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: «Информатика»

Тема: "Графы"

Семестр 2

Выполнил работу

Студент группы РИС-24-3Б

Гузий С.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2025

# Постановка задачи

Реализовать следующие алгоритмы для собственного варианта графа, имеющего не менее 6 вершин:

1. Обход в ширину;
2. Обход в глубину;
3. Алгоритм Дейкстры.

Требования:

1. Пользовательский интерфейс на усмотрение разработчика с условием кроссплатформенности;
2. Визуализация графа с использованием любой доступной графической библиотеки;
3. Реализованные алгоритмы должны справляться как с графом, представленным в задании варианта, так и с другими на усмотрение проверяющего;
4. Необходимо реализовать функции для редактирования графа:

* Создание новой вершины;
* Удаление вершины;
* Добавление и удаление ребра;
* Редактирование весов ребер;

# Код программы

#include <GL/glut.h>

#include <iostream>

#include <vector>

#include <climits>

#include <locale>

#include <string>

#include <algorithm>

const int N = 6;

int graph[N][N]; // Матрица смежности графа

float coords[N][2]; // Координаты вершин для визуализации

// Матрицы для алгоритма Флойда

int dist[N][N]; // Матрица кратчайших расстояний

int next[N][N]; // Матрица для восстановления пути

void initializeGraph() {

// Инициализация графа

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

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

graph[i][j] = (i == j) ? 0 : INT\_MAX;

}

}

// Задаем ребра и их веса согласно уточненным данным

// Вершины нумеруются от 0 до 5 (соответствуют 1-6)

graph[0][1] = 14; // 1->2 = 14

graph[1][4] = 5; // 2->5 = 5

graph[4][2] = 18; // 5->3 = 18

graph[2][3] = 9; // 3->4 = 9

graph[3][5] = 31; // 4->6 = 31

graph[5][0] = 28; // 6->1 = 28

graph[5][1] = 23; // 6->2 = 23

graph[1][5] = 23; // 2->6 = 23 (двунаправленное ребро)

// Инициализация координат вершин для визуализации

float angle = 0.0f;

float angleStep = 2 \* 3.1415926f / N;

float radius = 200.0f;

float centerX = 300.0f, centerY = 300.0f;

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

coords[i][0] = centerX + radius \* cos(angle);

coords[i][1] = centerY + radius \* sin(angle);

angle += angleStep;

}

}

void floydWarshall() {

// Инициализация матриц dist и next

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

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

dist[i][j] = graph[i][j];

if (graph[i][j] != INT\_MAX && i != j) {

next[i][j] = j;

}

else {

next[i][j] = -1;

}

}

}

// Алгоритм Флойда

for (int k = 0; k < N; ++k) {

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

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

if (dist[i][k] != INT\_MAX && dist[k][j] != INT\_MAX &&

dist[i][j] > dist[i][k] + dist[k][j]) {

dist[i][j] = dist[i][k] + dist[k][j];

next[i][j] = next[i][k];

}

}

}

}

}

void printShortestPaths(int start) {

std::cout << "Кратчайшие пути из вершины " << start + 1 << ":\n";

for (int end = 0; end < N; ++end) {

if (start == end) continue;

if (dist[start][end] == INT\_MAX) {

std::cout << "Пути из " << start + 1 << " в " << end + 1 << " не существует\n";

continue;

}

std::cout << "Вершина " << end + 1 << ": расстояние = " << dist[start][end]

<< ", путь: " << start + 1;

int current = start;

while (current != end) {

current = next[current][end];

std::cout << " -> " << current + 1;

}

std::cout << "\n";

}

}

void drawText(float x, float y, const std::string& text, float r = 0, float g = 0, float b = 0) {

glColor3f(r, g, b);

glRasterPos2f(x, y);

for (char c : text)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, c);

}

void drawCircle(float cx, float cy, float r) {

glBegin(GL\_TRIANGLE\_FAN);

for (int i = 0; i < 100; ++i) {

float theta = 2.0f \* 3.1415926f \* i / 100;

glVertex2f(cx + r \* cos(theta), cy + r \* sin(theta));

}

glEnd();

}

void drawLine(float x1, float y1, float x2, float y2, bool highlight, int weight = -1, bool drawWeight = true) {

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

else glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

if (weight >= 0 && drawWeight) {

float midX = (x1 + x2) / 2;

float midY = (y1 + y2) / 2;

// Вектор нормали (перпендикуляр к линии) для смещения надписи

float dx = x2 - x1;

float dy = y2 - y1;

float length = sqrt(dx \* dx + dy \* dy);

float offsetX = -dy / length \* 10;

float offsetY = dx / length \* 10;

drawText(midX + offsetX, midY + offsetY, std::to\_string(weight));

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

// Рисуем рёбра

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

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

if (graph[i][j] != INT\_MAX) {

bool isBidirectional = (graph[j][i] != INT\_MAX && i != j);

// Для двунаправленных ребер рисуем вес только один раз

if (isBidirectional && i < j) {

drawLine(coords[i][0], coords[i][1], coords[j][0], coords[j][1], false, graph[i][j], true);

}

// Для однонаправленных рисуем всегда

else if (!isBidirectional) {

drawLine(coords[i][0], coords[i][1], coords[j][0], coords[j][1], false, graph[i][j], true);

}

}

}

}

// Вершины

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

glColor3f(0.2, 0.6, 0.9);

drawCircle(coords[i][0], coords[i][1], 20);

glColor3f(1.0, 1.0, 1.0);

drawText(coords[i][0] - 5, coords[i][1] - 5, std::to\_string(i + 1));

}

// Вывод информации о графе

drawText(20, 20, "Граф по уточненным данным", 0.0f, 0.0f, 1.0f);

drawText(20, 40, "Алгоритм Флойда-Уоршелла", 0.0f, 0.0f, 1.0f);

drawText(20, 60, "Начинаем с вершины 3", 0.0f, 0.0f, 1.0f);

glutSwapBuffers();

}

void init() {

glClearColor(1, 1, 1, 1);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 600, 0, 600);

initializeGraph();

floydWarshall();

printShortestPaths(2); // Вершина 3 соответствует индексу 2

}

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

setlocale(LC\_ALL, "Russian");

std::cout << "Графы. Уточненный вариант\n";

std::cout << "Алгоритм Флойда-Уоршелла для нахождения кратчайших путей\n";

std::cout << "Список ребер:\n";

std::cout << "1->2 = 14\n";

std::cout << "2->5 = 5\n";

std::cout << "5->3 = 18\n";

std::cout << "3->4 = 9\n";

std::cout << "4->6 = 31\n";

std::cout << "6->1 = 28\n";

std::cout << "6<->2 = 23\n";

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(600, 600);

glutCreateWindow("Граф - Алгоритм Флойда (уточненный)");

init();

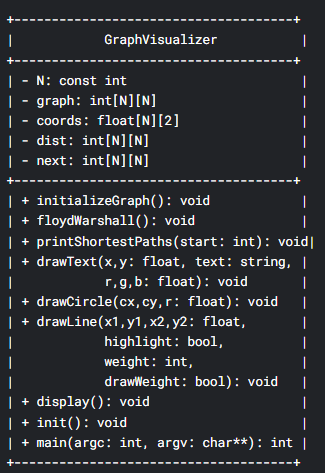
glutDisplayFunc(display);

glutMainLoop();

return 0;

}

UML – диаграмма



# Визуализация графа

