFlow(sourceVertex, INF)) > 0) {

for (int i = 0; i < numOfVertex; ++i)

visited[i] = 0;

maxFlow += iterationResult;

}

return maxFlow;

}

void addEdge(int edge, int vertex, int& edgeCount, int cap) {

// Прямое ребро

onEnd[edgeCount] = vertex; // на конце прямого vertex

nextEdge[edgeCount] = firstEdge[edge]; // добавляем в начало списка для edge

firstEdge[edge] = edgeCount; // теперь начало списка - новое ребро

capacity[edgeCount++] = cap; // устанавливаем пропускную способность

// Обратное ребро

onEnd[edgeCount] = edge; // на конце обратного edge

nextEdge[edgeCount] = firstEdge[vertex]; // добавляем в начало списка для vertex

firstEdge[vertex] = edgeCount; // теперь начало списка - новое ребро

capacity[edgeCount++] = 0; // устанавливаем пропускную способность

}

int findFlow(int edge, int flow) {

if (edge == destinationVertex)

return flow; //возвращаем полученный минимум на пути

visited[edge] = true;

for (int edges = firstEdge[edge]; edges != -1; edges = nextEdge[edges]) {

int to = onEnd[edges];

if (!visited[to] && capacity[edges] > 0) {

int minResult = findFlow(to, min(flow, capacity[edges])); // ищем поток в поддереве

if (minResult > 0) { // если нашли

capacity[edges] -= minResult; // у прямых ребер вычетаем поток

capacity[edges ^ 1] += minResult; // к обратным прибавляем

return minResult;

}

}

}

return 0; // если не нашли поток из этой вершины вернем 0

}

void CheckInput(ifstream& file, Map<char, int>\*& Map\_from\_char\_to\_number) {

int str\_num = 1;

while (!file.eof()) {

string s1;

getline(file, s1);

if (s1.size() >= 5) {//больше или равно 5, потому что это минимальный возможный ввод(две буквы, два пробела,одна цифра)

if (!((s1[0] >= 'A' || s1[0] <= 'Z') && (s1[1] == ' '))) {

throw std::exception(string(("Error entering the first character in the string or missing a space after it. Check the correctness of the input in the file and correct these errors in the line under the number: " + to\_string(str\_num))).c\_str());

}

if (!((s1[2] >= 'A' || s1[2] <= 'Z') && (s1[3] == ' '))) {

throw std::exception(string(("Error entering the second character in the string or missing a space after it. Check the correctness of the input in the file and correct these errors in the line under the numbe: " + to\_string(str\_num))).c\_str());

}

string cur;

for (int i = 4; i < s1.size(); ++i) {

if (s1[i] >= '0' || s1[i] <= '9')

cur += s1[i];

else {

throw std::exception(string(("Error entering the third character (bandwidth) in the string or the presence of a space after it. Check the correctness of the input in the file and correct these errors in the line under the number:" + to\_string(str\_num))).c\_str());

}

}

if (!Map\_from\_char\_to\_number->find\_is(s1[0])) {// если нету тогда добавляем инкрементируем кол-во вершин

Map\_from\_char\_to\_number->insert(s1[0], numOfVertex);

++numOfVertex;

}

if (!Map\_from\_char\_to\_number->find\_is(s1[2])) {// если нету тогда добавляем инкрементируем кол-во вершин

Map\_from\_char\_to\_number->insert(s1[2], numOfVertex);

++numOfVertex;

}

}

else

{

throw std::exception(string(("A data-entry error. Check the correctness of the input in the file and correct these errors in the line under the number: " + to\_string(str\_num))).c\_str());

}

++str\_num;

++numOfEdge;

}

if (Map\_from\_char\_to\_number->find\_is('S'))

sourceVertex = Map\_from\_char\_to\_number->find('S');

else {

throw std::exception("Source is missing");

}

if (Map\_from\_char\_to\_number->find\_is('T'))

destinationVertex = Map\_from\_char\_to\_number->find('T');

else {

throw std::exception("Sink is missing");

}

}

private:

int numOfVertex;

int numOfEdge;

int sourceVertex;

int destinationVertex;

int\* capacity;

int\* onEnd;

int\* nextEdge;

int\* firstEdge;

int\* visited;

int INF = INT\_MAX;

};

**Used\_function.h**

#pragma once

template<typename T>

T min(T a, T b) {

return a > b ? b : a;

Вывод

Я научилась работать с транспортной сетью с определенными стоками и истоками для нахождения максимального потока в сети, таким образом реализовала алгоритм Форда-Фалкерсона и закрепила навыки в ООП.