Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

Тема: Графы в С++

Выполнил студент ИВТ-22-2б:

Якушев Михаил Витальевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись)

Проверил доцент кафедры ИТАС:

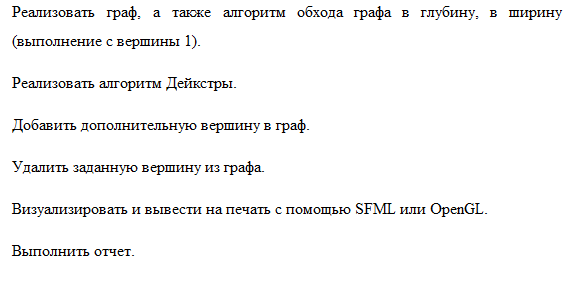
Полякова Ольга Андреевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись)

Пермь 2023

**1 Постановка задачи**

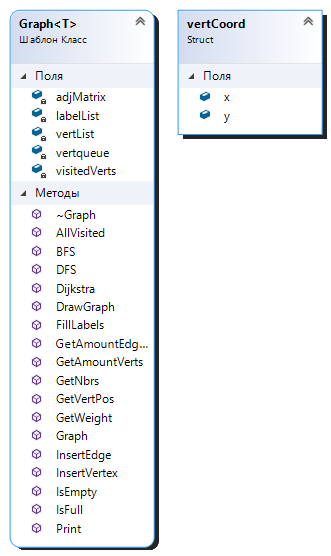
****

**2 Анализ задачи**

Создается класс graph, в котором реализованы методы для заполнения графа, добавления вершины, добавления ребра и тд. Так же есть методы для обхода в глубину, в ширину и алгоритм Дейкстры. Алгоритм обхода графа в глубину начинает выполнение с одной из вершин графа - начальной вершины, фиксирует информацию о посещении этой вершины, и, перемещаясь по ребру, посещает соседние вершины. Кроме того, правило левой руки при прохождении лабиринта (идти, ведя левой рукой по стенке) также является обходом в глубину. По завершении обхода все вершины окажутся пройденными - обработанными. Если при обходе встречается вершина, которая уже была пройдена, то повторной

обработки делать не нужно.

**3 UML-диаграммы**

****

**4 Код программы**

#include "graph.h"

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

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

{

setlocale(LC\_ALL, "rus");

glutInit(&argc, argv);

graph = makeGraph();

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(900, 720);

glutCreateWindow("Graph");

WinW = glutGet(GLUT\_WINDOW\_WIDTH);

WinH = glutGet(GLUT\_WINDOW\_HEIGHT);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}

#pragma once

#include <GL/glut.h>

#include <stdio.h>

#include <iostream>

#include <vector>

#include <queue>

#include <sstream>

using namespace std;

int WinW;

int WinH;

const int maxSize = 20;

template<class T>

class Graph

{

vector<T> vertList;

int adjMatrix[maxSize][maxSize] = { 0 };

vector<T> labelList;

queue<T> vertqueue;

bool\* visitedVerts = new bool[vertList.size()];

public:

Graph();

~Graph();

void DrawGraph();

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

void InsertVertex(const T& vertex);

int GetVertPos(const T& vertex);

bool IsEmpty();

bool IsFull();

int GetAmountVerts();

int GetAmountEdges();

int GetWeight(const T& vertex1, const T& vertex2);

vector<T> GetNbrs(const T& vertex);

void Print();

void DFS(T&, bool\*);

void BFS(T&, bool\*);

void FillLabels(T& startVertex);

int Dijkstra(T& startVertex);

bool AllVisited(bool\* visitedVerts);

};

int R;

struct vertCoord

{

int x, y;

};

vertCoord vertC[20];

Graph<int> graph;

template <class T>

void Graph<T>::DFS(T& startVertex, bool\* visitedVerts) {

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

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

vector<T> neighbors = this->GetNbrs(startVertex);

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

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

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

}

};

template <class T>

void Graph<T>::BFS(T& startvertex, bool\* visitedverts) {

if (visitedverts[this->GetVertPos(startvertex)] == false) {

this->vertqueue.push(startvertex);

cout << "Вершина " << startvertex << " обработана!" << endl;

visitedverts[this->GetVertPos(startvertex)] = true;

}

vector<T> neighbors = this->GetNbrs(startvertex);

this->vertqueue.pop();

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

if (!visitedverts[this->GetVertPos(neighbors[i])]) {

this->vertqueue.push(neighbors[i]);

visitedverts[this->GetVertPos(neighbors[i])] = true;

cout << "Вершина " << neighbors[i] << " обработана!" << endl;

}

}

if (this->vertqueue.empty())

return;

BFS(vertqueue.front(), visitedverts);

};

template<class T>

int Graph<T>::Dijkstra(T& startVertex)

{

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

visitedVerts[i] = false;

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

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

if (adjMatrix[i][j] < 0)

return -1;

if (GetVertPos(startVertex) == -1)

return -2;

T curSrc;

int counter = 0;

int minLabel = 1000000;

vector<T> neighbors = GetNbrs(startVertex);

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

{

int startLabel = labelList[GetVertPos(startVertex)];

int weight = GetWeight(startVertex, neighbors[i]);

int nIndex = GetVertPos(neighbors[i]);

int nextLabel = labelList[nIndex];

if (startLabel + weight < nextLabel)

labelList[nIndex] = startLabel + weight;

if (labelList[nIndex] < minLabel)

minLabel = labelList[nIndex];

}

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

if (labelList[GetVertPos(neighbors[i])] != 1000000)

counter += 1;

if (counter == neighbors.size())

visitedVerts[GetVertPos(startVertex)] = true;

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

if (labelList[GetVertPos(neighbors[i])] == minLabel)

curSrc = neighbors[i];

while (!AllVisited(visitedVerts))

{

neighbors = GetNbrs(curSrc);

int count = 0;

minLabel = 1000000;

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

{

int curLabel = labelList[GetVertPos(curSrc)];

int weight = GetWeight(curSrc, neighbors[i]);

int nIndex = GetVertPos(neighbors[i]);

int nextLabel = labelList[nIndex];

if (curLabel + weight < nextLabel)

labelList[nIndex] = curLabel + weight;

if (labelList[nIndex] < minLabel && visitedVerts[nIndex] != true)

minLabel = labelList[nIndex];

count += 1;

}

if (count == neighbors.size())

visitedVerts[GetVertPos(curSrc)] = true;

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

if (labelList[GetVertPos(neighbors[i])] == minLabel || visitedVerts[GetVertPos(neighbors[i])] != true)

curSrc = neighbors[i];

}

for (int i = 0; i < GetVertPos(startVertex); ++i)

{

cout << "Кратчайшее расстояние от вершины " << startVertex << " до вершины " << vertList[i] << " равно " << labelList[GetVertPos(vertList[i])] << endl;

}

for (int i = GetVertPos(startVertex) + 1; i < vertList.size(); ++i)

{

cout << "Кратчайшее расстояние от вершины " << startVertex << " до вершины " << vertList[i] << " равно " << labelList[GetVertPos(vertList[i])] << endl;

}

}

template <class T>

bool Graph<T>::AllVisited(bool\* visitedVerts)

{

bool flag = true;

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

if (visitedVerts[i] != true)

flag = false;

return flag;

}

template<class T>

void Graph<T>::FillLabels(T& startVertex)

{

for (int i = 0, size = vertList.size(); i < size; ++i)

{

labelList.push\_back(1000000);

}

int pos = GetVertPos(startVertex);

labelList[pos] = 0;

}

template<class T>

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

std::vector<T> nbrsList;

int vertPos = this->GetVertPos(vertex);

if (vertPos != (-1)) {

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

if (this->adjMatrix[vertPos][i] != 0 &&this->adjMatrix[i][vertPos] != 0)

nbrsList.push\_back(this->vertList[i]);

}

}

return nbrsList;

}

template<class T>

void Graph<T>::InsertVertex(const T& vertex) {

if (!this->IsFull()) {

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

}

else {

cout << "Невозможно добавить вершину! " << endl;

return;

}

}

template<class T>

int Graph<T>::GetAmountEdges() {

int amount = 0;

if (!this->IsEmpty()) {

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

{

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

{

if (this->adjMatrix[i][j] != 0 && this->adjMatrix[i][j] == this->adjMatrix[j][i])

{

amount++;

}

}

}

}

return amount/2;

}

template<class T>

int Graph<T>::GetWeight(const T& vertex1, const T& vertex2) {

if (!this->IsEmpty()) {

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

return adjMatrix[vertPos1][vertPos2];

}

return 0;

}

template<class T>

int Graph<T>::GetAmountVerts() {

return this->vertList.size();

}

template<class T>

bool Graph<T>::IsEmpty() {

if (this->vertList.size() != 0)

return false;

else

return true;

}

template<class T>

bool Graph<T>::IsFull() {

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

}

template <class T>

int Graph<T>::GetVertPos(const T& vertex) {

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

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

return i;

}

return -1;

}

template<class T>

Graph<T>::Graph() {

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

{

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

{

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

}

}

}

template<class T>

Graph<T>::~Graph() {

}

Graph<int> makeGraph()

{

bool\* visitedverts = new bool[20];

fill(visitedverts, visitedverts + 20, false);

Graph<int> graph;

int amountVerts, amountEdges, sourceVertex, targetVertex, edgeWeight,vertex;

cout << "Введите кол-во вершин графа: "; cin >> amountVerts; cout << endl;

cout << "Введите кол-во ребер графа: "; cin >> amountEdges; cout << endl;

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

int\* vertPtr = &i;

graph.InsertVertex(\*vertPtr);

}

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

cout << "Исходная вершина: "; cin >> sourceVertex; cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "Конечная вершина: "; cin >> targetVertex; cout << endl;

int\* targetVertPtr = &targetVertex;

cout << "Вес ребра: "; cin >> edgeWeight; cout << endl;

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

}

cout << endl;

cout << "Введите вершину с которой начать обход в глубину: "; cin >> vertex; cout << endl;

graph.DFS(vertex, visitedverts);

cout << "Введите вершину с которой начать обход в ширину: "; cin >> vertex; cout << endl;

fill(visitedverts, visitedverts + 20, false);

graph.BFS(vertex, visitedverts);

graph.Print();

return graph;

}

template<class T>

void Graph<T>::InsertEdge(const T& vertex1, const T& 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 && (this->adjMatrix[vertPos2][vertPos1] != 0))

{

cout << "Ребро между вершинами уже есть" << endl;

return;

}

else

{

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

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

}

}

else

{

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

return;

}

}

template<class T>

void Graph<T>::Print() {

if (!this->IsEmpty()) {

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

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

cout << this->vertList[i] << " ";

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

cout << " " << this->adjMatrix[i][j] << "\t";

}

cout << endl;

}

int startVertex;

cout << "С какой вершины обходить: ";

cin >> startVertex;

cout << endl;

FillLabels(startVertex);

Dijkstra(startVertex);

}

else {

cout << "Граф пустой " << endl;

}

}

void setCoord(int i, int n)

{

int R\_;

int x0 = WinW / 2;

int y0 = WinH / 2;

if (WinW > WinH)

{

R = 5 \* (WinH / 13) / n;

R\_ = WinH / 2 - R - 10;

}

else {

R = 5 \* (WinW / 13) / n;

R\_ = WinW / 2 - R - 10;

}

float theta = 2.0f \* 3.1415926f \* float(i) / float(n);

float y1 = R\_ \* cos(theta) + y0;

float x1 = R\_ \* sin(theta) + x0;

vertC[i].x = x1;

vertC[i].y = y1;

}

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

{

glColor3f(0.5, 1.0, 1.0);

float x1, y1;

glBegin(GL\_POLYGON);

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

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y1 = R \* cos(theta) + y;

x1 = R \* sin(theta) + x;;

glVertex2f(x1, y1);

}

glEnd();

glColor3f(0.0f, 0.0f, 0.0f);

float x2, y2;

glBegin(GL\_LINE\_LOOP);

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

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y2 = R \* cos(theta) + y;

x2 = R \* sin(theta) + x;

glVertex2f(x2, y2);

}

glEnd();

}

void drawText(int nom, int x1, int y1)

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = to\_string(nom);

glRasterPos2i(x1 - 5, y1 - 5);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void drawVertex(int n)

{

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

drawCircle(vertC[i].x, vertC[i].y, R);

drawText(i + 1, vertC[i].x, vertC[i].y);

}

}

void drawLine(int text, int x0, int y0, int x1, int y1)

{

glColor3f(0.0f, 0.0f, 0.0f);

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

drawText(text, (x0 + x1) / 2 + 10, (y0 + y1) / 2 + 10);

}

template<class T>

void Graph<T>::DrawGraph()

{

int n = vertList.size();

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

{

setCoord(i, n);

}

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

{

for (int j = i + 1; j < n; j++)

{

int a = adjMatrix[i][j];

if (a != 0)

{

drawLine(a, vertC[i].x, vertC[i].y, vertC[j].x, vertC[j].y);

}

}

}

drawVertex(n);

}

void reshape(int w, int h)

{

WinW = w;

WinH = h;

glViewport(0, 0, (GLsizei)WinW, (GLsizei)WinH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, (GLdouble)WinW, 0, (GLdouble)WinH);

glutPostRedisplay();

}

void display()

{

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WinW, 0, WinH);

glViewport(0, 0, WinW, WinH);

glClearColor(1.0, 0.1, 1.0, 1.0);

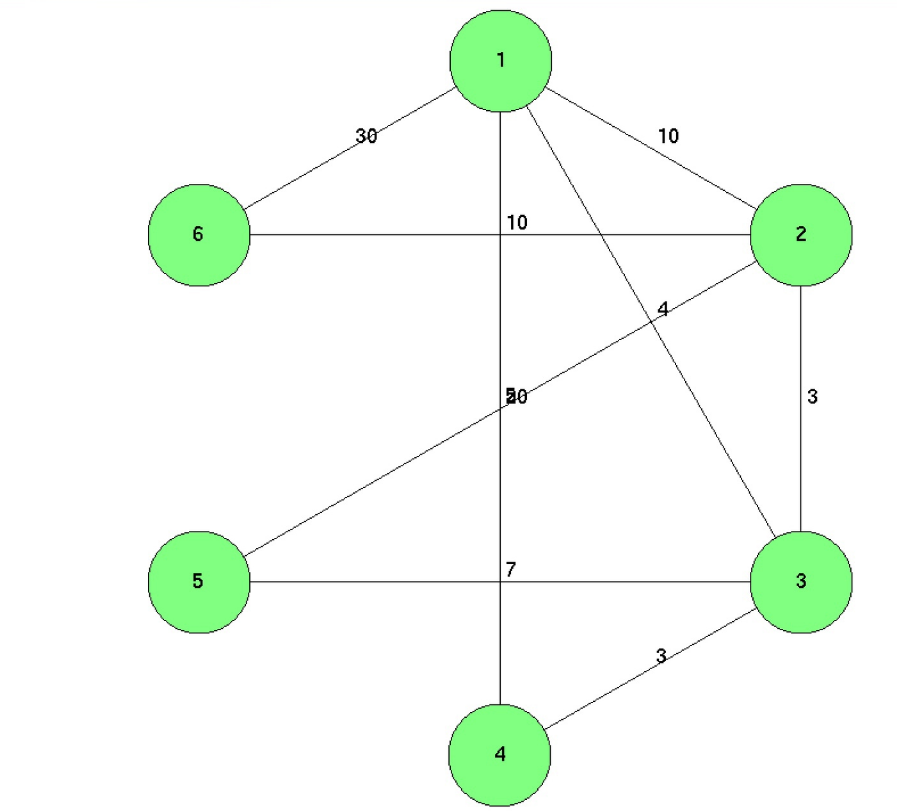
glClear(GL\_COLOR\_BUFFER\_BIT);

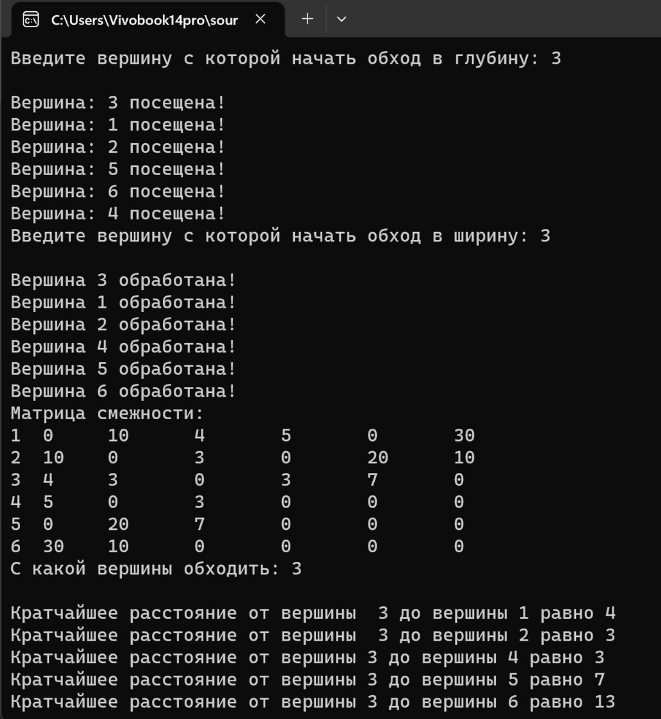
graph.DrawGraph();

glutSwapBuffers();

}

**5 Результаты работы программы**

****

****