Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №21**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: “Графы”

Вариант 16

Выполнил:

Студент группы ИВТ-20-2Б Тедеев Алесандр Зурабович

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**

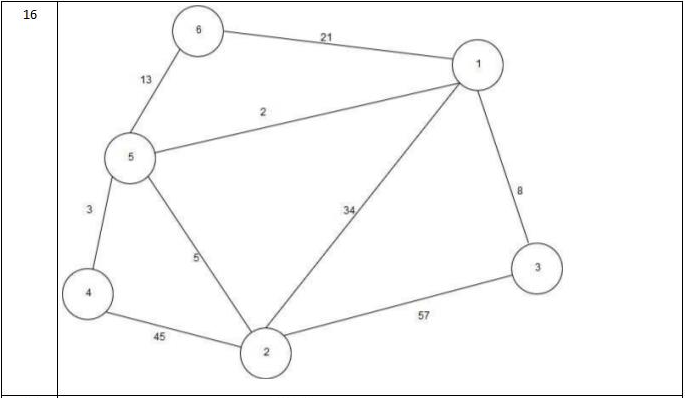
Реализовать алгоритм Дейкстры для графа, соответствующего выбранному варианту.

1. Без использования Qt.

2. Интерфейс на усмотрение разработчика.

3. Визуализация графа в OpenGL.

4. Реализуется только метод Дейкстры



**Анализ задачи**

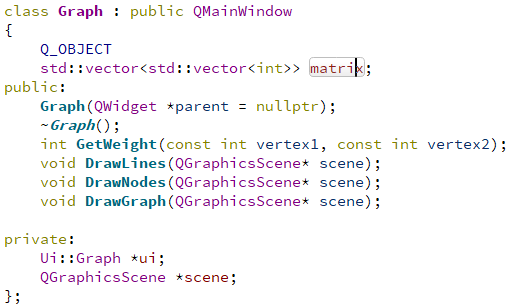
**1.** Для решения задач необходимо:

**1.1.** Организовать класс Graph, который наследуется от класса QMainWindow, чтобы была возможность использовать интерфейс.

**1.2.** Организовать функцию Dejkstra(), чтобы вывести кратчайшие пути до вершин в консоли.

**2.** В ходе работы были использованы следующие типы данных:

**2.1.** Класс Graph с необходимыми методами для отрисовки линий, узлов и самого графа в форму. В качестве поля используется двумерный вектор matrix и указатель класса QGraphicsScene - \*scene.



**2.2.** В главном файле main.cpp используется глобальные вектор mat, массив result, который будет использоваться для вывода ответа, массив visited для используемый для флажков.

vector <vector<int>> mat = { {0, 34, 8, 0, 2, 21 },

{24, 0, 57, 45, 5, 0 },

{8, 57, 0, 0, 0, 0 },

{0, 45, 0, 0, 3, 0 },

{2, 5, 0, 3, 0, 13 },

{21, 0, 0, 0, 13, 0 } };

int result[6] = { INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX,INT\_MAX, INT\_MAX };

bool visited[6] = { 0, 0, 0, 0, 0, 0 };

void Dejkstra();

**3.** Для решения задачи данные были представлены в следующем виде:

**3.1.** Глобальные переменные radiusA, radiusB отвечающие за радиус узлов и расстояние расположения этих узлов от центра.

int radiusA = 18,

radiusB = 170;

**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** За отрисовку сцены отвечает указатель scene класса QGraphicsScene

scene = new QGraphicsScene();

scene->setSceneRect(0, 0, ui->graphicsView->width(), ui->graphicsView->height());

ui->graphicsView->setScene(scene);

**4.2.** Добавление на сцену отрисованного узла/связи осуществляется через метод addItem() класса QGraphicsScene

scene->addItem(node);

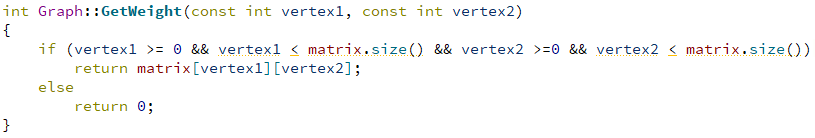
scene->addItem(line);

**5.** Поставленные задачи будут решены следующими действиями:

**5.1.** Функция Dejkstra() используется для вывода кратчайших путей до вершин – реализация метода Дейкстры.

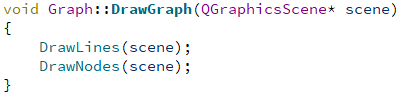


**5.2.** Метод GetWeight() необходим для получения длины пути (веса) от одного узла до другого.

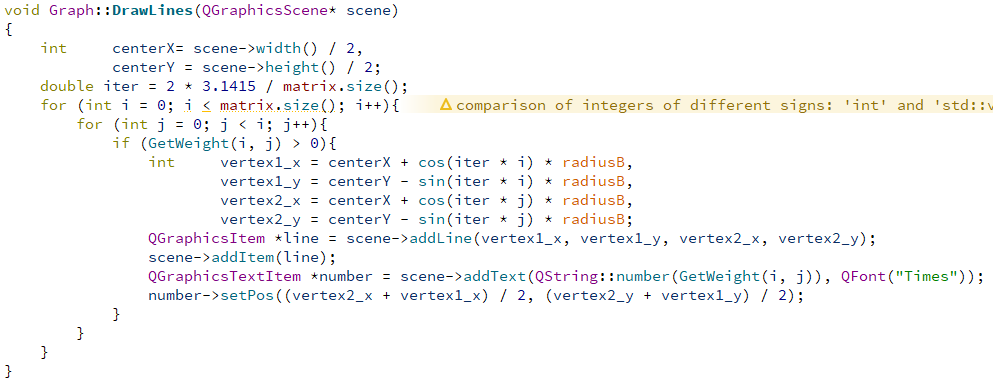


Возвращается значение из таблицы смежности присвоенной вектору векторов matrix в конструкторе.

**5.3.** Метод DrawGraph() вызывает методы отрисовки (DrawLines()) связей и узлов (DrawNodes).



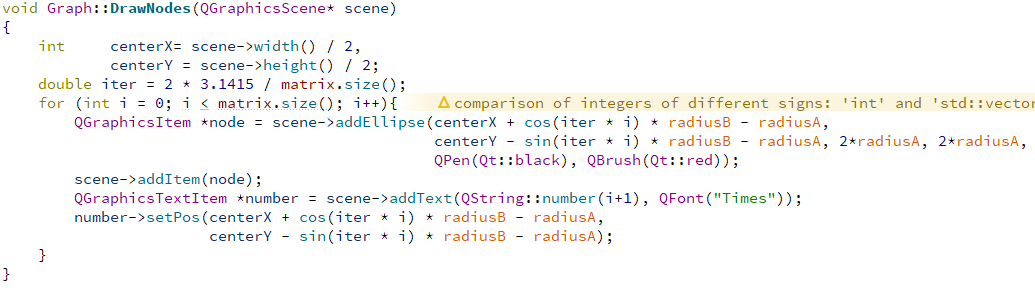
**5.4.** Метод DrawLines() отрисовывает связи.



Переменные centerX и centerY получают значение середины окна. Переменная iter отвечает за шаг, через который необходимо отрисовывать узлы. Во вложенном цикле for высчитываются координаты со сдвигом по окружности (используется полярная система координат). Указатель line типа QGraphicsItem задаёт линию с вычитанными координатами.

Указатель number типа QGraphicsTextItem отвечает за вывод веса.

**5.5.** Метод DrawNodes() отрисовывает узлы.



Переменные centerX и centerY получают значение середины окна. Переменная iter отвечает за шаг, через который необходимо отрисовывать узлы. Указатель node используется для отрисовки узлов, координаты высчитываются во вложенном цикле for по формуле окружности через полярную систему координат.

**Код**

#ifndef GRAPH\_H

#define GRAPH\_H

#include <QMainWindow>

#include <QGraphicsScene>

#include <QGraphicsItem>

#include <QPen>

#include <QBrush>

#include <QFont>

#include <QString>

#include <QGraphicsTextItem>

#include <vector>

QT\_BEGIN\_NAMESPACE

namespace Ui { class Graph; }

QT\_END\_NAMESPACE

class Graph : public QMainWindow

{

Q\_OBJECT

std::vector<std::vector<int>> matrix;

public:

Graph(QWidget \*parent = nullptr);

~Graph();

int GetWeight(const int vertex1, const int vertex2);

void DrawLines(QGraphicsScene\* scene);

void DrawNodes(QGraphicsScene\* scene);

void DrawGraph(QGraphicsScene\* scene);

private:

Ui::Graph \*ui;

QGraphicsScene \*scene;

};

#endif // GRAPH\_H

#include "graph.h"

#include "ui\_graph.h"

#include <cmath>

int radiusA = 18,

radiusB = 170;

int Graph::GetWeight(const int vertex1, const int vertex2)

{

if (vertex1 >= 0 && vertex1 < matrix.size() && vertex2 >=0 && vertex2 < matrix.size())

return matrix[vertex1][vertex2];

else

return 0;

}

void Graph::DrawLines(QGraphicsScene\* scene)

{

int centerX= scene->width() / 2,

centerY = scene->height() / 2;

double iter = 2 \* 3.1415 / matrix.size();

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

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

if (GetWeight(i, j) > 0){

int vertex1\_x = centerX + cos(iter \* i) \* radiusB,

vertex1\_y = centerY - sin(iter \* i) \* radiusB,

vertex2\_x = centerX + cos(iter \* j) \* radiusB,

vertex2\_y = centerY - sin(iter \* j) \* radiusB;

QGraphicsItem \*line = scene->addLine(vertex1\_x, vertex1\_y, vertex2\_x, vertex2\_y);

scene->addItem(line);

QGraphicsTextItem \*number = scene->addText(QString::number(GetWeight(i, j)), QFont("Times"));

number->setPos((vertex2\_x + vertex1\_x) / 2, (vertex2\_y + vertex1\_y) / 2);

}

}

}

}

void Graph::DrawNodes(QGraphicsScene\* scene)

{

int centerX= scene->width() / 2,

centerY = scene->height() / 2;

double iter = 2 \* 3.1415 / matrix.size();

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

QGraphicsItem \*node = scene->addEllipse(centerX + cos(iter \* i) \* radiusB - radiusA,

centerY - sin(iter \* i) \* radiusB - radiusA, 2\*radiusA, 2\*radiusA,

QPen(Qt::black), QBrush(Qt::red));

scene->addItem(node);

QGraphicsTextItem \*number = scene->addText(QString::number(i+1), QFont("Times"));

number->setPos(centerX + cos(iter \* i) \* radiusB - radiusA,

centerY - sin(iter \* i) \* radiusB - radiusA);

}

}

void Graph::DrawGraph(QGraphicsScene\* scene)

{

DrawLines(scene);

DrawNodes(scene);

}

Graph::Graph(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::Graph)

{

ui->setupUi(this);

matrix = { {0, 34, 8, 0, 2, 21 },

{24, 0, 57, 45, 5, 0 },

{8, 57, 0, 0, 0, 0 },

{0, 45, 0, 0, 3, 0 },

{2, 5, 0, 3, 0, 13 },

{21, 0, 0, 0, 13, 0 } };

scene = new QGraphicsScene();

scene->setSceneRect(0, 0, ui->graphicsView->width(), ui->graphicsView->height());

ui->graphicsView->setScene(scene);

scene->clear();

DrawGraph(scene);

}

Graph::~Graph()

{

delete ui;

}

#include "graph.h"

#include <QApplication>

#include <queue>

#include <iostream>

#include <Windows.h>

using namespace std;

vector <vector<int>> mat = { {0, 34, 8, 0, 2, 21 },

{24, 0, 57, 45, 5, 0 },

{8, 57, 0, 0, 0, 0 },

{0, 45, 0, 0, 3, 0 },

{2, 5, 0, 3, 0, 13 },

{21, 0, 0, 0, 13, 0 } };

int result[6] = { INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX,INT\_MAX, INT\_MAX };

bool visited[6] = { 0, 0, 0, 0, 0, 0 };

void Dejkstra();

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

{

cout << "Shortest paths from each point" << endl;

Dejkstra();

QApplication a(argc, argv);

Graph w;

w.show();

return a.exec();

}

void Dejkstra()

{

int top = 0; //вершина с номером 1

queue<int> line;

line.push(top);

result[0] = 0;

while (!line.empty()) {

int vertex = line.front();

line.pop();

for (int i = 0; i < mat[vertex].size(); i++) {

if (!visited[i] && mat[vertex][i] && (mat[vertex][i] + result[vertex] < result[i])) {

result[i] = mat[vertex][i] + result[vertex];

line.push(i);

}

}

}

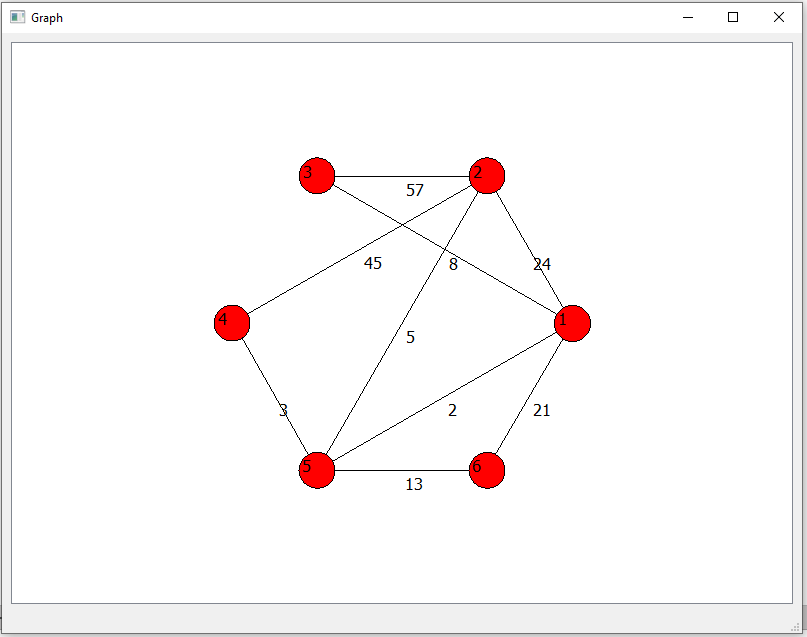
for (int i = 0; i < 6; i++) {

cout << i + 1 << ": " << result[i] << endl;

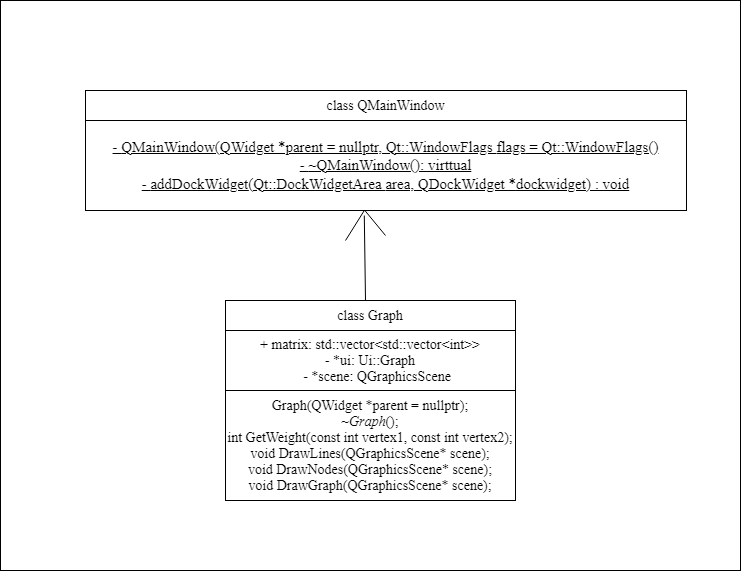
}

}

**Скриншоты**



**Диаграммы классов**

****