**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

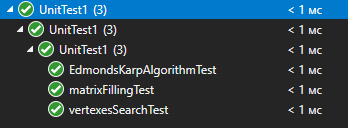
## Описание и оценка временной сложности функций

|  |  |  |
| --- | --- | --- |
| 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) |

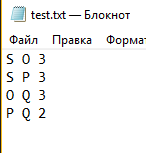
## Также в лабораторной работе был использован LinkedList и Queue

## Описание реализованный unit-тестов

|  |
| --- |
| Название теста |
| vertexesSearchTest |
| matrixFillingTest |
| EdmondsKarpAlgorithmTest |

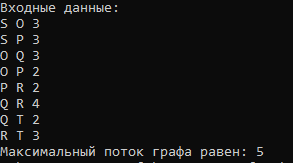


В тестах использовался данный текстовый файл:



## 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\*\* graph, int s, int t, int\* parent, size\_t size);

int EdmondsKarpAlgorithm(int\*\* capacityMatrix, int s, int t, size\_t size);

#include "EdmondsKarp.h"

#include <string>

#include <fstream>

#include "queue.h"

EdmondsKarp.cpp

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;

}

}

}

}

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) {

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

}

counter++;

}

}

}

bool breadthFirstSearch(int\*\* graph, int s, int t, int\* parent, size\_t size)

{

bool\* visited = new bool[size];

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

visited[i] = false;

}

Queue<int> queue;

queue.push(s);

visited[s] = true;

parent[s] = -1;

while (!queue.empty()) {

int cur = queue.first();

queue.pop();

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

if (visited[i] == false && graph[cur][i] > 0) {

queue.push(i);

parent[i] = cur;

visited[i] = true;

}

}

}

return visited[t];

}

int EdmondsKarpAlgorithm(int\*\* capacityMatrix, int s, int t, size\_t size)

{

int cur;

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

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

graph[i] = new int[size];

}

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

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

graph[i][j] = capacityMatrix[i][j];

}

}

int\* parent = new int[size];

int maxFlow = 0;

while (breadthFirstSearch(graph, s, t, parent, size)) {

int pathFlow = 1e9;

for (int i = t; i != s; i = parent[i]) {

cur = parent[i];

if (pathFlow > graph[cur][i]) {

pathFlow = graph[cur][i];

}

}

for (int i = t; i != s; i = parent[i]) {

cur = parent[i];

graph[cur][i] -= pathFlow;

graph[i][cur] += pathFlow;

}

maxFlow += pathFlow;

}

return maxFlow;

}