**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **Алгоритмы на графах. Кратчайшие пути в графе. Алгоритм Флойда-Уоршалла**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 5381 |  | Кочнева О.Р. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студентка Кочнева О.Р. | | |
| Группа 5381 | | |
| Тема работы: Алгоритмы на графах. Кратчайшие пути в графе. Алгоритм Флойда-Уоршалла | | |
| Исходные данные:  На вход подается взвешенный ориентированный граф без ребер с отрицательным весом. | | |
|  | | |
|  | | |
|  | | |
| Дата сдачи реферата: 29.05.2017 | | |
| Дата защиты реферата: 29.05.2017 | | |
| Студентка |  | Кочнева О.Р. |
| Преподаватель |  | Фирсов М.А. |

**Аннотация**

В данной курсовой работе была произведена визуализация алгоритма поиска кратчайших путей в графе. Разработанная программа имеет удобный графический интерфейс, позволяющий пользователю построение графов без познаний о формах его представления (матрица смежности, матрица инцидентности и т.д), но для тестирования и удобства работы в приложении есть возможность построения графа по матрице смежности.

С построенным графом можно производить действия: перемещение вершин с целью достижения более удобного для пользователя вида графа и увеличение размеров графа. Помимо изображения графа имеется список всех ребер графа и соответственно вершин, которые они соединяют.

Во время работы алгоритма на графе происходит визуализация действий алгоритма в текстовом виде и на изображении. По окончании работы алгоритма можно посмотреть результат его работы в виде списка кратчайших путей (если таковые имеются) из каждой вершины во все остальные. Имеется возможность визуализировать на графе кратчайший путь для конкретной пары вершин.

Работа выполнена в программной среде: Qt 5.7

**содержание**

Оглавление

[**введение** 5](#_Toc483782590)

[**Цель работы.** 6](#_Toc483782591)

[**Задание** 6](#_Toc483782592)

[**Анализ задачи.** 6](#_Toc483782593)

[**Формальная постановка задачи.** 6](#_Toc483782594)

[**Спецификация программы.** 7](#_Toc483782595)

[**Описание функций и структур данных.** 12](#_Toc483782596)

[**Тестирование программы.** 18](#_Toc483782597)

[**заключение** 22](#_Toc483782598)

[**Код программы.** 23](#_Toc483782599)

# **введение**

В данной работе был рассмотрен алгоритм Флойда-Уоршалла. Данный алгоритм предназначен для работы со взвешенным ориентированным графом без ребер отрицательного веса. Алгоритм позволяет найти стоимость кратчайших пути между всеми ребрами графа, но так как эта информация не имеет особой ценности, если мы не знаем эти пути, то было принято решение ,помимо нахождения стоимостей, также сохранять пути их достижения.

# **Цель работы.**

Визуализация алгоритма поиска кратчайших путей в графе.

**Задание**

Вариант 10.

Алгоритм Флойда-Уоршалла. Визуализация.

# **Анализ задачи.**

Визуализация включает в себя.

1. Разработку пользовательского интерфейса
2. Написание спецификации программы
3. Визуализацию графа в виде изображения
4. Отображение работы алгоритма в графическом и текстовом виде
5. Предоставление результата работы программы в графическом и текстовом виде

# **Формальная постановка задачи.**

*исходные данные*

Вектор ребер хранящий данные о вершине из которой исходит ребро и вершине куда входит, а также о весе ребра. На основе него строятся необходимые для алгоритма матрицы:

Матрица смежности типа int[i][j], где отрицательные веса принимаются программой за бесконечности.

Матрица путей типа int[i][j], заполненная в соответствие с матрицей смежности.

*результирующие данные*

Матрица типа int[i][j], хранящая стоимость кратчайшего пути из i в j.

Матрица путей типа int[i][j], хранящая какую вершину надо посетить следующей, чтобы путь был кратчайшим.

*связь результатов и исходных данных*

В соответствии с алгоритмом программа изменяет матрицу смежности так, что получаем измененную матрицу смежности, являющуюся теперь матрицей стоимостей. В процессе изменения матрицы смежности в матрицу путей записываются вершины, которые надо посетить следующими, чтобы путь был кратчайшим.

# **Спецификация программы.**

*Ввод входных данных*

Ограничения на входные данные: до 50 узлов.

* С файла

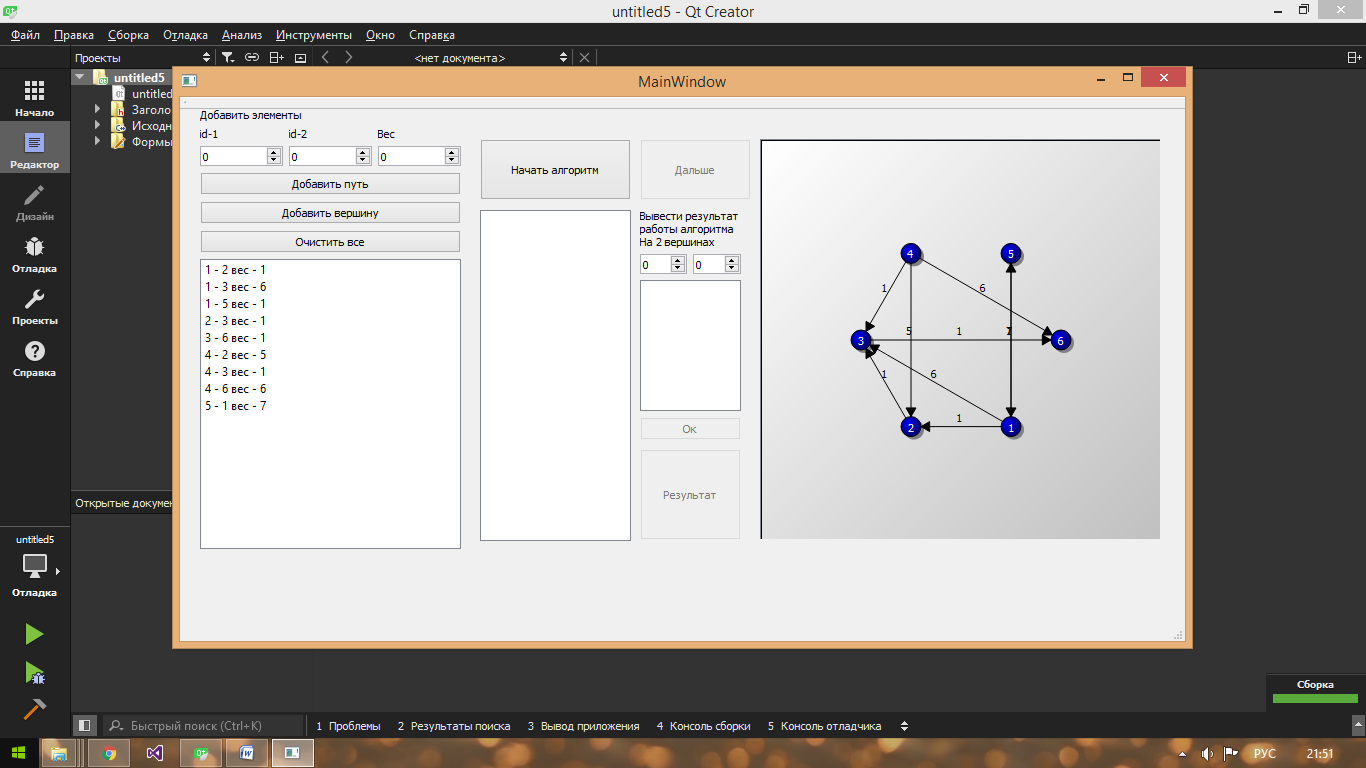
В 1 строке файла должно быть указано количество узлов графа. Затем следовать матрица смежности, в которой за бесконечность принимаются все отрицательные ячейки. Если файл не пуст и данные указаны корректно, то в окне при запуске программы мы увидим изображение графа в соответствие с матрицей смежности.

При вводе некорректных данных:

1. Ввели большее число узлов чем в матрице - матрица сместится, а значит и граф примет, не ожидаемый пользователем вид, но программа работать будет корректно,
2. Ввели символы в 1 строке. Программа будет работать в режиме пустого файла.
3. Ввели символы в месте, где должна быть матрица - в окне графа появится то количество узлов, которое указано в файле, но без связей. Связи всегда можно указать после в пользовательском интерфейсе

* В пользовательском интерфейсе

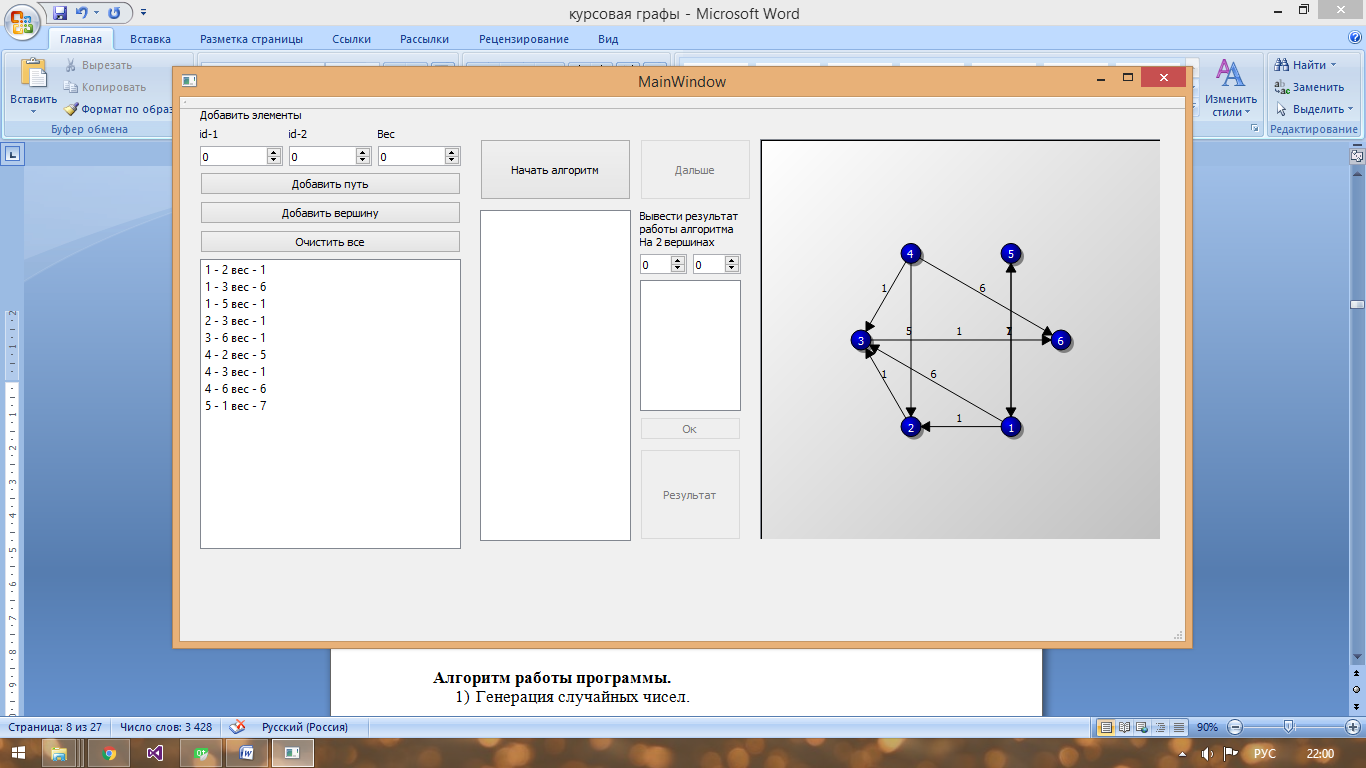
Для ввода данных разработано меню, которое расположено в левом верхнем углу (Рис. 1)



*Рис.1 Пользовательское меню*

Для создания новой вершины нажимаем кнопку “Добавить вершину”. В окне с изображением графа появится вершина с порядковым номером в соответствии с тем количеством, которое уже присутствует на экране.

Для создания ребра в окне “id-1” пишем, откуда выходит ребро, а в “id-2” куда входит, в окне “Вес” указываем вес ребра. После нажатия кнопки “Добавить ребро”. В окне с графом появится новое ребро и в окне под меню высветится новое ребро. См. Рис. 2.



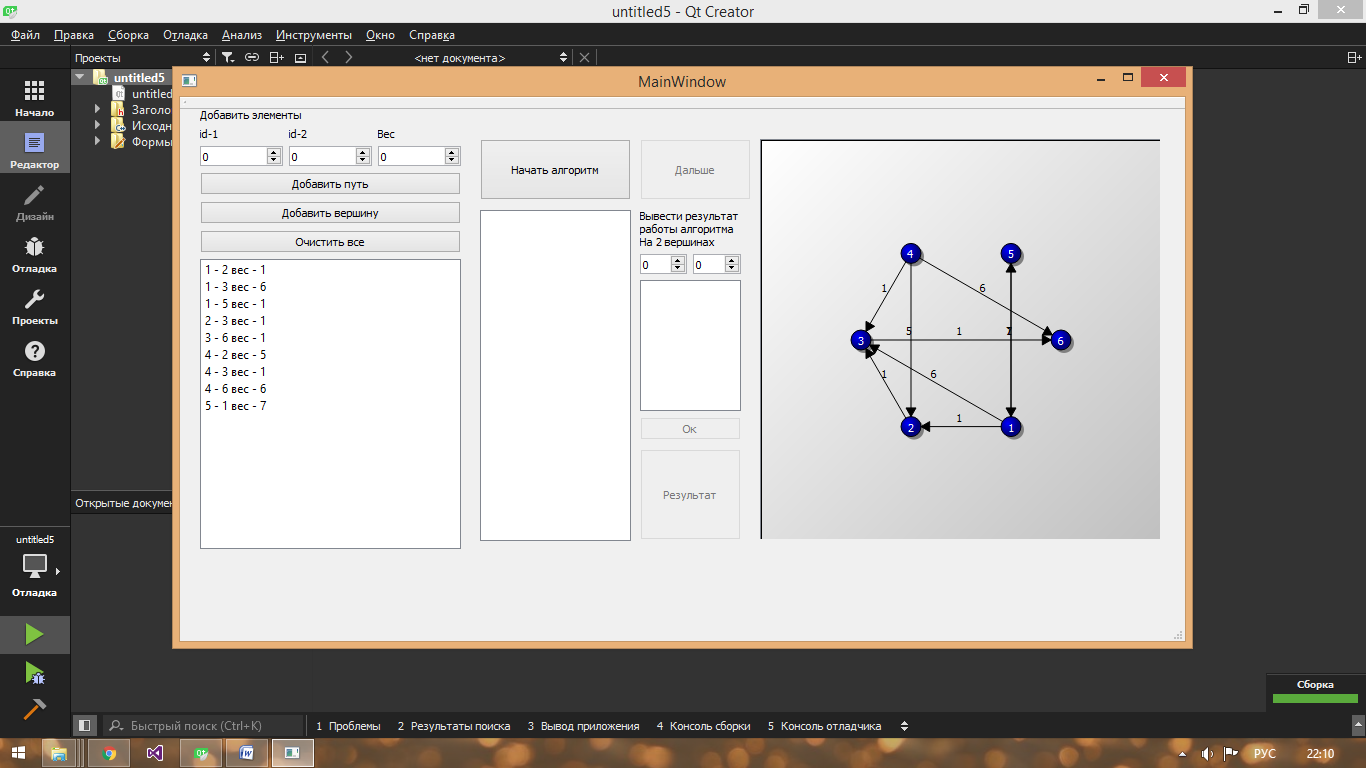
*Рис.2 Список ребер.*

При вводе некорректных данных:

1. Ввели 0 вес - не считается некорректным, программа закончит работу корректно
2. Попытались соединить несуществующие узлы. Ребро не добавится - программа проигнорирует данное действие.
3. Добавление ребра, если ребро уже существует. Существующее ребро заменится введенным.

*Работа с алгоритмом*

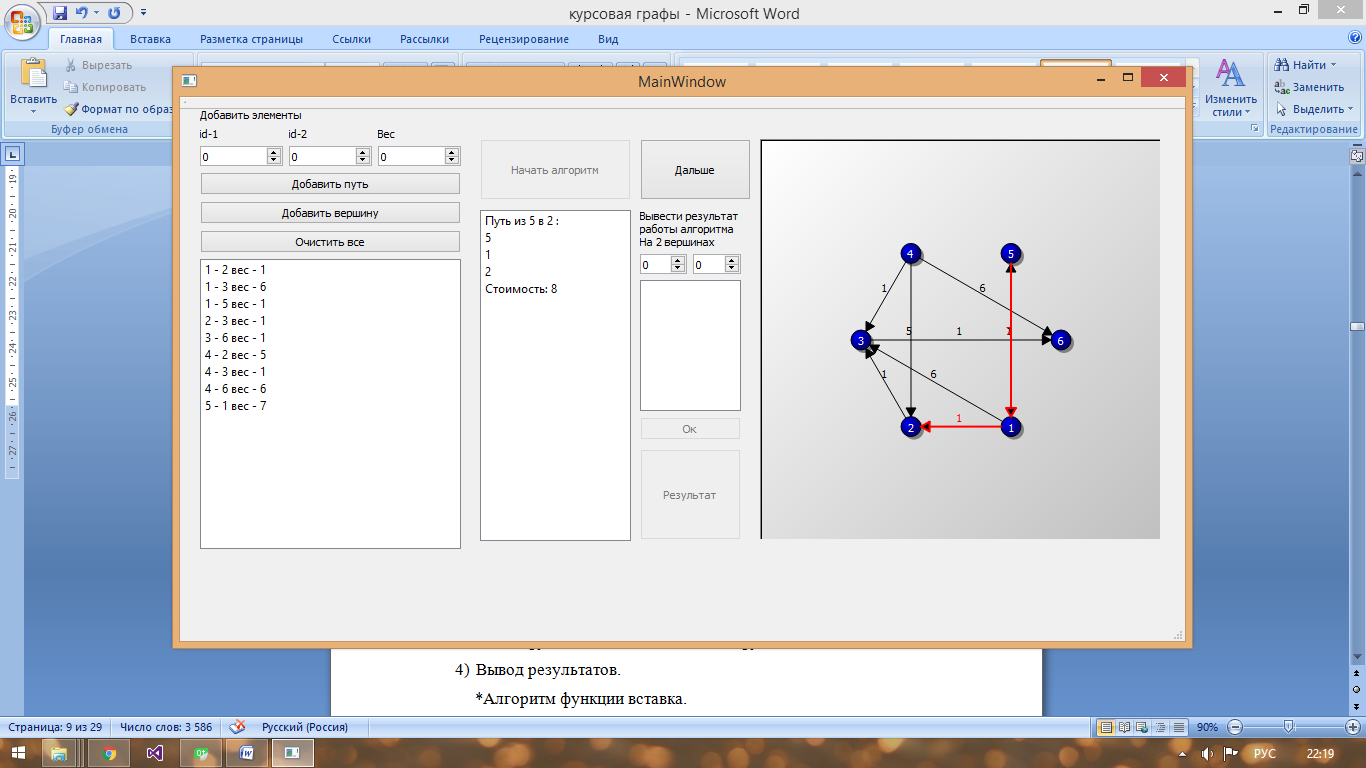
Если вы все данные были введены. Приступаем к работе с той частью меню, где производится работа с алгоритмом. См. Рис. 3



*Рис. 3 Меню работы с алгоритмом*

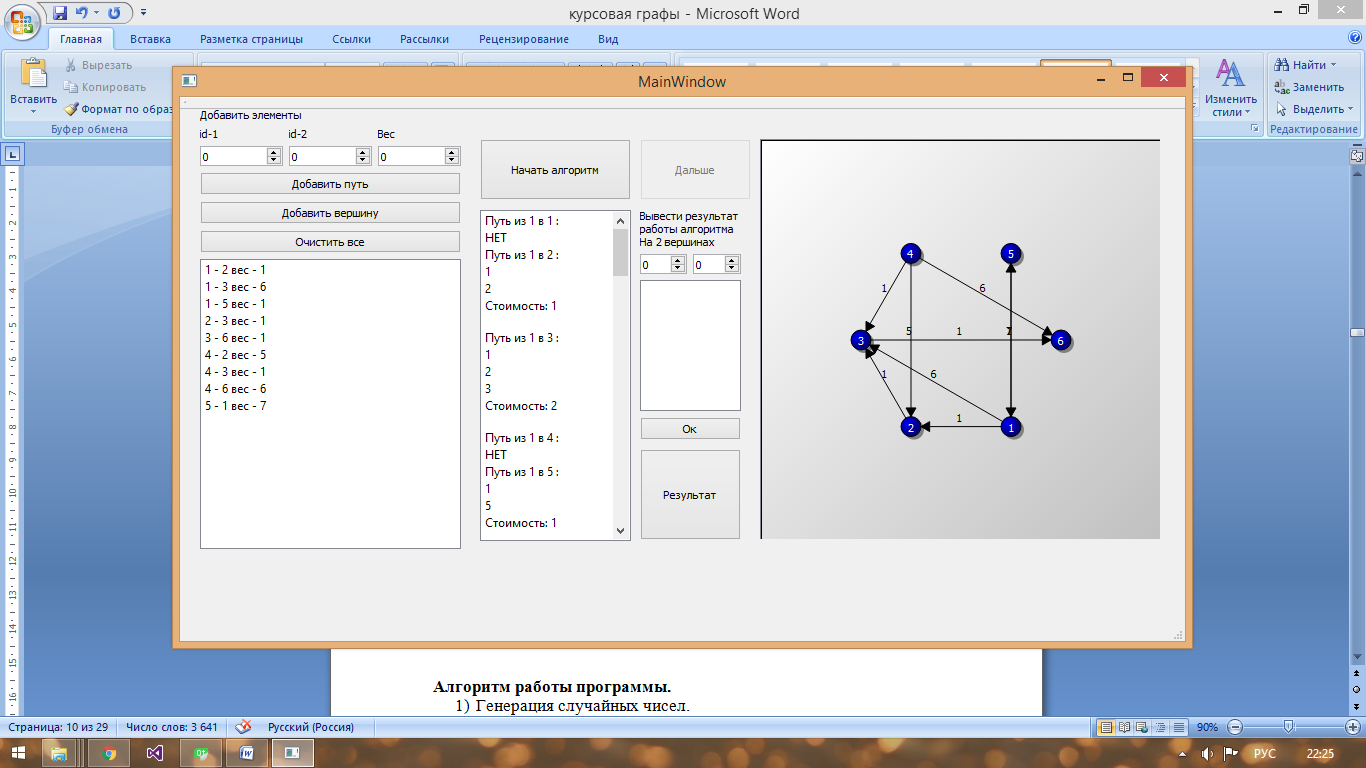
Для того, чтобы пользователь не испытывал затруднения, до начала работы только кнопка “Начать алгоритм” подсвечена, на все остальные кнопки нет возможности нажимать.

После нажатия кнопки “Начать алгоритм” подсветится кнопка “Дальше”, а “Начать алгоритм” – погаснет, так как не стоит начинать алгоритм, пока предыдущий раз не был завершен. Также в большом текстовом окне высветится 1 шаг алгоритма, а на изображении графа подсветится красным новый найденный путь. (см. Рис 4)



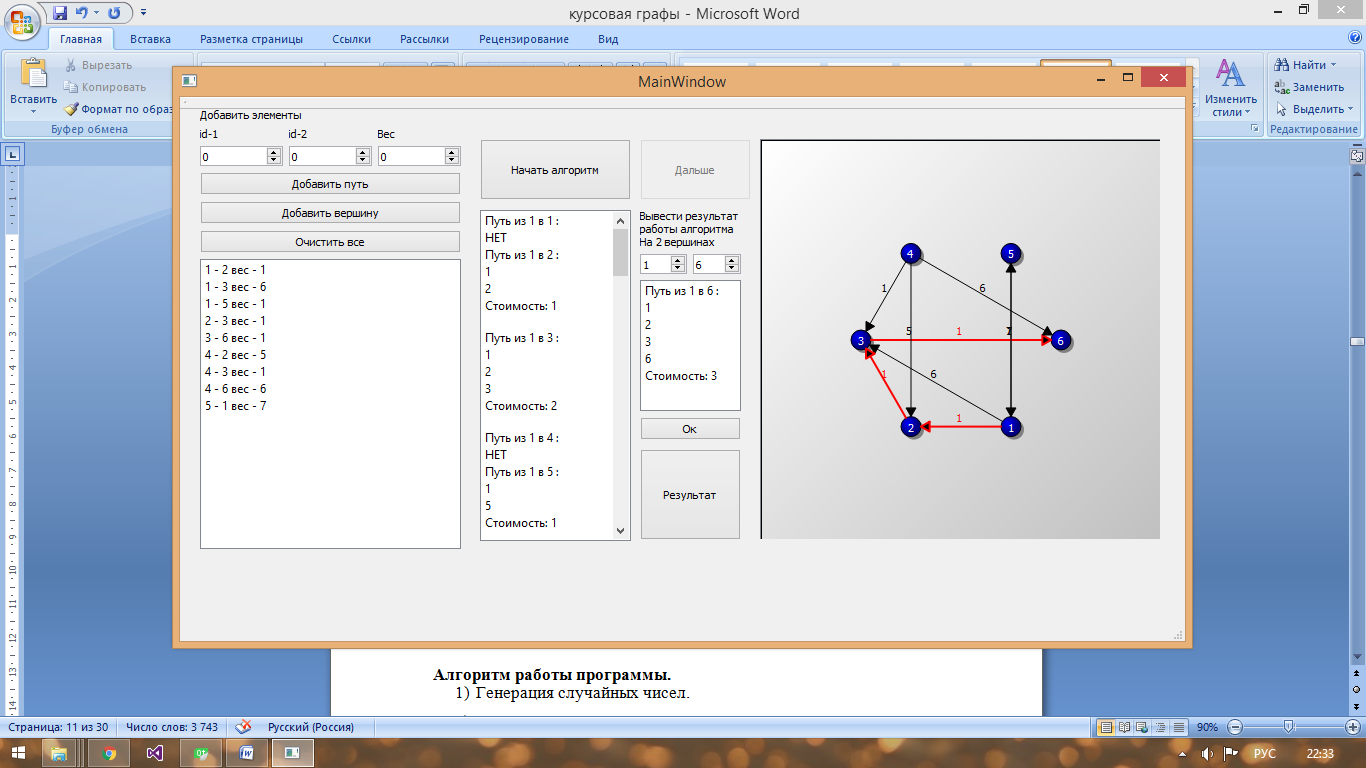
*Рис. 4 Шаги работы алгоритма.*

Чтобы увидеть следующий шаг – нажимаем кнопку “Дальше ” до тех пор, пока она не перестанет подсвечиваться, если она перестала подсвечиваться и загорелась кнопка “Результат ”, это означает, что алгоритм закончил свою работу. Нажимая кнопку “Результат ” в текстовом окне высвечиваются все возможные и невозможные кратчайшие пути, с указанием их стоимости. (См. Рис 5.)



*Рис. 5. Экран после нажатия кнопки “Результат”*

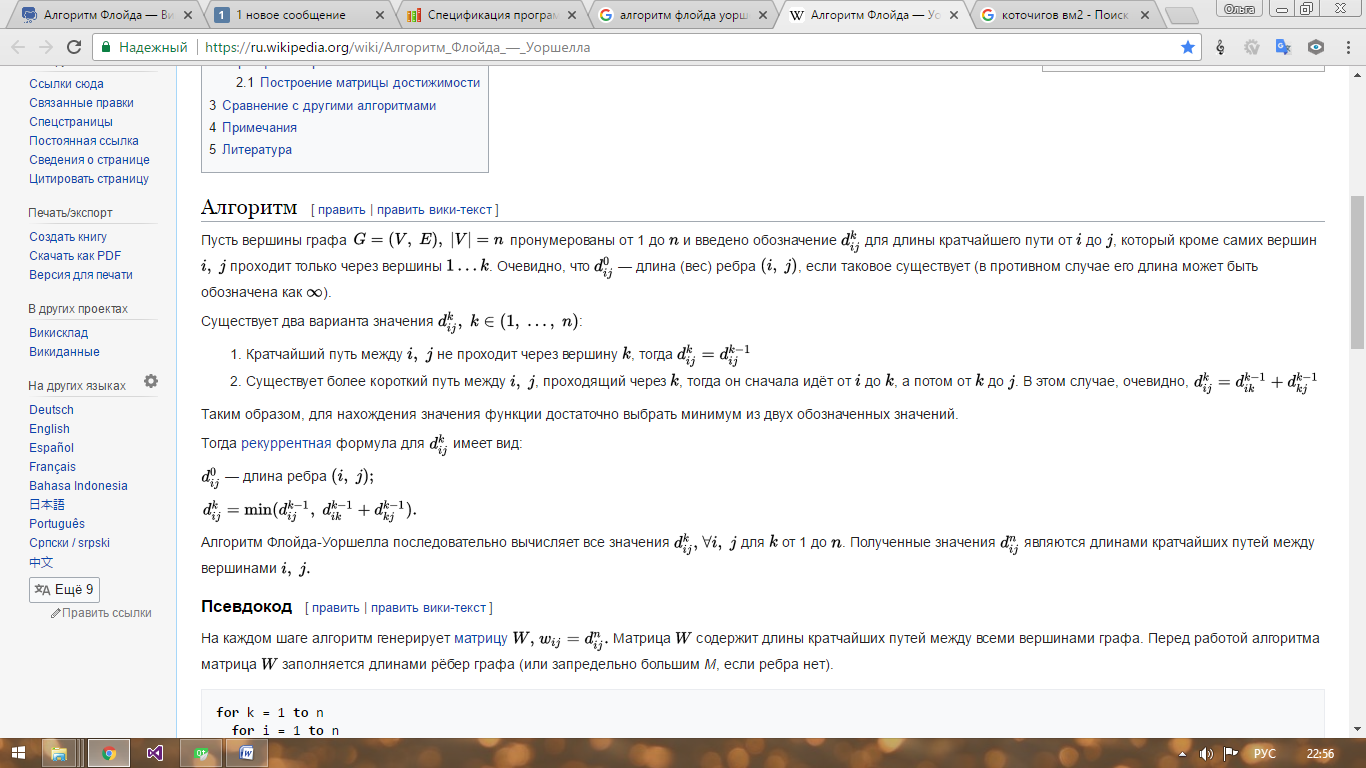
Теперь у пользователя есть возможность снова начать алгоритм, а также, что более существенно, увидеть кратчайший путь между конкретной парой вершин, для этого необходимо работать в меню, находящемся непосредственно над кнопкой “Результат ”. В 2 окнах, по аналогии, с тем как происходило создание нового пути в пользовательском интерфейсе, забиваем вершины откуда и куда мы хотим увидеть путь. После нажатия клавиши “Ок” на изображении графа подсветится данный путь, а в малом текстовом окне высветится список вершин в том порядке в каком их надо посетить а также стоимость прохождения по данному пути. (См. Рис. 6)

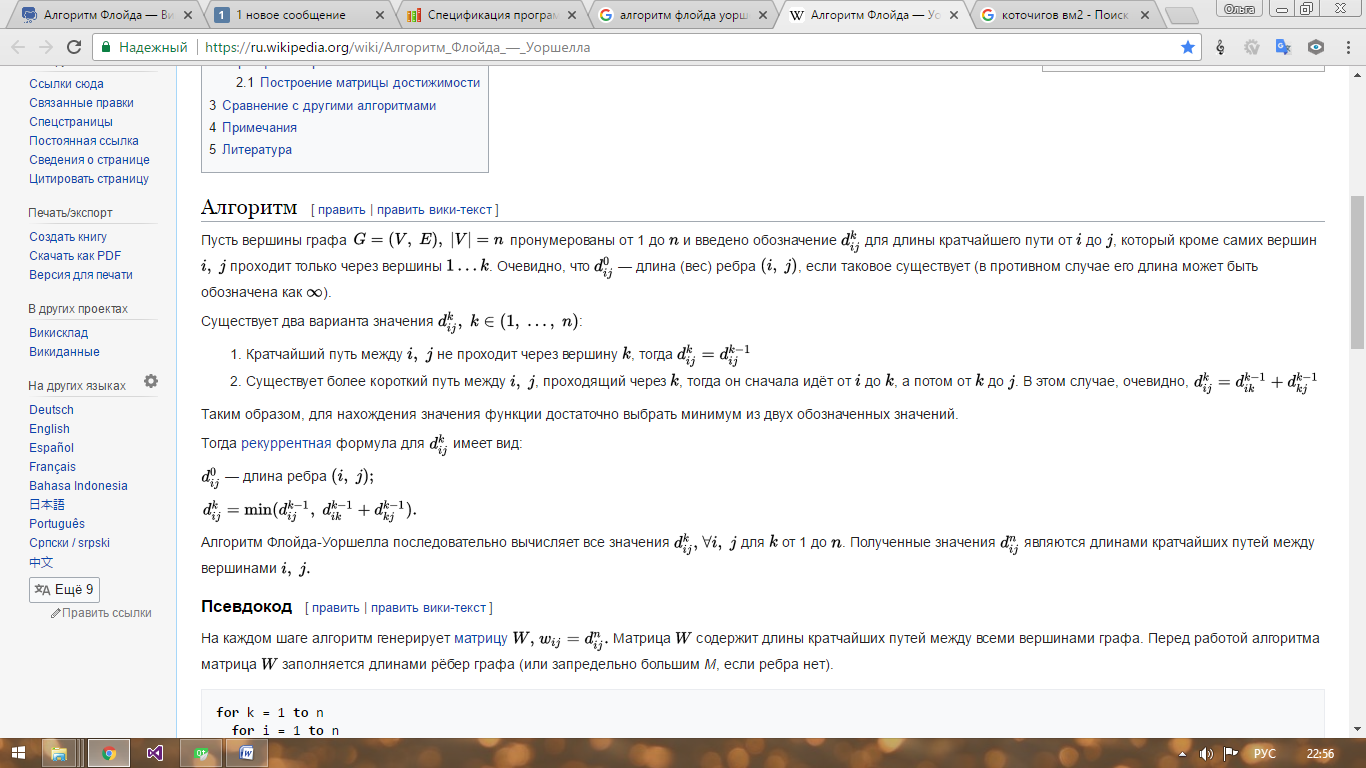
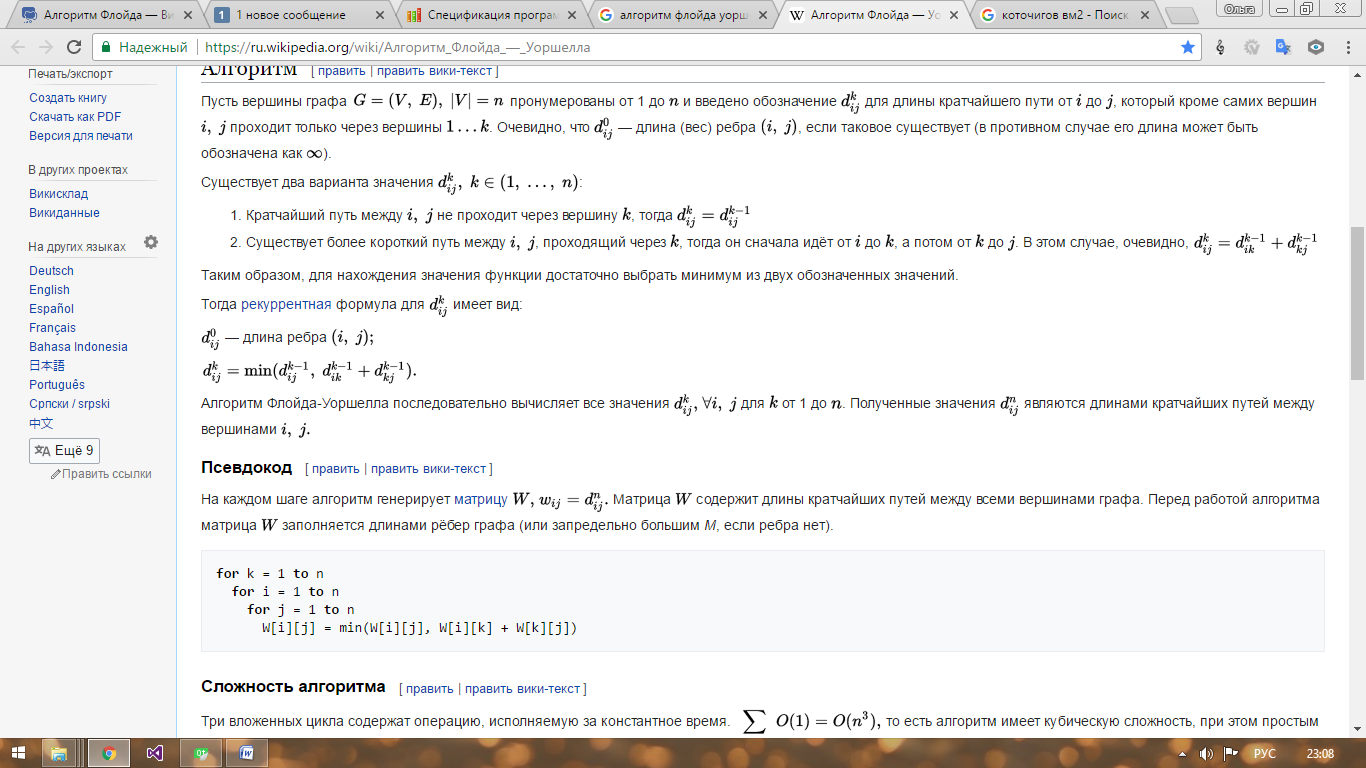


*Рис. 6. Кратчайший путь из 1 в 6 вершину.*

В случае необходимости можно стереть все данные, нажатием кнопки “Очистить все”.

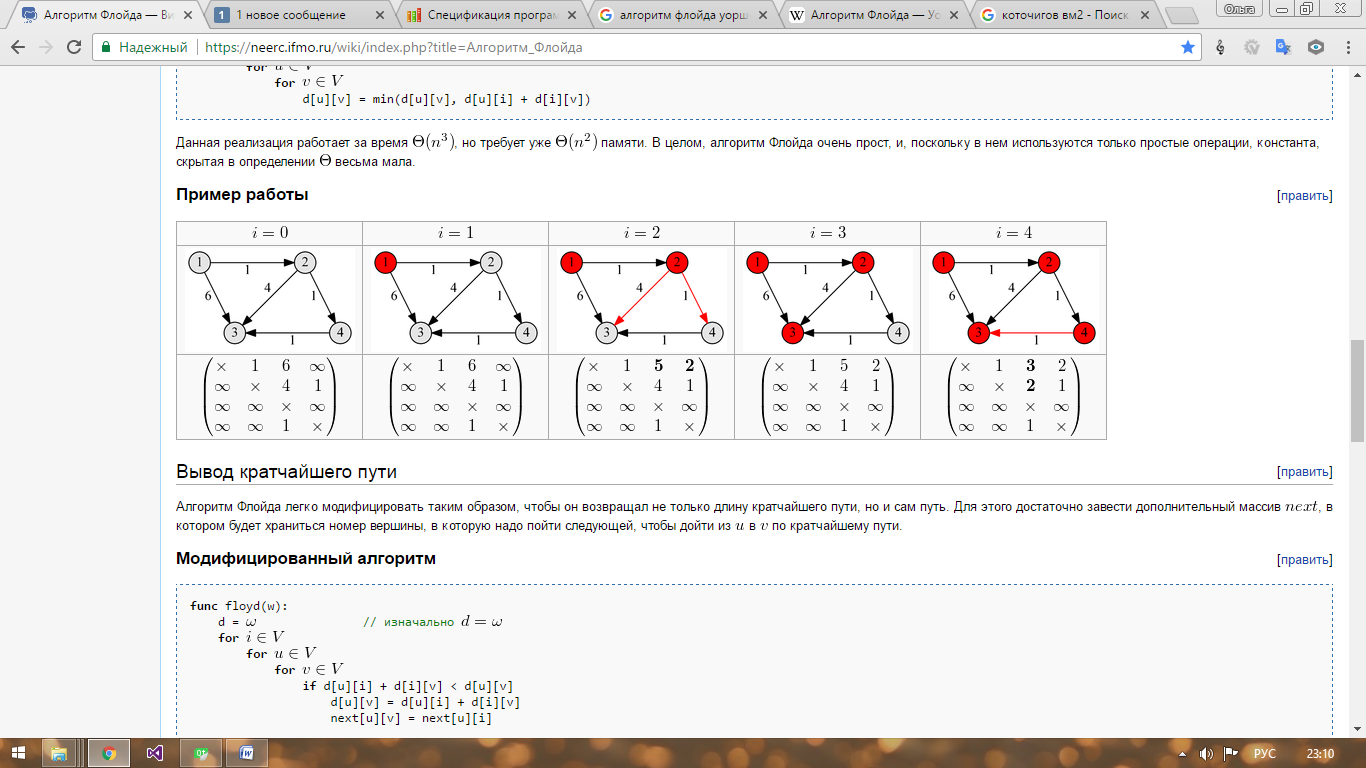
***Алгоритм работы программы.***

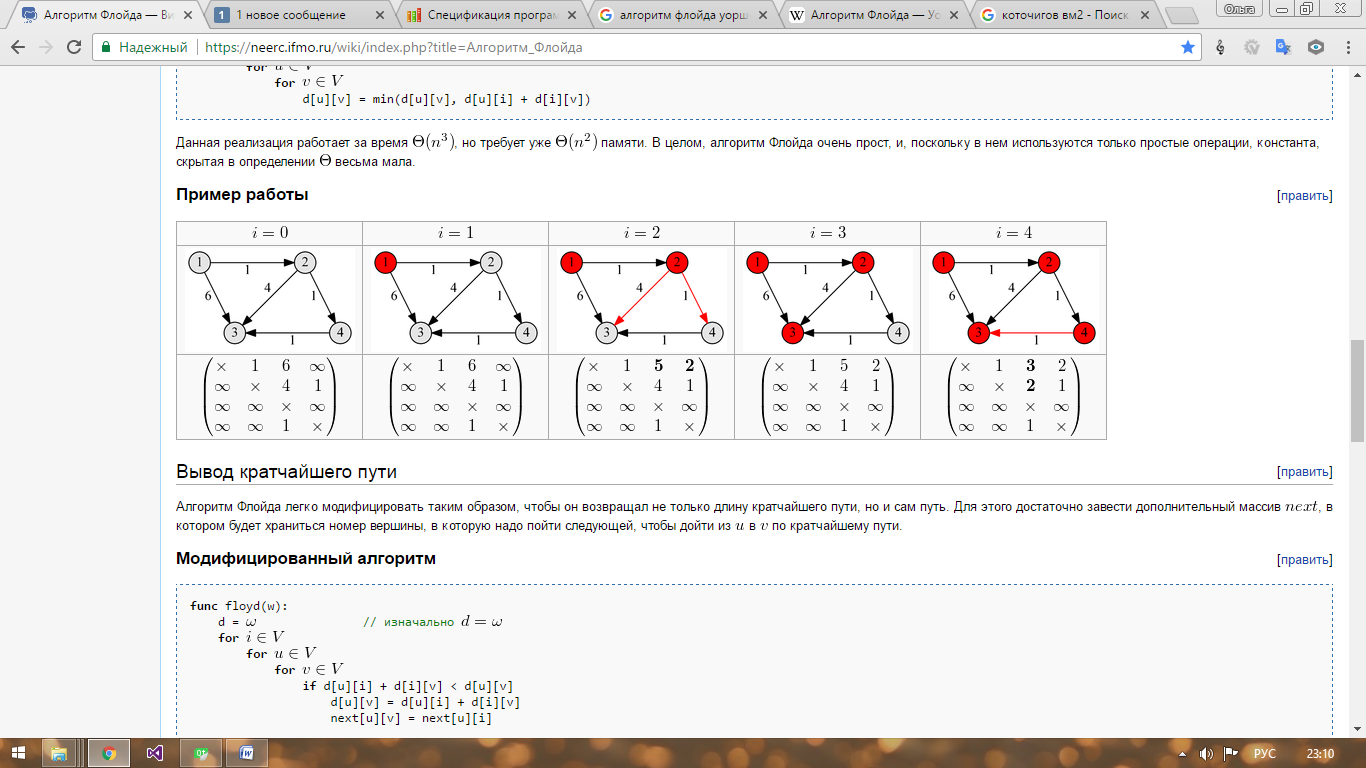
1. Построение графа.
2. Создание матрицы смежности на основе списка ребер и матрицы путей на основе матрицы смежности.
3. Подача матриц в алгоритм. Начало алгоритма.
4. Вычисление , где n это число узлов, выбором минимума из 2 случаев.

* Если кратчайший путь между i и j не проходит через k, то 
* Если существует более короткий путь между i и j, проходящий через k, тогда он сначала идет от i до k, а потом от i до j: 

1. Если существует более короткий путь, то запись этого пути в матрицу путей.
2. Вывод всех возможных путей и их стоимостей.

**Пример работы алгоритма:**

****

****

# **Описание функций и структур данных.**

*Описание классов*

**Класс Edge**

Объект ребро необходим для отображения ребер на графе.

class Edge : public QGraphicsItem

{

public:

Edge(Node \*sourceNode, Node \*destNode, int wt);

bool red;//окраска

int get\_weight();//возвращает вес вершины

Node \*sourceNode() const;//вершина начала

Node \*destNode() const;//вершина конца

void adjust();//чтобы начиналась в круге и кончалась там же

enum { Type = UserType + 2 };

int *type*() const override { return Type; }

protected:

QRectF *boundingRect*() const override;//занимаемое место треугольниками

void *paint*(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;

private:

Node \*source, \*dest;

int weight;//вес

QPointF sourcePoint;//координаты начала и конца

QPointF destPoint;

qreal arrowSize;//размеры стрелок

};

*Свойства:*

* Node \*source, \*dest;-указатели на вершины графа
* int weight;-вес ребра
* QPointF sourcePoint; - координаты начала и конца
* QPointF destPoint; - координаты конца
* qreal arrowSize; - размеры стрелок
* bool red;- окраска в ребра в красный, если true.

*Методы:*

* int get\_weight();-возвращает вес вершин
* Node \*sourceNode() const; - возвращает вершину начала ребра
* Node \*destNode() const; - возвращает вершину конца
* void adjust(); - отслеживает, чтобы стрелка начиналась в круге и кончалась там же
* QRectF *boundingRect*() const override; - возвращает занимаемое место треугольниками
* void *paint*(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;-рисует стрелку-ребро

**Node**

Класс узел-необходим для отображения узлов на графе.

class Node : public QGraphicsItem

{

public:

Node(GraphWidget \*graphWidget,int id);

void addEdge(Edge \*edge);//добавить ребро

QList<Edge \*> edges() const;

enum { Type = UserType + 1 };

int *type*() const override { return Type; }

QRectF *boundingRect*() const override;

QPainterPath *shape*() const override;

//рисуем

void *paint*(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;

protected:

QVariant *itemChange*(GraphicsItemChange change, const QVariant &value) override;

void *mousePressEvent*(QGraphicsSceneMouseEvent \*event) override;

void *mouseReleaseEvent*(QGraphicsSceneMouseEvent \*event) override;

private:

int id;

QList<Edge \*> edgeList;

QPointF newPos;

GraphWidget \*graph;

};

*Свойства:*

* int id;-индивидуальный номер вершины
* QPointF newPos;-новое расположение вершины при перемещении
* GraphWidget \*graph;-указатель на виджет, где происходит отрисовка графа

*Методы:*

* void addEdge(Edge \*edge);-добавить новое ребро, где Edge \*edge, указатель на член класса ребро
* QRectF *boundingRect*() const override;-возвращает границы изображения окружности вершины
* QPainterPath *shape*() const override;-создает фигуру эллипс
* void *paint*(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;-рисует фигуры
* void *mousePressEvent*(QGraphicsSceneMouseEvent \*event) override;-отслеживает нажатие клавиши, необходима для подсветки при нажатии на эллипс
* void *mouseReleaseEvent*(QGraphicsSceneMouseEvent \*event) override; -отслеживает удержание клавиши, необходима для подсветки при нажатии на эллипс

**GraphWidget**

Класс отвечающий за отрисовку графа в целом на виджете с изображением графа.

Данный класс использует 2 структуры данных: struct corner и struct tree.

struct tree-хранит указатель на объект типа Node, то есть на объект рисунок узла, и хранит индивидуальный номер узла. struct corner-хранит указатели на объекты типа tree, то есть на узлы откуда исходит и куда входит ребро, также хранит стоимость данного ребра и указатель на тип Edge \*elem, то есть на объект отрисовки ребра.

struct corner{

int weigth;

tree \*tr1,\*tr2;

Edge \*elem;

corner(tree \*tr1,tree \*tr2,Edge \*elem,int weig){

this->tr1=tr1;

this->tr2=tr2;

this->elem=elem;

weigth = weig;

}

};

struct tree{

int id;

Node \*elem;

tree(Node \*node,int id){

elem = node;

this->id=id;

}

};

class GraphWidget : public QGraphicsView

{

Q\_OBJECT

public:

GraphWidget(QWidget \*parent = 0);

void itemMoved();

//////////

void set\_tree\_id(){tree\_id=1;}//начинает отсчет заново

void next\_step(int k, int b);//рисует шаги

int Array[50][50];//массив данных

int path[50][50];//массив путей

void build\_matrix();//строим матрицу по данным

int numb(){return(vect\_tree.size());}//возвращает количество узлов

void all\_clear();//стирает с экрана цвет

const int inf = 1E9;//бесконечность

void add\_tree\_elem(int choice);

void add\_edge(int id1,int id2,int weigth);

void clear\_scene();//стереть все вообще

///////////

public slots:

void shuffle();

void zoomIn();

void zoomOut();

protected:

void *keyPressEvent*(QKeyEvent \*event) override;

#ifndef QT\_NO\_WHEELEVENT

void *wheelEvent*(QWheelEvent \*event) override;

#endif

void *drawBackground*(QPainter \*painter, const QRectF &rect) override;

void scaleView(qreal scaleFactor);

private:

struct corner;//хранит ребра

struct tree;//хранит узлы

tree\* get\_tree(int id);

int tree\_id;

struct corner{

int weigth;

tree \*tr1,\*tr2;

Edge \*elem;

corner(tree \*tr1,tree \*tr2,Edge \*elem,int weig){

this->tr1=tr1;

this->tr2=tr2;

this->elem=elem;

weigth = weig;

}

};

struct tree{

int id;

Node \*elem;

tree(Node \*node,int id){

elem = node;

this->id=id;

}

};

//списки путей и узлов

QVector<corner \*> vect\_corn;

QVector<tree\*> vect\_tree;

int timerId;

*Свойства*

* int Array[50][50];-массив смежности
* int path[50][50];массив путей
* struct corner;-тип ребра
* struct tree;-тип узел
* int tree\_id;-индивидуальный номер узла

struct corner

{

int weigth;-вес ребра

tree \*tr1,\*tr2;-указатели на представителей типа узел

Edge \*elem;-указатель на графический объект ребро

corner(tree \*tr1,tree \*tr2,Edge \*elem,int weig){

this->tr1=tr1;

this->tr2=tr2;

this->elem=elem;

weigth = weig;

}

};

struct tree{

int id; -индивидуальный номер узла

Node \*elem;-указатель на вершину

tree(Node \*node,int id){

elem = node;

this->id=id;

}

};

* QVector<corner \*> vect\_corn;-список ребер
* QVector<tree\*> vect\_tree;-список узлов
* const int inf = 1E9;- обозначаем бесконечность

*Методы*

* void set\_tree\_id();-начинает отсчет заново, если все стерли
* void next\_step(int k, int b);-рисует шаги на изображении графа, где int k-номер узла с началом ребра, int b- номер узла с концом ребра
* void build\_matrix();-построение матрицы по данным списка ребер
* int numb();-возвращает количество узлов
* void all\_clear();-стирает с экрана красный цвет
* void add\_tree\_elem(int choice);-добавляет новый узел
* void add\_edge(int id1,int id2,int weigth);-добавляет новый ребра
* void clear\_scene();-стирает все во всех рабочих окнах
* void *drawBackground*(QPainter \*painter, const QRectF &rect) override;-рисует фон
* tree\* get\_tree(int id);-возвращает тип узел по номеру

**MainWindow**

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

const int inf = 1E9;//бесконечность

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

void algorithm();//функция алгоритма

void show\_path(int k, int b);//вывод пути от k до b

void show\_result();//вывод результата в список

private slots:

void on\_pushButton\_clicked();//кнопка для добавления новой стрелки

void on\_pushButton\_2\_clicked();//кнопка для добавления нового узла

void on\_pushButton\_3\_clicked();//кнопка для очищения всего

void on\_pushButton\_4\_clicked();//начать алгоритм

void on\_pushButton\_5\_clicked();//дальше

void on\_pushButton\_6\_clicked();//вывод пути для 2 вершин

void on\_pushButton\_7\_clicked();//вывод результата

private:

QEventLoop\* myLoop\_ = new QEventLoop (this);

bool show\_to\_small=false;

Ui::MainWindow \*ui;

};

*Свойства*

* bool show\_to\_small=false;-отвечает за вывод информации на маленькое текстовое окно

*Слоты*

* void on\_pushButton\_clicked();-кнопка для добавления нового ребра
* void on\_pushButton\_2\_clicked();-кнопка для добавления нового узла
* void on\_pushButton\_3\_clicked();-кнопка для очищения всего
* void on\_pushButton\_4\_clicked();-кнопка начать алгоритм
* void on\_pushButton\_5\_clicked();-кнопка дальше
* void on\_pushButton\_6\_clicked();- кнопка Ок для вывода пути для 2 вершин
* void on\_pushButton\_7\_clicked();-кнопка для вывода результата

*Методы*

* void algorithm();-функция алгоритма
* void show\_path(int k, int b);-вывод пути от k до b в маленькое текстовое окно и во время работы алгоритма на каждом шаге
* void show\_result();-вывод результата в список в большое текстовое окно

# **Тестирование программы.**

|  |  |  |
| --- | --- | --- |
| **Входные данные** | | **Результат** |
| **Удачные** | | |
| 1  0 | Из 1 в 1- Нет | |
| 2  0 1  6 -1 | Из 1 в 1- Нет  Из 1 в 2 - 1 2 Стоимость 1  Из 2 в 1 - 2 1 Стоимость 6  Из 2 в 2- Нет | |
| 3  0 1 6  -1 1 -1  0 4 1 | Из 1 в 1- Нет  Из 1 в 2 - 1 2 Стоимость 1  Из 1 в 3 - 1 3 Стоимость 6  Из 2 в 1- Нет  Из 2 в 2 - Нет  Из 2 в 3- Нет  Из 3 в 1- Нет  Из 3 в 2 - 3 2 Стоимость 4  Из 3 в 3 - Нет | |
| 4  0 1 6 -1  1 -1 0 4  1 -2-1 -1  0 -1 4 -2 | Из 1 в 1- Нет  Из 1 в 2 - 1 2 Стоимость 1  Из 1 в 3 - 1 3 Стоимость 6  Из 1 в 4- 1 2 4 Стоимость 5  Из 2 в 1 - 2 1 Стоимость 1  Из 2 в 2- Нет  Из 2 в 3 - 2 1 3 Стоимость 7  Из 2 в 4- 2 4 Стоимость 4  Из 3 в 1 - 3 1 Стоимость 1  Из 3 в 2- 3 1 2 Стоимость 2  Из 3 в 3- Нет  Из 3 в 4 -3 1 2 4 Стоимость 6  Из 4 в 1 – 4 3 1 Стоимость 5  Из 4 в 2- 4 3 1 2 Стоимость 6  Из 4 в 3 – 4 3 Стоимость 4  Из 4 в 4- Нет | |
| 4  0 1 6 -1  -1 0 4 1  -1 -1 0 -1  -2 -1 1 -1 | Из 1 в 1- Нет  Из 1 в 2 - 1 2 Стоимость 1  Из 1 в 3 - 1 2 4 3 Стоимость 3  Из 1 в 4- 1 2 4 Стоимость 2  Из 2 в 1 - Нет  Из 2 в 2- Нет  Из 2 в 3 - 2 4 3 Стоимость 2  Из 2 в 4- 2 4 Стоимость 1  Из 3 в 1 - Нет  Из 3 в 2- Нет  Из 3 в 3- Нет  Из 3 в 4 - Нет  Из 4 в 1 – Нет  Из 4 в 2- Нет  Из 4 в 3 – 4 3 Стоимость 1  Из 4 в 4- Нет | |
| 5  0 1 6 -1 1  -1 0 4 1 -2  -1 -1 0 -1 4  -2 -1 1 -1 5  1 3 -1 6 7 | Из 1 в 1- Нет  Из 1 в 2 - 1 2 Стоимость 1  Из 1 в 3- 1 2 4 3 Стоимость 3  Из 1 в 4 - 1 2 4 Стоимость 2  Из 1 в 5- 1 5 Стоимость 1  Из 2 в 1 - 2 4 5 1 Стоимость 7  Из 2 в 2- Нет  Из 2 в 3 – 2 4 3 Стоимость 2  Из 2 в 4- 2 4 Стоимость 1  Из 2 в 5 – 2 4 5 Стоимость 6  Из 3 в 1- 3 5 1 Стоимость 5  Из 3 в 2 – 3 5 1 2 Стоимость 6  Из 3 в 3- Нет  Из 3 в 4 – 3 5 1 2 4 Стоимость 7  Из 3 в 5- 3 5 Стоимость 4  Из 4 в 1 – 4 5 1 Стоимость 6  Из 4 в 2- 4 5 1 2 Стоимость 7  Из 4 в 3 - 4 3 Стоимость 1  Из 4 в 4- Нет  Из 4 в 5 - 4 5 Стоимость 5  Из 5 в 1- Нет  Из 5 в 2 - 1 2 Стоимость 1  Из 5 в 3- Нет  Из 5 в 4 - 1 2 Стоимость 1  Из 5 в 5- Нет | |

|  |  |
| --- | --- |
| **Неудачные** | |
| 3 узла без ребер | Из 1 в 1- Нет  Из 1 в 2 - Нет  Из 1 в 3 - Нет  Из 2 в 1- Нет  Из 2 в 2 - Нет  Из 2 в 3- Нет  Из 3 в 1- Нет  Из 3 в 2 - Нет  Из 3 в 3 - Нет |

# **заключение**

В данной работе была реализована визуализация алгоритма поиска кратчайшего пути Флойда-Уоршалла. Был разработан пользовательский интерфейс и пошаговое изображение работы алгоритма.

# **Код программы.**

**edge.h**

#ifndef EDGE\_H

#define EDGE\_H

#include <QGraphicsItem>

class Node;

class Edge : public QGraphicsItem

{

public:

Edge(Node \*sourceNode, Node \*destNode, int wt);

bool red;//окраска

int get\_weight();//возвращает вес вершины

Node \*sourceNode() const;//вершина начала

Node \*destNode() const;//вершина конца

void adjust();//чтобы начиналась в круге и кончалась там же

enum { Type = UserType + 2 };

int *type*() const override { return Type; }

protected:

QRectF *boundingRect*() const override;//занимаемое место треугольниками

void *paint*(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;

private:

Node \*source, \*dest;

int weight;//вес

QPointF sourcePoint;//координаты начала и конца

QPointF destPoint;

qreal arrowSize;//размеры стрелок

};

#endif // EDGE\_H

**node.h**

#ifndef NODE\_H

#define NODE\_H

#include <QGraphicsItem>

#include <QList>

class Edge;

class GraphWidget;

QT\_BEGIN\_NAMESPACE

class QGraphicsSceneMouseEvent;

QT\_END\_NAMESPACE

//! [0]

class Node : public QGraphicsItem

{

public:

Node(GraphWidget \*graphWidget,int id);

void addEdge(Edge \*edge);//добавить ребро

QList<Edge \*> edges() const;

enum { Type = UserType + 1 };

int *type*() const override { return Type; }

QRectF *boundingRect*() const override;

QPainterPath *shape*() const override;

//рисуем

void *paint*(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;

protected:

QVariant *itemChange*(GraphicsItemChange change, const QVariant &value) override;

void *mousePressEvent*(QGraphicsSceneMouseEvent \*event) override;

void *mouseReleaseEvent*(QGraphicsSceneMouseEvent \*event) override;

private:

int id;

QList<Edge \*> edgeList;

QPointF newPos;

GraphWidget \*graph;

};

//! [0]

#endif // NODE\_H

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include<QMessageBox>

#include<QEventLoop>

namespace Ui {

class MainWindow;

class algorithm;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

const int inf = 1E9;//бесконечность

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

void algorithm();//функция алгоритма

void show\_path(int k, int b);//вывод пути от k до b

void show\_result();//вывод результата в список

private slots:

void on\_pushButton\_clicked();//кнопка для добавления новой стрелки

void on\_pushButton\_2\_clicked();//кнопка для добавления нового узла

void on\_pushButton\_3\_clicked();//кнопка для очищения всего

void on\_pushButton\_4\_clicked();//начать алгоритм

void on\_pushButton\_5\_clicked();//дальше

void on\_pushButton\_6\_clicked();//вывод пути для 2 вершин

void on\_pushButton\_7\_clicked();//вывод результата

private:

QEventLoop\* myLoop\_ = new QEventLoop (this);

bool show\_to\_small=false;

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**graphwidget.h**

#ifndef GRAPHWIDGET\_H

#define GRAPHWIDGET\_H

#include <QGraphicsView>

#include<fstream>

#include<QTextStream>

#include "edge.h"

#include "QVector"

class Node;

class GraphWidget : public QGraphicsView

{

Q\_OBJECT

public:

GraphWidget(QWidget \*parent = 0);

void itemMoved();

//////////

void set\_tree\_id(){tree\_id=1;}//начинает отсчет заново

void next\_step(int k, int b);//рисует шаги

int Array[50][50];//массив данных

int path[50][50];//массив путей

void build\_matrix();//строим матрицу по данным

int numb(){return(vect\_tree.size());}//возвращает количество узлов

void all\_clear();//стирает с экрана цвет

const int inf = 1E9;//бесконечность

void add\_tree\_elem(int choice);

void add\_edge(int id1,int id2,int weigth);

void clear\_scene();//стереть все вообще

///////////

public slots:

void shuffle();

void zoomIn();

void zoomOut();

protected:

void *keyPressEvent*(QKeyEvent \*event) override;

#ifndef QT\_NO\_WHEELEVENT

void *wheelEvent*(QWheelEvent \*event) override;

#endif

void *drawBackground*(QPainter \*painter, const QRectF &rect) override;

void scaleView(qreal scaleFactor);

private:

struct corner;//хранит пути

struct tree;//хранит узлы

tree\* get\_tree(int id);

int tree\_id;

struct corner{

int weigth;

tree \*tr1,\*tr2;

Edge \*elem;

corner(tree \*tr1,tree \*tr2,Edge \*elem,int weig){

this->tr1=tr1;

this->tr2=tr2;

this->elem=elem;

weigth = weig;

}

};

struct tree{

int id;

Node \*elem;

tree(Node \*node,int id){

elem = node;

this->id=id;

}

};

//списки путей и узлов

QVector<corner \*> vect\_corn;

QVector<tree\*> vect\_tree;

int timerId;

};

#endif // GRAPHWIDGET\_H

**edge.cpp**

#include "edge.h"

#include "node.h"

#include <math.h>

#include <QPainter>

static const double Pi = 3.14159265358979323846264338327950288419717;

static double TwoPi = 2.0 \* Pi;

Edge::Edge(Node \*sourceNode, Node \*destNode, int wt)

: arrowSize(10)

{

red=0;

this->weight=wt;

setAcceptedMouseButtons(0);

source = sourceNode;

dest = destNode;

source->addEdge(this);

dest->addEdge(this);

adjust();

}

Node \*Edge::sourceNode() const

{

return source;

}

Node \*Edge::destNode() const

{

return dest;

}

void Edge::adjust()

{

if (!source || !dest)

return;

QLineF line(mapFromItem(source, 0, 0), mapFromItem(dest, 0, 0));

qreal length = line.length();

prepareGeometryChange();

if (length > qreal(20.)) {

QPointF edgeOffset((line.dx() \* 10) / length, (line.dy() \* 10) / length);

sourcePoint = line.p1() + edgeOffset;

destPoint = line.p2() - edgeOffset;

} else {

sourcePoint = destPoint = line.p1();

}

}

QRectF Edge::*boundingRect*() const

{

if (!source || !dest)

return QRectF();

qreal penWidth = 1;

qreal extra = (penWidth + arrowSize) / 2.0;

return QRectF(sourcePoint, QSizeF(destPoint.x() - sourcePoint.x(),

destPoint.y() - sourcePoint.y()))

.normalized()

.adjusted(-extra, -extra, extra, extra);

}

void Edge::*paint*(QPainter \*painter, const QStyleOptionGraphicsItem \*, QWidget \*)

{

if (!source || !dest)

return;

QLineF line(sourcePoint, destPoint);

if (qFuzzyCompare(line.length(), qreal(0.)))

return;

// Draw the line itself

if(red) {painter->setPen(QPen(Qt::red, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));}

else{painter->setPen(QPen(Qt::black, 1, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));}

painter->drawLine(line);

// Draw the arrows

double angle = ::acos(line.dx() / line.length());

if (line.dy() >= 0)

angle = TwoPi - angle;

//для вывода наконечников стрел и весов

QPointF sourceArrowP1 = sourcePoint + QPointF(sin(angle + Pi / 3) \* arrowSize,

cos(angle + Pi / 3) \* arrowSize);

//против часовой

QPointF destArrowP1 = destPoint + QPointF(sin(angle - Pi / 3) \* arrowSize,

cos(angle - Pi / 3) \* arrowSize);

QPointF destArrowP2 = destPoint + QPointF(sin(angle - Pi + Pi / 3) \* arrowSize,

cos(angle - Pi + Pi / 3) \* arrowSize);

painter->drawText((sourceArrowP1+destArrowP2)/2+QPoint(-5,-5), QString::number(weight) );

painter->setBrush(Qt::black);

painter->drawPolygon(QPolygonF() <<line.p2() << destArrowP1 << destArrowP2);

}

int Edge::get\_weight(){

return weight;

}

**node.cpp**

#include "edge.h"

#include "node.h"

#include "graphwidget.h"

#include <QGraphicsScene>

#include <QGraphicsSceneMouseEvent>

#include <QPainter>

#include <QStyleOption>

Node::Node(GraphWidget \*graphWidget , int id)

: graph(graphWidget)

{

this->id=id;

setFlag(ItemIsMovable);

setFlag(ItemSendsGeometryChanges);

setCacheMode(DeviceCoordinateCache);

setZValue(-1);

}

void Node::addEdge(Edge \*edge)

{

edgeList << edge;

edge->adjust();

}

QList<Edge \*> Node::edges() const

{

return edgeList;

}

QRectF Node::*boundingRect*() const

{

qreal adjust = 2;

return QRectF( -10 - adjust, -10 - adjust, 23 + adjust, 23 + adjust);

}

QPainterPath Node::*shape*() const

{

QPainterPath path;

path.addEllipse(-10, -10, 20, 20);

return path;

}

void Node::*paint*(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*)

{

painter->setPen(Qt::NoPen);

painter->setBrush(Qt::darkGray);

painter->drawEllipse(-7, -7, 20, 20);

QRadialGradient gradient(-3, -3, 10);

if (option->state & QStyle::State\_Sunken) {//подсветка при клике

gradient.setCenter(3, 3);

gradient.setFocalPoint(3, 3);

gradient.setColorAt(1, QColor(Qt::blue).light(120));

gradient.setColorAt(0, QColor(Qt::darkBlue).light(120));

} else {

gradient.setColorAt(0, Qt::blue);

gradient.setColorAt(1, Qt::darkBlue);

}

painter->setBrush(gradient);

painter->setPen(QPen(Qt::black, 0));

painter->drawEllipse(-10, -10, 20, 20);

painter->setPen(QPen(Qt::white, 0));

painter->drawText(-3,5, QString::number(id) );

}

QVariant Node::*itemChange*(GraphicsItemChange change, const QVariant &value)

{

foreach (Edge \*edge, edgeList)

edge->adjust();

return QGraphicsItem::*itemChange*(change, value);

}

void Node::*mousePressEvent*(QGraphicsSceneMouseEvent \*event)

{

update();

QGraphicsItem::*mousePressEvent*(event);

}

void Node::*mouseReleaseEvent*(QGraphicsSceneMouseEvent \*event)

{

update();

QGraphicsItem::*mouseReleaseEvent*(event);

}

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "graphwidget.h"

#include "QListView"

#include "QTime"

#include <iostream>

GraphWidget \*widget;

QListWidget \*list;

QListWidget \*list2;

QListWidget \*list3;

void add\_edge(int id1,int id2,int weig);

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

list = ui->listWidget;

list2=ui->listWidget\_2;

list3=ui->listWidget\_3;

ui->pushButton\_6->setEnabled(false);

ui->pushButton\_5->setEnabled(false);

ui->pushButton\_7->setEnabled(false);

widget = new GraphWidget(ui->graphicsView);

std::ifstream file("Z:\\input.txt");

int n=0;

file>>n;

if(n>=49)

{n=49;}

for(int j=0;j<n;j++)

{

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

{

file>>widget->path[j][i];

if((widget->path[j][i]>0)&&(i!=j))

{

add\_edge(j+1,i+1,widget->path[j][i]);//при некорректных данных связи не выводит

}

}

}

}

MainWindow::~*MainWindow*()

{

delete ui;

}

void add\_edge(int id1,int id2,int weig){//добавить новую путь в список новых вершин

widget->add\_edge(id1,id2,weig);

list->addItem(QString::number(id1)+" - "+QString::number(id2)+" вес - "+QString::number(weig));

}

void MainWindow::on\_pushButton\_clicked()//добавить новую стрелку

{

int id1 = ui->spinBox\_3->value();

int id2 = ui->spinBox\_2->value();

if (id1==id2) return;

int weig = ui->spinBox->value();

ui->listWidget->addItem(QString::number(id1)+" - "+QString::number(id2)+" вес - "+QString::number(weig));

widget->add\_edge(id1,id2,weig);

}

void MainWindow::on\_pushButton\_2\_clicked()//добавить элемент

{

if(widget->numb()>=49)

{return;}

else

{widget->add\_tree\_elem(0);}

}

void MainWindow::on\_pushButton\_3\_clicked()//очистить все

{

widget->clear\_scene();//очищаем рисунок

widget->set\_tree\_id();//ставим нумерацию вершин в начало

list->clear();//очищаем все списки

list3->clear();

list2->clear();

ui->pushButton\_4->setEnabled(true);//кнопка начать алгоритм-доступна

ui->pushButton\_6->setEnabled(false);//ок-недоступна

ui->pushButton\_5->setEnabled(false);//дальше-недоступна

ui->pushButton\_7->setEnabled(false);//резудьтат недоступна

}

void MainWindow::on\_pushButton\_4\_clicked()//нажата кнопка начать алгоритм

{

list2->clear();

ui->pushButton\_4->setEnabled(false);

widget->build\_matrix();

ui->pushButton\_5->setEnabled(true);

algorithm();

}

void MainWindow ::on\_pushButton\_5\_clicked()//нажата кнопка дальше

{

myLoop\_->quit ();

}

void MainWindow::on\_pushButton\_6\_clicked()//нажата кнопка ок для вывода вершин

{

list3->clear();

show\_to\_small=true;

int id1 = ui->spinBox\_4->value();

int id2 = ui->spinBox\_5->value();

if ((id1==id2)||(id1>widget->numb())||(id2>widget->numb())||id1==0||id2==0) return;

else

{

show\_path(id1-1,id2-1);

widget->next\_step(id1-1,id2-1);

}

show\_to\_small=false;

}

void MainWindow::on\_pushButton\_7\_clicked()//нажата кнопка для вывода результата

{

widget->all\_clear();

widget->scene()->update();

list2->clear();

ui->pushButton\_4->setEnabled(true);

show\_result();

ui->pushButton\_6->setEnabled(true);

}

void MainWindow::algorithm()//алгоритм работы программы

{

ui->listWidget\_4->addItem("Начальная матрица");

int N=widget->numb();//число элементов

for (int k = 0; k < N; ++k)

{

QString list\_mass;

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

{

if(widget->Array[k][i]==inf)

{

list\_mass.push\_back("x");

}

else

{ list\_mass.push\_back(QString::number(widget->Array[k][i]));}

list\_mass.push\_back(" ");

for (int j = 0; j < N; ++j)

{

if ((widget->Array[i][k] < inf) && (widget->Array[k][j] < inf )&&

(widget->Array[i][k] + widget->Array[k][j]<widget->Array[i][j])) {

widget-> Array[i][j] = widget->Array[i][k] + widget->Array[k][j];

widget-> path[i][j] = widget->path[i][k];

show\_path(i,j);

widget->next\_step(i,j);

myLoop\_->exec();

}

}

}

list\_mass.push\_back("\n");

ui->listWidget\_4->addItem(list\_mass);

}

ui->pushButton\_5->setEnabled(false);//по окончании нельзя нажать дальше

ui->pushButton\_7->setEnabled(true);//по окончании можно нажать результат

void MainWindow::show\_result()//функция вывода результата

{

int N=widget->numb();

int sum;//суммарные веса

ui->listWidget\_4->addItem("Результат");

for(int j=0;j<N;j++)

{

QString list\_mass;

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

{

if(widget->Array[j][i]==inf)

{

list\_mass.push\_back("x");

}

else

{ list\_mass.push\_back(QString::number(widget->Array[j][i]));}

list\_mass.push\_back(" ");

list2->addItem( "Путь из " + QString::number(j+1) + " в " +QString::number( i+1) + " : ");

if (widget->path[j][i] == -1)

{

list2->addItem( "НЕТ" );

}

else

{

int prev = j;

int next;

sum=0;

while (widget->path[prev][i] != -1)

{

list2->addItem( QString::number(prev + 1) + " ");

next = widget->path[prev][i];

sum+=widget->Array[prev][next];

prev=next;

}

list2->addItem( QString::number(prev + 1) + " ");

list2->addItem("Стоимость: "+ QString::number(sum) + " \n");

}

}

list\_mass.push\_back("\n");

ui->listWidget\_4->addItem(list\_mass);

}

}

void MainWindow:: show\_path(int k, int b)

{

int j = k;

int i = b;

int sum;

if(show\_to\_small){list3->addItem( "Путь из " + QString::number(j+1) + " в " +QString::number( i+1) + " : ");}

else{list2->addItem( "Путь из " + QString::number(j+1) + " в " +QString::number( i+1) + " : ");}

if (widget->path[j][i] == -1)

{

if(show\_to\_small)

{list3->addItem( "НЕТ" );}

else{list2->addItem( "НЕТ" );}

}

else

{

int prev = j;

int next;

sum=0;

while (widget->path[prev][i] != -1)

{

if(show\_to\_small)

{ list3->addItem( QString::number(prev + 1) + " ");}

else{list2->addItem( QString::number(prev + 1) + " ");}

next = widget->path[prev][i];

sum+=widget->Array[prev][next];

prev=next;

}

if(show\_to\_small)

{ list3->addItem( QString::number(prev + 1) + " ");

list3->addItem("Стоимость: "+ QString::number(sum) + " \n");

}

else{list2->addItem( QString::number(prev + 1) + " ");

list2->addItem("Стоимость: "+ QString::number(sum) + " \n");}

}

}

**graphwidget.cpp**

#include "graphwidget.h"

#include "edge.h"

#include "node.h"

#include <math.h>

#include <iostream>

#include <QKeyEvent>

using namespace std;

static const double Pi = 3.14159265358979323846264338327950288419717;

//! [0]

GraphWidget::GraphWidget(QWidget \*parent)

: QGraphicsView(parent), timerId(0)

{

tree\_id=1;

QGraphicsScene \*scene = new QGraphicsScene(parent);

scene->setItemIndexMethod(QGraphicsScene::NoIndex);

scene->setSceneRect(-200, -200, 400, 400);

setScene(scene);

setCacheMode(CacheBackground);

setViewportUpdateMode(BoundingRectViewportUpdate);

setRenderHint(QPainter::Antialiasing);

setTransformationAnchor(AnchorUnderMouse);

// Ввод с файла или создание 1 вершины

ifstream file("Z:\\input.txt");

int choice=0;

file>>choice;

if(choice>=49)

{choice=49;}

add\_tree\_elem(choice);

}

void GraphWidget::itemMoved()

{

if (!timerId)

timerId = startTimer(1000 / 25);

}

void GraphWidget::*keyPressEvent*(QKeyEvent \*event)

{

switch (event->key()) {

case Qt::Key\_Plus:

zoomIn();

break;

case Qt::Key\_Minus:

zoomOut();

break;

case Qt::Key\_Space:

case Qt::Key\_Enter:

shuffle();

break;

default:

QGraphicsView::*keyPressEvent*(event);

}

scene()->update();

}

#ifndef QT\_NO\_WHEELEVENT

void GraphWidget::*wheelEvent*(QWheelEvent \*event)

{

scaleView(pow((double)2, -event->delta() / 240.0));

}

#endif

void GraphWidget::*drawBackground*(QPainter \*painter, const QRectF &rect)

{

Q\_UNUSED(rect);

// Shadow

QRectF sceneRect = this->sceneRect();

QRectF rightShadow(sceneRect.right(), sceneRect.top() + 5, 5, sceneRect.height());

QRectF bottomShadow(sceneRect.left() + 5, sceneRect.bottom(), sceneRect.width(), 5);

if (rightShadow.intersects(rect) || rightShadow.contains(rect))

painter->fillRect(rightShadow, Qt::darkGray);

if (bottomShadow.intersects(rect) || bottomShadow.contains(rect))

painter->fillRect(bottomShadow, Qt::darkGray);

// Fill

QLinearGradient gradient(sceneRect.topLeft(), sceneRect.bottomRight());

gradient.setColorAt(0, Qt::white);

gradient.setColorAt(1, Qt::lightGray);

painter->fillRect(rect.intersected(sceneRect), gradient);

painter->setBrush(Qt::NoBrush);

painter->drawRect(sceneRect);

}

void GraphWidget::scaleView(qreal scaleFactor)

{

qreal factor = transform().scale(scaleFactor, scaleFactor).mapRect(QRectF(0, 0, 1, 1)).width();

if (factor < 0.07 || factor > 100)

return;

scale(scaleFactor, scaleFactor);

}

void GraphWidget::shuffle()

{

foreach (QGraphicsItem \*item, scene()->items()) {

if (qgraphicsitem\_cast<Node \*>(item))

item->setPos(-150 + qrand() % 300, -150 + qrand() % 300);

}

}

void GraphWidget::zoomIn()

{

scaleView(qreal(1.2));

}

void GraphWidget::zoomOut()

{

scaleView(1 / qreal(1.2));

}

void GraphWidget::add\_tree\_elem(int choice){

if(choice<=0)

{ Node \*node = new Node(this,tree\_id);

node->setPos(qrand()%20-10,qrand()%20-10);

scene()->addItem(node);

tree \*tr = new tree(node,tree\_id++);

vect\_tree.push\_back(tr);}

else

{

const double bla=360/choice;

double coor=360/choice;

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

{

Node \*node = new Node(this,tree\_id);

node->setPos(100\*cos(coor\*Pi/180),100\*sin(coor\*Pi/180));

scene()->addItem(node);

tree \*tr = new tree(node,tree\_id++);

vect\_tree.push\_back(tr);

coor+=bla;

}

}

}

void GraphWidget::add\_edge(int id1,int id2,int weigth){

tree \*tr1 = get\_tree(id1);

tree \*tr2 = get\_tree(id2);

if(tr1==NULL || tr2==NULL) return ;

Edge \*edge = new Edge(tr1->elem,tr2->elem,weigth);

corner \*corn = new corner(tr1,tr2,edge,weigth);

scene()->addItem(edge);

vect\_corn.push\_back(corn);

}

GraphWidget::tree\* GraphWidget::get\_tree(int id){

for(int i=0;i<vect\_tree.length();i++){

if (vect\_tree[i]->id==id) return vect\_tree[i];

}

return NULL;

}

void GraphWidget::clear\_scene(){//стереть граф

vect\_corn.clear();

vect\_tree.clear();

scene()->clear();

}

void GraphWidget::all\_clear(){//убрать графическую подсветку

for(int i=0;i<vect\_corn.length();i++){

vect\_corn[i]->elem->red=0;

}

}

void GraphWidget::next\_step(int k, int b){//вывод графической подсветки

all\_clear();

int i = b;

int prev = k;

int next;

while (path[prev][i] != -1)

{

next =path[prev][i];

for(int j=0;j<vect\_corn.size();j++)

{

if(vect\_corn[j]->tr1->id==prev+1&&vect\_corn[j]->tr2->id==next+1)

{

vect\_corn[j]->elem->red=1;

scene()->update();

}

}

prev=next;

}

}

void GraphWidget::build\_matrix()

{

int N=vect\_tree.size();

for(int i=0;i<N;i++)//забиваем матрицу смежности бесконечностями

//а матрицу путей -1

{

for(int j=0;j<N;j++)

{

if(i==j)

{

Array[i][j]=0;

}

else

{Array[i][j]=inf;}

path[i][j]=-1;

}

}

for(int i=0;i<vect\_corn.size();i++)//забиваем матрицу по данным хранящимся в векторах

{

Array[vect\_corn[i]->tr1->id-1][vect\_corn[i]->tr2->id-1]=vect\_corn[i]->weigth;

path[vect\_corn[i]->tr1->id-1][vect\_corn[i]->tr2->id-1]=vect\_corn[i]->tr2->id-1;

}

}

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}