**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: Нахождение всех гамильтоновых циклов в графе

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Виноградов К.А. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2018

**Цель работы**

Разработать алгоритм нахождения всех гамильтоновых циклов в графе с помощью алгоритма с возвратом, используя возможности Qt и языка C++.

**Теория**

Очевидный алгоритм, который мы можем применить, это «полный перебор всех возможностей»: генерируем все n! различных последовательностей вершин и для каждой из них проверяем, определяет ли она гамильтонов путь. Такие действия требуют по меньшей мере n!n шагов, но функция от n подобного вида растет быстрее, чем произвольный многочлен, и даже быстрее, чем произвольная экспоненциальная функция вида аn, a > 1, ибо



Опишем теперь общий метод, позволяющий значительно сократить число шагов в алгоритмах типа полного перебора всех возможностей. Чтобы применить этот метод, искомое решение должно иметь вид последовательности [x1, . . . , xn]. Основная идея метода состоит в том, что мы строим наше решение

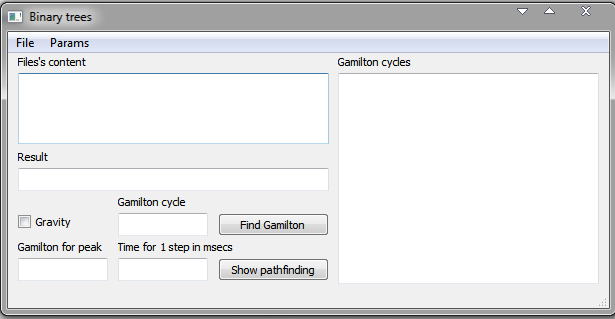
последовательно, начиная с пустой последовательности ε (длины 0). Вообще, имея данное частичное решение [x1, . . . , xi], мы стараемся найти такое допустимое значение xi+1, относительно которого мы не можем сразу заключить, что [x1, . . . , хi, xi+1] можно расширить до некоторого решения (либо [x1, . . . , xi+1] уже является решением). Если такое предполагаемое, но еще не использованное значение xi+1 существует, то мы добавляем его к нашему части ному решению и продолжаем процесс для последовательности [x1, . . . , xi, xi+1]. Если его не существует, то мы возвращаемся к нашему частичному решению [x1, . . . , xi-1] и продолжаем наш процесс, отыскивая новое, еще не использованное допустимое значение xi — отсюда название «алгоритм с возвратом» (англ.: backtracking).

**Ход работы**

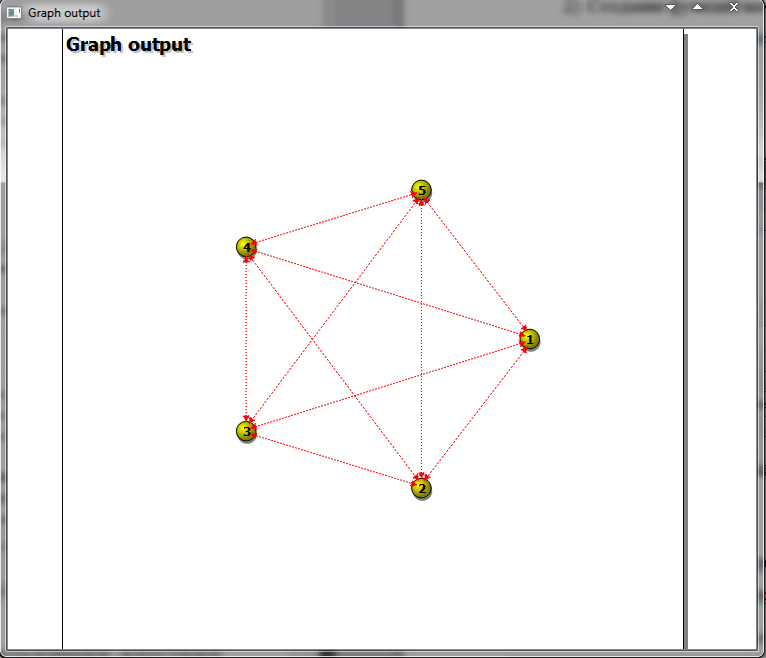
1. Создание структуры данных
2. Создание функций нахождения гамильтоновых циклов
3. Разработка интерфейса программы

**Интерфейс**

а) основной



б) сцена



**Тестирование**

Таблица 1. Результаты тестирования программы

|  |  |
| --- | --- |
| Таблица инцидентности | Гамильтоновы пути |
| 1, 3  2, 1  3, 2 | 1) |1|3|2|  2) |2|1|3|  3) |3|2|1| |
| 1,2,4  2,1,3,5  3,2,4,5  4,1,3,5  5,2,3,4 | 1) |1|2|3|5|4| 2) |1|2|5|3|4|  3) |1|4|3|5|2| 4) |1|4|5|3|2|  5) |2|1|4|3|5| 6) |2|1|4|5|3|  7) |2|3|5|4|1| 8) |2|5|3|4|1|  9) |3|2|1|4|5| 10) |3|4|1|2|5|  11) |3|5|2|1|4| 12) |3|5|4|1|2|  13) |4|1|2|3|5| 14) |4|1|2|5|3|  15) |4|3|5|2|1| 16) |4|5|3|2|1|  17) |5|2|1|4|3| 18) |5|3|2|1|4|  19) |5|3|4|1|2| 20) |5|4|1|2|3| |

**Вывод.**

В ходе данной работы был разработан алгоритм с возвратом, позволяющий найти все гамильтоновы циклы в графе.

**ПРИЛОЖЕНИЕ**

**Код программы**

**struct.h**

#ifndef BINTREE\_H

#define BINTREE\_H

#include <QChar>

#include "node.h"

typedef QString elem;

typedef struct INCTR

{

struct INCTR\* prev=NULL;

elem name;

int turn=0;

int deg=0;

int intdeg=0;

int outdeg=0;

struct INCTR\* next=NULL;

Node\* node;

}INCTR;

INCTR \***delete\_tree**(INCTR\* tree, int size);

#endif // BINTREE\_H

**struct.cpp**

#include "struct.h"

INCTR\* **delete\_tree**(INCTR\* tree, int size)

{

if(!tree)

return tree;

delete [] tree;

return tree;

}

**grapwidget.h**

#ifndef GRAPHWIDGET\_H

#define GRAPHWIDGET\_H

#include <QGraphicsView>

#include <QTimer>

#include "struct.h"

#include "node.h"

class Node;

//! [0]

class GraphWidget : public QGraphicsView

{

Q\_OBJECT

public:

GraphWidget(QWidget \*parent = 0, INCTR \*tree = nullptr, bool \*gr = nullptr, int size = 0, int mseconds = 0, int \*gampath = nullptr);

QTimer\* timer;

void **itemMoved**();

int **Build**(QWidget \*parent = 0, INCTR\* tree=nullptr, qreal x=0, qreal y=0, QGraphicsScene \*scene = nullptr, bool\* gr = nullptr, int size = 0, int \*gampath = nullptr);

int **Build**(QWidget \*parent = 0, INCTR\* tree=nullptr, qreal x=0, qreal y=0, QGraphicsScene \*scene = nullptr, bool\* gr = nullptr, int size = 0, int mseconds = 0, int \*fullgampath = nullptr);

public slots:

/\* void shuffle();

void zoomIn();

void zoomOut(); \*/

protected:

void ***timerEvent***(QTimerEvent \*event) override;

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

void **scaleView**(qreal scaleFactor);

private:

int timerId;

Node \*centerNode;

private slots:

void **slotAlarmTimer**();

};

//! [0]

#endif // GRAPHWIDGET\_H

**grapwidget.cpp**

#include "graphwidget.h"

#include "edge.h"

#include "node.h"

#include "struct.h"

#include <math.h>

#include <QKeyEvent>

//! [0]

static const double Pi = 3.14159265358979323846264338327950288419717;

int i=0;

int Xd=1500;

int\* this\_gampath= new int;

INCTR\* this\_tree= new INCTR;

int GraphWidget::**Build**(QWidget\* parent, INCTR \*tree, qreal x, qreal y, QGraphicsScene \*scene, bool\* gr, int size, int \*gampath)

{

Q\_UNUSED(parent);

Q\_UNUSED(x);

Q\_UNUSED(y);

int R=100\*log(size);

qreal delta1=R, delta2=0;

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

{

tree[j].node = new Node(this, gr);

//if(!scene)

scene->addItem(tree[j].node);

tree[j].node->setPos(delta1, delta2);

tree[j].node->text\_orig=tree[j].name;//fix\_needed

tree[j].node->text=tree[j].name;

delta1=R\*cos((j+1)\*(360/(size))\*(Pi/180));

delta2=R\*sin((j+1)\*(360/(size))\*(Pi/180));

}

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

{

if(tree[j].next)

{

INCTR\* temp=tree[j].next;

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

{

if(temp->name==tree[k].name)

{

temp->node=tree[k].node;

if(tree[j].node->text\_orig.size()<temp->node->text\_orig)

temp->node->text\_orig=tree[j].node->text\_orig;

}

}

while(temp->next)

{

temp=temp->next;

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

{

if(temp->name==tree[k].name)

{

temp->node=tree[k].node;

if(tree[j].node->text\_orig.size()<temp->node->text\_orig)

temp->node->text\_orig=tree[j].node->text\_orig;

}

}

}

}

}

if(gampath==nullptr)

{

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

{

if(tree[j].next)

{

INCTR\* temp=tree[j].next;

scene->addItem((new Edge(temp->node, tree[j].node, true, temp->name.size(), tree[j].name.size(), false)));

while(temp->next)

{

temp=temp->next;

scene->addItem((new Edge(temp->node, tree[j].node, true, temp->name.size(), tree[j].name.size(), false)));

}

}

}

}

else

{

int add=5;

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

{

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

{

if(tree[j].next)

{

INCTR\* temp=tree[j].next;

{

if((tree[j].turn==gampath[i] && temp->turn==gampath[i+1]) /\*|| (tree[j].turn==gampath[i+1] && temp->turn==gampath[i])\*/

/\*|| (tree[j].turn==gampath[0] && temp->turn==gampath[size-1])\*/ || (tree[j].turn==gampath[size-1] && temp->turn==gampath[0])){

// if((tree[j].turn==gampath[i] && temp->turn==gampath[i+1]) || /\*(tree[j].turn==gampath[i+1] && temp->turn==gampath[i])\*/

// /\*|| (tree[j].turn==gampath[0] && temp->turn==gampath[size-1]) ||\*/ (tree[j].turn==gampath[size-1] && temp->turn==gampath[0]))

while(add!=0)

{

scene->addItem((new Edge(temp->node, tree[j].node, true, temp->name.size(), tree[j].name.size(), true)));

add--;

}

add=5;

}

else

scene->addItem((new Edge(temp->node, tree[j].node, true, temp->name.size(), tree[j].name.size(), false)));

while(temp->next)

{

temp=temp->next;

if((tree[j].turn==gampath[i] && temp->turn==gampath[i+1]) /\*|| (tree[j].turn==gampath[i+1] && temp->turn==gampath[i])\*/

/\*|| (tree[j].turn==gampath[0] && temp->turn==gampath[size-1])\*/ || (tree[j].turn==gampath[size-1] && temp->turn==gampath[0])){

while(add!=0)

{

scene->addItem((new Edge(temp->node, tree[j].node, true, temp->name.size(), tree[j].name.size(), true)));

add--;

}

add=5;

}

else

scene->addItem((new Edge(temp->node, tree[j].node, true, temp->name.size(), tree[j].name.size(), false)));

}

}

}

}

}

}

return 0;

}

int GraphWidget::**Build**(QWidget\* parent, INCTR \*tree, qreal x, qreal y, QGraphicsScene \*scene, bool\* gr, int size, int mseconds, int \*fullgampath)

{

Q\_UNUSED(parent);

Q\_UNUSED(x);

Q\_UNUSED(y);

int R=100\*log(size);

qreal delta1=R, delta2=0;

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

{

tree[j].node = new Node(this, gr);

//if(!scene)

scene->addItem(tree[j].node);

tree[j].node->setPos(delta1, delta2);

tree[j].node->text\_orig=tree[j].name;//fix\_needed

tree[j].node->text=tree[j].name;

delta1=R\*cos((j+1)\*360\*Pi/(size\*180));

delta2=R\*sin((j+1)\*360\*Pi/(size\*180));

}

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

{

if(tree[j].next)

{

INCTR\* temp=tree[j].next;

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

{

if(temp->name==tree[k].name)

{

temp->node=tree[k].node;

if(tree[j].node->text\_orig.size()<temp->node->text\_orig)

temp->node->text\_orig=tree[j].node->text\_orig;

}

}

while(temp->next)

{

temp=temp->next;

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

{

if(temp->name==tree[k].name)

{

temp->node=tree[k].node;

if(tree[j].node->text\_orig.size()<temp->node->text\_orig)

temp->node->text\_orig=tree[j].node->text\_orig;

}

}

}

}

}

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

{

if(tree[j].next)

{

INCTR\* temp=tree[j].next;

scene->addItem((new Edge(temp->node, tree[j].node, true, temp->name.size(), tree[j].name.size(), false)));

while(temp->next)

{

temp=temp->next;

scene->addItem((new Edge(temp->node, tree[j].node, true, temp->name.size(), tree[j].name.size(), false)));

}

}

}

this\_gampath=fullgampath;

this\_tree=tree;

timer=new QTimer();

i=0;

connect(timer, SIGNAL(timeout()), this, SLOT(slotAlarmTimer()));

timer->start(mseconds);

i=0;

return 0;

}

void GraphWidget::**slotAlarmTimer**()

{

bool flag=false;

if(this\_gampath[i+1]!=0)

{

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

{

Edge \*line = qgraphicsitem\_cast<Edge \*>(item);

if (line)

{

if(line->source==this\_tree[this\_gampath[i]-1].node && line->dest==this\_tree[this\_gampath[i+1]-1].node && line->gamilt\_clr==true)

{

this->scene()->removeItem(item);

flag=true;

}

}

}

if(flag!=true)

{

this->scene()->addItem((new Edge(this\_tree[this\_gampath[i+1]-1].node, this\_tree[this\_gampath[i]-1].node, true, this\_tree[this\_gampath[i+1]-1].name.size(), this\_tree[this\_gampath[i]-1].name.size(), true)));

flag=false;

}

i++;

}

else

timer->stop();

}

GraphWidget::**GraphWidget**(QWidget \*parent, INCTR \*tree, bool\* gr, int size, int mseconds, int\* gampath)

: QGraphicsView(parent), timerId(0)

{

QGraphicsScene \*scene = new QGraphicsScene(this);

scene->setItemIndexMethod(QGraphicsScene::NoIndex);

scene->setSceneRect(-((300\*log(size)+150)/2), -((300\*log(size)+150)/2), (300\*log(size)+150), (300\*log(size)+150));

setScene(scene);

setCacheMode(CacheBackground);

setViewportUpdateMode(BoundingRectViewportUpdate);

setRenderHint(QPainter::Antialiasing);

setTransformationAnchor(AnchorUnderMouse);

scale(qreal(0.98), qreal(0.98));

setMinimumSize(750, 550);

setWindowTitle(tr("Elastic Nodes"));

//! [0]

//! [1]

if(mseconds==0)

Build(this, &tree[0], 0, 0, scene, gr, size, gampath);

else

Build(this, &tree[0], 0, 0, scene, gr, size, mseconds, gampath);

}

//! [1]

//! [2]

void GraphWidget::**itemMoved**()

{

if (!timerId)

timerId = startTimer(1000 / 25);

}

void GraphWidget::***timerEvent***(QTimerEvent \*event)

{

Q\_UNUSED(event);

QList<Node \*> nodes;

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

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

nodes << node;

}

foreach (Node \*node, nodes)

node->calculateForces();

bool itemsMoved = false;

foreach (Node \*node, nodes) {

if (node->advance())

itemsMoved = true;

}

if (!itemsMoved) {

killTimer(timerId);

timerId = 0;

}

}

//! [4]

//! [6]

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::green);

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

painter->setBrush(Qt::NoBrush); \*/

painter->drawRect(sceneRect);

// Text

QRectF textRect(sceneRect.left() + 4, sceneRect.top() + 4,

sceneRect.width() - 4, sceneRect.height() - 4);

QString message("Graph output");

QFont font = painter->font();

font.setBold(true);

font.setPointSize(14);

painter->setFont(font);

painter->setPen(Qt::lightGray);

painter->drawText(textRect.translated(2, 2), message);

painter->setPen(Qt::black);

painter->drawText(textRect, message);

}

//! [6]

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

private slots:

void **on\_action\_7\_triggered**();

void **on\_action\_5\_triggered**();

void **on\_action\_6\_triggered**();

void **on\_action\_triggered**();

void **on\_action\_2\_triggered**();

void **on\_checkBox\_clicked**();

void **on\_pushButton\_clicked**();

void **on\_pushButton\_2\_clicked**();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QFile>

#include <QFileDialog>

#include <QString>

#include <QTextStream>

#include <QMessageBox>

#include "graphwidget.h"

#include "struct.h"

#include "node.h"

#include <QWidget>

INCTR\* tree = new INCTR;

int peaks\_i;

bool gravity = false;

int\*\* steps = new int\*;

int **GhouilaHouriCheck**(INCTR\* peaks, int size)

{

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

{

if(peaks[i].intdeg+peaks[i].outdeg<size)

return 1;

}

return 0;

}

int **Factor**(int size)

{

if(size==1)

return 1;

size\*=Factor(size-1);

return size;

}

int\*\* **Gamilton**(INCTR\* genparent, INCTR\* parent, int peaks\_i, int step, int fullstep, int circle, int\*\* gampath, int\* fullgampath)

{

bool flag=false;

INCTR\* temp= new INCTR;

temp=parent;

while(temp->next)

{

temp=temp->next;

flag=false;

for(int j=0; j<peaks\_i; j++)

{

if(step!=peaks\_i)

{

if(temp->turn==tree[j].turn /\*&& j!=peaks\_i-1\*/)

{

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

{

if(gampath[circle][k]==tree[j].turn)

{

if(!temp->next)

{

gampath[/\*circle\*/0][peaks\_i+1]=circle;

gampath[/\*circle\*/0][peaks\_i+2]=step-1;

gampath[/\*circle\*/0][peaks\_i+3]=fullstep;

return gampath;

}

else

{

flag=true;

}

}

}

if(flag!=true)

{

flag=false;

gampath[circle][step]=temp->turn;

fullgampath[fullstep]=temp->turn;

//if(step<peaks\_i-1)

step++;

fullstep++;

gampath=Gamilton(genparent, &tree[j], peaks\_i, step, fullstep, circle, gampath, fullgampath);

fullstep=gampath[0][peaks\_i+3];

step=gampath[/\*circle\*/0][peaks\_i+2];

circle=gampath[/\*circle\*/0][peaks\_i+1];

fullgampath[fullstep]=parent->turn;

fullstep++;

gampath[0][peaks\_i+3]=fullstep;

}

}

}

else

{

if(temp->turn==genparent->turn)

{

fullgampath[fullstep]=temp->turn;

fullgampath[fullstep+1]=parent->turn;

circle++;

fullstep+=2;

// if(temp->next)

// {

gampath[circle]=new int[peaks\_i+10];

for(int i=0; i<step-1; i++)

gampath[circle][i]=gampath[circle-1][i];

// }

step-=1;

gampath[circle-1][peaks\_i]=-1;

gampath[/\*circle-1\*/0][peaks\_i+1]=circle;

gampath[/\*circle-1\*/0][peaks\_i+2]=step;

gampath[0][peaks\_i+3]=fullstep;

return gampath;

}

else

{

while(temp->next)

{

temp=temp->next;

if(temp->turn==genparent->turn)

{

fullgampath[fullstep]=temp->turn;

fullgampath[fullstep+1]=parent->turn;

circle++;

fullstep+=2;

// if(temp->next)

// {

gampath[circle]=new int[peaks\_i+10];

for(int i=0; i<step-1; i++)

gampath[circle][i]=gampath[circle-1][i];

// }

step-=1;

gampath[circle-1][peaks\_i]=-1;

gampath[/\*circle-1\*/0][peaks\_i+1]=circle;

gampath[/\*circle-1\*/0][peaks\_i+2]=step;

gampath[0][peaks\_i+3]=fullstep;

return gampath;

}

}

step-=1;

gampath[/\*circle\*/0][peaks\_i+1]=circle;

gampath[/\*circle\*/0][peaks\_i+2]=step;

gampath[/\*circle\*/0][peaks\_i+3]=fullstep;

return gampath;

}

}

}

}

//if(!temp->next)

gampath[/\*circle-1\*/0][peaks\_i+2]=step-1;

gampath[/\*circle\*/0][peaks\_i+3]=fullstep;

//delete temp;

return gampath;

}

int **analyser**(QString str, int k)

{

int peaks=0;

int com=0;

while(str[k]==' ' || str[k]=='\t' || str[k]=='\r' || str[k]=='\n')

k++;

if(str[k]!='\0')

peaks++;

while(str[k]!='\0' /\*&& com<3\*/)

{

if(str[k]==',')

com++;

if(str[k]=='\n')

{

while(str[k+1]==' ' || str[k+1]=='\t' || str[k+1]=='\r')

k++;

if(str[k+1]!='\0' && str[k+1]!='\r' && str[k+1]!='\n')

peaks++;

com=0;

}

k++;

}

if(k!=str.size())

return -1;

return peaks;

}

QString **corrector**(QString str)

{

int k=0, i=0;

QString new\_str;

bool flag=false;

if(str[k]!='\t' && str[k]!=' ' && str[k]!='\n' && str[k]!='\r')

flag=true;

if(flag==true)

{

new\_str[i]=str[k];

i++;

}

for(k=1; k<str.size(); k++)

{

if(str[k-1]=='\n' || str[k-1]==',' || str[k]=='\n' || str[k]==',')

flag=false;

if(str[k]!='\t' && str[k]!=' ' && str[k]!='\n' && str[k]!='\r')

flag=true;

if(flag==true)

{

new\_str[i]=str[k];

i++;

}

}

return new\_str;

}

INCTR\* **Read**(QString str, INCTR\* peaks, int size)

{

int i=0, j=0, k=0;

while(str[k]!='\0')

{

while(str[k]!=',' && str[k]!='\r' && str[k]!='\0')

{

peaks[i].name[j]=str[k];

peaks[i].turn=i+1;

k++;

j++;

}

j=0;

if(str[k]!='\0' && str[k]!='\r')

{

k++;

INCTR\* temp=new INCTR;

peaks[i].next=temp;

peaks[i].outdeg++;

temp->next=NULL;

temp->intdeg++;

temp->prev=&peaks[i];

while(str[k]!=',' && str[k]!='\r' && str[k]!='\0')

{

while(str[k]!=',' && str[k]!='\r' && str[k]!='\0')

{

temp->name[j]=str[k];

j++;

k++;

}

if(str[k]!='\0' && str[k]!='\r')

{

INCTR\* temp2=new INCTR;

peaks[i].outdeg++;

temp->next=temp2;

temp2->next=NULL;

temp2->intdeg++;

temp2->prev=temp;

temp=temp2;

k++;

}

j=0;

}

//delete temp;

}

i++;

k++;

}

for(i=0; i<size; i++)

{

for(j=0; j<size; j++)

{

if(peaks[j].next)

{

INCTR\* temp=new INCTR;

temp=peaks[j].next;

if(peaks[i].name==temp->name)

{

peaks[i].intdeg+=temp->intdeg;

temp->turn=peaks[i].turn;

}

while(temp->next)

{

temp=temp->next;

if(peaks[i].name==temp->name)

{

peaks[i].intdeg+=temp->intdeg;

temp->turn=peaks[i].turn;

}

}

//delete temp;

}

}

peaks[i].deg=peaks[i].intdeg+peaks[i].outdeg;

}

return peaks;

}

INCTR\* **ReadFile**(QString fileName, QTextEdit\* te){

QFile file1(fileName);

file1.*open*(QIODevice::ReadOnly);

if (!file1.isOpen()){

QMessageBox msg;

msg.setText("File isn't open!");

msg.setWindowTitle("ERROR");

msg.*exec*();

return nullptr;

}

int i = 0;

QTextStream file1s(&file1);

QString temp = file1s.readAll();

te->setPlainText(temp);

peaks\_i=analyser(temp, i);

if(peaks\_i==-1)

return nullptr;

temp=corrector(temp);

INCTR\* peaks = new INCTR[peaks\_i];

peaks=Read(temp, peaks, peaks\_i);

return peaks;

}

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

tree = nullptr;

}

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

{

delete ui;

}

void MainWindow::**on\_action\_7\_triggered**()

{

tree=delete\_tree(tree, peaks\_i);

exit(0);

}

void MainWindow::**on\_action\_5\_triggered**()

{

tree=delete\_tree(tree, peaks\_i);

QString fileName = QFileDialog::getOpenFileName(this, "Open file");

tree = ReadFile(fileName, ui->textEdit);

ui->action\_3->setEnabled(true);

}

void MainWindow::**on\_action\_6\_triggered**()

{

QString text=ui->textEdit->toPlainText();

if(analyser(text, text.size())==1)

{

ui->textEdit\_3->setText("ERROR, You entered wrong tree!");

return;

}

if(tree==nullptr)

{

ui->textEdit\_3->setText("ERROR, no info for graph construction!");

return;

}

GraphWidget \*widget = new GraphWidget(this, tree, &gravity, peaks\_i, 0, nullptr);

QMainWindow\* temp = new QMainWindow;

temp->setCentralWidget(widget);

temp->setWindowTitle("Graph output");

temp->show();

}

void MainWindow::**on\_action\_triggered**()

{

ui->checkBox->setCheckState(Qt::Checked);

gravity = true;

}

void MainWindow::**on\_action\_2\_triggered**()

{

ui->checkBox->setCheckState(Qt::Unchecked);

gravity = false;

}

void MainWindow::**on\_checkBox\_clicked**()

{

if (ui->checkBox->checkState()){

gravity = true;

}

else

gravity = false;

}

void MainWindow::**on\_pushButton\_2\_clicked**()

{

//ui->textEdit\_5->clear();

if(tree!=nullptr)

{

if(ui->textEdit\_5->toPlainText()!="" && ui->textEdit\_6->toPlainText()!="")

{

//ui->textEdit\_4->clear();

int chosen\_peak = ui->textEdit\_5->toPlainText().toInt();

int secs = ui->textEdit\_6->toPlainText().toInt();

if(chosen\_peak<peaks\_i+1)

{

QMainWindow\* temp = new QMainWindow;

GraphWidget\* widget = new GraphWidget(this, tree, &gravity, peaks\_i, secs, steps[chosen\_peak-1]);

temp->setWindowTitle("Graph output");

temp->setCentralWidget(widget);

temp->show();

}

}

}

else

{

ui->textEdit\_2->clear();

ui->textEdit\_2->setText("Find circles first");

return;

}

}

void MainWindow::**on\_pushButton\_clicked**()

{

if(tree!=nullptr)

{

int step=1, fullstep=1, circle=0;

int\*\* gampath=new int\*[Factor(peaks\_i)+1];

for(int i=0; i<Factor(peaks\_i)+1; i++)

{

gampath[i]=nullptr;

gampath[i]=new int[peaks\_i+3];

}

int\*\* fullgampath=new int\*[peaks\_i];

for(int i=0; i<peaks\_i; i++)

{

fullgampath[i]=nullptr;

fullgampath[i]=new int[2\*Factor(peaks\_i)+1];

memset(fullgampath[i], 0, 2\*Factor(peaks\_i)+1\*sizeof(int));

}

steps=fullgampath;

for(int i=0; i<peaks\_i; i++)

{

gampath[circle][0]=tree[i].turn;

fullgampath[i][0]=tree[i].turn;

gampath=Gamilton(&tree[i], &tree[i], peaks\_i, step, fullstep, circle, gampath, fullgampath[i]);

while(gampath[circle] && gampath[circle][peaks\_i]==-1)

circle++;

}

gampath[circle]=NULL;

ui->textEdit\_3->clear();

ui->textEdit\_2->clear();

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

{

ui->textEdit\_2->append(QString::number(i+1)+')'+'\t'+'|');

for(int j=0; j<peaks\_i; j++)

{

for(int k=0; k<peaks\_i; k++)

{

if(gampath[i][j]==tree[k].turn)

ui->textEdit\_2->insertPlainText(QString::number(tree[k].turn)+'|');

}

}

ui->textEdit\_2->append((QString)'\n');

}

if(ui->textEdit\_4->toPlainText()!="")

{

int chosen\_circ = ui->textEdit\_4->toPlainText().toInt();

if(chosen\_circ<circle)

{

QMainWindow\* temp = new QMainWindow;

GraphWidget\* widget = new GraphWidget(this, tree, &gravity, peaks\_i, 0, gampath[chosen\_circ-1]);

temp->setWindowTitle("Graph output");

temp->setCentralWidget(widget);

temp->show();

}

else

{

ui->textEdit\_2->clear();

ui->textEdit\_2->setText("Wrong cycle");

return;

}

}

}

else

{

ui->textEdit\_3->clear();

ui->textEdit\_3->setText("No graph info");

return;

}

**node.h**

#ifndef NODE\_H

#define NODE\_H

#include <QGraphicsItem>

#include <QList>

#include <QTimer>

class Edge;

class GraphWidget;

QT\_BEGIN\_NAMESPACE

class QGraphicsSceneMouseEvent;

QT\_END\_NAMESPACE

//! [0]

class Node : public QGraphicsItem

{

public:

Node(GraphWidget \*graphWidget, bool \*gr);

void **addEdge**(Edge \*edge);

QList<Edge \*> **edges**() const;

enum { Type = UserType + 1 };

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

void **calculateForces**();

bool **advance**();

QRectF ***boundingRect***() const override;

QPainterPath ***shape***() const override;

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

QString text;

QString text\_orig;

protected:

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

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

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

private:

int tmr;

QList<Edge \*> edgeList;

QPointF newPos;

GraphWidget \*graph;

bool\* gravity;

};

//! [0]

#endif // NODE\_H

**node.cpp**

#include "edge.h"

#include "node.h"

#include "graphwidget.h"

#include "struct.h"

#include <QGraphicsScene>

#include <QGraphicsSceneMouseEvent>

#include <QPainter>

#include <QStyleOption>

#include <QString>

//! [0]

Node::**Node**(GraphWidget \*graphWidget, bool\* gr)

: graph(graphWidget)

{

setFlag(ItemIsMovable);

setFlag(ItemSendsGeometryChanges);

setCacheMode(DeviceCoordinateCache);

setZValue(-1);

gravity = gr;

}

//! [0]

//! [1]

void Node::**addEdge**(Edge \*edge)

{

edgeList << edge;

edge->adjust(text.size(), text\_orig.size());//fix\_needed

}

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

{

return edgeList;

}

//! [1]

//! [2]

//!

void Node::**calculateForces**()

{

if (!scene() || scene()->mouseGrabberItem() == this) {

newPos = pos();

return;

}

//! [2]

//! [3]

// Sum up all forces pushing this item away

qreal xvel = 0;

qreal yvel = 0;

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

Node \*node = qgraphicsitem\_cast<Node \*>(item);

if (!node)

continue;

QPointF vec = mapToItem(node, 0, 0);

qreal dx = vec.x();

qreal dy = vec.y();

double l = 2 \* (dx \* dx + dy \* dy);

if (l > 0) {

xvel += (dx \* 150.0) / l;

yvel += (dy \* 150.0) / l;

}

}

//! [3]

//! [4]

// Now subtract all forces pulling items together

double weight = (edgeList.size() + 1) \* 10;

foreach (Edge \*edge, edgeList) {

QPointF vec;

if (edge->sourceNode() == this)

vec = mapToItem(edge->destNode(), 0, 0);

else

vec = mapToItem(edge->sourceNode(), 0, 0);

xvel -= vec.x() / weight;

yvel -= vec.y() / weight;

}

//! [4]

//! [5]

qreal lim = 0.01;

// if(!\*gravity==false)

// {

// }

if (\*gravity == false){

lim = 1000;

}

if (qAbs(xvel) < lim && qAbs(yvel) < lim)

xvel = yvel = 0;

//! [5]

//! [6]

QRectF sceneRect = scene()->sceneRect();

newPos = pos() + QPointF(xvel, yvel);

newPos.setX(qMin(qMax(newPos.x(), sceneRect.left() + 100), sceneRect.right() - 100));

newPos.setY(qMin(qMax(newPos.y(), sceneRect.top() + 100), sceneRect.bottom() - 100));

}

//! [6]

//! [7]

bool Node::**advance**()

{

if (newPos == pos())

return false;

setPos(newPos);

return true;

}

//! [7]

//! [8]

QRectF Node::***boundingRect***() const

{

qreal adjust = 2;

qreal mult = 6.25;

if(text.size()==1)

return QRectF( -(2\*mult\*text.size()) - adjust, -10 - adjust, 4\*mult\*text.size() + adjust, 23 + adjust);

return QRectF( -(2\*mult\*text.size()) - adjust, -10 - adjust, 4\*mult\*text.size() + adjust, 23 + adjust);

}

//! [8]elements and text size

//! [9]

QPainterPath Node::***shape***() const

{

qreal mult=5;

QPainterPath path;

path.addEllipse(-mult\*(text.size()+1), -10, 2\*mult\*(text.size()+1), 20);

return path;

}

//! [9]

//! [10]

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

{

qreal mult=5;

painter->setPen(Qt::NoPen);

painter->setBrush(Qt::darkGray);

if(text.size()==1)

painter->drawEllipse(-(mult-0.5)\*(text.size()+1), -7, 2\*mult\*(text.size()+1), 20);

else

painter->drawEllipse(-(mult-0.5)\*(text.size()), -7, 2\*mult\*(text.size()), 20);

QRadialGradient gradient(-3, -3, mult\*(text.size()+1.25));

if (option->state & QStyle::State\_Sunken) {

gradient.setCenter(3, 3);

gradient.setFocalPoint(3, 3);

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

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

} else {

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

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

}

painter->setBrush(gradient);

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

if(text.size()==1)

painter->drawEllipse(-mult\*(text.size()+1), -10, 2\*mult\*(text.size()+1), 20);

else

painter->drawEllipse(-mult\*(text.size()), -10, 2\*mult\*(text.size()), 20);

QFont font = painter->font();

font.setBold(true);

font.setPointSize(10);

painter->setFont(font);

double offset=-3.8;

painter->drawText(offset\*text.size(), 5, text);

}

//! [10]

//! [11]

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

{

switch (change) {

case ItemPositionHasChanged:

foreach (Edge \*edge, edgeList)

edge->adjust(text.size(), text\_orig.size());//fix\_needed

graph->itemMoved();

break;

default:

break;

};

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

}

//! [11]

//! [12]

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

{

update();

QGraphicsItem::*mousePressEvent*(event);

}

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

{

update();

QGraphicsItem::*mouseReleaseEvent*(event);

}

//! [12]

**edge.h**

#ifndef EDGE\_H

#define EDGE\_H

#include <QGraphicsItem>

class Node;

//! [0]

class Edge : public QGraphicsItem

{

public:

Node \*source, \*dest;

bool gamilt\_clr;

Edge(Node \*sourceNode, Node \*destNode, bool vis, int size\_source, int size\_dest, bool gamilt\_colour);

Node \***sourceNode**() const;

Node \***destNode**() const;

void **adjust**(int size\_source, int size\_dest);

enum { Type = UserType + 2 };

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

protected:

QRectF ***boundingRect***() const override;

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

private:

QPointF sourcePoint;

QPointF destPoint;

qreal arrowSize;

bool visible;

};

//! [0]

#endif // EDGE\_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;

//! [0]

Edge::**Edge**(Node \*sourceNode, Node \*destNode, bool vis, int size\_source, int size\_dest, bool gamilt\_colour)

: arrowSize(10)

{

setAcceptedMouseButtons(0);

source = sourceNode;

dest = destNode;

visible = vis;

source->addEdge(this);

dest->addEdge(this);

gamilt\_clr = gamilt\_colour;

adjust(size\_source, size\_dest);

}

//! [0]

//! [1]

Node \*Edge::**sourceNode**() const

{

return source;

}

Node \*Edge::**destNode**() const

{

return dest;

}

//! [1]

//! [2]

void Edge::**adjust**(int size\_source, int size\_dest)

{

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\_source((line.dx() \* size\_source \* 5) / length, (line.dy() \* 10) / length);

QPointF edgeOffset\_dest((line.dx() \* size\_dest \* 5) / length, (line.dy() \* 10) / length);

sourcePoint = line.p1() + edgeOffset\_source;

destPoint = line.p2() - edgeOffset\_dest;

} else {

sourcePoint = destPoint = line.p1();

}

}

//! [2]

//! [3]

QRectF Edge::***boundingRect***() const

{

if (!source || !dest)

return QRectF();

qreal penWidth = 10;

qreal extra = (penWidth + arrowSize) / 2.0;

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

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

.normalized()

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

}

//! [3]

//! [4]

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

{

if (!source || !dest)

return;

QLineF line(sourcePoint, destPoint);

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

return;

//! [4]

//! [5]

if (visible){

// Draw the line itself

if(gamilt\_clr==true)

{

painter->setPen(QPen(Qt::blue, 1, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter->drawLine(line);

}

else

{

painter->setPen(QPen(Qt::red, 1, Qt::DotLine, Qt::RoundCap, Qt::RoundJoin));

painter->drawLine(line);

}

//! [5]

//! [6]

// Draw the arrows

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

if (line.dy() >= 0)

angle = TwoPi - angle;

if(gamilt\_clr==true)

{

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

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

QPointF sourceArrowP2 = sourcePoint + QPointF(sin(angle + Pi - Pi / 3) \* arrowSize,

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

painter->setBrush(Qt::blue);

painter->drawPolygon(QPolygonF() << line.p1() << sourceArrowP1 << sourceArrowP2);

}

else

{

painter->setBrush(Qt::red);

QPointF sourceArrowP1 = sourcePoint + QPointF(sin(angle + Pi / 3) \* arrowSize/1.5,

cos(angle + Pi / 3) \* arrowSize/1.5);

QPointF sourceArrowP2 = sourcePoint + QPointF(sin(angle + Pi - Pi / 3) \* arrowSize/1.5,

cos(angle + Pi - Pi / 3) \* arrowSize/1.5);

painter->drawPolygon(QPolygonF() << line.p1() << sourceArrowP1 << sourceArrowP2);

}

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

}

}

//! [6]

**main.cpp**

#include "mainwindow.h"

#include "graphwidget.h"

#include <QApplication>

#include <QTime>

#include <QMainWindow>

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

{

QApplication a(argc, argv);

qsrand(QTime(0,0,0).secsTo(QTime::currentTime()));

MainWindow w;

w.show();

return a.exec();

}