Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Графы»

Выполнил работу

Студент группы РИС-23-3Б

Шуракова А.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2024

**Постановка задачи:**

Реализовать алгоритмы для собственного, придуманного самим автором, варианта двунаправленного графа, имеющего не менее 6 вершин.

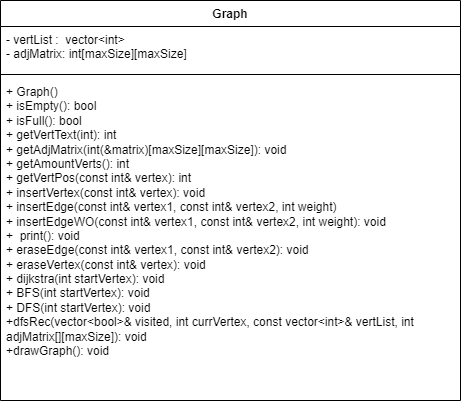
Алгоритмы:

1. Обход в ширину.

2. Обход в глубину.

3. Алгоритм Дейкстры.

**Uml-диаграмма:**

****

**Код программы:**

**graph.h**

#pragma once

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

#include "freeglut.h"

#include <string>

#include <thread>

using namespace std;

const int maxSize = 8;

class Graph

{

private:

vector<int> vertList; //список вершин графа

int adjMatrix[maxSize][maxSize]; //матрица смежности для представления ребер

public:

Graph();

bool isEmpty(); //проверка на пустоту графа

bool isFull(); //проверка на заполненность

int getVertText(int i); //получение номера вершины

void getAdjMatrix(int(&matrix)[maxSize][maxSize]);

int getAmountVerts(); //получение количества вершин

int getVertPos(const int& vertex); //получение позиции вершины

void insertVertex(const int& vertex); //добавление вершины

void insertEdge(const int& vertex1, const int& vertex2, int weight); //добавление направленного ребра

void insertEdgeWO(const int& vertex1, const int& vertex2, int weight); //добавление ненаправленного ребра

void print(); //печать матрицы

void eraseEdge(const int& vertex1, const int& vertex2); // удаление ребра

void eraseVertex(const int& vertex); //удаление вершины

void dijkstra(int startVertex); // дейкстра

void BFS(int startVertex); // обход в ширину

void DFS(int startVertex); // обход в глубину

void dfsRec(vector<bool>& visited, int currVertex, const vector<int>& vertList, int adjMatrix[][maxSize]); //рекурсия для обхода в глубину

void drawGraph();

};

void drawCircle(int x, int y, int R); //рисует круг на экране

void display(); //Функция отображения для графического интерфейса

void drawVertex(int n); //Рисует вершину графа

void setCoords(int i, int n); //Задает координаты вершины

void drawLine(int text, int x0, int y0, int x1, int y1, int start, int end, bool f); //Рисует линию между двумя вершинами графа

void drawText(int text, int x1, int y1, bool f); //Отображает текст на экране

void menu(); //Отображает меню

void makegraph(); //Создает и отображает граф на экране

**graph.cpp**

#include"graph.h"

#include "iostream"

struct vertCoord {

int x, y; //координаты вершины

};

int WinW = 800; //ширина графического окна

int WinH = 800; //высота графического окна

vertCoord vertC[maxSize]; //массив координат вершин

Graph graph;

int R = 50; //радиус вершин при рисовании

using namespace std;

//Инициализирует матрицу смежности нулями

Graph::Graph() {

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

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

adjMatrix[i][j] = 0;

}

//Проверяет, пуст ли граф, проверяя, есть ли вершины в списке вершин

bool Graph::isEmpty() {

return (vertList.size() == 0);

}

//Проверяет, заполнен ли граф, проверяя, достигло ли количество вершин максимального размера

bool Graph::isFull() {

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

}

//Возвращает номер вершины по ее индексу в списке вершин

int Graph::getVertText(int i) {

return vertList[i];

}

//Копирует матрицу смежности во внешнюю матрицу, переданную по ссылке

void Graph::getAdjMatrix(int(&matrix)[maxSize][maxSize])

{

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

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

matrix[i][j] = adjMatrix[i][j];

}

}

}

//Метод возвращает количество вершин в графе, т.е. размер вектора вершин

//Если граф пуст, то метод вернет 0.

int Graph::getAmountVerts() {

return vertList.size();

}

//Ищет позицию вершины в списке вершин

int Graph::getVertPos(const int& vertex) {

//Проходит по списку вершин и возвращает индекс вершины, соответствующей переданному значению вершины.

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

if (vertList[i] == vertex)

return i;

//Если вершина не найдена, возвращает - 1

return -1;

}

//Добавляет новую вершину в граф

void Graph::insertVertex(const int& vertex) {

//Проверяет, заполнен ли граф, и выводит сообщение, если он заполнен

if (isFull()) {

cout << "Граф уже заполнен. Невозможнодобавить новую вершину" << endl;

return;

}

vertList.push\_back(vertex); //Добавляет вершину в список вершин

}

//Добавляет направленное ребро в граф

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

//Проверяет, допустимо ли значение веса ребра и существуют ли обе вершины

if (weight < 1) {

cout << "Нельзя ввести значение меньше 1! " << endl;

return;

}

if (getVertPos(vertex1) == -1 || getVertPos(vertex2) == -1) {

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

return;

}

//Добавляет ребро в матрицу смежности, используя позиции вершин в списке вершин

int vertPos1 = getVertPos(vertex1);

int vertPos2 = getVertPos(vertex2);

if (adjMatrix[vertPos1][vertPos2] != 0) {

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

return;

}

adjMatrix[vertPos1][vertPos2] = weight;

}

//Добавляет ненаправленное ребро в граф

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

if (weight < 1) {

cout << "Нельзя ввести значение меньше 1! " << endl;

return;

}

if (getVertPos(vertex1) == -1 || getVertPos(vertex2) == -1) {

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

return;

}

//Добавляет ребро в матрицу смежности

int vertPos1 = getVertPos(vertex1);

int vertPos2 = getVertPos(vertex2);

if (adjMatrix[vertPos1][vertPos2] != 0) {

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

return;

}

adjMatrix[vertPos1][vertPos2] = weight;

adjMatrix[vertPos2][vertPos1] = weight;

}

//Печатает матрицу смежности графа

void Graph::print() {

//Проверяет, пуст ли граф, и выводит сообщение, если это так

if (isEmpty()) {

cout << "Граф пуст " << endl;

return;

}

cout << " ";

//Печатает заголовки столбцов, соответствующие номерам вершин

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

cout << vertList[i] << " ";

cout << endl;

//Печатает каждую строку матрицы смежности, соответствующую вершине, с ее номером вершины в качестве заголовка строки

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

cout << vertList[i] << "| ";

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

cout << adjMatrix[i][j] << " ";

cout << endl;

}

}

void Graph::eraseEdge(const int& vertex1, const int& vertex2) {

if (getVertPos(vertex1) == -1 || getVertPos(vertex2) == -1) {

cout << "Одной или обоих вершин не существует " << endl;

return;

}

int vertPos1 = getVertPos(vertex1);

int vertPos2 = getVertPos(vertex2);

if (adjMatrix[vertPos1][vertPos2] == 0) {

cout << " Ребра и так нету" << endl;

return;

}

adjMatrix[vertPos1][vertPos2] = 0;

}

void Graph::eraseVertex(const int& vertex)

{

auto it = find(vertList.begin(), vertList.end(), vertex);

if (it == vertList.end()) {

cout << "Вершина не найдена" << endl;

return;

}

int index = distance(vertList.begin(), it);

vertList.erase(it);

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

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

adjMatrix[i][j] = adjMatrix[i + 1][j];

adjMatrix[j][i] = adjMatrix[j][i + 1];

}

}

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

adjMatrix[vertList.size()][i] = 0;

adjMatrix[i][vertList.size()] = 0;

}

}

void Graph::dijkstra(int startVertex)

{

int n = vertList.size();

vector<int> distance(n, INT\_MAX);

vector<bool> visited(n, false);

distance[startVertex] = 0;

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

pq.push({ 0, startVertex });

while (!pq.empty()) {

int currVertex = pq.top().second;

pq.pop();

if (visited[currVertex])

continue;

visited[currVertex] = true;

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

if (adjMatrix[currVertex][i] != 0) {

int nextVertex = i;

int weight = adjMatrix[currVertex][i];

if (!visited[nextVertex] && distance[currVertex] != INT\_MAX && distance[currVertex] + weight < distance[nextVertex]) {

distance[nextVertex] = distance[currVertex] + weight;

pq.push({ distance[nextVertex], nextVertex });

}

}

}

}

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

cout << "Самый короткий путь из " << vertList[startVertex] << " до " << vertList[i] << ": ";

if (distance[i] == INT\_MAX)

cout << " пути нет ";

else

cout << distance[i];

cout << endl;

}

}

void Graph::BFS(int startVertex)

{

if (isEmpty()) {

cout << "Граф пуст" << endl;

return;

}

int n = vertList.size();

vector<bool> visited(n, false);

queue<int> q;

visited[startVertex] = true;

q.push(startVertex);

while (!q.empty()) {

int currVertex = q.front();

q.pop();

cout << this->getVertText(currVertex) << " ";

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

if (adjMatrix[currVertex][i] != 0 && !visited[i]) {

visited[i] = true;

q.push(i);

}

}

}

cout << endl;

}

void Graph::DFS(int startVertex) {

if (isEmpty()) {

cout << "Граф пуст" << endl;

return;

}

int n = vertList.size();

vector<bool> visited(n, false);

cout << "Обход в глубину: ";

dfsRec(visited, startVertex, vertList, adjMatrix);

cout << endl;

}

void Graph::dfsRec(vector<bool>& visited, int currVertex, const vector<int>& vertList, int adjMatrix[][maxSize]) {

visited[currVertex] = true;

cout << this->getVertText(currVertex) << " ";

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

if (adjMatrix[currVertex][i] != 0 && !visited[i]) {

dfsRec(visited, i, vertList, adjMatrix);

}

}

}

void Graph::drawGraph() {

int n = getAmountVerts();

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

setCoords(i, n);

}

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

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

int a = adjMatrix[i][j];

if (a == 0)

continue;

bool isPath = false;

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

}

}

drawVertex(n);

glutPostRedisplay();

}

void makegraph()

{

int numVertices, numEdges;

int vertexName;

// Ввод количества вершин

cout << "Введите количество вершин графа: ";

cin >> numVertices;

// Добавление вершин

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

cout << "Введите имя вершины " << i + 1 << ": ";

cin >> vertexName;

graph.insertVertex(vertexName);

}

// Ввод количества ребер

cout << "Введите количество ребер графа: ";

cin >> numEdges;

// Добавление ребер

int startVertex, endVertex;

int edgeWeight;

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

cout << "Введите начальную вершину ребра " << i + 1 << ": ";

cin >> startVertex;

cout << "Введите конечную вершину ребра " << i + 1 << ": ";

cin >> endVertex;

cout << "Введите вес ребра " << i + 1 << ": ";

cin >> edgeWeight;

graph.insertEdgeWO(startVertex, endVertex, edgeWeight);

}

}

void menu()

{

while (true)

{

cout << "1. Обход в ширину " << endl; //BFS

cout << "2. Обход в глубину " << endl; //DFS

cout << "3. Алгоритм Дейкстры " << endl; // dijkstra

cout << "5. Добавить вершину " << endl; //insertVertex

cout << "6. Удалить вершину " << endl;//eraseVertex

cout << "7. Печать матрицы " << endl; //print

cout << "8. Добавить направленное ребро " << endl; //insertEdge

cout << "9. Удалить ребро " << endl; //eraseEdge

int n, temp, temp1, temp2;

bool f = true;

cin >> n;

switch (n)

{

case 1:

graph.print();

cout << " Введите стартовую вершину " << endl; cin >> temp2;

temp = graph.getVertPos(temp2);

if (temp != -1)

graph.BFS(temp);

break;

case 2:

graph.print();

cout << " Введите стартовую вершину " << endl; cin >> temp2;

temp = graph.getVertPos(temp2);

if (temp != -1)

graph.DFS(temp);

break;

case 3:

graph.print();

cout << " Введите стартовую вершину " << endl; cin >> temp2;

temp = graph.getVertPos(temp2);

if (temp != -1)

graph.dijkstra(temp);

break;

case 5:

graph.print();

while (f)

{

cout << "Введите номер вершины " << endl; cin >> temp;

if (graph.getVertPos(temp) != -1) cout << " уже есть вершина с таким номером " << endl;

else f = false;

}

graph.insertVertex(temp);

f = true;

break;

case 6:

graph.print();

cout << "Введите номер вершины " << endl; cin >> temp;

graph.eraseVertex(temp);

break;

case 7:

graph.print();

break;

case 8:

graph.print();

cout << "Введите первую вершину "; cin >> temp;

cout << "Введите вторую вершину "; cin >> temp1;

cout << "Введите вес ребра "; cin >> temp2;

graph.insertEdge(temp, temp1, temp2);

break;

case 9:

graph.print();

cout << "Введите первую вершину "; cin >> temp;

cout << "Введите вторую вершину "; cin >> temp1;

graph.eraseEdge(temp, temp1);

break;

default:

cout << "Неверный ввод" << endl;

break;

}

system("pause");

system("cls");

}

}

void drawText(int text, int x1, int y1, bool f) {

if (f) glColor3f(0.0f, 0.0f, 1.0f);//синий для веса ребра

else glColor3f(0.0f, 0.0f, 0.0f);//черный для номера вершины

GLvoid\* font = GLUT\_BITMAP\_TIMES\_ROMAN\_24; // Выбор шрифта для отображения текста

string s = to\_string(text); // Преобразование числа в строку

glRasterPos2i(x1 - 5, y1 - 5);// Установка позиции для рендеринга текста

for (size\_t j = 0; j < s.length(); j++) // Цикл для рендеринга каждого символа строки

glutBitmapCharacter(font, s[j]); // Рендеринг отдельного символа строки

}

void drawLine(int text, int x0, int y0, int x1, int y1, int start, int end, bool f) {

if (f) glColor3f(1.0, 0.0, 0.0);

else glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES); // Начало рисования линии

glVertex2i(x0, y0); // Первая вершина линии

glVertex2i(x1, y1); // Вторая вершина линии

glEnd(); // Конец рисования линии

}

//функция вычисляет координаты вершин графа, расположенных по окружности

void setCoords(int i, int n) {

int R\_;

int x0 = WinW / 2;

int y0 = WinH / 2;

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

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

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

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

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

vertC[i].x = x1;

vertC[i].y = y1;

}

void drawVertex(int n) {

// Рисование окружности для вершины

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

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

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

// Рисование номера вершины

drawText(graph.getVertText(i), vertC[i].x, vertC[i].y, false);

}

}

//функция отвечает за отображение графа на экране

void display() {

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WinW, 0, WinH);

glViewport(0, 0, WinW, WinH);

glClear(GL\_COLOR\_BUFFER\_BIT);

graph.drawGraph();

glutSwapBuffers();

}

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

glColor3f(1.0f, 1.0f, 1.0f);

float x1, y1;

glBegin(GL\_POLYGON); // Начинаем рисование многоугольника (окружности).

for (int i = 0; i < 360; i++) { // Итерируемся по углам от 0 до 360 градусов.

float theta = 2.0f \* 3.1415926f \* float(i) / float(360); // Вычисляем угол в радианах.

x1 = R \* sin(theta) + x; // Вычисляем координату x текущей точки окружности.

y1 = R \* cos(theta) + y; // Вычисляем координату y текущей точки окружности.

glVertex2f(x1, y1); // Устанавливаем точку в текущих координатах.

}

glEnd(); // Завершаем рисование многоугольника (окружности).

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

float x2, y2;

glBegin(GL\_LINE\_LOOP); // Начинаем рисование линии контура окружности.

for (int i = 0; i < 360; i++) { // Итерируемся по углам от 0 до 360 градусов.

float theta = 2.0f \* 3.1415926f \* float(i) / float(360); // Вычисляем угол в радианах.

x2 = R \* sin(theta) + x; // Вычисляем координату x текущей точки контура окружности.

y2 = R \* cos(theta) + y; // Вычисляем координату y текущей точки контура окружности.

glVertex2f(x2, y2); // Устанавливаем точку в текущих координатах.

}

glEnd(); // Завершаем рисование линии контура окружности.

}

**main.cpp**

// graph.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include "graph.h"

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

//argc (argument count) - количество аргументов командной строки

//argv (argument vector) - массив указателей на строки, содержащие аргументы командной строки.

{

system("chcp 1251");

makegraph(); //создание графа

cout << endl;

thread console(menu);// Запуск потока для работы с консолью

// Ввод количества ребер графа в переменную amountVerts

glutInit(&argc, argv); //начальная инициализация окна

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

//установка режима отображения:

// GLUT\_DEPTH включает Z-буфер (буфер глубины)

// GLUT\_DOUBLE включает двойной экранный буфер

// GLUT\_RGBA включает четырехкомпонентный цвет (используется по умолчанию)

glutInitWindowSize(800, 800); //размер окна

glutCreateWindow("Графы"); //заголовок окна

glClearColor(0.0f, 1.0f, 1.0f, 1.0f); //цвет фона

glutDisplayFunc(display); //функция которой мы передаем имя функции для отрисовки окна

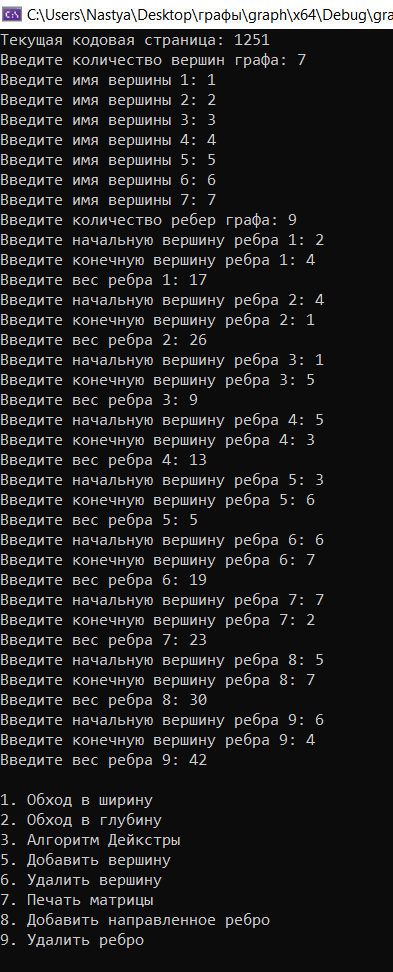
glutMainLoop(); //запускаем всё проинициализированное, проще говоря та же ф-я main

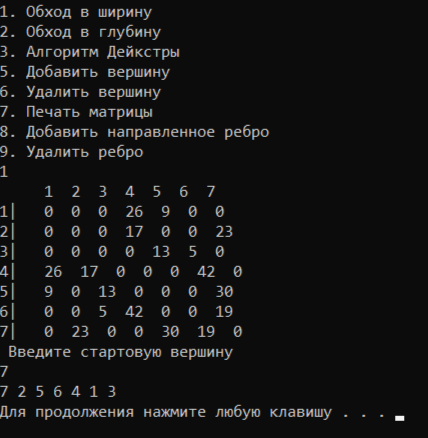
console.join(); //Основной поток ожидает завершения потока console, созданного ранее.

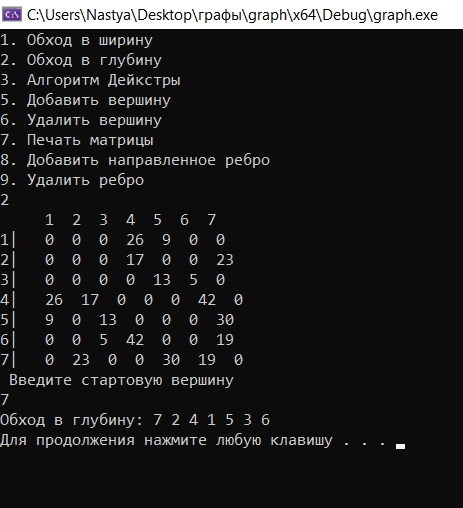
return 0;

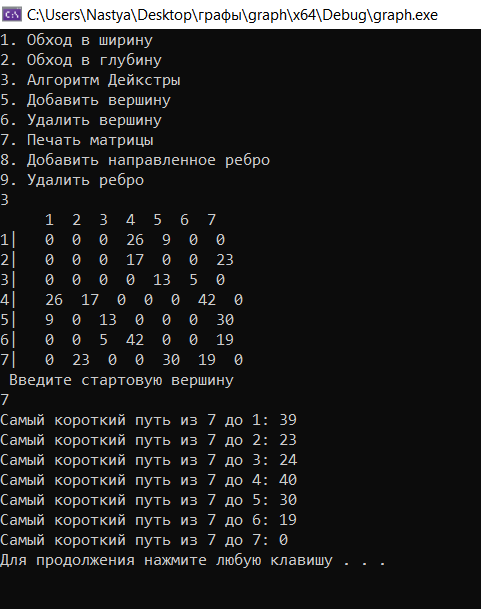
}

**Результат работы программы:**

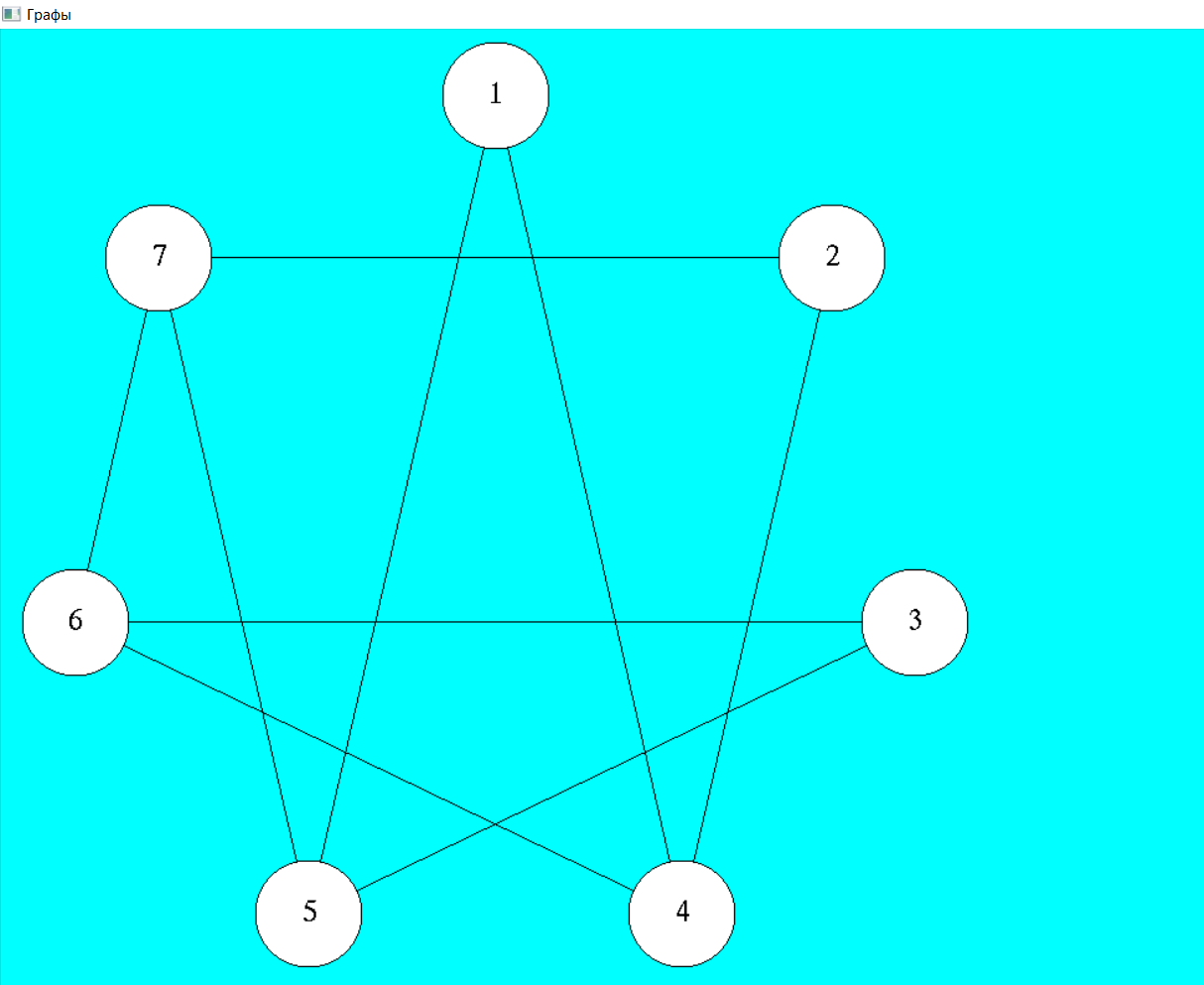
****

****

****

****

**Результат работы программы в OpenJL:**

****

**Ссылка на Git:**

https://github.com/Ananasic07/labs\_2sem/tree/main/graph