МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Череповецкий государственный университет»

**Лабораторная работа № 2**

**«Графы»**

**Выполнил:**

**студент гр.** 1ИВТпб-01-21оп

Климов А.Г. **Проверил: преподаватель**

Пышницкий К.М. **Отметка о зачете:**

Череповец

2017 год

**Ход работы**

**Задание 5.** Составить программу, решающую задачу. Входные данные находятся в файле INPUT.TXT. Для задачи изобразить на экране соответствующий граф.

Город расположен на нескольких островах, связанных между собой мостами. Между двумя островами может быть более одного моста. Выяснить, можно ли, выйдя из дома, вернуться обратно, пройдя по каждому мосту только один раз. Если это возможно, то предложить один из путей.

Технические требования:

Количество островов N<=30. Острова пронумерованы числами от 1 до N.

Формат файла INPUT.TXT:

1-я строка – количество островов, последующие строки содержат по два натуральных числа – номера островов, связанных мостом.

**Функции:**

Стек и функции для записи, извлечения и получения без извлечения.

Добавление смежной вершины и удаление узла.

Определение эйлеровости графа.

**Логическое проектирование:**

1. Считываем количество вершин N из файла.
2. Создаём динамический массив N\*N и заполняем нулями.
3. Считываем “мосты” и формируем матрицу смежности.
4. Выводим матрицу смежности.
5. Рисуем граф по матрице смежности.
6. Проверяем через функцию граф на эйлеровость. Если граф имеет эйлеров цикл, то выводим его.

**Текст программы:**

**а)**

#pragma once

#include <cmath>

#include <iostream>

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

#include <fstream>

#include <string>

#define PI 3.14159265

int result[];

struct Node

{

int inf;

Node \*next;

};

//============================Stack==============================

void push(Node \*&st, int dat)

{ // Загрузка числа в стек

Node \*el = new Node;

el->inf = dat;

el->next = st;

st = el;

}

int pop(Node \*&st)

{ // Извлечение из стека

int value = st->inf;

Node \*temp = st;

st = st->next;

delete temp;

return value;

}

int peek(Node \*st)

{ // Получение числа без его извлечения

return st->inf;

}

//==============================================================

Node \*\*graph; // Массив списков смежности

const int vertex = 1; // Первая вершина

void add(Node\*& list, int data)

{ //Добавление смежной вершины

if (!list) { list = new Node; list->inf = data; list->next = 0; return; }

Node \*temp = list;

while (temp->next)temp = temp->next;

Node \*elem = new Node;

elem->inf = data;

elem->next = NULL;

temp->next = elem;

}

void del(Node\* &l, int key)

{ // Удаление вершины key из списка

if (l->inf == key) { Node \*tmp = l; l = l->next; delete tmp; }

else

{

Node \*tmp = l;

while (tmp)

{

if (tmp->next) // есть следующая вершина

if (tmp->next->inf == key)

{ // и она искомая

Node \*tmp2 = tmp->next;

tmp->next = tmp->next->next;

delete tmp2;

}

tmp = tmp->next;

}

}

}

int eiler(Node \*\*gr, int num)

{ // Определение эйлеровости графа

int count;

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

{ //проходим все вершины

count = 0;

Node \*tmp = gr[i];

while (tmp)

{ // считаем степень

count++;

tmp = tmp->next;

}

if (count % 2 == 1)return 0; // степень нечетная

}

return 1; // все степени четные

}

void eiler\_path(Node \*\*gr)

{ //Построение цикла

Node \*S = NULL;// Стек для пройденных вершин

int v = vertex;// 1я вершина (произвольная)

int u;

push(S, v); //сохраняем ее в стек

while (S)

{ //пока стек не пуст

v = peek(S); // текущая вершина

if (!gr[v]) { // если нет инцидентных ребер

v = pop(S); std::cout << v + 1 << " "; //выводим вершину (у нас отсчет от 1, поэтому +1)

}

else

{

u = gr[v]->inf; push(S, u); //проходим в следующую вершину

del(gr[v], u); del(gr[u], v); //удаляем пройденное ребро

}

}

}

char\* SystemStringToChar(System::String^ string) {

return (char\*)(void\*)System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(string);

}

namespace Проект1 {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace System::IO;

using namespace System::Text::RegularExpressions;

/// <summary>

/// Сводка для MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктора

//

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::Button^ button1;

protected:

private:

/// <summary>

/// Требуется переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Обязательный метод для поддержки конструктора - не изменяйте

/// содержимое данного метода при помощи редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->button1 = (gcnew System::Windows::Forms::Button());

this->SuspendLayout();

//

// button1

//

this->button1->Location = System::Drawing::Point(21, 413);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(75, 23);

this->button1->TabIndex = 0;

this->button1->Text = L"Жми";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(984, 461);

this->Controls->Add(this->button1);

this->Name = L"MyForm";

this->Text = L"MyForm";

this->Load += gcnew System::EventHandler(this, &MyForm::MyForm\_Load);

this->Paint += gcnew System::Windows::Forms::PaintEventHandler(this, &MyForm::MyForm\_Paint);

this->MouseDown += gcnew System::Windows::Forms::MouseEventHandler(this, &MyForm::MyForm\_MouseDown);

this->ResumeLayout(false);

}

#pragma endregion

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

Graphics^ g = this->CreateGraphics();

Pen^ MyPen = gcnew Pen(System::Drawing::Color::Green);

Pen^ MyPen1 = gcnew Pen(System::Drawing::Color::Blue);

SolidBrush^ mySolidBrush = gcnew SolidBrush(System::Drawing::Color:: White);

setlocale(LC\_ALL, "Russian");

int NN;

int check1,check2;

int \*num;

int count = 0;

std::string x2x[10]; // работает

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

file >> NN;

std::cout << "Количество вершин N=" << NN;

std::cout << "\n";

std::cout << "\n";

std::cout <<"Мосты: "<< "\n";

// динамическое создание двумерного массива чисел на NN\*NN элементов

int \*\*ptrarray = new int\*[NN]; // NN строк в массиве

for (int count = 0; count < NN; count++)

ptrarray[count] = new int[NN]; // NN столбцов

// заполнение массива нулями

for (int count\_row = 0; count\_row < NN; count\_row++)

for (int count\_column = 0; count\_column < NN; count\_column++)

ptrarray[count\_row][count\_column] = 0;

// считываем соединённые вершины из файлаи формируем матрицу смежности

while (!file.eof()) {

if (!(file >> check1>>check2)) break;

std::cout << check1 <<" "<< check2<<" | ";

ptrarray[check1-1][check2-1] = 1;

ptrarray[check2-1][check1-1] = 1;

}

std::cout << "\n";

std::cout << "\n";

std::cout << "Матрица смежности: " << "\n";

file.close();

// вывод массива

for (int count\_row = 0; count\_row < NN; count\_row++)

{

for (int count\_column = 0; count\_column < NN; count\_column++)

std::cout << ptrarray[count\_row][count\_column] << " ";

std::cout << "\n";

}

String^ label44 = gcnew String(x2x[1].c\_str());

double x, y,w\_h=300, x0=350, y0=50;

//g->DrawEllipse(MyPen, Rectangle(x0, y0, w\_h, w\_h)); // окружность, вокруг которой размещаются вершины

System::Drawing::Font^ drawFont = gcnew System::Drawing::Font("Arial", 16);

SolidBrush^ drawBrush = gcnew SolidBrush(Color::Black);

int output\_num = 1;

String^ drawString;

//int N = 5;

int N = NN;

double r = w\_h / 2;

double \*temp\_x;

double \*temp\_y;

temp\_x = new double[N];

temp\_y =new double[N];

double xx, yy;

double gr = 360 / N;

double x\_xx;

double y\_yy;

double aa;

double x\_Ellipse;

double y\_Ellipse;

for (int i = 0; i <= N - 1; i++) {

aa += gr;

x\_xx = r\*cos(aa\*PI / 180);

y\_yy = r\*sin(aa\*PI / 180);

x\_Ellipse=x\_xx + 150 + x0 - 25;

y\_Ellipse=y\_yy + 150 + y0 - 25;

// рисуем окружность и круг

g->FillEllipse(mySolidBrush, x\_Ellipse, y\_Ellipse, 50, 50);

g->DrawEllipse(MyPen, Rectangle(x\_Ellipse, y\_Ellipse, 50, 50));

// рисуем подярковый номер вершины

drawString = System::Convert::ToString(output\_num);

output\_num++;

PointF drawPoint = PointF(x\_Ellipse+15, y\_Ellipse+15);

g->DrawString(drawString, drawFont, drawBrush, drawPoint);

//записываем координаты

temp\_x[i] = x\_Ellipse+25;

temp\_y[i] = y\_Ellipse+25;

}

for (int i = 0; i <= N - 1; i++) {

for (int j = 0; j <= N - 1; j++) {

int fff = temp\_y[j];

if (ptrarray[i][j] == 1&& ptrarray[j][i] == 1) { g->DrawLine(MyPen1, temp\_x[j], fff, temp\_x[i], temp\_y[i]); } //если есть мост, то рисуем его на графе

}

}

int n=N; //cin >> n; // Количество вершин

int zn;// Текущее значение

graph = new Node\*[n];

for (int i = 0; i<n; i++)graph[i] = NULL;

for (int i = 0; i<n; i++) // заполняем массив списков

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

{

//cin >> zn;

zn=ptrarray[i][j]; //std::cout << ptrarray[i][j];

if (zn) add(graph[i], j);

}

std::cout << "\n\nЭйлеров цикл: ";

if (eiler(graph, n))eiler\_path(graph);

else std::cout << "Граф не является эйлеровым.";

}

private: System::Void MyForm\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {

Graphics^ g=e->Graphics;

Pen^ MyPen=gcnew Pen(System::Drawing::Color::Red);

}

private: System::Void MyForm\_MouseDown(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

}

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

}

};

}