**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

ОТЧЁТ

**по курсовой работе**

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

**Тема: «Потоки в сетях»**

**Вариант 2.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Бабкин И.А. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

**Постановка задачи**

Входные данные: текстовый файлы со строками 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. Необходимо найти максимальный поток в сети используя алгоритм Эдмондса-Карпа

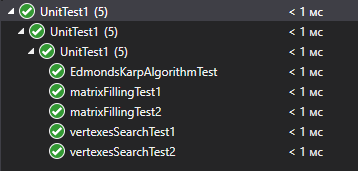
## Описание и оценка временной сложности функций

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Временная сложность |
| void vertexesSearch | Ищет уникальные вершины графа и заносит их в список | O(n) |
| void matrixFilling | Заполняет матрицу пропускных способностей | O(n) |
| bool breadthFirstSearch | Обход графа в ширину | O(V+E) |
| int EdmondsKarpAlgorithm | Алгоритм Эдмондса-Форда | O(V\*E^2) |

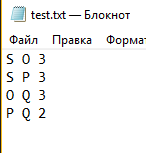
## Также в лабораторной работе были использованы LinkedList и Queue

## Описание реализованный unit-тестов

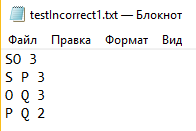
|  |
| --- |
| Название теста |
| vertexesSearchTest1 |
| vertexesSearchTest2 |
| matrixFillingTest1 |
| matrixFillingTest2 |
| EdmondsKarpAlgorithmTest |



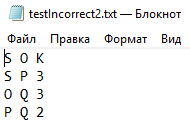
В тестах vertexesSearchTest1, matrixFillingTest1 и EdmondsKarpAlgorithmTest использовался данный текстовый файл:



В тесте vertexesSearchTest2 использовался данный текстовый файл:

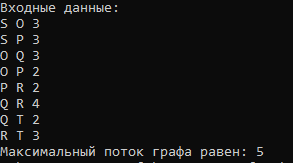


В тесте matrixFillingTest2 использовался данный текстовый файл:



## Пример работы





## Листинг

EdmondsKarp.h

#pragma once

#include "list.h"

void vertexesSearch(LinkedList<char>\* listOfCities, ifstream& fin);

void matrixFilling(LinkedList<char>\* listOfVertexes, ifstream& fin, int\*\* capacityMatrix);

bool breadthFirstSearch(int\*\* flowMatrix, int source, int target, int\* parent, size\_t size);

int EdmondsKarpAlgorithm(int\*\* capacityMatrix, int source, int target, size\_t size);

EdmondsKarp.cpp

#include "EdmondsKarp.h"

#include <string>

#include <fstream>

#include "queue.h"

void vertexesSearch(LinkedList<char>\* listOfCities, ifstream& fin)

{

string str;

char tempChar;

int counter, index;

while (!fin.eof()) {

getline(fin, str);

counter = 0;

for (size\_t i = 0; i < str.size(); i++) {

tempChar = str[i++];

index = listOfCities->find(tempChar);

if (index == -1) {

listOfCities->push\_back(tempChar);

}

counter++;

if (counter == 2) {

break;

}

if (str[i] != ' ') {

throw("Incorrect input");

}

}

}

}

void matrixFilling(LinkedList<char>\* listOfVertexes, ifstream& fin, int\*\* capacityMatrix)

{

string str;

char tempChar;

int indexI, indexJ;

int counter;

while (!fin.eof()) {

getline(fin, str);

counter = 0;

for (size\_t i = 0; i < str.size(); i++) {

tempChar = str[i++];

if (counter == 0) {

indexI = listOfVertexes->find(tempChar);

}

if (counter == 1) {

indexJ = listOfVertexes->find(tempChar);

}

if (counter == 2) {

if (int(tempChar) < int('0') || int(tempChar) > int('9')) {

throw("Incorrect input");

}

else {

capacityMatrix[indexI][indexJ] = int(tempChar) - int('0');

}

}

counter++;

}

}

}

bool breadthFirstSearch(int\*\* flowMatrix, int source, int target, int\* parent, size\_t size)

{

bool\* visitedVertexes = new bool[size];

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

visitedVertexes[i] = false;

}

Queue<int> queue;

queue.push(source);

visitedVertexes[source] = true;

parent[source] = -1;

while (!queue.empty()) {

int currentVertex = queue.first();

queue.pop();

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

if (visitedVertexes[i] == false && flowMatrix[currentVertex][i] > 0) {

queue.push(i);

parent[i] = currentVertex;

visitedVertexes[i] = true;

}

}

}

return visitedVertexes[target];

}

int EdmondsKarpAlgorithm(int\*\* capacityMatrix, int source, int target, size\_t size)

{

int currentVertex;

int\*\* flowMatrix = new int\* [size];

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

flowMatrix[i] = new int[size];

}

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

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

flowMatrix[i][j] = capacityMatrix[i][j];

}

}

int\* parent = new int[size];

int maxFlow = 0;

int pathFlow;

while (breadthFirstSearch(flowMatrix, source, target, parent, size)) {

pathFlow = 1e9;

for (int i = target; i != source; i = parent[i]) {

currentVertex = parent[i];

if (pathFlow > flowMatrix[currentVertex][i]) {

pathFlow = flowMatrix[currentVertex][i];

}

}

for (int i = target; i != source; i = parent[i]) {

currentVertex = parent[i];

flowMatrix[currentVertex][i] -= pathFlow;

flowMatrix[i][currentVertex] += pathFlow;

}

maxFlow += pathFlow;

}

return maxFlow;

}