Министерство общего и профессионального

образования Российской Федерации

Пермский государственный технический университет

Лабораторная работа №2

“Введение в теорию графов. Алгоритмы Дейкстры и Флойда”

Вариант №12

Выполнила студентка группы РИС 23-3б:

Федорова О.И.

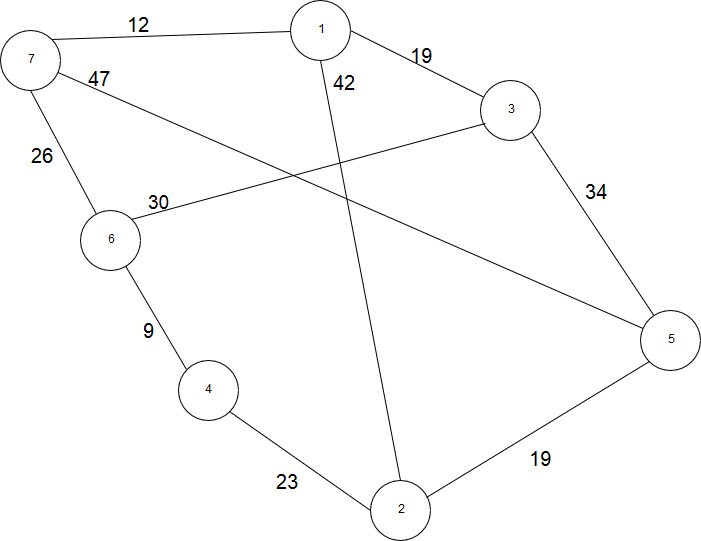
Проверила

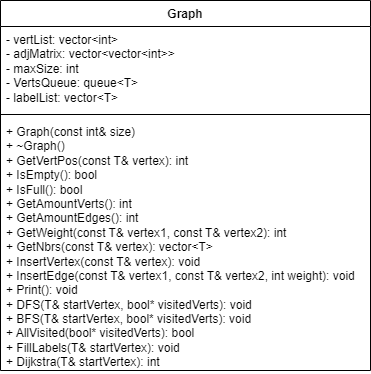
Доцент кафедры ИТАС

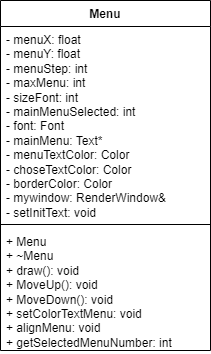
Полякова О.А.

Пермь 2024

1. Постановка задачи:

 Реализовать граф, а также алгоритм Дейкстры, выполнив все необходимые действия. Выполнение начать с вершины 6.

1. Анализ задачи:
2. В классе Graph определены различные методы для работы с графом, включая методы для добавления вершин и рёбер, обхода графа в глубину и ширину, а также алгоритма Дейкстры для поиска кратчайших путей.   
   Основные поля класса включают:   
   - vertList - вектор вершин графа.  
   - adjMatrix - двумерный вектор, представляющий матрицу смежности графа.  
   - maxSize - максимальное количество вершин в графе.  
   - VertsQueue - очередь вершин для использования в алгоритмах обхода.  
   - labelList - вектор меток вершин.   
   Методы класса включают:   
   - Graph(const int& size) - конструктор, создающий новый экземпляр графа с заданным размером.  
   - ~Graph() - деструктор, освобождающий память, занятую графом.  
   - GetVertPos(const T& vertex) - метод для получения позиции вершины в векторе вершин.  
   - IsEmpty() - метод для проверки, пуст ли граф.  
   - IsFull() - метод для проверки, полон ли граф.  
   - GetAmountVerts() - метод для получения количества вершин в графе.  
   - GetAmountEdges() - метод для получения количества рёбер в графе.  
   - GetWeight(const T& vertex1, const T& vertex2) - метод для получения веса рёбра между двумя вершинами.  
   - GetNbrs(const T& vertex) - метод для получения списка соседей вершины.  
   - InsertVertex(const T& vertex) - метод для добавления новой вершины в граф.  
   - InsertEdge(const T& vertex1, const T& vertex2, int weight) - метод для добавления нового ребра между двумя вершинами с заданным весом.  
   - Print() - метод для печати матрицы смежности графа.  
   - DFS(T& startVertex, bool \* visitedVerts) - метод для выполнения обхода графа в глубину.  
   - BFS(T& startVertex, bool \* visitedVerts) - метод для выполнения обхода графа в ширину.  
   - AllVisited(bool \* visitedVerts) - метод для проверки, были ли посещены все вершины при обходе.  
   - FillLabels(T& startVertex) - метод для заполнения очереди вершин метками.  
   - Dijkstra(T& startVertex) - метод для выполнения алгоритма Дейкстры для поиска кратчайших путей от начальной вершины.
3. Функция drawGraph создает окно игры с использованием библиотеки SFML и рисует в нем граф. Окно имеет размеры 1300x700 пикселей и стиль закрытия. Граф состоит из семи окружностей, каждая из которых соответствует вершине графа, и десяти линий, каждая из которых соответствует ребру графа. Позиции вершин и рёбер задаются в виде векторов координат. Данные о весе рёбер хранятся в массиве masWeight. Функция drawGraph выполняется в цикле, пока окно открыто. Внутри цикла обрабатываются события, такие как закрытие окна. После обработки событий очищается фон окна и рисуются элементы графа: окружности, линии и текстовые данные. После рисования всех элементов выполняется перерисовка окна с помощью метода display.
4. Функция drawDFS отвечает за отрисовку графа в глубину (DFS). Она работает в цикле, пока окно приложения открыто. Каждый раз, когда пользователь нажимает клавишу Enter, функция добавляет новую окружность и её номер к списку circles и numbers, соответственно. Также добавляется текст с весом ребра к списку weights. Эти элементы затем отрисовываются на экране. Цикл for используется для рисования всех окружностей, номеров вершин и весов рёбер. Ещё один цикл for используется для рисования линий, соединяющих вершины.
5. Функция drawBFS выполняет аналогичную роль, что и drawDFS, но она отвечает за обход графа в ширину (BFS). Эта функция также работает в цикле, пока окно приложения открыто, и реагирует на события, такие как закрытие окна или нажатие клавиши Enter. При каждом нажатии Enter функция добавляет новые элементы графа (окружности, номера вершин и веса рёбер) в соответствующие списки и отрисовывает их на экране.
6. Функция `drawDijkstra() рисует граф и результаты алгоритма Дейкстры, используя SFML. Она создает окно, рисует вершины как круги с номерами, отображает расстояния и выводит текстовую информацию о кратчайших путях от выбранной вершины. В целом, функция визуализирует результат алгоритма Дейкстры.
7. Функция drawMatrix отвечает за отрисовку матрицы смежности. Она создает новое окно для рисования, загружает шрифт "ArialRegular.ttf" и устанавливает фоновый цвет окна. Затем она входит в бесконечный цикл, в котором проверяет события, связанные с окном, включая его закрытие. Внутри этого цикла функция очищает окно, рисует текст с названием "Матрица смежности" и затем рисует семь текстовых строк, представляющих элементы матрицы смежности. После отрисовки каждого текста функция вызывает window.display() для обновления экрана.
8. Класс Menu представляет собой структуру для создания и управления меню в графическом интерфейсе. Он содержит следующие атрибуты:  
   - menuX и menuY: координаты меню по осям X и Y соответственно.  
   - menuStep: расстояние между пунктами меню.  
   - maxMenu: максимальное количество пунктов меню.  
   - sizeFont: размер шрифта для пунктов меню.  
   - mainMenuSelected: номер текущего выбранного пункта меню.  
   - font: шрифт, используемый для пунктов меню.  
   - mainMenu: динамический массив текстовых объектов, представляющих названия пунктов меню.  
   - menuTextColor, choseTextColor, borderColor: цвета текста, выбранного текста и обводки пунктов меню соответственно.  
   - mywindow: ссылка на графическое окно, в котором будет отображаться меню.  
   Класс предоставляет следующие методы:  
   - setInitText: метод для настройки текста пунктов меню.  
   - Menu: конструктор класса, который инициализирует все атрибуты и создает пункты меню.  
   - ~Menu: деструктор, который освобождает память, выделенную для текстовых объектов пунктов меню.  
   - draw: метод для рисовки меню на экране.  
   - MoveUp и MoveDown: методы для перемещения выбора меню вверх или вниз соответственно.  
   - setColorTextMenu: метод для установки цветов элементов меню.  
   - alignMenu: метод для выравнивания положения меню.  
   - getSelectedMenuNumber: метод для получения номера выбранного элемента меню.
9. UML диаграмма:



1. Код программы:

Graphs.h:

#pragma once

#include <vector>

#include <iostream>

#include <queue>

using namespace std;

bool\* visitedVerts = new bool[20];

template <class T>

class Graph {

vector<int> vertList;

vector<vector<int>> adjMatrix;

int maxSize;

queue<T> VertsQueue;

vector<T> labelList;

public:

Graph(const int& size); // конструктор

~Graph() { } // деструктор

int GetVertPos(const T& vertex); // нахождение позиции вершины в векторе вершин

bool IsEmpty(); // проверка, пуст ли граф

bool IsFull(); // проверка, полон ли граф

int GetAmountVerts(); // возвращает количество вершин в графе

int GetAmountEdges(); // возвращает количество ребер в графе

int GetWeight(const T& vertex1, const T& vertex2); // получить вес

vector<T> GetNbrs(const T& vertex); // получить соседей

void InsertVertex(const T& vertex); // добавление вершины

void InsertEdge(const T& vertex1, const T& vertex2, int weight); // вставляет между вершинами ребро

void Print(); // печать матрицы смежности графа

void DFS(T& startVertex, bool\* visitedVerts); // обход графа в глубину

void BFS(T& startVertex, bool\* visitedVerts); // обход графа в ширину

bool AllVisited(bool\* visitedVerts); // проверка, обработаны ли все вершины

void FillLabels(T& startVertex); // заполнение очереди вершинами

int Dijkstra(T& startVertex); // алгоритм Дейкстры

};

template<class T>

Graph<T>::Graph(const int& size) { // конструктор

this->maxSize = size; // максимальный размер графа

this->adjMatrix = vector<vector<T>>(size, vector<T>(size));

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

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

this->adjMatrix[i][j] = 0; // по умолчанию устанавливаются все 0

}

}

}

template<class T>

int Graph<T>::GetVertPos(const T& vertex) { // нахождение позиции вершины в векторе вершин

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

if (this->vertList[i] == vertex) // если найдено

return i;

}

return -1;

}

template<class T>

bool Graph<T>::IsEmpty() { // проверка, пуст ли граф

if (this->vertList.size() != 0) return false;

else

return true;

}

template<class T>

bool Graph<T>::IsFull() { // проверка, полон ли граф

return (this->vertList.size() == maxSize);

}

template<class T>

int Graph<T>::GetAmountVerts() { // возвращает количество вершин в графе

return this->vertList.size();

}

template<class T>

int Graph<T>::GetAmountEdges() { // возвращает количество ребер в графе

int amount = 0; // обнуление счетчика

if (!this->IsEmpty()) { // проверка, не пуст ли граф

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i) {

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

if (this->adjMatrix[i][j] == this->adjMatrix[j][i] && this->adjMatrix[i][j] != 0) // нахождение ребер

amount += 1; // счет количества ребер

}

}

return (amount / 2); // возврат количества ребер

}

else

return 0; // если граф пуст, возврат 0

}

template<class T>

int Graph<T>::GetWeight(const T& vertex1, const T& vertex2) { // получить вес ребра

if (!this->IsEmpty()) {

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

return adjMatrix[vertPos1][vertPos2];

}

return 0;

}

template<class T>

vector<T> Graph<T>::GetNbrs(const T& vertex) { // получить соседей

vector<T> nbrsList; // создание списка соседей

int pos = this->GetVertPos(vertex); // вычисление позиции vertex в матрице смежности

if (pos != (-1)) { // проверка, что vertex есть в матрице смежности

for (int i = 0, vertListSize = this -> vertList.size(); i < vertListSize; ++i) {

if (this->adjMatrix[pos][i] != 0) // вычисление соседей

nbrsList.push\_back(this->vertList[i]); // запись соседей в вектор

}

}

return nbrsList; // возврат списка соседей

}

template<class T>

void Graph<T>::InsertVertex(const T& vertex) { // добавление вершины

if (!this->IsFull()) {

this->vertList.push\_back(vertex); // добавление вершин в вектор вершин

}

else {

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

return;

}

}

template<class T>

void Graph<T>::InsertEdge(const T& vertex1, const T& vertex2, int weight) { // вставляет между вершинами ребро

if (this->GetVertPos(vertex1) != (-1) && this->GetVertPos(vertex2) != (-1)) {

int vertPos1 = GetVertPos(vertex1); // переменная с номером вершины

int vertPos2 = GetVertPos(vertex2); // переменная с номером вершины

if (this->adjMatrix[vertPos1][vertPos2] != 0 && this->adjMatrix[vertPos2][vertPos1] != 0) {

cout << "Ребро между вершинами уже есть" << endl;

return;

}

else {

this->adjMatrix[vertPos1][vertPos2] = weight; // установка ребра

this->adjMatrix[vertPos2][vertPos1] = weight; // установка ребра

}

}

else {

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

return;

}

}

template<class T>

void Graph<T>::Print() { // печать матрицы смежности

if (!this->IsEmpty()) {

cout << "Матрица смежности графа: " << endl;

for (int i = 0, vertListSize = this -> vertList.size(); i < vertListSize; ++i) {

cout << this->vertList[i] << " ";

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

cout << " " << this-> adjMatrix[i][j] << " ";

}

cout << endl;

}

}

else {

cout << "Граф пуст " << endl;

}

}

template<class T>

void Graph<T>::DFS(T& startVertex, bool\* visitedVerts) { // обход графа в глубину

cout << "Вершина " << startVertex << " посещена" << endl;

visitedVerts[this->GetVertPos(startVertex)] = true; // пометка в массиве посещенных вершин

vector<T> neighbors = this -> GetNbrs(startVertex); // вектор соседей

for (int i = 0, size = this -> GetNbrs(startVertex).size(); i < size; ++i) {

if (visitedVerts[this -> GetVertPos(neighbors[i])] != true) // проверка, не посещены ли соседи

this->DFS(neighbors[i], visitedVerts);

}

}

template<class T>

void Graph<T>::BFS(T& startVertex, bool \* visitedVerts) { // обход графа в ширину

if (visitedVerts[this->GetVertPos(startVertex)] == false) { // проверка, была ли ранее посещена вершина startVertex

this->VertsQueue.push(startVertex);

cout << "Вершина " << startVertex << " обработана" << endl;

visitedVerts[this->GetVertPos(startVertex)] = true;

}

vector<T> neighbors = this->GetNbrs(startVertex); // вектор соседей

if (!VertsQueue.empty())

this->VertsQueue.pop(); // удаление головы очереди

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

if (visitedVerts[this->GetVertPos(neighbors[i])] != true) {

this->VertsQueue.push(neighbors[i]);

visitedVerts[this->GetVertPos(neighbors[i])] = true; // пометка в векторе о том, что вершина обработана

cout << "Вершина " << neighbors[i] << " обработана" << endl;

}

}

while (!this->VertsQueue.empty()) {

BFS(this->VertsQueue.front(), visitedVerts); // повторение цикла для остальных вершин

}

}

template<class T>

bool Graph<T>::AllVisited(bool\* visitedVerts) { // проверка, обработаны ли все вершины

bool flag = true;

for (int i = 0, size = this->vertList.size(); i < size; ++i) { // если в массиве посещенных вершин есть хотя бы одно значение false

if (visitedVerts[i] != true) flag = false;

}

if (flag == false) // еще не все вершины посещены

return false;

else return true; // все вершины посещены

}

template<class T>

void Graph<T>::FillLabels(T& startVertex) { // заполнение очереди вершинами

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

this->labelList.push\_back(1000000); // изначально все расстояние равны недостижимо большому числу

}

int pos = this->GetVertPos(startVertex);

(this->labelList)[pos] = 0;

}

template<class T>

int Graph<T>::Dijkstra(T& startVertex) { // алгоритм Дейкстры

for (int i = 0, size = this->vertList.size(); i < size; ++i) {

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

if (this->adjMatrix[i][j] < 0) // проверка, нет ли ребер с отрицательным весом

return -1;

}

}

T curSrc;

int counter = 0;

int minLabel = 1000000;

vector<T> neighbors = this -> GetNbrs(startVertex); // вектор из соседей текущей начальной вершины

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

if (this->labelList[this->GetVertPos(startVertex)] + this->GetWeight(startVertex, neighbors[i]) <

this->labelList[this->GetVertPos(neighbors[i])]) { // проверка, есть ли в векторе меток метка по индексу соседа в векторе вершин

this->labelList[this->GetVertPos(neighbors[i])] = this->GetWeight(startVertex, neighbors[i]);

}

if (this->labelList[this->GetVertPos(neighbors[i])]

< minLabel)

minLabel = this->labelList[this->GetVertPos(neighbors[i])]; // наименьшая метка у соседних опорной вершине вершин

}

for (int i = 0; i < neighbors.size(); ++i) { // проверка, рассмотрели ли всех соседей

if (this->labelList[this -> GetVertPos(neighbors[i])] != 1000000)

counter += 1;

}

if (counter == neighbors.size())

visitedVerts[this->GetVertPos(startVertex)] = true; // начальная вершина теперь считается посещенной

for (int i = 0; i < neighbors.size(); ++i) { // поиск новуй опорной вершины, у которой из всех соседей метка наименьшая

if (this->labelList[this->GetVertPos(neighbors[i])] == minLabel)

curSrc = neighbors[i];

}

neighbors = this->GetNbrs(curSrc);

while (!AllVisited(visitedVerts)) { // пока все вершины не будут считаться посещенными

int count = 0;

minLabel = 10000;

for (int i = 0; i < neighbors.size(); ++i) { // проверка, была ли вершина-соседуже рассмотрена ранее

if (visitedVerts[this->GetVertPos(neighbors[i])] != true) {

if (this->labelList[this->GetVertPos(curSrc)] + GetWeight(curSrc, neighbors[i]) < this->labelList[this->GetVertPos(neighbors[i])]) {

this->labelList[this->GetVertPos(neighbors[i])] = (this->labelList[this->GetVertPos(curSrc)]

+ this->GetWeight(curSrc, neighbors[i]));

}

if (this->labelList[this->GetVertPos(neighbors[i])] < minLabel) {

minLabel = this->labelList[this->GetVertPos(neighbors[i])];

}

}

count += 1; // счет соседей

}

if (count == neighbors.size()) // проверка, посетили ли всех соседей текущей опорной вершины

visitedVerts[this->GetVertPos(curSrc)] = true; // текущая опорная вершина отмечается как посещенная

for (int i = 0; i < neighbors.size(); ++i) { // поиск среди соседей новую опорную вершину

if (this->labelList[this->GetVertPos(neighbors[i])] == minLabel)

curSrc = neighbors[i];

}

neighbors = this->GetNbrs(curSrc); // занесение в вектор соседей соседей новой опорной вершины

}

for (int i = 0; i < this->GetVertPos(startVertex); ++i) {

cout << "Кратчайшее расстояние от вершины " << startVertex << " до вершины " << this->vertList[i] <<

" равно " << this->labelList[this ->GetVertPos(this->vertList[i])] << endl;

}

for (int i = this->GetVertPos(startVertex) + 1; i < this->vertList.size(); ++i) {

cout << "Кратчайшее расстояние от вершины " << startVertex << " до вершины " << this->vertList[i] <<

" равно " << this->labelList[this -> GetVertPos(this->vertList[i])] << endl;

}

}

Menu.h:

#pragma once

#include <SFML/Graphics.hpp>

using namespace sf;

class Menu {

float menuX; // координаты меню по x

float menuY; // координаты меню по y

int menuStep; // расстояние между пунктами меню

int maxMenu; // максимальное количество пунктов меню

int sizeFont; // размер шрифта

int mainMenuSelected; // номер текущего пункта меню

sf::Font font; // шрифт меню

sf::Text\* mainMenu; // динамический массив текстовых объектов названий пунктов меню

sf::Color menuTextColor = sf::Color::White; // цвет пунктов меню

sf::Color choseTextColor = sf::Color::Yellow; // цвет выбора пункта меню

sf::Color borderColor = sf::Color::Black; // цвет обводки текста пунктов меню

sf::RenderWindow& mywindow; // ссылка на графическое окно

void setInitText(sf::Text& text, sf::String str, float xpos, float ypos); // настройка текста пунктов меню

public:

Menu(sf::RenderWindow& window, float menux, float menuy,

int index, sf::String name[], int sizeFont = 60, int step = 80); // конструктор

~Menu() { // деструктор

delete[] mainMenu;

}

void draw(); // рисовка меню

void MoveUp(); // перемещение выбора меню вверх

void MoveDown(); // перемещение выбора меню вниз

void setColorTextMenu(sf::Color menColor, sf::Color ChoColor,

sf::Color BordColor); // цвет элементов игрового меню

void alignMenu(int posx); // выравнивание положения меню

int getSelectedMenuNumber() { // возвращает номер выбранного элемента меню

return mainMenuSelected;

}

};

Menu.cpp:

#include "Menu.h"

void Menu::setInitText(sf::Text& text, sf::String str, float xpos, float ypos) { //настройки текстовых объектов пунктов игрового меню

text.setFont(font); // шрифт

text.setFillColor(menuTextColor); // цвет

text.setString(str); // текст

text.setCharacterSize(sizeFont); // размер шрифта

text.setPosition(xpos, ypos); // координаты текстового объекта

text.setOutlineThickness(3); // толщина контура обводки текста

text.setOutlineColor(borderColor); // цвет контура обводки текста

}

void Menu::alignMenu(int posx) { // выравнивание пунктов меню по левому по правому по центру

float nullx = 0;

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

switch (posx)

{

case 0:

nullx = 0; // выравнивание по правому краю от установленных координат

break;

case 1:

nullx = mainMenu[i].getLocalBounds().width; // по левому краю

break;

case 2:

nullx = nullx = mainMenu[i].getLocalBounds().width / 2; // по центру

break;

}

mainMenu[i].setPosition(mainMenu[i].getPosition().x - nullx, mainMenu[i].getPosition().y);

}

}

Menu::Menu(sf::RenderWindow& window, float menux, float menuy,

int index, sf::String name[], int sizeFont, int step)

:mywindow(window), menuX(menux), menuY(menuy), sizeFont(sizeFont), menuStep(step) {

// Загрузка шрифта

if (!font.loadFromFile("ArialRegular.ttf")) exit(32); // загрузка шрифта

maxMenu = index; // количество элементов меню

mainMenu = new sf::Text[maxMenu]; // динамический массив пунктов меню

for (int i = 0, ypos = menuY; i < maxMenu; i++, ypos += menuStep) // выстраивание элементов меню

setInitText(mainMenu[i], name[i], menuX, ypos);

mainMenuSelected = 0; // задание начальное положения выбраного пункта меню

mainMenu[mainMenuSelected].setFillColor(choseTextColor); // цвет выбраного пункта меню

}

void Menu::MoveUp()

{

mainMenuSelected--;

if (mainMenuSelected >= 0) { // подсвечивание выбранного пункта меню

mainMenu[mainMenuSelected].setFillColor(choseTextColor);

mainMenu[mainMenuSelected + 1].setFillColor(menuTextColor);

}

else

{

mainMenu[0].setFillColor(menuTextColor);

mainMenuSelected = maxMenu - 1;

mainMenu[mainMenuSelected].setFillColor(choseTextColor);

}

}

void Menu::MoveDown() {

mainMenuSelected++;

if (mainMenuSelected < maxMenu) { // подсвечивание выбранного пункта меню

mainMenu[mainMenuSelected - 1].setFillColor(menuTextColor);

mainMenu[mainMenuSelected].setFillColor(choseTextColor);

}

else

{

mainMenu[maxMenu - 1].setFillColor(menuTextColor);

mainMenuSelected = 0;

mainMenu[mainMenuSelected].setFillColor(choseTextColor);

}

}

void Menu::draw() {

for (int i = 0; i < maxMenu; i++) mywindow.draw(mainMenu[i]); // перебор для отрисовки существующих текстовых объектов пунктов меню

}

void Menu::setColorTextMenu(sf::Color menColor, sf::Color ChoColor, sf::Color BordColor) {

menuTextColor = menColor; // цвет пунктов меню

choseTextColor = ChoColor; // цвет выбраного пункта меню

borderColor = BordColor; // цвет контура пунктов меню

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

mainMenu[i].setFillColor(menuTextColor);

mainMenu[i].setOutlineColor(borderColor);

}

mainMenu[mainMenuSelected].setFillColor(choseTextColor);

}

WorkButton.h:

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Window.hpp>

using namespace sf;

void drawGraph() { // отрисовка графа

RenderWindow window(VideoMode(1300, 700), L"Граф", sf::Style::Close);

Font font;

if (!font.loadFromFile("ArialRegular.ttf")) exit(32); // установка шрифта для названия

sf::Color backgroundСolor(255, 192, 203);

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) { // событие для закрытия окна

window.close();

}

}

window.clear(backgroundСolor); // установка фона

sf::Vector2f positions[] = { // позиции для каждой окружности

sf::Vector2f(200.f, 50.f),

sf::Vector2f(280.f, 200.f),

sf::Vector2f(360.f, 350.f),

sf::Vector2f(550.f, 490.f),

sf::Vector2f(540.f, 10.f),

sf::Vector2f(750.f, 100.f),

sf::Vector2f(900.f, 300.f)

};

int masData[7] = { 7, 6, 4, 2, 1 , 3, 5 };

for (int i = 0; i < 7; ++i) {

sf::CircleShape circle(50.f); // создание окружности

circle.setPosition(positions[i]);

circle.setFillColor(sf::Color(245, 245, 220));

string data = to\_string(masData[i]);

sf::Text text(data, font, 24); // устновка данных и их параметров

text.setPosition(positions[i].x + 30, positions[i].y + 30);

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

}

sf::Vector2f positionsS[] = { // позиции начала линий, соединяющих окружности

sf::Vector2f(275.f, 145.f),

sf::Vector2f(350.f, 295.f),

sf::Vector2f(440.f, 440.f),

sf::Vector2f(640.f, 510.f),

sf::Vector2f(540.f, 70.f),

sf::Vector2f(755.f, 130.f),

sf::Vector2f(940.f, 305.f),

sf::Vector2f(295.f, 120.f),

sf::Vector2f(375.f, 250.f),

sf::Vector2f(600.f, 490.f),

};

sf::Vector2f positionsS2[] = { // позиции конца линий, соединяющих окружности

sf::Vector2f(310.f, 208.f),

sf::Vector2f(390.f, 360.f),

sf::Vector2f(560.f, 510.f),

sf::Vector2f(910.f, 380.f),

sf::Vector2f(300.f, 90.f),

sf::Vector2f(630.f, 80.f),

sf::Vector2f(830.f, 190.f),

sf::Vector2f(900.f, 340.f),

sf::Vector2f(750.f, 160.f),

sf::Vector2f(590.f, 110.f),

};

sf::Vector2f positionsForWeight[] = { // позиции конца линий, соединяющих окружности

sf::Vector2f(260.f, 170.f),

sf::Vector2f(350.f, 330.f),

sf::Vector2f(485.f, 480.f),

sf::Vector2f(780.f, 440.f),

sf::Vector2f(440.f, 50.f),

sf::Vector2f(690.f, 80.f),

sf::Vector2f(875.f, 190.f),

sf::Vector2f(320.f, 100.f),

sf::Vector2f(450.f, 240.f),

sf::Vector2f(610.f, 110.f),

};

int masWeight[10] = { 26, 9, 23, 19, 12 , 19, 34, 47, 30, 42 };

for (size\_t i = 0; i < 10; ++i) { // добавление линий между окружностями

sf::Vertex line[] = {

sf::Vertex(positionsS[i]),

sf::Vertex(positionsS2[i]),

};

string weight = to\_string(masWeight[i]);

sf::Text text(weight, font, 24); // устновка данных и их параметров

text.setPosition(positionsForWeight[i].x, positionsForWeight[i].y);

text.setFillColor(sf::Color::Black);

window.draw(text);

window.draw(line, 2, sf::Lines);

}

window.display(); // отображение на экране рисунков

}

}

void drawDFS() { // обход графа в глубину

sf::RenderWindow window(sf::VideoMode(1300, 700), L"Обход графа в глубину"); // создание окна

sf::Color backgroundСolor(255, 192, 203);

Font font;

if (!font.loadFromFile("ArialRegular.ttf")) exit(32); // установка шрифта для названия

vector<CircleShape> circles; // вектор для хранения всех окружностей

vector<Text> numbers; // вектор для хранения всех номеров вершин

vector<Vertex> lines; // вектор для хранения всех координат начала линий

vector<Vertex> Lines; // вектор для хранения всех координат конца линий

vector<Text> weights; // вектор для хранения всех весов

int masData[7] = { 1, 2, 3, 4, 5 , 6, 7 };

sf::Vector2f positions[] = { // позиции для каждой окружности

sf::Vector2f(540.f, 10.f),

sf::Vector2f(550.f, 490.f),

sf::Vector2f(750.f, 100.f),

sf::Vector2f(360.f, 350.f),

sf::Vector2f(900.f, 300.f),

sf::Vector2f(280.f, 200.f),

sf::Vector2f(200.f, 50.f),

};

sf::Vector2f positionsForWeight[] = { // позиции конца линий, соединяющих окружности

sf::Vector2f(260.f, 170.f),

sf::Vector2f(350.f, 330.f),

sf::Vector2f(485.f, 480.f),

sf::Vector2f(780.f, 440.f),

sf::Vector2f(440.f, 50.f),

sf::Vector2f(690.f, 80.f),

sf::Vector2f(875.f, 190.f),

sf::Vector2f(320.f, 100.f),

sf::Vector2f(450.f, 240.f),

sf::Vector2f(610.f, 110.f),

};

sf::Vector2f positionsS[] = { // позиции начала линий, соединяющих окружности

sf::Vector2f(375.f, 250.f),

sf::Vector2f(755.f, 130.f),

sf::Vector2f(600.f, 490.f),

sf::Vector2f(440.f, 440.f),

sf::Vector2f(940.f, 305.f),

sf::Vector2f(295.f, 120.f),

sf::Vector2f(275.f, 145.f),

sf::Vector2f(350.f, 295.f),

sf::Vector2f(640.f, 510.f),

sf::Vector2f(540.f, 70.f),

};

sf::Vector2f positionsS2[] = { // позиции конца линий, соединяющих окружности

sf::Vector2f(750.f, 160.f),

sf::Vector2f(630.f, 80.f),

sf::Vector2f(590.f, 110.f),

sf::Vector2f(560.f, 510.f),

sf::Vector2f(830.f, 190.f),

sf::Vector2f(900.f, 340.f),

sf::Vector2f(310.f, 208.f),

sf::Vector2f(390.f, 360.f),

sf::Vector2f(910.f, 380.f),

sf::Vector2f(300.f, 90.f),

};

int masWeight[10] = { 26, 9, 23, 19, 12 , 19, 34, 47, 30, 42 };

int order[7] = { 6,3,1,2,4,5,7 };

int cnt = 0;

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

window.close();

}

else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Enter && cnt < 10) { // рисование при нажатии Enter

if (cnt < 7) {

sf::CircleShape circle(50.f); // размер окружности

circle.setFillColor(sf::Color(245, 245, 220)); // цвет заливки

circle.setPosition(positions[order[cnt]-1].x, positions[order[cnt]-1].y); // позиция окружности

circle.setOutlineThickness(4);

circles.push\_back(circle); // добавление окружности в список

string data = to\_string(masData[order[cnt]-1]);

sf::Text text(data, font, 24); // устновка данных и их параметров

text.setPosition(positions[order[cnt]-1].x + 30, positions[order[cnt]-1].y + 30);

text.setFillColor(sf::Color::Black);

numbers.push\_back(text);

}

string weight = to\_string(masWeight[cnt]);

sf::Text text(weight, font, 24); // устновка данных и их параметров

text.setPosition(positionsForWeight[cnt].x, positionsForWeight[cnt].y);

text.setFillColor(sf::Color::Black);

weights.push\_back(text);

lines.push\_back(positionsS[cnt]);

Lines.push\_back(positionsS2[cnt]);

cnt++;

}

}

window.clear(backgroundСolor); // установка фона

for (const auto& circle : circles) { // рисование окружностей

window.draw(circle);

}

for (const auto& number : numbers) { // рисование номеров вершин

window.draw(number);

}

for (const auto& text : weights) { // рисование весов

window.draw(text);

}

for (size\_t i = 0; i < lines.size(); ++i) { // рисование линий

const Vertex& linE = lines[i];

const Vertex& linE2 = Lines[i];

sf::Vertex line0[] = {

{linE},

{linE2}

};

window.draw(line0, 2, sf::Lines);

}

sf::Text text1;

text1.setFont(font);

text1.setString(L"Порядок посещенных вершин: ");

text1.setFillColor(sf::Color::Black);

text1.setCharacterSize(30);

text1.setPosition(450, 600);

window.draw(text1);

sf::Text text2;

text2.setFont(font);

text2.setString(L"6->3->1->2->4->5->7");

text2.setFillColor(sf::Color::Black);

text2.setCharacterSize(30);

text2.setPosition(900, 600);

window.draw(text2);

window.display();

}

}

void drawBFS() { // обход графа в ширину

sf::RenderWindow window(sf::VideoMode(1300, 700), L"Обход графа в ширину"); // создание окна

sf::Color backgroundСolor(255, 192, 203);

Font font;

if (!font.loadFromFile("ArialRegular.ttf")) exit(32); // установка шрифта для названия

vector<CircleShape> circles; // вектор для хранения всех окружностей

vector<Text> numbers; // вектор для хранения всех номеров вершин

vector<Vertex> lines; // вектор для хранения всех координат начала линий

vector<Vertex> Lines; // вектор для хранения всех координат конца линий

vector<Text> weights; // вектор для хранения всех весов

int masData[7] = { 1, 2, 3, 4, 5 , 6, 7 };

sf::Vector2f positions[] = { // позиции для каждой окружности

sf::Vector2f(540.f, 10.f),

sf::Vector2f(550.f, 490.f),

sf::Vector2f(750.f, 100.f),

sf::Vector2f(360.f, 350.f),

sf::Vector2f(900.f, 300.f),

sf::Vector2f(280.f, 200.f),

sf::Vector2f(200.f, 50.f),

};

sf::Vector2f positionsForWeight[] = { // позиции конца линий, соединяющих окружности

sf::Vector2f(260.f, 170.f),

sf::Vector2f(350.f, 330.f),

sf::Vector2f(485.f, 480.f),

sf::Vector2f(780.f, 440.f),

sf::Vector2f(440.f, 50.f),

sf::Vector2f(690.f, 80.f),

sf::Vector2f(875.f, 190.f),

sf::Vector2f(320.f, 100.f),

sf::Vector2f(450.f, 240.f),

sf::Vector2f(610.f, 110.f),

};

sf::Vector2f positionsS[] = { // позиции начала линий, соединяющих окружности

sf::Vector2f(375.f, 250.f),

sf::Vector2f(350.f, 295.f),

sf::Vector2f(275.f, 145.f),

sf::Vector2f(540.f, 70.f),

sf::Vector2f(940.f, 305.f),

sf::Vector2f(440.f, 440.f),

sf::Vector2f(640.f, 510.f),

sf::Vector2f(755.f, 130.f),

sf::Vector2f(295.f, 120.f),

sf::Vector2f(600.f, 490.f),

};

sf::Vector2f positionsS2[] = { // позиции конца линий, соединяющих окружности

sf::Vector2f(750.f, 160.f),

sf::Vector2f(390.f, 360.f),

sf::Vector2f(310.f, 208.f),

sf::Vector2f(300.f, 90.f),

sf::Vector2f(830.f, 190.f),

sf::Vector2f(560.f, 510.f),

sf::Vector2f(910.f, 380.f),

sf::Vector2f(630.f, 80.f),

sf::Vector2f(900.f, 340.f),

sf::Vector2f(590.f, 110.f),

};

int masWeight[10] = { 26, 9, 23, 19, 12 , 19, 34, 47, 30, 42 };

int order[7] = { 6,3,4,7,1,5,2 };

int cnt = 0;

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

window.close();

}

else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Enter && cnt < 10) { // рисование при нажатии Enter

if (cnt < 7) {

sf::CircleShape circle(50.f); // размер окружности

circle.setFillColor(sf::Color(245, 245, 220)); // цвет заливки

circle.setPosition(positions[order[cnt] - 1].x, positions[order[cnt] - 1].y); // позиция окружности

circle.setOutlineThickness(4);

circles.push\_back(circle); // добавление окружности в список

string data = to\_string(masData[order[cnt] - 1]);

sf::Text text(data, font, 24); // устновка данных и их параметров

text.setPosition(positions[order[cnt] - 1].x + 30, positions[order[cnt] - 1].y + 30);

text.setFillColor(sf::Color::Black);

numbers.push\_back(text);

}

string weight = to\_string(masWeight[cnt]);

sf::Text text(weight, font, 24); // устновка данных и их параметров

text.setPosition(positionsForWeight[cnt].x, positionsForWeight[cnt].y);

text.setFillColor(sf::Color::Black);

weights.push\_back(text);

lines.push\_back(positionsS[cnt]);

Lines.push\_back(positionsS2[cnt]);

cnt++;

}

}

window.clear(backgroundСolor); // установка фона

for (const auto& circle : circles) { // рисование всех окружностей

window.draw(circle);

}

for (const auto& number : numbers) { // рисование всех номеров вершин

window.draw(number);

}

for (const auto& text : weights) { // рисование всех весов

window.draw(text);

}

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

const Vertex& linE = lines[i];

const Vertex& linE2 = Lines[i];

sf::Vertex line0[] = {

{linE},

{linE2}

};

window.draw(line0, 2, sf::Lines);

}

sf::Text text1;

text1.setFont(font);

text1.setString(L"Порядок посещенных вершин: ");

text1.setFillColor(sf::Color::Black);

text1.setCharacterSize(30);

text1.setPosition(450, 600);

window.draw(text1);

sf::Text text2;

text2.setFont(font);

text2.setString(L"6->3->4->7->1->5->2");

text2.setFillColor(sf::Color::Black);

text2.setCharacterSize(30);

text2.setPosition(900, 600);

window.draw(text2);

window.display();

}

return;

}

void drawDijkstra() { // алгоритм декстра

RenderWindow window(VideoMode(1500, 700), L"Алгоритм Дейкстры", sf::Style::Close);

Font font;

Font font2;

if (!font2.loadFromFile("a\_PlakatTitul ExtraBold.ttf")) exit(32); // установка шрифта для расстояний

if (!font.loadFromFile("ArialRegular.ttf")) exit(32); // установка шрифта для названия

sf::Color backgroundСolor(255, 192, 203);

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) { // событие для закрытия окна

window.close();

}

}

window.clear(backgroundСolor); // установка фона

sf::Vector2f positions[] = { // позиции для каждой окружности

sf::Vector2f(340.f, 100.f),

sf::Vector2f(350.f, 590.f),

sf::Vector2f(550.f, 200.f),

sf::Vector2f(160.f, 450.f),

sf::Vector2f(700.f, 400.f),

sf::Vector2f(80.f, 300.f),

sf::Vector2f(10.f, 150.f),

};

int masData[7] = { 1, 2, 3, 4, 5 , 6, 7 };

int masDistance[7] = { 38, 32, 30, 9, 51 , 0, 26 };

for (int i = 0; i < 7; ++i) {

sf::CircleShape circle(50.f); // создание окружности

circle.setPosition(positions[i]);

circle.setFillColor(sf::Color(245, 245, 220));

string data = to\_string(masData[i]);

sf::Text text(data, font, 24); // устновка данных и их параметров

text.setPosition(positions[i].x + 30, positions[i].y + 30);

text.setFillColor(sf::Color::Black);

string distance = to\_string(masDistance[i]);

sf::Text text2(distance, font2, 24); // устновка данных и их параметров

text2.setPosition(positions[i].x +30, positions[i].y - 40);

text2.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

window.draw(text2);

}

sf::Vector2f positionsS[] = { // позиции начала линий, соединяющих окружности

sf::Vector2f(75.f, 245.f),

sf::Vector2f(150.f, 395.f),

sf::Vector2f(240.f, 540.f),

sf::Vector2f(440.f, 610.f),

sf::Vector2f(340.f, 170.f),

sf::Vector2f(555.f, 230.f),

sf::Vector2f(740.f, 405.f),

sf::Vector2f(95.f, 220.f),

sf::Vector2f(175.f, 350.f),

sf::Vector2f(400.f, 590.f),

};

sf::Vector2f positionsS2[] = { // позиции конца линий, соединяющих окружности

sf::Vector2f(110.f, 308.f),

sf::Vector2f(190.f, 460.f),

sf::Vector2f(360.f, 610.f),

sf::Vector2f(710.f, 480.f),

sf::Vector2f(100.f, 190.f),

sf::Vector2f(430.f, 180.f),

sf::Vector2f(630.f, 290.f),

sf::Vector2f(700.f, 440.f),

sf::Vector2f(550.f, 260.f),

sf::Vector2f(390.f, 210.f),

};

sf::Vector2f positionsForWeight[] = { // позиции конца линий, соединяющих окружности

sf::Vector2f(60.f, 270.f),

sf::Vector2f(150.f, 430.f),

sf::Vector2f(285.f, 580.f),

sf::Vector2f(580.f, 540.f),

sf::Vector2f(240.f, 150.f),

sf::Vector2f(490.f, 180.f),

sf::Vector2f(675.f, 290.f),

sf::Vector2f(120.f, 200.f),

sf::Vector2f(250.f, 340.f),

sf::Vector2f(410.f, 210.f),

};

int masWeight[10] = { 26, 9, 23, 19, 12 , 19, 34, 47, 30, 42 };

for (size\_t i = 0; i < 10; ++i) { // добавление линий между окружностями

sf::Vertex line[] = {

sf::Vertex(positionsS[i]),

sf::Vertex(positionsS2[i]),

};

string weight = to\_string(masWeight[i]);

sf::Text text(weight, font, 24); // устновка данных и их параметров

text.setPosition(positionsForWeight[i].x, positionsForWeight[i].y);

text.setFillColor(sf::Color::Black);

window.draw(text);

window.draw(line, 2, sf::Lines);

}

sf::Text text(L"Кратчайшее расстояние от вершины 6 до вершины 1 равно 38", font, 24); // устновка данных и их параметров

text.setPosition(700, 10);

text.setFillColor(sf::Color::Black);

window.draw(text);

sf::Text text2(L"Кратчайшее расстояние от вершины 6 до вершины 2 равно 32", font, 24); // устновка данных и их параметров

text2.setPosition(700, 40);

text2.setFillColor(sf::Color::Black);

window.draw(text2);

sf::Text text3(L"Кратчайшее расстояние от вершины 6 до вершины 3 равно 30", font, 24); // устновка данных и их параметров

text3.setPosition(700, 70);

text3.setFillColor(sf::Color::Black);

window.draw(text3);

sf::Text text4(L"Кратчайшее расстояние от вершины 6 до вершины 4 равно 9", font, 24); // устновка данных и их параметров

text4.setPosition(700, 100);

text4.setFillColor(sf::Color::Black);

window.draw(text4);

sf::Text text5(L"Кратчайшее расстояние от вершины 6 до вершины 5 равно 51", font, 24); // устновка данных и их параметров

text5.setPosition(700, 130);

text5.setFillColor(sf::Color::Black);

window.draw(text5);

sf::Text text6(L"Кратчайшее расстояние от вершины 6 до вершины 7 равно 26", font, 24); // устновка данных и их параметров

text6.setPosition(700, 160);

text6.setFillColor(sf::Color::Black);

window.draw(text6);

window.display(); // отображение на экране рисунков

}

}

void drawMatrix() { // отрисовка матрицы смежности

RenderWindow window(VideoMode(600, 500), L"Матрица смежности", sf::Style::Close);

Font font;

if (!font.loadFromFile("ArialRegular.ttf")) exit(32); // установка шрифта для названия

sf::Color backgroundСolor(255, 192, 203);

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) { // событие для закрытия окна

window.close();

}

}

window.clear(backgroundСolor); // установка фона

sf::Text text(L"Матрица смежности", font, 50); // устновка данных и их параметров

text.setPosition(70, 50);

text.setFillColor(sf::Color::Black);

window.draw(text);

sf::Text text1(L"1 0 42 19 0 0 0 12", font, 35); // устновка данных и их параметров

text1.setPosition(170, 150);

text1.setFillColor(sf::Color::Black);

window.draw(text1);

sf::Text text2(L"2 42 0 0 23 19 0 0", font, 35); // устновка данных и их параметров

text2.setPosition(170, 190);

text2.setFillColor(sf::Color::Black);

window.draw(text2);

sf::Text text3(L"3 19 0 0 0 34 30 0", font, 35); // устновка данных и их параметров

text3.setPosition(170, 230);

text3.setFillColor(sf::Color::Black);

window.draw(text3);

sf::Text text4(L"4 0 23 0 0 0 9 0", font, 35); // устновка данных и их параметров

text4.setPosition(170, 270);

text4.setFillColor(sf::Color::Black);

window.draw(text4);

sf::Text text5(L"5 0 19 34 0 0 0 47", font, 35); // устновка данных и их параметров

text5.setPosition(170, 310);

text5.setFillColor(sf::Color::Black);

window.draw(text5);

sf::Text text6(L"6 0 0 30 9 0 0 26", font, 35); // устновка данных и их параметров

text6.setPosition(170, 350);

text6.setFillColor(sf::Color::Black);

window.draw(text6);

sf::Text text7(L"7 12 0 0 0 47 26 0", font, 35); // устновка данных и их параметров

text7.setPosition(170, 390);

text7.setFillColor(sf::Color::Black);

window.draw(text7);

window.display(); // отображение на экране рисунков

}

}

main.cpp:

#include "Graphs.h"

#include "Menu.h"

#include "WorkButton.h"

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Window.hpp>

#include <iostream>

using namespace std;

using namespace sf;

void InitText(Text& mtext, float xpos, float ypos, String str, int size\_font = 60,

Color menu\_text\_color = Color::White, int bord = 0, Color border\_color = Color::Black);

int main() {

system("chcp 1251 > Null");

RenderWindow window; // создание окна windows

window.create(VideoMode(1300, 900), L"Меню", sf::Style::Close); // параметры окна

float width = VideoMode::getDesktopMode().width; // получение текущего размера экрана

float height = VideoMode::getDesktopMode().height; // получение текущего размера экрана

RectangleShape background(Vector2f(width, height)); // создаем прямоугольник для изображения

Texture texture\_window;

if (!texture\_window.loadFromFile("koshka.jpg")) return 4; // загрузка в прямоугольник текстуру с изображением

background.setTexture(&texture\_window);

Font font;

if (!font.loadFromFile("ArialRegular.ttf")) return 5; // установка шрифта для названия

Text Titul;

Titul.setFont(font);

InitText(Titul, 590, 50, L"Граф", 70, Color(245, 245, 220), 3); // параметры для заголовка

String nameMenu[]{ L"Печать графа", L"Матрица смежности графа",L"Обход графа в глубину", L"Обход графа в ширину", L"Алгоритм Дейкстры", L"Выход"};

Menu mymenu(window, 650, 200, 6, nameMenu, 55, 120); // объект меню

mymenu.setColorTextMenu(sf::Color(245, 245, 220), Color::Magenta, Color::Black); // установка цвета элементов пунктов меню

mymenu.alignMenu(2); // выравнивание по центру пунктов меню

while (window.isOpen()) {

Event event;

while (window.pollEvent(event)) {

if (event.type == Event::KeyReleased) {

if (event.key.code == Keyboard::Up) { mymenu.MoveUp(); } // событие нажатия на клавиатуре стрелки вверх

if (event.key.code == Keyboard::Down) { mymenu.MoveDown(); } // событие нажатия на клавиатуре стрелки вниз

if (event.key.code == Keyboard::Return) { // событие нажатия на клавиатуре клавиши Enter

switch (mymenu.getSelectedMenuNumber()) {

case 0: drawGraph(); break;

case 1: drawMatrix(); break;

case 2: drawDFS(); break;

case 3: drawBFS(); break;

case 4: drawDijkstra(); break;

case 5: window.close();

}

}

}

}

window.draw(background);

window.draw(Titul);

mymenu.draw();

window.display();

}

return 0;

}

void InitText(Text& mtext, float xpos, float ypos, String str, int size\_font,

Color menu\_text\_color, int bord, Color border\_color) { // установка настроек

mtext.setCharacterSize(size\_font);

mtext.setPosition(xpos, ypos);

mtext.setString(str);

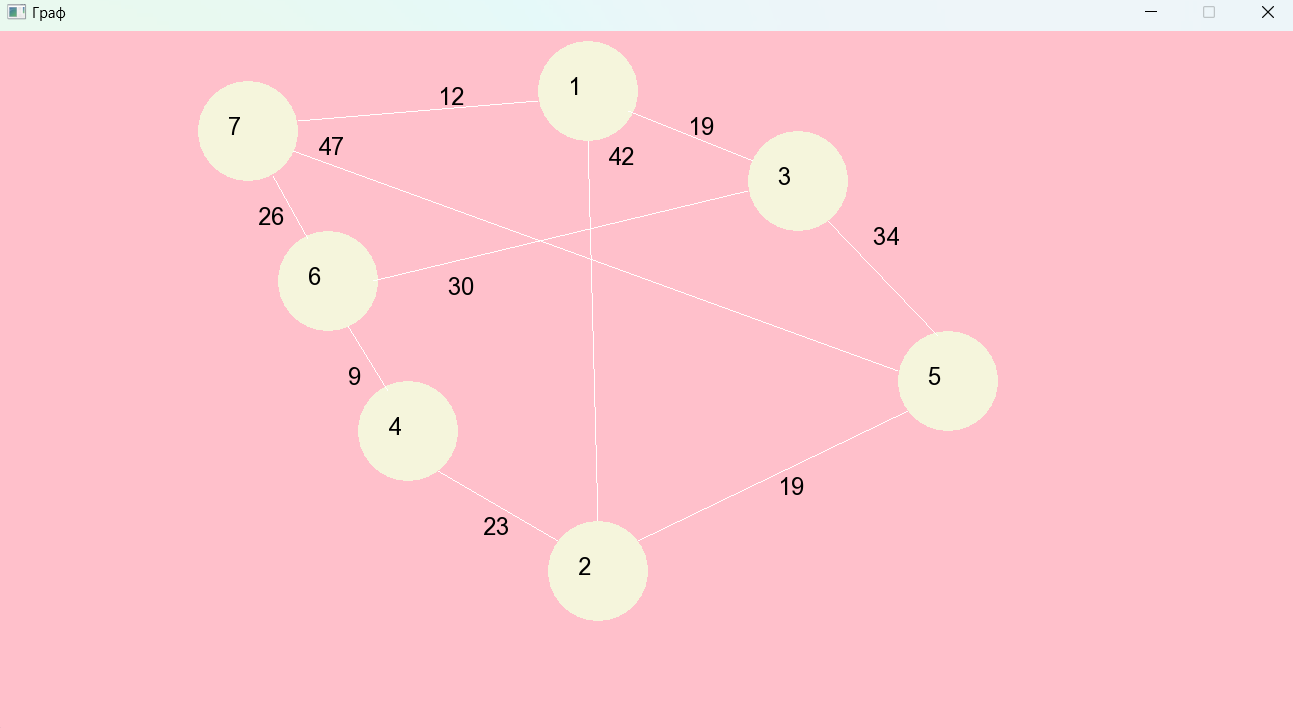
mtext.setFillColor(menu\_text\_color);

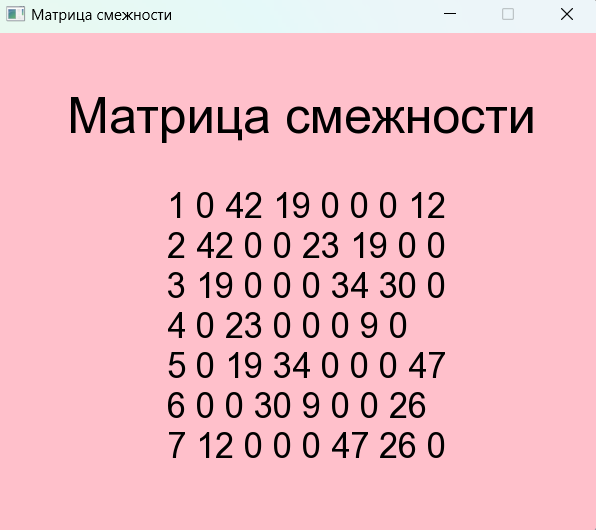
mtext.setOutlineThickness(bord);

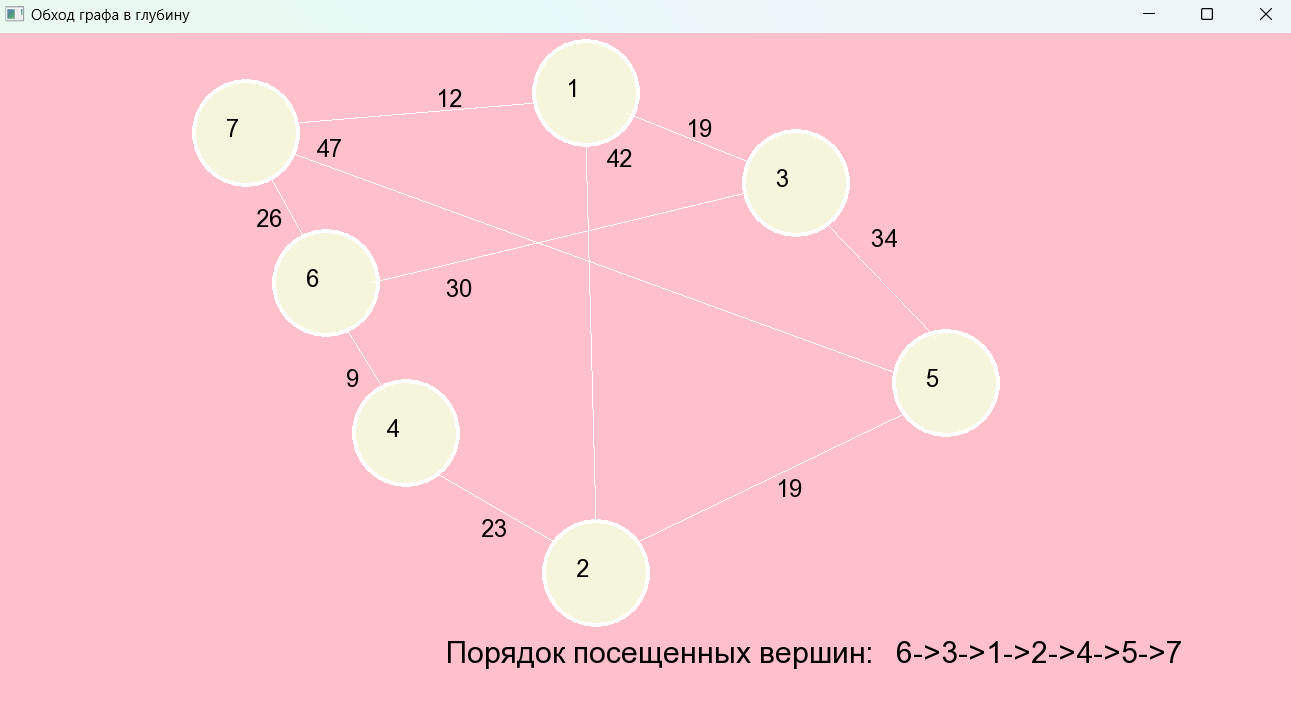
mtext.setOutlineColor(border\_color);

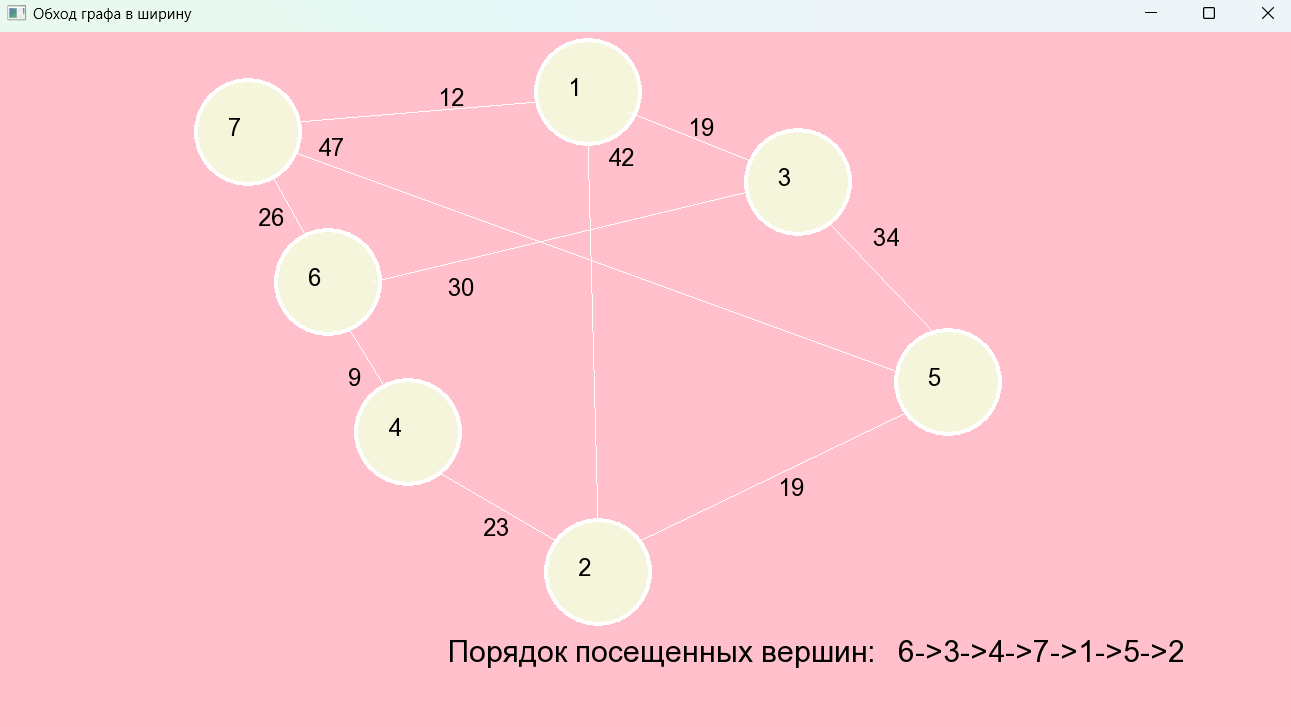
}

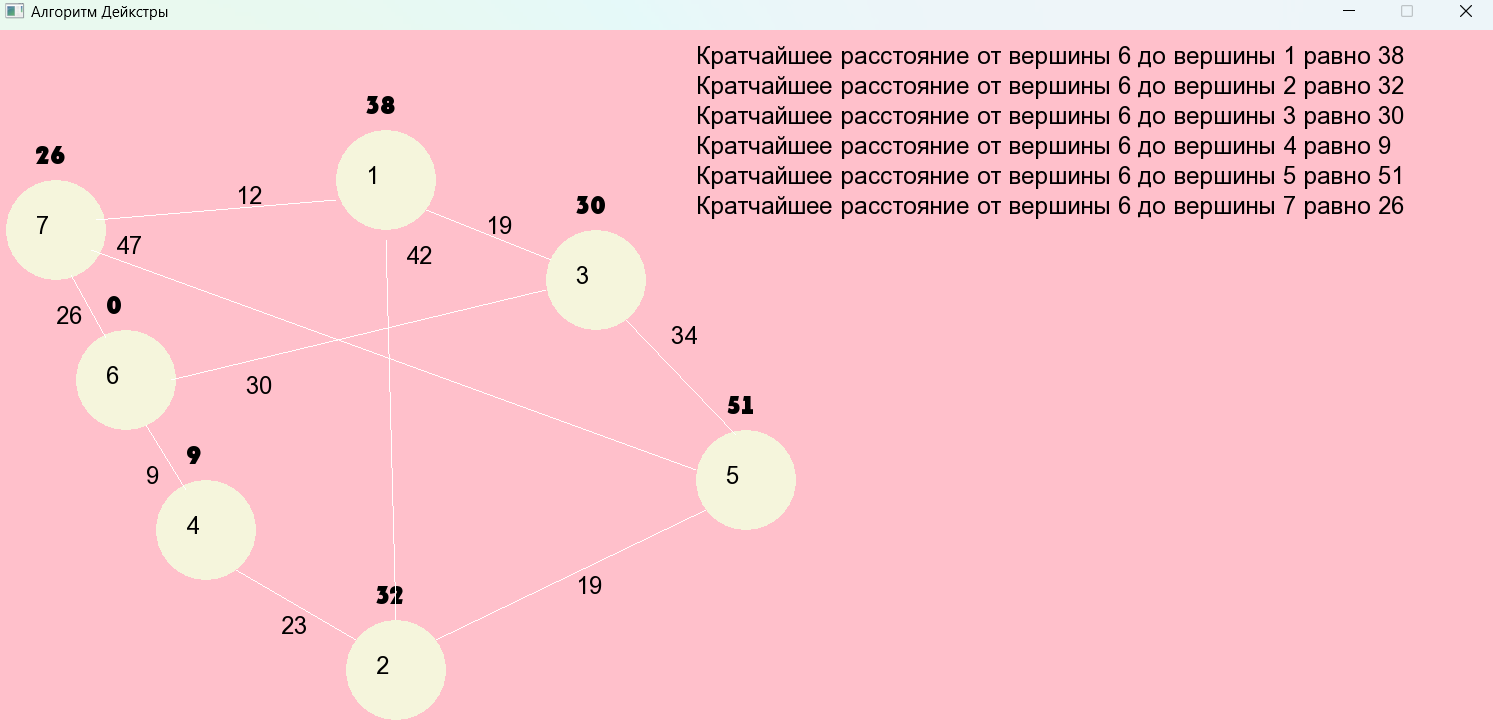
1. Результат работы программы:



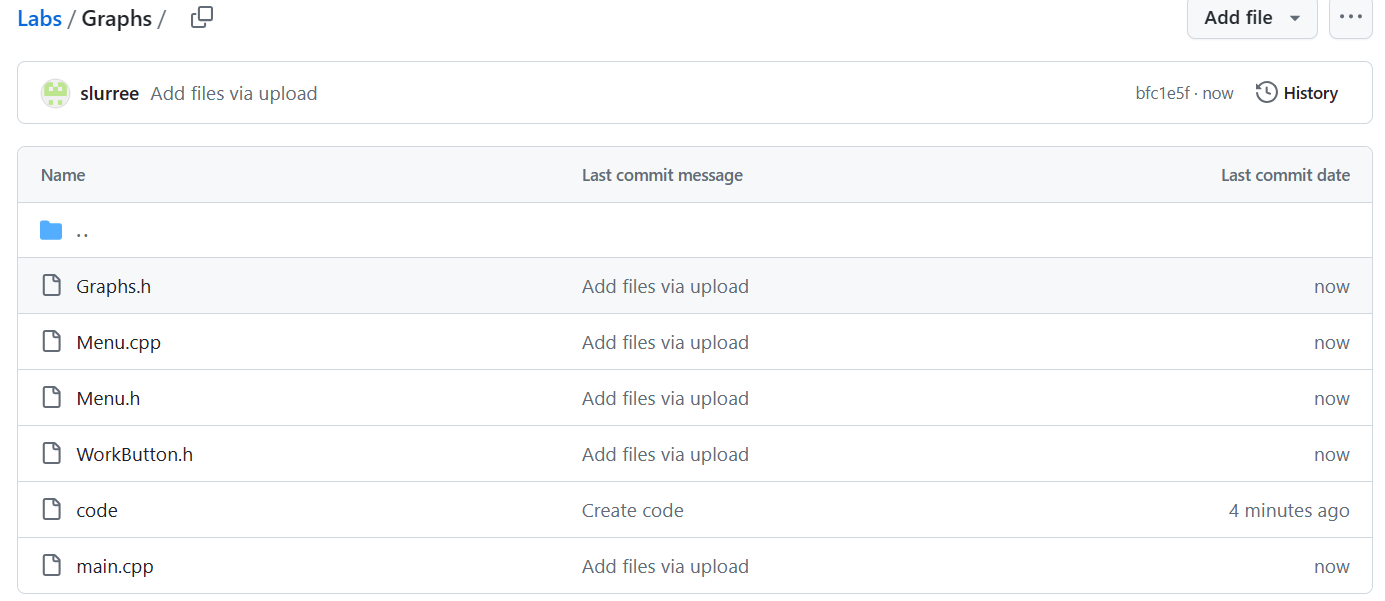








Вывод: В ходе выполнения лабораторной работы была реализована программа на языке программирования C++, использующая библиотеку SFML для визуализации графа. Программа позволяет пользователю задавать количество вершин и рёбер, а также их свойства. Каждое ребро графа представлено линией, соединяющей две вершины, а каждая вершина — кругом, внутри которого отображается её номер.

Ссылка на работу в GitHub: <https://github.com/slurree/Labs.git>