Министерство высшего 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»

ОТЧЁТ

Тема: «Графы»

Выполнил

Студент группы РИС-22-2б

Чиртулов М.В.

Проверил доц. Кафедры ИТАС

Полякова Ольга Андреевна

Пермь 2023

**Постановка задачи**

Требуется реализовать работу с графом.  
1.3 Реализовать Алгоритмы на  
C++:  
1)Обход в ширину.  
2)Обход в глубину.  
3)Алгоритм Дейкстры.  
  
1.4 Требования:  
1) Пользовательский интерфейс - инструменты по решению разработчика (Windows Forms, например)  
2) Визуализация графа с использованием любой доступной графической библиотеки – SFML(Предпочтительно), SDL, OpenGL ( пример реализации представлен в файле: «Визуализация деревьев.docx»)  
3) Реализованные алгоритмы должны справляться не только с графом, представленным автором, а также применяться к другим  
Графам.  
4) Необходимы функции для редактирования графа: Создание(добавление) и удаление вершины и ребра. Редактирование весов ребер. Редактирование матрицы смежности (или инцидентности - в зависимости от реализации).

**Код.**

#pragma once

#include <glut.h>

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

#include <queue>

#include <stack>

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;

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

public:

Graph();

~Graph();

void DrawGraph();

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

void InputVert(const T& vertex);

int GetVertPos(const T& vertex);

bool Empty();

bool Full();

int GetVerts();

int GetEdges();

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

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

void Print();

void FillLabels(T& startVertex);

int Dijkstra(T& startVertex);

bool AllVisited(bool\* visitedVerts);

void BFS(T& startVertex);

void DFS(T& startVertex);

};

int R;

struct vertCoord

{

int x, y;

};

vertCoord vertC[20];

Graph<int> graph;

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)]; //текущая вершина

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

int nIndex = GetVertPos(neighbors[i]); //номер соседней вершины

int nextLabel = labelList[nIndex];

if (startLabel + weight < nextLabel) //проверка

labelList[nIndex] = startLabel + weight; //Меняем вершину

if (labelList[nIndex] < minLabel)

minLabel = labelList[nIndex]; //Наименьшая вершина

}

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) //Ищем новую вершину

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++) //Проверка соседних вершин

{

int curLabel = labelList[GetVertPos(curSrc)];

int weight = GetWeight(curSrc, neighbors[i]); //Вес соседнего ребра

int nIndex = GetVertPos(neighbors[i]); //Индекс соседней вершины

int nextLabel = labelList[nIndex];

if (curLabel + weight < nextLabel)

labelList[nIndex] = curLabel + weight; //Обновление информации о номере вершины

if (labelList[nIndex] < minLabel && visitedVerts[nIndex] != true) //Поиск минимального номера

minLabel = labelList[nIndex];

count += 1; //Считаем посещения

}

if (count == neighbors.size()) //Если все вершины посещены

visitedVerts[GetVertPos(curSrc)] = true; //Вершина стала обработаной

for (int i = 0; i < neighbors.size(); ++i) //Ищем новую опорную вершину

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

curSrc = neighbors[i];

}

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

{

cout << "Кратчайшая дорога от " << startVertex << " до " << vertList[i] << " будет " << labelList[GetVertPos(vertList[i])] << endl;

}

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

{

cout << "Кратчайшая дорога от" << startVertex << " до " << vertList[i] << " будет " << 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>::InputVert(const T& vertex) {

if (!this->Full()) {

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

}

else {

cout << "Ошибка! Граф заполнен! " << endl;

return;

}

}

template<class T>

int Graph<T>::GetEdges() {

int amount = 0; // обнуляем счетчик

if (!this->Empty()) { // Граф пуст?

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

i < vertListSize; ++i) {

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

if (this->adjMatrix[i][j] ==

this->adjMatrix[j][i] &&

this->adjMatrix[i][j] != 0) // ищем ребра

amount += 1; // ищем количество ребер

}

}

return (amount / 2);

}

else

return 0; //возвращаем 0 если граф пуст

}

template<class T>

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

if (!this->Empty()) {

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

return adjMatrix[vertPos1][vertPos2];

}

return 0;

}

template<class T>

int Graph<T>::GetVerts() {

return this->vertList.size();

}

template<class T>

bool Graph<T>::Empty() {

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

return false;

else

return true;

}

template<class T>

bool Graph<T>::Full() {

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()

{

Graph<int> graph; // создание графа с соответствующими вершинами

int amountVerts, amountEdges, sourceVertex, targetVertex, edgeWeight;

cout << "Количество вершин графа "; cin >> amountVerts; cout << endl; //фиксируем количество вершин

cout << "Количество ребер графа: "; cin >> amountEdges; cout << endl; // фиксируем количество ребер

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

int\* vertPtr = &i; // Запоминаем адрес вершины

graph.InputVert(\*vertPtr);

}

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

cout << "Начальная вершина: "; cin >> sourceVertex; cout << endl; //ввод исходной вершины

int\* sourceVertPtr = &sourceVertex;

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

int\* targetVertPtr = &targetVertex; // адрес конечной вершины

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

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

}

cout << endl;

graph.Print();//печать матрицы смежности

return graph;

}

template<class T>

void Graph<T>::InputEdge(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 << "Ошибка! Ребро уже существует" << endl;

return;

}

else {

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

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

}

}

else {

cout << "Ошибка!Нет одной или двух вершин " << endl;

return;

}

}

template<class T>

void Graph<T>::Print() {

if (!this->Empty()) {

cout << "Матрица смежности графа: " << 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] << " ";

}

cout << endl;

}

T startVertex;

cout << "Введите начальную вершину: ";

cin >> startVertex;

cout << endl;

FillLabels(startVertex);

//Dijkstra(startVertex);

std::cout << "Обход в ширину: " << std::endl;

BFS(startVertex);

std::cout << std::endl;

std::cout << "Обход в глубину: " << std::endl;

DFS(startVertex);

}

else {

cout << "Граф пуст! " << 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) //рисуем круг в заданных координатах

{

glColor3f(1.0, 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) //ребро неориентированный взвешенный граф

{

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); //ставим начало координат в левый нижний угол

glViewport(0, 0, WinW, WinH);

glClearColor(1.0, 1.0, 1.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

graph.DrawGraph();

glutSwapBuffers();

}

template<class T>

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

{

queue<T> q;

int svPos = GetVertPos(startVertex);

VertexColor colorList[100];

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

{

colorList[i] = WHITE;

}

colorList[svPos] = GRAY;

int count = 1;

cout << "BFS: ";

q.push(startVertex);

while (!q.empty())

{

T u = q.front();

q.pop();

vector<T> nbrsList = GetNbrs(u);

typename vector<T>::iterator i;

for (i = nbrsList.begin(); i != nbrsList.end(); ++i)

{

int pos = GetVertPos(\*i);

if (colorList[pos] == WHITE)

{

count++;

colorList[pos] = GRAY;

q.push(\*i);

}

}

colorList[GetVertPos(u)] = BLACK;

cout << u << " ";

}

cout << endl;

cout << "BFS count: " << count << endl;

}

// обход в ширину

template<class T>

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

{

// обнунялем посещенные

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

visitedVerts[i] = false;

queue<T> q;

q.push(startVertex); //добавляем стартовую вершину

visitedVerts[GetVertPos(startVertex)] = true; //помечаем ее как посещенную

while (!q.empty())

{

T curVertex = q.front(); //берем первую вершину в очереди

q.pop(); //убираем ее из очереди

cout << curVertex << " "; //выводим ее на экран

vector<T> neighbors = GetNbrs(curVertex); //ищем соседей

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

{

if (visitedVerts[GetVertPos(neighbors[i])] != true) //если соседняя вершина не была посещена

{

q.push(neighbors[i]); //добавляем ее в очередь

visitedVerts[GetVertPos(neighbors[i])] = true; //помечаем ее как посещенную

}

}

}

}

//обход в глубину

template<class T>

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

{

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

visitedVerts[i] = false;

stack<T> s;

s.push(startVertex); //добавляем стартовую вершину

visitedVerts[GetVertPos(startVertex)] = true; //помечаем ее как посещенную

while (!s.empty())

{

T curVertex = s.top(); //берем последнюю вершину в стэке

s.pop(); //убираем ее из стэка

cout << curVertex << " "; //выводим ее на экран

vector<T> neighbors = GetNbrs(curVertex); //ищем соседей

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

{

if (visitedVerts[GetVertPos(neighbors[i])] != true) //если соседняя вершина не была посещена

{

s.push(neighbors[i]); //добавляем ее в очередь

visitedVerts[GetVertPos(neighbors[i])] = true; //помечаем ее как посещенную

}

}

}

}

float WinWid = 800, WinHei = 800;

float angle = 0.0, scale = 1.0;

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

setlocale(0, "Rus");

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(WinWid, WinHei);

glutInitWindowPosition(400, 400);

glutCreateWindow("WebWindow");

graph = makeGraph();

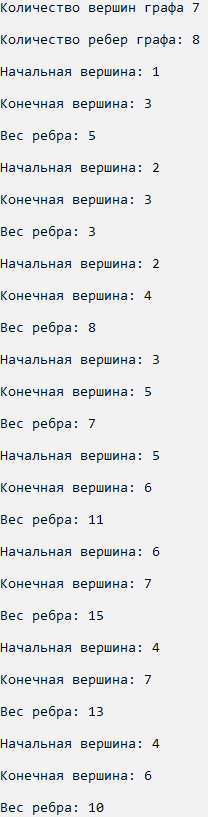
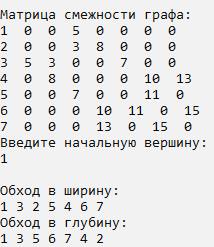
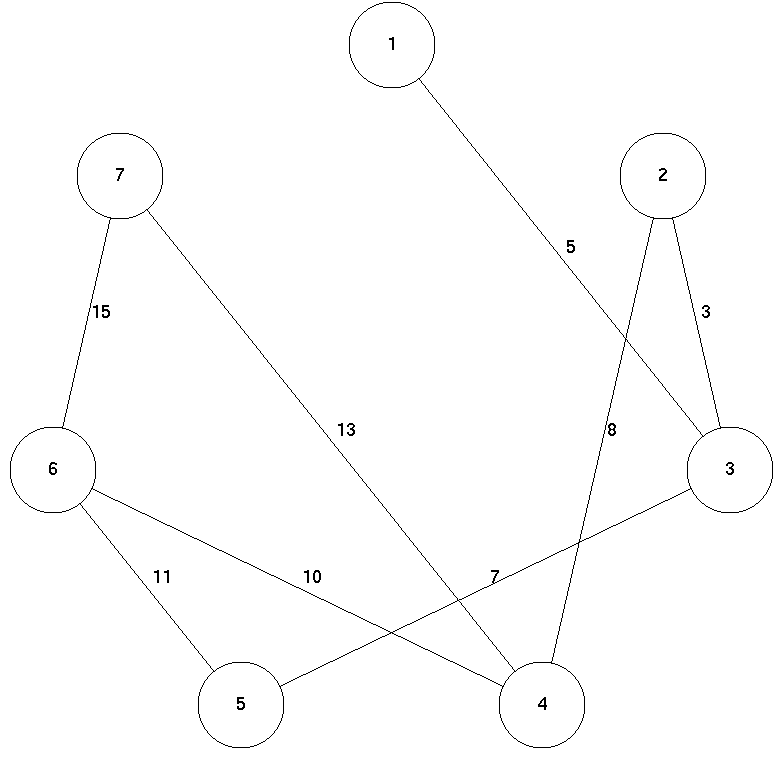
glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

}

**Результаты выполнения программы.**



**UML-диаграмма**

