Министерство образования и науки Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования

«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ПНИПУ)

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа по

«Теория алгоритмов и структуры данных»

вариант № 15

за 2 семестр

Выполнил:

студент группы РИС-21-1бз

Тимолянов Григорий Константинович

Проверила:

Доцент кафедры ИТАС,

к.т.н. Полякова О.А.

2024

###### Лабораторная работа №2

###### Введение в теорию графов

**Цель:** Получить практические навыки работы с графами.

###### Постановка задачи

###### 

###### Функции для решения задачи

Класс Graph

Graph.h

#pragma once

#include<vector>

#include<iostream>

const int maxSize = 20;

const int WINDOW\_WIDTH = 800;

const int WINDOW\_HEIGHT = 600;

class Graph

{

std::vector<int> vertList;

int adjMatrix[maxSize][maxSize];

public:

Graph();

~Graph();

int GetVertPos(const int&);

bool IsEmpty();

bool IsFull();

int GetAmountVerts();

int GetAmountEdges();

int GetWeight(const int&, const int&);

std::vector<int> GetNbrs(const int&);

void InsertVertex(const int&);

void InsertEdge(const int& vertex1, const int& vertex2, int weight = 0);

void Print();

void DFS(int, bool\*);

static void initWindow(int argc, char\*\* argv);

};

Graph.cpp

#include "Graph.h"

#include "GL/glut.h"

#include <iostream>

#include "draw.h"

Graph::Graph()

{

// Перебор строк и столбцов матрицы смежности и заполнение ее нулями

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

{

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

{

this->adjMatrix[i][j] = 0;

}

}

}

Graph::~Graph()

{

}

int Graph::GetVertPos(const int& vertex)

{

for (int i = 0; i < this->vertList.size(); ++i)

{

if (this->vertList[i] == vertex)

return i;

}

return -1;

}

bool Graph::IsEmpty()

{

return !(this->vertList.size());

}

bool Graph::IsFull()

{

return (vertList.size() == maxSize);

}

int Graph::GetAmountVerts()

{

return this->vertList.size();

}

int Graph::GetAmountEdges()

{

int amount = 0; // Обнуляем счетчик

if (!this->IsEmpty()) // Проверяем, что граф не пуст

{

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i)

{

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

{

if (this->adjMatrix[i][j] == 1) // Находим ребра

++amount; // Считаем количество ребер

}

}

return amount; // Возвращаем количество ребер

}

else

return 0; // Если граф пуст, возвращаем 0

}

int Graph::GetWeight(const int& vertex1, const int& vertex2)

{

if (!this->IsEmpty())

{

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

return adjMatrix[vertPos1][vertPos2];

}

return 0;

}

std::vector<int> Graph::GetNbrs(const int& vertex)

{

std::vector<int> nbrsList; // Создание списка соседей

int pos = this->GetVertPos(vertex); // Вычисление позиции vertex в матрице смежности

if (pos != (-1)) // Проверка, что vertex есть в матрице смежности

{

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i)

{

if (this->adjMatrix[pos][i] != 0) // Вычисление соседей

nbrsList.push\_back(this->vertList[i]); // Запись соседей в вектор

}

}

return nbrsList; // Возврат списка соседей

}

void Graph::InsertVertex(const int& vertex)

{

if (!this->IsFull())

{

this->vertList.push\_back(vertex);

}

else

{

std::cout << "Граф уже заполнен. Невозможно добавить новую вершину " << std::endl;

return;

}

}

void Graph::InsertEdge(const int& vertex1, const int& vertex2, int weight)

{

if (this->GetVertPos(vertex1) != (-1) && this->GetVertPos(vertex2) != (-1))

{

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

if (this->adjMatrix[vertPos1][vertPos2] != 0)

{

std::cout << "Ребро между этими вершинами уже существует" << std::endl;

return;

}

else

this->adjMatrix[vertPos1][vertPos2] = weight;

}

else

std::cout << "Обеих вершин (или одной из них) нет в графе " << std::endl;

}

void Graph::Print()

{

if (!this->IsEmpty())

{

std::cout << "Матрица смежности графа: " << std::endl;

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i)

{

std::cout << this->vertList[i] << "|\t";

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

{

std::cout /\*<< "\t"\*/ << this->adjMatrix[i][j] << "\t";

}

std::cout << std::endl;

}

}

else

std::cout << "Граф пуст " << std::endl;

}

void Graph::DFS(int startVertex, bool\* visitedVerts)

{

// Выводим сообщение о том, что вершина посещена

std::cout << "Вершина " << startVertex << " посещена" << std::endl;

// Отмечаем в массиве посещенных вершин, что вершина посещена

visitedVerts[this->GetVertPos(startVertex)] = true;

std::vector<int> neighbors = this -> GetNbrs(startVertex);

// Создаем вектор соседей

for (int i = 0, size = this -> GetNbrs(startVertex).size();

i < size; ++i)

{

if (visitedVerts[this -> GetVertPos(neighbors[i])] != true)

// В цикле проверяем, что соседи текущей вершины еще не посещены и вызываем функцию обхода графа в глубину

this->DFS(neighbors[i], visitedVerts);

}

}

void Graph::initWindow(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutCreateWindow("My Graph");

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

}

draw.h

#pragma once

void reshape(int w, int h);

void display();

void drawFillCircle(int x, int y, int R);

void drawLine(int x1, int y1, int x2, int y2);

void DrawString(void\* p\_font, int x, int y, int vertexNum);

draw.cpp

#include "draw.h"

#include "GL/glut.h"

#include <cmath>

#include <string>

void reshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, w, 0, h);

glMatrixMode(GL\_MODELVIEW);

glutPostRedisplay();

}

void display()

{

glClearColor(0.19, 0.84, 0.76, 1.0); //Фон

glClear(GL\_COLOR\_BUFFER\_BIT);

constexpr int x[7]{ 0, 111, 350, 630, 430, 340, 650}; //x - координаты окружностей

constexpr int y[7]{ 0, 370, 360, 440, 240, 490, 310}; //y - координаты окружностей

glLineWidth(2.0); //Толщина линии, связывающей окружности

drawLine(x[1], y[1], x[2], y[2]); //Координаты начала и конца линии. Берётся из массива.

DrawString(GLUT\_BITMAP\_HELVETICA\_18, (x[1] + x[2]) / 2, (y[1] + y[2]) / 2 + 15, 4);

drawLine(x[1], y[1], x[5], y[5]); //Координаты начала и конца линии. Берётся из массива.

DrawString(GLUT\_BITMAP\_HELVETICA\_18, (x[1] + x[5]) / 2, (y[1] + y[5]) / 2 + 15, 8);

drawLine(x[2], y[2], x[3], y[3]); //Координаты начала и конца линии. Берётся из массива.

DrawString(GLUT\_BITMAP\_HELVETICA\_18, (x[2] + x[3]) / 2, (y[2] + y[3]) / 2 + 15, 15);

drawLine(x[2], y[2], x[4], y[4]); //Координаты начала и конца линии. Берётся из массива.

DrawString(GLUT\_BITMAP\_HELVETICA\_18, (x[2] + x[4]) / 2 + 5, (y[2] + y[4]) / 2 + 15, 9);

drawLine(x[4], y[4], x[6], y[6]); //Координаты начала и конца линии. Берётся из массива.

DrawString(GLUT\_BITMAP\_HELVETICA\_18, (x[4] + x[6]) / 2, (y[4] + y[6]) / 2 + 20, 11);

drawLine(x[6], y[6], x[3], y[3]); //Координаты начала и конца линии. Берётся из массива.

DrawString(GLUT\_BITMAP\_HELVETICA\_18, (x[6] + x[3]) / 2 + 15, (y[6] + y[3]) / 2, 6);

drawLine(x[5], y[5], x[2], y[2]); //Координаты начала и конца линии. Берётся из массива.

DrawString(GLUT\_BITMAP\_HELVETICA\_18, (x[5] + x[2]) / 2 + 15, (y[5] + y[2]) / 2, 23);

drawLine(x[5], y[5], x[3], y[3]); //Координаты начала и конца линии. Берётся из массива.

DrawString(GLUT\_BITMAP\_HELVETICA\_18, (x[5] + x[3]) / 2 + 5, (y[5] + y[3]) / 2 + 15, 12);

constexpr int vertexNum[7]{ 0, 1, 2, 3, 4, 5, 6}; //Номер вершины

constexpr int R = 30; //Радиус окружности

constexpr int vertexCount = 6;

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

{

drawFillCircle(x[i], y[i], R); //Отрисовываем круги по кол-ву вершин.

DrawString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, x[i], y[i], vertexNum[i]); //Числа на вершинах

}

glEnd();

glutSwapBuffers();

}

void drawLine(int x1, int y1, int x2, int y2)

{

glBegin(GL\_LINES);

glColor3f(0.0, 0.0, 0.0);

glVertex2i(x1, y1);

glVertex2i(x2, y2);

glEnd();

}

void drawFillCircle(int x, int y, int R)

{

glColor3f(1.0, 1.0, 1.0); //Цвет круга

float x1{}, y1{};

glBegin(GL\_POLYGON); //Посекторная отрисовка круга

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

{

for (int t = 0; t <= 360; t++)

{

x1 = static\_cast<float> (i \* std::sin(t) + x);

y1 = static\_cast<float> (i \* std::cos(t) + y);

glVertex2f(x1, y1);

}

}

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_POINTS); //Отрисовка контура круга

for (int t = 0; t <= 360; t++)

{

x1 = static\_cast<float> (R \* std::sin(t) + x);

y1 = static\_cast<float> (R \* std::cos(t) + y);

glVertex2f(x1, y1);

}

glEnd();

}

void DrawString(void\* p\_font, int x, int y, int vertexNum)

{

std::string s = std::to\_string(vertexNum);

glRasterPos2i(x - 6, y - 6); //Выравнивание символа по центру окружности

for (const char& i : s)

{

glutBitmapCharacter(p\_font, i); //Шрифт и отрисовываемый символ

}

}

Main.cpp

#include "Graph.h"

#include <iostream>

#include<conio.h>

int main(int argc, char\*\* argv)

{

setlocale(LC\_ALL, "Russian");

Graph graph; // Cоздание графа, содержащего вершины с номерами целого типа

int amountVerts{}, amountEdges{}, vertex{}, sourceVertex{}, targetVertex{}, weight{};;

std::cout << "Введите количество вершин графа: ";

std::cin >> amountVerts; std::cout << std::endl; // Ввод количества ребер графа в переменную amountVerts

std::cout << "Введите количество ребер графа: ";

std::cin >> amountEdges; std::cout << std::endl; // Ввод количества ребер графа в переменную amountEdges

std::cout << "///////////////////////////////////////////////////////////////" << std::endl;

std::cout << "///////////////////////////////////////////////////////////////" << std::endl;

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

{

std::cout << "Вершина: ";

std::cin >> vertex; // Ввод вершины

int\* vertPtr = &vertex; // Запоминаем адрес вершины с помощью указателя

graph.InsertVertex(\*vertPtr);

// Передаем ссылку на вершину в функцию InsertVertex происходит вставка вершины в вектор вершин

std::cout << std::endl;

std::cout << "///////////////////////////////////////////////////////////////" << std::endl;

}

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

{

std::cout << "Исходная вершина: ";

std::cin >> sourceVertex;

std::cout << std::endl; // ввод исходной вершины

int\* sourceVertPtr = &sourceVertex; // Запоминаем адрес исходной вершины

std::cout << "Конечная вершина: ";

std::cin >> targetVertex; // Ввод вершины, до которой будет идти ребро от исходной вершины

std::cout << std::endl;

std::cout << "Вес ребра: ";

std::cin >> weight;

std::cout << "///////////////////////////////////////////////////////////////" << std::endl;

int\* targetVertPtr = &targetVertex; // Запоминаем адрес конечной вершины (до которой будет идти ребро от исходной вершины)

graph.InsertEdge(\*sourceVertPtr, \*targetVertPtr, weight);

// Вставка ребра между исходной и конечной вершинами, в функцию передаются ссылки на исходную и конечную вершины

}

std::cout << "///////////////////////////////////////////////////////////////" << std::endl;

std::cout << std::endl;

graph.Print(); // Печать матрицы смежности графа

std::cout << "Введите вершину, с которой начать обход: ";

std::cin >> vertex;

std::cout << std::endl; // Ввод начальной вершины, с которой начнется обход графа в глубину (в нашем случае это 0)

bool\* visitedVerts = new bool[20]{};

int\* vertPtr = &vertex; // Запоминаем адрес введенной вершины

// Вызываем функцию обхода графа в глубину, в функцию передаются ссылка на введенную вершину и вектор посещенных вершин

graph.DFS(\*vertPtr, visitedVerts);

delete[] visitedVerts;

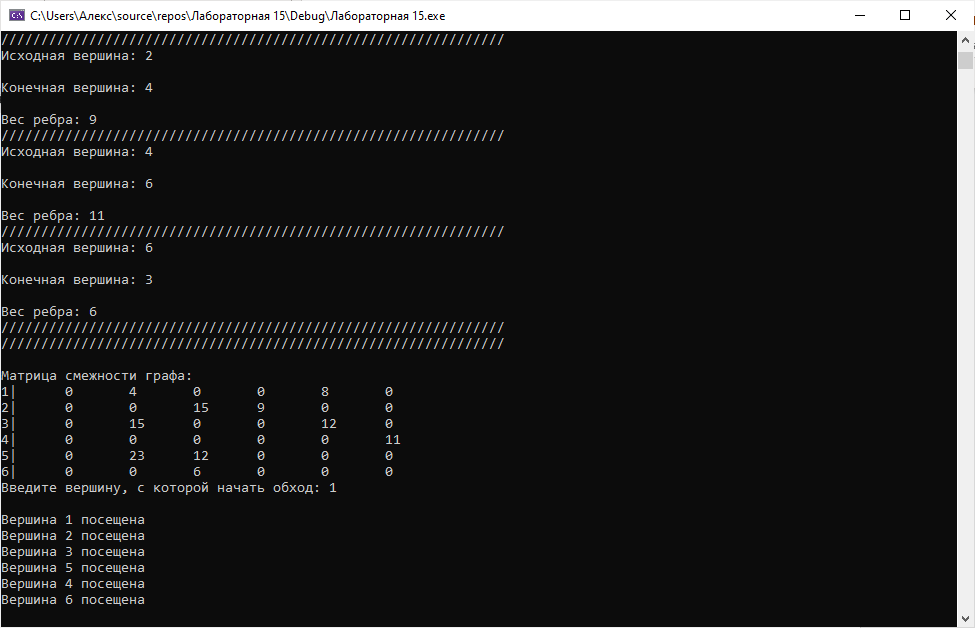
graph.initWindow(argc, argv);

return 0;

}

###### Тесты:

Создаётся граф из 6 вершин и производится обход в глубину (начиная с 1 вершины):



Печать графа в OpenGL:

