Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**Тема:** Лабораторная работа «задача комивояжера» и АРМ

Семестр: 2

Выполнил студент ИВТ-23-2б:

Синицын Владислав Сергеевич

(дата, подпись)

Проверила:

Полякова Ольга Андреевна

(дата, подпись)

Пермь 2024

**Комивояжер**

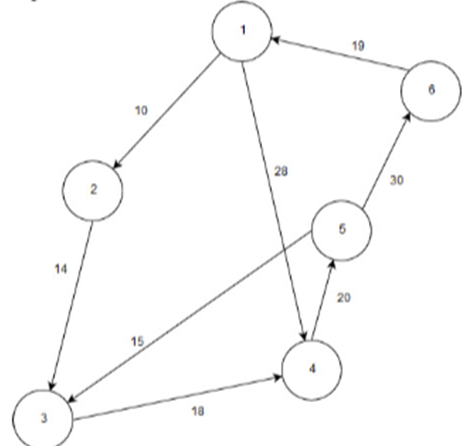
**Задание**

-задача коммивояжера

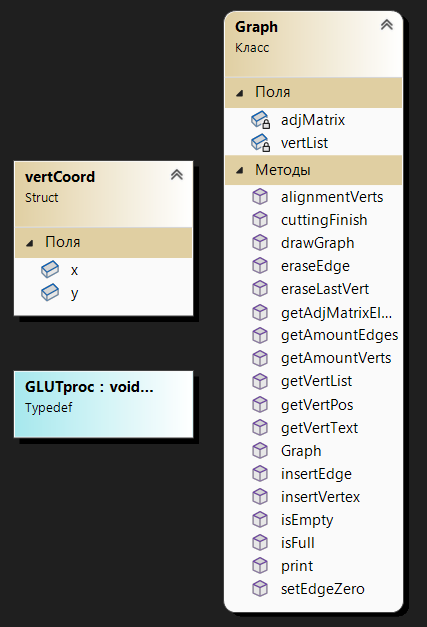
- интерфейса

- текстового сопровождения

- вертикальная печать

****

**UML-Диаграмма**



**Программное решение**

**Main.cpp**

#include "Graph\_.h"

#include <Windows.h>

#include <GL\glew.h>

#include <GL\freeglut.h>

#include <iostream>

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

setlocale(LC\_ALL, "Russian");

makeGraph();

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(WinW, WinH);

glutCreateWindow("Графы");

glClearColor(0.8, 0.8, 0.8, 0.8);

WinW = glutGet(GLUT\_WINDOW\_WIDTH);

WinH = glutGet(GLUT\_WINDOW\_HEIGHT);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMotionFunc(mouseMove);

glutPassiveMotionFunc(mouseMove);

glutMouseFunc(mouseClick);

glutMainLoop();

return 0;

}

**Graph.h**

#ifndef GRAPH\_GRAPH\_H

#define GRAPH\_GRAPH\_H

#include <Windows.h>

#include <iostream>

#include <vector>

#include <iomanip>

#include <string>

using namespace std;

const int maxSize = 20; // Максимальное количество вершин

extern int WinW; // Ширина окна

extern int WinH; // Высота окна

// Структура координат вершни

struct vertCoord {

int x, y;

};

class Graph

{

private:

vector<int> vertList; // Вектор вершин

int adjMatrix[maxSize][maxSize]; // Матрица смежности

public:

Graph(); // Конструктор

bool isEmpty(); // Проверка на пустоту

bool isFull(); // Проверка на полноту

int getVertText(int i); // Получить значение вершины

vector<int> getVertList(); // Получить вектор вершин

int getAdjMatrixElem(int i, int j); // Получить элемент матрицы смежности

int getAmountVerts(); // Получить количество вершин графа

int getVertPos(const int& vertex); // Получить позицию вершины в векторе вершин

int getAmountEdges(); // Получить количество ребер графа

void setEdgeZero(int i, int j); // Занулить элемент матрицы (для Коммивояжера)

void insertVertex(const int& vertex); // Добавить вершину (в конец вектора вершин)

void insertEdge(const int& vertex1, const int& vertex2, int weight); // Добавить ребро весом weight между vertex1 и vertex2

void print(); // Вывод матрицы смежности в консоль

void eraseLastVert(); // Удалить последнюю вершину в векторе вершин

void eraseEdge(const int& vertex1, const int& vertex2); // Удалить ребро между vertex1 и vertex2

void drawGraph(); // Нарисовать граф в OpenGL

void cuttingFinish(); // Закончили резать

void alignmentVerts(); // Выравниваем вершины

};

extern Graph graph; // Создается объект класса Graph

/\*

// Функции для решения задачи Коммивояжера

\*/

void printMatrix(int\*\* matrix); // Вывод текущей матрицы

bool checkSalesman(int\*\* matrix); // Проверка: можно ли решить задачу Коммивояжера

int\*\* Change\_Matrix(); // Создается матрица, заменяя 0 на -1

int\* searchMinElem(int\* line, int n); // Поиск минимального элемента в строке матрицы

int\*\* reductMatrix(int\*\* oldmatrix); // Редукция матрицы

int\*\* highZero(int\*\* oldmatrix); // Нуль с наибольшим потенциалом

void printResult(); // Вывод результата решения Коммивояжера

/\*

// Рисование кнопок OpenGL

\*/

void drawButton1(); // Кнопка "Make new graph"

void drawButton2(); // Кнопка "Salesman Task"

void drawButton3(); // Кнопка "Print Matrix In Console"

void drawButton4(); // Кнопка "Add Vertex"

void drawButton5(); // Кнопка "Delete Vertex"

void drawButton6(); // Кнопка "Add Edge"

void drawButton7(); // Кнопка "Delete Edge"

void drawButton11(); // Кнопка "Draw path"

void drawInformation(); // Значение радиуса в окне OpenGL

/\*

// Рисование элементов графа OpenGL

\*/

void drawCircle(int x, int y, int R, bool aim); // Рисование круга (aim - навелись на кружок)

void drawText(int nom, int x1, int y1); // Рисование веса ребра

void drawLine(int text, int x0, int y0, int x1, int y1, int start, int end); // Рисование ребра

void drawVertex(int n); // Рисование вершины

void drawPath(int n); // Рисование пути Коммивояжера

void drawCreateEdge(); // Рисуем линию для путей

void drawCutting(); // Рисуем линию для "отрезания"

void makeGraph(); // Создание графа

void setCoords(int i, int n); // Установка координат для вершин графа

/\*

// Функции - проверка

\*/

int checkCircle(int x, int y); // Проверка курсора на кружке

void checkButton(int x, int y); // Проверка курсора на кнопке

/\*

// Функции отслеживания поведения мыши

\*/

void mouseMove(int x, int y); // Отслеживание движения курсора

void mouseClick(int button, int state, int x, int y); // Отслеживание нажатия клавиш на мыши

/\*

// Изображение

\*/

void reshape(int w, int h); // Изменение размера экрана

void display(); // Создание изображения

/\*

// Вспомогательные

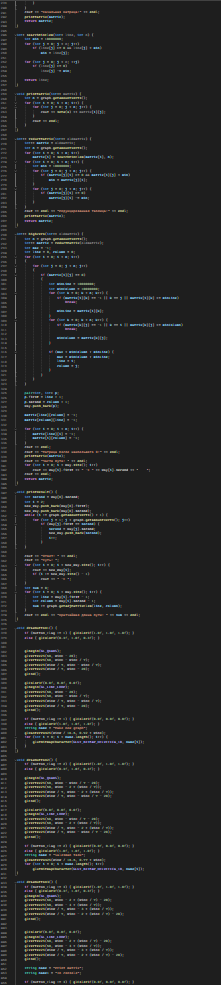
\*/

bool intersect(vertCoord a1, vertCoord a2, vertCoord b1, vertCoord b2); // Проверка на пересечение двух отрезков

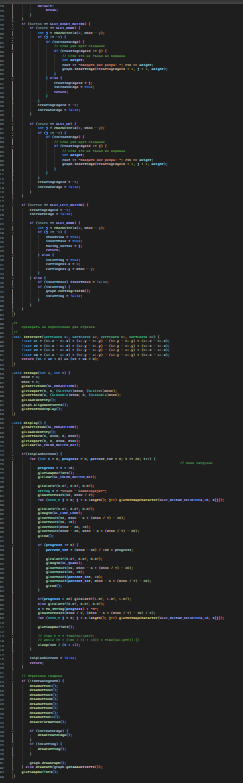
#endif // GRAPH\_GRAPH\_H

**Graph.cpp**

****

****

**…**

****

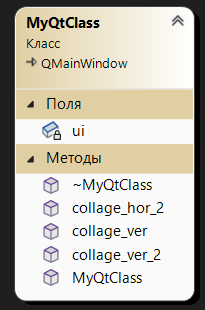
**АРМ**

**Задание**

- интерфейса

**-**арм

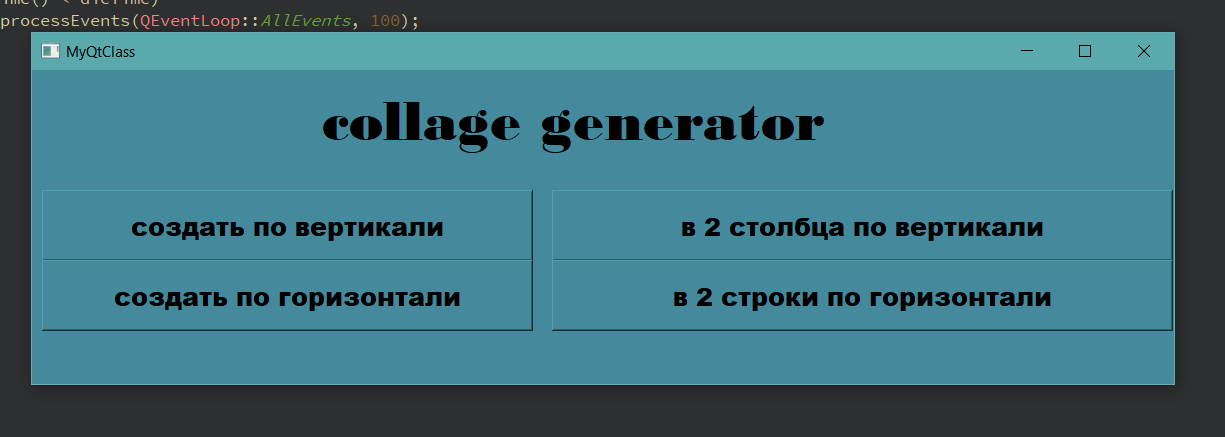
**UML-Диаграмма**



**Программное решение**

** **

****

****

**Main.cpp**

#include "myqtclass.h"

#include <QLabel>

#include <QTime>

#include <QApplication>

#include <QSplashScreen>

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

{

QApplication a(argc, argv);

// Создаем и показываем окно загрузки

QLabel \*loadingLabel = new QLabel("Loading...");

loadingLabel->show();

QTime dieTime = QTime::currentTime().addSecs(2);

while (QTime::currentTime() < dieTime)

QCoreApplication::processEvents(QEventLoop::AllEvents, 100);

loadingLabel->close();

MyQtClass w;

w.show();

return a.exec();

}

**Myqtclass.cpp**

#include <QRandomGenerator>

#include <QTimer>

#include "myqtclass.h"

#include "ui\_myqtclass.h"

#include <QImage>

#include <QPixmap>

#include <QPainter>

#include <QFileDialog>

#include <QDir>

#include <QDebug>

MyQtClass::MyQtClass(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MyQtClass)

{

ui->setupUi(this);

connect(ui->collagebtn\_ver, &QPushButton::clicked, this, &MyQtClass::collage\_ver); //кнопка коллажа по вертикали

connect(ui->collagebtn\_hor, &QPushButton::clicked, this, &MyQtClass::collage\_hor); //кнопка коллажа по горизонтали

connect(ui->collagebtn\_ver\_2, &QPushButton::clicked, this, &MyQtClass::collage\_ver\_2); //кнопка коллажа по вертикали 2

connect(ui->collagebtn\_hor\_2, &QPushButton::clicked, this, &MyQtClass::collage\_hor\_2); //кнопка коллажа по горизонтали 2

/\* connect(ui->userinput, &QLineEdit::textChanged, this, &MyQtClass::on\_userInput\_textChanged);

connect(ui->usertextbtr, &QPushButton::clicked, this, &MyQtClass::changeText);

QTimer \*timer = new QTimer(this);

connect(timer, &QTimer::timeout, this, &MyQtClass:: updatePressure);

timer->start(44);

connect(this, &MyQtClass::pressureChanged, this, &MyQtClass::on\_PressureChanged);

\*/

}

void MyQtClass::collage\_ver(){

// Выбираем изображения для создания коллажа

QStringList imagePaths = QFileDialog::getOpenFileNames(nullptr, "Выберите изображения", QDir::homePath(), "Images (\*.png \*.jpg)");

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

qDebug() << "Нет выбранных изображений";

return;

}

// Вычисляем общую высоту и ширину для коллажа

int collageWidth = 1080; // Минимальная ширина

int collageHeight = 0;

foreach (QString imagePath, imagePaths) {

QImage image(imagePath);

collageHeight += image.height();

if (image.width() > collageWidth) {

collageWidth = image.width(); // Обновляем ширину, если обнаружен более широкий образ

}

}

// Создаем QImage для коллажа

QImage collage(collageWidth, collageHeight, QImage::Format\_ARGB32);

collage.fill(Qt::white);

// Создаем QPainter для рисования на коллаже

QPainter painter(&collage);

int y = 0;

// Рисуем каждое изображение на коллаже

foreach (QString imagePath, imagePaths) {

QImage image(imagePath);

int imageWidth = image.width();

int imageHeight = image.height();

painter.drawImage(QPoint((collageWidth - imageWidth) / 2, y), image);

y += imageHeight;

}

// Сохраняем коллаж в файл

QString collagePath = QFileDialog::getSaveFileName(nullptr, "Сохранить коллаж", QDir::homePath(), "Images (\*.png)");

collage.save(collagePath);

}

void MyQtClass::collage\_hor(){

// Выбираем изображения для создания коллажа

QStringList imagePaths = QFileDialog::getOpenFileNames(nullptr, "Выберите изображения", QDir::homePath(), "Images (\*.png \*.jpg)");

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

qDebug() << "Нет выбранных изображений";

return;

}

// Вычисляем общую высоту и ширину для коллажа

int collageHeight = 0; // Минимальная высота

int collageWidth = 0;

foreach (QString imagePath, imagePaths) {

QImage image(imagePath);

collageWidth += image.width();

if (image.height() > collageHeight) {

collageHeight = image.height(); // Обновляем высоту, если обнаружена более высокая картинка

}

}

// Создаем QImage для коллажа

QImage collage(collageWidth, collageHeight, QImage::Format\_ARGB32);

collage.fill(Qt::white);

// Создаем QPainter для рисования на коллаже

QPainter painter(&collage);

int x = 0;

// Рисуем каждое изображение на коллаже

foreach (QString imagePath, imagePaths) {

QImage image(imagePath);

int imageWidth = image.width();

int imageHeight = image.height();

painter.drawImage(QPoint(x, (collageHeight - imageHeight) / 2), image);

x += imageWidth;

}

// Сохраняем коллаж в файл

QString collagePath = QFileDialog::getSaveFileName(nullptr, "Сохранить коллаж", QDir::homePath(), "Images (\*.png)");

collage.save(collagePath);

}

void MyQtClass::collage\_ver\_2(){

// Выбираем изображения для создания коллажа

QStringList imagePaths = QFileDialog::getOpenFileNames(nullptr, "Выберите изображения", QDir::homePath(), "Images (\*.png \*.jpg)");

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

qDebug() << "Нет выбранных изображений";

}

// Проверяем, что общее количество изображений делится на 2

if (imagePaths.size() % 2 != 0) {

qDebug() << "Количество изображений должно быть кратно двум для создания двух колонн";

}

// Вычисляем количество изображений в каждой колонне

int imagesPerColumn = imagePaths.size() / 2;

// Вычисляем ширину коллажа по максимальной ширине изображений в каждой колонне

int maxColumnWidth = 0;

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

QImage image(imagePaths.at(i));

if (image.width() > maxColumnWidth) {

maxColumnWidth = image.width();

}

}

int collageWidth = maxColumnWidth \* 2;

// Вычисляем высоту коллажа

int collageHeight = 0;

for (int i = 0; i < imagePaths.size()/2; ++i) {

QImage image(imagePaths.at(i));

collageHeight += image.height();

}

// collageHeight = collageHeight/2;

// Создаем QImage для коллажа

QImage collage(collageWidth, collageHeight, QImage::Format\_ARGB32);

collage.fill(Qt::white);

// Создаем QPainter для рисования на коллаже

QPainter painter(&collage);

int x = 0;

int y = 0;

// Рисуем изображения в двух колоннах

for (int col = 0; col < 2; ++col) {

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

QImage image(imagePaths.at(col \* imagesPerColumn + j));

painter.drawImage(x, y, image);

y += image.height();

}

x += maxColumnWidth;

y = 0; // Сброс координаты y для новой колонны

}

// Сохраняем коллаж в файл

QString collagePath = QFileDialog::getSaveFileName(nullptr, "Сохранить коллаж", QDir::homePath(), "Images (\*.png)");

collage.save(collagePath);

}

void MyQtClass::collage\_hor\_2(){

// Выбор изображений для коллажа

QStringList imagePaths = QFileDialog::getOpenFileNames(nullptr, "Выберите изображения", QDir::homePath(), "Images (\*.png \*.jpg)");

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

qDebug() << "Нет выбранных изображений";

}

// Убеждаемся, что количество изображений делится на 2

if (imagePaths.size() % 2 != 0) {

qDebug() << "Количество изображений должно быть кратно двум для создания двух строк";

}

int imagesPerRow = imagePaths.size() / 2; // количество изображений в каждой строке

// Вычисляем максимальную ширину и высоту изображений

int maxImageWidth = 0, maxImageHeight = 0;

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

QImage image1(imagePaths.at(i));

QImage image2(imagePaths.at(i + imagesPerRow));

maxImageWidth = qMax(maxImageWidth, qMax(image1.width(), image2.width()));

maxImageHeight = qMax(maxImageHeight, qMax(image1.height(), image2.height()));

}

int collageWidth = maxImageWidth \* imagesPerRow;

int collageHeight = 2 \* maxImageHeight;

// Создаем QImage для коллажа

QImage collage(collageWidth, collageHeight, QImage::Format\_ARGB32);

collage.fill(Qt::white);

// Создаем QPainter для рисования на коллаже

QPainter painter(&collage);

int x = 0, y = 0;

// Рисуем изображения в двух строках

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

QImage image1(imagePaths.at(i));

QImage image2(imagePaths.at(i + imagesPerRow));

painter.drawImage(x, y, image1);

painter.drawImage(x + maxImageWidth, y, image2);

y += maxImageHeight;

}

// Сохраняем коллаж в файл

QString collagePath = QFileDialog::getSaveFileName(nullptr, "Сохранить коллаж", QDir::homePath(), "Images (\*.png)");

collage.save(collagePath);

}

MyQtClass::~MyQtClass()

{

delete ui;

}

/\*

void MyQtClass::changeText(){

ui->usertext->setText("Привет!");

ui->usertext->setGeometry(0, 0, 1200, 150);

const QString s = ui->userinput->text();

}

void MyQtClass::on\_userInput\_textChanged(const QString &arg1){

ui->usertext->setText(arg1);

}

void MyQtClass::updatePressure(){

float newRandomPressure =

QRandomGenerator::global()->bounded(50);

emit pressureChanged(newRandomPressure);

}

void MyQtClass::on\_PressureChanged(const float newValue){

ui->pressureDisplay->display(newValue);

}

\*/

**Myqtclass.h**

#ifndef MYQTCLASS\_H

#define MYQTCLASS\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MyQtClass; }

QT\_END\_NAMESPACE

class MyQtClass : public QMainWindow

{

Q\_OBJECT

public:

MyQtClass(QWidget \*parent = nullptr);

~MyQtClass();

signals:

void pressureChanged(float currentValue);

public slots:

void collage\_hor();

void collage\_ver();

void collage\_hor\_2();

void collage\_ver\_2();

/\*

void on\_userInput\_textChanged(const QString &arg1);

void changeText();

void on\_PressureChanged(const float newValue);

void updatePressure();

\*/

private:

Ui::MyQtClass \*ui;

};

#endif // MYQTCLASS\_H