**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №5**

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

**Тема: Бинарное дерево поиска**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9382 |  | Сорокумов С. В. |
| Преподаватель |  | Фирсов М. А. |

Санкт-Петербург

2020

## **Цель работы.**

Изучить бинарное дерево поиска. Решить задачу реализации бинарного дерева поиска с рандомизацией на С++.

## **Задание.**

Вариант 10. Бинарное дерево поиска с рандомизацией, задание 1+2а:

1) По заданной последовательности элементов Elem построить структуру данных определённого типа – БДП или хеш-таблицу;

2а) Для построенной структуры данных проверить, входит ли в неё элемент е типа Elem, и если входит, то в скольких экземплярах. Добавить элемент е в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.

**Пояснение задания.**

На вход программе подаётся файл со случайной последовательностью символов (ASCII). Требуется: построить случайное БДП с рандомизацией, для построенного БДП проверить, входит ли в него элемент е типа Elem, если входит, то вывести количество элементов в дереве, если не входит, то добавить элемент е в дерево поиска. А также нужно предусмотреть добавление элементов.

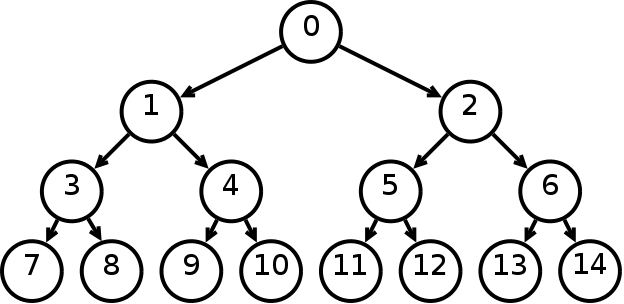
## **Основные теоретические положения.**

Бинарное дерево поиска (БДП) — это бинарное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

• Оба поддерева — левое и правое — являются БДП.

• У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X.

• У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X. Пример БДП представлен на рис. 1.



**Рис 1 - Пример БДП**

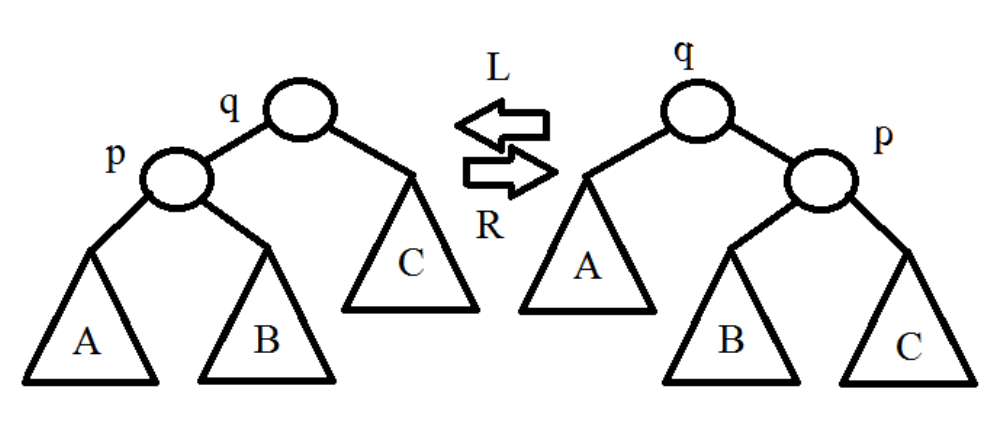
При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется

Рандомизированным бинарным деревом поиска (РБДП) называется такое БДП, в котором последовательность значений, задающая конечное, образовано случайно. Часто в реализации этого типа деревьев учитывают количество повторяющихся элементов. Для сортировки всех ключей РБДП используют обратный порядок обхода дерева (ЛКП-обход).

При поиске элемента сравнивается искомое значение с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается. Вставка в корень происходит следующим образом:

1) Сначала рекурсивно вставляем новый ключ в корень левого или правого поддеревьев (в зависимости от результата сравнения с корневым ключом).

2) Выполняем правый (левый) поворот, который поднимает нужный нам узел в корень дерева. Алгоритм поворотов представлен на рисунке 2.



**Рис 2 - Алгоритм поворотов**

Случайная вставка в дерево включает шанс того, что следующий элемент дерева будет помещен с помощью обычной вставки в дерево или вставкой в корень.

**Структура бинарного дерева, функции для работы с ним:**

**binSTree -** класс бинарного дерева, который хранит в себе:

info - значение ключа

count - количество вхождений элемента

number - количество узлов поддерева элемента

lt - левый потомок узла

rt - правый потомок узла

А также в нем описан конструктор, деструктор класса, метод для проверки инициализации узла.

**void rotateRight(binSTree\*&)** - вращение БДП вправо.

Принимает ссылку на указатель БДП **binSTree\*&**

**void rotateLeft(binSTree\*&)** - вращение БДП влево

Принимает ссылку на указатель БДП **binSTree\*&**

**void printBT(binSTree\*&, std::string)** - печать БДП

Принимает ссылку на указатель БДП **binSTree\*&**, а так же строку из стандартной библиотеки **std::string**

**void insertInRoot(binSTree\*&, char info) -** вставка элемента в корень БДП

Принимает ссылку на указатель БДП **binSTree\*&**, а так же элемент **char info**

**void randomInsert(binSTree\*&, char info)** - рандомизированная вставка в БДП

Принимает ссылку на указатель БДП **binSTree\*&**, а так же элемент **char info**

**int find(binSTree\*& bt, char el)** - поиск элемента в БДП

Принимает ссылку на указатель БДП **binSTree\*&**, а так же элемент **char el**

## **Описание алгоритма.**

Сначала программа запрашивает у пользователя путь до файла, в котором написана последовательность элементов из которых будет состоять дерево, а после посимвольно считывает их. Каждый символ при помощи рандомизированной вставки добавится в дерево. После программа запрашивает какой элемент нужно найти в БДП и выводит количество. Если пользователь захочет дальше добавлять элементы, то это предусмотрено и будет запрошено.

**Тестирование программы**

|  |  |
| --- | --- |
| Входные данные: | Результат |
| sdfghjkls3d4f5g6h7nj8mawrher |  |
| sdfghjkls3d4f5g6h7nj8mawrher |  |
| rtc5yutb7u9jmiumirfv |  |

**Выводы.**

Были изучена структура БДП, функции для работы с ней: вставка в корень, рандомизированная вставка. Решена задача реализации бинарного дерева поиска с рандомизацией на С++. Было проведено тестирование полученной программы. Исходный код представлен в Приложение А.

# **ПРИЛОЖЕНИЕ А**

# **ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Файл source.cpp:**

#include <fstream>

#include "binSTree.h"

#include <ctime>

int main()

{

srand(time(NULL));

std::cout << "Write f - if reading from file, any other - if console" << std::endl;

char symbol;

std::cin >> symbol;

char elem;

/\*

\* Считывания данных из файла

\*/

binSTree\* bt = nullptr;

if (symbol == 'f'){

std::ifstream file;

std::string path;

std::cout << "Enter the name of the file where the tree will be read from" << std::endl;

std::cin >> path;

file.open(path);

if (!file) {

std::cout << "File could not be opened, program terminated" << std::endl;

system("pause");

std::cout << "To continue the program, press any key ...";

std::cin.get();

exit(1);

}

std::cout << "File opened successfully, start building" << std::endl;

while (file >> elem)

{

/\*

\* Занесение данных в дерево

\*/

randomInsert(bt, elem);

}

printBT(bt, "");

} else{

/\*

\* Считывание данных из стандартного потока

\*/

std::cout<< "Enter count of elem"<< std::endl;

int count;

std::cin >> count;

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

{

std::cin >> elem;

/\*

\* Занесение данных в дерево

\*/

randomInsert(bt, elem);

}

printBT(bt, "");

}

std::cout << "Which item to find?" << std::endl;

std::cin >> elem;

int count = find(bt, elem);

if (count == 0) std::cout << "There is no such element in the tree" << std::endl;

else std::cout << "In the tree is " << count << " an instance of the injected element" << std::endl;

/\*

\* Вызов функции вставки элемента

\*/

randomInsert(bt, elem);

std::cout << "The tree after adding:" << std::endl;

printBT(bt, "");

/\*

\* Работа с пользователем

\* Возможность закончить программу или продолжить работу

\*/

char isWork = '\0';

while (isWork != 'n')

{

std::cout << "Do you want to continue working with the tree? y - yes, n - end the program" << std::endl;

std::cin >> isWork;

if (isWork == 'y')

{

std::cout << "Which element to add to the tree?" << std::endl;

std::cin >> elem;

randomInsert(bt, elem);

std::cout << "The tree after adding:" << std::endl;

printBT(bt, "");

}

}

return 0;

}

**Файл binSTree.h:**

#pragma once

#include <iostream>

#include <string>

class binSTree

{

public:

binSTree();

~binSTree();

bool isNull();

char info;

int count, number;

binSTree\* lt;

binSTree\* rt;

};

int find(binSTree\*& bt, char el); // Функция поиска элемента в БДП

void printBT(binSTree\*&, std::string); //Функция печати БДП

void rotateRight(binSTree\*&); //Функция вращение БДП вправо

void rotateLeft(binSTree\*&); //Функция вращение БДП влево

void insertInRoot(binSTree\*&, char info); //Функция вставки элемента в корень БДП

void randomInsert(binSTree\*&, char info); //Функция рандомизированной вставки в БДП

**Файл binSTree.cpp:**

#include "binSTree.h"

binSTree::binSTree()

{

this->info = '\0';

this->count = 0;

this->number = 0;

this->lt = nullptr;

this->rt = nullptr;

}

binSTree::~binSTree()

{

delete this->lt;

delete this->rt;

}

bool binSTree::isNull()

{

return this->info == '\0';

}

/\*

\* рекурсивная функция поиска элемента в БДП

\*/

int find(binSTree\*& bt, char el)

{

if (bt == nullptr || bt->isNull()) return 0;

if (el == bt->info) return bt->count;

if (el < bt->info)

return find(bt->lt, el);

else

return find(bt->rt, el);

}

void rotateRight(binSTree\*& temp)

{

binSTree\* x;

x = temp->lt;

temp->lt = x->rt;

x->rt = temp;

temp = x;

if (temp->lt != nullptr)

{

temp->lt->number = temp->lt->count;

if (temp->lt->lt != nullptr)

temp->lt->number += temp->lt->lt->number;

if (temp->lt->rt != nullptr)

temp->lt->number += temp->lt->rt->number;

}

temp->number = temp->count;

if (temp->lt != nullptr)

temp->number += temp->lt->number;

if (temp->rt != nullptr)

temp->number += temp->rt->number;

}

void rotateLeft(binSTree\*& temp)

{

binSTree\* x;

x = temp->rt;

temp->rt = x->lt;

x->lt = temp;

temp = x;

if (temp->rt != nullptr)

{

temp->rt->number = temp->rt->count;

if (temp->rt->lt != nullptr)

temp->rt->number += temp->rt->lt->number;

if (temp->rt->rt != nullptr)

temp->rt->number += temp->rt->rt->number;

}

temp->number = temp->count;

if (temp->lt != nullptr)

temp->number += temp->lt->number;

if (temp->rt != nullptr)

temp->number += temp->rt->number;

}

/\*

\* Рекурсивная функция вставки элемента в БДП

\*/

void insertInRoot(binSTree\*& bt, char x)

{

if (bt == nullptr || bt->isNull())

{

bt = new binSTree();

bt->info = x;

bt->count = 1;

bt->number = 1;

}

else

{

if (x < bt->info)

{

insertInRoot(bt->lt, x);

rotateRight(bt);

}

else

{

if (x > bt->info)

{

insertInRoot(bt->rt, x);

rotateLeft(bt);

}

else

{

bt->count++;

bt->number++;

}

}

}

}

/\*

\* Рекурсивная функция рандомной вставки элемента в БДП

\* в зависимости от условий может вызвать как себя, так и функцию insertInRoot

\*/

void randomInsert(binSTree\*& bt, char x)

{

if (bt == nullptr || bt->isNull())

{

bt = new binSTree();

bt->info = x;

bt->count = 1;

bt->number = 1;

return;

}

if (rand() % (bt->number + 1) == 0)

{

insertInRoot(bt, x);

return;

}

else if (x < bt->info)

{

randomInsert(bt->lt, x);

}

else if (x == bt->info)

{

bt->count++;

}

else

{

randomInsert(bt->rt, x);

}

bt->number++;

}

/\*

\* Функция печати БДП

\*/

void printBT(binSTree\*& bt, std::string str)

{

if (bt == nullptr) return;

std::string \_str = str;

static bool isFirstCall = true;

if (isFirstCall)

{

std::cout << "▶ ";

isFirstCall = false;

}

std::cout << bt->info << " (" << bt->number << ")" << std::endl;

if (bt->rt != nullptr)

{

std::cout << str;

}

if (bt->lt == nullptr && bt->rt != nullptr)

{

std::cout << "└▶ ";

}

if (bt->lt != nullptr && bt->rt != nullptr)

{

std::cout << "├▶ ";

}

if (bt->lt != nullptr)

{

printBT(bt->rt, str.append("│ "));

}

else

{

printBT(bt->rt, str.append(""));

}

if (bt->lt != nullptr)

{

std::cout << \_str;

}

if (bt->lt != nullptr)

{

std::cout << "└▶ ";

}

printBT(bt->lt, \_str.append(" "));

}