Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Алтайский государственный технический университет

им. И.И. Ползунова»

Факультет (институт) Информационных технологий

Кафедра Прикладная математика

Отчет защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

А.И.Потупчик

(подпись преподавателя) (инициалы, фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Отчет

по лабораторной (практической) работе №5

**Разработка и анализ программ обхода графа**

(название лабораторной (практической) работы)

по дисциплине \_\_\_\_\_\_\_\_Типы и структуры данных \_\_\_\_\_\_\_\_\_\_\_\_

(наименование дисциплины)

ЛР 09.03.04.15.000 ОТ

(обозначение документа)

Студент группы ПИ-02 В.И. Немчинов

(инициалы, фамилия)

Преподаватель доцент, доцент А.И.Потупчик

(должность, ученое звание) (инициалы, фамилия)

Барнаул 2022

**Задание:**

1. Разработать и отладить программу на языке C++, реализующую работу с графом в соответствии с вариантом. Выполнить оценку временной и емкостной сложности программы.
2. Исходные данные поместить в файл input.dat
3. Результаты вывести на экран. Исходные данные и результаты вывести также в выходной файл output.dat

**Вариант 5:** Найти все простые пути между двумя заданными вершинами, если граф задан списковой структурой.

**Код программы:**

**5.cpp:**

#include <iostream>

#include "Graph.h"

using namespace std;

int main()

{

Graph<int> graph;

string path = "D:/ВУЗ/Программирование/Типы и структуры/лаб\_5 (графы)/input.dat";

graph.create\_graph\_out\_of\_file(path);

graph.show\_graph\_table();

graph.find\_the\_way(1, 3, WAY::SHORTEST);

graph.create\_file\_out\_of\_graph(WAY::SHORTEST);

///\*x1\*/ graph.add\_node(1);

///\*x2\*/ graph.add\_node(2);

///\*x3\*/ graph.add\_node(3);

///\*x4\*/ graph.add\_node(4);

///\*x5\*/ graph.add\_node(5);

///\*x6\*/ graph.add\_node(6);

///\*x7\*/ graph.add\_node(7);

///\*x8\*/ graph.add\_node(8);

///\*x9\*/ graph.add\_node(9);

///\*x10\*/ graph.add\_node(10);

///\*a1\*/ graph.change\_connections(1, 2, CONNECTION::CREATE);

///\*a2\*/ graph.change\_connections(1, 3, CONNECTION::CREATE);

///\*a3\*/ graph.change\_connections(2, 2, CONNECTION::CREATE);

///\*a4\*/ graph.change\_connections(2, 5, CONNECTION::CREATE);

///\*a5\*/ graph.change\_connections(4, 3, CONNECTION::CREATE);

///\*a6\*/ graph.change\_connections(5, 4, CONNECTION::CREATE); //

///\*a7\*/ graph.change\_connections(6, 5, CONNECTION::CREATE);

///\*a8\*/ graph.change\_connections(1, 5, CONNECTION::CREATE);

///\*a9\*/ graph.change\_connections(6, 1, CONNECTION::CREATE);

///\*a10\*/ graph.change\_connections(6, 6, CONNECTION::CREATE);

///\*a11\*/ graph.change\_connections(2, 7, CONNECTION::CREATE);

///\*a12\*/ graph.change\_connections(7, 9, CONNECTION::CREATE);

///\*a13\*/ graph.change\_connections(2, 8, CONNECTION::CREATE);

///\*a14\*/ graph.change\_connections(1, 8, CONNECTION::CREATE);

///\*a15\*/ graph.change\_connections(8, 9, CONNECTION::CREATE);

///\*a16\*/ graph.change\_connections(8, 10, CONNECTION::CREATE);

///\*a17\*/ graph.change\_connections(9, 3, CONNECTION::CREATE);

///\*a18\*/ graph.change\_connections(9, 4, CONNECTION::CREATE);

///\*a19\*/ graph.change\_connections(10, 4, CONNECTION::CREATE);

///\*a20\*/ graph.change\_connections(5, 9, CONNECTION::CREATE);

//graph.show\_graph\_table();

//graph.find\_the\_way(1, 4, WAY::SHORTEST);

}

**Graph.h:**

#pragma once

#include <iostream>

#include <vector>

#include <map>

#include <set>

#include <stack>

#include <string>

#include <fstream>

#include "Queue.h"

using namespace std;

enum class CONNECTION

{

INVERT,

DELETE,

CREATE

};

enum class WAY

{

LONGEST,

SHORTEST,

ALL

};

template<class T>

class Graph

{

class Node

{

public:

T data;

map<int, Node\*> connection\_out;

map<int, Node\*> connection\_in;

bool is\_passed;

bool is\_node\_connected\_out\_to(int node\_this\_node\_connected\_to)

{

for (auto iter = connection\_out.begin(); iter != connection\_out.end(); iter++)

{

if (iter->first == node\_this\_node\_connected\_to)

return true;

}

return false;

}

bool is\_node\_connected\_in\_from(int node\_this\_node\_connected\_from)

{

for (auto iter = connection\_in.begin(); iter != connection\_in.end(); iter++)

{

if (iter->first == node\_this\_node\_connected\_from)

return true;

}

return false;

}

Node(T new\_data)

{

this->data = new\_data;

this->is\_passed = false;

}

};

map<int, Node\*> all\_nodes;

int size;

set< vector<int> > all\_ways; // здесь находятся все пути из ноды в ноду после выполнения void find\_the\_shortest\_way(int node\_from, int node\_to, WAY action)

void connect\_node\_to\_node(int node\_from, int node\_to)

{

auto iter\_1 = all\_nodes.find(node\_to);

auto iter\_2 = all\_nodes.find(node\_from);

if (iter\_1 != all\_nodes.end() && iter\_2 != all\_nodes.end())

{

all\_nodes[node\_from]->connection\_out.insert(make\_pair(iter\_1->first, iter\_1->second));

all\_nodes[node\_to]->connection\_in.insert(make\_pair(iter\_2->first, iter\_2->second));

}

}

void disconnect\_node\_from\_node(int node\_from, int node\_to)

{

auto iter\_1 = all\_nodes.find(node\_to);

auto iter\_2 = all\_nodes.find(node\_from);

if (iter\_1 != all\_nodes.end() && iter\_2 != all\_nodes.end())

{

all\_nodes[node\_from]->connection\_out.erase(iter\_1);

all\_nodes[node\_to]->connection\_in.erase(iter\_2);

}

}

public:

Graph()

{

size = 0;

}

void add\_node(T value)

{

Node\* new\_node = new Node(value);

all\_nodes.insert(make\_pair(++size, new\_node));

}

void change\_connections(int number\_node\_from, int number\_node\_to, CONNECTION action)

{

if (all\_nodes[number\_node\_from] != nullptr && all\_nodes[number\_node\_to] != nullptr)

switch (action)

{

case CONNECTION::CREATE:

connect\_node\_to\_node(number\_node\_from, number\_node\_to);

break;

case CONNECTION::DELETE:

disconnect\_node\_from\_node(number\_node\_from, number\_node\_to);

break;

case CONNECTION::INVERT:

disconnect\_node\_from\_node(number\_node\_from, number\_node\_to);

connect\_node\_to\_node(number\_node\_to, number\_node\_from);

break;

}

}

void show\_graph\_table()

{

cout << " ";

for (auto iter = all\_nodes.begin(); iter != all\_nodes.end(); iter++)

{

cout << "N" << iter->first << " ";

}

cout << endl;

for (auto iter\_1 = all\_nodes.begin(); iter\_1 != all\_nodes.end(); iter\_1++)

{

if (iter\_1->first < 10)

cout << "N" << iter\_1->first << " ";

else

cout << "N" << iter\_1->first;

for (auto iter\_2 = all\_nodes.begin(); iter\_2 != all\_nodes.end(); iter\_2++)

{

if ((iter\_1->second)->is\_node\_connected\_out\_to(iter\_2->first) == true)

{

cout << " 1 ";

}

else

{

cout << " 0 ";

}

}

cout << endl;

}

}

void clear\_node\_passes()

{

for (auto iter = all\_nodes.begin(); iter != all\_nodes.end(); iter++)

{

(iter->second)->is\_passed = false;

}

}

void bypass\_in\_death(int first\_el)

{

all\_nodes[first\_el]->is\_passed = true;

for (auto iter = all\_nodes[first\_el]->connection\_out.begin(); iter != all\_nodes[first\_el]->connection\_out.end(); iter++)

{

if ((iter->second)->is\_passed == false)

{

bypass\_in\_death(iter->first);

}

}

}

void find\_the\_way(int node\_from, int node\_to, WAY action)

{

int min;

vector<int> temp\_arr;

stack<int> connections\_to\_pass;

static Queue<int> bypassed\_nodes;

bypassed\_nodes.push(node\_from);

for (auto iter = all\_nodes[node\_from]->connection\_out.begin(); iter != all\_nodes[node\_from]->connection\_out.end(); iter++)

{

if (iter->first == node\_to)

{

for (int i = 0; i < bypassed\_nodes.size(); i++)

{

temp\_arr.push\_back(bypassed\_nodes.front());

bypassed\_nodes.push(bypassed\_nodes.pop());

}

temp\_arr.push\_back(node\_to);

all\_ways.insert(temp\_arr);

}

else if (iter->first != node\_from)

connections\_to\_pass.push(iter->first);

}

for (int i = 0; i < connections\_to\_pass.size(); i++)

{

find\_the\_way(connections\_to\_pass.top(), node\_to, action);

connections\_to\_pass.pop(); i--;

}

bypassed\_nodes.pop\_back();

if (bypassed\_nodes.empty())

{

switch (action)

{

case WAY::ALL:

cout << endl;

cout << "All ways:" << endl;

for (auto iter = all\_ways.begin(); iter != all\_ways.end(); iter++)

{

for (int i = 0; i < (\*iter).size(); i++)

if (i == 0)

cout << (\*iter)[i];

else

cout << " -> " << (\*iter)[i];

cout << endl;

}

break;

case WAY::SHORTEST:

cout << endl;

cout << "Shortest ways:" << endl;

for (auto iter = all\_ways.begin(); iter != all\_ways.end(); iter++)

{

if (iter == all\_ways.begin())

min = iter->size();

else if (iter->size() < min)

min = iter->size();

}

for (auto iter = all\_ways.begin(); iter != all\_ways.end(); iter++)

{

if (iter->size() == min)

{

for (int i = 0; i < (\*iter).size(); i++)

if (i == 0)

cout << (\*iter)[i];

else

cout << " -> " << (\*iter)[i];

cout << endl;

}

}

break;

}

}

}

bool create\_graph\_out\_of\_file(const string path)

{

int temp\_fin, size;

int\*\* matrix;

ifstream fin;

fin.exceptions(ifstream::badbit | ifstream::failbit);

try

{

fin.open(path);

}

catch (const ifstream::failure& ex)

{

cout << ex.what() << endl;

return false;

}

if (fin.is\_open())

{

fin >> size;

matrix = new int\* [size];

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

matrix[i] = new int[size];

while (!fin.eof())

{

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

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

{

fin >> temp\_fin;

matrix[i][j] = temp\_fin;

}

}

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

add\_node(i + 1);

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

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

{

if (matrix[i][j] == 1)

change\_connections(i+1, j+1, CONNECTION::CREATE);

}

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

delete[]matrix[i];

delete[]matrix;

fin.close();

return true;

}

}

bool create\_file\_out\_of\_graph(WAY action)

{

int min;

string path = "D:/ВУЗ/Программирование/Типы и структуры/лаб\_5 (графы)/output.dat";

fstream fout;

fout.exceptions(ofstream::badbit | ofstream::failbit);

try

{

fout.open(path, ios\_base::out);

}

catch (const ofstream::failure& ex)

{

cout << ex.what() << endl;

return false;

}

if (fout.is\_open())

{

fout << " ";

for (auto iter = all\_nodes.begin(); iter != all\_nodes.end(); iter++)

{

fout << "N" << iter->first << " ";

}

fout << endl;

for (auto iter\_1 = all\_nodes.begin(); iter\_1 != all\_nodes.end(); iter\_1++)

{

if (iter\_1->first < 10)

fout << "N" << iter\_1->first << " ";

else

fout << "N" << iter\_1->first;

for (auto iter\_2 = all\_nodes.begin(); iter\_2 != all\_nodes.end(); iter\_2++)

{

if ((iter\_1->second)->is\_node\_connected\_out\_to(iter\_2->first) == true)

{

fout << " 1 ";

}

else

{

fout << " 0 ";

}

}

fout << endl;

}

switch (action)

{

case WAY::ALL:

fout << endl;

fout << "All ways:" << endl;

for (auto iter = all\_ways.begin(); iter != all\_ways.end(); iter++)

{

for (int i = 0; i < (\*iter).size(); i++)

if (i == 0)

fout << (\*iter)[i];

else

fout << " -> " << (\*iter)[i];

fout << endl;

}

break;

case WAY::SHORTEST:

fout << endl;

fout << "Shortest ways:" << endl;

for (auto iter = all\_ways.begin(); iter != all\_ways.end(); iter++)

{

if (iter == all\_ways.begin())

min = iter->size();

else if (iter->size() < min)

min = iter->size();

}

for (auto iter = all\_ways.begin(); iter != all\_ways.end(); iter++)

{

if (iter->size() == min)

{

for (int i = 0; i < (\*iter).size(); i++)

if (i == 0)

fout << (\*iter)[i];

else

fout << " -> " << (\*iter)[i];

fout << endl;

}

}

break;

}

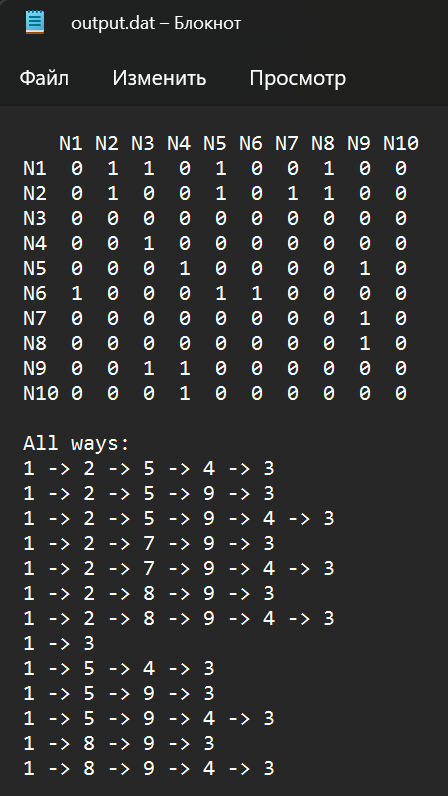
fout.close();

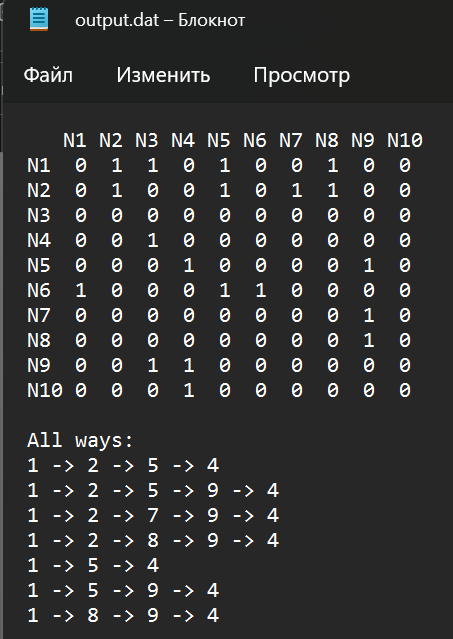
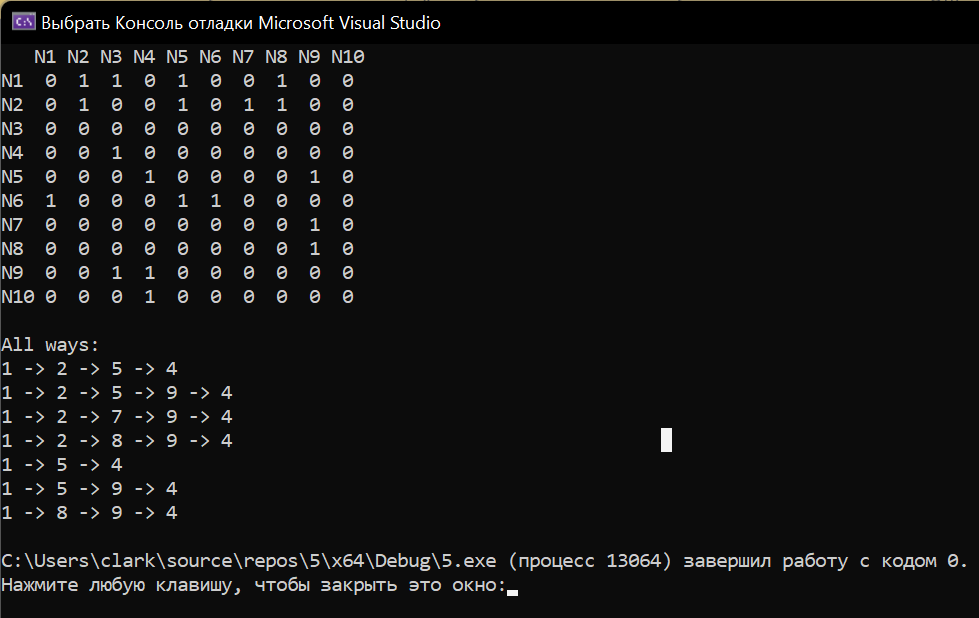
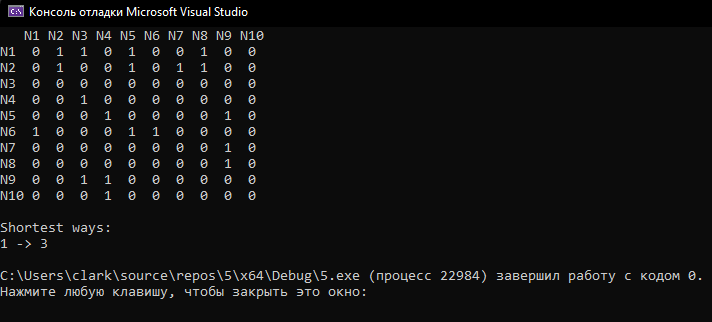
return true;

}

}

};

**Результаты работы программы:**

****

