**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: Алгоритмы с возвратом. Поиск Гамильтоновых путей в графе

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Пискунов Я.А. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2018

**Цель работы.**

Изучить реализацию алгоритма поиска всех Гамильтоновых циклов в графе на языке программирования C++. Реализовать возможность пошагового выполнения алгоритма с отслеживаем бэктрекинга.

**Основные теоретические положения.**

Гамильтоновым путём (англ. Hamiltonian path) называется простой путь, проходящий через каждую вершину графа ровно один раз.

Гамильтоновым циклом (англ. Hamiltonian cycle) называют замкнутый гамильтонов путь.

Граф называется полугамильтоновым (англ. Semihamiltonian graph), если он содержит гамильтонов путь.

**Ход работы**

За основу алгоритма поиска гамильтоновых путей возьмем алгоритм на языке Pascal, представленный в книге В. Липского «Комбинаторика для программистов». Этот алгоритм представлен на рис. 0.

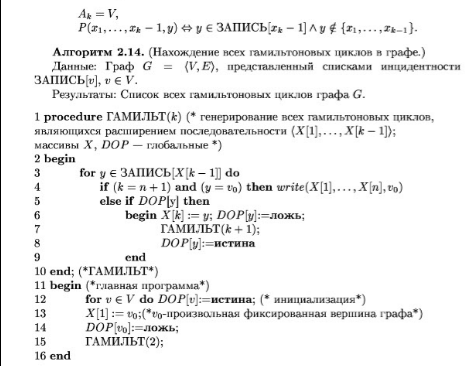


Рисунок 0 – алгоритм из книги

Этот алгоритм подходит для неориентированного графа. Модифицируем его для применения на ориентированном графе.

Работать будем в Qt Creator, так как данная программа предоставляет широкий функционал, который значительно упростит создание программы. В дальнейшем будут рассмотрены файлы, созданные непосредственно «руками». Все автоматически созданные файлы, а также файлы, которые необходимы для графического отображения результата, но не для обработки алгоритма, в данный отчет не включены ввиду их громоздкости и отсутствия критической необходимости в их описании.

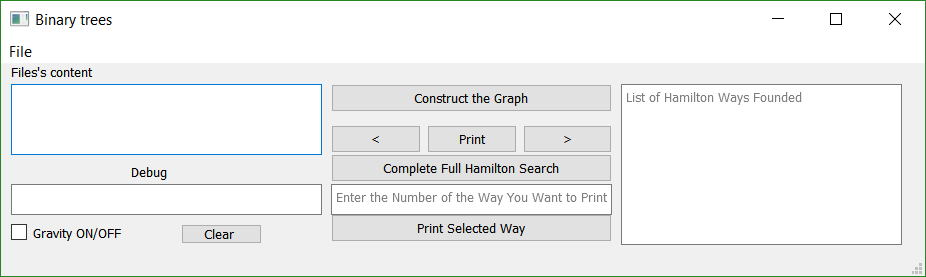
Первым делом создадим интерфейс программы для работы с пользователем. Скриншот интерфейса представлен на рис. 1.

Рисунок 1 – Интерфейс программы

Интерфейс содержит текстовое поле, в котором будет отображаться содержимое читаемого файла (список инцидентности вершин графа); поле «Debug» с кнопкой для его очищения, предназначенное для вывода различных сообщений об ошибках и прочих подобных сообщений; переключатель гравитации, включающий и отключающий гравитацию на изображении графа; текстовое поле для вывода найденных циклов; кнопки, запускающие построение графа, поиск гамильтоновых путей, и печать выбранного пути, если пользователю нет необходимости пошагово повторять процесс; кнопки навигации по шагам процесса поиска с возможностью печати каждого шага.

Первая задача программы – чтение данных из файла. Оно реализовано с использованием стандартной библиотеки Qt «QFileDialog». Код, который активируется по нажатию на кнопку «Open» представлен в приложении А.

После чтения данных из файла необходимо их интерпретировать и проверить на корректность. Для этого используются функции «ReadFile», «Analyser» и «Corrector». Код данных функций представлен в приложении Б.

После анализа данных они интерпретируются как список инцидентности. Вся обработка данного списка основывается на массиве структур INCTR. Каждый элемент массива хранит информацию о имени узла, степени, количестве входящих и выходящих ребер графа, номер вершины в списке инцидентности, а также указатели на предыдущий и последующий элементы списка инцидентности данной вершины. Кроме этого, структуры содержат ссылку на элемент типа node, необходимый для рисования этих объектов. Код, отражающий формат структур INCTR представлен в приложении В.

Программа имеет возможность отрисовки имеющегося графа. В таком случае отключается функционал рисования циклов. Код, запускаемый при этом, показан в приложении Г.

Основная задача программы – поиск всех Гамильтоновых путей в графе. Для этого необходима отдельная функция, анализирующая граф, а также преобразующая его для отображения циклов. Для начал необходим проверить, является ли граф гамильтоновым или полугамильтоновым. Для этого используется функция, представленная в приложении Д. В случае несоответствия графа условию выводится сообщение об ошибке. Далее пользователь имеет возможность запустить автоматический поиск Гамильтоновых путей. Все гамильтоновы пути сохраняются в двумерный массив. В результате работы алгоритма в текстовое окно выводится информация обо всех гамильтоновых путях, найденных в графе. Для удобства восприятия и краткости записи в качестве идентификатора вершин используется их порядковый номер в списке инцидентности. Код, запускаемых для поиска функций представлен в приложении Е.

Любой из найденных путей можно отобразить. Для этого используется код, представленный в приложении Ж.

В алгоритм поиска также добавлен функционал отслеживания каждого шага алгоритма. Для этого используется очередь, реализованная в стандартной библиотеке Qt «QQueue», что значительно упрощает реализацию алгоритма. В данную очередь записываются состояния массива гамильтоновых путей при каждом изменении этого самого массива. Переменная очереди глобальна, так что данные о состояниях массива можно извлекать из очереди, таким образом проходя алгоритм шаг за шагом. Для возможности возврата назад взятые из очереди элементы не удаляются, а сохраняются в стэк, также реализованный на основе Qt библиотеки «QStack». Таким, образом имеется текущий элемент, не принадлежащий очереди состояний или стэку уже просмотренных состояний. Описываемое этим элементом состояние можно в любой момент времени вывести на печать. Ребра, входящие в гамильтонов путь, отмечаются зеленым цветом. Код, отвечающий за функцию пошаговой обработки алгоритма представлен в приложении З.

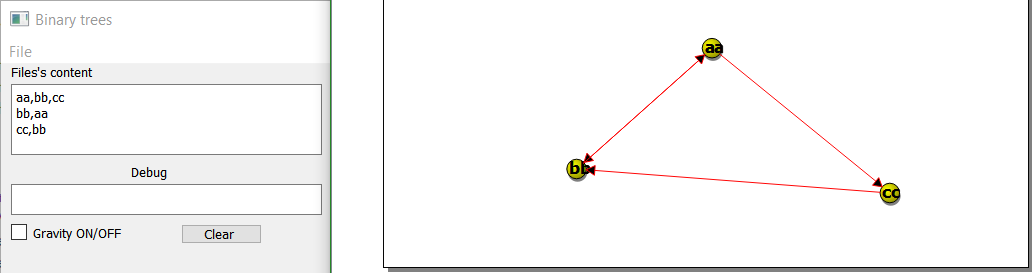
Теперь необходимо проверить работоспособность программы. Для начала попробуем просто построить граф. Исходные данные и результат представлены на рис. 2.

Рисунок 2 – построение графа

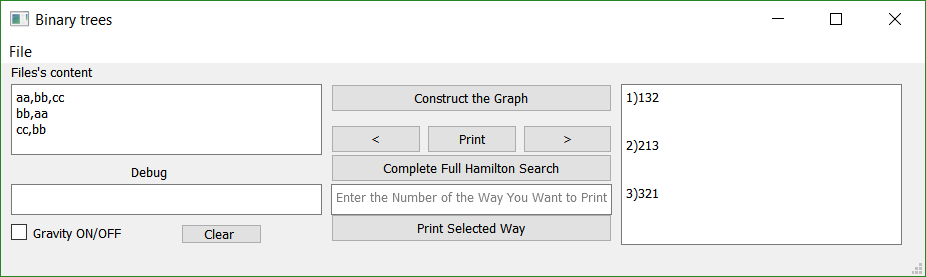
Все тесты прошли успешно. Теперь попробуем найти гамильтоновы пути в этом графе. Результат выполнения представлен на рис. 3.

Рисунок 3 – поиск гамильтоновых путей

После нахождения путей открывается возможность их вывода на экран. В данном графе было найдено 3 гамильтоновых пути. Они представлены на рис. 4, рис. 5 и рис. 6.

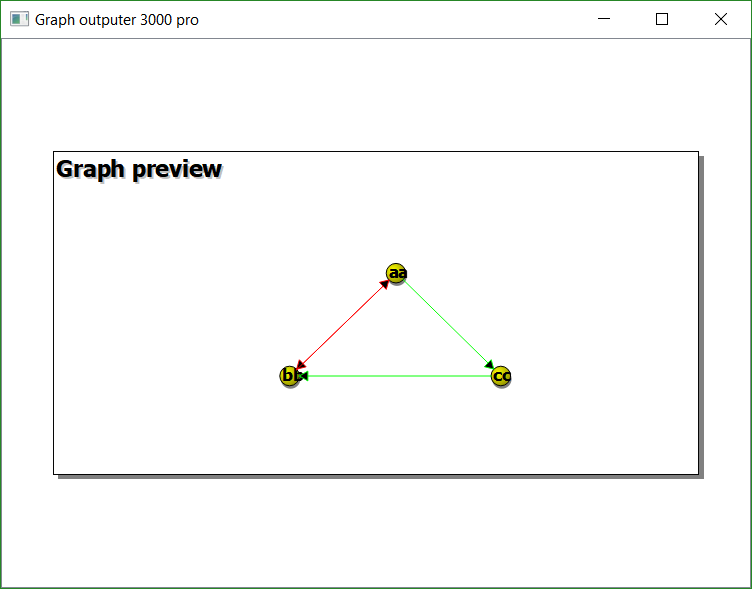


Рисунок 4 – первый гамильтонов путь

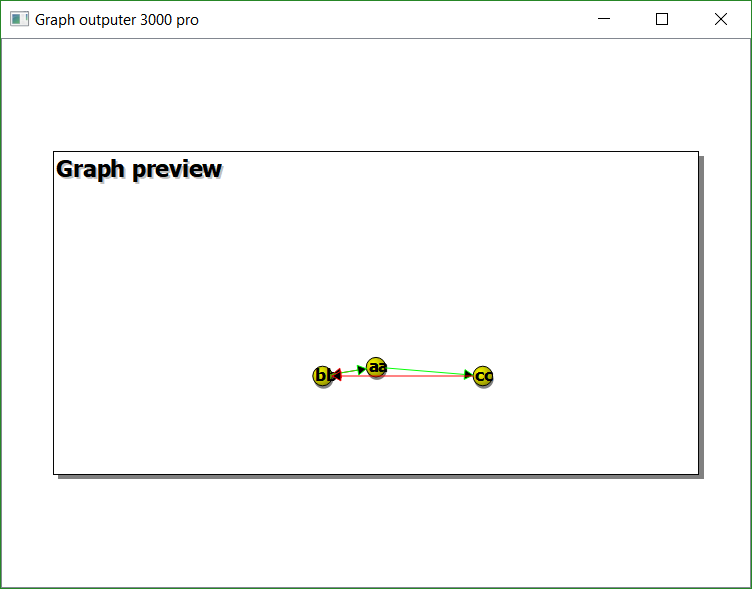


Рисунок 5 – второй гамильтонов путь

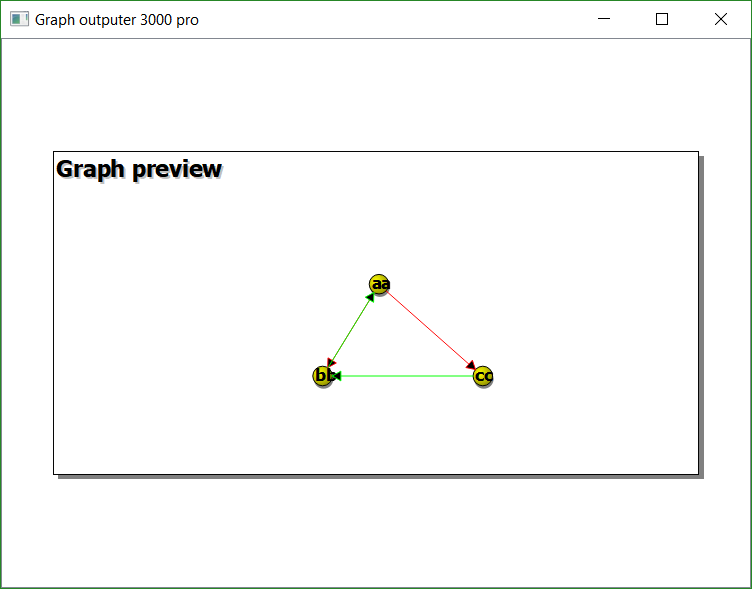


Рисунок 6 – третий гамильтонов путь

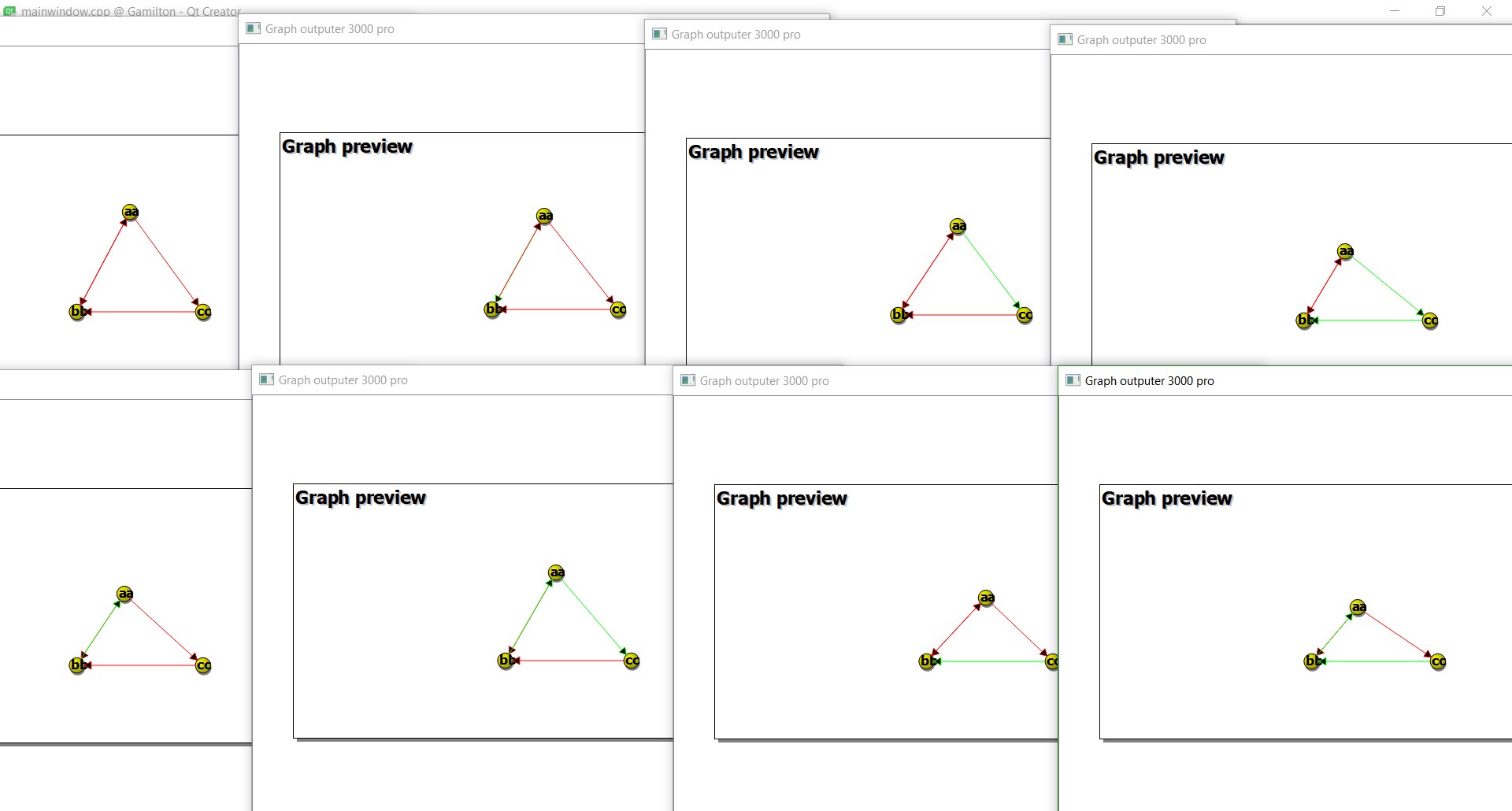
Аналогичным образом проверим пошаговое выполнение алгоритма. Комплексный снимок результатов на каждом шаге представлен на рис. 7.

Рисунок 7 – пошаговое выполнение программы

Также были проверены многочисленные случае пользовательских ошибок, таких как отсутствие в файле информации о графе, неверный выбор номера пути для печати и тому подобные. При всех таких случаях программа выдавала соответствующее сообщение об ошибке и продолжала нормально функционировать в ожидании следующих команд.

**Выводы.**

В ходе данной работы была изучена реализация алгоритма поиска гамильтоновых путей и циклов в графе, выявлены возможности пошаговой реализации рекурсивных алгоритмов а также изучена работа с классами QStack и QQueue.

Приложение А

Код для открытия файла

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

{

\_cycle = 0;

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

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

**}**

Приложение Б

Интерпретация и проверка данных на корректность

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;

}

}

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;

}

}

}

}

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 NULL;

}

int i = 0;

QTextStream file1s(&file1);

QString temp = file1s.readAll();

te->setPlainText(temp);

peaks\_i=analyser(temp, i);

if(peaks\_i==-1)

return NULL;

temp=corrector(temp);

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

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

return peaks;

}

Приложение В

Структура INCTR

typedef struct INCTR

{

struct INCTR\* prev=NULL;

elem name;

int deg=0;

int intdeg=0;

int outdeg=0;

int turn = 0;

struct INCTR\* next=NULL;

Node\* node;

}INCTR;

Приложение Г

Простое рисование графа

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

{

if(tree==NULL)

{

ui->textEdit\_3->setText("Error 404: no info for graph construction!");

return;

}

else

{

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

QMainWindow\* temp = new QMainWindow;

temp->setCentralWidget(widget);

temp->setWindowTitle("Graph outputer 3000 pro");

temp->show();

}

}

Приложение Д

Проверка графа на корректность

bool **TeoremCheck** (INCTR\* peaks, int size)

{

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

if (peaks[i].intdeg<size/2 || peaks[i].outdeg<size/2)

return false;

return true;

}

приложение Е

Функции поиска гамильтоновых путей

int\*\* MainWindow::**FindGamilton** (INCTR\* grandparent, INCTR\* parent, int peaks, int step, int cycle, int\*\* gampath)

{

bool flag = false;

INCTR\* temp = new INCTR;

temp = parent;

while (temp->next!=NULL)

{

temp = temp->next;

flag = false;

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

{

if (step!=peaks)

{

if (temp->turn==tree[i].turn)

{

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

{

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

{

if (temp->next==NULL)

{

gampath[0][peaks+1] = cycle;

gampath[0][peaks+2] = step-1;

return gampath;

}

else

{

flag = true;

}

}

}

if (!flag)

{

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

PTP vazn;

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

vazn.sequen.enqueue(gampath[cycle][z]);

steps.enqueue(vazn);

step++;

gampath = FindGamilton(grandparent, &tree[i], peaks, step, cycle, gampath);

step = gampath[0][peaks+2];

cycle = gampath[0][peaks+1];

}

}

}

else

{

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

{

cycle++;

gampath[cycle] = new int [peaks+10];

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

gampath[cycle][j] = gampath [cycle-1][j];

step--;

gampath[cycle-1][peaks]=-1;

gampath[0][peaks+1] = cycle;

gampath[0][peaks+2] = step;

return gampath;

}

else

{

while (temp->next!=NULL)

{

temp = temp->next;

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

{

cycle++;

gampath[cycle] = new int[peaks+10];

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

gampath[cycle][j]=gampath[cycle-1][j];

step--;

gampath[cycle-1][peaks] = -1;

gampath[0][peaks+1] = cycle;

gampath[0][peaks+2] = step;

return gampath;

}

}

step--;

gampath[0][peaks+1] = cycle;

gampath[0][peaks+2] = step;

return gampath;

}

}

}

}

gampath[0][peaks+2] = step-1;

return gampath;

}

void MainWindow::**on\_pushButton\_3\_clicked**()

{

if (tree!=NULL)

{

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

int step = 1;

int cycle = 0;

int i;

steps.clear();

complete.clear();

int\*\* Gamilton\_Path = new int\*[100];

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

{

Gamilton\_Path[i] = NULL;

Gamilton\_Path[i] = new int[100];

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

Gamilton\_Path[i][o] = 0;

}

if (!TeoremCheck(tree, peaks\_i))

{

ui->textEdit\_3->setPlainText("Error 403: Forbinden (Wrong Graph)");

return;

}

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

{

Gamilton\_Path[cycle][0] = tree[i].turn;

Gamilton\_Path = FindGamilton (&tree[i], &tree[i], peaks\_i, step, cycle, Gamilton\_Path);

while (Gamilton\_Path[cycle] && Gamilton\_Path[cycle][peaks\_i]==-1)

cycle++;

}

Gamilton\_Path[cycle] = NULL;

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

{

ui->textEdit\_4->append(QString::number(i+1)+')');

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

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

if (Gamilton\_Path[i][j]==tree[k].turn)

ui->textEdit\_4->insertPlainText(QString::number(tree[k].turn));

ui->textEdit\_4->append((QString)'\n');

}

\_cycle = cycle;

\_Gamilton\_Path = Gamilton\_Path;

}

else ui->textEdit\_3->append("Error 404: No info for search");

}

Приложение з

Вывод выбранного пути на экран

void MainWindow::**on\_pushButton\_4\_clicked**()

{

if (ui->textEdit\_2->toPlainText()!="")

{

int chosenCycle = ui->textEdit\_2->toPlainText().toInt();

if (chosenCycle<=\_cycle)

{

GraphWidget\* widget = new GraphWidget(this, tree, &gravity, peaks\_i , \_Gamilton\_Path[chosenCycle-1]);

QMainWindow\* temp = new QMainWindow;

temp->setCentralWidget(widget);

temp->setWindowTitle("Graph outputer 3000 pro");

temp->show();

}

else

{

ui->textEdit\_3->append("Error 404: This cycle was not found");

return;

}

}

}

Приложение З

Пошаговое выполнение алгоритма

void MainWindow::**on\_pushButton\_7\_clicked**()

{

if ((!steps.isEmpty())||(!complete.isEmpty()))

{

int i = 0;

int\* tempor = new int;

while (!current.sequen.isEmpty())

tempor[i++] = current.sequen.dequeue();

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

QMainWindow\* temp = new QMainWindow;

temp->setCentralWidget(widget);

temp->setWindowTitle("Graph outputer 3000 pro");

temp->show();

}

else ui->textEdit\_3->append("Error 404: Full search wasn't complete yet");

}

void MainWindow::**on\_pushButton\_6\_clicked**()

{

if (!steps.isEmpty())

{

current = steps.dequeue();

complete.push(current);

ui->textEdit\_3->append("One more step right");

}

else ui->textEdit\_3->append("Error 404: All steps proceeded or search wasn't started.");

}

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

{

if (!complete.isEmpty())

{

current = complete.pop();

steps.enqueue(current);

ui->textEdit\_3->append("One more step left");

}

else ui->textEdit\_3->append("Error 404: No steps proceeded or search wasn't started.");

}