**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: «Потоки в сетях»**

**Вариант 18(3)**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9302 |  | Плюснина Е.Ю. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

**Постановка задачи**

Найти максимальный поток в сети используя алгоритм проталкивания предпотока.

# Описание реализуемого класса, оценка временной сложности

**Функции:**

void MinOn(int A, int\* Array1, int\*\* Array2, int\*\* Array3, int size)– поднятие вершины (переполненной). Оценка временной сложности: О(1)

void push(int A, int B, int\*\* Array1, int\* Array2, int\*\* Array3) – увеличение потока. Оценка временной сложности: О(n)

void Reading(Queue\* Queue1, std::ifstream& text) – чтение и поиск новых вершин графа. Оценка временной сложности: О(n)

void matrix(Queue\* Queue1, std::ifstream& text, int\*\* Matrix, int N) – матрица смежности. Оценка временной сложности: О(n)

void ProtPred(int size, int\*\* Matrix, int\*\* Matrix2) – алгоритм проталкивания предпотока. Оценка временной сложности: О(n3)

# Листинг

**Queue.cpp**

#include <iostream>

#include <string>

#include <fstream>

#ifndef Queue\_H

#define Queue\_H

class Queue

{

private:

class NodeQ {

public:

NodeQ\* next;

std::string Graph;

NodeQ(NodeQ\* next = NULL)

{

this->next = next;

};

NodeQ(std::string Graph, NodeQ\* next = NULL)

{

this->Graph = Graph;

this->next = next;

};

~NodeQ() {}

};

public:

NodeQ\* first;

NodeQ\* last;

unsigned int size;

Queue() {

last = NULL;

first = NULL;

size = 0;

}

~Queue() {

clear();

}

int get\_size() {

return size;

}

void clear() {

while (size != 0)

{

pop();

}

}

void push(std::string Graph) {

if (size == 0) {

first = new NodeQ(Graph);

last = first;

}

else {

last->next = new NodeQ(Graph);

last = last->next;

}

size++;

}

void pop() {

if (size == 0) {

return;

}

if (size == 1) {

delete first;

deletel();

}

else {

NodeQ\* current = first;

first = first->next;

delete current;

}

size--;

}

void deletel()

{

first = NULL;

last = NULL;

}

std::string at(unsigned int index)

{

if (index >= size) {

throw "Error";

}

else {

NodeQ\* current = first;

unsigned int i = 0;

while (i != index) {

current = current->next;

i++;

}

return current->Graph;

}

}

};

#endif

**Program.cpp**

#include<iostream>

#include<vector>

#include<fstream>

#include<iomanip>

#include<Windows.h>

#include "Queue.cpp"

#include "ProtPred.cpp"

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::ifstream file;

file.open("in.txt", std::ios::in);

Queue\* Queue1 = new Queue;

Reading(Queue1, file);

int size = Queue1->get\_size();

int\*\* Matrix, \*\* Matrix2;

Matrix = new int\* [size];

Matrix2 = new int\* [size];

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

Matrix[i] = new int[size];

}

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

Matrix2[i] = new int[size];

}

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

for (int j = 0; j < size; j++) {

Matrix[i][j] = 0;

}

}

file.clear();

file.seekg(0);

matrix(Queue1, file, Matrix, size);

file.close();

ProtPred(size, Matrix, Matrix2);

int path = 0;

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

if (Matrix[0][i]) {

path += Matrix2[0][i];

}

}

std::cout << "Максимальный поток:" << max(path, 0);

**ProtPred.cpp**

#include<iostream>

#include<fstream>

#include<string>

#include "Queue.cpp"

void MinOn(int A, int\* Array1, int\*\* Array2, int\*\* Array3, int size)

{

int Min = 1000000;

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

if (Array3[A][i] - Array2[A][i] > 0) {

Min = std::min(Min, Array1[i]);

}

}

if (Min == 1000000) {

return;

}

Array1[A] = Min + 1;

}

void push(int A, int B, int\*\* Array1, int\* Array2, int\*\* Array3)

{

int Min = std::min(Array2[A], Array3[A][B] - Array1[A][B]);

Array1[A][B] += Min;

Array1[B][A] = -Array1[A][B];

Array2[A] -= Min;

Array2[B] += Min;

}

void Reading(Queue\* Queue1, std::ifstream& text) {

std::string str, Graph;

int in = 0;

bool flag = true;

while (!text.eof()) {

std::getline(text, str);

for (unsigned int i = 0; i < str.size(); i++) {

if (str[i] != ' ')

{

Graph += str[i];

if (str[i + 1] != ' ') {

throw "Error";

}

}

if (in == 2) {

break;

}

else {

if (in != 2) {

for (int j = 0; j < Queue1->get\_size(); j++) {

if (Graph == Queue1->at(j)) {

flag = false;

}

}

if (flag) {

Queue1->push(Graph);

}

flag = true;

in++;

Graph.clear();

}

else {

Graph.clear();

break;

}

}

}

Graph.clear();

in = 0;

}

str.clear();

Graph.clear();

}

void matrix(Queue\* Queue1, std::ifstream& text, int\*\* Matrix, int N) {

std::string str, str2;

int A = 0, B = 0;

int in = 0;

while (!text.eof()) {

std::getline(text, str);

for (unsigned int i = 0; i < str.size() + 1; i++) {

if (str[i] != ' ' && i < str.size())

{

str2 += str[i];

}

else if (in < 2) {

for (int j = 0; j < Queue1->get\_size(); j++) {

if (str2 == Queue1->at(j)) {

if (in == 0) {

A = j;

}

else {

B = j;

}

break;

}

}

in++;

str2.clear();

}

else {

if (in == 2) {

Matrix[A][B] = atoi(str2.c\_str());

}

in++;

str2.clear();

}

}

in = 0;

str2.clear();

}

str.clear();

str2.clear();

}

void ProtPred(int size, int\*\* Matrix, int\*\* Matrix2) {

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

for (int j = 0; j < size; j++) {

Matrix2[i][j] = 0;

}

}

for (int i = 1; i < size; i++) {

Matrix2[0][i] = Matrix[0][i];

Matrix2[i][0] = -Matrix[0][i];

}

int\* Array1 = new int[size];

Array1[0] = size;

int\* Array2 = new int[size];

for (int i = 1; i < size; i++) {

Array2[i] = Matrix2[0][i];

}

int i;

for (i = 1; i < size - 1; i++) {

if (Array2[i] > 0) {

break;

}

}

int j;

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

if (Matrix[i][j] - Matrix2[i][j] > 0 && Array1[i] == Array1[j] + 1) {

break;

}

}

if (j < size) {

push(i, j, Matrix2, Array2, Matrix);

}

else {

MinOn(i, Array1, Matrix2, Matrix, size);

}

}

**UnitTest1.cpp**

#include "CppUnitTest.h"

#include "..\AlgStrD\_4sem\_Kurs\ProtPred.cpp"

#include "..\AlgStrD\_4sem\_Kurs\Queue.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(UnitTest1)

{

public:

TEST\_METHOD(TestMethod1)

{

Queue\* Queue1 = new Queue;

std::ifstream file;

file.open("C:\\Users\\walke\\source\\repos\\AlgStrD\_4sem\_Kurs\\AlgStrD\_4sem\_Kurs\\in.txt", std::ios::in);

Reading(Queue1, file);

int size = Queue1->get\_size();

int\*\* Matrix;

Matrix = new int\* [size];

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

Matrix[i] = new int[size];

}

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

for (int j = 0; j < size; j++) {

Matrix[i][j] = 0;

}

}

file.clear();

file.seekg(0);

int Matrix2[4][4] = { {0,6,5,0},{0,0,0,9},{0,2,3,4},{0,0,0,1} };

matrix(Queue1, file, Matrix, size);

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

Assert::IsTrue(Matrix[i][j] == Matrix2[i][j]);

}

}

file.close();

delete Matrix;

}

TEST\_METHOD(TestMethod2)

{

Queue\* Queue1 = new Queue;

std::ifstream file;

std::string Array1[6] = { "S","O","Q","R","T" };

file.open("C:\\Users\\walke\\source\\repos\\AlgStrD\_4sem\_Kurs\\AlgStrD\_4sem\_Kurs\\in.txt", std::ios::in);

Reading(Queue1, file);

for (int i = 0; i < Queue1->get\_size(); i++) {

Assert::IsTrue(Queue1->at(i) == Array1[i]);

}

file.close();

}

};

# }

# Вывод

Я научилась реализовывать алгоритм проталкивания предпотока.