Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

(ФГАОУ ВО ПНИПУ)

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ

ОТЧЁТ

О ЛАБОРАТОРНОЙ РАБОТЕ

На тему: ГРАФЫ

Выполнил:

Студент группы РИС-23-3б

Асташин Д.А.

Проверил:

доц. кафедры ИТАС

Полякова О.А.

Пермь 2024

**Постановка задачи**

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

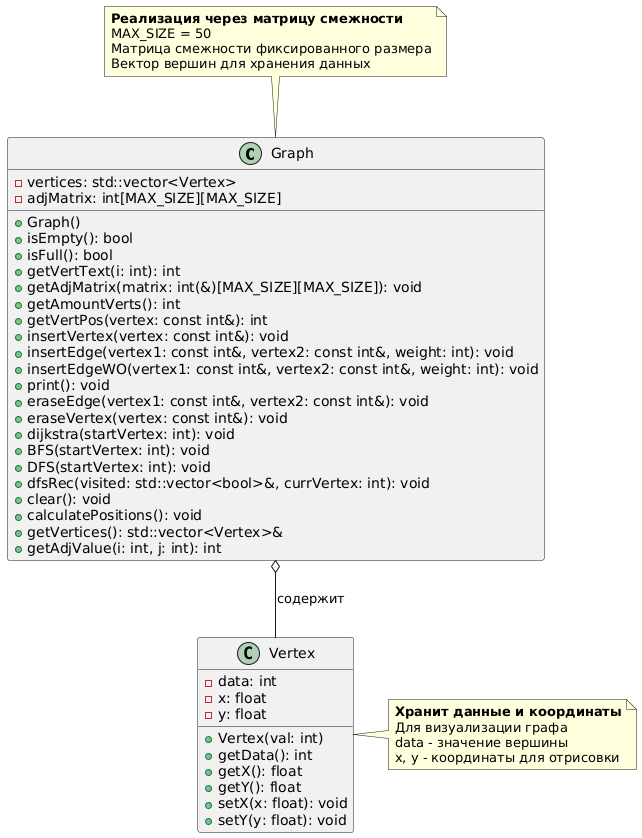
Алгоритмы:

1. Обход в ширину.

2. Обход в глубину.

3. Алгоритм Дейкстры.

**UML Диаграмма**



**Код программы**

**Файл graph.h**

#ifndef GRAPH\_H

#define GRAPH\_H

#include <iostream>

#include <vector>

#include <queue>

#include <stack>

#include <algorithm>

#include <limits>

#include <functional>

#include <cmath>

const int MAX\_SIZE = 50;

struct Vertex {

int data;

float x, y;

Vertex(int val) : data(val), x(0), y(0) {}

};

class Graph {

private:

std::vector<Vertex> vertices;

int adjMatrix[MAX\_SIZE][MAX\_SIZE];

public:

Graph();

bool isEmpty();

bool isFull();

int getVertText(int i);

void getAdjMatrix(int(&matrix)[MAX\_SIZE][MAX\_SIZE]);

int getAmountVerts();

int getVertPos(const int& vertex);

void insertVertex(const int& vertex);

void insertEdge(const int& vertex1, const int& vertex2, int weight);

void insertEdgeWO(const int& vertex1, const int& vertex2, int weight);

void print();

void eraseEdge(const int& vertex1, const int& vertex2);

void eraseVertex(const int& vertex);

void dijkstra(int startVertex);

void BFS(int startVertex);

void DFS(int startVertex);

void dfsRec(std::vector<bool>& visited, int currVertex);

// Новый метод для очистки графа

void clear();

// Методы для визуализации

void calculatePositions();

std::vector<Vertex>& getVertices() { return vertices; }

int getAdjValue(int i, int j) { return adjMatrix[i][j]; }

};

#endif

**Файл graph.cpp**

#include "graph.h"

Graph::Graph() {

for (int i = 0; i < MAX\_SIZE; i++)

for (int j = 0; j < MAX\_SIZE; j++)

adjMatrix[i][j] = 0;

}

bool Graph::isEmpty() {

return vertices.empty();

}

bool Graph::isFull() {

return vertices.size() == MAX\_SIZE;

}

int Graph::getVertText(int i) {

return vertices[i].data;

}

void Graph::getAdjMatrix(int(&matrix)[MAX\_SIZE][MAX\_SIZE]) {

for (int i = 0; i < MAX\_SIZE; i++) {

for (int j = 0; j < MAX\_SIZE; j++) {

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

}

}

}

int Graph::getAmountVerts() {

return vertices.size();

}

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

for (size\_t i = 0; i < vertices.size(); i++) {

if (vertices[i].data == vertex)

return i;

}

return -1;

}

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

if (isFull()) {

std::cout << "Граф уже заполнен. Невозможно добавить новую вершину" << std::endl;

return;

}

vertices.push\_back(Vertex(vertex));

calculatePositions();

}

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

if (weight < 1) {

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

return;

}

int vertPos1 = getVertPos(vertex1);

int vertPos2 = getVertPos(vertex2);

if (vertPos1 == -1 || vertPos2 == -1) {

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

return;

}

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

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

return;

}

adjMatrix[vertPos1][vertPos2] = weight;

}

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

if (weight < 1) {

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

return;

}

int vertPos1 = getVertPos(vertex1);

int vertPos2 = getVertPos(vertex2);

if (vertPos1 == -1 || vertPos2 == -1) {

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

return;

}

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

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

return;

}

adjMatrix[vertPos1][vertPos2] = weight;

adjMatrix[vertPos2][vertPos1] = weight;

}

void Graph::print() {

if (isEmpty()) {

std::cout << "Граф пуст" << std::endl;

return;

}

std::cout << " ";

for (size\_t i = 0; i < vertices.size(); i++)

std::cout << vertices[i].data << " ";

std::cout << std::endl;

for (size\_t i = 0; i < vertices.size(); i++) {

std::cout << vertices[i].data << "| ";

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

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

std::cout << std::endl;

}

}

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

int vertPos1 = getVertPos(vertex1);

int vertPos2 = getVertPos(vertex2);

if (vertPos1 == -1 || vertPos2 == -1) {

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

return;

}

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

std::cout << "Ребра и так нет" << std::endl;

return;

}

adjMatrix[vertPos1][vertPos2] = 0;

adjMatrix[vertPos2][vertPos1] = 0;

}

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

int index = getVertPos(vertex);

if (index == -1) {

std::cout << "Вершина не найдена" << std::endl;

return;

}

// Удаляем вершину

vertices.erase(vertices.begin() + index);

// Обновляем матрицу смежности

for (size\_t i = index; i < vertices.size(); i++) {

for (size\_t j = 0; j < vertices.size(); j++) {

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

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

}

}

// Обнуляем последнюю строку и столбец

for (size\_t i = 0; i <= vertices.size(); i++) {

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

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

}

calculatePositions();

}

void Graph::dijkstra(int startVertex) {

int n = vertices.size();

std::vector<int> distance(n, std::numeric\_limits<int>::max());

std::vector<bool> visited(n, false);

distance[startVertex] = 0;

for (int count = 0; count < n - 1; count++) {

int minDistance = std::numeric\_limits<int>::max();

int minIndex = -1;

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

if (!visited[v] && distance[v] <= minDistance) {

minDistance = distance[v];

minIndex = v;

}

}

if (minIndex == -1) break;

visited[minIndex] = true;

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

if (!visited[v] && adjMatrix[minIndex][v] &&

distance[minIndex] != std::numeric\_limits<int>::max() &&

distance[minIndex] + adjMatrix[minIndex][v] < distance[v]) {

distance[v] = distance[minIndex] + adjMatrix[minIndex][v];

}

}

}

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

std::cout << "Самый короткий путь из " << vertices[startVertex].data

<< " до " << vertices[i].data << ": ";

if (distance[i] == std::numeric\_limits<int>::max())

std::cout << "пути нет";

else

std::cout << distance[i];

std::cout << std::endl;

}

}

void Graph::BFS(int startVertex) {

if (isEmpty()) {

std::cout << "Граф пуст" << std::endl;

return;

}

int n = vertices.size();

std::vector<bool> visited(n, false);

std::queue<int> q;

visited[startVertex] = true;

q.push(startVertex);

std::cout << "Обход в ширину: ";

while (!q.empty()) {

int currVertex = q.front();

q.pop();

std::cout << vertices[currVertex].data << " ";

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

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

visited[i] = true;

q.push(i);

}

}

}

std::cout << std::endl;

}

void Graph::DFS(int startVertex) {

if (isEmpty()) {

std::cout << "Граф пуст" << std::endl;

return;

}

int n = vertices.size();

std::vector<bool> visited(n, false);

std::cout << "Обход в глубину: ";

dfsRec(visited, startVertex);

std::cout << std::endl;

}

void Graph::dfsRec(std::vector<bool>& visited, int currVertex) {

visited[currVertex] = true;

std::cout << vertices[currVertex].data << " ";

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

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

dfsRec(visited, i);

}

}

}

void Graph::calculatePositions() {

int n = vertices.size();

if (n == 0) return;

float centerX = 400.0f;

float centerY = 300.0f;

float radius = 200.0f;

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

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

vertices[i].x = centerX + radius \* std::sin(angle);

vertices[i].y = centerY + radius \* std::cos(angle);

}

}

void Graph::clear() {

vertices.clear();

for (int i = 0; i < MAX\_SIZE; i++) {

for (int j = 0; j < MAX\_SIZE; j++) {

adjMatrix[i][j] = 0;

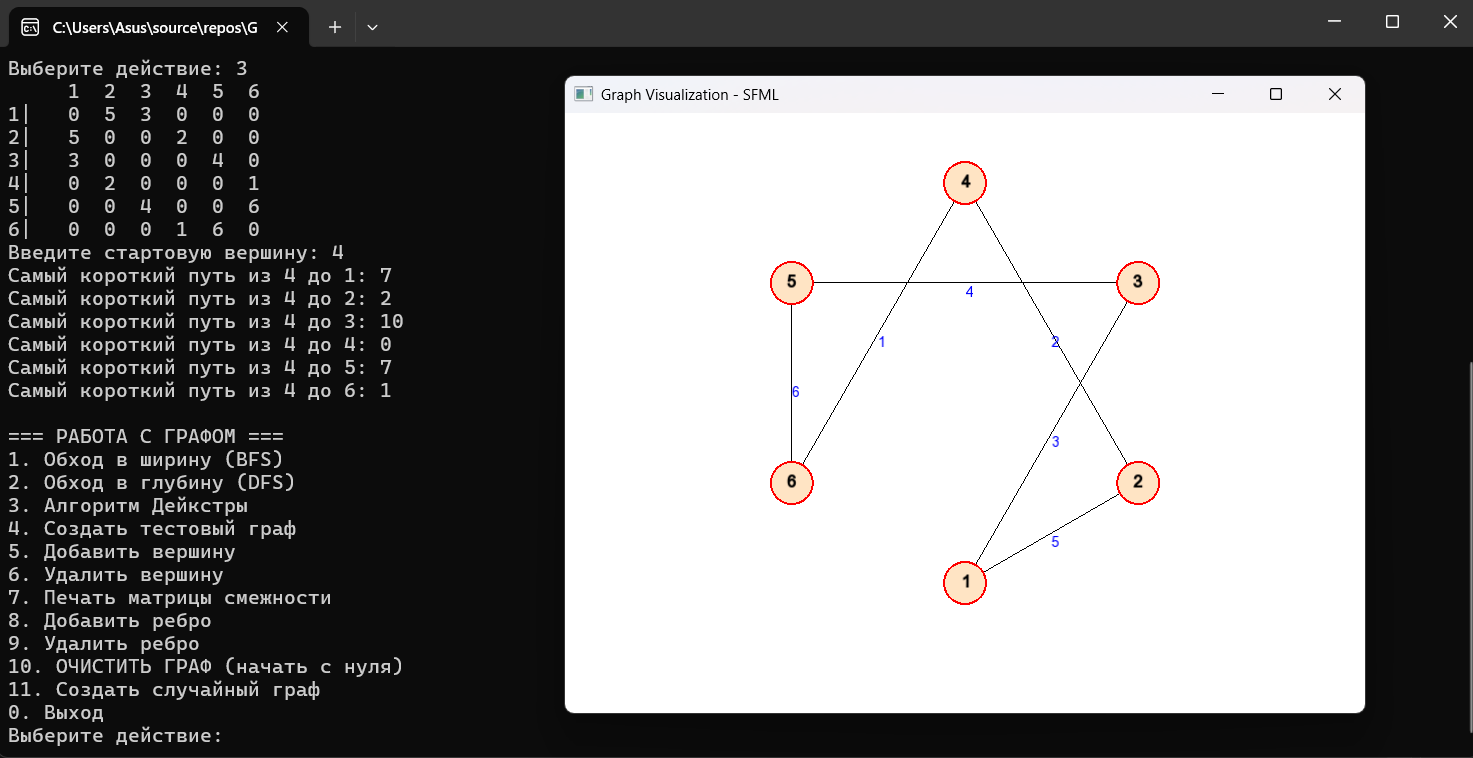
}

}

std::cout << "Граф полностью очищен" << std::endl;

}

**Результат работы программы**

****

**Ссылка GitHub**

https://github.com/astidii/PNRPU