

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: Демонстрация операций конструирования, поиска и вставки в
случайных БДП с рандомизацией

Студент гр. 9304

Афанасьев А

Преподаватель

Филатов Ар.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Афанасьев А.

Группа 9304

Тема работы: Демонстрация операций конструирования, поиска и вставки в случайных БДП с рандомизацией

Исходные данные:

Содержание пояснительной записки:

- Аннотация
- Содержание
- Введение
- Формальная постановка задачи
- Описание структуры данных случайное БДП с рандомизацией
- Тестирование
- Демонстрация
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 18 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 28.12.2020

Дата защиты реферата: 28.12.2020

Студент		Афанасьев А.
Преподаватель		Филатов Ар.Ю.

АННОТАЦИЯ

В данной курсовой работе производится демонстрация конструирования случайного БДП с рандомизацией и операции поиска заданного элемента e и его вставки после окончания поиска. Демонстрация происходит под средством разработанной программы на языке программирования C++ для работы с случайными БДП с рандомизацией. В программе предусмотрен вывод поясняющих сообщений для пользователя.

SUMMARY

In this course work, we demonstrate the construction of a random BST with randomization and the operation of searching for the number of occurrences of a given element e and its insertion after the search is completed. The demonstration takes place under the tool of a developed program in the C++ programming language for working with random BST with randomization. The program provides an output of explanatory messages for the user.

СОДЕРЖАНИЕ

	Введение	5
1	Формальная постановка задачи	6
2	Ход выполнения работы	7
2.1	Определение случайного БДП с рандомизацией	7
2.2	Описание структур данных и функций	7
2.3	Описание алгоритмов поиска, вставки и конструирования.	9
3	Тестирование	11
4	Демонстрация	13
4.1	Реализация демонстрации и запуск	13
4.2	Демонстрация конструирования и вставки	13
4.3	Демонстрация поиска	15
	Заключение	17
	Список использованных источников	18
	Приложение А. Исходный код тестирующего скрипта	19
	Приложение Б. Исходный код программы	21

ВВЕДЕНИЕ

Целью данной курсовой работы является изучение структуры данных случайное БДП с рандомизацией, а также алгоритма поиска в нем заданного элемента и его вставки. Также было необходимо сделать визуализацию с объяснениями, чтобы программа была пригодна для обучения.

1. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Реализовать структуру данных случайное БДП с рандомизацией.

Реализовать демонстрацию работы алгоритма поиска и вставки. Демонстрация должна полностью отражать происходящие с деревом действия и объяснять их, чтобы программу можно было использовать в обучении.

2. ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ СЛУЧАЙНОЕ БДП С РАНДОМИЗАЦИЕЙ

2.1. Определение случайного БДП с рандомизацией

Случайное БДП с рандомизацией — это структура данных, реализующая бинарное дерево поиска. Его отличия от обычного бинарного дерева поиска:

- Дерево конструируется по случайной перестановке заданной последовательности элементов.
- При вставке элемента e в дерево или его удалении из него используется генератор случайных чисел.

Подробно вставка будет описана в следующих пунктах. Обе эти модификации снижают вероятность выпадения худшего случая скорости $O(n)$, например, при поиске элемента в дереве.

2.2. Описание структур данных и функций

Для удобства чтения сократили `std::shared_ptr<>` до `sh_ptr<>`, `std::weak_ptr<>` до `wk_ptr<>` и `std::vector<>` до `vec<>`. Также сокращено `RandomBinarySearchTree` до `RBST`, а `Node` до `Node`.

- Класс `BinaryTreeNode` (далее `Node`) — шаблонный класс для узла БДП:

Поля класса:

`T data` — хранимый в узле элемент.

`sh_ptr<Node<T>> left` — указатель на корень левого поддерева;

`sh_ptr<Node<T>> right` — указатель на корень правого поддерева;

`wk_ptr<Node<T>> parent` — указатель на родителя.

Методы класса:

`const T getData()` - метод, который возвращает значение `data`;

`const size_t size()` - метод, возвращающий количество узлов, начиная от текущего.

- Класс `RandomBinarySearchTree` (далее `RBST`) — шаблонный класс случайного БДП с рандомизацией:

Поля класса:

sh_ptr<Node<T>> head – указатель на корень дерева;
size_t __treeSize – высота всего дерева;
vec<T> randomPermutation — временное поле для конструктора от последовательности. Существует только тогда, когда включен макрос *DEMO_ON*.

Ключивые методы класса:

RBST(RBST<T> &&tree) — конструктор перемещения;
RBST(const RBST<T> &tree) — конструктор копирования;
RBST<T> &operator=(RBST<T> &&tree) — оператор перемещения;
RBST<T> &operator= (const RBST<T> &tree) — оператор копирования;
const vec<T> prefixTraverse() - ЛКП обход;
const vec<T> postfixTraverse() - ПКЛ обход;
const vec<T> infixTraverse() - КЛП обход;
const size_t size() - возвращает количество элементов в дереве;
const size_t height() - возвращает высоту дерева;
const bool find(const T &val) — поиск элемента КЛП обходом;
bool empty() - возвращает true, если дерево пусто;
void erase() - очищает дерево;
void insert(const T &val) — метод вставки элемента в дерево;
void remove(const T &val) — метод удаления элемента из дерева;
const bool taskFindAndInsert(const T &val);
std::ostream &operator<<(std::ostream &out, const RBST<C> &bTree) — оператор вывода в поток;
sh_ptr<Node<T>> merge(sh_ptr<Node<T>> &ptrLeft,
sh_ptr<Node<T>> &ptrRight) — объединяет случайным образом левое и правое поддерево.

void insertAtRoot(const T &val, sh_ptr<Node<T>> &nodePtr, const sh_ptr<Node<T>> &ptrParent) — вставляет элемент в корень.

Подробнее в следующих пунктах;

void leftTreeRotate(const sh_ptr<Node<T>> &ptrNode) — реализует поворот поддерева влево;

void rightTreeRotate(const sh_ptr<Node<T>> &ptrNode) — реализует поворот поддерева вправо;

Следующие методы существуют только тогда, когда включен макрос DEMO_ON.

void printWithColors(std::ostream &out, const vec<sh_ptr<Node<T>>> &nodes, const vec<Color> &color) — метод выводящий в поток *out* дерево с покрашенными в выбранные цвета элементами;

int getKey() - метод, ждущий нажатие клавиши;

void printRandomPermutation(std::ostream &out) — метод выводящий случайную перестановку в поток *out*;

void ptrPrefixTraverse(vec<sh_ptr<Node<T>>> &vecOfNodes, const sh_ptr<Node<T>> &nodePtr) — собирает в *vecOfNodes* все узлы обходом ЛКП;

2.3. Описание алгоритмов поиска, вставки и конструирования.

Алгоритм поиска.

Поиск осуществляет метод *find(const T &val)*, который работает через рекурсивный приватный метод *recursiveFind(const T &val, const sh_ptr<Node<T>> &ptrNode)*. *recursiveFind()* в свою очередь, если не встречает элемент в узле, то спускается в левое или в правое поддерево, что зависит от величины искомого элемента. Если искомый элемент больше встреченного, то спуск в правое поддерево, иначе — в левое. Если метод встретил искомый элемент, то возвращается true.

Алгоритм вставки.

Вставку осуществляет метод `insert(const T &val)`. Метод повторяет классический алгоритм вставки в БДП, но каждый спуск производит генерацию случайного числа, от которого зависит, вставится ли этот элемент в корень на этом уровне или спустится ниже, там он повторит генерацию. Шанс вставить элемент в корень на текущем уровне всегда равен $\frac{1}{(N+1)}$. Если нужное число выпадает, то алгоритм классической вставки продолжается уже без генерации, но продолжается это в методе `insertAsRoot()`, пока элемент не вставится на свое место. Далее поворотами в обратные спуску стороны элемент поднимается до узла, в котором сгенерировалось нужное число.

Алгоритм конструирования дерева

Пользователь загружает последовательность элементов в вектор. Этот вектор случайным образом перемешивается внутри. Далее каждый элемент из получившейся перестановки по очереди вставляется в дерево методом `insert()`.

Исходный код программы представлен в приложении Б.

3. ТЕСТИРОВАНИЕ

Программу можно собрать командой *make*, после этого создается исполняемый файл *sw*.

Входные данные: на вход программе подается строка из элементов типа *char*. Затем программа принимает на вход число элементов, которые будут введены и сами эти элементы.

Тестирование производится с помощью скрипта, написанного на языке программирования Python. Для запуска скрипта для тестирования нужно запустить файл *testScript.py*, конфигурационный файл которого лежит в папке с исполняемым файлом. В конфигурационном файле можно настроить многие параметры, включая количество тестов и директорию, в которой они находятся. В тестовом файле должна находиться только лишь одна строка – сам тест. Пример вызова скрипта представлен на рисунке 1. Исходный код скрипта представлен в приложении А. Результаты тестирования представлены в таблице 1

```
strx@strxpc:~/gitreps/main/Programs/ETU/3SEM/AaDS/cw$ python testScript.py
Make sure that this script is in the same directory as the program execute file.

test0:
Input: "324io43y5u34" 1 3
CorrectAnswer: 1
Answer: 1
Result: success

test1:
Input: "324" 3 3 2 4
CorrectAnswer: 1 1 1
Answer: 1 1 1
Result: success

Total: Successes: 2. Fails: 0
```

Рисунок 1 - Пример вызова скрипта

Таблица 1. Примеры входных и выходных данных

№	Входные данные	Выходные данные
1	"324io43y5u34" 1 3	1
2	"324" 3 3 2 4	1 1 1
3	"8765456878" 8	incorrect input
4	"oifnfdnfd" 1 d	1
5	"324io43y5u34" 3 3 i o	1 1 1
6	"adsfdsakljf" 2 f a	1 1
7	"8945putr;jds" 2 2 p	0 1
8	"kjsadkfjhdsa" 5 ' ' d s t d	0 1 1 0 1
9	"iodf;fdk;fdk" 1 k	1
10	"nkfkjv kfdfdk" 2 j j	1 1

4. ДЕМОНСТРАЦИЯ

4.1. Реализация демонстрации и запуск

Для демонстрации были написаны дополнительные методы и структуры данных, которые включаются, когда включает макрос *DEMO_ON*. Соответственно, чтобы запустить программу в режиме демонстрации, нужно в Makefile добавить `-D DEMO_ON` в поле с флагами компиляции. Пользователь запускает исполняемый файл `cw` с переданными туда аргументами. Все, что требуется от пользователя для просмотра демонстрации, - это нажатие любой клавиши для перехода к следующему этапу демонстрации.

4.2. Демонстрация конструирования и вставки

Демонстрация начинается с конструирования дерева по случайной перестановке. Этот этап представлен на рисунке 2.

```
strx@strxpc:~/gitreps/main/Programs/ETU/3SEM/AaDS/cw$ ./cw "12323546765213" 2 2 0
[CONSTRUCTOR] Your input: 1 2 3 2 3 5 4 6 7 6 5 2 1 3
Need to choose a random permutation. Press any key
[CONSTRUCTOR] Random permutation: 3 5 2 1 1 5 3 2 4 6 2 6 7 3
The next step is to build a tree by inserting items from a random permutation. Press
any key
```

Рисунок 2 - Выбор случайной перестановки

После выбора перестановки начинается процесс вставки элементов в дерево из этой последовательности. Примеры разных этапов этого действия, включая поворот, представлены на рисунках 3-8. Пример окончания конструирования представлен на рисунке 9.

```
Permutation: 3 5 2 1 1 5 3 2 4 6 2 6 7 3
Need to insert 1 into the tree:
  5
 2 #
1 3 # #
```

Рисунок 3 - Пример этапа вставки

```

Permutation: 3 5 2 1 1 5 3 2 4 6 2 6 7 3
Inserting 1 into the tree
  5
 2 #
1 3 # #
[INSERT] We move to the left subtree because 1 < 2. Press any to continue

```

Рисунок 4 - Пример спуска элемента при вставке

```

[INSERT] 1 already exists in the tree
  5
 2 #
1 3 # #

```

Рисунок 5 - Пример окончания вставки существующего элемента

```

[INSERT] Random number is 5 = 5. This means that the node will be inserted here by ro
tates
Permutation: 3 5 2 1 1 5 3 2 4 6 2 6 7 3
Inserting 4 into the tree
  5
 2 #
1 3 # #
[INSERT] We move to the right subtree because 4 >= 3. Press any to continue

```

Рисунок 6 - Пример выпадения нужного числа для вставки в корень

```

Permutation: 3 5 2 1 1 5 3 2 4 6 2 6 7 3
Inserting 4 into the tree
  5
 2 #
1 3 # #
# # # 4 # # # #
[INSERT] Inserted. Press any to continue

```

Рисунок 7 - Конец классичесвкой вставки в дерево

[CONSTRUCTOR] Success

Press any key to continue

```

      1
      #
    #      2
  #      #      #      7
#      #      #      #      3      #
#      #      #      #      #      #      #      5      #      #
#####46#####
[Find] No. Move to the right subtree because 1 < 2 . Press any to continue

```

15

Looking for 2 into the tree

```

                                1
                                #
                        #           #           #           2           7
                    #   #   #   #   #   #   #   #   #   #   #   #   #   #   #
                #   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #
            #   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #
        #####
[FIND] Finded. Press any to continue
```

Рисунок 11 - Пример окончания поиска

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была реализована на языке программирования C++ структура случайного БДП с рандомизацией, а также ключевые методы для него. Была реализована логика демонстрации алгоритмов конструирования, поиска и вставки с сообщениями для пользователя в целях его обучения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рандомизированные деревья поиска URL:
<https://habr.com/ru/post/145388/> (дата обращения: 11.12.2020).

2. Рандомизированное бинарное дерево поиска URL:
http://neerc.ifmo.ru/wiki/index.php?title=%D0%A0%D0%B0%D0%BD%D0%B4%D0%BE%D0%BC%D0%B8%D0%B7%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B1%D0%B8%D0%BD%D0%B0%D1%80%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0 (дата обращения: 11.12.2020).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ТЕСТИРУЮЩЕГО СКРИПТА

Имя файла: testScript.py

```
import sys
import os

print("Make sure that this script is in the same directory as the
program execute file.\n")
if not os.path.isfile("testScript.conf"):
    f = open("testScript.conf", mode = 'w')
    f.write("ProgramName=\n")
    f.write("DirNameOfTests=\n")
    f.write("NameOfTestFile=\n")
    f.write("AmountOfTests=\n")
    f.write("DirNameOfAnswers=\n")
    f.write("NameOfAnswerFile=")
    f.close()
    print("Please fill the config file")
else:
    sys.stdin = open("testScript.conf", mode = 'r')
    programName = input()[len("programName="):]
    testDir = input()[len("DirNameOfTests="):] + '/'
    testName = input()[len("NameOfTestFile="):]
    amountOfTests = int(input()[len("AmountOfTests="):])
    correctAnsDir = input()[len("DirNameOfAnswers="):] + '/'
    correctAnsName = input()[len("NameOfAnswerFile="):]
    sys.stdin.close()

    if not os.path.isdir(testDir + "ProgramAns/"):
        os.mkdir(testDir + "ProgramAns/")
    else:
        os.system("rm -f -R " + testDir + "ProgramAns/testAns?")
    ansDirInTestDir = "ProgramAns/"
    ansName = "testAns"
    amountOfFails = 0
    amountOfSuccesses = 0
    for i in range(0, amountOfTests):
        fil = open(testDir + testName + str(i))
        inputData = fil.readline().rstrip()
        os.system("./" + programName + ' ' + inputData + " > " +
testDir + ansDirInTestDir + ansName + str(i) + " 2>&1\n")
```

```

        sys.stdin = open(testDir + ansDirInTestDir + ansName +
str(i), mode = 'r')
        answer = input()
        sys.stdin.close()

        sys.stdin = open(testDir + correctAnsDir + correctAnsName
+ str(i), mode = 'r')
        correctAnswer = input()
        sys.stdin.close()

        print(testName + str(i) + ":\nInput: " + inputData + "\
nCorrectAnswer: " + correctAnswer + "\nAnswer: " + answer + "\
nResult: ", end = "")
        if answer == correctAnswer:
            amountOfSuccesses += 1
            print("success")
        else:
            amountOfFails += 1
            print("fail")
        print('\n', end = '')
        print("Total: Successes: " + str(amountOfSuccesses) + ".
Fails: " + str(amountOfFails))

```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ

Имя файла: main.cpp

```
// включает демонстрацию
// #define DEMO_ON

#include "../libs/RandomBinarySearchTree.h"

#include <iostream>

int main(int argc, char const *argv[])
{
    size_t cntOfTests = std::stoull(std::string(argv[2]));
    if (argc >= 4 && (cntOfTests + 3 == (size_t)argc))
    {
        std::string str = argv[1];
        std::vector<char> arr;
        for (auto it = str.begin(); it != str.end(); ++it)
            arr.push_back(*it);
        RandomBinarySearchTree<char> tree(arr);
        std::stoull(std::string(argv[2]));
        for (size_t i = 0; i < std::stoull(std::string(argv[2]));
        ++i)
        {
            if (i > 0)
                std::cout << ' ';
            std::cout << tree.findAndInsert(argv[3 + i][0]);
        }
        std::cout << '\n';
    }
    else
        std::cerr << "incorrect input\n";

    return 0;
}
```

Имя файла: BinaryTreeNode.h

```
#ifndef __BINARYTREENODE__H__
```

```

#define __BINARYTREENODE__H__

#include <memory>

template <typename T>
class RandomBinarySearchTree;

template <typename T>
class BinaryTreeNode
{
    friend class RandomBinarySearchTree<T>;
    T data;
    std::shared_ptr<BinaryTreeNode<T>> left;
    std::shared_ptr<BinaryTreeNode<T>> right;
    std::weak_ptr<BinaryTreeNode<T>> parent;

    const size_t recursiveSize(const
std::shared_ptr<BinaryTreeNode<T>> &ptrNode) const;

public:
    BinaryTreeNode(const T &val, const
std::shared_ptr<BinaryTreeNode<T>> &ptrParent = nullptr, const
std::shared_ptr<BinaryTreeNode<T>> &ptrLeft = nullptr, const
std::shared_ptr<BinaryTreeNode<T>> &ptrRight = nullptr);
    const size_t size() const;
    const T getData() const;
    template <typename C>
    friend bool operator==(const BinaryTreeNode<C> &left, const
BinaryTreeNode<C> &right);

```

```

    BinaryTreeNode(BinaryTreeNode<T> &&toMove);
    BinaryTreeNode(const BinaryTreeNode<T> &toCopy);
    BinaryTreeNode<T> &operator=(BinaryTreeNode<T> &&toMove);
    BinaryTreeNode<T> &operator=(const BinaryTreeNode<T> &toMove);

    template <typename C>
        friend std::ostream &operator<<(std::ostream &out, const
BinaryTreeNode<C> &right);
};

template <typename T>
BinaryTreeNode<T> &BinaryTreeNode<T>::operator=(const
BinaryTreeNode<T> &toMove)
{
    if (this != &toMove)
    {
        this->data = toMove->data;
        this->left = toMove->left;
        this->right = toMove->right;
        this->parent = toMove->parent;
    }
    return *this;
}

template <typename T>
BinaryTreeNode<T> &BinaryTreeNode<T>::operator=(BinaryTreeNode<T>
&&toMove)
{

```

```

    if (this != &toMove)
    {
        this->data = toMove->data;
        this->left = toMove->left;
        this->right = toMove->right;
        this->parent = toMove->parent;
        toMove->left = nullptr;
        toMove->right = nullptr;
        toMove->parent = nullptr;
    }
    return *this;
}

```

```

template <typename T>
BinaryTreeNode<T>::BinaryTreeNode(BinaryTreeNode<T> &&toMove)
{
    this->data = toMove->data;
    this->left = toMove->left;
    this->right = toMove->right;
    this->parent = toMove->parent;
    toMove->left = nullptr;
    toMove->right = nullptr;
    toMove->parent = nullptr;
}

```

```

template <typename T>
BinaryTreeNode<T>::BinaryTreeNode(const BinaryTreeNode<T> &toCopy)
{
    this->data = toCopy->data;

```



```

        this->left = toCopy->left;
        this->right = toCopy->right;
        this->parent = toCopy->parent;
    }

template <typename T>
const size_t BinaryTreeNode<T>::size() const
{
    return (1 + this->recursiveSize(this->left) + this->recursiveSize(this->right));
}

template <typename T>
const size_t BinaryTreeNode<T>::recursiveSize(const
std::shared_ptr<BinaryTreeNode<T>> &ptrNode) const
{
    return (ptrNode != nullptr) ? (1 + this->recursiveSize(ptrNode->left) + this->recursiveSize(ptrNode->right)) : 0;
}

template <typename C>
std::ostream &operator<<(std::ostream &out, const
BinaryTreeNode<C> &right)
{
    out << right.data;
    return out;
}

```

```

template <typename T>
const T BinaryTreeNode<T>::getData() const
{
    return this->data;
}

template <typename T>
BinaryTreeNode<T>::BinaryTreeNode(const T &val, const
std::shared_ptr<BinaryTreeNode<T>> &ptrParent, const
std::shared_ptr<BinaryTreeNode<T>> &ptrLeft, const
std::shared_ptr<BinaryTreeNode<T>> &ptrRight) : data(val),
left(ptrLeft), right(ptrRight), parent(ptrParent) {}

template <typename C>
bool operator==(const BinaryTreeNode<C> &left, const
BinaryTreeNode<C> &right)
{
    if (left.data != right.data)
        return 0;
    else
    {
        if ((left.left == nullptr || right.left == nullptr) &&
right.left != left.left)
            return 0;
        else
        {
            if ((left.right == nullptr || right.right == nullptr)
&& right.right != left.right)
                return 0;
        }
    }
}

```

```

        else
        {
            if (left.left != nullptr && left.right == nullptr)
                return *(left.left) == *(right.left);
            else if (left.left == nullptr && left.right !=
nullptr)
                return *(left.right) == *(right.right);
            else if (left.left != nullptr && left.right !=
nullptr)
                return *(left.right) == *(right.right) &&
*(left.left) == *(right.left);
            else
                return 1;
        }
    }
}

```

```

#endif // !__BINARYTREENODE__H__

```

Имя файла: RandomBinarySearchTree.h

```

#ifndef __RANDOMBINARYSEARCHTREE__H__
#define __RANDOMBINARYSEARCHTREE__H__

```

```

#include <iostream>
#include <ostream>
#include <vector>
#include <ctime>

```

```

#include <iomanip>
#include <cmath>
#include <sstream>
#include <algorithm>

#ifdef DEMO_ON
#include <termios.h>
#include <unistd.h>

enum class Color
{
    kRED = 31,
    kGREEN = 32,
    kBLUE = 34
};

#endif //DEMO_ON

#include "BinaryTreeNode.h"

template <typename T>
class RandomBinarySearchTree
{
    std::shared_ptr<BinaryTreeNode<T>> head = nullptr;
    size_t __treeSize = 0;
    bool doubleData = 0;

#ifdef DEMO_ON
    std::vector<T> randomPermutation;

```

```
#endif //DEMO_ON
```

```
    std::shared_ptr<BinaryTreeNode<T>>
merge(std::shared_ptr<BinaryTreeNode<T>> &ptrLeft,
std::shared_ptr<BinaryTreeNode<T>> &ptrRight);
    void recursivePrefixTraverse(std::vector<T> &vectorOfNodes,
const std::shared_ptr<BinaryTreeNode<T>> &nodePtr) const;
    void recursivePostfixTraverse(std::vector<T> &vectorOfNodes,
const std::shared_ptr<BinaryTreeNode<T>> &nodePtr) const;
    void recursiveInfixTraverse(std::vector<T> &vectorOfNodes,
const std::shared_ptr<BinaryTreeNode<T>> &nodePtr) const;
    void recursiveInsert(const T &val,
std::shared_ptr<BinaryTreeNode<T>> &ptrNode, const
std::shared_ptr<BinaryTreeNode<T>> &ptrParent = nullptr);
    void recursiveRemove(const T &val,
std::shared_ptr<BinaryTreeNode<T>> &ptrNode);
    const size_t recursiveSize(const
std::shared_ptr<BinaryTreeNode<T>> &ptrNode) const;
    std::shared_ptr<BinaryTreeNode<T>> recursiveCopy(const
std::shared_ptr<BinaryTreeNode<T>> &ptrNodeToCopy, const
std::shared_ptr<BinaryTreeNode<T>> &ptrParent = nullptr);
    void insertAtRoot(const T &val,
std::shared_ptr<BinaryTreeNode<T>> &nodePtr, const
std::shared_ptr<BinaryTreeNode<T>> &ptrParent);
    void
recursiveGetLevel(std::vector<std::shared_ptr<BinaryTreeNode<T>>>
&vec, const std::shared_ptr<BinaryTreeNode<T>> &ptrNode, const
size_t &level, const size_t &curLevel) const;
```

```

        const size_t recursiveHeight(const
std::shared_ptr<BinaryTreeNode<T>> &ptrNode) const;
        void leftTreeRotate(const std::shared_ptr<BinaryTreeNode<T>>
&ptrNode);
        void rightTreeRotate(const std::shared_ptr<BinaryTreeNode<T>>
&ptrNode);
        const bool recursiveFind(const T &val, const
std::shared_ptr<BinaryTreeNode<T>> &ptrNode) const;
#ifdef DEMO_ON
        void printWithColors(std::ostream &out, const
std::vector<std::shared_ptr<BinaryTreeNode<T>>> &nodes, const
std::vector<Color> &color) const;
        int getKey() const;
        void printRandomPermutation(std::ostream &out) const;
        void
ptrPrefixTraverse(std::vector<std::shared_ptr<BinaryTreeNode<T>>>
&vectorOfNodes, const std::shared_ptr<BinaryTreeNode<T>> &nodePtr)
const;

#endif //DEMO_ON
public:
    ~RandomBinarySearchTree() = default;
    RandomBinarySearchTree();
    RandomBinarySearchTree(std::vector<T> elems);
    RandomBinarySearchTree(RandomBinarySearchTree<T> &&tree);
    RandomBinarySearchTree(const RandomBinarySearchTree<T> &tree);
    RandomBinarySearchTree<T> &operator=(RandomBinarySearchTree<T>
&&tree);

```

```

    RandomBinarySearchTree<T> &operator=(const
RandomBinarySearchTree<T> &tree);

    const std::vector<T> prefixTraverse() const;
    const std::vector<T> postfixTraverse() const;
    const std::vector<T> infixTraverse() const;
    const std::vector<std::shared_ptr<BinaryTreeNode<T>>>
getLevel(const size_t &level) const;
    const size_t size() const;
    const size_t height() const;
    const bool find(const T &val) const;
    bool empty() const;
    void erase();
    void insert(const T &val);
    void remove(const T &val);
    const bool findAndInsert(const T &val);

    template <typename C>
        friend std::ostream &operator<<(std::ostream &out, const
RandomBinarySearchTree<C> &bTree);
};

#ifdef DEMO_ON

template <typename T>
void
RandomBinarySearchTree<T>::ptrPrefixTraverse(std::vector<std::shar
ed_ptr<BinaryTreeNode<T>>> &vectorOfNodes, const
std::shared_ptr<BinaryTreeNode<T>> &nodePtr) const
{

```

```

    if (nodePtr != nullptr)
    {
        vectorOfNodes.push_back(nodePtr);
        this->ptrPrefixTraverse(vectorOfNodes, nodePtr->left);
        this->ptrPrefixTraverse(vectorOfNodes, nodePtr->right);
    }
}

```

```

template <typename T>
void
RandomBinarySearchTree<T>::printRandomPermutation(std::ostream
&out) const
{
    for (auto it = this->randomPermutation.begin(); it != this-
>randomPermutation.end(); ++it)
        out << *it << ' ';
}

```

```

template <typename T>
int RandomBinarySearchTree<T>::getKey() const
{
    int pressedKey = 0;
    termios oldt, newt; // oldt - состояние терминала до приема
клавиши, newt - во время приема
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= static_cast<tcflag_t>(~(ICANON | ECHO));
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    pressedKey = getchar();
}

```



```

        tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
        return pressedKey;
    }

template <typename T>
void RandomBinarySearchTree<T>::printWithColors(std::ostream &out,
const std::vector<std::shared_ptr<BinaryTreeNode<T>>> &nodes,
const std::vector<Color> &colors) const
{
    std::vector<std::vector<std::shared_ptr<BinaryTreeNode<T>>>>
vecOfDatasInCurrentLevel;

    const size_t h = this->height();
    size_t maxLen = 0;

    for (size_t currentLevel = 0; currentLevel < h; +
+currentLevel)
    {
        vecOfDatasInCurrentLevel.push_back(this-
>getLevel(currentLevel));
        for (auto it =
vecOfDatasInCurrentLevel[currentLevel].begin(); it !=
vecOfDatasInCurrentLevel[currentLevel].end(); ++it)
        {
            if ((*it) != nullptr)
            {
                std::ostringstream strToCheckLen;
                strToCheckLen << (*it)->getData();
                const size_t newLen =
strToCheckLen.str().length();

```

```

        if (newLen > maxLen)
            maxLen = newLen;
    }
}

for (size_t currentLevel = 0; currentLevel < h; ++currentLevel)
{
    if (currentLevel > 0)
        out << '\n';
    for (auto it =
vecOfDatasInCurrentLevel[currentLevel].begin(); it !=
vecOfDatasInCurrentLevel[currentLevel].end(); ++it)
    {
        bool isColored = 0;
        for (size_t indexOfColoredNode = 0; indexOfColoredNode
< nodes.size(); ++indexOfColoredNode)
            if (nodes[indexOfColoredNode] == *it)
            {
                isColored = 1;
                out << "\e[" <<
(int)colors[indexOfColoredNode] << "m";
                break;
            }

        if (it ==
vecOfDatasInCurrentLevel[currentLevel].begin())

```

```

        out << std::setw((1 << (h - 1 - currentLevel)) *
maxLen);
    else
        out << std::setw((1 << (h - currentLevel)) *
maxLen);

    if (*it == nullptr)
        out << '#';
    else
    {
        out << (*it)->getData();
        if (isColored == 1)
            out << "\e[0m";
    }
}
}
}
#endif //DEMO_ON

```

```

template <typename T>
void RandomBinarySearchTree<T>::erase()
{
    this->head = nullptr;
    this->__treeSize = 0;
}

template <typename T>
const bool RandomBinarySearchTree<T>::findAndInsert(const T &val)
{
#ifdef DEMO_ON

```

```

        std::cout << "Find and insert " << val << '\n';
    #endif // DEMO_ON

    const bool cnt = this->find(val);
    this->insert(val);
    return cnt;
}

template <typename T>
RandomBinarySearchTree<T>::RandomBinarySearchTree(std::vector<T>
elems)
{
    #ifndef DEMO_ON
        std::random_shuffle(elems.begin(), elems.end());
        for (auto it = elems.begin(); it != elems.end(); ++it)
            this->insert(*it);
    #endif

    #ifdef DEMO_ON
        this->randomPermutation = elems;
        std::cout << "[CONSTRUCTOR] Your input: ";
        this->printRandomPermutation(std::cout);
        std::cout << '\n';
        std::cout << "Need to choose a random permutation. Press any
key\n";
        this->getKey();
        std::random_shuffle(this->randomPermutation.begin(), this-
>randomPermutation.end());
        std::cout << "[CONSTRUCTOR] Random permutation: ";
        this->printRandomPermutation(std::cout);
    #endif
}

```

```

        std::cout << '\n';
        std::cout << "The next step is to build a tree by inserting
items from a random permutation. Press any key\n";
        this->getKey();
        system("clear");

        size_t sizeOfPermutation = this->randomPermutation.size();
        for (size_t it = 0; it < sizeOfPermutation; ++it)
        {
            this->insert(this->randomPermutation[it]);
        }
        std::cout << "[CONSTRUCTOR] Success\n"
            << *this << "\nPress any key to continue\n";
        this->getKey();
        system("clear");
    #endif
}

template <typename T>
const bool RandomBinarySearchTree<T>::find(const T &val) const
{
    return this->recursiveFind(val, this->head);
}

template <typename T>
const bool RandomBinarySearchTree<T>::recursiveFind(const T &val,
const std::shared_ptr<BinaryTreeNode<T>> &ptrNode) const
{
    if (ptrNode != nullptr)

```

```

        {
            if (ptrNode->getData() == val)
            {
#ifdef DEMO_ON
                std::cout << "Looking for " << val << " into the tree\
n";

                std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;

                nodesToColor.push_back(ptrNode);
                std::vector<Color> colors;
                colors.push_back(Color::kGREEN);
                this->printWithColors(std::cout, nodesToColor,
colors);

                std::cout << "\n[FIND] Finded. Press any to continue\
n";

                this->getKey();
                system("clear");
#endif // DEMO_ON
                return 1;
            }
            else if (ptrNode->getData() < val)
            {
#ifdef DEMO_ON
                std::cout << "Looking for " << val << " into the tree\
n";

                std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;

                nodesToColor.push_back(ptrNode);
                std::vector<Color> colors;

```

```

        colors.push_back(Color::kRED);
        this->printWithColors(std::cout, nodesToColor,
colors);

        std::cout << "\n[FIND] No. Move to the right subtree
because " << ptrNode->getData() << " < " << val << " . Press any
to continue\n";

        this->getKey();
        system("clear");
#endif // DEMO_ON

        return this->recursiveFind(val, ptrNode->right);
    }
    else
    {
#ifdef DEMO_ON
        std::cout << "Looking for " << val << " into the tree\
n";

        std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;
        nodesToColor.push_back(ptrNode);
        std::vector<Color> colors;
        colors.push_back(Color::kRED);
        this->printWithColors(std::cout, nodesToColor,
colors);

        std::cout << "\n[FIND] No. Move to the left subtree
because " << ptrNode->getData() << " > " << val << " . Press any
to continue\n";

        this->getKey();
        system("clear");

```

```

#endif // DEMO_ON

        return this->recursiveFind(val, ptrNode->left);
    }
}
else
{
#ifdef DEMO_ON
    std::cout << "Looking for " << val << " into the tree\n";
    std::cout << *this;
    std::cout << "\n[FIND] No . Press any to continue\n";
    this->getKey();
    system("clear");
#endif // DEMO_ON
    return 0;
}
}

```

```

template <typename T>
const size_t RandomBinarySearchTree<T>::recursiveHeight(const
std::shared_ptr<BinaryTreeNode<T>> &ptrNode) const
{
    return (ptrNode != nullptr) ? (1 +
std::max<size_t>(recursiveHeight(ptrNode->left),
recursiveHeight(ptrNode->right))) : 0;
}

```

```

template <typename T>
const size_t RandomBinarySearchTree<T>::height() const

```



```

{
    return this->recursiveHeight(this->head);
}

template <typename T>
const std::vector<std::shared_ptr<BinaryTreeNode<T>>>
RandomBinarySearchTree<T>::getLevel(const size_t &level) const
{
    std::vector<std::shared_ptr<BinaryTreeNode<T>>> vecOfDatas;
    this->recursiveGetLevel(vecOfDatas, this->head, level, 0);
    return vecOfDatas;
}

template <typename T>
void
RandomBinarySearchTree<T>::recursiveGetLevel(std::vector<std::shared_ptr<BinaryTreeNode<T>>> &vec, const
std::shared_ptr<BinaryTreeNode<T>> &ptrNode, const size_t &level,
const size_t &curLevel) const
{
    std::shared_ptr<BinaryTreeNode<T>> left = nullptr;
    std::shared_ptr<BinaryTreeNode<T>> right = nullptr;
    std::shared_ptr<BinaryTreeNode<T>> root = nullptr;
    if (ptrNode != nullptr)
    {
        left = ptrNode->left;
        right = ptrNode->right;
        root = ptrNode;
    }
}

```

```

        if (curLevel == level)
            vec.push_back(root);
        else
        {
            this->recursiveGetLevel(vec, left, level, curLevel + 1);
            this->recursiveGetLevel(vec, right, level, curLevel + 1);
        }
    }
}

```

```

template <typename T>
void RandomBinarySearchTree<T>::leftTreeRotate(const
std::shared_ptr<BinaryTreeNode<T>> &ptrNode)
{
    if (ptrNode != nullptr)
    {
#ifdef DEMO_ON
        std::cout << "[LEFT ROTATE] Rotating right subtree to
left\n";

        std::cout << "The root of the \e[32mright\e[0m subtree,
along with the lower part, must be inserted in place of the \
\e[34mcurrent root\e[0m.\nThe \e[34mprevious root\e[0m, along with
its entire left subtree, must be attached to the new root on the
left.\nThe \e[31mprevious left\e[0m subtree of the new root must
be inserted to the root of the new left subtree (previous \
\e[34mcurrent root\e[0m) on the right.\n";

        std::vector<std::shared_ptr<BinaryTreeNode<T>>> nodes;
        std::vector<Color> colors;
        if (ptrNode->right != nullptr)
        {

```

```

        this->ptrPrefixTraverse(nodes, ptrNode->right->right);
        nodes.push_back(ptrNode->right);
        for (size_t it = 0; it < nodes.size(); ++it)
            colors.push_back(Color::kGREEN);
        this->ptrPrefixTraverse(nodes, ptrNode->right->left);
        for (size_t it = colors.size(); it < nodes.size(); +
+it)
            colors.push_back(Color::kRED);
    }
    nodes.push_back(ptrNode);
    colors.push_back(Color::kBLUE);
    this->printWithColors(std::cout, nodes, colors);
    std::cout << '\n';
    this->getKey();
#endif //DEMO_ON

    std::shared_ptr<BinaryTreeNode<T>> newRoot = ptrNode-
>right;
    ptrNode->right = newRoot->left;
    if (ptrNode->right != nullptr)
        ptrNode->right->parent = ptrNode;
    newRoot->parent = ptrNode->parent;
    newRoot->left = ptrNode;
    newRoot->left->parent = newRoot;

    if (newRoot->parent.lock() != nullptr)
    {
        if (newRoot->parent.lock()->left == ptrNode)
            newRoot->parent.lock()->left = newRoot;
    }

```

```

        if (newRoot->parent.lock()->right == ptrNode)
            newRoot->parent.lock()->right = newRoot;
    }

    if (ptrNode == this->head)
        this->head = newRoot;

#ifdef DEMO_ON
        std::cout << "[LEFT ROTATE] Success\n";
        this->printWithColors(std::cout, nodes, colors);
        std::cout << '\n';
        this->getKey();
        system("clear");
#endif //DEMO_ON
    }
}

template <typename T>
void RandomBinarySearchTree<T>::rightTreeRotate(const
std::shared_ptr<BinaryTreeNode<T>> &ptrNode)
{
    if (ptrNode != nullptr)
    {
#ifdef DEMO_ON
        std::cout << "[RIGHT ROTATE] Rotating left subtree to
right\n";

        std::cout << "The root of the \e[32mleft\e[0m subtree,
along with the lower part, must be inserted in place of the \
\e[34mcurrent root\e[0m.\nThe \e[34mprevious root\e[0m, along with
its entire right subtree, must be attached to the new root on the

```

right.\n\nThe \nprevious right subtree of the new root must be inserted to the root of the new right subtree (previous \ncurrent root) on the left.\n";

```

        std::vector<std::shared_ptr<BinaryTreeNode<T>>> nodes;
        std::vector<Color> colors;
        if (ptrNode->left != nullptr)
        {
            this->ptrPrefixTraverse(nodes, ptrNode->left->left);
            nodes.push_back(ptrNode->left);
            for (size_t it = 0; it < nodes.size(); ++it)
                colors.push_back(Color::kGREEN);
            this->ptrPrefixTraverse(nodes, ptrNode->left->right);
            for (size_t it = colors.size(); it < nodes.size(); +
+it)
                colors.push_back(Color::kRED);
        }
        nodes.push_back(ptrNode);
        colors.push_back(Color::kBLUE);
        this->printWithColors(std::cout, nodes, colors);
        std::cout << '\n';
        this->getKey();
        system("clear");
#endif //DEMO_ON

        std::shared_ptr<BinaryTreeNode<T>> newRoot = ptrNode-
>left;

        ptrNode->left = newRoot->right;
        if (ptrNode->left != nullptr)
            ptrNode->left->parent = ptrNode;
        newRoot->parent = ptrNode->parent;

```

```

newRoot->right = ptrNode;
newRoot->right->parent = newRoot;

if (newRoot->parent.lock() != nullptr)
{
    if (newRoot->parent.lock()->left == ptrNode)
        newRoot->parent.lock()->left = newRoot;

    if (newRoot->parent.lock()->right == ptrNode)
        newRoot->parent.lock()->right = newRoot;
}
if (ptrNode == this->head)
    this->head = newRoot;

#ifdef DEMO_ON
    std::cout << "[RIGHT ROTATE] Success\n";
    this->printWithColors(std::cout, nodes, colors);
    std::cout << '\n';
    this->getKey();
    system("clear");
#endif //DEMO_ON
}

}

template <typename T>
const size_t RandomBinarySearchTree<T>::size() const
{
    return this->__treeSize;
}

```

```

template <typename T>
const size_t RandomBinarySearchTree<T>::recursiveSize(const
std::shared_ptr<BinaryTreeNode<T>> &ptrNode) const
{
    return (ptrNode != nullptr) ? (1 + this-
>recursiveSize(ptrNode->left) + this->recursiveSize(ptrNode-
>right)) : 0;
}

```

```

template <typename T>
RandomBinarySearchTree<T>::RandomBinarySearchTree(const
RandomBinarySearchTree<T> &tree) : head(this-
>recursiveCopy(tree.head, nullptr)) {}

```

```

template <typename T>
RandomBinarySearchTree<T>
&RandomBinarySearchTree<T>::operator=(const
RandomBinarySearchTree<T> &tree)
{
    if (this != &tree)
        this->head = this->recursiveCopy(tree.head, nullptr);
    return *this;
}

```

```

template <typename T>
std::shared_ptr<BinaryTreeNode<T>>
RandomBinarySearchTree<T>::recursiveCopy(const

```

```

std::shared_ptr<BinaryTreeNode<T>> &ptrNodeToCopy, const
std::shared_ptr<BinaryTreeNode<T>> &ptrParent)
{
    if (ptrNodeToCopy != nullptr)
    {
        std::shared_ptr<BinaryTreeNode<T>> newObj(new
BinaryTreeNode<T>(ptrNodeToCopy->getData(), ptrParent));
        newObj->left = recursiveCopy(ptrNodeToCopy->left, newObj);
        newObj->right = recursiveCopy(ptrNodeToCopy->right,
newObj);
        return newObj;
    }
    else
        return nullptr;
}

```

```

template <typename T>
RandomBinarySearchTree<T>::RandomBinarySearchTree(RandomBinarySear
chTree<T> &&tree) : head(std::move(tree.head)) {}

```

```

template <typename T>
RandomBinarySearchTree<T>
&RandomBinarySearchTree<T>::operator=(RandomBinarySearchTree<T>
&&tree)
{
    if (&tree != this)
        this->head = std::move(tree.head);
    return *this;
}

```



```

template <typename T>
void RandomBinarySearchTree<T>::remove(const T &val)
{
    this->recursiveRemove(val, this->head);
}

template <typename T>
void RandomBinarySearchTree<T>::recursiveRemove(const T &val,
std::shared_ptr<BinaryTreeNode<T>> &ptrNode)
{
    if (ptrNode != nullptr)
    {
        if (ptrNode->getData() < val)
            recursiveRemove(val, ptrNode->right);
        else if (ptrNode->getData() > val)
            recursiveRemove(val, ptrNode->left);
        else
        {
            std::shared_ptr<BinaryTreeNode<T>> ptrParent =
ptrNode->parent.lock();
            std::shared_ptr<BinaryTreeNode<T>> ptrTmp = this-
>merge(ptrNode->left, ptrNode->right);
            ptrNode->parent = ptrParent;
            if (ptrTmp == nullptr)
            {
                if (ptrParent != nullptr)
                {
                    if (ptrParent->right == ptrNode)

```

```

        ptrParent->right = nullptr;
    else
        ptrParent->left = nullptr;
    }
    if (this->head == ptrNode)
        this->head = nullptr;
    }
    ptrNode = ptrTmp;
    --(this->__treeSize);
}
}
}

```

```

template <typename T>
std::shared_ptr<BinaryTreeNode<T>>
RandomBinarySearchTree<T>::merge(std::shared_ptr<BinaryTreeNode<T>
> &ptrLeft, std::shared_ptr<BinaryTreeNode<T>> &ptrRight)
{
    std::shared_ptr<BinaryTreeNode<T>> ptrNode = nullptr;
    const size_t leftSize = (ptrLeft == nullptr) ? 0 : ptrLeft-
>size();
    const size_t rightSize = (ptrRight == nullptr) ? 0 : ptrRight-
>size();
    const size_t totalSize = leftSize + rightSize;
    if (totalSize != 0)
    {
        srand(clock());
        const size_t randNum = 1 + ((size_t)rand() % totalSize);
        if (randNum <= leftSize)

```

```

        {
            ptrNode = ptrLeft;
            ptrNode->right = merge(ptrNode->right, ptrRight);
        }
        else
        {
            ptrNode = ptrRight;
            ptrNode->left = merge(ptrLeft, ptrNode->left);
        }
    }
    return ptrNode;
}

```

```

template <typename T>
void RandomBinarySearchTree<T>::insert(const T &val)
{
#ifdef DEMO_ON
    std::cout << "Permutation: ";
    this->printRandomPermutation(std::cout);
    std::cout << '\n';
    std::cout << "Need to insert " << val << " into the";
    if (!(this->empty()))
        std::cout << " tree:\n"
                    << *this << '\n';
    else
        std::cout << " empty tree.";
    this->getKey();
    system("clear");
#endif //DEMO_ON
}

```

```

        this->recursiveInsert(val, this->head);
        this->doubleData = 0;
    }

template <typename T>
void RandomBinarySearchTree<T>::recursiveInsert(const T &val,
std::shared_ptr<BinaryTreeNode<T>> &ptrNode, const
std::shared_ptr<BinaryTreeNode<T>> &ptrParent)
{
    if (ptrNode == nullptr)
    {
        ptrNode = std::shared_ptr<BinaryTreeNode<T>>(new
BinaryTreeNode<T>(val, ptrParent));
        ++(this->__treeSize);
#ifdef DEMO_ON
        std::cout << "Permutation: ";
        this->printRandomPermutation(std::cout);
        std::cout << '\n';
        std::cout << "Inserting " << val << " into the tree\n";
        std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;
        std::vector<Color> colors;
        colors.push_back(Color::kGREEN);
        nodesToColor.push_back(ptrNode);
        this->printWithColors(std::cout, nodesToColor, colors);
        std::cout << "\n[INSERT] Inserted. Press any to continue\
n";

        this->getKey();
        system("clear");

```

```

#endif //DEMO_ON
    }
    else
    {
        const size_t treeSize = this->size();
        srand(clock());
        const size_t randNum = 1 + (size_t)rand() % (treeSize +
1);

        if (randNum == treeSize + 1)
        {
#ifdef DEMO_ON
            std::cout << "[INSERT] Random number is " << randNum
<< " = " << treeSize + 1 << ". This means that the node will be
inserted here by rotates\n";
#endif //DEMO_ON
            this->insertAtRoot(val, ptrNode, ptrParent);
        }
        else
        {
#ifdef DEMO_ON
            std::cout << "[INSERT] Random number is " << randNum
<< " != " << treeSize + 1 << '\n';
#endif //DEMO_ON
            if (ptrNode->getData() < val)
            {
#ifdef DEMO_ON

                std::cout << "Permutation: ";

```

```

        this->printRandomPermutation(std::cout);
        std::cout << '\n';
        std::cout << "Inserting " << val << " into the
tree\n";

        std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;

        std::vector<Color> colors;
        colors.push_back(Color::kRED);
        nodesToColor.push_back(ptrNode);
        this->printWithColors(std::cout, nodesToColor,
colors);

        std::cout << "\n[INSERT] We move to the right
subtree because " << val << " >= " << ptrNode->getData() << ".
Press any to continue\n";

        this->getKey();
        system("clear");
#endif //DEMO_ON

        this->recursiveInsert(val, ptrNode->right,
ptrNode);
    }
    else if (val < ptrNode->getData())
    {
#ifdef DEMO_ON
        std::cout << "Permutation: ";
        this->printRandomPermutation(std::cout);
        std::cout << '\n';
        std::cout << "Inserting " << val << " into the
tree\n";

```

```

        std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;

        std::vector<Color> colors;
        colors.push_back(Color::kRED);
        nodesToColor.push_back(ptrNode);
        this->printWithColors(std::cout, nodesToColor,
colors);

        std::cout << "\n[INSERT] We move to the left
subtree because " << val << " < " << ptrNode->getData() << ".
Press any to continue\n";

        this->getKey();
        system("clear");
#endif //DEMO_ON

        this->recursiveInsert(val, ptrNode->left,
ptrNode);
    }
#ifdef DEMO_ON
    else
    {
        std::cout << "[INSERT] " << val << " already
exists in the tree\n";

        std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;

        std::vector<Color> colors;
        colors.push_back(Color::kBLUE);
        nodesToColor.push_back(ptrNode);
        this->printWithColors(std::cout, nodesToColor,
colors);

        this->getKey();

```

```

        system("clear");
    }
#endif

    }

}

template <typename T>
void RandomBinarySearchTree<T>::insertAtRoot(const T &val,
std::shared_ptr<BinaryTreeNode<T>> &ptrNode, const
std::shared_ptr<BinaryTreeNode<T>> &ptrParent)
{
    if (ptrNode == nullptr)
    {
        ptrNode = std::shared_ptr<BinaryTreeNode<T>>(new
BinaryTreeNode<T>(val, ptrParent));
        ++(this->__treeSize);
#ifdef DEMO_ON
        std::cout << "Permutation: ";
        this->printRandomPermutation(std::cout);
        std::cout << '\n';
        std::cout << "Inserting " << val << " into the tree\n";
        std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;
        std::vector<Color> colors;
        colors.push_back(Color::kGREEN);
        nodesToColor.push_back(ptrNode);
        this->printWithColors(std::cout, nodesToColor, colors);

```



```

        std::cout << "\n[INSERT] Inserted. Press any to continue\n";

        this->getKey();
        system("clear");
    #endif //DEMO_ON
    }
    else
    {
        if (ptrNode->getData() < val)
        {
            #ifdef DEMO_ON
                std::cout << "Permutation: ";
                this->printRandomPermutation(std::cout);
                std::cout << '\n';
                std::cout << "Inserting " << val << " into the tree\n";

                std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;

                std::vector<Color> colors;
                colors.push_back(Color::kRED);
                nodesToColor.push_back(ptrNode);
                this->printWithColors(std::cout, nodesToColor,
colors);

                std::cout << "\n[INSERT] We move to the right subtree
because " << val << " >= " << ptrNode->getData() << ". Press any
to continue\n";

                this->getKey();
                system("clear");
            #endif //DEMO_ON
        }
    }
}

```

```

        this->insertAtRoot(val, ptrNode->right, ptrNode);
        if (this->doubleData == 0)
            this->leftTreeRotate(ptrNode);
    }
    else if (val < ptrNode->getData())
    {
#ifdef DEMO_ON
        std::cout << "Permutation: ";
        this->printRandomPermutation(std::cout);
        std::cout << '\n';
        std::cout << "Inserting " << val << " into the tree\
n";

        std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;

        std::vector<Color> colors;
        colors.push_back(Color::kRED);
        nodesToColor.push_back(ptrNode);
        this->printWithColors(std::cout, nodesToColor,
colors);

        std::cout << "\n[INSERT] We move to the left subtree
because " << val << " < " << ptrNode->getData() << ". Press any to
continue\n";

        this->getKey();
        system("clear");
#endif //DEMO_ON

        this->insertAtRoot(val, ptrNode->left, ptrNode);
        if (this->doubleData == 0)
            this->rightTreeRotate(ptrNode);
    }
}

```

```

        else
        {
            this->doubleData = 1;
#ifdef DEMO_ON
            std::cout << "[INSERT] " << val << " already exists in
the tree\n";
            std::vector<std::shared_ptr<BinaryTreeNode<T>>>
nodesToColor;
            std::vector<Color> colors;
            colors.push_back(Color::kBLUE);
            nodesToColor.push_back(ptrNode);
            this->printWithColors(std::cout, nodesToColor,
colors);
            this->getKey();
            system("clear");
#endif
        }
    }
}

template <typename T>
std::ostream &operator<<(std::ostream &out, const
RandomBinarySearchTree<T> &bTree)
{
    std::vector<std::vector<std::shared_ptr<BinaryTreeNode<T>>>>
vecOfDatasInCurrentLevel;
    const size_t h = bTree.height();
    size_t maxLen = 0;

```

```

        for (size_t currentLevel = 0; currentLevel < h; +
+currentLevel)
        {

vecOfDatasInCurrentLevel.push_back(bTree.getLevel(currentLevel));

            for (auto it =
vecOfDatasInCurrentLevel[currentLevel].begin(); it !=
vecOfDatasInCurrentLevel[currentLevel].end(); ++it)
            {
                if ((*it) != nullptr)
                {
                    std::ostringstream strToCheckLen;
                    strToCheckLen << (*it)->getData();
                    const size_t newLen =
strToCheckLen.str().length();
                    if (newLen > maxLen)
                        maxLen = newLen;
                }
            }
        }

        for (size_t currentLevel = 0; currentLevel < h; +
+currentLevel)
        {
            if (currentLevel > 0)
                out << '\n';

            for (auto it =
vecOfDatasInCurrentLevel[currentLevel].begin(); it !=
vecOfDatasInCurrentLevel[currentLevel].end(); ++it)
            {

```

```

        if (it ==
vecOfDatasInCurrentLevel[currentLevel].begin())
            out << std::setw((1 << (h - 1 - currentLevel)) *
maxLen);
        else
            out << std::setw((1 << (h - currentLevel)) *
maxLen);
        if (*it == nullptr)
            out << '#';
        else
            out << (*it)->getData();
    }
}
return out;
}

```

```

template <typename T>
RandomBinarySearchTree<T>::RandomBinarySearchTree()
{
    this->head = std::unique_ptr<BinaryTreeNode<T>>(nullptr);
    this->__treeSize = 0;
}

```

```

template <typename T>
const std::vector<T> RandomBinarySearchTree<T>::prefixTraverse()
const
{
    std::vector<T> vectorOfNodes;
    recursivePrefixTraverse(vectorOfNodes, this->head);
}

```

```

        return vectorOfNodes;
    }

template <typename T>
const std::vector<T> RandomBinarySearchTree<T>::postfixTraverse()
const
{
    std::vector<T> vectorOfNodes;
    recursivePostfixTraverse(vectorOfNodes, this->head);
    return vectorOfNodes;
}

template <typename T>
const std::vector<T> RandomBinarySearchTree<T>::infixTraverse()
const
{
    std::vector<T> vectorOfNodes;
    recursiveInfixTraverse(vectorOfNodes, this->head);
    return vectorOfNodes;
}

template <typename T>
void
RandomBinarySearchTree<T>::recursivePrefixTraverse(std::vector<T>
&vectorOfNodes, const std::shared_ptr<BinaryTreeNode<T>> &nodePtr)
const
{
    if (nodePtr != nullptr)
    {

```

```

        vectorOfNodes.push_back(nodePtr->getData());
        this->recursivePrefixTraverse(vectorOfNodes, nodePtr-
>left);
        this->recursivePrefixTraverse(vectorOfNodes, nodePtr-
>right);
    }
}

```

```

template <typename T>
void
RandomBinarySearchTree<T>::recursivePostfixTraverse(std::vector<T>
&vectorOfNodes, const std::shared_ptr<BinaryTreeNode<T>> &nodePtr)
const
{
    if (nodePtr != nullptr)
    {
        this->recursivePostfixTraverse(vectorOfNodes, nodePtr-
>left);
        this->recursivePostfixTraverse(vectorOfNodes, nodePtr-
>right);
        vectorOfNodes.push_back(nodePtr->getData());
    }
}

```

```

template <typename T>
void
RandomBinarySearchTree<T>::recursiveInfixTraverse(std::vector<T>
&vectorOfNodes, const std::shared_ptr<BinaryTreeNode<T>> &nodePtr)
const

```

```

{
    if (nodePtr != nullptr)
    {
        this->recursiveInfixTraverse(vectorOfNodes, nodePtr-
>left);
        vectorOfNodes.push_back(nodePtr->getData());
        this->recursiveInfixTraverse(vectorOfNodes, nodePtr-
>right);
    }
}

template <typename T>
bool RandomBinarySearchTree<T>::empty() const
{
    return (this->head == nullptr) ? 1 : 0;
}

#endif //!__RANDOMBINARYSEARCHTREE__H__

```