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: Бинарные деревья поиска

Студентка гр. 7383 _____ Иолшина В.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург

2018

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Иолшина В.

Группа 7383

Тема работы: Бинарные деревья поиска

Содержание пояснительной записки:

- Содержание
- Введение
- Теоретические сведения
- Решение задачи
- Примеры работы программы
- Заключение
- Приложение А. Исходный код программы

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 19.10.2018

Дата сдачи курсовой работы:

Дата защиты курсовой работы:

Студентка _____ Иолшина В.

Преподаватель _____ Размочаева Н. В.

АННОТАЦИЯ

В курсовой работе реализована программа, визуализирующая вставку и исключение элемента в бинарном дереве поиска на языке C++, с использованием фреймворка Qt.

SUMMARY

In the course work was implemented a program that visualizes the insertion and elimination of an element in a binary search tree in C ++ using the Qt framework.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является создание GUI приложения на языке C++ с использованием фреймворка Qt для визуализации вставки и исключения элемента в бинарное дерево поиска.

Задачей является написание программы, которая ищет, вставляет или исключает элемент из бинарного дерева поиска, а также пошагово выводит на экран результат этих действий.

1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Бинарное дерево поиска

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).

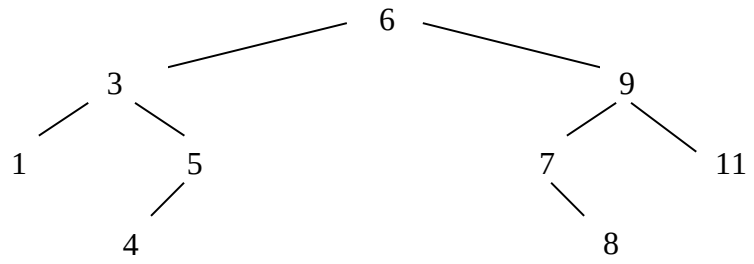

Рисунок 1 - Пример дерева поиска

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.

Вставка элемента со значением key в дерево tree.

Если дерево пусто, создаем новый узел со значением key, иначе, если key(tree)>key то выполняем рекурсивную вставку в левое поддерево tree, иначе — в правое поддерево.

Удаление элемента со значением key из дерева tree.

Если удаляемый узел b — лист, то просто удаляем его. Если у узла b один ребенок x, то удаляем узел b, вместо него помещаем узел x. Если у узла b два ребенка, то находим узел x, который непосредственно следует за узлом b при ЛКП обходе, т. е. узел с минимальным значением в правом поддереве. Обмениваем значения узлов b и x и удаляем узел x , имеющий не более одного ребенка (рис. 2).

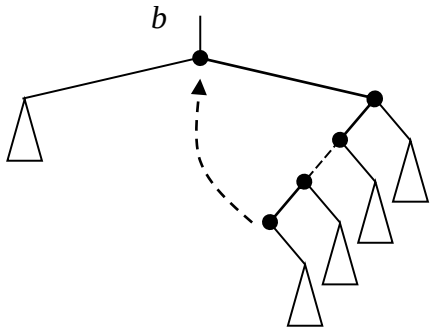

Рисунок 2 – Удаление из дерева узла с двумя детьми

2. РЕШЕНИЕ ЗАДАЧИ

Входными данными для программы является последовательность значений типа int, представляющая собой вставляемые и удаляемые элементы из дерева. Выходными данными является визуализация текущего дерева и информация о том, был ли данный элемент в дереве.

Был создан класс BTS, 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.
2) bool find_add(Elem &x) – проверяет равенство key==x (ищет x в БДП), если не найдёт – создаёт новый элемент, в котором выполняется данное равенство;
3) bool delete_el(Elem k, BTS *&upper, int dir) – удаляет из дерева элемент k;
4) bool find(BST* tree, int &k) – проверяет, есть ли элемент k в дереве;
5) BST* merge(BST* l, BST* r) – вызывается из функции delete_el и объединяет левое и правое поддеревья после удаления их родителя.

3. ПРИМЕР РАБОТЫ ПРОГРАММЫ

Результат работы программы по добавлению в дерево 4 элементов представлен на рис.3.а-г:

А) Добавляемый элемент «5»

Рисунок 3.а – Построение дерева (1)

Б) Добавляемый элемент «3»

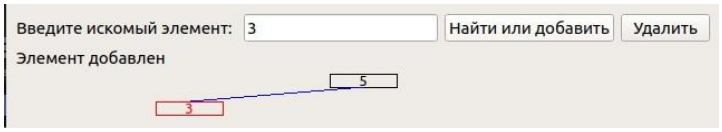

Рисунок 3.б – Построение дерева (2)

В) Добавляемый элемент «9»

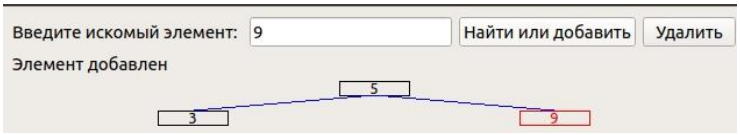

Рисунок 3.в – Построение дерева (3)

Г) Добавляемый элемент «4»

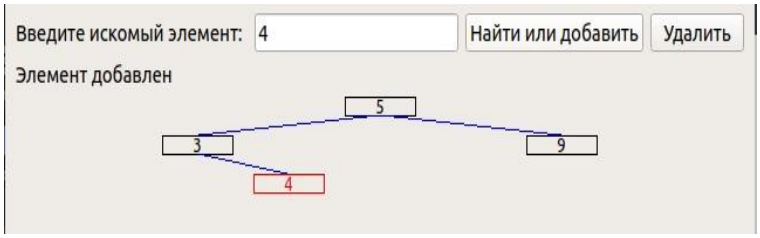

Рисунок 3.г – Построение дерева (4)

Результат работы программы по удалению из дерева (из рис. 3.г) 4-х элементов представлен на рис.4.а-г.

А) Удаляемый элемент «3»

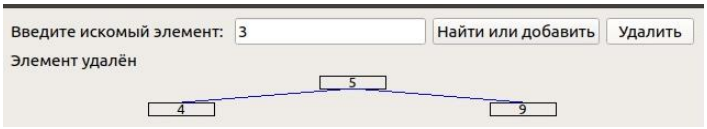

Рисунок 4.а – Удаление из дерева (1)

Б) Удаляемый элемент «9»

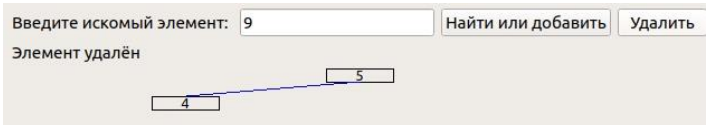

Рисунок 4.б – Удаление из дерева (2)

В) Удаляемый элемент «5»

Рисунок 4.в – Удаление из дерева (3)

Г) Удаляемый элемент «4»

Рисунок 4.г – Удаление из дерева (4)

ЗАКЛЮЧЕНИЕ

В ходе работы была реализована программа вставки и исключения элемента в бинарном дереве на языке программирования C++ с использованием фреймворка Qt Creator. Для взаимодействия с программой был релизован графический интерфейс.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методические указания к лабораторным работам и практическим занятиям, по выполнению курсовой работы по дисциплине «Алгоритмы и структуры данных»»: учеб.-метод. пособие / сост.: С.А Ивановский , Т.Г. Фомичева , О.М. Шолохова. СПб. 2017. 86 с.

2. Динамические структуры данных: учеб. пособие / А.Ю. Алексеев, С.А. Ивановский, Д.В. Куликов. - СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2004.

3. Книга пяти и более авторов / И. И. Иванов, П. П. Петров, С. С. Сидоров и др.. СПб.: Издательство, 2010. 000 с.

4. Макконелл Дж. Анализ алгоритмов. Вводный курс: учебное пособие / пер. с англ. С.К.Ландо. - М. : Техносфера, 2002

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

bst.h

```
#ifndef BST_H
#define BST_H

#include <QWidget>
#include<iostream>

class BST
{
public:
  int key;
  BST* left;
  BST* right;
  BST(int x);
bool find_add(BST* tree, int &x);
bool find(BST* tree, int &k);
BST* delete_el(BST *&tree, int &k, int gl);
BST* merge(BST* l, BST* r);
};
#endif // BST_H
```

bts.cpp

```
#include "bst.h"
#include <iostream>
using namespace std;

BST::BST(int x)
{
    key = x;
    left = right = NULL;
}

BST* destroy(BST* tree)
{
  if (left)
    delete tree->left;
  if (right)
    delete tree->right;
  delete tree;
  return tree = NULL;
    }

bool BST::find_add (BST* tree, int &k)
{
  if (tree->key > k)
  {
    if (tree->left == NULL)
    {
      tree->left = new BST(k);
      return false;
    }
      return find_add(tree->left, k);
  }
  if (tree->key < k)
  {
    if (tree->right == NULL)
    {
```

```
      tree->right = new BST(k);
      return false;
    }
    return find_add(tree->right, k);
  }
  if (tree->key == k)
    return true;
}

bool BST::find(BST* tree, int &k)
{
    if(tree == NULL)
        return false;
    if(k==tree->key)
        return true;
    else if (k<tree->key)
        return find(tree->left, k);
    else
        return find(tree->right, k);

}

BST* BST::merge(BST* l, BST* r)
{
    if( !l )
        return r;
    if( !r )
        return l;
    BST* h = r;
    BST* k;
    if (!h->left)
    {
     h->left=l;
     return h;
    }
    while(h->left)
    {
     k=h;
     h=h->left;
    }
    if (h->right)
    {
        k->left=h->right;
    }
    else
     k->left=NULL;
    h->right=r;
    h->left=l;
    return h;
}

BST* BST::delete_el(BST *&tree, int &k, int gl)
{
    if( tree==NULL )
        return tree;
    if( tree->key==k )
    {
         BST* tmp = tree->merge(tree->left, tree->right);
         tree=tmp;
         return tmp;
    }
    else if( k < tree->key )
```

```
    tree->left = tree->delete_el(tree->left, k, gl);
    else
    tree->right = tree->delete_el(tree->right, k, gl);
    return tree;
}
```

treeview.h

```
#ifndef TREEVIEW_H
#define TREEVIEW_H
#include "bst.h"

class TreeView : public QWidget
{
Q_OBJECT
public:
explicit TreeView(QWidget *parent = nullptr);
void Push_key(int a);
void Push_flag(bool a);
BST* tree;
protected:
void paintEvent(QPaintEvent *event) override;
int Pop_key();
bool Pop_flag();

private:
void drawNode(QPainter *painter, BST *root, int x, int y);
// Ширина/Высота элемента
int nodeWidth{ 120 };
int nodeHeight{ 25 };
int key; // Имя искомого/добавляемого элемента
bool flag; // true - элемент был, false - элемент добавили
};
#endif // TREEVIEW_H
```

treeview.cpp

```
#include <QPainter>
#include <QFontMetrics>
#include <sstream>
#include <cmath>
#include "treeview.h"
#include "bst.h"
#include<iostream>

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(int a)
{
    key = a;
}
```

```
int 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, BST* root, int x, int y)
{
   if(root==nullptr)
      return;
   painter->save();
   int k = Pop_key();
   int root_k = root->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->Height(root) << std::endl <<root->key;
   str << root->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 != 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, 2*x, y+1);
   }
// Рисуем правого ребенка
   if(root->right != 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, 2*x + 1, y+1);
    }
    painter->restore();
    }
```

treewidget.h

```
#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);
TreeView *view_;

private:
QLineEdit *lineEdit_;
QPushButton *button_ins;
QPushButton *button_del;
QLabel *step;

public slots:
void insert();
void del();

};

#endif // TREEWIDGET_
```

treewidget.cpp

```
#include<QRegExpValidator>
#include "treewidget.h"
#include <iostream>

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 ) );
// Кнопки
button_ins = new QPushButton( "Найти или добавить" );
button_del = new QPushButton("Удалить");
step = new QLabel("Начните строить дерево");
// Горизонтальные виджеты (1)
QHBoxLayout *layout = new QHBoxLayout;
layout->addWidget(label);
layout->addWidget(lineEdit_);
layout->addWidget(button_ins);
layout->addWidget(button_del);
```

```
// 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_ins, &QPushButton::clicked, this, &TreeWidget::insert);
connect(button_del, &QPushButton::clicked, this, &TreeWidget::del );
}

void TreeWidget::del()
{
//Считываем число из строчки
int k = lineEdit_->text().toInt();
view_->Push_key(k);
//Если дерево пусто - игнорируем запрос на удаление
if (view_->tree == NULL)
{
   step->setText(QString::asprintf("Дерево пусто"));
   return;
}
if (view_->tree->find(view_->tree, k))
{
   int gl = view_->tree->key;
   if((view_->tree->delete_el((view_->tree), k, gl))!=NULL)
   step->setText(QString::asprintf("Элемент удалён"));
}
else
   step->setText(QString::asprintf("Элемент отсутствует"));
update();
}

void TreeWidget::insert()
{
int a = lineEdit_->text().toInt();
view_->Push_key(a);
if (view_->tree == NULL)
{
   step->setText(QString::asprintf("Постройка дерева начата"));
   view_->tree = new BST(a);
   view_->Push_flag(false);
}
else if (view_->tree->find_add(view_->tree, 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

```
#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

```
#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

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
   QApplication a(argc, argv);
   MainWindow w;
   w.show();

   return a.exec();
}
```