**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**Пояснительная записка**

**к курсовой работе**

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

**Тема: Бинарные алгоритмы поиска**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Азаревич А.Д. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Постановка задачи.**

Разработать программу, реализующую постройку бинарного дерева поиска случайного типа, по которому можно искать нужные элементы, а в случае их отсутствия – добавлять.

Программа должна визуализировать бинарное дерево, чтобы её можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с ней действий.

**Основные теоретические положения.**

Бинарное дерево называется бинарным деревом поиска(БДП), если для каждого узла x выполняется следующее условие: для каждого узла l в левом поддереве справедливо key(l) < key(x), а для каждого узла r в правом поддереве – key(r) > key(x).

Можно дать другое определение БДП: бинарное дерево является БДП, если ключи узлов дерева упорядочены при ЛКП-обходе. Например (рисунок 1).

6

3

9

1

5

7

11

8

4

Рис. 1 - Пример дерева поиска

Во всех бинарных деревьях поиска процедура поиска элемента выполняется по следующему алгоритму. Если корень дерева содержит искомый ключ, то поиск завершен. Иначе, если значение в корне дерева меньше искомого, то выполняем поиск в правом поддереве, иначе – в левом поддереве. Если мы переходим в нулевое поддерево, это означает, что искомого элемента в дереве нет. Эта процедура схожа с бинарным поиском.

В случайных БДП структура дерева определяется последовательностью вставок и удалений элементов. При этом высота дерева никак не регулируется, поэтому на некоторых последовательностях операций (например, вставка упорядоченной последовательности элементов), дерево будет вырождаться в линейный список с максимальной высотой, равной n.

**Спецификация программы.**

*Назначение программы*.

Программа предназначена для создания случайного бинарного дерева поиска и добавления в него новых элементов, а так же визуализации данных процессов.

*Описание программы*.

Программа написана на языке С++ с использованием компилятора MinGW 5.3.0 32-bit (код смотри в приложении А). Входными данными для программы является последовательность значений типа int (одно значение встречается не более одного раза), являющаяся основой, по которой собирается случайное БДП. В случае, если введённый отсутствует в дереве, то он к нему присоединяется, а на картинке выделяется красным, если же элемент в дереве уже был – выделяется зелёным. Выходными данными является визуализация текущего дерева и информация, есть ли данный элемент в списке или нет.

*Реализация*.

Классы:

1. BTS – класс, представляющий собой случайное БДП.

Переменные класса:

1. BT \*left – указатель на левое поддерево;
2. BT \*right – указатель на правое поддерево;
3. Elem key – значение, содержащееся в данном элементе БДП (Elem = double).

Функции класса:

1. BTS( Elem x);

Конструктор класса. Присваивает key значение x, приравнивает left и right к 0.

1. ~BTS();

Деструктор класса. Вызывает деструктор для левого и правого поддерева.

1. BTS \*Left\_Child();

Возвращает адрес левого сына;

1. BTS \*Right\_Child();

Возвращает адрес правого сына;

1. Elem Ret\_Key();

Возвращает значение key текущего элемента;

1. bool Find\_Add(Elem &x)

Проверяет равенство key==x (ищет х в БДП), если не найдёт – создаёт новый элемент, в котором выполняется данное равенство ;

Входные параметры:

Искомый/вставляемый элемент x;

Выходное значение:

True, если x является элементом бинарного дерева поиска, false – не являлся, но теперь есть;

Постусловие:

Сравнивает значение x с key. Если x< key идёт в левое поддерево, если x>key – в правое, если x==key возвращает true. В случае x<>key, а нужное поддерево отсутствует, то оно создаётся с key=x, возвращается значение false.

Сторонние классы:

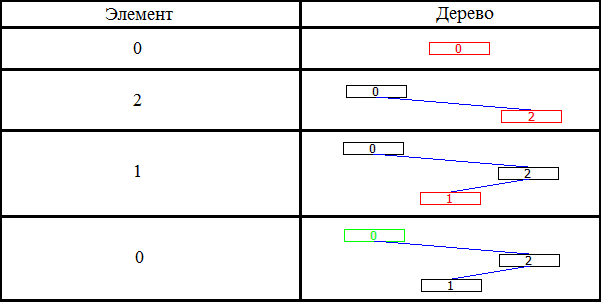
TreeView – отвечает за рисование дерева;

TreeWidget – отвечает за «общение» с пользователем.

**Тестирование.**

Результат прогона программы по 4 элементам представлен в табл. 1.

Табл. 1 – Результаты прогона программы

****

**Зависимость времени работы алгоритма поиска от размера БДП.**

Пусть по случайному БДП легко искать нужные ключи(элементы) за счёт того, что значение корня всегда больше значения правого поддерева и меньше левого поддерева, однако из-за того, что соотношение элементов определяется только их добавлением и удалением, в итоге дерево может выродиться в линейный список. А это приводит к тому, что поиск нужного элемента может затянуться (особенно для огромных БДП).

Во избежание подобных случаев могут использовать 3 вида бинарных деревьев поиска:

* БДП с рандомизацией – элементы добавляются в конец или в основание дерева;
* АВЛ-дерево и красно-черное дерево – в случае, если в дереве разница между высотой левого и правого поддеревьев больше 1, происходит балансировка дерева.

**Выводы.**

В ходе работы была разработана программа, визуализирующая построение бинарного дерева поиска случайного типа и поиск в нём элементов по заданным ключам.

Было выявлено, что данным вид дерева поиска пусть и является самым простым в реализации, однако на практике он весьма неэффективен, особенно при огромном количестве элементов. Из-за этого чаще используют такие деревья поиска как: БДП с рандомизацией, АВЛ-деревье и красно-черные деревья.

**Приложение А**

**Исходный код**

bts.h

// Курсовая работа

// bts.h

// Азаревич Артём, гр. 6383

// 11.12.17

//

// Бинарные алгоритмы поиска. БДП случайного типа

#ifndef BTS\_H

#define BTS\_H

#include <iostream>

#include <fstream>

#include <string.h>

#include <vector>

typedef int Elem;

class BTS

{

public:

BTS( Elem x);

~BTS();

bool **Find\_Add** (Elem &x);

BTS \***Left\_Child**();

BTS \***Right\_Child**();

Elem **Ret\_key**();

private:

Elem key;

BTS \*left;

BTS \*right;

};

#endif // BTS\_H

bts.cpp

// Курсовая работа

// bts.cpp

// Азаревич Артём, гр. 6383

// 11.12.17

//

// Бинарные алгоритмы поиска. БДП случайного типа

#include "bts.h"

BTS::**BTS**( Elem x )

{

key = x;

left = right = NULL;

}

BTS::~BTS()

{

if (left != NULL)

delete left;

if (right != NULL)

delete right;

}

bool BTS::**Find\_Add** (Elem &x)

{

if ( key > x)

{

if ( left == NULL )

{

left = new BTS(x);

return false;

}

return left->Find\_Add(x);

}

if ( key < x)

{

if ( right == NULL )

{

right = new BTS(x);

return false;

}

return right->Find\_Add(x);

}

if ( key == x )

return true;

}

Elem BTS::**Ret\_key**()

{

return key;

}

BTS \* BTS::**Left\_Child**()

{

return left;

}

BTS \* BTS::**Right\_Child**()

{

return right;

}

treeview.h

// Курсовая работа

// treeview.h

// Азаревич Артём, гр. 6383

// 11.12.17

//

// Бинарные алгоритмы поиска. БДП случайного типа

#ifndef TREEVIEW\_H

#define TREEVIEW\_H

#include <QWidget>

#include "bts.h"

class TreeView : public QWidget

{

Q\_OBJECT

public:

explicit TreeView(QWidget \*parent = nullptr);

void **Push\_key**(Elem a);

void **Push\_flag**(bool a);

BTS\* tree;

protected:

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

Elem **Pop\_key**();

bool **Pop\_flag**();

private:

void **drawNode**(QPainter \*painter, BTS \*root, int x, int y);

// Ширина/Высота элемента

int nodeWidth{ 120 };

int nodeHeight{ 25 };

Elem key; // Имя искомого/добавляемого элемента

bool flag; // true - элемент был, false - элемент добавили

};

#endif // TREEVIEW\_H

treeview.cpp

// Курсовая работа

// treeview.cpp

// Азаревич Артём, гр. 6383

// 11.12.17

//

// Бинарные алгоритмы поиска. БДП случайного типа

#include <QPainter>

#include <QFontMetrics>

#include <sstream>

#include "treeview.h"

TreeView::**TreeView**(QWidget \*parent) : QWidget(parent), tree(nullptr)

{

tree = NULL;

}

void TreeView::***paintEvent***(QPaintEvent \*)

{

QPainter painter(this);

drawNode(&painter, tree, 0, 0);

}

// Ставим, проверяем искомый элемент

void TreeView::**Push\_key**(Elem a)

{

key = a;

}

Elem TreeView::**Pop\_key**()

{

return key;

}

// Ставим, проверяем флаг

void TreeView::**Push\_flag**(bool a)

{

flag = a;

}

bool TreeView::**Pop\_flag**()

{

return flag;

}

// Отрисовываем дерево

void TreeView::**drawNode**(QPainter \*painter, BTS\* root, int x, int y)

{

if(root==nullptr)return;

painter->save();

Elem k = Pop\_key();

Elem root\_k = root->Ret\_key();

bool f = Pop\_flag();

// Описываем точку, от которой рисуем элемент

QPoint p(width() / pow(2, y) \* x + width() / pow(2, y+1), nodeHeight \* y);

// Описываем прямоугольник (рамку)

QRect rec(p.x() - nodeWidth/4, p.y(), nodeWidth/2, nodeHeight/2);

// Веделяем цветом искомый/добавленый элемент

if (k == root\_k)

{

if ( f )

painter->setPen(Qt::green);

else

painter->setPen(Qt::red);

}

// Рисуем рамку

painter->drawRect(rec);

// Шрифт текста

QFont font = painter->font() ;

font.setPointSize(10);

painter->setFont(font);

// Текст

std::stringstream str;

str<<root->Ret\_key();

// Пишем текст

QString text = fontMetrics().elidedText( QString::fromLatin1(str.str().c\_str()), Qt::ElideLeft, rec.width() );

painter->drawText(rec, Qt::AlignCenter, text);

// Рисуем левго ребенка

if(root->Left\_Child() != nullptr)

{

painter->setPen(Qt::blue);

painter->drawLine(p.x(), p.y() + nodeHeight / 2, p.x() - width()/pow(2, y+2), p.y() + nodeHeight);

painter->setPen(Qt::black);

drawNode(painter, root->Left\_Child(), 2\*x, y+1);

}

// Рисуем правого ребенка

if(root->Right\_Child() != nullptr)

{

painter->setPen(Qt::blue);

painter->drawLine(p.x(), p.y() + nodeHeight / 2, p.x() + width()/pow(2, y+2), p.y() + nodeHeight);

painter->setPen(Qt::black);

drawNode(painter, root->Right\_Child(), 2\*x + 1, y+1);

}

painter->restore();

}

treewidget.h

// Курсовая работа

// treewidget.h

// Азаревич Артём, гр. 6383

// 11.12.17

//

// Бинарные алгоритмы поиска. БДП случайного типа

#ifndef TREEWIDGET\_H

#define TREEWIDGET\_H

#include <QWidget>

#include <QLineEdit>

#include <QPushButton>

#include <QLabel>

#include "treeview.h"

class TreeWidget : public QWidget

{

Q\_OBJECT

public:

explicit TreeWidget(QWidget \*parent = nullptr);

private:

TreeView \*view\_;

QLineEdit \*lineEdit\_;

QPushButton \*button\_;

QLabel \*code;

QLabel \*step;

unsigned int stateIndex=0;

public slots:

void **insert**();

void **next**();

void **back**();

};

#endif // TREEWIDGET\_H

treewidget.cpp

// Курсовая работа

// treewidget.cpp

// Азаревич Артём, гр. 6383

// 11.12.17

//

// Бинарные алгоритмы поиска. БДП случайного типа

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QLabel>

#include <QMessageBox>

#include<QRegExpValidator>

#include "treewidget.h"

TreeWidget::**TreeWidget**(QWidget \*parent) : QWidget(parent)

{

view\_ = new TreeView(this);

view\_->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

// "Введите искомый элемент: "

QLabel \*label = new QLabel("Введите искомый элемент: ");

// Окошко для вводимых значений

lineEdit\_ = new QLineEdit();

lineEdit\_->setValidator( new QIntValidator( -2147483648, 2147483647 ) );

// Кнопка

button\_ = new QPushButton("Искать");

step = new QLabel("Начните строить дерево");

// Горизонтальные виджеты (1)

QHBoxLayout \*layout = new QHBoxLayout;

layout->addWidget(label);

layout->addWidget(lineEdit\_);

layout->addWidget(button\_);

// Горизонтальные виджеты (2)

QHBoxLayout \*infLayout=new QHBoxLayout;

infLayout->addWidget(step);

// Выравниваем виджеты

QVBoxLayout \*mainLayout = new QVBoxLayout;

mainLayout->addLayout(layout);

mainLayout->addLayout(infLayout);

mainLayout->addWidget(view\_);

setLayout(mainLayout);

connect(button\_, &QPushButton::clicked, this, &TreeWidget::insert);

}

void TreeWidget::**insert**()

{

Elem a = lineEdit\_->text().toInt();

view\_->Push\_key(a);

if (view\_->tree == NULL)

{

step->setText(QString::asprintf("Постройка дерева начата"));

view\_->tree = new BTS(a);

view\_->Push\_flag(false);

}

else

if (view\_->tree->Find\_Add(a))

{

step->setText(QString::asprintf("Элемент уже есть"));

view\_->Push\_flag(true);

}

else

{

step->setText(QString::asprintf("Элемент добавлен"));

view\_->Push\_flag(false);

}

update();

}

mainwindow.h

// Курсовая работа

// mainwindow.h

// Азаревич Артём, гр. 6383

// 11.12.17

//

// Бинарные алгоритмы поиска. БДП случайного типа

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "treewidget.h"

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

private:

TreeWidget \*widget\_;

};

#endif // MAINWINDOW\_H

mainwindow.cpp

// Курсовая работа

// mainwindow.cpp

// Азаревич Артём, гр. 6383

// 11.12.17

//

// Бинарные алгоритмы поиска. БДП случайного типа

#include "mainwindow.h"

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

{

setWindowTitle("Визуализация бинарного дерева поиска");

setMinimumSize(640, 480);

widget\_ = new TreeWidget(this);

setCentralWidget(widget\_);

}

main.cpp

// Курсовая работа

// main.cpp

// Азаревич Артём, гр. 6383

// 11.12.17

//

// Бинарные алгоритмы поиска. БДП случайного типа

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}