МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра информационные технологии и автоматизированные системы

**Дисциплина Информатика**

**Графы**

Выполнил студент ИВТ-22-2б:

Токмаков Герман Максимович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Пермь, 2023

**Постановка задачи**

Требуется реализовать работу с графом.  
1.3 Реализовать Алгоритмы на  
C++:  
1)Обход в ширину.  
2)Обход в глубину.  
3)Алгоритм Дейкстры.  
  
1.4 Требования:  
1) Пользовательский интерфейс - инструменты по решению разработчика (Windows Forms, например)  
2) Визуализация графа с использованием любой доступной графической библиотеки – SFML(Предпочтительно), SDL, OpenGL ( пример реализации представлен в файле: «Визуализация деревьев.docx»)  
3) Реализованные алгоритмы должны справляться не только с графом, представленным автором, а также применяться к другим  
Графам.  
4) Необходимы функции для редактирования графа: Создание(добавление) и удаление вершины и ребра. Редактирование весов ребер. Редактирование матрицы смежности (или инцидентности - в зависимости от реализации).

**Анализ задачи**

Для решения данной задачи использовать qt, так как он прост в использовании.

**Код.**

#define \_USE\_MATH\_DEFINES

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QDebug>

#include <math.h>

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

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

connect(ui->pushButton\_addVertex, &QPushButton::clicked,

this, &MainWindow::on\_pushButtonAddVertex);

connect(ui->pushButton\_deleteVertex, &QPushButton::clicked,

this, &MainWindow::on\_pushButtonDeleteVertex);

connect(ui->pushButton\_updateButton, &QPushButton::clicked,

this, &MainWindow::on\_updateButtonClicked);

connect(ui->pushButton\_TSP, &QPushButton::clicked,

this, &MainWindow::on\_TSPButtonClicked);

connect(ui->pushButton\_DFS, &QPushButton::clicked,

this, &MainWindow::on\_DFSButtonClicked);

connect(ui->pushButton\_BFS, &QPushButton::clicked,

this, &MainWindow::on\_BFSButtonClicked);

solution = QString("");

}

void MainWindow::***paintEvent***(QPaintEvent \*event) {

Q\_UNUSED(event);

paint.begin(this);

int numberOfNodes = ui->matrix->rowCount();

paintEllipse(*paint*, numberOfNodes);

bindEllipseWithDirection(*paint*, numberOfNodes);

paint.end();

}

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

{

delete ui;

}

void MainWindow::**addVertex**() {

ui->matrix->setColumnCount(ui->matrix->columnCount() + 1);

ui->matrix->setRowCount(ui->matrix->rowCount() + 1);

isVisitedVertecies = QVector<bool>(ui->matrix->rowCount(), 0);

}

void MainWindow::**deleteVertex**() {

ui->matrix->setColumnCount(ui->matrix->columnCount() - 1);

ui->matrix->setRowCount(ui->matrix->rowCount() - 1);

isVisitedVertecies = QVector<bool>(ui->matrix->rowCount(), 0);

}

void MainWindow::**on\_pushButtonAddVertex**() {

addVertex();

ui->matrix->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->matrix->verticalHeader()->setSectionResizeMode(QHeaderView::Stretch);

}

void MainWindow::**on\_pushButtonDeleteVertex**() {

deleteVertex();

ui->matrix->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->matrix->verticalHeader()->setSectionResizeMode(QHeaderView::Stretch);

}

QVector<int> MainWindow::**getNeighbours**(int vertex) {

QVector<int> rVect;

for (int i = 0; i < ui->matrix->rowCount(); i++) {

if (ui->matrix->item(i, vertex) != nullptr && !ui->matrix->item(i, vertex)->text().isEmpty()) { // здесь возникает ошибка

rVect.append(i);

}

}

return rVect;

}

void MainWindow::**DFS**(int vertex) {

QVector<int> neighbours = getNeighbours(vertex);

if (neighbours.size() == 0) {

return;

}

solution += QString::fromStdString(std::to\_string(vertex + 1)) + QString("->");

isVisitedVertecies[vertex] = 1;

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

if (isVisitedVertecies[neighbours[i]] == 0) {

DFS(neighbours[i]);

}

}

}

void MainWindow::**BFS**(int vertex) {

if (isVisitedVertecies[vertex] == 0) {

this->tmpQueue.enqueue(vertex);

solution += QString::fromStdString(std::to\_string(vertex + 1)) + QString("->");

isVisitedVertecies[vertex] = 1;

}

QVector<int> tmpVectorNeighbors = this->getNeighbours(vertex);

this->tmpQueue.dequeue();

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

if (isVisitedVertecies[tmpVectorNeighbors[i]] == 0) {

this->tmpQueue.enqueue(tmpVectorNeighbors[i]);

isVisitedVertecies[tmpVectorNeighbors[i]] = 1;

solution += QString::fromStdString(std::to\_string(tmpVectorNeighbors[i] + 1)) + QString("->");

}

}

if (this->tmpQueue.empty()) {

return;

}

int a = this->tmpQueue.front();

BFS(a);

}

void MainWindow::**on\_updateButtonClicked**() {

isVisitedVertecies = QVector<bool>(ui->matrix->columnCount(), 0);

solution = QString("");

update();

}

void MainWindow::**paintEllipse**(QPainter &paint, int numberOfVerticies) {

paint.setPen(QColor(0, 0, 0));

paint.setBrush(Qt::Dense7Pattern);

int shiftX = 700,

shiftY = 300;

int radiusCircle = 160,

radiusEllipse = 40;

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

int xCoord = shiftX + radiusCircle\*cos(2\*M\_PI/numberOfVerticies \* i);

int yCoord = shiftY + -radiusCircle\*sin(2\*M\_PI/numberOfVerticies \* i);

paint.drawText(xCoord, yCoord, QString("%1").arg(i + 1));

paint.drawEllipse(xCoord, yCoord, radiusEllipse, radiusEllipse);

}

}

void MainWindow::**bindEllipseWithDirection**(QPainter &paint, int numberOfVerticies) {

paint.setPen(QColor(0, 0, 0));

paint.setBrush(Qt::Dense7Pattern);

int shiftX = 700,

shiftY = 300;

int radiusCircle = 160,

radiusEllipse = 40;

for (int vertexCount = 0; vertexCount < numberOfVerticies; vertexCount++) {

QVector<int> neighbours = getNeighbours(vertexCount);

int xCoordVertexFirst = shiftX + radiusCircle\*cos(2\*M\_PI/numberOfVerticies\*vertexCount) + radiusEllipse/2;

int yCoordVertexFirst = shiftY + -radiusCircle\*sin(2\*M\_PI/numberOfVerticies\*vertexCount) + radiusEllipse/2;

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

int xCoordVertexSecond = shiftX + radiusCircle\*cos(2\*M\_PI/numberOfVerticies\*neighbours[i]) + radiusEllipse/2;

int yCoordVertexSecond = shiftY + -radiusCircle\*sin(2\*M\_PI/numberOfVerticies\*neighbours[i]) + radiusEllipse/2;

paint.drawLine(xCoordVertexFirst, yCoordVertexFirst, xCoordVertexSecond, yCoordVertexSecond);

}

}

}

void MainWindow::**getMatrix**() {

path.clear();

matrix.clear();

for (int i = 0; i < ui->matrix->rowCount(); i++) {

QVector<int> qvect = QVector<int>(ui->matrix->rowCount(), 0);

for (int j = 0; j < ui->matrix->rowCount(); j++) {

if ((ui->matrix->item(i, j) != nullptr && !ui->matrix->item(i, j)->text().isEmpty())) {

qvect[j] = ui->matrix->item(i, j)->*data*(0).toInt();

}

else qvect[j] = 0;

}

matrix.push\_back(qvect);

}

}

void MainWindow::**commi**()

{

int i = 0;

while (i < matrix.size()) {

setInfinity();

rowReduction();

columnReduction();

int i\_pos = -1, j\_pos = -1;

int max = 0;

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

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

if (matrix[i][j] == 0) {

int minInRow = findMinInRow(i, j);

int minInColumn = findMinInColumn(i, j);

if (minInRow == INT\_MAX) minInRow = 0;

if (minInColumn == INT\_MAX) minInColumn = 0;

int value = minInRow + minInColumn;

if (value >= max) {

max = value;

i\_pos = i;

j\_pos = j;

}

}

}

}

path.push\_back(i\_pos);

path.push\_back(j\_pos);

setInfinity(i\_pos, j\_pos);

matrix[j\_pos][i\_pos] = INT\_MAX;

i++;

}

}

void MainWindow::**rowReduction**()

{

//минимумы cтрок

QVector<int> minimums;

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

int minimum = INT\_MAX;

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

if (matrix[i][j] < minimum) minimum = matrix[i][j];

}

minimums.push\_back(minimum);

}

//редукция

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

for (int j = 0; j < matrix.size() && minimums[i] != INT\_MAX; j++) {

if (minimums[i] < 10000) matrix[i][j] = matrix[i][j] - minimums[i];

}

}

}

void MainWindow::**columnReduction**()

{

QVector<int> minimumsST;

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

int minimum = INT\_MAX;

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

if (matrix[j][i] < minimum) minimum = matrix[j][i];

}

minimumsST.push\_back(minimum);

}

//редукция

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

for (int j = 0; j < matrix.size() && minimumsST[i] != INT\_MAX; j++) {

if (minimumsST[i] < 10000) matrix[j][i] = matrix[j][i] - minimumsST[i];

}

}

}

int MainWindow::**findMinInRow**(int i\_p, int j\_p)

{

int minimum = INT\_MAX;

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

if (j == j\_p) continue;

else if (matrix[i\_p][j] < minimum) minimum = matrix[i\_p][j];

}

return minimum;

}

int MainWindow::**findMinInColumn**(int i\_p, int j\_p)

{

int minimum = INT\_MAX;

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

if (i == i\_p) continue;

else if (matrix[i][j\_p] < minimum) minimum = matrix[i][j\_p];

}

return minimum;

}

void MainWindow::**setInfinity**(int i\_p, int j\_p)

{

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

matrix[i][i] = INT\_MAX;

}

if (i\_p != -1 && j\_p != -1) {

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

matrix[i\_p][i] = INT\_MAX;

matrix[i][j\_p] = INT\_MAX;

}

}

}

void MainWindow::**printPath**()

{

solution = "";

QVector<int> newPath;

int begin = 0;

while (newPath.size() < path.size()) {

bool flag = true;

for (int i = 0; i < path.size() && flag; i++) {

if (path[i] == begin && i % 2 == 0) {

newPath.push\_back(path[i]);

newPath.push\_back(path[i + 1]);

begin = newPath[newPath.size() - 1];

flag = false;

}

}

}

for (int i = 0; i < newPath.size() - 1; i++) {

if (newPath[i] == newPath[i + 1]) {

newPath.erase(newPath.begin() + i);

i--;

}

}

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

int k = newPath[i] + 1;

solution += QString::fromStdString(std::to\_string(k)) + QString("->");

}

}

void MainWindow::**on\_TSPButtonClicked**() {

getMatrix();

commi();

printPath();

QMessageBox msgBox;

msgBox.setWindowTitle("TSP");

msgBox.setText(solution);

msgBox.*exec*();

}

void MainWindow::**on\_DFSButtonClicked**() {

on\_updateButtonClicked();

DFS(0);

QMessageBox msgBox;

msgBox.setWindowTitle("DFS");

msgBox.setText(solution);

msgBox.*exec*();

}

void MainWindow::**on\_BFSButtonClicked**() {

on\_updateButtonClicked();

BFS(0);

QMessageBox msgBox;

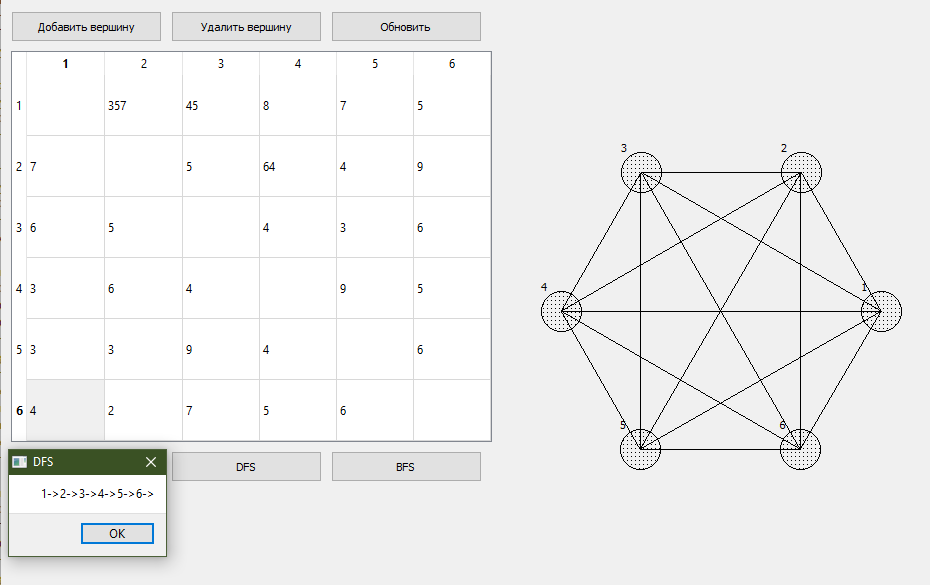
msgBox.setWindowTitle("BFS");

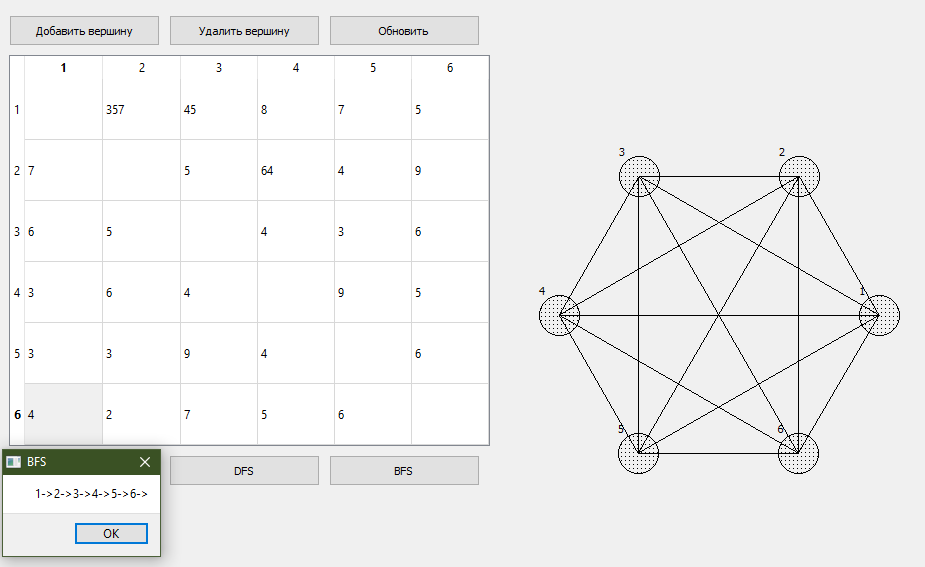
msgBox.setText(solution);

msgBox.*exec*();

}

**Вывод программы**

**\_**

****