Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.04– «Программная инженерия»

**Лабораторные работы**

**по дисциплине**

**«Теория Алгоритмов»**

Граф

Выполнил студент гр. РИС-22-1бзу

Иванов Александр Станиславович

Проверил:

доцент каф. ИТАС

Полякова О.А.

(оценка) (подпись)

(дата)

Пермь 2023

**Графы**

**Задачи работы**

Требуется реализовать Графы с алгоритмами:

1)Обход в ширину.

2)Обход в глубину.

3)Алгоритм Дейкстры.

С использованием графической библиотеки SFML C++.

**Анализ задачи**

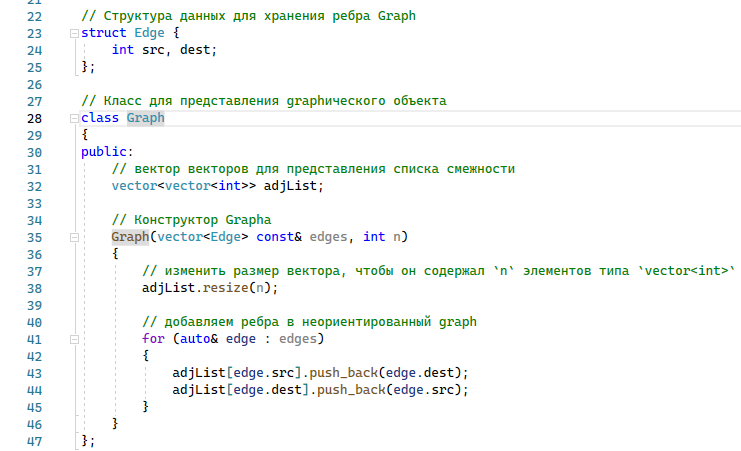
Определения функций для реализации поставленных задач

Алгоритм обхода в Ширину и Глубину

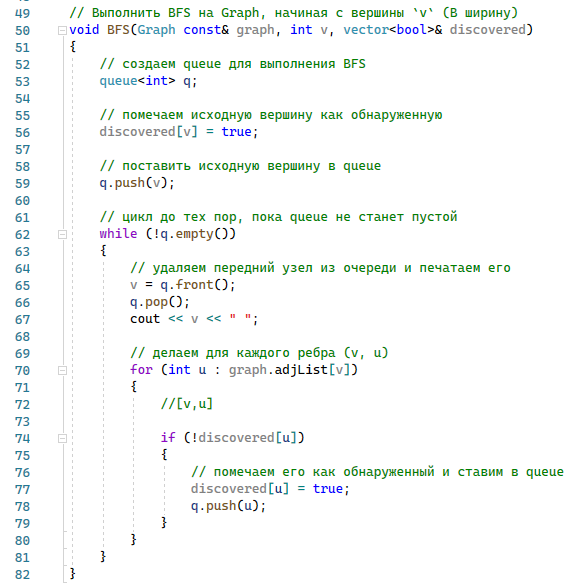
Поиск в ширину (BFS) — это алгоритм обхода Graph, который исследует вершины в порядке их удаления от исходной вершины, где расстояние — это минимальная длина пути от исходной вершины до узла.

Поиск в глубину (DFS) — это алгоритм обхода или поиска древовидных или графовых структур данных. Начинают с корня (выбирая произвольный узел в качестве корня Graph) и исследуют как можно дальше каждую ветвь, прежде чем перейти к следующей. Графы во всех алгоритмах неориентированные.

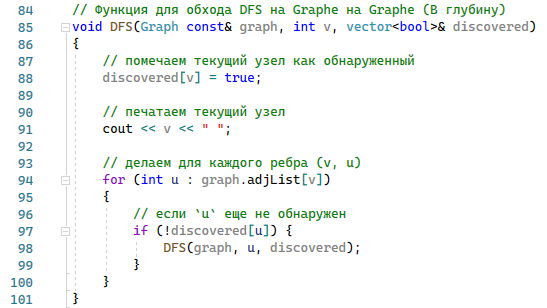
Объявление класса Граф и структуры Edge(ребра).



Реализация алгоритма обхода в ширину:



Реализация алгоритма обхода в глубину:



Main.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <SFML/Graphics.hpp>

#include <windows.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <string>

#include <sstream>

#include <map>

#include <queue> // очередь

#include <vector>

using namespace std;

using namespace sf;

// https://www.techiedelight.com/ru/breadth-first-search/ Статья

// https://www.techiedelight.com/ru/depth-first-search/

// Структура данных для хранения ребра Graph

struct Edge {

int src, dest;

};

// Класс для представления graphического объекта

class Graph

{

public:

// вектор векторов для представления списка смежности

vector<vector<int>> adjList;

// Конструктор Graphа

Graph(vector<Edge> const& edges, int n)

{

// изменить размер вектора, чтобы он содержал `n` элементов типа `vector<int>`

adjList.resize(n);

// добавляем ребра в неориентированный graph

for (auto& edge : edges)

{

adjList[edge.src].push\_back(edge.dest);

adjList[edge.dest].push\_back(edge.src);

}

}

};

// Выполнить BFS на Graph, начиная с вершины `v` (В ширину)

void BFS(Graph const& graph, int v, vector<bool>& discovered)

{

// создаем queue для выполнения BFS

queue<int> q;

// помечаем исходную вершину как обнаруженную

discovered[v] = true;

// поставить исходную вершину в queue

q.push(v);

// цикл до тех пор, пока queue не станет пустой

while (!q.empty())

{

// удаляем передний узел из очереди и печатаем его

v = q.front();

q.pop();

cout << v << " ";

// делаем для каждого ребра (v, u)

for (int u : graph.adjList[v])

{

//[v,u]

if (!discovered[u])

{

// помечаем его как обнаруженный и ставим в queue

discovered[u] = true;

q.push(u);

}

}

}

}

// Функция для обхода DFS на Graphе на Graphе (В глубину)

void DFS(Graph const& graph, int v, vector<bool>& discovered)

{

// помечаем текущий узел как обнаруженный

discovered[v] = true;

// печатаем текущий узел

cout << v << " ";

// делаем для каждого ребра (v, u)

for (int u : graph.adjList[v])

{

// если `u` еще не обнаружен

if (!discovered[u]) {

DFS(graph, u, discovered);

}

}

}

vector<Edge> edges;

int main()

{

setlocale(LC\_ALL, "Russian");

int x; // Для выбора обхода

cout << "Выберите обход графа: \n" << endl;

cout << "Обходы: 1 - В ширину, 2 - В глубину \n"; cin >> x;

switch (x)

{

case 1:

cout << "Обход: В ширину \n";

edges =

{

{1, 2}, {1, 3}, {2, 4}, {3, 5}

};

break;

case 2:

cout << "Обход: В глубину \n";

edges =

{

{1, 2}, {2, 4}, {2, 5}, {1, 3}

};

break;

default:

cout << "Выбрано неверное число";

}

//vector<Edge> edges = {

// //{1, 2}, {1, 3}, {1, 4}, {2, 5}, {2, 6}, {5, 9},

// //{5, 10}, {4, 7}, {4, 8}, {7, 11}, {7, 12}

// // {1, 2}, {1, 3}, {2, 4}, {3, 5}

// {1, 2}, {2, 4}, {2, 5}, {1, 3}

// // вершины 0, 13 и 14 — отдельные узлы

//};

// общее количество узлов в Graph (от 0 до 14)

//int n = 15;

int n = 6;

// строим graph из заданных ребер

Graph graph(edges, n);

// чтобы отслеживать, открыта вершина или нет

vector<bool> discovered(n, false);

// Выполняем обход BFS от всех необнаруженных узлов к

// покрываем все связные компоненты Graph

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

{

if (discovered[i] == false)

{

// начинаем обход BFS с вершины `i`

BFS(graph, i, discovered);

}

}

cout << "\n";

// Свойства окна Windows

sf::RenderWindow window(sf::VideoMode(1280, 720), L"Граф с использованием SFML");

window.setFramerateLimit(60); //16,6ms оптимизация 60 кадров в сек.

Font font;

Text value1, value2, value3, value4, value5, valueLine1, valueLine2, valueLine3, valueLine4, valueLine5,

valueLine6, valueLine7, valueLine8, valueLine9, valueLine10, // Длина пути (отрезка)

valueNode1, valueNode2, valueNode3, valueNode4, valueNode5; // Весы ребер

// Create a graphical text to display

if (!font.loadFromFile("Font/BankGothic Md BT Medium.otf"))

return EXIT\_FAILURE;

sf::CircleShape shapeFirst(40.f), shapeSecond(40.f), shapeThird(40.f), shapeFour(40.f), shapeFive(40.f); // рисуем круг

// 1 нода

shapeFirst.setFillColor(sf::Color::Green);

shapeFirst.setPosition(550, 80);

shapeFirst.setOutlineThickness(5.f); // Обводка

shapeFirst.setOutlineColor(sf::Color(250, 150, 100));

value1.setFont(font); // текст

value1.setFillColor(Color::Black);

value1.setCharacterSize(50);

value1.setPosition(570, 80);

value1.setString("1");

RectangleShape line1(Vector2f(285, 5)); // длина, толщина (1-2)

line1.rotate(160); // Поворот

line1.setFillColor(Color::Black);

line1.setPosition(550, 130);

// 2 нода

shapeSecond.setFillColor(sf::Color::Green);

shapeSecond.setPosition(200, 200);

shapeSecond.setOutlineThickness(5.f);

shapeSecond.setOutlineColor(sf::Color(250, 150, 100));

value2.setFont(font);

value2.setFillColor(Color::Black);

value2.setCharacterSize(50);

value2.setPosition(220, 200);

value2.setString("2");

RectangleShape line7(Vector2f(270, 5)); // длина, толщина (2-3)

line7.rotate(80); // Поворот

line7.setFillColor(Color::Black);

line7.setPosition(250, 280);

RectangleShape line10(Vector2f(520, 5)); // длина, толщина (2-5)

line10.rotate(183); // Поворот

line10.setFillColor(Color::Black);

line10.setPosition(800, 270);

// 3 нода

shapeThird.setFillColor(sf::Color::Green);

shapeThird.setPosition(260, 550);

shapeThird.setOutlineThickness(5.f);

shapeThird.setOutlineColor(sf::Color(250, 150, 100));

value3.setFont(font);

value3.setFillColor(Color::Black);

value3.setCharacterSize(50);

value3.setPosition(280, 550);

value3.setString("3");

RectangleShape line3(Vector2f(470, 5)); // длина, толщина (1-3)

line3.rotate(120); // Поворот

line3.setFillColor(Color::Black);

line3.setPosition(565, 150);

RectangleShape line2(Vector2f(550, 5)); // длина, толщина (3-5)

line2.rotate(328); // Поворот

line2.setFillColor(Color::Black);

line2.setPosition(335, 570);

// Четвертая нода

shapeFour.setFillColor(sf::Color::Green);

shapeFour.setPosition(670, 530);

shapeFour.setOutlineThickness(5.f);

shapeFour.setOutlineColor(sf::Color(250, 150, 100));

value4.setFont(font);

value4.setFillColor(Color::Black);

value4.setCharacterSize(50);

value4.setPosition(690, 530);

value4.setString("4");

RectangleShape line4(Vector2f(385, 5)); // длина, толщина (1-4)

line4.rotate(75); // Поворот

line4.setFillColor(Color::Black);

line4.setPosition(600, 158);

RectangleShape line8(Vector2f(325, 5)); // длина, толщина (4-3)

line8.rotate(178); // Поворот

line8.setFillColor(Color::Black);

line8.setPosition(670, 580);

RectangleShape line9(Vector2f(495, 5)); // длина, толщина (4-2)

line9.rotate(216); // Поворот

line9.setFillColor(Color::Black);

line9.setPosition(670, 560);

// Пятая нода

shapeFive.setFillColor(sf::Color::Green);

shapeFive.setPosition(800, 230);

shapeFive.setOutlineThickness(5.f);

shapeFive.setOutlineColor(sf::Color(250, 150, 100));

value5.setFont(font);

value5.setFillColor(Color::Black);

value5.setCharacterSize(50);

value5.setPosition(820, 230);

value5.setString("5");

RectangleShape line5(Vector2f(210, 5)); // длина, толщина (1-5)

line5.rotate(35); // Поворот

line5.setFillColor(Color::Black);

line5.setPosition(630, 130);

RectangleShape line6(Vector2f(240, 5)); // длина, толщина (5-4)

line6.rotate(110); // Поворот

line6.setFillColor(Color::Black);

line6.setPosition(820, 310);

map <pair<int, int>, RectangleShape> mp; // map 1 - составной ключ из двух интов a[i][j], 2 - линия

mp[make\_pair(1, 2)] = line1;

mp[make\_pair(1, 3)] = line3;

mp[make\_pair(1, 4)] = line4;

mp[make\_pair(1, 5)] = line5;

mp[make\_pair(2, 3)] = line7;

mp[make\_pair(2, 4)] = line9;

mp[make\_pair(2, 5)] = line10;

mp[make\_pair(3, 4)] = line8;

mp[make\_pair(3, 5)] = line2;

mp[make\_pair(4, 5)] = line6;

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

// проверяем тип события...

switch (event.type)

{

// окно закрыто

case Event::Closed:

window.close();

break;

// клавиша нажата

case Event::KeyPressed:

// Управление для разработчика

switch (event.key.code)

{

case Keyboard::Escape:window.close(); break; // закрытие программы на ESC

}

}

}

window.clear(sf::Color::Cyan);

window.draw(shapeFirst);

window.draw(shapeSecond);

window.draw(shapeThird);

window.draw(shapeFour);

window.draw(shapeFive);

// Отрисовка номера Ноды

window.draw(value1);

window.draw(value2);

window.draw(value3);

window.draw(value4);

window.draw(value5);

window.display();

sf::sleep(sf::seconds(1));

window.display();

for (auto node : edges)

{

cout << node.src << " - " << node.dest << "\n";

window.draw(mp[make\_pair(node.src, node.dest)]);

window.display();

sf::sleep(sf::seconds(1));

window.display();

}

window.display();

}

return 0;

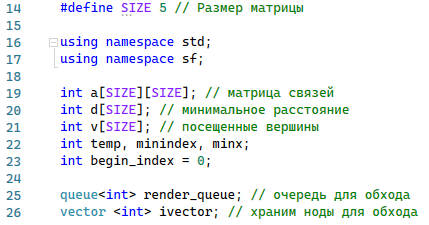
}

**Алгоритм Дейкстры**

Алгоритм Дейкстры — алгоритм на графах, изобретённый нидерландским ученым Э. Дейкстрой в 1959 году. Находит кратчайшее расстояние от одной из вершин графа до всех остальных. Работает только для графов без рёбер отрицательного веса.

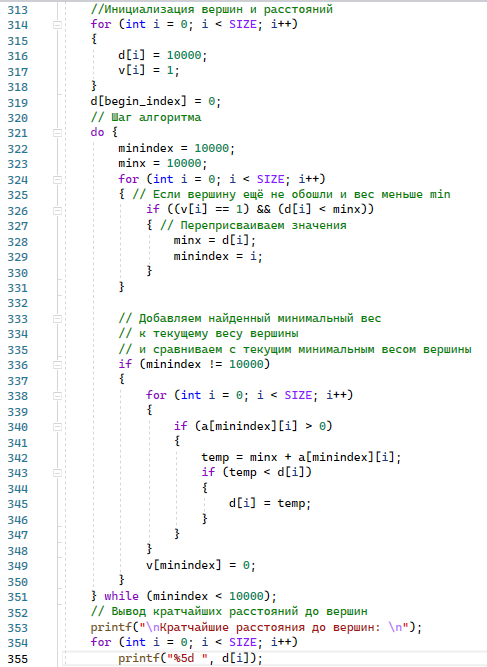
В алгоритме Дейкстры я решил использовать матрицу, вместо вектора.

Структура Графа:

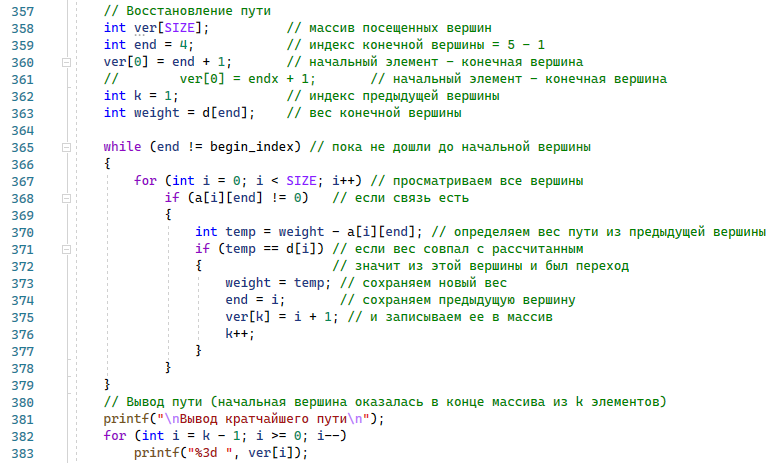


Двухмерный массив A – Граф Неориентированный.

Обход графа по Дейкстры:



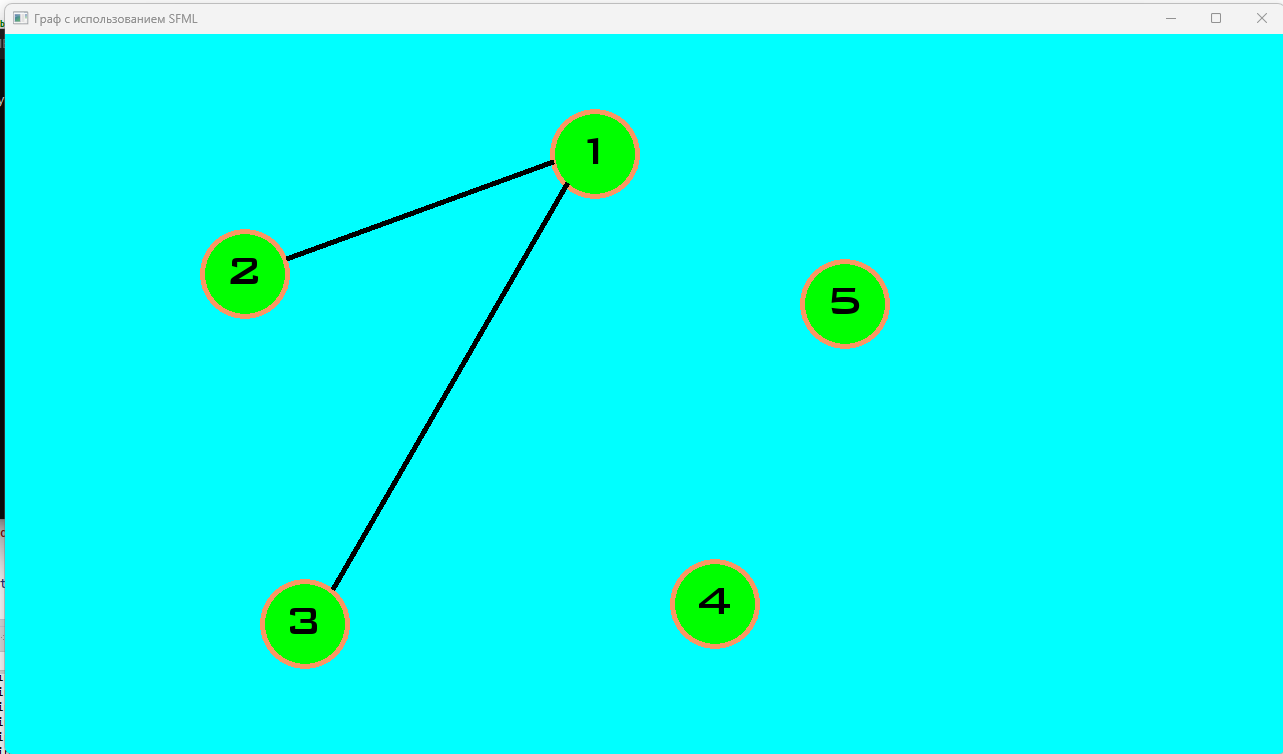
Алгоритм на поиск кратчайшего пути

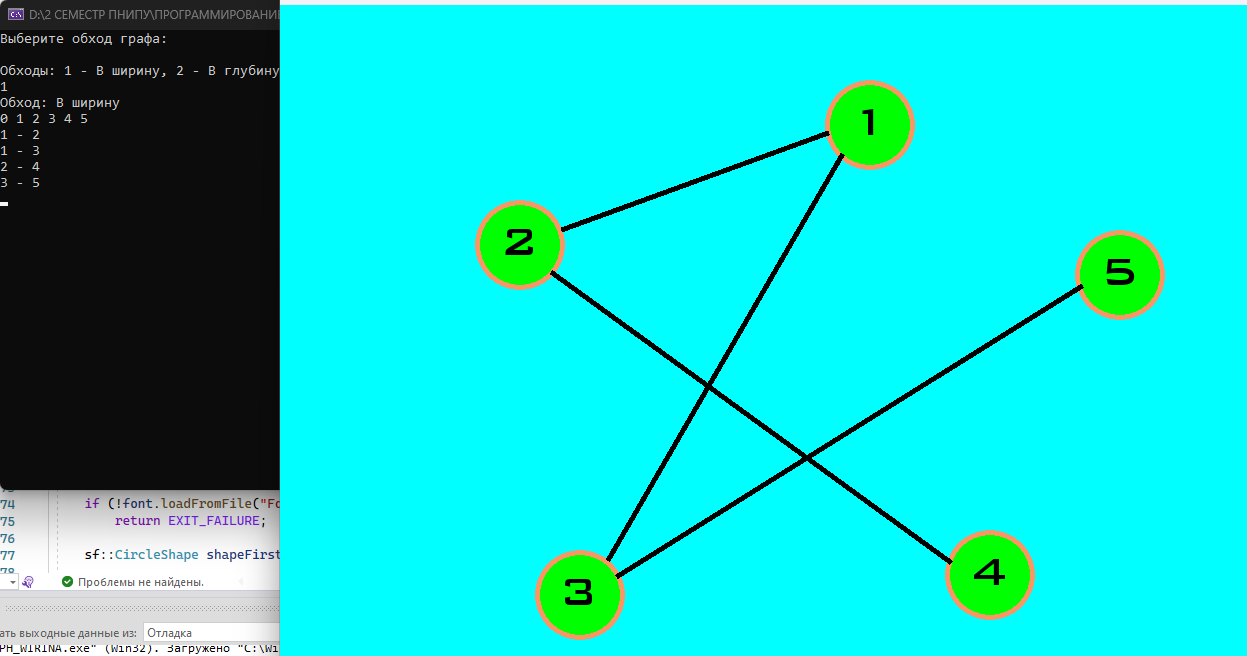


**Работа программы**

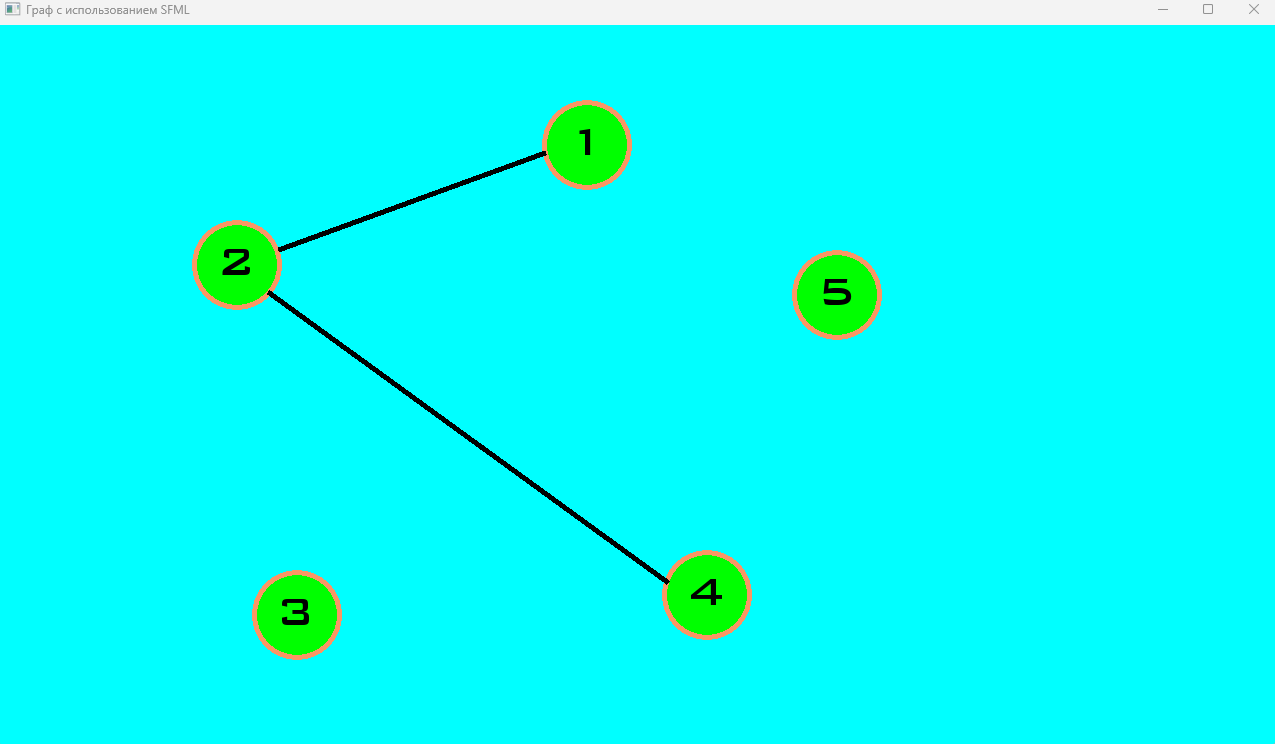
Обход в Ширину

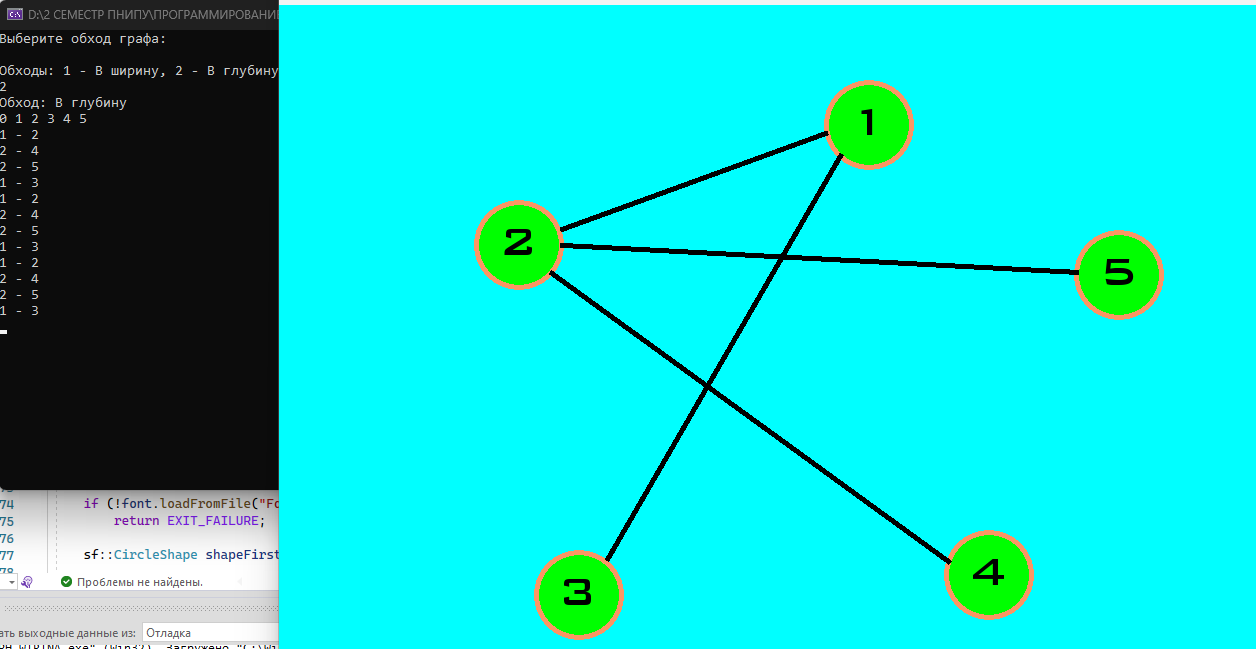
Программа постепенно рисует ребра Графа, в соответствии с алгоритмом обхода.



****

Обход в глубину





Алгоритм Дейкстры

Черными цифрами обозначены длины путей (вес ребер).

Красными цифрами (над нодами) обозначены мтки пройденных нод.

