Министерство образования и науки Российской Федерации

Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №21**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: Графы. Алгоритм Дейкстры

Вариант 18

Выполнил:

Студент группы РИС-20-1б

Тараканов Д. М.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

Пермь

2021 год

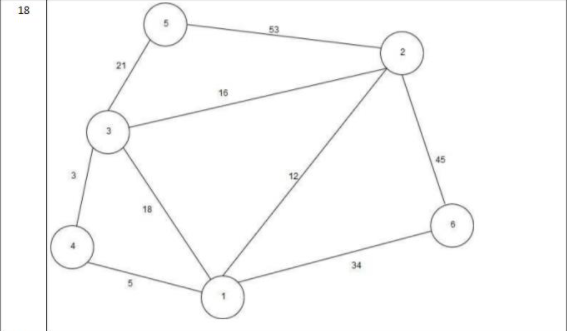
**Цель работы**

1. Получить практические навыки работы с графами и в реализации алгоритма Дейкстры.

**Постановка задачи**

Задача состоит в разработке программы, в которой реализуется алгоритм Дейкстры для графа. Реализовать визуализацию графа в OpenGL.

(18 вариант)



**Анализ задачи**

1. Определить какие необходимо выполнить действия для решения задачи:

* Разработать структуру Edge для хранения номеров городов (откуда-куда).

struct Edge {

int from;

int to;

};

* Реализовать матрицу смежности, в которой указаны длины дорог между городами в соответствии с вариантом.

vector<vector<int>> matrix = { {-1, 12, 18, 5, 0, 34},

{12, -1, 16, 0, 53, 45},

{18, 16, -1, 3, 21, 0},

{5, 0, 3, -1, 0, 0},

{0, 53, 21, 0, -1, 0},

{34, 45, 0, 0, 0, -1} };

* Реализовать алгоритм Дейкстры для нахождения минимального маршрута.

markup[from - 1] = 0;

do

{

min\_index = 1000;

int min = 1000;

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

{

if (flag[i] == 0 && markup[i] < min)

{

min = markup[i];

min\_index = i;

}

}

if (min\_index != 1000)

{

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

{

if (matrix[min\_index][i] > 0)

{

int temp = min + matrix[min\_index][i];

if (temp < markup[i])

{

markup[i] = temp;

}

}

}

flag[min\_index] = 1;

}

} while (min\_index < 1000);

* Реализовать построение оптимального пути.

queue<int> Queue;

stack<Edge> Edges;

Edge edge;

int visit[6] = { 0, 0, 0, 0, 0, 0 };

Queue.push(from);

while (!Queue.empty()) {

int node = Queue.front();

Queue.pop();

flag[node - 1] = 2;

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

if (matrix[node - 1][i] > 0 && visit[i] == 0) {

Queue.push(i + 1);

visit[i] = 1;

edge.from = node;

edge.to = i + 1;

Edges.push(edge);

if (node == to)

{

break;

}

}

}

}

* Реализовать вывод в консоль оптимального пути.

if (Edges.empty() == 0 && (markup[to - 1] < 1 || markup[to - 1] == 1000)) {

cout << "\nНеудалось найти оптимальный путь для заданного маршрута!\n";

}

else {

cout << "\nРазмер оптимального пути = " << markup[to - 1] << endl;

cout << "\nОптимальный путь: \n";

cout << to;

while (!Edges.empty()) {

edge = Edges.top();

Edges.pop();

if (edge.to == to) {

to = edge.from;

cout << " <- " << to;

}

}

cout << endl;

}

1. С какими типами данных действие надо сделать, в каком виде эти данные будут представлены:

* Переменные отвечающий за номер города откуда и номер города куда типа int.

int from, to;

* Массив минимальных возможных путей от города from типа int.

int markup[6] = { 1000, 1000, 1000, 1000, 1000, 1000 };

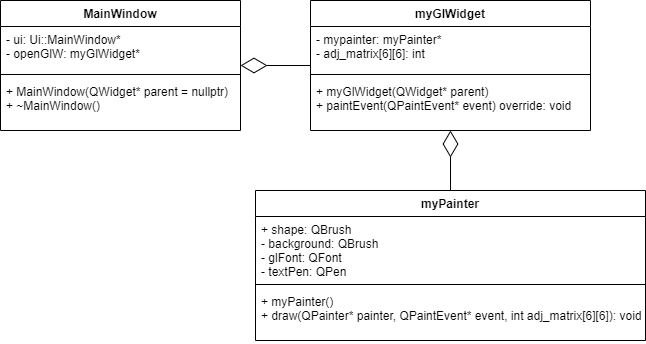
* Массив флагов отвечающий за посещение вершин типа bool.

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

* Стек содержащий в себе пути (откуда, куда) типа Edge.

stack<Edge> Edges;

**UML диаграмма классов**



**Код на С++**

#include <iostream>

#include <vector>

#include <queue>

#include <stack>

using namespace std;

vector<vector<int>> matrix = { {-1, 12, 18, 5, 0, 34},

{12, -1, 16, 0, 53, 45},

{18, 16, -1, 3, 21, 0},

{5, 0, 3, -1, 0, 0},

{0, 53, 21, 0, -1, 0},

{34, 45, 0, 0, 0, -1} };

struct Edge {

int from;

int to;

};

void show\_map() {

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

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

if (matrix[i][j] == -1) {

cout << " ";

}

else if (matrix[i][j] < 10) {

cout << " " << matrix[i][j];

}

else {

cout << " " << matrix[i][j];

}

}

cout << endl;

}

}

int main()

{

system("chcp 1251");

int markup[6] = { 1000, 1000, 1000, 1000, 1000, 1000 };

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

int from, to, min\_index = 1000;

cout << "\nТаблица графа:\n";

show\_map();

cout << "\nНайти оптимальный путь:\n\nОт города ";

do {

cin >> from;

if (from > 6 || from < 1) {

cout << "\n1 <= Номер города <=6\n";

}

} while (from > 6 || from < 1);

cout << "\nДо города ";

do {

cin >> to;

if (to > 6 || to < 1) {

cout << "\n1 <= Номер города <=6\n";

}

} while (to > 6 || to < 1);

markup[from - 1] = 0;

do

{

min\_index = 1000;

int min = 1000;

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

{

if (flag[i] == 0 && markup[i] < min)

{

min = markup[i];

min\_index = i;

}

}

if (min\_index != 1000)

{

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

{

if (matrix[min\_index][i] > 0)

{

int temp = min + matrix[min\_index][i];

if (temp < markup[i])

{

markup[i] = temp;

}

}

}

flag[min\_index] = 1;

}

} while (min\_index < 1000);

queue<int> Queue;

stack<Edge> Edges;

Edge edge;

int visit[6] = { 0, 0, 0, 0, 0, 0 };

Queue.push(from);

while (!Queue.empty()) {

int node = Queue.front();

Queue.pop();

flag[node - 1] = 2;

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

if (matrix[node - 1][i] > 0 && visit[i] == 0) {

Queue.push(i + 1);

visit[i] = 1;

edge.from = node;

edge.to = i + 1;

Edges.push(edge);

if (node == to)

{

break;

}

}

}

}

if (Edges.empty() == 0 && (markup[to - 1] < 1 || markup[to - 1] == 1000)) {

cout << "\nНеудалось найти оптимальный путь для заданного маршрута!\n";

}

else {

cout << "\nРазмер оптимального пути = " << markup[to - 1] << endl;

cout << "\nОптимальный путь: \n";

cout << to;

while (!Edges.empty()) {

edge = Edges.top();

Edges.pop();

if (edge.to == to) {

to = edge.from;

cout << " <- " << to;

}

}

cout << endl;

}

system("pause");

}

**Код QT**

main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

*return* a.exec();

}

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "myglwidget.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*private*:

Ui::MainWindow \*ui;

myGlWidget \*openGlW;

};

#endif *//* *MAINWINDOW\_H*

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

openGlW = *new* myGlWidget(*this*);

ui->verticalLayout->addWidget(openGlW);

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

myglwidget.h

#ifndef MYGLWIDGET\_H

#define MYGLWIDGET\_H

#include <QOpenGLWidget>

#include "mypainter.h"

*class* **myGlWidget** : *public* QOpenGLWidget

{

*public*:

**myGlWidget**(QWidget \*parent);

*public* slots:

void **draw\_new\_graph**();

*protected*:

void ***paintEvent***(QPaintEvent \*event) *override*;

*private*:

myPainter \*mypainter;

int adj\_matrix[6][6] = {{-1, 12, 18, 5, 0, 34},

{12, -1, 16, 0, 53, 45},

{18, 16, -1, 3, 21, 0},

{5, 0, 3, -1, 0, 0},

{0, 53, 21, 0, -1, 0},

{34, 45, 0, 0, 0, -1}};

};

#endif *//* *MYGLWIDGET\_H*

myglwidget.cpp

#include "myglwidget.h"

#include <QPainter>

myGlWidget::**myGlWidget**(QWidget \*parent)

: QOpenGLWidget(parent)

{

mypainter = *new* myPainter;

}

void myGlWidget::**draw\_new\_graph**()

{

*this*->update();

}

void myGlWidget::***paintEvent***(QPaintEvent \*event)

{

QPainter painter;

painter.begin(*this*);

mypainter->draw(&painter, event, adj\_matrix);

painter.end();

}

mypainter.h

#ifndef MYPAINTER\_H

#define MYPAINTER\_H

#include <QBrush>

#include <QFont>

#include <QPen>

#include <QWidget>

*class* **myPainter**

{

*public*:

**myPainter**();

*public*:

void **draw**(QPainter \*painter, QPaintEvent \*event, int adj\_matrix[6][6]);

QBrush shape;

*private*:

QBrush background;

QFont glFont;

QPen textPen;

};

#endif *//* *MYPAINTER\_H*

mypainter.cpp

#include "mypainter.h"

#include <QPainter>

#include <QPaintEvent>

#include <QWidget>

#include <QtMath>

*const* int knot\_count = 7;

*const* int text\_size = 24;

*const* int graph\_R = 200;

*const* double pi = 3.1415;

*const* int shift\_x = 250;

*const* int shift\_y = 250;

*const* int knot\_R = 34;

myPainter::**myPainter**()

{

shape = QBrush(QColor(168, 168, 129));

background = QBrush(QColor(235, 235, 235));

textPen = QPen(QColor(71, 98, 68));

glFont.setPixelSize(text\_size);

}

void myPainter::**draw**(QPainter \*painter, QPaintEvent \*event, int adj\_matrix[knot\_count-1][knot\_count-1])

{

painter->fillRect(event->rect(), background);

painter->setBrush(shape);

painter->setPen(textPen);

painter->setFont(glFont);

textPen.setWidth(4);

*for* (int i = 0; i < knot\_count-1; i++)

{

double angel = pi \* 3/2 + pi \* 2 / knot\_count \* i;

int knot\_x = cos(angel) \* graph\_R + shift\_x;

int knot\_y = sin(angel) \* graph\_R + shift\_y;

painter->drawEllipse(QRectF(knot\_x - knot\_R, knot\_y - knot\_R,

2 \* knot\_R, 2 \* knot\_R));

QString text;

int text\_x = knot\_x - knot\_R / 2;

int text\_y = knot\_y - text\_size / 2;

text.setNum(i + 1);

painter->drawText(QRect(text\_x, text\_y, knot\_R, text\_size),

Qt::*AlignCenter*, text);

*for* (int k = 0; k < knot\_count-1; k++)

{

*if* (adj\_matrix[i][k] > 0)

{

angel = pi \* 3/2 + pi \* 2 / knot\_count \* k;

double line\_x = cos(angel) \* graph\_R + shift\_x;

double line\_y = sin(angel) \* graph\_R + shift\_y;

double line\_length = sqrt((line\_x - knot\_x) \*

(line\_x - knot\_x) +

(line\_y - knot\_y) \*

(line\_y - knot\_y));

int l\_x1 = (line\_x - knot\_x) / line\_length \* knot\_R

+ knot\_x;

int l\_y1 = (line\_y - knot\_y) / line\_length \* knot\_R

+ knot\_y;

int l\_x2 = line\_x - (line\_x - knot\_x) / line\_length \*

knot\_R;

int l\_y2 = line\_y - (line\_y - knot\_y) / line\_length \*

knot\_R;

painter->drawLine(l\_x1, l\_y1, l\_x2, l\_y2);

text\_x = (l\_x2 + l\_x1) / 2 - knot\_R / 2;

text\_y = (l\_y2 + l\_y1) / 2 - text\_size / 2;

text.setNum(adj\_matrix[i][k]);

painter->drawRect(QRectF(text\_x, text\_y, knot\_R, text\_size));

painter->drawText(QRect(text\_x, text\_y, knot\_R, text\_size),

Qt::*AlignCenter*, text);

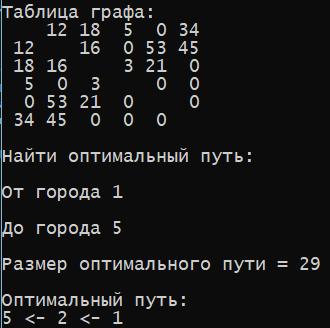
}

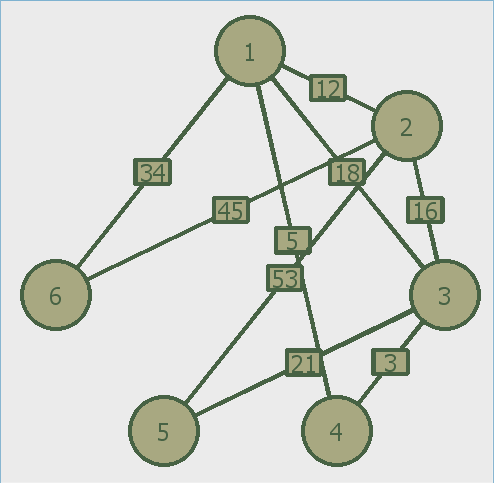
}

}

}

**Скриншоты тестов**

****

****