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-22-1Б

Аксёнов Д.О.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

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

Алгоритмы:

1. Обход в ширину.

2. Обход в глубину.

3. Алгоритм Дейкстры.

Требования:

1. Пользовательский интерфейс на усмотрение разработчика с условием кроссплатформенности (поощряется использование Qt или иных фреймворков)

2. Визуализация графа с использованием любой доступной графической библиотеки (SFML, SDL, OpenGL и подобных)

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.

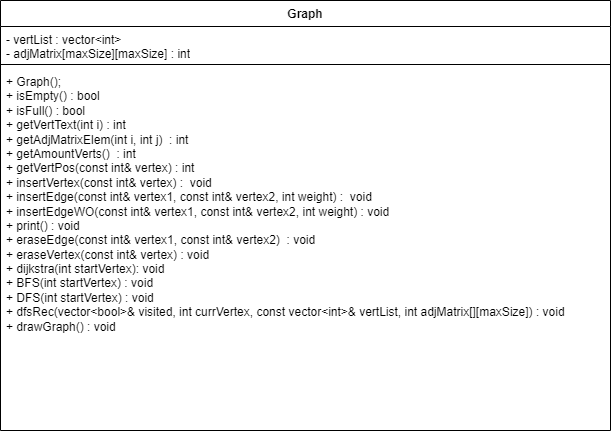
- Удаление вершины.

- Добавление и удаление ребра.

- Редактирование весов ребер.

- Редактирование матрицы смежности (или инцидентности в зависимости от реализации).

**UML-диаграмма**

****

**Описание пользовательского класса.**

const int maxSize = 8;

class Graph

{

private:

vector<int> vertList;

int adjMatrix[maxSize][maxSize];

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();

};

**Определение функций для визуализации графа**

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;

string s = to\_string(text);

glRasterPos2i(x1 - 5, y1 - 5);

for (size\_t j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

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);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

int textX = (x1 - x0) / 4 + x0;

int textY = (y1 - y0) / 4 + y0;

drawText(text, textX, textY,true); // вес ребра у начала стрелки

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

else glColor3f(0.0, 0.0, 0.0);

float vx = x0 - x1;

float vy = y0 - y1;

float s = 1.0f / sqrt(vx \* vx + vy \* vy);

vx \*= s;

vy \*= s;

x1 = x1 + R \* vx;

y1 = y1 + R \* vy;

glBegin(GL\_TRIANGLES);

glVertex2f(x1, y1);

glVertex2f(x1 + 10 \* (vx + vy), y1 + 10 \* (vy - vx));//стрелки

glVertex2f(x1 + 10 \* (vx - vy), y1 + 10 \* (vy + vx));

glEnd();

}

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) {

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

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

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

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

}

}

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.

}

**Функция main()**

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

{

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

makegraph();

setlocale(LC\_ALL, "Russian");

glutInit(&argc, argv);

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

glutInitWindowSize(800, 800);

glutCreateWindow("Графы");

glClearColor(1.0, 1.0, 1.0, 1.0);

glutDisplayFunc(display);

glutMainLoop();

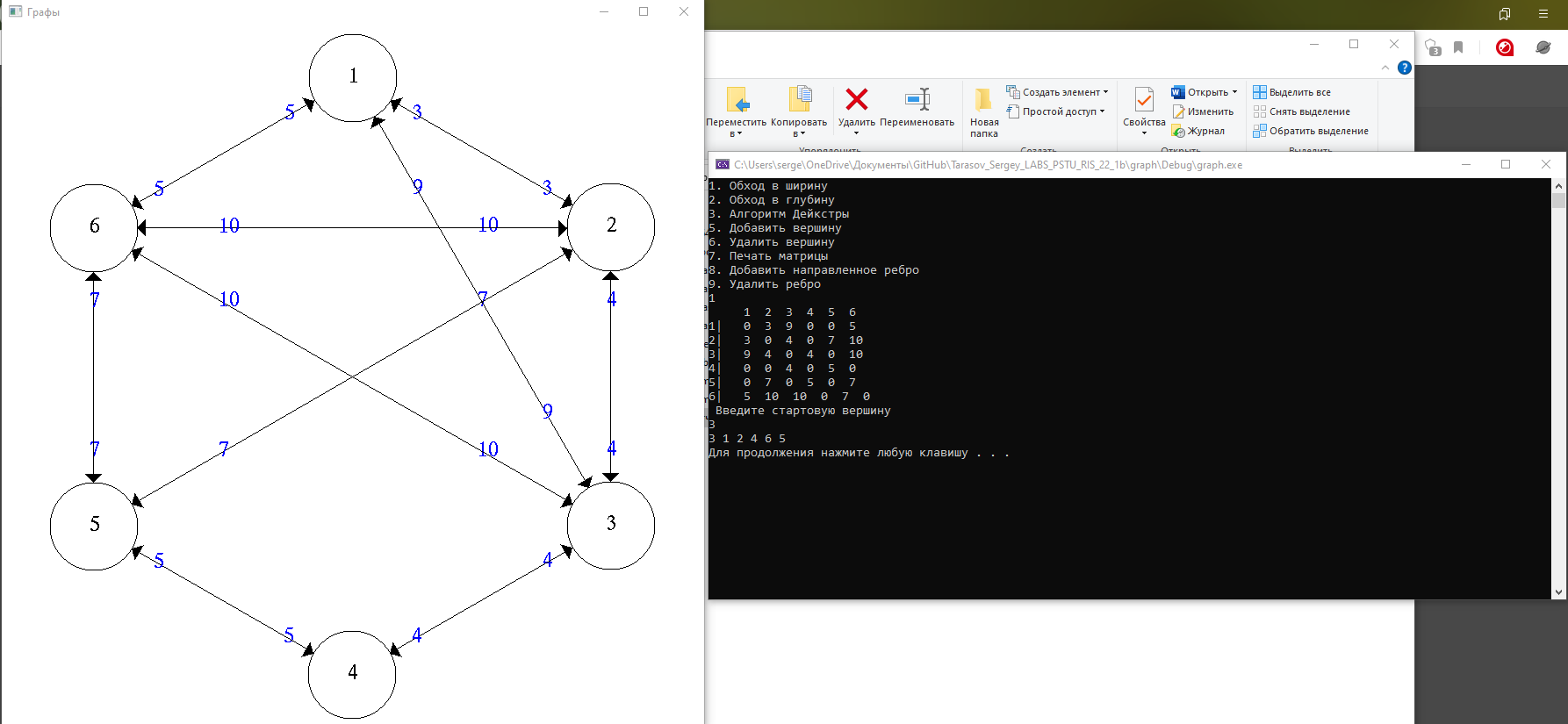
console.join();

return 0;

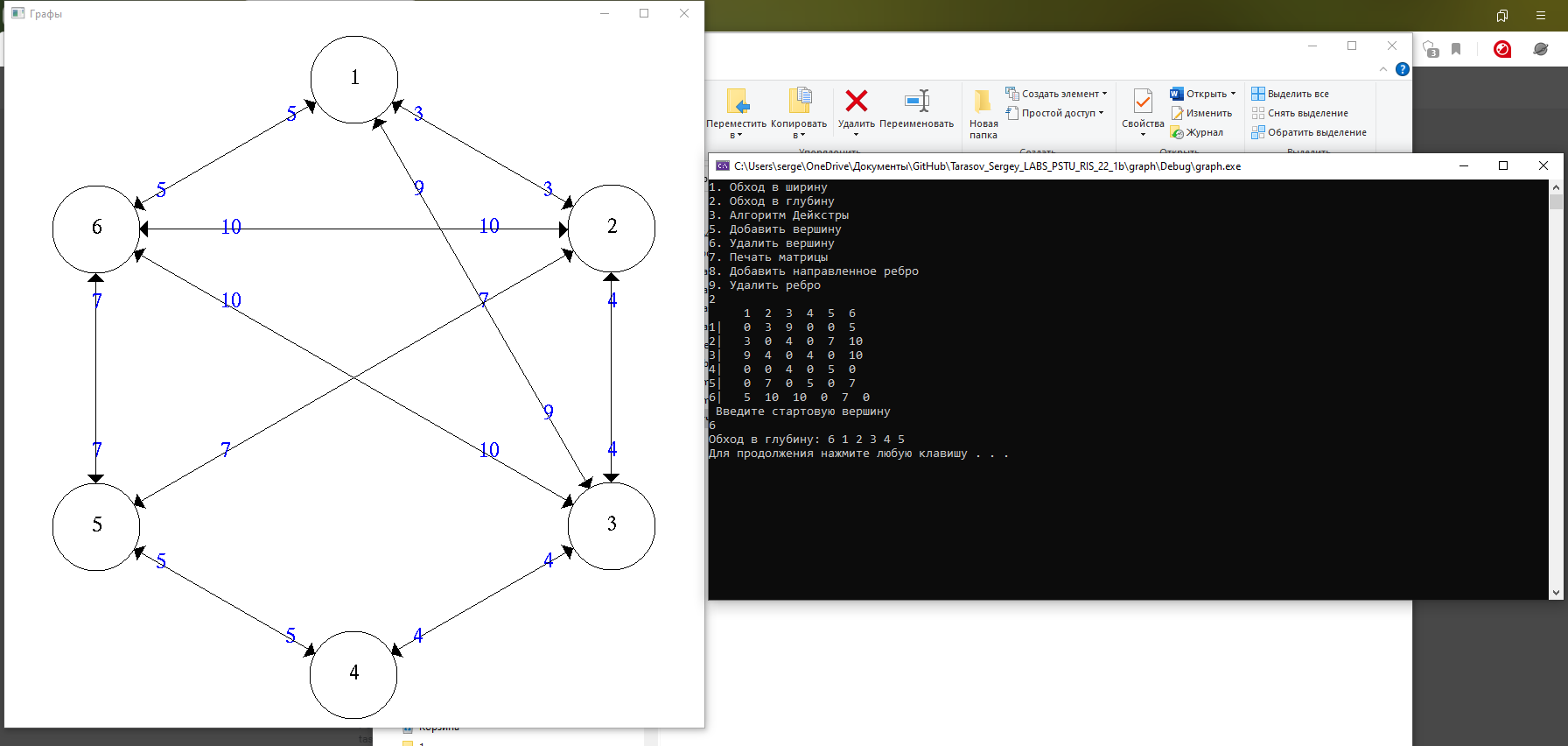
}

**Объяснение результатов программы**

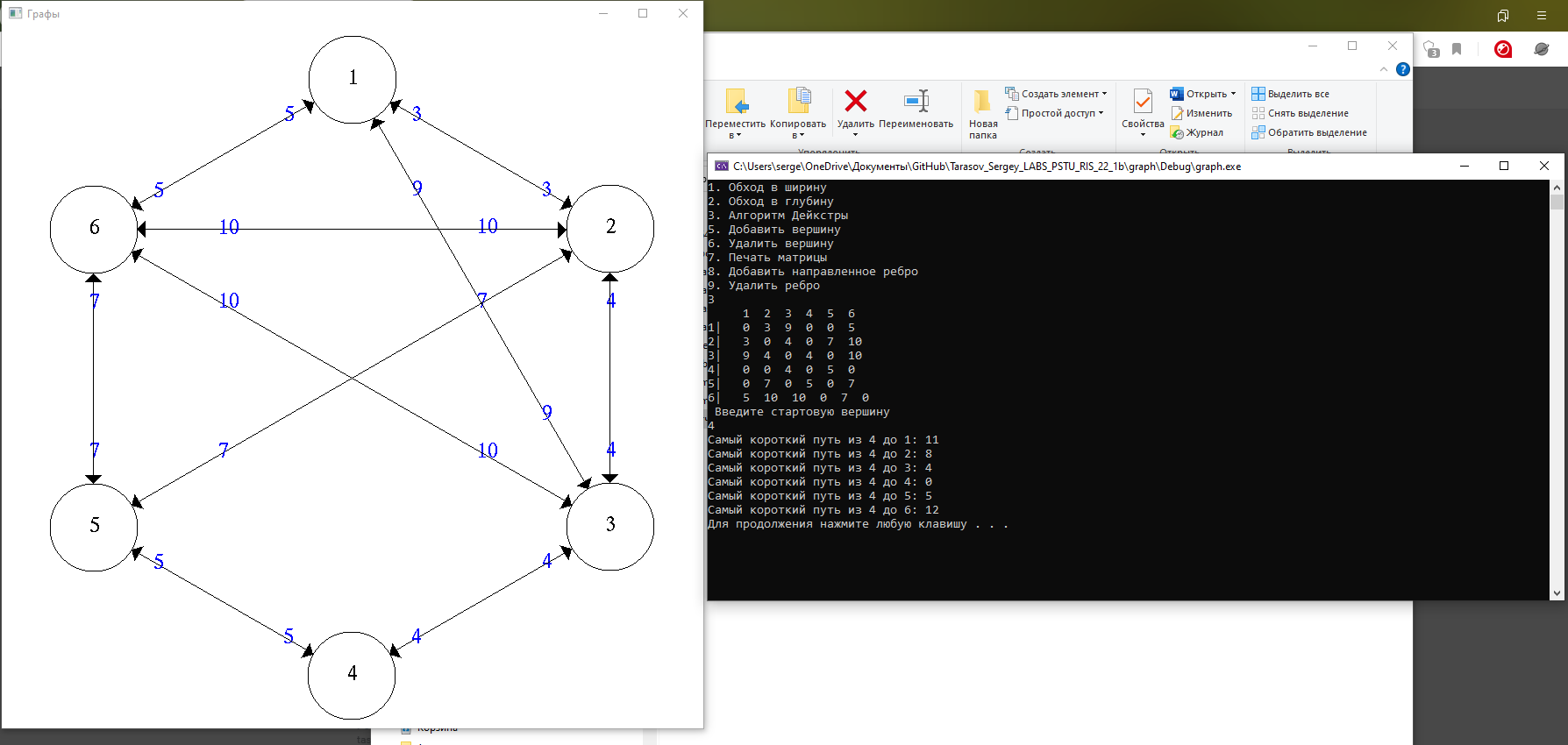
**Обход в ширину**



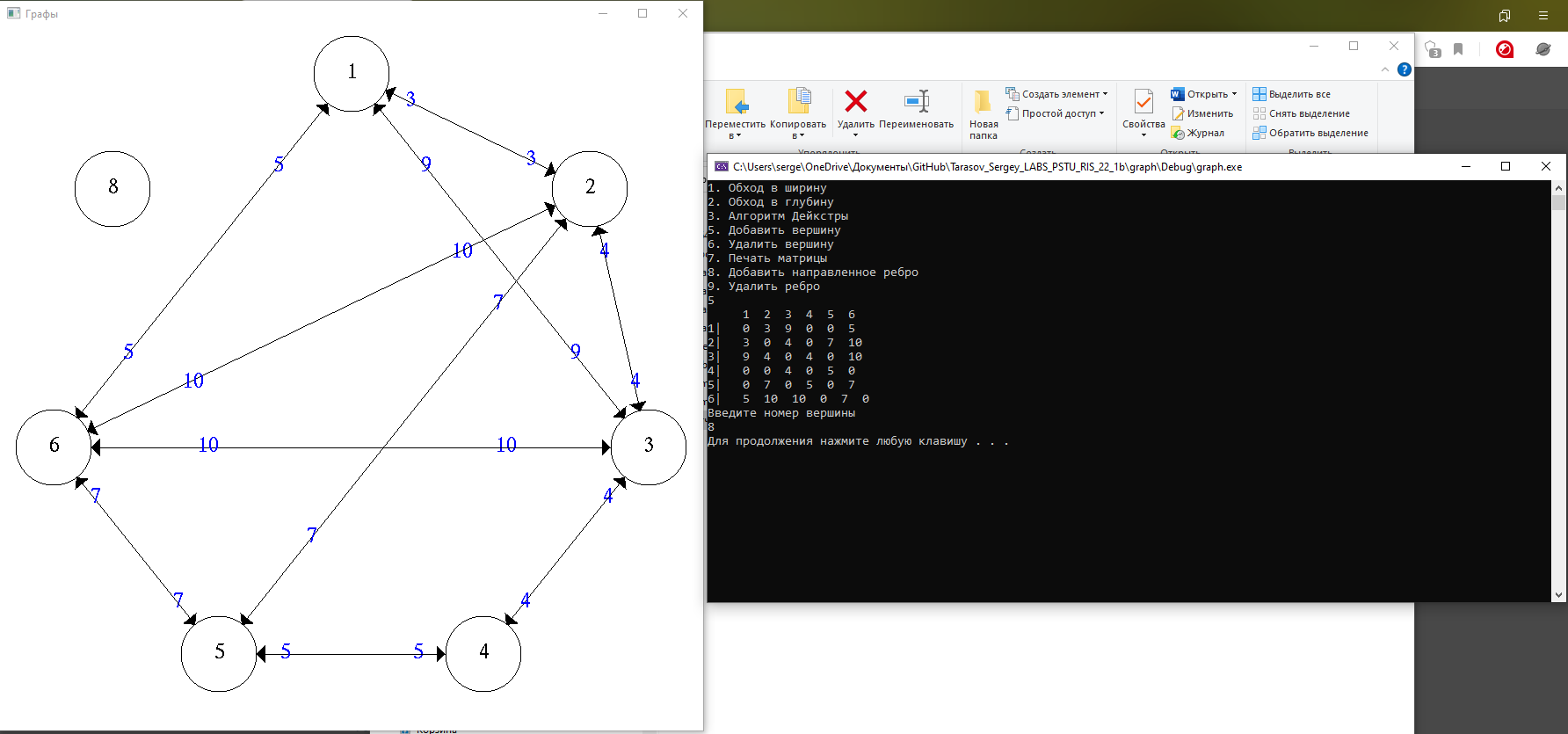
**Обход в глубину**



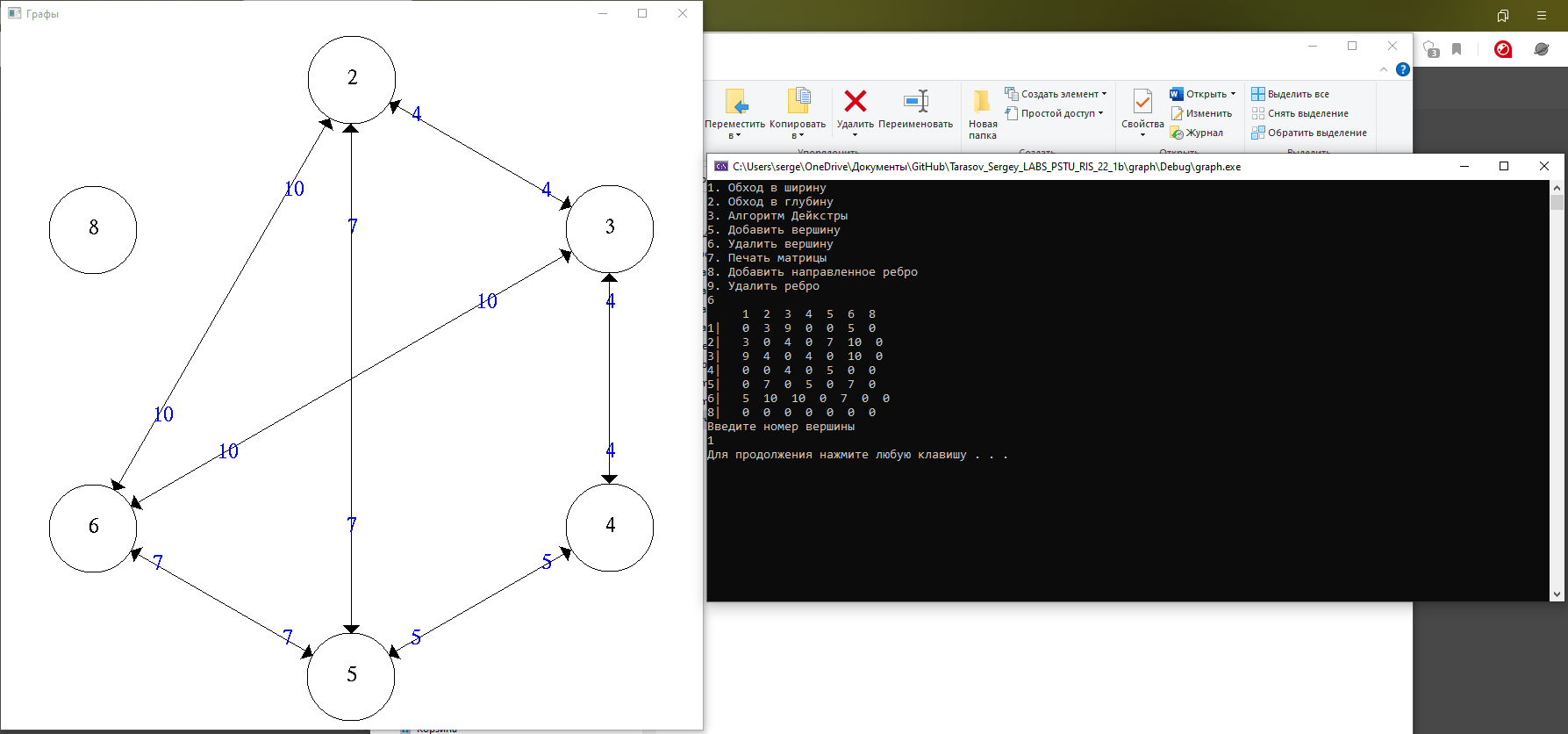
**Алгоритм Дейкстры**



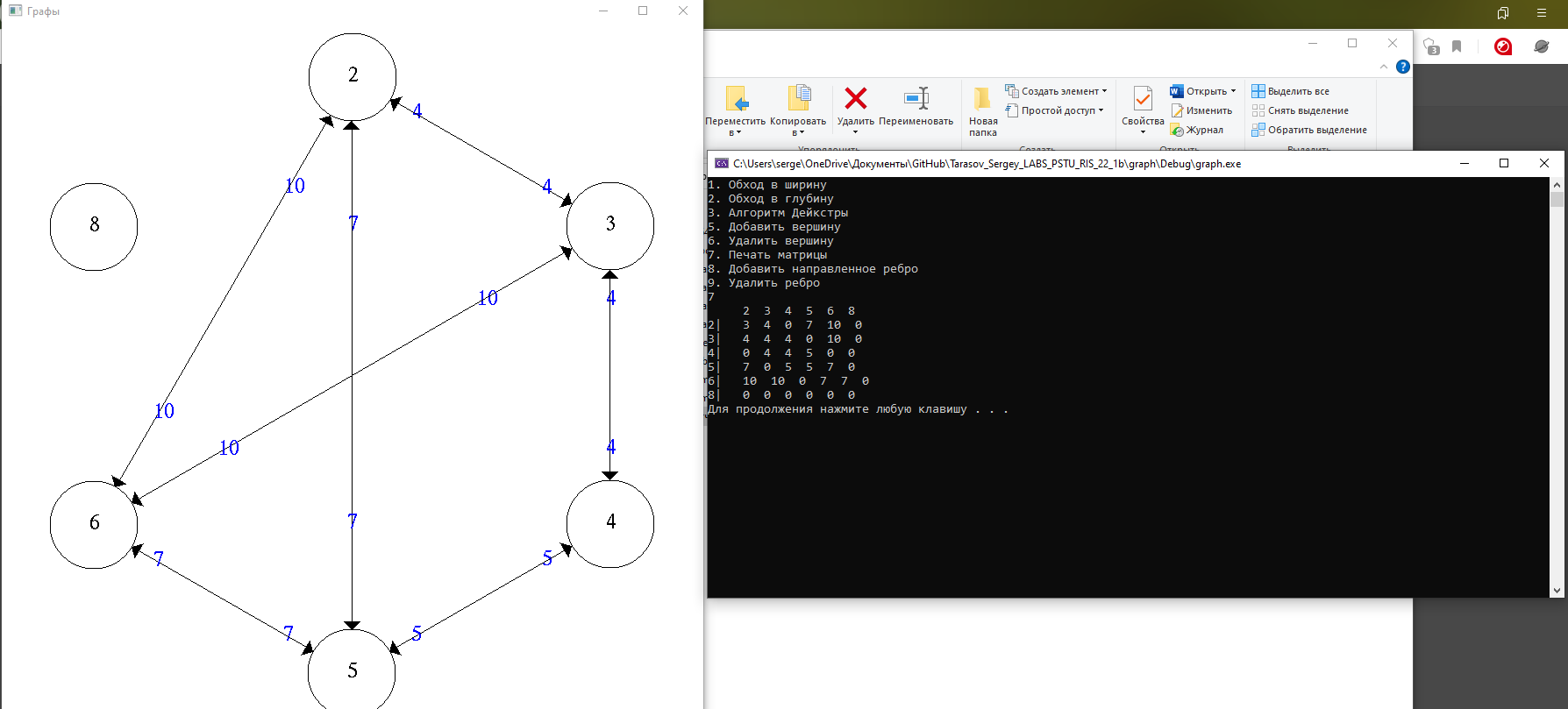
**Добавление вершины**



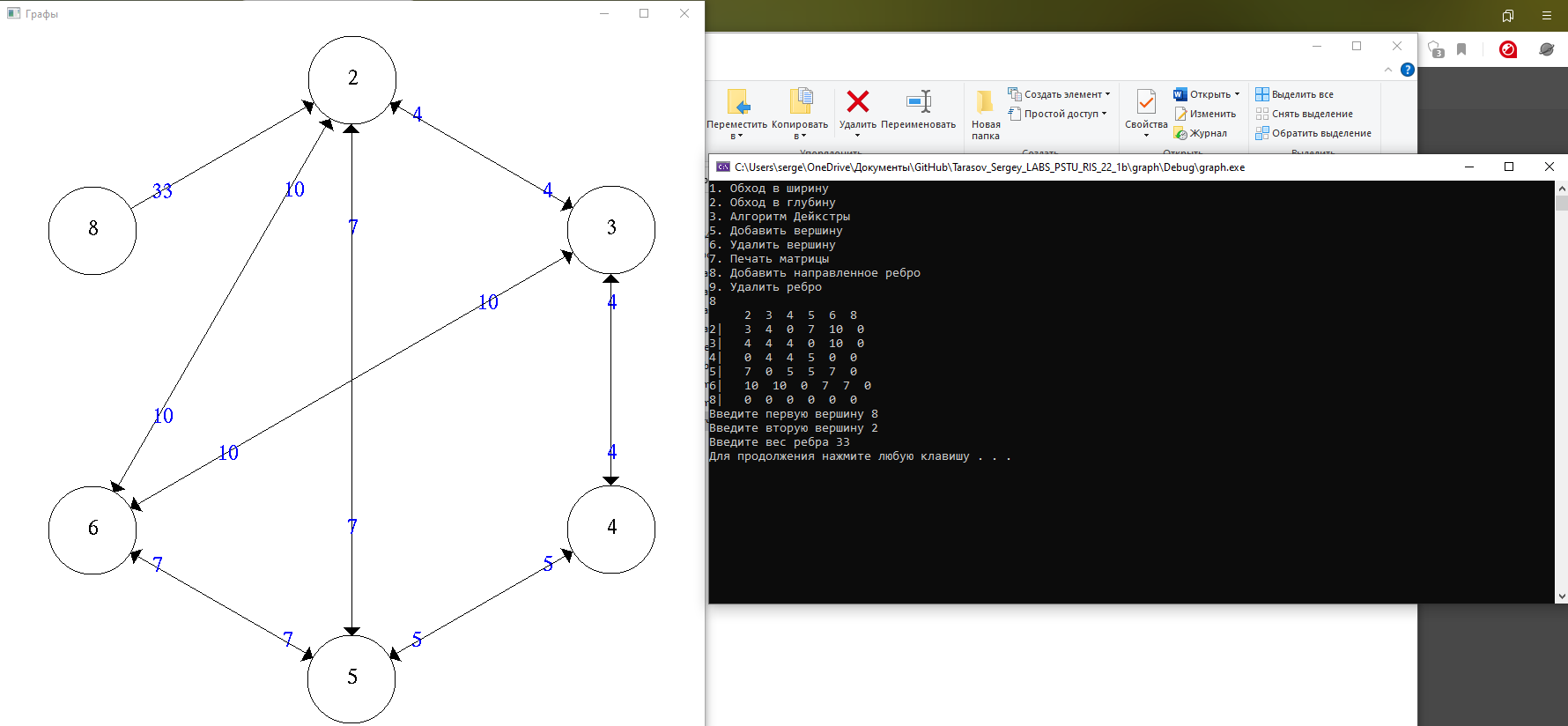
**Удаление вершины**



**Печать матрицы**



**Добавление ребра**



**Удаление ребра**

