**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 3.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Баязитов О.О. |
| 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 проталкивания предпотока

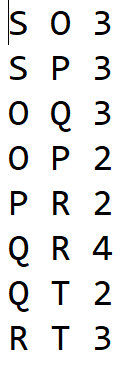
## Описание и оценка временной сложности функций

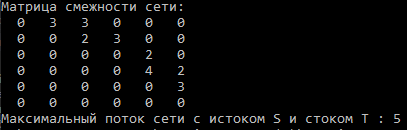
|  |  |  |
| --- | --- | --- |
| Функция | Описание | Временная сложность |
| void reading | Нахождение уникальных вершин графа | O(N) |
| void creating\_matrix | Заполнение матрицы | O(N) |
| void Pushing\_preflow | Алгоритм проталкивания предпотока | O(N^4) |
| void lift | Поднятие переполненной вершины на максимальную высоту | O(1) |
| void push | Максимальное увеличении потока из V1 в V2 | O(N) |

## Описание реализованный unit-тестов

|  |  |
| --- | --- |
| Название теста |  |
| TestMethod1 | Проверка очереди с уникальными вершинами |
| TestMethod2 | Проверка составления матрицы |
| TestMethod3 | Проверка алгоритма проталкивания предпотока |
| TestMethod4 | Проверка исключения |
| TestMethod5 | Проверка исключения |

## Пример работы





## Листинг

**function.h**

#include"queue.h"

void push(int u, int v, int\*\* f, int\* e, int\*\* c)

{

int d = 0;

if (e[u] < c[u][v] - f[u][v])

{

d = e[u];

}

else

{

d = c[u][v] - f[u][v];

}

f[u][v] += d;

f[v][u] = -f[u][v];

e[u] -= d;

e[v] += d;

}

void lift(int u, int\* h, int\*\* f, int\*\* c, int size)

{

int d = 999999;

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

{

if (c[u][i] - f[u][i] > 0)

{

if (d < h[i])

{

}

else

{

d = h[i];

}

}

}

if (d == 999999)

return;

h[u] = d + 1;

}

void Pushing\_preflow(int size, int\*\* matrix, int\*\* matrix\_1)

{

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

{

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

{

matrix\_1[i][j] = 0;

}

}

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

{

matrix\_1[0][i] = matrix[0][i];

matrix\_1[i][0] = -matrix[0][i];

}

int\* vect\_1 = new int[size];

vect\_1[0] = size;

int\* vect\_2 = new int[size];

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

vect\_2[i] = matrix\_1[0][i];

for (; ; )

{

int i;

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

if (vect\_2[i] > 0)

break;

if (i == size - 1)

break;

int j;

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

if (matrix[i][j] - matrix\_1[i][j] > 0 && vect\_1[i] == vect\_1[j] + 1)

break;

if (j < size)

push(i, j, matrix\_1, vect\_2, matrix);

else

lift(i, vect\_1, matrix\_1, matrix, size);

}

}

void reading(queue\* que, std::ifstream& text)

{

std::string str, graf;

int count = 0;

bool flag = true;

while (!text.eof())

{

std::getline(text, str);

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

{

if (count == 2)

{

break;

}

if (str[i] != ' ')

{

graf += str[i];

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

throw "incorrect input";

}

else if (count != 2) {

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

{

if (graf == que->at(j))

{

flag = false;

}

}

if (flag)

{

que->push(graf);

}

flag = true;

count++;

graf.clear();

}

else {

graf.clear();

break;

}

}

graf.clear();

count = 0;

}

str.clear();

graf.clear();

}

void creating\_matrix(queue\* que, std::ifstream& text, int\*\* matrix, int N)

{

std::string str, word;

int count = 0;

int unit\_1 = 0, unit\_2 = 0;

while (!text.eof()) {

std::getline(text, str);

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

{

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

{

word += str[i];

}

else if (count < 2)

{

if (count == 1)

{

if (str[i + 1] != '0' && str[i + 1] != '1' && str[i + 1] != '2' && str[i + 1] != '3' && str[i + 1] != '4' && str[i + 1] != '5' && str[i + 1] != '6' && str[i + 1] != '7' & str[i + 1] != '8' && str[i + 1] != '9')

throw "incorrect input";

}

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

{

if (word == que->at(j))

{

if (count == 0)

unit\_1 = j;

else

unit\_2 = j;

break;

}

}

count++;

word.clear();

}

else

{

if (count == 2)

{

matrix[unit\_1][unit\_2] = atoi(word.c\_str());

}

count++;

word.clear();

}

}

count = 0;

word.clear();

}

str.clear();

word.clear();

}

**queue.h**

#ifndef queue\_H

#define queue\_H

#include <iostream>

#include <string>

#include <fstream>

class queue

{

private:

class queueNode

{

public:

queueNode\* next;

std::string graf;

queueNode(queueNode\* next = nullptr)

{

this->next = next;

};

queueNode(std::string graf, queueNode\* next = nullptr)

{

this->graf = graf;

this->next = next;

};

~queueNode()

{

}

};

public:

queueNode\* head;

queueNode\* tail;

unsigned int size;

queue() {

tail = nullptr;

head = nullptr;

size = 0;

}

~queue()

{

clear();

}

int get\_size()

{

return size;

}

void clear()

{

while (size != 0)

{

pop();

}

}

void push(std::string graf)

{

if (size == 0) {

head = new queueNode(graf);

tail = head;

}

else {

tail->next = new queueNode(graf);

tail = tail->next;

}

size++;

}

void pop()

{

if (size == 0)

{

return;

}

if (size == 1)

{

delete head;

reset\_list();

}

else {

queueNode\* current = head;

head = head->next;

delete current;

}

size--;

}

void reset\_list()

{

head = nullptr;

tail = nullptr;

}

std::string at(unsigned int index)

{

if (index >= size)

{

throw "Index is greater than list size";

}

else {

queueNode\* current = head;

unsigned int counter = 0;

while (counter != index) {

current = current->next;

counter++;

}

return current->graf;

}

}

};

#endif

**Source.cpp**

#include<iostream>

#include<fstream>

#include<iomanip>

#include<Windows.h>

#include"queue.h"

#include"function.h"

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::ifstream file;

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

queue\* que = new queue;

reading(que, file);

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

int\*\* matrix, \*\* matrix\_1;

matrix = new int\* [size];

matrix\_1 = new int\* [size];

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

{

matrix[i] = new int[size];

}

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

{

matrix\_1[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);

creating\_matrix(que, file, matrix, size);

file.close();

Pushing\_preflow(size, matrix, matrix\_1);

int flow = 0;

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

if (matrix[0][i])

flow += matrix\_1[0][i];

cout << "Матрица смежности сети:\n";

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

{

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

{

std::cout << std::setw(3) << matrix[i][j] << " ";

}

std::cout << std::endl;

}

cout << "Максимальный поток сети c истоком "<<que->at(0) <<" и стоком "<< que->at(que->get\_size()-1)<<" : ";

cout << max(flow, 0);

}