**Пермский национальный исследовательский  
политехнический университет**

**О Т Ч Ё Т**

**по творческой работе №2**

Задача коммивояжера

Дисциплина: информатика

Вариант: 22

Выполнил работу

студент группы РИС-20-1б

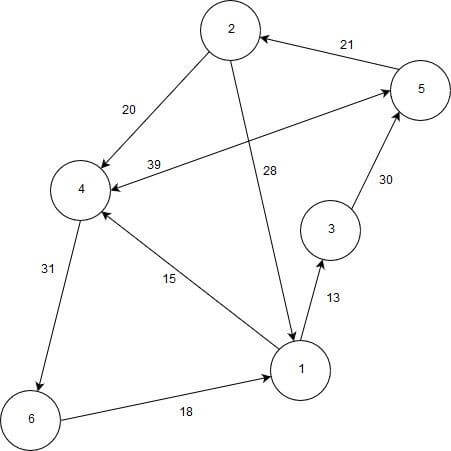
Рябов Никита Андреевич

Проверила

Доцент кафедры ИТАС Полякова О.А.

Пермь, 2021

**Постановка задачи**



**Анализ задачи**

*Какие предстоит выполнить действия.*

Функции загрузки и сохранения графа. По сути просто записывание всех полей объектов в файл csv с помощью стандартных файловых потоков

void MainWindow::**MSave**()

{

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern QVector <QVector<int>> matrix;

QString path = QFileDialog::getSaveFileName(this,

QString::fromUtf8("Сохранить файл"),

QDir::currentPath(),

"Tables (\*.csv);;All files (\*.\*)");

QFile file(path);

if ( !file.*open*(QFile::WriteOnly | QFile::Text) )

{

QMessageBox::critical(this, "Ошибка","не удалось отрыть файл");

return;

}

else

{

QTextStream out(*&file*);

for (int i=0;i<graph1.size();i++)

{

out<<1<<","<<graph1[i].x<<","<<graph1[i].y<<"\n";

}

for (int i=0;i<graph2.size();i++)

{

out<<2<<","<<graph2[i].x1<<","<<graph2[i].x2<<","<<graph2[i].y1<<","

<<graph2[i].y2<<","<<graph2[i].ind\_in<<","<<graph2[i].ind\_out<<","

<<graph2[i].mass<<","<<graph2[i].type<<"\n";

}

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

{

out<<3;

for (int j=0;j<matrix.size();j++) out<<","<<matrix[i][j];

out<<"\n";

}

file.*close*();

}

}

void MainWindow::**MLoad**()

{

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern QVector <QVector<int>> matrix;

graph1.clear();

graph2.clear();

matrix.clear();

QString path = QFileDialog::getOpenFileName(this,

QString::fromUtf8("Открыть файл"),

QDir::currentPath(),

"Tables (\*.csv);;All files (\*.\*)");

QFile file(path);

if ( !file.*open*(QFile::ReadOnly | QFile::Text) )

{

QMessageBox::critical(this, "Ошибка","не удалось отрыть файл");

return;

}

else

{

QTextStream in(*&file*);

while (!in.atEnd())

{

QStringList data = in.readLine().split(",");

if (data[0]=="1")

{

TOP a;

a.x=data[1].toInt();

a.y=data[2].toInt();

graph1.push\_back(a);

}

if (data[0]=="2")

{

EDGE a;

a.x1=data[1].toInt();

a.x2=data[2].toInt();

a.y1=data[3].toInt();

a.y2=data[4].toInt();

a.ind\_in=data[5].toInt();

a.ind\_out=data[6].toInt();

a.mass=data[7].toInt();

a.type=data[8].toInt();

graph2.push\_back(a);

}

if (data[0]=="3")

{

QVector <int> matrix\_line;

for (int i=1;i<data.size();i++) matrix\_line.push\_back(data[i].toInt());

matrix.push\_back(matrix\_line);

}

}

file.*close*();

}

}

Пролистывание истории изменения матрицы. Функции лишь изменяют переменную, отвечающую за индекс показываемого вектора, остальное – забота отрисовщика.

void MainWindow::**FR**()

{

extern QVector <QVector<QVector <int>>> data\_history;

extern int drawing\_index;

if (drawing\_index!=data\_history.size()-1) drawing\_index++;

openGLW->repaint();

}

void MainWindow::**FL**()

{

extern QVector <QVector<QVector <int>>> data\_history;

extern int drawing\_index;

if (drawing\_index!=0) drawing\_index--;

openGLW->repaint();

}

Удаление результатов анимации и приведение программы в исходный вид, активация виджетов.

void MainWindow::**Fdel**()

{

//cleaning

extern QVector <QVector<QVector<int>>> data\_history;

extern int drawing\_index;

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern int len\_del1;

extern int len\_del2;

graph1.erase(graph1.end()-len\_del1,graph1.end());

graph2.erase(graph2.end()-len\_del2,graph2.end());

data\_history.clear();

drawing\_index=-1;

//enabled

btn\_calc->setEnabled(true);

btn\_del->setEnabled(false);

for (int i=0;i<6;i++) G->button(i)->setEnabled(true);

ent1->setEnabled(true);

ent2->setEnabled(true);

btn\_R->setEnabled(false);

btn\_L->setEnabled(false);

ui->menu\_save->setEnabled(true);

ui->menu\_load->setEnabled(true);

openGLW->repaint();

update();

}

В данной программе нажатие на кнопку не означает действие. Нажав на кнопку, пользователь говорит программе, что он планирует сделать с opengl виджетом. За смену таких рабочих состояний и отвечает данная функция.

void MainWindow::**Fswitch**()

{

QPushButton\* button = qobject\_cast<QPushButton\*>(*sender()*);

int buttonID = G->id(*button*);

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

{

G->button(i)->setStyleSheet("QPushButton { background-color: rgb(240,240,240); }\n");

}

if (buttonID!=5) G->button(buttonID)->setStyleSheet("QPushButton { background-color: rgb(200,200,200); }\n");

extern QString status;

extern QString text\_for\_arrows;

extern QString statusBook[7];

if (buttonID==1) text\_for\_arrows=ent1->text();

if (buttonID==2) text\_for\_arrows=ent2->text();

status=statusBook[buttonID];

}

Метод ветвей и границ. В первой части функции реализуется решение задачи коммивояжера, результатом которого является массив целых чисел. Затем прогружается анимация результата. Важный момент, анимация записывается в те же вектора, что и основной граф, нарисованный пользователем. Поэтому данная функция делает неактивными большинство виджетов, иначе, пользователь сможет редактировать результат анимации.

void MainWindow::**F**()

{

//calculation

extern QVector <QVector<QVector <int>>> data\_history;

extern QVector <QVector<int>> matrix;

QVector <int> result\_arr\_1;

QVector <int> result\_arr\_2;

int len=matrix.size();

int sum=0;

bool flag=true;

int M=1000;

QVector <QVector<int>> data=matrix;

for (int i=0;i<len;i++) for (int k=0;k<len;k++) if (i==k || data[i][k]==0) data[i][k]=M;

data\_history.push\_back(data);

while (flag)

{

QVector <int> row(len);

row.fill(M);

QVector <int> column(len);

column.fill(M);

for (int i=0;i<len;i++) for (int k=0;k<len;k++) row[i]=qMin(row[i],data[i][k]);

for (int i=0;i<len;i++) for (int k=0;k<len;k++) if (row[i]!=M && data[i][k]!=M) data[i][k]-=row[i];

data\_history.push\_back(data);

for (int i=0;i<len;i++) for (int k=0;k<len;k++) column[k]=qMin(column[k],data[i][k]);

for (int i=0;i<len;i++) for (int k=0;k<len;k++) if (column[k]!=M && data[i][k]!=M) data[i][k]-=column[k];

data\_history.push\_back(data);

int max\_mark\_x;

int max\_mark\_y;

int max\_mark=-1;

for (int i=0;i<len;i++) for (int k=0;k<len;k++) if (data[i][k]==0)

{

int min1=M;

int min2=M;

for (int j=0;j<len;j++) if (j!=k) min1=qMin(min1,data[i][j]);

for (int j=0;j<len;j++) if (j!=i) min2=qMin(min2,data[j][k]);

if (min1+min2>max\_mark)

{

max\_mark=min1+min2;

max\_mark\_x=i;

max\_mark\_y=k;

}

}

sum+=matrix[max\_mark\_x][max\_mark\_y];

data[max\_mark\_y][max\_mark\_x]=M;

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

{

data[max\_mark\_x][i]=M;

data[i][max\_mark\_y]=M;

}

data\_history.push\_back(data);

result\_arr\_1.push\_back(max\_mark\_x);

result\_arr\_2.push\_back(max\_mark\_y);

flag=false;

for (int i=0;i<len;i++) for (int k=0;k<len;k++) if (data[i][k]!=M) flag=true;

}

QVector <int> result;

result.push\_back(result\_arr\_1[0]);

result.push\_back(result\_arr\_2[0]);

int x=result\_arr\_2[0];

result\_arr\_1.erase(result\_arr\_1.begin());

result\_arr\_2.erase(result\_arr\_2.begin());

while (result\_arr\_1.size()!=0)

{

for (int i=0;i<result\_arr\_1.size();i++) if (result\_arr\_1[i]==x)

{

result.push\_back(result\_arr\_2[i]);

x=result\_arr\_2[i];

result\_arr\_1.erase(result\_arr\_1.begin()+i);

result\_arr\_2.erase(result\_arr\_2.begin()+i);

break;

}

}

if (result[0]!=result[result.size()-1])

{

QMessageBox::warning(this, "Внимание","невозможно вычислить");

return;

}

//animation

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern int len\_del1; //кол-во добавленных узлов

extern int len\_del2; //кол-во добавленных ребер

len\_del1=result.size();

len\_del2=result.size()-1;

int len\_g2=graph2.size(); //кол-во ребер

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

{

TOP a;

a.x=graph1[result[i]].x;

a.y=graph1[result[i]].y;

a.id=result[i];

a.Vmark=true;

graph1.push\_back(a);

}

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

{

EDGE a;

a.ind\_in=i; //другая система индексов

a.ind\_out=i+1;//нежели у основного графа

a.mass=matrix[result[i]][result[i+1]];

a.type=(matrix[result[i]][result[i+1]]==matrix[result[i+1]][result[i]])?2:1;

a.x1=graph1[result[i]].x;

a.x2=graph1[result[i+1]].x;

a.y1=graph1[result[i]].y;

a.y2=graph1[result[i+1]].y;

graph2.push\_back(a);

}

for (int i=0;i<100;i++)

{

for (int j=len;j<graph1.size();j++)//move tops

{

int ind=j-len;

graph1[j].x+=(60+ind\*180-graph1[j].x)/(100-i);

graph1[j].y+=(60-graph1[j].y)/(100-i);

}

for (int j=len\_g2;j<graph2.size();j++)//move edges

{

int ind1=graph2[j].ind\_in;

int ind2=graph2[j].ind\_out;

graph2[j].x1+=(60+ind1\*180-graph2[j].x1)/(100-i);

graph2[j].y1+=(60-graph2[j].y1)/(100-i);

graph2[j].x2+=(60+ind2\*180-graph2[j].x2)/(100-i);

graph2[j].y2+=(60-graph2[j].y2)/(100-i);

}

Sleep(20);

openGLW->repaint();

}

//settings

extern int drawing\_index;

drawing\_index=0;

btn\_del->setEnabled(true);

for (int i=0;i<6;i++) G->button(i)->setEnabled(false);

btn\_calc->setEnabled(false);

ent1->setEnabled(false);

ent2->setEnabled(false);

btn\_R->setEnabled(true);

btn\_L->setEnabled(true);

ui->menu\_save->setEnabled(false);

ui->menu\_load->setEnabled(false);

openGLW->repaint();

}

Смещение графа в стороны. На случай, если все не влезло в кадр.

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

{

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

int dx=0;

int dy=0;

if (event->key()==Qt::Key\_A) dx=10;

else if (event->key()==Qt::Key\_D) dx=-10;

else if (event->key()==Qt::Key\_W) dy=10;

else if (event->key()==Qt::Key\_S) dy=-10;

for (int i=0;i<graph1.size();i++)

{

graph1[i].x+=dx;

graph1[i].y+=dy;

}

for (int i=0;i<graph2.size();i++)

{

graph2[i].x1+=dx;

graph2[i].y1+=dy;

graph2[i].x2+=dx;

graph2[i].y2+=dy;

}

openGLW->repaint();

}

В зависимости от значения переменной status эта функция выполняет соответствующее действие на opengl виджете.

void myGLWidget::***mousePressEvent***(QMouseEvent \*event)

{

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern QVector <QVector<int>> matrix;

int len=graph1.size();

extern int mem\_x;

extern int mem\_y;

extern int mem\_num;

extern QString status;

extern QString text\_for\_arrows;

int m=text\_for\_arrows.toInt();

if (status=="top")

{

TOP a;

a.x=event->x();

a.y=event->y();

graph1.push\_back(a);

len++;

//matrix

QVector <int> row(len);

row.fill(0);

matrix.append(row);

for (int i=0;i<len-1;i++) matrix[i].append(0);

}

else if (status=="edge1.1" || status=="edge2.1")

{

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

{

int x1=event->x();

int y1=event->y();

int x2=graph1[i].x;

int y2=graph1[i].y;

if ((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2)<1600)

{

mem\_x=x2;

mem\_y=y2;

mem\_num=i;

status=(status=="edge1.1")? "edge1.2":"edge2.2";

break;

}

}

}

else if (status=="edge1.2" || status=="edge2.2")

{

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

{

int x1=event->x();

int y1=event->y();

int x2=graph1[i].x;

int y2=graph1[i].y;

if ((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2)<1600)

{

if (status=="edge1.2")

{

EDGE a;

a.x1=mem\_x;

a.y1=mem\_y;

a.x2=x2;

a.y2=y2;

a.ind\_in=mem\_num;

a.ind\_out=i;

a.type=1;

a.mass=m;

graph2.push\_back(a);

//matrix

matrix[mem\_num][i]=m;

}

else

{

EDGE a;

a.x1=mem\_x;

a.y1=mem\_y;

a.x2=x2;

a.y2=y2;

a.ind\_in=mem\_num;

a.ind\_out=i;

a.type=2;

a.mass=m;

graph2.push\_back(a);

//matrix

matrix[mem\_num][i]=m;

matrix[i][mem\_num]=m;

}

status=(status=="edge1.2")? "edge1.1":"edge2.1";

break;

}

}

}

else if (status=="move")

{

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

{

int x1=event->x();

int y1=event->y();

int x2=graph1[i].x;

int y2=graph1[i].y;

if ((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2)<1600)

{

graph1[i].mark=true;

break;

}

}

}

else if (status=="del")

{

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

{

int x1=event->x();

int y1=event->y();

int x2=graph1[i].x;

int y2=graph1[i].y;

if ((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2)<1600)

{

QVector<EDGE> graph2\_new;

for (int j=0;j<graph2.size();j++)

{

if (!(graph2[j].x1==graph1[i].x && graph2[j].y1==graph1[i].y)

&& !(graph2[j].x2==graph1[i].x && graph2[j].y2==graph1[i].y))

{

graph2\_new.push\_back(graph2[j]);

}

}

graph2=graph2\_new;

graph1.erase(graph1.begin()+i);

len--;

//matrix

for (int j=0;j<len;j++) matrix[j].erase(matrix[j].begin()+i);

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

//correct edge's indexes

for (int j=0;j<graph2.size();j++)

{

if (graph2[j].ind\_in>i) graph2[j].ind\_in--;

if (graph2[j].ind\_out>i) graph2[j].ind\_out--;

}

break;

}

}

}

update();

}

В случае, если пользователь нажал клавишу мыши в режиме «движения», то узел графа, оказавшийся под курсором, помечается меткой. Данная же функция снимает все метки при отпускании клавиши.

void myGLWidget::***mouseReleaseEvent***(QMouseEvent \*event)

{

extern QVector<TOP> graph1;

extern QString status;

if (status=="move") for (int i=0;i<graph1.size();i++) graph1[i].mark=false;

update();

}

Двигает все узлы (и связанные с ними ребра), которые помечены меткой.

void myGLWidget::***mouseMoveEvent***(QMouseEvent \*event)

{

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern QString status;

if (status=="move")

{

for (int i=0;i<graph1.size();i++)

{

if (graph1[i].mark)

{

for (int j=0;j<graph2.size();j++)

{

if (graph2[j].x1==graph1[i].x && graph2[j].y1==graph1[i].y)

{

graph2[j].x1=event->x();

graph2[j].y1=event->y();

}

if (graph2[j].x2==graph1[i].x && graph2[j].y2==graph1[i].y)

{

graph2[j].x2=event->x();

graph2[j].y2=event->y();

}

}

graph1[i].x=event->x();

graph1[i].y=event->y();

break;

}

}

}

update();

}

Рисует вершину

void myPainter::**draw\_top**(int x, int y, QString text, QPainter \*painter)

{

int c1=x-20;

int c2=y-15;

int c3=40;

int c4=30;

painter->drawEllipse(QRectF(x-40,y-40,80,80));

painter->drawText(QRect(c1,c2,c3,c4),Qt::AlignCenter,text);

}

Рисует ребро и, если надо, стрелочку

void myPainter::**draw\_edge**(int x1, int y1, int x2, int y2, int type,QPainter \*painter,QString text)

{

double l=sqrt((x2-x1)\*(x2-x1)+(y2-y1)\*(y2-y1));

int xl1=(x2-x1)/l\*40+x1;

int yl1=(y2-y1)/l\*40+y1;

int xl2=x2-(x2-x1)/l\*40;

int yl2=y2-(y2-y1)/l\*40;

painter->drawLine(xl1,yl1,xl2,yl2);

if (type==1)

{

double b;

if (x2==xl2 && yl2<y2) b=pi\*3/2;

else if (y2==yl2 && xl2<x2) b=pi;

else if (x2==xl2 && yl2>y2) b=pi/2;

else if (y2==yl2 && xl2>x2) b=0;

else if (xl2>x2 && yl2<y2) b=2\*pi+atan(1.0\*(y2-yl2)/(x2-xl2));

else if (xl2<x2 && yl2<y2) b=atan(1.0\*(y2-yl2)/(x2-xl2))+pi;

else if (xl2<x2 && yl2>y2) b=atan(1.0\*(y2-yl2)/(x2-xl2))+pi;

else if (xl2>x2 && yl2>y2) b=atan(1.0\*(y2-yl2)/(x2-xl2));

int xs1=cos(b+angle)\*ls+xl2;

int ys1=sin(b+angle)\*ls+yl2;

int xs2=cos(b-angle)\*ls+xl2;

int ys2=sin(b-angle)\*ls+yl2;

painter->drawLine(xl2,yl2,xs1,ys1);

painter->drawLine(xl2,yl2,xs2,ys2);

}

//текст

int c1=(xl2+xl1)/2-20;

int c2=(yl2+yl1)/2-15;

int c3=30;

int c4=25;

QBrush B=QBrush(QColor(255,255,255));

painter->setBrush(B);

painter->drawRect(c1-5,c2-5,c3+10,c4+10);

painter->setBrush(front);

painter->drawText(QRect(c1,c2,c3,c4),Qt::AlignCenter,text);

}

Всемогущая функция, которая рисует все: от узлов и вершин, до анимации и матрицы смежности.

void myPainter::**draw**(QPainter \*painter, QPaintEvent \*event)

{

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

painter->setBrush(front);

painter->setPen(textPen);

painter->setFont(glFont);

textPen.setWidth(4);

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

//draw graph

for (int i=0;i<graph1.size();i++)

{

QString text=(graph1[i].Vmark)? QString::number(graph1[i].id):QString::number(i);

draw\_top(graph1[i].x,

graph1[i].y,

text,

*painter*);

}

for (int i=0;i<graph2.size();i++)

{

draw\_edge(graph2[i].x1,

graph2[i].y1,

graph2[i].x2,

graph2[i].y2,

graph2[i].type,

*painter*,

QString::number(graph2[i].mass));

}

//draw matrix

extern QTableView\* display;

extern QVector <QVector<int>> matrix;

extern QVector <QVector<QVector<int>>> data\_history;

extern int drawing\_index;

QVector<QVector<int>> data;

if (drawing\_index==-1) data=matrix;

else data=data\_history[drawing\_index];

QStringList header;

for (int i=0;i<data.size();i++) header.append(QString::number(i));

QStandardItemModel \*model = new QStandardItemModel;

QStandardItem \*item;

model->setHorizontalHeaderLabels(header);

model->setVerticalHeaderLabels(header);

for (int i=0;i<data.size();i++) for (int j=0;j<data.size();j++)

{

if (data[i][j]==1000) item = new QStandardItem("M");

else item = new QStandardItem(QString::number(data[i][j]));

model->setItem(i, j, *item*);

}

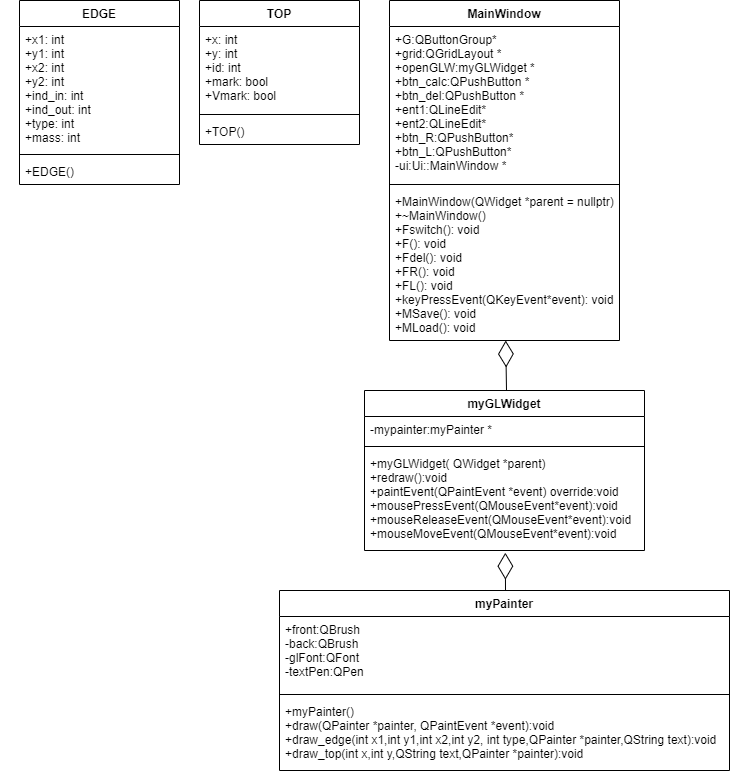
display->*setModel*(*model*);

display->resizeRowsToContents();

display->resizeColumnsToContents();

}

**UML диаграмма**



**Код**

**Edge.h**

#ifndef EDGE\_H

#define EDGE\_H

class **EDGE**

{

public:

**EDGE**();

public:

int x1;

int y1;

int x2;

int y2;

int ind\_in;

int ind\_out;

int type;

int mass;

};

#endif // EDGE\_H

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "myglwidget.h"

#include <QPainter>

#include <QGridLayout>

#include <QPushButton>

#include <QVector>

#include <QButtonGroup>

#include <QLineEdit>

#include <QCheckBox>

QT\_BEGIN\_NAMESPACE

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

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

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

~***MainWindow***();

void **Fswitch**();

void **F**();

void **Fdel**();

void **FR**();

void **FL**();

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

void **MSave**();

void **MLoad**();

public:

QButtonGroup\* G=new QButtonGroup;

QGridLayout \*grid;

myGLWidget \*openGLW;

QPushButton \*btn\_calc;

QPushButton \*btn\_del;

QLineEdit\* ent1;

QLineEdit\* ent2;

QPushButton\* btn\_R;

QPushButton\* btn\_L;

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**Myglwidget.h**

#ifndef MYGLWIDGET\_H

#define MYGLWIDGET\_H

#include "mypainter.h"

#include <QOpenGLWidget>

class **myGLWidget** : public QOpenGLWidget

{

Q\_OBJECT

public:

**myGLWidget**( QWidget \*parent);

public slots:

void **redraw**();

public:

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

void ***mousePressEvent***(QMouseEvent\*event);

void ***mouseReleaseEvent***(QMouseEvent\*event);

void ***mouseMoveEvent***(QMouseEvent\*event);

private:

myPainter \*mypainter;

};

#endif // MYGLWIDGET\_H

**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);

void **draw\_edge**(int x1,int y1,int x2,int y2, int type,QPainter \*painter,QString text);

void **draw\_top**(int x,int y,QString text,QPainter \*painter);

QBrush front;

private:

QBrush back;

QFont glFont;

QPen textPen;

};

#endif //MYPAINTER\_H

**Top.h**

#ifndef TOP\_H

#define TOP\_H

class **TOP**

{

public:

**TOP**();

public:

int x;

int y;

int id;

bool mark;

bool Vmark;

};

#endif // TOP\_H

**Edge.cpp**

#include "edge.h"

EDGE::**EDGE**()

{}

**Main.cpp**

#include "mainwindow.h"

#include "top.h"

#include "edge.h"

#include <QApplication>

#include <QLineEdit>

#include <QTableView>

QString status="none";

int mem\_x;

int mem\_y;

int mem\_num;

int len\_del1;

int len\_del2;

QTableView\* display;

QVector <TOP> graph1;

QVector <EDGE> graph2;

QVector <QVector<int>> matrix;

//QVector <int> row\_history;

//QVector <int> column\_history;

int drawing\_index=-1;

QVector <QVector<QVector <int>>> data\_history;

QString statusBook[]={"top","edge1.1","edge2.1","move","del","none"};

QString text\_for\_arrows;

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

{

QApplication a(*argc*, *argv*);

MainWindow w;

w.setFixedSize(1700,700);

w.show();

return a.exec();

}

**Mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "myglwidget.h"

#include <QPainter>

#include <QGridLayout>

#include <QPushButton>

#include <QLineEdit>

#include <QMessageBox>

#include <top.h>

#include <edge.h>

#include <QTableView>

#include <QKeyEvent>

#include <QFileDialog>

#include <QTextStream>

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

:QMainWindow(*parent*),ui(new Ui::MainWindow)

{

ui->setupUi(this);

//widgets

ent1=new QLineEdit;

ent2=new QLineEdit;

grid=new QGridLayout();

openGLW=new myGLWidget(this);

btn\_calc=new QPushButton;

btn\_del=new QPushButton;

btn\_del->setEnabled(false);

extern QTableView\* display;

display=new QTableView;

btn\_R=new QPushButton;

btn\_L=new QPushButton;

btn\_R->setEnabled(false);

btn\_L->setEnabled(false);

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

{

QPushButton\* btn=new QPushButton;

btn->setFixedSize(200,50);

grid->addWidget(*btn*,i,1,1,2);

connect(btn,&QPushButton::clicked,this,&MainWindow::Fswitch);

G->addButton(*btn*,i);

}

//icons

ui->menu\_save->setIcon(QIcon("C:/qtprojects/ArtProject2/icon2.png"));

ui->menu\_load->setIcon(QIcon("C:/qtprojects/ArtProject2/icon3.png"));

//texts

G->button(0)->setText("◯");

G->button(1)->setText("-->");

G->button(2)->setText("<->");

G->button(3)->setText("<<◯>>");

G->button(4)->setText("✖");

G->button(5)->setText("Отмена");

btn\_calc->setText("Рассчитать");

btn\_del->setText("Удалить");

btn\_R->setText("▶");

btn\_L->setText("◀");

ent1->setText("1");

ent2->setText("1");

//sizes

openGLW->setFixedSize(700,700);

display->setFixedSize(700,550);

G->button(1)->setFixedSize(100,50);

G->button(2)->setFixedSize(100,50);

btn\_calc->setFixedSize(100,50);

btn\_del->setFixedSize(100,50);

ent1->setFixedSize(100,50);

ent2->setFixedSize(100,50);

btn\_R->setFixedSize(50,50);

btn\_L->setFixedSize(50,50);

//locations

grid->addWidget(*openGLW*,0,0,8,1);

grid->addWidget(*ent1*,1,1);

grid->addWidget(*ent2*,2,1);

grid->addWidget(*G->button(*1*)*,1,2);

grid->addWidget(*G->button(*2*)*,2,2);

grid->addWidget(*btn\_calc*,7,1);

grid->addWidget(*btn\_del*,7,2);

grid->addWidget(*btn\_R*,7,4,1,1,Qt::AlignLeft);

grid->addWidget(*btn\_L*,7,3,1,1,Qt::AlignRight);

grid->addWidget(*display*,0,3,8,2,Qt::AlignTop);

connect(btn\_calc,&QPushButton::clicked,this,&MainWindow::F);

connect(btn\_del,&QPushButton::clicked,this,&MainWindow::Fdel);

connect(btn\_R,&QPushButton::clicked,this,&MainWindow::FR);

connect(btn\_L,&QPushButton::clicked,this,&MainWindow::FL);

connect(ui->menu\_save,&QAction::triggered,this,&MainWindow::MSave);

connect(ui->menu\_load,&QAction::triggered,this,&MainWindow::MLoad);

ui->centralwidget->setLayout(*grid*);

}

void MainWindow::**MSave**()

{

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern QVector <QVector<int>> matrix;

QString path = QFileDialog::getSaveFileName(this,

QString::fromUtf8("Сохранить файл"),

QDir::currentPath(),

"Tables (\*.csv);;All files (\*.\*)");

QFile file(path);

if ( !file.*open*(QFile::WriteOnly | QFile::Text) )

{

QMessageBox::critical(this, "Ошибка","не удалось отрыть файл");

return;

}

else

{

QTextStream out(*&file*);

for (int i=0;i<graph1.size();i++)

{

out<<1<<","<<graph1[i].x<<","<<graph1[i].y<<"\n";

}

for (int i=0;i<graph2.size();i++)

{

out<<2<<","<<graph2[i].x1<<","<<graph2[i].x2<<","<<graph2[i].y1<<","

<<graph2[i].y2<<","<<graph2[i].ind\_in<<","<<graph2[i].ind\_out<<","

<<graph2[i].mass<<","<<graph2[i].type<<"\n";

}

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

{

out<<3;

for (int j=0;j<matrix.size();j++) out<<","<<matrix[i][j];

out<<"\n";

}

file.*close*();

}

}

void MainWindow::**MLoad**()

{

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern QVector <QVector<int>> matrix;

graph1.clear();

graph2.clear();

matrix.clear();

QString path = QFileDialog::getOpenFileName(this,

QString::fromUtf8("Открыть файл"),

QDir::currentPath(),

"Tables (\*.csv);;All files (\*.\*)");

QFile file(path);

if ( !file.*open*(QFile::ReadOnly | QFile::Text) )

{

QMessageBox::critical(this, "Ошибка","не удалось отрыть файл");

return;

}

else

{

QTextStream in(*&file*);

while (!in.atEnd())

{

QStringList data = in.readLine().split(",");

if (data[0]=="1")

{

TOP a;

a.x=data[1].toInt();

a.y=data[2].toInt();

graph1.push\_back(a);

}

if (data[0]=="2")

{

EDGE a;

a.x1=data[1].toInt();

a.x2=data[2].toInt();

a.y1=data[3].toInt();

a.y2=data[4].toInt();

a.ind\_in=data[5].toInt();

a.ind\_out=data[6].toInt();

a.mass=data[7].toInt();

a.type=data[8].toInt();

graph2.push\_back(a);

}

if (data[0]=="3")

{

QVector <int> matrix\_line;

for (int i=1;i<data.size();i++) matrix\_line.push\_back(data[i].toInt());

matrix.push\_back(matrix\_line);

}

}

file.*close*();

}

}

void MainWindow::**FR**()

{

extern QVector <QVector<QVector <int>>> data\_history;

extern int drawing\_index;

if (drawing\_index!=data\_history.size()-1) drawing\_index++;

openGLW->repaint();

}

void MainWindow::**FL**()

{

extern QVector <QVector<QVector <int>>> data\_history;

extern int drawing\_index;

if (drawing\_index!=0) drawing\_index--;

openGLW->repaint();

}

void MainWindow::**Fdel**()

{

//cleaning

extern QVector <QVector<QVector<int>>> data\_history;

extern int drawing\_index;

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern int len\_del1;

extern int len\_del2;

graph1.erase(graph1.end()-len\_del1,graph1.end());

graph2.erase(graph2.end()-len\_del2,graph2.end());

data\_history.clear();

drawing\_index=-1;

//enabled

btn\_calc->setEnabled(true);

btn\_del->setEnabled(false);

for (int i=0;i<6;i++) G->button(i)->setEnabled(true);

ent1->setEnabled(true);

ent2->setEnabled(true);

btn\_R->setEnabled(false);

btn\_L->setEnabled(false);

ui->menu\_save->setEnabled(true);

ui->menu\_load->setEnabled(true);

openGLW->repaint();

update();

}

void MainWindow::**Fswitch**()

{

QPushButton\* button = qobject\_cast<QPushButton\*>(*sender()*);

int buttonID = G->id(*button*);

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

{

G->button(i)->setStyleSheet("QPushButton { background-color: rgb(240,240,240); }\n");

}

if (buttonID!=5) G->button(buttonID)->setStyleSheet("QPushButton { background-color: rgb(200,200,200); }\n");

extern QString status;

extern QString text\_for\_arrows;

extern QString statusBook[7];

if (buttonID==1) text\_for\_arrows=ent1->text();

if (buttonID==2) text\_for\_arrows=ent2->text();

status=statusBook[buttonID];

}

void MainWindow::**F**()

{

//calculation

extern QVector <QVector<QVector <int>>> data\_history;

extern QVector <QVector<int>> matrix;

QVector <int> result\_arr\_1;

QVector <int> result\_arr\_2;

int len=matrix.size();

int sum=0;

bool flag=true;

int M=1000;

QVector <QVector<int>> data=matrix;

for (int i=0;i<len;i++) for (int k=0;k<len;k++) if (i==k || data[i][k]==0) data[i][k]=M;

data\_history.push\_back(data);

while (flag)

{

QVector <int> row(len);

row.fill(M);

QVector <int> column(len);

column.fill(M);

for (int i=0;i<len;i++) for (int k=0;k<len;k++) row[i]=qMin(row[i],data[i][k]);

for (int i=0;i<len;i++) for (int k=0;k<len;k++) if (row[i]!=M && data[i][k]!=M) data[i][k]-=row[i];

data\_history.push\_back(data);

for (int i=0;i<len;i++) for (int k=0;k<len;k++) column[k]=qMin(column[k],data[i][k]);

for (int i=0;i<len;i++) for (int k=0;k<len;k++) if (column[k]!=M && data[i][k]!=M) data[i][k]-=column[k];

data\_history.push\_back(data);

int max\_mark\_x;

int max\_mark\_y;

int max\_mark=-1;

for (int i=0;i<len;i++) for (int k=0;k<len;k++) if (data[i][k]==0)

{

int min1=M;

int min2=M;

for (int j=0;j<len;j++) if (j!=k) min1=qMin(min1,data[i][j]);

for (int j=0;j<len;j++) if (j!=i) min2=qMin(min2,data[j][k]);

if (min1+min2>max\_mark)

{

max\_mark=min1+min2;

max\_mark\_x=i;

max\_mark\_y=k;

}

}

sum+=matrix[max\_mark\_x][max\_mark\_y];

data[max\_mark\_y][max\_mark\_x]=M;

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

{

data[max\_mark\_x][i]=M;

data[i][max\_mark\_y]=M;

}

data\_history.push\_back(data);

result\_arr\_1.push\_back(max\_mark\_x);

result\_arr\_2.push\_back(max\_mark\_y);

flag=false;

for (int i=0;i<len;i++) for (int k=0;k<len;k++) if (data[i][k]!=M) flag=true;

}

QVector <int> result;

result.push\_back(result\_arr\_1[0]);

result.push\_back(result\_arr\_2[0]);

int x=result\_arr\_2[0];

result\_arr\_1.erase(result\_arr\_1.begin());

result\_arr\_2.erase(result\_arr\_2.begin());

while (result\_arr\_1.size()!=0)

{

for (int i=0;i<result\_arr\_1.size();i++) if (result\_arr\_1[i]==x)

{

result.push\_back(result\_arr\_2[i]);

x=result\_arr\_2[i];

result\_arr\_1.erase(result\_arr\_1.begin()+i);

result\_arr\_2.erase(result\_arr\_2.begin()+i);

break;

}

}

if (result[0]!=result[result.size()-1])

{

QMessageBox::warning(this, "Внимание","невозможно вычислить");

return;

}

//animation

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern int len\_del1; //кол-во добавленных узлов

extern int len\_del2; //кол-во добавленных ребер

len\_del1=result.size();

len\_del2=result.size()-1;

int len\_g2=graph2.size(); //кол-во ребер

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

{

TOP a;

a.x=graph1[result[i]].x;

a.y=graph1[result[i]].y;

a.id=result[i];

a.Vmark=true;

graph1.push\_back(a);

}

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

{

EDGE a;

a.ind\_in=i; //другая система индексов

a.ind\_out=i+1;//нежели у основного графа

a.mass=matrix[result[i]][result[i+1]];

a.type=(matrix[result[i]][result[i+1]]==matrix[result[i+1]][result[i]])?2:1;

a.x1=graph1[result[i]].x;

a.x2=graph1[result[i+1]].x;

a.y1=graph1[result[i]].y;

a.y2=graph1[result[i+1]].y;

graph2.push\_back(a);

}

for (int i=0;i<100;i++)

{

for (int j=len;j<graph1.size();j++)//move tops

{

int ind=j-len;

graph1[j].x+=(60+ind\*180-graph1[j].x)/(100-i);

graph1[j].y+=(60-graph1[j].y)/(100-i);

}

for (int j=len\_g2;j<graph2.size();j++)//move edges

{

int ind1=graph2[j].ind\_in;

int ind2=graph2[j].ind\_out;

graph2[j].x1+=(60+ind1\*180-graph2[j].x1)/(100-i);

graph2[j].y1+=(60-graph2[j].y1)/(100-i);

graph2[j].x2+=(60+ind2\*180-graph2[j].x2)/(100-i);

graph2[j].y2+=(60-graph2[j].y2)/(100-i);

}

Sleep(20);

openGLW->repaint();

}

//settings

extern int drawing\_index;

drawing\_index=0;

btn\_del->setEnabled(true);

for (int i=0;i<6;i++) G->button(i)->setEnabled(false);

btn\_calc->setEnabled(false);

ent1->setEnabled(false);

ent2->setEnabled(false);

btn\_R->setEnabled(true);

btn\_L->setEnabled(true);

ui->menu\_save->setEnabled(false);

ui->menu\_load->setEnabled(false);

openGLW->repaint();

}

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

{

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

int dx=0;

int dy=0;

if (event->key()==Qt::Key\_A) dx=10;

else if (event->key()==Qt::Key\_D) dx=-10;

else if (event->key()==Qt::Key\_W) dy=10;

else if (event->key()==Qt::Key\_S) dy=-10;

for (int i=0;i<graph1.size();i++)

{

graph1[i].x+=dx;

graph1[i].y+=dy;

}

for (int i=0;i<graph2.size();i++)

{

graph2[i].x1+=dx;

graph2[i].y1+=dy;

graph2[i].x2+=dx;

graph2[i].y2+=dy;

}

openGLW->repaint();

}

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

{

delete ui;

}

**Myglwidget.cpp**

#include "myglwidget.h"

#include "mypainter.h"

#include <QPainter>

#include <top.h>

#include <edge.h>

#include <QMouseEvent>

#include <QMessageBox>

myGLWidget::**myGLWidget**(QWidget \*parent)

:QOpenGLWidget(*parent*)

{

mypainter=new myPainter;

}

void myGLWidget::**redraw**()

{

this->update();

}

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

{

QPainter painter;

painter.begin(this);

mypainter->draw(*&painter*,*event*);

painter.end();

}

void myGLWidget::***mousePressEvent***(QMouseEvent \*event)

{

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern QVector <QVector<int>> matrix;

int len=graph1.size();

extern int mem\_x;

extern int mem\_y;

extern int mem\_num;

extern QString status;

extern QString text\_for\_arrows;

int m=text\_for\_arrows.toInt();

if (status=="top")

{

TOP a;

a.x=event->x();

a.y=event->y();

graph1.push\_back(a);

len++;

//matrix

QVector <int> row(len);

row.fill(0);

matrix.append(row);

for (int i=0;i<len-1;i++) matrix[i].append(0);

}

else if (status=="edge1.1" || status=="edge2.1")

{

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

{

int x1=event->x();

int y1=event->y();

int x2=graph1[i].x;

int y2=graph1[i].y;

if ((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2)<1600)

{

mem\_x=x2;

mem\_y=y2;

mem\_num=i;

status=(status=="edge1.1")? "edge1.2":"edge2.2";

break;

}

}

}

else if (status=="edge1.2" || status=="edge2.2")

{

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

{

int x1=event->x();

int y1=event->y();

int x2=graph1[i].x;

int y2=graph1[i].y;

if ((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2)<1600)

{

if (status=="edge1.2")

{

EDGE a;

a.x1=mem\_x;

a.y1=mem\_y;

a.x2=x2;

a.y2=y2;

a.ind\_in=mem\_num;

a.ind\_out=i;

a.type=1;

a.mass=m;

graph2.push\_back(a);

//matrix

matrix[mem\_num][i]=m;

}

else

{

EDGE a;

a.x1=mem\_x;

a.y1=mem\_y;

a.x2=x2;

a.y2=y2;

a.ind\_in=mem\_num;

a.ind\_out=i;

a.type=2;

a.mass=m;

graph2.push\_back(a);

//matrix

matrix[mem\_num][i]=m;

matrix[i][mem\_num]=m;

}

status=(status=="edge1.2")? "edge1.1":"edge2.1";

break;

}

}

}

else if (status=="move")

{

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

{

int x1=event->x();

int y1=event->y();

int x2=graph1[i].x;

int y2=graph1[i].y;

if ((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2)<1600)

{

graph1[i].mark=true;

break;

}

}

}

else if (status=="del")

{

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

{

int x1=event->x();

int y1=event->y();

int x2=graph1[i].x;

int y2=graph1[i].y;

if ((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2)<1600)

{

QVector<EDGE> graph2\_new;

for (int j=0;j<graph2.size();j++)

{

if (!(graph2[j].x1==graph1[i].x && graph2[j].y1==graph1[i].y)

&& !(graph2[j].x2==graph1[i].x && graph2[j].y2==graph1[i].y))

{

graph2\_new.push\_back(graph2[j]);

}

}

graph2=graph2\_new;

graph1.erase(graph1.begin()+i);

len--;

//matrix

for (int j=0;j<len;j++) matrix[j].erase(matrix[j].begin()+i);

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

//correct edge's indexes

for (int j=0;j<graph2.size();j++)

{

if (graph2[j].ind\_in>i) graph2[j].ind\_in--;

if (graph2[j].ind\_out>i) graph2[j].ind\_out--;

}

break;

}

}

}

update();

}

void myGLWidget::***mouseReleaseEvent***(QMouseEvent \*event)

{

extern QVector<TOP> graph1;

extern QString status;

if (status=="move") for (int i=0;i<graph1.size();i++) graph1[i].mark=false;

update();

}

void myGLWidget::***mouseMoveEvent***(QMouseEvent \*event)

{

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

extern QString status;

if (status=="move")

{

for (int i=0;i<graph1.size();i++)

{

if (graph1[i].mark)

{

for (int j=0;j<graph2.size();j++)

{

if (graph2[j].x1==graph1[i].x && graph2[j].y1==graph1[i].y)

{

graph2[j].x1=event->x();

graph2[j].y1=event->y();

}

if (graph2[j].x2==graph1[i].x && graph2[j].y2==graph1[i].y)

{

graph2[j].x2=event->x();

graph2[j].y2=event->y();

}

}

graph1[i].x=event->x();

graph1[i].y=event->y();

break;

}

}

}

update();

}

**Mypainter.cpp**

#include "mypainter.h"

#include <QPaintEvent>

#include <QPainter>

#include <QWidget>

#include <QtCore/qmath.h>

#include "top.h"

#include "edge.h"

#include <QTableView>

#include "QStandardItemModel"

#include "QStandardItem"

double pi=3.1415;

double angle=pi/8;//угол раствора стрелки

int ls=30; //длина кончика стрелки

myPainter::**myPainter**()

{

front=QBrush(QColor(127,255,212));

back=QBrush(QColor(200,200,200));

textPen=QPen(QColor(33,33,33));

glFont.setPixelSize(30);

}

void myPainter::**draw\_top**(int x, int y, QString text, QPainter \*painter)

{

int c1=x-20;

int c2=y-15;

int c3=40;

int c4=30;

painter->drawEllipse(QRectF(x-40,y-40,80,80));

painter->drawText(QRect(c1,c2,c3,c4),Qt::AlignCenter,text);

}

void myPainter::**draw\_edge**(int x1, int y1, int x2, int y2, int type,QPainter \*painter,QString text)

{

double l=sqrt((x2-x1)\*(x2-x1)+(y2-y1)\*(y2-y1));

int xl1=(x2-x1)/l\*40+x1;

int yl1=(y2-y1)/l\*40+y1;

int xl2=x2-(x2-x1)/l\*40;

int yl2=y2-(y2-y1)/l\*40;

painter->drawLine(xl1,yl1,xl2,yl2);

if (type==1)

{

double b;

if (x2==xl2 && yl2<y2) b=pi\*3/2;

else if (y2==yl2 && xl2<x2) b=pi;

else if (x2==xl2 && yl2>y2) b=pi/2;

else if (y2==yl2 && xl2>x2) b=0;

else if (xl2>x2 && yl2<y2) b=2\*pi+atan(1.0\*(y2-yl2)/(x2-xl2));

else if (xl2<x2 && yl2<y2) b=atan(1.0\*(y2-yl2)/(x2-xl2))+pi;

else if (xl2<x2 && yl2>y2) b=atan(1.0\*(y2-yl2)/(x2-xl2))+pi;

else if (xl2>x2 && yl2>y2) b=atan(1.0\*(y2-yl2)/(x2-xl2));

int xs1=cos(b+angle)\*ls+xl2;

int ys1=sin(b+angle)\*ls+yl2;

int xs2=cos(b-angle)\*ls+xl2;

int ys2=sin(b-angle)\*ls+yl2;

painter->drawLine(xl2,yl2,xs1,ys1);

painter->drawLine(xl2,yl2,xs2,ys2);

}

//текст

int c1=(xl2+xl1)/2-20;

int c2=(yl2+yl1)/2-15;

int c3=30;

int c4=25;

QBrush B=QBrush(QColor(255,255,255));

painter->setBrush(B);

painter->drawRect(c1-5,c2-5,c3+10,c4+10);

painter->setBrush(front);

painter->drawText(QRect(c1,c2,c3,c4),Qt::AlignCenter,text);

}

void myPainter::**draw**(QPainter \*painter, QPaintEvent \*event)

{

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

painter->setBrush(front);

painter->setPen(textPen);

painter->setFont(glFont);

textPen.setWidth(4);

extern QVector<TOP> graph1;

extern QVector<EDGE> graph2;

//draw graph

for (int i=0;i<graph1.size();i++)

{

QString text=(graph1[i].Vmark)? QString::number(graph1[i].id):QString::number(i);

draw\_top(graph1[i].x,

graph1[i].y,

text,

*painter*);

}

for (int i=0;i<graph2.size();i++)

{

draw\_edge(graph2[i].x1,

graph2[i].y1,

graph2[i].x2,

graph2[i].y2,

graph2[i].type,

*painter*,

QString::number(graph2[i].mass));

}

//draw matrix

extern QTableView\* display;

extern QVector <QVector<int>> matrix;

extern QVector <QVector<QVector<int>>> data\_history;

extern int drawing\_index;

QVector<QVector<int>> data;

if (drawing\_index==-1) data=matrix;

else data=data\_history[drawing\_index];

QStringList header;

for (int i=0;i<data.size();i++) header.append(QString::number(i));

QStandardItemModel \*model = new QStandardItemModel;

QStandardItem \*item;

model->setHorizontalHeaderLabels(header);

model->setVerticalHeaderLabels(header);

for (int i=0;i<data.size();i++) for (int j=0;j<data.size();j++)

{

if (data[i][j]==1000) item = new QStandardItem("M");

else item = new QStandardItem(QString::number(data[i][j]));

model->setItem(i, j, *item*);

}

display->*setModel*(*model*);

display->resizeRowsToContents();

display->resizeColumnsToContents();

}

**Top.cpp**

#include "top.h"

TOP::**TOP**()

{

mark=false;

Vmark=false;

}

**Скриншот**

