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:** Творческая работа

Семестр: 2

Выполнил студент ИВТ-22-2б:

Мельников Глеб Владимирович

(дата, подпись)

Проверила:

Полякова Ольга Андреевна

(дата, подпись)

Пермь 2023

**АРМ Актёра**

Идея:

Работа актера очень трудна и требует тщательного распределения времени. Актеру нужно планировать свой день чтобы успеть к множеству мероприятий, для этого авторы создали АРМ, чтобы актер мог контролировать свое время. У актёра будет системы с календарём, где он сможет удобно хранить мероприятия и иметь по ним всю нужную информацию.

ПО:

Для выполнение творческой работы выбран фреймворк Qt, видео снято с помощью программы OBS Studio.

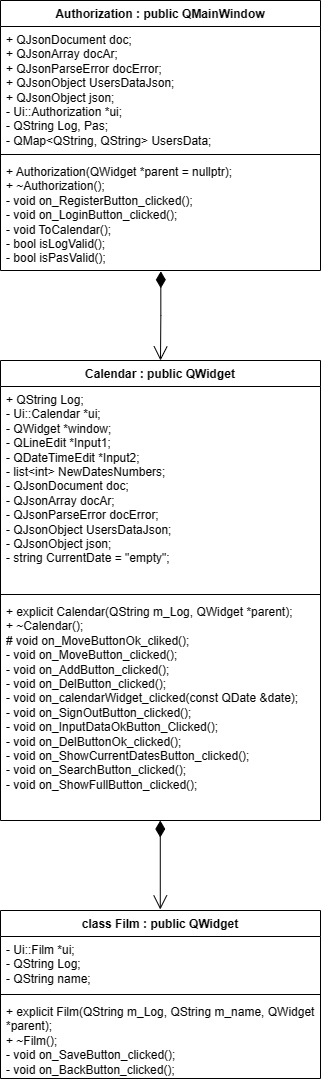
Классы:

Authorization – отвечает за окно авторизации пользователя в системе. Введённые логин и пароль сохраняются в файл в формате Json. После ввода пользователем логина, логин передаётся в атрибут класса Calendar.

Класс Calendar отвечает за отображение, добавление, поиск, удаление данных. Данные сохраняются в файл в формате Json, в виде Дата и время : мероприятие. Функция “Перейти” открывает окно класса Film и передаёт ему значения логина и названия мероприятия.

Класс Film позволяет заполнить актёру информацию о выбранном мероприятии и сохранить её. Информация хранится при помощи текстового файла.

UML-диаграмма:



**Решение задачи Коммивояжёра**

ПО:

Для выполнение творческой работы выбран фреймворк Qt, видео снято с помощью программы OBS Studio.

Классы:

Класс Vertex отвечает за объект вершины. Выполняет отрисовку вершин, хранения количества вершин, хранение ребер вершин.

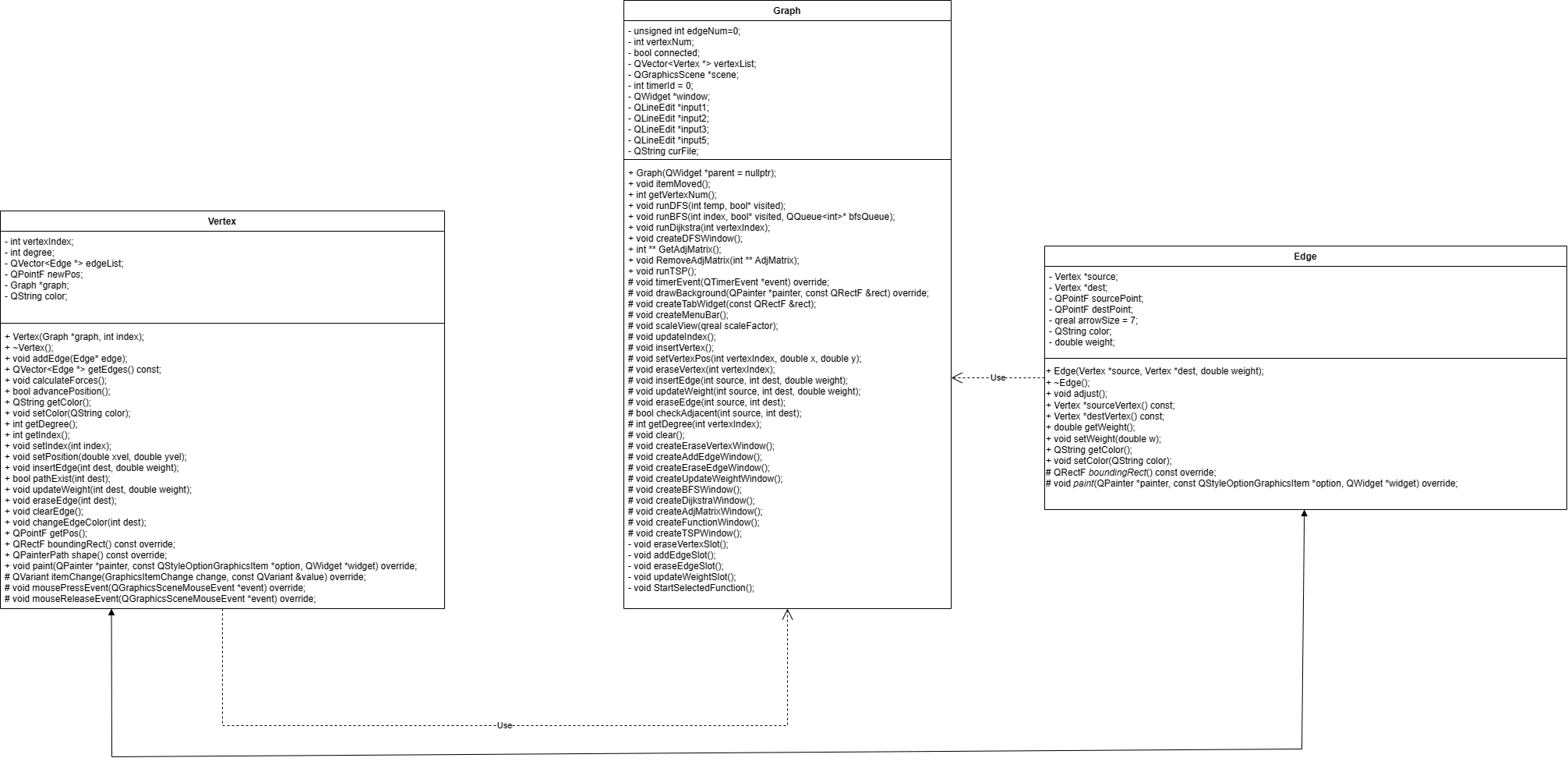
Класс Edge отвечает за объект ребра. Выполняет отрисовку ребра, хранение начальной вершины - источника и конечной вершины – пункта назначения.

КлассGraph отвечает за выполнение функций: добавление вершины, удаление вершины, добавление ребра, обновление веса ребра, удаление ребра, вывод матрицы смежности графа, удаление графа, поиск в глубину, поиск в ширину, алгоритм Дейкстры, решение задачи Коммивояжёра.

Реализация решения задачи Коммивояжёра:

Задача Коммивояжёра решена методом ветвей и границ. Для вывода пути из проходящих пунктов используется словарь.

UML-диаграмма:



Код программы:

void Graph::**runTSP**() {

int \*\*AdjMatrix = GetAdjMatrix();//получение матрицы смежности

//проверка на возможность выполнения функции

bool FlagToContinue = (vertexNum > 2);

for (int i = 0; i < vertexNum && FlagToContinue; i++) {

FlagToContinue = (vertexList[i]->getEdges().size() > 1);

}

if (FlagToContinue) {

int \*\* OrigianalAdjMatrix = GetAdjMatrix();

int m = INT\_MAX, Sum = 0;

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

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

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

AdjMatrix[i][j] = m;

}

}

}

QMap<int,int> roads = {};

for (int z = 0; z < vertexNum; z++) {

int maxi = 0, maxj = 0;

int di[vertexNum], dj[vertexNum];

int maxScore = 0;

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

di[i] = dj[i] = m;

}

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

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

if (AdjMatrix[i][j] < di[i]) {

di[i] = AdjMatrix[i][j];

}

}

}

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

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

if (AdjMatrix[i][j] != m) {

AdjMatrix[i][j] -= di[i];

}

}

}

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

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

if (AdjMatrix[j][i] < dj[i]) {

dj[i] = AdjMatrix[j][i];

}

}

}

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

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

if (AdjMatrix[j][i] != m) {

AdjMatrix[j][i] -= dj[i];

}

}

}

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

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

int imin = m, jmin = m;

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

for (int k = 0; k < vertexNum; k++) {

if (AdjMatrix[k][j] < imin && i != k) {

imin = AdjMatrix[k][j];

}

if (AdjMatrix[i][k] < jmin) {

jmin = AdjMatrix[i][k];

}

}

if (maxScore < imin + jmin) {

maxScore = imin + jmin;

maxi = i;

maxj = j;

}

}

}

}

roads[maxj] = maxi;

AdjMatrix[maxj][maxi] = m;

Sum += OrigianalAdjMatrix[maxi][maxj];

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

AdjMatrix[i][maxj] = m;

AdjMatrix[maxi][i] = m;

}

}

int i = 0;

str += QString::number(i);

while(FlagToContinue) {

str += "-> " + QString::number(roads[i]);

i = roads[i];

FlagToContinue = (i != 0);

}

str += " Сумма = " + QString::number(Sum);

RemoveAdjMatrix(*OrigianalAdjMatrix*);

}

else {

str += "Решение задачи Коммивояжёра невозможно!\nКаждая вершина графа должна иметь как минимум два ребра!";

}

RemoveAdjMatrix(*AdjMatrix*);

}