**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

Курсовая РАБОТА

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

Тема: Рандомизированные деревья поиска

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

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Виноградов К. А. | | |
| Группа 6304 | | |
| Тема работы Рандомизированные деревья поиска | | |
| Исходные данные:  Случайные БДП с рандомизацией - демонстрация | | |
| Содержание пояснительной записки:  Перечисляются требуемые разделы пояснительной записки (обязательны разделы «Содержание», «Введение», «Заключение», «Список использованных источников») | | |
| Предполагаемый объем пояснительной записки:  Не менее 00 страниц. | | |
| Дата выдачи задания: 30.11.2017 | | |
| Дата сдачи реферата: \_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| Дата защиты реферата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| Студент |  | Виноградов К. А. |
| Преподаватель |  | Балтрашевич В.Э. |

**Аннотация**

В данной работе рассмотрена реализация рандомизированного дерева поиска на языке C++. Программе на вход подается последовательность слов и ключей. К этой последовательности применяется алгоритм построения дерева который на выходе дает граф рандомизированного бинарного дерева. К дереву возможно применить алгоритмы поиска элементов, их добавления и удаления.**содержаниЕ**

[ВВЕДЕНИЕ 5](#_Toc501613382)

[1. ОПЕРАЦИИ С БИНАРНЫМ ДЕРЕВОМ 6](#_Toc501613383)

[1.1. Структура дерева. Базовые операции с бинарным деревом 6](#_Toc501613384)

[1.2. Поиск элемента 7](#_Toc501613385)

[1.3. Добавление элемента 7](#_Toc501613386)

[1.4. Вычисление весов 9](#_Toc501613387)

[1.5. Удаление элементов 9](#_Toc501613388)

[ИНТЕРФЕЙС ПРОГРАММЫ 11](#_Toc501613389)

[1.6. Графический интерфейс 11](#_Toc501613390)

[1.7. Общие переменные 11](#_Toc501613391)

[2. ГРАФИЧЕСКОЕ ОТОБРАЖЕНИЕ ДЕРЕВА 12](#_Toc501613392)

[3. ТЕСТИРОВАНИЕ 13](#_Toc501613393)

[3.1. Тестирование на примере простой последовательности 13](#_Toc501613394)

[3.2. Тестирования на большом объеме данных 15](#_Toc501613395)

[ПРИЛОЖЕНИЕ А. КОД BINTREE.H 18](#_Toc501613396)

[ПРИЛОЖЕНИЕ Б. КОД BINTREE.CPP 19](#_Toc501613397)

[ПРИЛОЖЕНИЕ В. КОД MAINWINDOW.CPP 23](#_Toc501613398)

[ПРИЛОЖЕНИЕ Г. КОД NODE.H 27](#_Toc501613399)

[ПРИЛОЖЕНИЕ Д. КОД NODE.CPP 28](#_Toc501613400)

[ПРИЛОЖЕНИЕ Е. КОД EDGE.H 31](#_Toc501613401)

[ПРИЛОЖЕНИЕ Ж. КОД EDGE.CPP 32](#_Toc501613402)

[ПРИЛОЖЕНИЕ З. КОД GRAPHWIDGET.H 34](#_Toc501613403)

[ПРИЛОЖЕНИЕ И. КОД GRAPHWIDGET.CPP 35](#_Toc501613404)

# ВВЕДЕНИЕ

Цель данной работы – написание алгоритма создания бинарного дерева поиска с рандомизацией и функций для работы с ним, а конкретно:

1. Обеспечить интерфейс работы с бинарным деревом: базовые операции, обмен элементов. .
2. Реализовать операции с бинарным деревом, применяющиеся для бинарных деревьев поиска: удаление элементов, вставка элементов, поиск по ключу.
3. Реализовать интерфейс, обеспечивающий удобную работу с программой.
4. Реализовать графический вывод бинарного дерева поиска.

После этого рационально будет провести проверку программы и посчитать эффективность работы.

# ОПЕРАЦИИ С БИНАРНЫМ ДЕРЕВОМ

## Структура дерева. Базовые операции с бинарным деревом

Все прототипы и коды функций из этого пункта находятся в приложениях А и Б. Структура бинарного дерева в данной работе выглядит следующим образом:

typedef QString elem;

typedef struct BT{

struct BT\* Root;

struct BT\* Left;

struct BT\* Right;

elem info;

int key;

int size;

Node\* node;

BT(int k, BT\* root, elem inf) {key=k; Root=root; info=inf; Left=Right=nullptr; size=1;}

}BT;

Далее необходимо запрограммировать базовые операции:

BT\* **RootBT**(BT\* tree);

BT\* **LeftBT**(BT\* tree);

BT\* **RightBT**(BT\* tree);

bool NullBT(BT\* tree);

BT \***delete\_tree**(BT\* tree);

Их реализация не представляет сложности:

BT\* **RootBT**(BT\* tree){

return tree->Root;

}

BT\* **LeftBT**(BT\* tree){

return tree->Left;

}

BT\* **RightBT**(BT\* tree){

return tree->Right;

}

bool **NullBT**(BT\* tree){

if (tree == nullptr){

return 1;

}

return 0;

}

BT\* **delete\_tree**(BT\* tree)

{

if(!tree)

return tree;

if(tree->Left)

delete\_tree(tree->Left);

if(tree->Right)

delete\_tree(tree->Right);

tree=nullptr;

delete tree;

return tree;

}

## Поиск элемента

Основная задача деревьев поиска ­– поиск элементов:

BT\* **Find**(BT\* tree, int key);

Реализация функции:

BT\* **Find**(BT\* tree, int key)

{

if(!tree)

return nullptr;

if(key==tree->key)

return tree;

if(key<tree->key)

return Find(tree->Left, key);

if(key>tree->key)

return Find(tree->Right, key);

}

Данная функция получает на вход значение элемента и дерево. Если текущий элемент – не искомый, функция применяет сама себя к правому и левому поддереву, если это возможно. При этом, если элемент найден, то функция возвратит адрес элемента, если же элемент не найден, то возвращенный адрес будет нулевым.

## Добавление элемента

Для добавления элементов и для конструирования всего дерева используются следующие функции:

BT\* **rotateright**(BT\* tree)

{

BT\* child=tree->Left;

if(!child)

return tree;

tree->Left=child->Right;

child->Root=tree->Root;

tree->Root=child;

child->Right=tree;

child->size=tree->size;

fixsize(tree);

return child;

}

BT\* **rotateleft**(BT\* tree)

{

BT\* child=tree->Right;

if(!child)

return tree;

tree->Right=child->Left;

child->Root=tree->Root;

tree->Root=child;

child->Left=tree;

child->size=tree->size;

fixsize(tree);

return child;

}

BT\* **insertroot**(BT\* tree, BT \*root, int key, QString key\_ch)

{

if(!tree)

{

return new BT(key, root, key\_ch);

}

if(key<tree->key)

{

tree->Left=insertroot(tree->Left, tree, key, key\_ch);

return rotateright(tree);

}

else

{

tree->Right=insertroot(tree->Right, tree, key, key\_ch);

return rotateleft(tree);

}

}

BT\* **insertrand**(BT\* tree, BT \*root, int key, QString key\_ch)

{

QChar conv;

if(!tree)

{

return new BT(key, root, key\_ch);

}

if(rand()%(tree->size+1)==0)

return tree=insertroot(tree, tree->Root, key, key\_ch);

if(tree->key>key)

{

tree->Left=insertrand(tree->Left, tree, key, key\_ch);

}

else

{

tree->Right=insertrand(tree->Right, tree, key, key\_ch);

}

fixsize(tree);

return tree;

}

В дереве используется обычное построение и построение путем вставки в корень. Для обычного используется функция:

BT\* **insertroot**(BT\* tree, BT\* root, int key, QString key\_ch);

а для вставочного целый набор:

BT\* **rotateright**(BT\* tree);

BT\* **rotateleft**(BT\* tree);

BT\* **insertrand**(BT\* tree, BT\* root, int key, QString key\_ch);

две верхние функции обеспечивают правый и левый поворот вокруг узла соответственно.

## Вычисление весов

Для вычисления весов отведена отдельная переменная size в структуре а за их корректировку отвечают функции:

int getsize(BT\* tree);

int fixsize(BT\* tree);

int **getsize**(BT\* tree)

{

if(!tree)

return 0;

return tree->size;

}

int **fixsize**(BT\* tree)

{

tree->size=getsize(tree->Left)+getsize(tree->Right)+1;

}

Данная функция рекурсивная: вес каждого элемента – сумма весов его сыновей плюс свой собственный. Таким образом, для каждого элемента функция применяет саму себя к своим поддеревьям, пока не дойдет до листьев. Из листьев сложатся веса дерева.

## Удаление элементов

Удаление элементов происходит с помощью функции remove а восстановление дерева с помощью функции merge:

BT\* **merge**(BT\* tree1, BT\* tree2);

BT\* **remove**(BT\* tree, int key);

BT\* **merge**(BT\* tree1, BT\* tree2)

{

if(!tree1)

return tree2;

if(!tree2)

return tree1;

if(rand()%(tree1->size+tree2->size) < tree1->size )

{

tree1->Right = merge(tree1->Right,tree2);

tree1->Root=tree1->Root->Root;

fixsize(tree1);

return tree1;

}

else

{

tree2->Left = merge(tree1,tree2->Left);

tree2->Root=tree2->Root->Root;

fixsize(tree2);

return tree2;

}

}

BT\* **remove**(BT\* tree, int key)

{

if( !tree )

return tree;

if( tree->key==key )

{

BT\* temp = merge(tree->Left,tree->Right);

delete tree;

return temp;

}

else if( key<tree->key )

tree->Left = remove(tree->Left,key);

else

tree->Right = remove(tree->Right,key);

return tree;

}

# ИНТЕРФЕЙС ПРОГРАММЫ

## Графический интерфейс

Для удобной работы с программой необходимо сделать некоторый графический интерфейс.

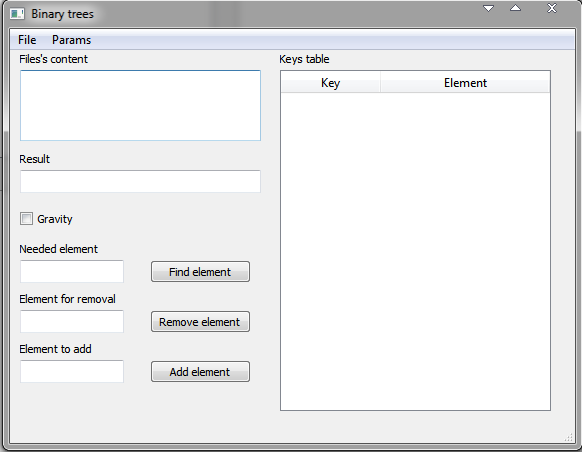


Рисунок 1 – главное окно программы

## Общие переменные

Основная часть программы, взаимодействующая с пользователем, определяется файлами mainwindow.h, mainwindow.cpp (код последнего – в приложении В) Соответственно, именно там необходимо расположить некоторые общие переменные.

BT\* tree;

bool gravity = false;

# ГРАФИЧЕСКОЕ ОТОБРАЖЕНИЕ ДЕРЕВА

Для лучшего понимания работы алгоритма и для контроля за каждым шагом представляется разумным осуществить вывод бинарного дерева кодирования в отдельно окно. К счастью, в Qt есть встроенный пример – Qt Elastic Nodes, реализующий сходную задачу вывода графа.

Реализация достаточно громоздка, поэтому представляется разумным ограничиться перечислением изменений, внесенных в пример:

* В класс Node (файл node.h, код в приложении Г) добавлен параметр, регулирующий гравитацию.
* В файл с функциями для Node добавлен обработчик параметра (файл node.cpp, код в приложении Д).
* В том же файле размеры эллипсов подстраиваются под размер элемента.
* В класс Edge – параметр, регулирующий видимость стрелки (edge.h, код в приложении Е)
* В файл с функциями для Edge при конструкторе добавлен обработчик параметра (edge.cpp, код в приложении Ж)
* Переделан конструктор для GraphWidget (graphwidget.h, graphwidget.cpp, коды в приложениях З и И), в класс GraphWidget добавлена рекурсивная процедура Build, переводящая бинарное дерево методом прямого обхода в совокупность элементов Node и Edge (типы QGraphicsItem), содержащихся в QGraphicsScene.

# ТЕСТИРОВАНИЕ

## Тестирование на примере простой последовательности

Проведем тестирование программы. Сначала протестируем функции удаления.

Таблица 1. Тестирование удаления

|  |  |  |
| --- | --- | --- |
| Шаг | Вывод | Дерево |
| 0 |  |  |
| 1  (-6) |  |  |
| 2  (-5) |  |  |
| 3  (-1) |  |  |
| 4  (-7) |  |  |
| 5  (-8) |  |  |

Теперь проверим добавление новых значений

Таблица 2. Тестирование добавления элементов

|  |  |  |
| --- | --- | --- |
| Шаг | Вывод | Дерево |
| 0  (+11) |  |  |
| 1  (+12) |  |  |
| 2  (+13) |  |  |

## Тестирования на большом объеме данных

Финальное построение проведено на примерно 250 узлах. Большего количества узлов не позволяет показать графическое оболочка графа.

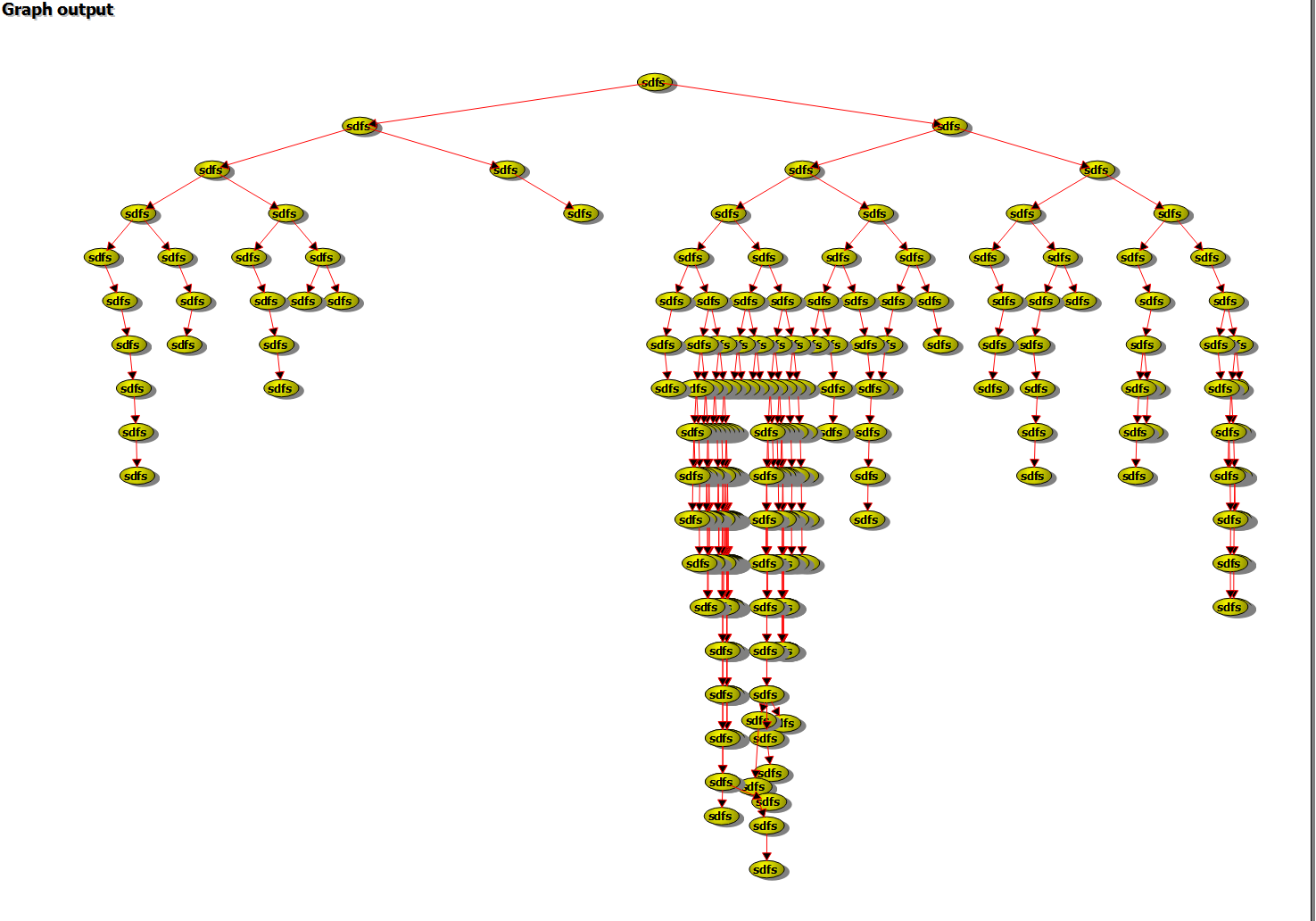


Рисунок 5 – Построение дерева сна большом объеме входных данных

Как видно в среднем длина пути до самого крайнего элемента составляет от 2log2(n) до 3log2(n) , что является неплохим результатом из-за простоты алгоритма.ЗАКЛЮЧЕНИЕ

В ходе работы написана программа, осуществляющая ряд операций с рандомизированными бинарными деревьями. Программа имеет возможность строить дерево, добавлять в него элементы, удалять из него элементы, а также осуществлять поиск по дереву в целом.

Для написания программы использована среда Qt Creator 4.4.0 Изучены и использованы классы QFile, QFileDialog, QTextStream, QMessageBox, QWidget, QMainWindow.

Эффективность поиска в данном типе деревьев не так высока как в АВЛ– деревьях и отстает примерно в 3 раза, но из-за простоты реализации и относительно неплохих показателей этот способ вполне применим так где не быстрота поиска несущественно влияет на ситуацию.

# ПРИЛОЖЕНИЕ А. КОД BINTREE.H

#ifndef BINTREE\_H

#define BINTREE\_H

#include <QChar>

#include "node.h"

typedef QString elem;

typedef struct BT{

struct BT\* Root;

struct BT\* Left;

struct BT\* Right;

elem info;

int key;

int size;

Node\* node;

BT(int k, BT\* root, elem inf) {key=k; Root=root; info=inf; Left=Right=nullptr; size=1;}

}BT;

BT\* **RootBT**(BT\* tree);

BT\* **LeftBT**(BT\* tree);

BT\* **RightBT**(BT\* tree);

bool NullBT(BT\* tree);

int getsize(BT\* tree);

int fixsize(BT\* tree);

BT\* **rotateright**(BT\* tree);

BT\* **rotateleft**(BT\* tree);

BT\* **insertroot**(BT\* tree, BT\* root, int key, QString key\_ch);

BT\* **insertrand**(BT\* tree, BT\* root, int key, QString key\_ch);

BT\* **Find**(BT\* tree, int key);

BT \***delete\_tree**(BT\* tree);

void fixRoots(BT\* tree, BT\* past);

BT\* **merge**(BT\* tree1, BT\* tree2);

BT\* **remove**(BT\* tree, int key);

#endif // BINTREE\_H

# ПРИЛОЖЕНИЕ Б. КОД BINTREE.CPP

#include "bintree.h"

BT\* **RootBT**(BT\* tree){

return tree->Root;

}

BT\* **LeftBT**(BT\* tree){

return tree->Left;

}

BT\* **RightBT**(BT\* tree){

return tree->Right;

}

bool **NullBT**(BT\* tree){

if (tree == nullptr){

return 1;

}

return 0;

}

void **fixRoots**(BT\* tree, BT\* past)

{

if (!NullBT(past)){

tree->Root=past;

}

if (!NullBT(LeftBT(tree))){

fixRoots(LeftBT(tree), tree);

}

if (!NullBT(RightBT(tree))){

fixRoots(RightBT(tree), tree);

}

}

BT\* **delete\_tree**(BT\* tree)

{

if(!tree)

return tree;

if(tree->Left)

delete\_tree(tree->Left);

if(tree->Right)

delete\_tree(tree->Right);

tree=nullptr;

delete tree;

return tree;

}

int **getsize**(BT\* tree)

{

if(!tree)

return 0;

return tree->size;

}

int **fixsize**(BT\* tree)

{

tree->size=getsize(tree->Left)+getsize(tree->Right)+1;

}

BT\* **rotateright**(BT\* tree)

{

BT\* child=tree->Left;

if(!child)

return tree;

tree->Left=child->Right;

child->Root=tree->Root;

tree->Root=child;

child->Right=tree;

child->size=tree->size;

fixsize(tree);

//fixsize(child);

return child;

}

BT\* **rotateleft**(BT\* tree)

{

BT\* child=tree->Right;

if(!child)

return tree;

tree->Right=child->Left;

child->Root=tree->Root;

tree->Root=child;

child->Left=tree;

child->size=tree->size;

//fixsize(child);

fixsize(tree);

return child;

}

BT\* **insertroot**(BT\* tree, BT \*root, int key, QString key\_ch)

{

if(!tree)

{

return new BT(key, root, key\_ch);

}

if(key<tree->key)

{

tree->Left=insertroot(tree->Left, tree, key, key\_ch);

return rotateright(tree);

}

else

{

tree->Right=insertroot(tree->Right, tree, key, key\_ch);

return rotateleft(tree);

}

}

BT\* **insertrand**(BT\* tree, BT \*root, int key, QString key\_ch)

{

QChar conv;

if(!tree)

{

return new BT(key, root, key\_ch);

}

if(rand()%(tree->size+1)==0)

return tree=insertroot(tree, tree->Root, key, key\_ch);

if(tree->key>key)

{

tree->Left=insertrand(tree->Left, tree, key, key\_ch);

}

else

{

tree->Right=insertrand(tree->Right, tree, key, key\_ch);

}

fixsize(tree);

return tree;

}

BT\* **merge**(BT\* tree1, BT\* tree2)

{

if(!tree1)

return tree2;

if(!tree2)

return tree1;

if(rand()%(tree1->size+tree2->size) < tree1->size )

{

tree1->Right = merge(tree1->Right,tree2);

tree1->Root=tree1->Root->Root;

fixsize(tree1);

return tree1;

}

else

{

tree2->Left = merge(tree1,tree2->Left);

tree2->Root=tree2->Root->Root;

fixsize(tree2);

return tree2;

}

}

BT\* **remove**(BT\* tree, int key)

{

if( !tree )

return tree;

if( tree->key==key )

{

BT\* temp = merge(tree->Left,tree->Right);

delete tree;

return temp;

}

else if( key<tree->key )

tree->Left = remove(tree->Left,key);

else

tree->Right = remove(tree->Right,key);

return tree;

}

BT\* **Find**(BT\* tree, int key)

{

if(!tree)

return nullptr;

if(key==tree->key)

return tree;

if(key<tree->key)

return Find(tree->Left, key);

if(key>tree->key)

return Find(tree->Right, key);

}

# ПРИЛОЖЕНИЕ В. КОД MAINWINDOW.CPP

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QFile>

#include <QFileDialog>

#include <QString>

#include <QTextStream>

#include <QMessageBox>

#include "graphwidget.h"

#include "bintree.h"

#include "node.h"

#include <QWidget>

BT\* tree;

bool gravity = false;

int **strsize**(QString str)

{

int size=0;

int i=0;

while(i<str.size())

{

if(str[i]==' ')

size++;

i++;

}

return size;

}

BT\* **Read**(QString str, int i, QTableWidget\* tw){

int j=0;

//QStringList str\_l=str.split(" ", QString::SkipEmptyParts);

QByteArray str\_b=str.toLatin1();

char\* str\_ch=str\_b.data();

int str\_size=strsize(str)+1;

QString\* arr\_ch2=new QString[str\_size+1];

QString\* arr\_ch=new QString[str\_size+1];

int\* arr=new int[str\_size];

char\* end=strtok(str\_ch, " ;");

while(end!=NULL)

{

QString end\_s(end);

arr\_ch[i]=end\_s;

i++;

end=strtok(NULL, " ;");

QString end\_s2(end);

arr\_ch[i]=end\_s2;

arr\_ch2[j]=arr\_ch[i];

arr[j]=arr\_ch[i].toInt();

j++;

end=strtok(NULL, " ;");

}

for(i=0; i<str\_size; i++)

{

tw->insertRow(i);

tw->setItem(i, 0, new QTableWidgetItem(arr\_ch2[i]));

tw->setItem(i, 1, new QTableWidgetItem(arr\_ch[i]));

}

i=0;

tree=insertrand(tree, NULL, arr[0], arr\_ch[0]);

for((i)=1; (i)<str\_size; (i)++)

{

tree=insertrand(tree, NULL, arr[i], arr\_ch[i]);

}

fixRoots(tree, nullptr);

return tree;

}

BT\* **ReadFile**(QString fileName, QTextEdit\* te, QTableWidget\* tw){

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);

return Read(temp, i, tw);

}

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

tree = nullptr;

this->createUI(QStringList()<<trUtf8("Key")

<<trUtf8("Element"));

}

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

{

delete ui;

}

void MainWindow::**createUI**(const QStringList &headers)

{

ui->tableWidget->setColumnCount(2);

ui->tableWidget->setShowGrid(true);

ui->tableWidget->setSelectionMode(QAbstractItemView::SingleSelection);

ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->tableWidget->setHorizontalHeaderLabels(headers);

ui->tableWidget->horizontalHeader()->setStretchLastSection(true);

ui->tableWidget->hideColumn(3);

}

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

{

tree=delete\_tree(tree);

exit(0);

}

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

{

tree=delete\_tree(tree);

ui->tableWidget->clear();

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

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

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

}

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

{

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

if(tree==nullptr)

{

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

return;

}

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

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\_action\_11\_triggered**()

{

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

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

{

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

return;

}

QString points\_arr=ui->textEdit\_2->toPlainText();

QStringList points\_arr\_l=points\_arr.split(" ", QString::SkipEmptyParts);

BT\* temp=Find(tree, points\_arr\_l.first().toInt());

if(temp!=nullptr)

{

ui->textEdit\_3->insertPlainText(temp->info);

return;

}

else

{

ui->textEdit\_3->setText("ERROR, no such element");

return;

}

}

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

{

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

gravity = true;

}

else

gravity = false;

}

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

{

int point\_i;

QString point;

point=ui->textEdit\_4->toPlainText();

point\_i=point.toInt();

tree=remove(tree, point\_i);

ui->tableWidget->removeRow(point\_i-1);

fixRoots(tree, nullptr);

if(tree!=nullptr)

{

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

QMainWindow\* temp = new QMainWindow;

temp->setCentralWidget(widget);

temp->setWindowTitle("Graph output");

temp->show();

}

}

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

{

QString element=ui->textEdit\_5->toPlainText();

QStringList list=element.split(";", QString::SkipEmptyParts);

tree=insertrand(tree, nullptr, list[1].toInt(), list[0]);

fixRoots(tree, nullptr);

int rows=ui->tableWidget->rowCount();

ui->tableWidget->insertRow(rows);

ui->tableWidget->setItem(rows, 0, new QTableWidgetItem(list[1]));

ui->tableWidget->setItem(rows, 1, new QTableWidgetItem(list[0]));

if(tree!=nullptr)

{

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

QMainWindow\* temp = new QMainWindow;

temp->setCentralWidget(widget);

temp->setWindowTitle("Graph output");

temp->show();

}

}

# ПРИЛОЖЕНИЕ Г. КОД NODE.H

#ifndef NODE\_H

#define NODE\_H

#include <QGraphicsItem>

#include <QList>

class Edge;

class GraphWidget;

QT\_BEGIN\_NAMESPACE

class QGraphicsSceneMouseEvent;

QT\_END\_NAMESPACE

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;

//int text;

QString text;

protected:

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

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

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

private:

QList<Edge \*> edgeList;

QPointF newPos;

GraphWidget \*graph;

bool\* gravity;

};

#endif // NODE\_H

# ПРИЛОЖЕНИЕ Д. КОД NODE.CPP

#include "edge.h"

#include "node.h"

#include "graphwidget.h"

#include "bintree.h"

#include <QGraphicsScene>

#include <QGraphicsSceneMouseEvent>

#include <QPainter>

#include <QStyleOption>

#include <QString>

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

: graph(graphWidget)

{

setFlag(ItemIsMovable);

setFlag(ItemSendsGeometryChanges);

setCacheMode(DeviceCoordinateCache);

setZValue(-1);

gravity = gr;

}

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

{

edgeList << edge;

edge->adjust();

}

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

{

return edgeList;

}

void Node::**calculateForces**()

{

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

newPos = pos();

return;

}

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;

}

}

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;

}

qreal lim = 0.1;

if (\*gravity == false){

lim = 1000;

}

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

xvel = yvel = 0;

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));

}

bool Node::**advance**()

{

if (newPos == pos())

return false;

setPos(newPos);

return true;

}

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

}

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

{

qreal mult=5;

QPainterPath path;

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

return path;

}

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-1.5)\*(text.size()+1), -7, 2\*mult\*(text.size()+1), 20);

else

painter->drawEllipse(-(mult-1.5)\*(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);

}

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);

}

//! [11]

//! [12]

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

{

update();

QGraphicsItem::*mousePressEvent*(event);

}

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

{

update();

QGraphicsItem::*mouseReleaseEvent*(event);

}

# ПРИЛОЖЕНИЕ Е. КОД EDGE.H

#ifndef EDGE\_H

#define EDGE\_H

#include <QGraphicsItem>

class Node;

class Edge : public QGraphicsItem

{

public:

Edge(Node \*sourceNode, Node \*destNode, bool vis);

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;

QPointF sourcePoint;

QPointF destPoint;

qreal arrowSize;

bool visible;

};

#endif // EDGE\_H

# ПРИЛОЖЕНИЕ Ж. КОД EDGE.CPP

#include "edge.h"

#include "node.h"

#include <math.h>

#include <QPainter>

static const double Pi = 3.14159265358979323846264338327950288419717;

static double TwoPi = 2.0 \* Pi;

Edge::**Edge**(Node \*sourceNode, Node \*destNode, bool vis)

: arrowSize(10)

{

setAcceptedMouseButtons(0);

source = sourceNode;

dest = destNode;

visible = vis;

source->addEdge(this);

dest->addEdge(this);

adjust();

}

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

{

return source;

}

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

{

return dest;

}

void Edge::**adjust**()

{

if (!source || !dest)

return;

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

qreal length = line.length();

prepareGeometryChange();

if (length > qreal(20.)) {

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

sourcePoint = line.p1() + edgeOffset;

destPoint = line.p2() - edgeOffset;

} else {

sourcePoint = destPoint = line.p1();

}

}

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

{

if (!source || !dest)

return QRectF();

qreal penWidth = 1;

qreal extra = (penWidth + arrowSize) / 2.0;

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

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

.normalized()

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

}

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

{

if (!source || !dest)

return;

QLineF line(sourcePoint, destPoint);

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

return;

if (visible){

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

painter->drawLine(line);

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

if (line.dy() >= 0)

angle = TwoPi - angle;

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::black);

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

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

}

}

# ПРИЛОЖЕНИЕ З. КОД GRAPHWIDGET.H

#ifndef GRAPHWIDGET\_H

#define GRAPHWIDGET\_H

#include <QGraphicsView>

#include "bintree.h"

#include "node.h"

class Node;

class GraphWidget : public QGraphicsView

{

Q\_OBJECT

public:

GraphWidget(QWidget \*parent = 0, BT \*tree = nullptr, bool \*gr = nullptr);

void **itemMoved**();

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

public slots:

protected:

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

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

void **scaleView**(qreal scaleFactor);

private:

int timerId;

Node \*centerNode;

};

#endif // GRAPHWIDGET\_H

# ПРИЛОЖЕНИЕ И. КОД GRAPHWIDGET.CPP

#include "graphwidget.h"

#include "edge.h"

#include "node.h"

#include "bintree.h"

#include <math.h>

#include <QKeyEvent>

Оглавление

[ВВЕДЕНИЕ 5](#_Toc501613359)

[1. ОПЕРАЦИИ С БИНАРНЫМ ДЕРЕВОМ 6](#_Toc501613360)

[1.1. Структура дерева. Базовые операции с бинарным деревом 6](#_Toc501613361)

[1.2. Поиск элемента 7](#_Toc501613362)

[1.3. Добавление элемента 7](#_Toc501613363)

[1.4. Вычисление весов 9](#_Toc501613364)

[1.5. Удаление элементов 9](#_Toc501613365)

[ИНТЕРФЕЙС ПРОГРАММЫ 11](#_Toc501613366)

[1.6. Графический интерфейс 11](#_Toc501613367)

[1.7. Общие переменные 11](#_Toc501613368)

[2. ГРАФИЧЕСКОЕ ОТОБРАЖЕНИЕ ДЕРЕВА 12](#_Toc501613369)

[3. ТЕСТИРОВАНИЕ 13](#_Toc501613370)

[3.1. Тестирование на примере простой последовательности 13](#_Toc501613371)

[3.2. Тестирования на большом объеме данных 15](#_Toc501613372)

[ПРИЛОЖЕНИЕ А. КОД BINTREE.H 18](#_Toc501613373)

[ПРИЛОЖЕНИЕ Б. КОД BINTREE.CPP 19](#_Toc501613374)

[ПРИЛОЖЕНИЕ В. КОД MAINWINDOW.CPP 23](#_Toc501613375)

[ПРИЛОЖЕНИЕ Г. КОД NODE.H 27](#_Toc501613376)

[ПРИЛОЖЕНИЕ Д. КОД NODE.CPP 28](#_Toc501613377)

[ПРИЛОЖЕНИЕ Е. КОД EDGE.H 31](#_Toc501613378)

[ПРИЛОЖЕНИЕ Ж. КОД EDGE.CPP 32](#_Toc501613379)

[ПРИЛОЖЕНИЕ З. КОД GRAPHWIDGET.H 34](#_Toc501613380)

[ПРИЛОЖЕНИЕ И. КОД GRAPHWIDGET.CPP 35](#_Toc501613381)

int i=0;

int Xd=1500;

int GraphWidget::**Build**(QWidget\* parent, BT \*tree, qreal x, qreal y, QGraphicsScene \*scene, bool\* gr){

tree->node = new Node(this, gr);

scene->addItem(tree->node);

tree->node->setPos(x, y);

while(i<tree->info.size())

{

tree->node->text[i] = tree->info[i];

i++;

}

i=0;

if ((RootBT(tree)!=nullptr)&&(tree->Root->node!=nullptr)){

scene->addItem(new Edge(tree->node, tree->Root->node, true));

}

qreal delta;

int level = (y+200)/50;

delta = 0.9\*Xd/pow(2, level+2);

if (LeftBT(tree)!=nullptr)

Build(parent, LeftBT(tree), x+delta, y+50, scene, gr);

if (RightBT(tree)!=nullptr)

Build(parent, RightBT(tree), x-delta, y+50, scene, gr);

if ((LeftBT(tree)!=nullptr) && (RightBT(tree)!=nullptr)){

scene->addItem(new Edge(LeftBT(tree)->node, RightBT(tree)->node, false));

if ((RightBT(LeftBT(tree))!=nullptr) && ((LeftBT(RightBT(tree)))!=nullptr)){

scene->addItem(new Edge(RightBT(LeftBT(tree))->node, LeftBT(RightBT(tree))->node, false));

// if ((RightBT(LeftBT(LeftBT(tree)))!=nullptr) && ((LeftBT(RightBT(RightBT(tree))))!=nullptr))

// scene->addItem(new Edge(RightBT(RightBT(LeftBT(tree)))->node, LeftBT(LeftBT(RightBT(tree)))->node, false));

}

}

return 0;

}

GraphWidget::**GraphWidget**(QWidget \*parent, BT\* tree, bool\* gr)

: QGraphicsView(parent), timerId(0)

{

QGraphicsScene \*scene = new QGraphicsScene(this);

scene->setItemIndexMethod(QGraphicsScene::NoIndex);

scene->setSceneRect(-750, -300, 1500, 1200);

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"));

Build(this, tree, 0, -200, scene, gr);

}

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;

}

}

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

{

Q\_UNUSED(rect);

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);

painter->drawRect(sceneRect);

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);

}