**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

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

Тема: Поиск Эйлерового цикла

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Корытов П.В. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2018

**Цель работы.**

Изучить алгоритм поиска Эйлерового цикла в графе и его реализацию на языке C++.

**Основные теоретические положения.**

*Эйлеров путь* – путь, проходящий по всем ребрам графа ровно один раз. Если начало пути совпадает с концом, это *эйлеров цикл*.

Согласно теореме, чтобы в графе имелся эйлеров цикл, необходимо и достаточно, чтобы вершины графа имели четную степень.

Алгоритм поиска эйлерового цикла на псевдокоде в книге Либского выглядит следующим образом:

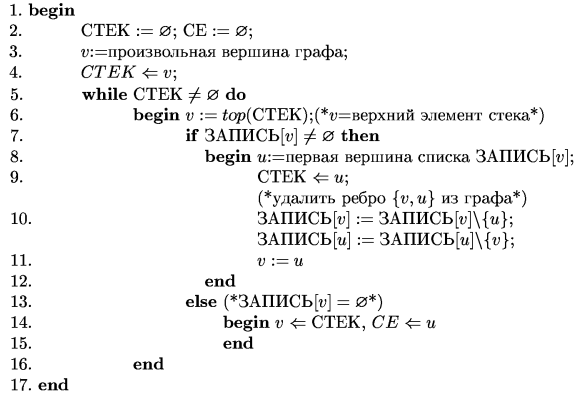


Рисунок 1 – алгоритм поиска эйлерового цикла на псевдокоде

Алгоритм начинает строить путь с определенной вершины. Вершины помещаются во временный стек СТЕК, а ребра удаляются из графа. Действия продолжаются до тех пор, пока список инцидентности текущей вершины не опустеет.

Для того, чтобы начальная вершина была равна конечной, степень вершины очевидно должна быть четной. Кроме того, после завершения текущего шага все вершины в графе остаются четными.

В результате этих шагов из графа был удален цикл, а его вершины были записаны во временный стек. Текущая вершина записывается во второй стек СЕ, и очередной вершиной становится верхний элемент стека.

Процесс повторяется, и в стеке снова оказывается цикл, содержащий данную вершину. Все продолжается до того момента, пока стек не опустеет.

Очевидно, что вершина записывается во второй стек только после того, как все инцидентные ей ребра были удалены из стека. Значит, по окончании работы второй стек содержит эйлеров цикл.

**Ход работы**

1. Написана основа для работа с графами – класс Graph.

Сам граф хранится с помощью двух структур:

typedef struct Elem{ //Элементы графа

Elem() = default;

char name[Numb];

Elem\* next=nullptr;

List\* childs=nullptr; //Дети элемента

Node\* node;

}Elem;

typedef struct List{

List() = default;

char name[Numb];

bool mark = false;

Elem\* node = nullptr; //Сам элемент

List\* next = nullptr;

Edge\* edge;

}List;

В первой структуре хранятся элементы графа, а во второй – его сыновья. Каждая структура содержит указатель на своего представителя в графической сцене. Ребра содержат не только указатель на второй элемент, но и его имя. Это связано с тем, что при вводе из файла может быть дана ссылка на вершину, которой ещё нет в основном списке. В таком случае записывается только имя, и зависимости разрешаются позднее.

Сам класс выглядит следующим образом:

class Graph

{

public:

Graph();

~Graph();

void **Clear**(); //Очистка

Elem\* **it**(); //Итератор через вершины

List\* **it**(Elem\* el, bool marked = 1); //Итератор через сыновей

void **ResetIts**();

void **ReadFile**(QString fileName); //Считать из файла

void **SaveFile**(QString fileName); //Сохранить в файл

char\* **GetMinStupidName**(); //Получение минимального числового имени

char\* **GetLastStupidName**(); //Получение последнего такого

void **AddElem**(char\* name); //Добавить элемент

void **AddEdge**(Elem\* el1, Elem\* el2); //Добавить линию

void **AddEdge**(Elem\* el1, char\* name); //Добавить "заготовку" для линии

bool **Solve**(); //Разрешить зависимости "заготовок" для линий

void **RemoveElem**(char\* name); //Удалить элемент

void **RemoveEdge**(Elem\* el1, Elem\* el2); //Удалить линию

void **RemoveEdges**(Elem\* el); //Удалить связи элемента

void **RenameElem**(char\* oldname, char\* newname); //Переименовать элемент

void **Desorientate**(); //Снять ориентацию графа

Elem\* **FindElem**(char\* name); //Найти элемент

int **CountChildren**(Elem\* el, bool marked = 1); //Сколько детей

int **CountElems**(); //Сколько элементов

bool **Is\_Egde**(Elem\* el1, Elem\* el2); //Есть ли связь от 1 к 2

List\* **GetEdge**(Elem\* el1, Elem\* el2); //Обратить связь

int **Max\_Width**(); //Максимальная длина имени в графе

void **Inc\_Matr**(QTextStream& os); //Матрица инцидентности

bool **Euler**(); //Один шаг поиска эйлерового цикла

void **ResetEuler**(); //Сброс эйлеровых циклов

void **ClearMarks**(); //Очистка меток исключений на графе

Elem\* **operator**[](int i); //Доступ к элементам по индексу

GraphWidget\* widget;

QStack<Elem\*> SE;

QStack<Elem\*> Stack;

Elem\* v0 = nullptr;

private:

int stupidnames = 0;

Elem\* gr;

Elem\* pos;

List\* lpos;

void **Clear**(List\* ls, Elem \*el);

void **Clear**(Elem\* gr);

Elem\* **KeepItE**();

List\* **KeepItL**();

void **RestoreItE**(Elem\* t\_pos);

void **RestoreItL**(List\* t\_lpos);

};

Пояснения ко всем публичным методам даны в комментариях.

Помимо всего прочего, класс включает в себя макросы для сохранения и восстановления итераторов:

#define SAVEITS List\* t\_lpos = KeepItL(); Elem\* t\_pos = KeepItE();

#define RESTOREITS RestoreItL(t\_lpos); RestoreItE(t\_pos);

Необходимость в них связана с тем, что запуск методов класса не должен ломать итераторы, запущенные на уровень выше.

1. Для обеспечения наглядной работы данный алгоритм изменен – так, цикл while заменен на if, чтобы обеспечит пошаговое выполнение. Кроме того, ребра не удаляются из графа, а «помечаются». Это необходимо для возможности повторной прогонки алгоритма. Также необходимо периодическое обновление элементов графической сцены – текущий элемент и «помеченные» ребра отображаются особым образом.

В остальном алгоритм остался идентичен данному на рисунке 1.

bool Graph::**Euler**()

{

SAVEITS; //Сохранение и сброс итераторов

Elem\* v;

List\* u;

if (Stack.isEmpty()){ //Первый запуск

ResetEuler();

Stack.push(gr);

}

if (!Stack.isEmpty()){

v = Stack.top();

if (CountChildren(v, 0)){

if (!v0)

v0 = v;

u = it(v, 0); //Итератор через непомеченные ребра вершины

Stack.push(u->node);

v->node->update(); //Обновление картинки в графе

u->mark = 1; //Пометка ребра в одну сторону

if (Is\_Egde(u->node, v))

GetEdge(u->node, v)->mark = 1; //Пометка ребра в другую сторону

u->edge->update(); //Обновление картинки

v = u->node;

}

else{

if ((v0!=v) && (v0!=nullptr)){

QMessageBox msg; //Если зашли в тупик

msg.setText("В графе тупик. Эйлерова цикла нет");

msg.*exec*();

return 1;

}

else{

v0 = nullptr;

v = Stack.pop();

if (v)

v->node->update(); //Обновление картинки

SE.push(v);

}

}

if (!Stack.isEmpty())

Stack.top()->node->update();

}

RESTOREITS; //Восстановление итераторов

return 0;

}

1. Реализовано отображение графа на QGraphicsWidget с применением информации из примера Elastic Nodes.

Оригинальные методы изменены и расширены. Существенные изменения перечислены ниже:

* Класс GraphWidget теперь полностью взаимосвязан с классом Graph. Работа с элементами графа в основном осуществляется через последний – так, изменение в классе (с помощью публичных методов) Graph неизменно влечет за собой изменение в GraphWidget, но и обратное тоже верно.
* Расширены возможности форматирования графа – ребра теперь могут быть направлены в одну сторону, на вершинах размещается текст, величина которого настраивается динамически так, чтобы он помещался. Вершины можно перекрасить – цвет шрифта также подстраивается автоматически.



Рисунок 2 – новые вершины

* Добавлена возможность выключения взаимодействия между вершинами – «гравитации».

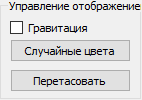


Рисунок 3 – некоторые элементы управления сценой

* Ребра графа теперь можно «отмечать» для наглядной демонстрации работы алгоритма.
* Добавлена возможность выделения вершин при помощи клавиши Ctrl. Выделенные вершины перемещаются вместе.
* Выделенные вершины можно удалить кнопкой Delete.
* Добавлено добавление вершины двойным шелчком. Вершине присваивается числовое имя.
* Добавлено контекстное меню для каждой вершины с важными действиями.

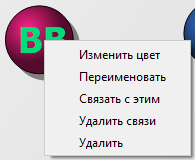


Рисунок 4 – контекстное меню.

1. Реализован графический интерфейс для работы с алгоритмом.

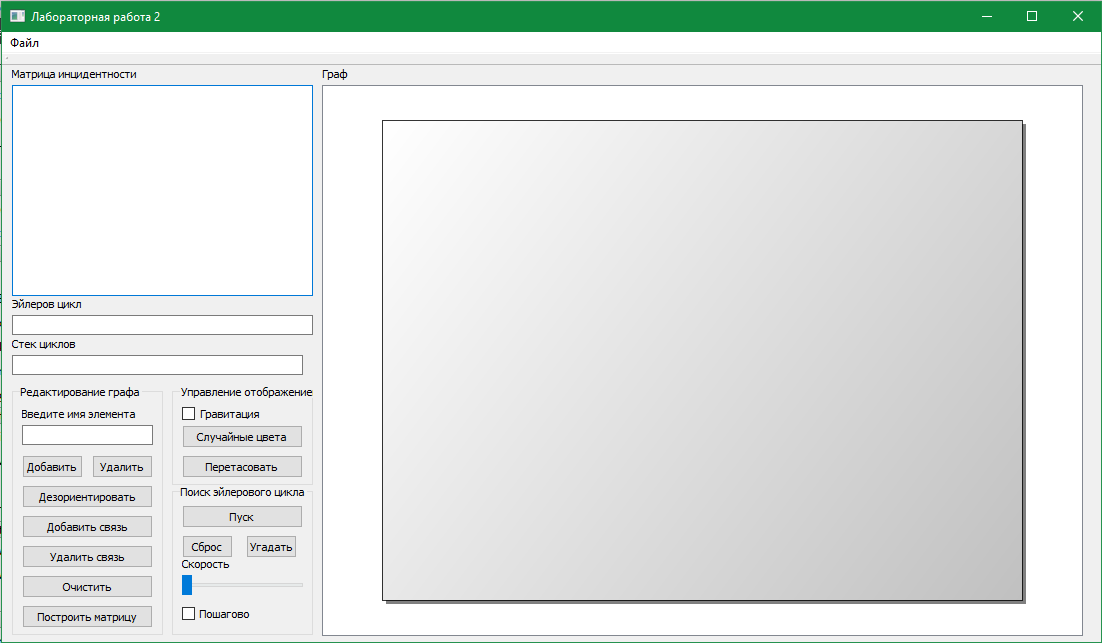


Рисунок 5 – графический интерфейс программы.

1. Проведено тестирование программы на простом примере

|  |  |  |  |
| --- | --- | --- | --- |
| Шаг | Стек | СЕ | Вид графа |
| 1 | 1 0 |  |  |
| 2 | 2 1 0 |  |  |
| 3 | 0 2 1 0 |  |  |
| 4 | 2 1 0 | 0 |  |
| 5 | 1 0 | 2 0 |  |
| 6 | 0 | 1 2 0 |  |
| 7 |  | 0 1 2 0 |  |

Как видно, алгоритм успешно завершил работу. Найден очевидный цикл – 0 1 2 0.

1. Взят пример из книги Либского:

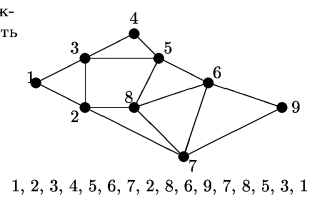


Рисунок 6 – пример из Либского

После работы программы получен Эйлеров цикл:

1 2 3 4 5 6 7 2 8 6 9 7 8 5 3 1

Как видно, результат совпадает с результатом, данным в книге.

1. В соответствии с указаниями, реализована возможность угадать некоторые из вершин эйлерового цикла.

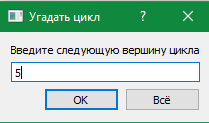


Рисунок 7 – угадывание цикла.

Естественно, результат выполнения алгоритма скрывается, пока активен диалог с пользователем. После неудачной попытки угадать, нажатия кнопки «Всё» или угадывания последней вершины выводится количество правильно угаданных вершин.

**Выводы.**

Изучена работа с графами на языке C++ - хранение графа в виде связанных списков, добавление, изменение и удаление ребер и вершин, получение матрицы инцидентности и т.п. Изучено написание итераторов для удобного прохода через списки. Написан и отлажен алгоритм поиска Эйлерового цикла.

Изучена работа с классами QGraphicsWidget, QGraphicsScene, QGraphicsItem для рисования графа. Изучены перезаписи событий классов QMouseEvent и QContextMenuEvent (а также их аналогов в графической сцене) для обеспечения взаимодействия с графами. Для этой же цели изучен QKeyEvent.

Изучено создание диалогов класса QInputDialog для взаимодействия с пользователем.

Для обеспечения хорошей работы алгоритма изучен класс QTimer и возможности сигналов и слотов для обеспечения работоспособности второго.

Приложение А

Код MAIN.CPP

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.app = a.instance();

w.show();

return a.exec();

}

Приложение Б

Код MAINWINDOW.H

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QApplication>

#include <QDebug>

#include <QTimer>

#include <QMessageBox>

#include <QKeyEvent>

#include "graph.h"

namespace Ui {

class MainWindow;

}

class Graph;

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

QCoreApplication\* app;

~***MainWindow***();

void **update\_matr**();

int **get\_speed**();

public slots:

void **One\_Step**();

protected:

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

private slots:

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

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

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

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

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

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

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

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

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

void **on\_gravityBox\_stateChanged**(int arg1);

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

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

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

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

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

void **on\_speedSlider\_sliderMoved**(int position);

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

private:

Graph\* gr1;

Ui::MainWindow \*ui;

QTimer\* timer;

};

#endif // MAINWINDOW\_H

Приложение В

Код MAINWINDOW.CPP

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "graphwidget.h"

#include <QTextStream>

#include <QByteArray>

#include <QInputDialog>

#include <QFileDialog>

#include <cstdlib>

#include <ctime>

#include <QTimer>

#include <QGraphicsScene>

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

gr1 = new Graph;

gr1->widget = new GraphWidget(0, gr1, this);

ui->graphicsView->setViewport(gr1->widget);

srand(time(nullptr));

timer = nullptr;

}

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

{

delete ui;

}

void MainWindow::**update\_matr**()

{

on\_matrButton\_clicked();

}

int MainWindow::**get\_speed**()

{

return ui->speedSlider->value();

}

void MainWindow::**One\_Step**() //Один шаг алгоритма

{

if ((!ui->cycleEdit->text().isEmpty()) && (gr1->Stack.isEmpty())){ //Последний шаг

QMessageBox msg;

msg.setText("Цикл найден");

msg.*exec*();

if (timer){

timer->stop();

delete timer;

timer = nullptr;

return;

}

}

bool res = gr1->Euler(); //Один шаг делает метод в Graph

QString str;

QTextStream strm(&str);

for (auto it : gr1->SE){

strm << it->name << " "; //Вывод данных стека для цикла Эйлера

}

std::reverse(str.begin(), str.end());

ui->cycleEdit->setText(str);

str.clear();

strm.reset();

for (auto it : gr1->Stack){

strm << it->name << " "; //Вывод данных временного цикла

}

std::reverse(str.begin(), str.end());

ui->stackEdit->setText(str);

if (res){ //Остановка таймера, если что-то не так

timer->stop();

delete timer;

timer = nullptr;

}

}

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

{

gr1->widget->*keyPressEvent*(event); //Передача управления виджету

}

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

{

exit(0); //Выход из программы

}

void MainWindow::**on\_matrButton\_clicked**() //Построение матрицы

{

QString matr;

QTextStream ms(&matr);

gr1->Inc\_Matr(ms);

ui->incEdit->setPlainText(matr);

}

void MainWindow::**on\_addNodeButton\_clicked**() //Добавление вершины

{

QString name = ui->NodeNameEdit->text();

if (name.length() > Numb){

QMessageBox msg;

msg.setText("Слишком длинное имя");

msg.*exec*();

ui->NodeNameEdit->clear();

return;

}

QByteArray arr = name.toLocal8Bit(); //Конвертация QString в char\*

char\* cname = arr.data();

if (!gr1->FindElem(cname))

gr1->AddElem(cname);

else{

QMessageBox msg;

msg.setText("Элемент уже существует");

msg.*exec*();

}

ui->NodeNameEdit->clear();

on\_matrButton\_clicked(); //После каждой процедуры матрица обновляется

}

void MainWindow::**on\_deleteNodeButton\_clicked**() //Удаление вершины

{

QString name = ui->NodeNameEdit->text();

QByteArray arr = name.toLocal8Bit();

char\* cname = arr.data();

if (gr1->FindElem(cname))

gr1->RemoveElem(cname);

else{

QMessageBox msg;

msg.setText("Элемент не существует");

msg.*exec*();

}

ui->NodeNameEdit->clear();

on\_matrButton\_clicked();

}

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

{

QString name = ui->NodeNameEdit->text();

QByteArray arr = name.toLocal8Bit();

char\* cname = arr.data();

Elem\* el1 = gr1->FindElem(cname);

if (!el1){

QMessageBox msg;

msg.setText("Элемент 1 не существует");

msg.*exec*();

ui->NodeNameEdit->clear();

return;

}

bool ok;

QString name2 = QInputDialog::getText(this, "Добавить связь", "Введите имя", QLineEdit::Normal, "", &ok);

QByteArray arr2 = name2.toLocal8Bit();

char\* cname2 = arr2.data();

Elem\* el2 = gr1->FindElem(cname2);

if (!el2){

QMessageBox msg;

msg.setText("Элемент 2 не существует");

msg.*exec*();

ui->NodeNameEdit->clear();

return;

}

if (!gr1->Is\_Egde(el1, el2)){

gr1->AddEdge(el1, el2);

}

else{

QMessageBox msg;

msg.setText("Связь уже есть");

msg.*exec*();

}

on\_matrButton\_clicked();

ui->NodeNameEdit->clear();

}

void MainWindow::**on\_DeleteEdgeButton\_clicked**() //Удаление ребра

{

QString name = ui->NodeNameEdit->text();

QByteArray arr = name.toLocal8Bit();

char\* cname = arr.data();

Elem\* el1 = gr1->FindElem(cname);

if (!el1){

QMessageBox msg;

msg.setText("Элемент не существует");

msg.*exec*();

ui->NodeNameEdit->clear();

return;

}

bool ok;

QString name2 = QInputDialog::getText(this, "Удалить связь", "Введите имя", QLineEdit::Normal, "", &ok);

QByteArray arr2 = name2.toLocal8Bit();

char\* cname2 = arr2.data();

Elem\* el2 = gr1->FindElem(cname2);

if (!el2){

QMessageBox msg;

msg.setText("Элемент 2 не существует");

msg.*exec*();

ui->NodeNameEdit->clear();

return;

}

if (gr1->Is\_Egde(el1, el2)){

gr1->RemoveEdge(el1, el2);

}

else{

QMessageBox msg;

msg.setText("Связи нет");

msg.*exec*();

}

on\_matrButton\_clicked();

ui->NodeNameEdit->clear();

}

void MainWindow::**on\_ClearButton\_clicked**() //Сброс графа

{

gr1->Clear();

on\_matrButton\_clicked();

ui->NodeNameEdit->clear();

gr1->widget->update();

}

void MainWindow::**on\_open\_action\_triggered**() //Открытие из файла

{

QString fileName = QFileDialog::getOpenFileName(this, "Открыть файл");

gr1->Clear();

gr1->ReadFile(fileName);

on\_matrButton\_clicked();

}

void MainWindow::**on\_save\_action\_triggered**() //Сохранение в файл

{

QString fileName = QFileDialog::getSaveFileName(this, "Открыть файл");

gr1->SaveFile(fileName);

}

void MainWindow::**on\_gravityBox\_stateChanged**(int arg1) //Изменение гравитации

{

if (arg1 == 0){

gr1->widget->SetGravity(0);

}

else

gr1->widget->SetGravity(1);

}

void MainWindow::**on\_colorButton\_clicked**() //Рандомизация цветов

{

gr1->widget->RandomColors();

}

void MainWindow::**on\_shuffleButton\_clicked**() //Перетасовка вершин

{

gr1->widget->shuffle();

}

void MainWindow::**on\_pushButton\_clicked**() //Запуск алгоритма

{

if (ui->stepBox->isChecked()) //Если пошаговое выполнение, просто выполнить один шаг

One\_Step();

else if (!timer){

timer = new QTimer(this); //Иначе выполнение по таймеру

connect(timer, SIGNAL(timeout()), this, SLOT(One\_Step()));

timer->start(ui->speedSlider->value());

}

}

void MainWindow::**on\_clrMarksButton\_clicked**() //Сброс алгоритма

{

gr1->ResetEuler();

gr1->ClearMarks();

ui->stackEdit->clear();

ui->cycleEdit->clear();

if (timer){

timer->stop();

delete timer;

timer = nullptr;

}

}

void MainWindow::**on\_desButton\_clicked**() //Убрать ориентацию графа

{

gr1->Desorientate();

gr1->widget->update();

on\_matrButton\_clicked();

}

void MainWindow::**on\_speedSlider\_sliderMoved**(int position) //Изменение скорости

{

if (timer){

timer->setInterval(position);

}

}

void MainWindow::**on\_guessButoon\_clicked**() //Угадать цикл

{

ui->speedSlider->setValue(0);

if (!timer)

on\_pushButton\_clicked();

QPalette p1 = ui->cycleEdit->palette(); //Изменение цвета шрифта

QPalette\* p2 = new QPalette(p1);

p2->setColor(QPalette::Text, Qt::white);

ui->cycleEdit->setPalette(\*p2);

while (timer){

app->processEvents(); //Пока идет таймер, обрабатывать ивенты приложения

}

if (!(ui->stackEdit->text().isEmpty()) || (ui->cycleEdit->text().isEmpty())){ //Если некорректно завершилось

ui->cycleEdit->setPalette(p1);

delete p2;

return;

}

QString txt; int g = gr1->SE.count()-1; bool alright = 0;

do{ //Цикл с угадыванием

alright = 1;

QInputDialog\* dia = new QInputDialog;

dia->setInputMode(QInputDialog::TextInput);

dia->setWindowTitle("Угадать цикл");

dia->setLabelText("Введите следующую вершину цикла");

dia->setCancelButtonText("Всё");

dia->*exec*();

txt = dia->textValue();

if (txt.isSimpleText()){

bool res = !(QString::compare(txt, gr1->SE[g]->name));

if ((!res) && (!txt.isEmpty())){

QMessageBox box;

box.setText("Неверно!");

box.*exec*();

break;

}

}

g--;

delete dia;

} while (!txt.isEmpty() && (g>=0));

ui->cycleEdit->setPalette(p1); //Восстановление цвета

delete p2;

if (alright){ //Подсчёт угаданного

QString final;

QTextStream finals(&final);

finals << "Correct: " << gr1->SE.count() - g - 1 << " of " << gr1->SE.count() << endl;;

QMessageBox box;

box.setText(final);

box.*exec*();

}

}

Приложение Г

Код GRAPH.H

#ifndef GRAPH\_H

#define GRAPH\_H

#define SAVEITS List\* t\_lpos = KeepItL(); Elem\* t\_pos = KeepItE();

#define RESTOREITS RestoreItL(t\_lpos); RestoreItE(t\_pos);

#define Numb 80

#define Wid 800

#define Hei 600

#include <iostream>

#include <cstring>

#include <QTextStream>

#include <QString>

#include <QFile>

#include <QStack>

#include <QMessageBox>

#include <QTime>

#include "graphwidget.h"

using namespace std;

typedef struct List List;

typedef struct Elem{ //Элементы графа

Elem() = default;

char name[Numb];

Elem\* next=nullptr;

List\* childs=nullptr; //Дети элемента

Node\* node;

}Elem;

typedef struct List{

List() = default;

char name[Numb];

bool mark = false;

Elem\* node = nullptr; //Сам элемент

List\* next = nullptr;

Edge\* edge;

}List;

class Graph

{

public:

Graph();

~Graph();

void **Clear**(); //Очистка

Elem\* **it**(); //Итератор через вершины

List\* **it**(Elem\* el, bool marked = 1); //Итератор через сыновей

void **ResetIts**();

void **ReadFile**(QString fileName); //Считать из файла

void **SaveFile**(QString fileName); //Сохранить в файл

char\* **GetMinStupidName**(); //Получение минимального числового имени

char\* **GetLastStupidName**(); //Получение последнего такого

void **AddElem**(char\* name); //Добавить элемент

void **AddEdge**(Elem\* el1, Elem\* el2); //Добавить линию

void **AddEdge**(Elem\* el1, char\* name); //Добавить "заготовку" для линии

bool **Solve**(); //Разрешить зависимости "заготовок" для линий

void **RemoveElem**(char\* name); //Удалить элемент

void **RemoveEdge**(Elem\* el1, Elem\* el2); //Удалить линию

void **RemoveEdges**(Elem\* el); //Удалить связи элемента

void **RenameElem**(char\* oldname, char\* newname); //Переименовать элемент

void **Desorientate**(); //Снять ориентацию графа

Elem\* **FindElem**(char\* name); //Найти элемент

int **CountChildren**(Elem\* el, bool marked = 1); //Сколько детей

int **CountElems**(); //Сколько элементов

bool **Is\_Egde**(Elem\* el1, Elem\* el2); //Есть ли связь от 1 к 2

List\* **GetEdge**(Elem\* el1, Elem\* el2); //Обратить связь

int **Max\_Width**(); //Максимальная длина имени в графе

void **Inc\_Matr**(QTextStream& os); //Матрица инцидентности

bool **Euler**(); //Один шаг поиска эйлерового цикла

void **ResetEuler**(); //Сброс эйлеровых циклов

void **ClearMarks**(); //Очистка меток исключений на графе

Elem\* **operator**[](int i); //Доступ к элементам по индексу

GraphWidget\* widget;

QStack<Elem\*> SE;

QStack<Elem\*> Stack;

Elem\* v0 = nullptr;

private:

int stupidnames = 0;

Elem\* gr;

Elem\* pos;

List\* lpos;

void **Clear**(List\* ls, Elem \*el);

void **Clear**(Elem\* gr);

Elem\* **KeepItE**();

List\* **KeepItL**();

void **RestoreItE**(Elem\* t\_pos);

void **RestoreItL**(List\* t\_lpos);

};

#endif // GRAPH\_H

Приложение Д

Код GRAPH.CPP

#include "graph.h"

#include <cstdio>

Graph::**Graph**()

{

gr = nullptr;

ResetIts();

}

Graph::~Graph()

{

Clear();

}

void Graph::**Clear**()

{

Clear(gr);

gr = nullptr;

stupidnames = 0;

}

void Graph::**Clear**(List \*ls, Elem\* el)

{

if (ls != nullptr) {

if (ls->next != nullptr)

Clear(ls->next, el);

RemoveEdge(el, ls->node);

}

}

void Graph::**Clear**(Elem \*gr)

{

if (gr){

if (gr->next != nullptr) {

Clear(gr->next);

}

Clear(gr->childs, gr);

RemoveElem(gr->name);

}

}

Elem \*Graph::**it**()

{

if (pos == nullptr)

pos = gr;

else{

if (gr){

pos = pos->next;

}

}

return pos;

}

List \*Graph::**it**(Elem \*el, bool marked)

{

if (el == nullptr)

lpos = nullptr;

else{

if (lpos == nullptr){

lpos = el->childs;

if (!marked){

while ((lpos!=nullptr) && (lpos->mark))

lpos = lpos->next;

}

}

else{

lpos = lpos->next;

if (!marked){

while ((lpos!=nullptr) && (lpos->mark))

lpos = lpos->next;

}

}

}

return lpos;

}

void Graph::**ResetIts**()

{

lpos = nullptr;

pos = nullptr;

}

void Graph::**ReadFile**(QString fileName)

{

SAVEITS;

QFile file(fileName);

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

QTextStream file1s(&file);

QString str;

QString tmp;

int i; int k = 0;

while (!file1s.atEnd()){

str = file1s.readLine();

QStringList strl = str.split(' ');

QByteArray arr;

arr = strl.at(0).toLocal8Bit();

char\* name = arr.data();

AddElem(name);

for (i=1; i < strl.size(); i++){

arr = strl.at(i).toLocal8Bit();

name = arr.data();

AddEdge(this->operator [](k), name);

}

k++;

}

Solve();

file.*close*();

RESTOREITS;

}

void Graph::**SaveFile**(QString fileName)

{

SAVEITS;

QFile file(fileName);

file.*open*(QIODevice::WriteOnly);

QTextStream files(&file);

Elem\* el;

List\* ls;

while ((el = it())!=nullptr){

files << el->name << " ";

while ((ls = it(el))!=nullptr){

files << ls->name << " ";

}

files << "\r\n";

}

file.*close*();

RESTOREITS;

}

char \*Graph::**GetMinStupidName**()

{

char\* name = new char[80];

sprintf(name, "%d", stupidnames);

while (FindElem(name)){

stupidnames++;

sprintf(name, "%d", stupidnames);

}

return name;

}

char \*Graph::**GetLastStupidName**()

{

char\* name = new char[80];

sprintf(name, "%d", stupidnames);

return name;

}

void Graph::**AddElem**(char \*name)

{

SAVEITS;

Elem\* el;

if (!gr){ //Если добавляется первый элемент

gr = new Elem;

strcpy\_s(gr->name, Numb, name);

gr->node = new Node(widget);

gr->node->name = (gr->name);

widget->centerNode = gr->node;

widget->scene()->addItem(widget->centerNode);

widget->centerNode->setPos(rand()%Wid - Wid/2 ,rand()%Hei - Hei/2);

}

else{

while ((el = it())->next!=nullptr);

el->next = new Elem;

el = el->next;

strcpy\_s(el->name, Numb, name);

el->node = new Node(widget);

el->node->name = el->name;

el->node->setPos(rand()%Wid - Wid/2 ,rand()%Hei - Hei/2);

widget->scene()->addItem(el->node);

}

RESTOREITS;

}

Elem \*Graph::**FindElem**(char \*name)

{

SAVEITS;

Elem\* el;

bool res = 0;

while ((el = it())!=nullptr){

res = (strncmp(name, el->name, Numb))==0;

if (res)

break;

}

RESTOREITS;

return el;

}

void Graph::**AddEdge**(Elem \*el1, Elem \*el2)

{

SAVEITS;

List\* ls;

if (Is\_Egde(el1, el2))

return;

if (el1 && el2){

if (!el1->childs){

el1->childs = new List;

strcpy\_s(el1->childs->name, el2->name);

el1->childs->node = el2;

el1->childs->edge = new Edge(el1->node, el2->node, el1->childs);

el1->childs->next = nullptr;

widget->scene()->addItem(el1->childs->edge);

}

else{

while ((ls = it(el1))->next!=nullptr);

ls->next = new List;

ls = ls->next;

strcpy\_s(ls->name, el2->name);

ls->node = el2;

ls->edge = new Edge(el1->node, el2->node, ls);

ls->next = nullptr;

widget->scene()->addItem(ls->edge);

}

}

RESTOREITS;

}

void Graph::**AddEdge**(Elem \*el1, char \*name)

{

SAVEITS;

List\* ls;

if (el1 && (strlen(name)!=0)){

if (!el1->childs){

el1->childs = new List;

strcpy\_s(el1->childs->name, name);

}

else{

while ((ls = it(el1))->next!=nullptr);

ls->next = new List;

ls = ls->next;

strcpy\_s(ls->name, name);

}

}

RESTOREITS;

}

bool Graph::**Solve**()

{

SAVEITS;

bool res = 1;

Elem\* el;

List\* ls;

Elem\* el2;

while ((el = it())!=nullptr){

while ((ls = it(el))!=nullptr){

if (ls->node == nullptr){

el2 = FindElem(ls->name);

if (el2){

ls->node = el2;

ls->edge = new Edge(el->node, el2->node, ls);

widget->scene()->addItem(ls->edge);

}

else

res = 0;

}

}

}

RESTOREITS;

return res;

}

void Graph::**RemoveElem**(char \*name)

{

SAVEITS;

if (gr){

Elem\* el = gr;

bool res = strncmp(name, el->name, Numb) == 0;

if (!res){

while (((el = it())->next)!=nullptr){

res = strncmp(el->next->name, name, Numb) == 0;

if (res)

break;

}

if (res){

RemoveEdges(el->next);

widget->scene()->removeItem(el->next->node);

delete el->next->node;

Elem\* el2 = el->next->next;

delete el->next;

el->next = el2;

}

}

else{

RemoveEdges(gr);

widget->scene()->removeItem(gr->node);

delete gr->node;

Elem\* el2 = gr->next;

delete gr;

gr = el2;

}

}

RESTOREITS;

}

void Graph::**RemoveEdge**(Elem \*el1, Elem \*el2)

{

SAVEITS;

if (el1 && el2){

bool res = (el1->childs->node == el2);

if (!res){

List\* ls;

while ((ls = it(el1))->next != nullptr){

res = (ls->next->node == el2);

if (res)

break;

}

List\* ls2 = ls->next->next;

ls->next->edge->clear();

widget->scene()->removeItem(ls->next->edge);

delete ls->next->edge;

delete ls->next;

ls->next = ls2;

}

else{

List\* ls2 = el1->childs->next;

el1->childs->edge->clear();

widget->scene()->removeItem(el1->childs->edge);

delete el1->childs->edge;

delete el1->childs;

el1->childs = ls2;

}

}

RESTOREITS;

}

void Graph::**RenameElem**(char \*oldname, char \*newname)

{

Elem\* el = FindElem(oldname);

if ((!FindElem(newname)) && strlen(newname)!=0){

strcpy\_s(el->name, newname);

}

}

void Graph::**Desorientate**()

{

SAVEITS;

Elem\* el;

List\* ls;

while ((el = it())!=nullptr){

while ((ls = it(el))!=nullptr){

if (!Is\_Egde(ls->node, el))

AddEdge(ls->node, el);

}

}

RESTOREITS;

}

int Graph::**CountChildren**(Elem \*el, bool marked)

{

SAVEITS;

int i = 0;

List\* ls;

while ((ls = it(el))!=nullptr){

if ((marked) || !(ls->mark))

i++;

}

return i;

RESTOREITS;

}

int Graph::**CountElems**()

{

SAVEITS;

int i = 0;

while (it()!=nullptr)

i++;

return i;

RESTOREITS;

}

bool Graph::**Is\_Egde**(Elem \*el1, Elem \*el2)

{

List\* t\_lpos = KeepItL();

List\* curr;

bool res = 0;

while ((curr = it(el1))!=nullptr){

if (curr->node == el2){

res = 1;

break;

}

}

RestoreItL(t\_lpos);

return res;

}

List \*Graph::**GetEdge**(Elem \*el1, Elem \*el2)

{

if (Is\_Egde(el1, el2)){

SAVEITS;

List\* curr;

while ((curr = it(el1))!=nullptr){

if (curr->node == el2){

break;

}

}

RESTOREITS;

return curr;

}

return nullptr;

}

int Graph::**Max\_Width**()

{

SAVEITS;

int i = 0;

int l;

Elem\* el;

while ((el = it())!=nullptr){

if ((l=strlen(el->name)) > i)

i = l;

}

RESTOREITS;

return i;

}

void Graph::**Inc\_Matr**(QTextStream &os)

{

SAVEITS;

int len = Max\_Width();

Elem\* el;

int z = CountElems();

int i; int k;

os.setFieldWidth(len + 1);

os << " ";

while ((el = it())!=nullptr){

os << el->name;

}

os << endl;

for (i=0; i<z; i++){

os << this->operator [](i)->name;

for (k=0; k<z; k++)

os << Is\_Egde(this->operator [](i), this->operator [](k));

os << endl;

}

RESTOREITS;

}

bool Graph::**Euler**()

{

SAVEITS; //Сохранение и сброс итераторов

Elem\* v;

List\* u;

if (Stack.isEmpty()){ //Первый запуск

ResetEuler();

Stack.push(gr);

}

if (!Stack.isEmpty()){

v = Stack.top();

if (CountChildren(v, 0)){

if (!v0)

v0 = v;

u = it(v, 0); //Итератор через непомеченные ребра вершины

Stack.push(u->node);

v->node->update(); //Обновление картинки в графе

u->mark = 1; //Пометка ребра в одну сторону

if (Is\_Egde(u->node, v))

GetEdge(u->node, v)->mark = 1; //Пометка ребра в другую сторону

u->edge->update(); //Обновление картинки

v = u->node;

}

else{

if ((v0!=v) && (v0!=nullptr)){

QMessageBox msg; //Если зашли в тупик

msg.setText("В графе тупик. Эйлерова цикла нет");

msg.*exec*();

return 1;

}

else{

v0 = nullptr;

v = Stack.pop();

if (v)

v->node->update(); //Обновление картинки

SE.push(v);

}

}

if (!Stack.isEmpty())

Stack.top()->node->update();

}

RESTOREITS; //Восстановление итераторов

return 0;

}

void Graph::**ResetEuler**()

{

SE.clear();

Stack.clear();

v0 = nullptr;

}

void Graph::**ClearMarks**()

{

SAVEITS;

Elem\* el;

List\* ls;

while ((el = it())!=nullptr){

el->node->update();

while ((ls = it(el))!=nullptr){

ls->mark = 0;

ls->edge->update();

}

}

RESTOREITS;

}

Elem \*Graph::**operator**[](int i)

{

SAVEITS;

int k=0;

Elem\* el = nullptr;

while (k<=i){

el = it();

if (el!=nullptr)

k++;

else

break;

}

RESTOREITS;

return el;

}

Elem \*Graph::**KeepItE**()

{

Elem\* t\_pos = pos;

pos = nullptr;

return t\_pos;

}

List \*Graph::**KeepItL**()

{

List\* t\_lpos = lpos;

lpos = nullptr;

return t\_lpos;

}

void Graph::**RestoreItE**(Elem \*t\_pos)

{

pos = t\_pos;

}

void Graph::**RestoreItL**(List \*t\_lpos)

{

lpos = t\_lpos;

}

void Graph::**RemoveEdges**(Elem \*el)

{

SAVEITS;

Elem\* el1;

while ((el1 = it())!=nullptr){

if (Is\_Egde(el1, el))

RemoveEdge(el1, el);

if (Is\_Egde(el, el1))

RemoveEdge(el, el1);

}

RESTOREITS;

}

Приложение Е

Код GRAPHWIDGET.H

#ifndef GRAPHWIDGET\_H

#define GRAPHWIDGET\_H

#include <QGraphicsView>

#include <QContextMenuEvent>

#include <cstdlib>

#include <QRubberBand>

#include <QMouseEvent>

#include "node.h"

#include "edge.h"

#include "graph.h"

#include "mainwindow.h"

class Node;

class Graph;

class MainWindow;

//! [0]

class GraphWidget : public QGraphicsView

{

Q\_OBJECT

public:

GraphWidget(QWidget \*parent = 0, Graph\* graph = 0, MainWindow \*Main = 0);

Node \*centerNode;

void **itemMoved**();

MainWindow\* mw;

Graph\* gr;

public slots:

void **shuffle**();

void **zoomIn**();

void **zoomOut**();

void **SetGravity**(bool grav);

void **RandomColors**();

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

protected:

// void mouseDoubleClickEvent(QMouseEvent \*event) override;

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

//#if QT\_CONFIG(wheelevent)

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

//#endif

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

void **scaleView**(qreal scaleFactor);

private:

int timerId;

};

//! [0]

#endif // GRAPHWIDGET\_H

Приложение Ё

Код GRAPHWIDGET.CPP

#define Wid 800

#define Hei 600

#include "graphwidget.h"

#include "edge.h"

#include "node.h"

#include "scene.h"

#include <math.h>

#include <qDebug>

#include <QKeyEvent>

#include <QRect>

//! [0]

GraphWidget::**GraphWidget**(QWidget \*parent, Graph \*graph, MainWindow\* main)

: QGraphicsView(parent), timerId(0)

{

gr = graph;

mw = main;

GraphicsScene \*scene = new GraphicsScene(this, this);

scene->setItemIndexMethod(QGraphicsScene::NoIndex);

scene->setSceneRect(-400, -300, Wid, Hei);

setScene(scene);

setCacheMode(CacheBackground);

setViewportUpdateMode(BoundingRectViewportUpdate);

setRenderHint(QPainter::Antialiasing);

setTransformationAnchor(AnchorUnderMouse);

scale(qreal(0.8), qreal(0.8));

setMinimumSize(400, 400);

//! [0]

//! [1]

}

//! [1]

//! [2]

void GraphWidget::**itemMoved**()

{

if (!timerId)

timerId = startTimer(1000 / 25);

}

//! [2]

//! [3]

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;

case Qt::Key\_Delete:

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

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

if (node->isSelected())

gr->RemoveElem(node->name);

}

}

mw->update\_matr();

break;

default:

QGraphicsView::*keyPressEvent*(event);

}

}

//! [3]

//! [4]

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]

//#if QT\_CONFIG(wheelevent)

//! [5]

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

{

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

}

//! [5]

//#endif

//! [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::lightGray);

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

painter->setBrush(Qt::NoBrush);

painter->drawRect(sceneRect);

}

//! [6]

//! [7]

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);

}

//! [7]

void GraphWidget::**shuffle**()

{

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

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

item->setPos(-300 + rand()%600, -400 + rand()%800);

}

}

void GraphWidget::**zoomIn**()

{

scaleView(qreal(1.2));

}

void GraphWidget::**zoomOut**()

{

scaleView(1 / qreal(1.2));

}

void GraphWidget::**SetGravity**(bool grav)

{

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

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

node->gravity = grav;

}

*timerEvent*(0);

}

void GraphWidget::**RandomColors**()

{

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

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

QColor r(rand()%255, rand()%255, rand()%255);

node->grad0 = r;

node->grad1 = r.darker(300);

node->update();

}

}

}

Приложение Ж

Код scene.H

#ifndef SCENE\_H

#define SCENE\_H

#include "graphwidget.h"

#include <QGraphicsScene>

QT\_BEGIN\_NAMESPACE

class QGraphicsSceneMouseEvent;

QT\_END\_NAMESPACE

//! [0]

class GraphicsScene : public QGraphicsScene

{

Q\_OBJECT

public:

explicit GraphicsScene(QObject \*parent = 0, GraphWidget \*widg = 0);

protected:

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

private:

GraphWidget\* widget;

};

//! [0]

#endif // SCENE\_H

Приложение З

Код scene.cpp

#include "scene.h"

#include <QGraphicsSceneMouseEvent>

#include <qDebug>

#include <QPointF>

//! [0]

GraphicsScene::**GraphicsScene**(QObject \*parent, GraphWidget\* widg)

: QGraphicsScene(parent), widget(widg)

{

}

void GraphicsScene::***mouseDoubleClickEvent***(QGraphicsSceneMouseEvent \*event)

{

widget->gr->AddElem(widget->gr->GetMinStupidName());

widget->gr->FindElem(widget->gr->GetLastStupidName())->node->setPos(event->scenePos());

widget->mw->update\_matr();

}

Приложение и

Код edge.h

#ifndef EDGE\_H

#define EDGE\_H

#include <QGraphicsItem>

#include "graph.h"

class Node;

class List;

//! [0]

class Edge : public QGraphicsItem

{

public:

Edge(Node \*sourceNode, Node \*destNode, List\* lst);

void **clear**();

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

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

void **adjust**();

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:

Node \*source, \*dest;

List\* list;

QPointF sourcePoint;

QPointF destPoint;

qreal arrowSize;

};

//! [0]

#endif // EDGE\_H

Приложение й

Код edge.cpp

#include "edge.h"

#include "node.h"

#include <qmath.h>

#include <QPainter>

//! [0]

Edge::**Edge**(Node \*sourceNode, Node \*destNode, List\* lst)

: arrowSize(10)

{

list = lst;

setAcceptedMouseButtons(0);

source = sourceNode;

dest = destNode;

source->addEdge(this);

dest->addEdge(this);

adjust();

}

void Edge::**clear**()

{

sourceNode()->edgeList.removeOne(this);

destNode()->edgeList.removeOne(this);

}

//! [0]

//! [1]

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

{

return source;

}

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

{

return dest;

}

//! [1]

//! [2]

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() \* 15) / length, (line.dy() \* 15) / length);

sourcePoint = line.p1() + edgeOffset;

destPoint = line.p2() - edgeOffset;

} else {

sourcePoint = destPoint = line.p1();

}

}

//! [2]

//! [3]

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);

}

//! [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]

// Draw the line itself

QColor col;

if (list->mark)

col = Qt::red;

else

col = Qt::black;

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

painter->drawLine(line);

//! [5]

//! [6]

// Draw the arrows

double angle = std::atan2(-line.dy(), line.dx());

/\*

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

cos(angle + M\_PI / 3) \* arrowSize);

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

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

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

cos(angle - M\_PI / 3) \* arrowSize);

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

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

painter->setBrush(col);

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

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

}

//! [6]

Приложение к

Код node.h

#ifndef NODE\_H

#define NODE\_H

#include <QGraphicsItem>

#include <QList>

#include <QContextMenuEvent>

#include <QColor>

class Edge;

class GraphWidget;

QT\_BEGIN\_NAMESPACE

class QGraphicsSceneMouseEvent;

QT\_END\_NAMESPACE

//! [0]

class Node : public QGraphicsItem

{

// Q\_OBJECT

public:

Node(GraphWidget \*graphWidget);

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;

bool gravity = false;

char\* name;

QColor grad0 = Qt::white;

QColor grad1 = Qt::gray;

QList<Edge \*> edgeList;

void ***contextMenuEvent***(QGraphicsSceneContextMenuEvent\* event) override;

protected:

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

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

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

private:

QPointF newPos;

GraphWidget \*graph;

};

//! [0]

Приложение л

Код node.CPP

#include "edge.h"

#include "node.h"

#include "graphwidget.h"

#include <qmath.h>

#include <memory>

#include <QGraphicsScene>

#include <QGraphicsSceneMouseEvent>

#include <QPainter>

#include <QStyleOption>

#include <QColorDialog>

#include <QDebug>

#include <QMenu>

#include <QInputDialog>

#include <QMessageBox>

#include <QByteArray>

#include <QString>

//! [0]

Node::**Node**(GraphWidget \*graphWidget)

: graph(graphWidget)

{

setFlag(ItemIsMovable);

setFlag(ItemIsSelectable);

setFlag(ItemSendsGeometryChanges);

setCacheMode(DeviceCoordinateCache);

setZValue(-1);

}

//! [0]

//! [1]

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

{

edgeList << edge;

edge->adjust();

}

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 = 0.5 \* (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]

if (!gravity){

xvel = yvel = 0;

}

else if (qAbs(xvel) < 0.1 && qAbs(yvel) < 0.1)

xvel = yvel = 0;

//! [5]

//! [6]

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

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

newPos.setX(qMin(qMax(newPos.x(), sceneRect.left() + 10), sceneRect.right() - 10));

newPos.setY(qMin(qMax(newPos.y(), sceneRect.top() + 10), sceneRect.bottom() - 10));

}

//! [6]

//! [7]

bool Node::**advance**()

{

if (newPos == pos())

return false;

setPos(newPos);

return true;

}

//! [7]

//! [8]

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

{

qreal adjust = 2;

return QRectF( -15 - adjust, -15 - adjust, 33 + adjust, 33 + adjust);

}

//! [8]

//! [9]

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

{

QPainterPath path;

path.addEllipse(-15, -15, 30, 30);

return path;

}

//! [9]

//! [10]

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

{

painter->setPen(Qt::NoPen);

painter->setBrush(Qt::darkGray);

painter->drawEllipse(-12, -12, 30, 30);

QColor grad0t = grad0;

QColor grad1t = grad1;

QRadialGradient gradient(-5, -5, 15);

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

grad1t = grad0.light(120);

grad0t = grad1.light(120);

} else if (option->state & QStyle::State\_Selected){

grad0t = Qt::blue;

grad1t = Qt::darkBlue;

}

if ((!graph->gr->Stack.isEmpty()) && (graph->gr->Stack.top()->node == this)){

grad0t = Qt::red;

grad1t = Qt::darkRed;

}

gradient.setColorAt(0, grad0t);

gradient.setColorAt(1, grad1t);

painter->setBrush(gradient);

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

painter->drawEllipse(-15, -15, 30, 30);

QString temp(name);

QFont font = painter->font();

font.setBold(true);

int fs = 16 - 5.5\*log(temp.length());

font.setPointSize(fs);

painter->setFont(font);

QColor col(255 - grad0t.red(), 255 - grad0t.green(), 255 - grad0t.blue());

painter->setPen(QPen(col, 0));

painter->drawText(0-temp.length()\*fs/2.5, 0+fs/2, temp);

}

//! [10]

//! [11]

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

{

switch (change) {

case ItemPositionHasChanged:

foreach (Edge \*edge, edgeList)

edge->adjust();

graph->itemMoved();

break;

default:

break;

};

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

}

void Node::***contextMenuEvent***(QGraphicsSceneContextMenuEvent \*event)

{

std::unique\_ptr<QMenu> menu(new QMenu(graph));

QAction\* colorAction = menu->addAction("Изменить цвет");

QAction\* renameAction = menu->addAction("Переименовать");

QAction\* linkAction = menu->addAction("Связать с этим");

QAction\* freeAction = menu->addAction("Удалить связи");

QAction\* deleteAction = menu->addAction("Удалить");

QAction\* selectedAction = menu->exec(event->screenPos());

if (selectedAction == colorAction){

QColorDialog dia;

dia.*exec*();

grad0 = dia.currentColor();

grad1 = grad0.darker(300);

update();

}

else if(selectedAction == renameAction){

bool ok;

QString name2 = QInputDialog::getText(graph, "Добавить связь", "Введите имя", QLineEdit::Normal, "", &ok);

QByteArray arr2 = name2.toLocal8Bit();

char\* cname2 = arr2.data();

if (graph->gr->FindElem(cname2)){

QMessageBox msg;

msg.setText("Элемент с таким именем уже существует");

msg.*exec*();

}

else

graph->gr->RenameElem(name, cname2);

update();

}

else if (selectedAction == deleteAction){

graph->gr->RemoveElem(name);

}

else if (selectedAction == freeAction){

graph->gr->RemoveEdges(graph->gr->FindElem(name));

}

else if (selectedAction == linkAction){

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

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

if (node->isSelected())

graph->gr->AddEdge(graph->gr->FindElem(node->name), graph->gr->FindElem(name));

}

}

}

graph->mw->update\_matr();

}

//! [11]

//! [12]

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

{

update();

QGraphicsItem::*mousePressEvent*(event);

}

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

{

update();

QGraphicsItem::*mouseReleaseEvent*(event);

}

//! [12]