Gebze Technical University

Computer Engineering

CSE222-2021-SPRING

Homework-7 Report

Ferdi Sönmez

161044046

Part1:

1)INTRODUCTİON

1.1)Problem Definition

In this homework , we are asked to implement the NavigableSet interface from AVL tree and SkipList and add some functions and write them.

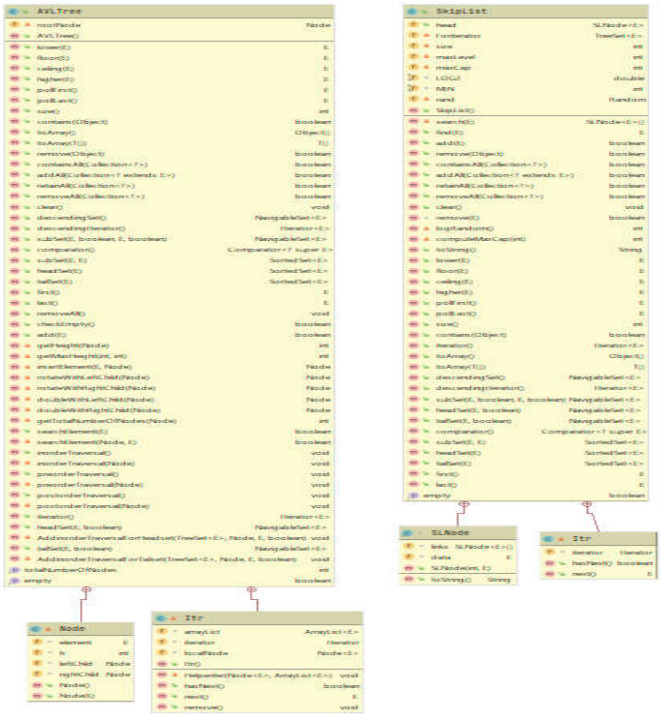
1.2- System Requirements

From the definition of the problem, the solution to this problem is solved by writing the AVL tree and SkipList structure and implementing the NavigableSet interface to them.

1.2.1-Users of the System

There is only 1 user in the system and can perform all the operations requested in the problem definition.

2) Class Diagram



3) Problem Solution Approach

Avl tree structure was created and NavigableSet interface was implemented to this structure.

Skiplist: Insert ,Delete, descendingIterator

AVLTree: insert, iterator, headSet,tailSet

4)Test Case

a)SkipList

```
3
4   public class Main {
5       public static void main(String[] args) {
6           SkipList<Integer> myskiplist = new SkipList<Integer>();
7           AVLTree<Integer> myavlTree = new AVLTree<Integer>();
8           System.out.println("*********SkipList*************");
9           myskiplist.add(60);
10          myskiplist.add(70);
11          myskiplist.add(80);
12          myskiplist.add(90);
13          myskiplist.add(100);
14          Iterator skiplistiterator=myskiplist.descendingIterator();
15          while (skiplistiterator.hasNext()){
16              System.out.println(skiplistiterator.next());
```

```
Run:  Main
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Progr
*********SkipList*************
100
90
80
70
60
```

b)AvlTree

```
17          }
18          System.out.println("************AVLTREE***************");
19          myavlTree.add(5);
20          myavlTree.add(20);
21          myavlTree.add(30);
22          myavlTree.add(40);
23          myavlTree.add(50);
24          Iterator iter=myavlTree.iterator();
25          System.out.println("FirsElement:"+iter.next());
26          System.out.println("SecondElement:"+iter.next());
27          System.out.println("****HeadSet****");
28          NavigableSet<Integer> ns1=myavlTree.headSet( toElement: 30, inclusive: false);
29          Iterator iterns1=ns1.iterator();
30          while (iterns1.hasNext()){
31              System.out.print(iterns1.next()+" ");
32          }
33          System.out.println("\n****TailSet****");
34          NavigableSet<Integer> ns2=myavlTree.tailSet( fromElement: 30, inclusive: true);
35          Iterator iterns2=ns2.iterator();
36          while (iterns2.hasNext()){
37              System.out.print(iterns2.next()+" ");
```

```
Run:  Main
************AVLTREE***************
FirsElement:5
SecondElement:20
****HeadSet****
5 20
****TailSet****
30 40 50
Process finished with exit code 0
```

Part2:

1)INTRODUCTİON

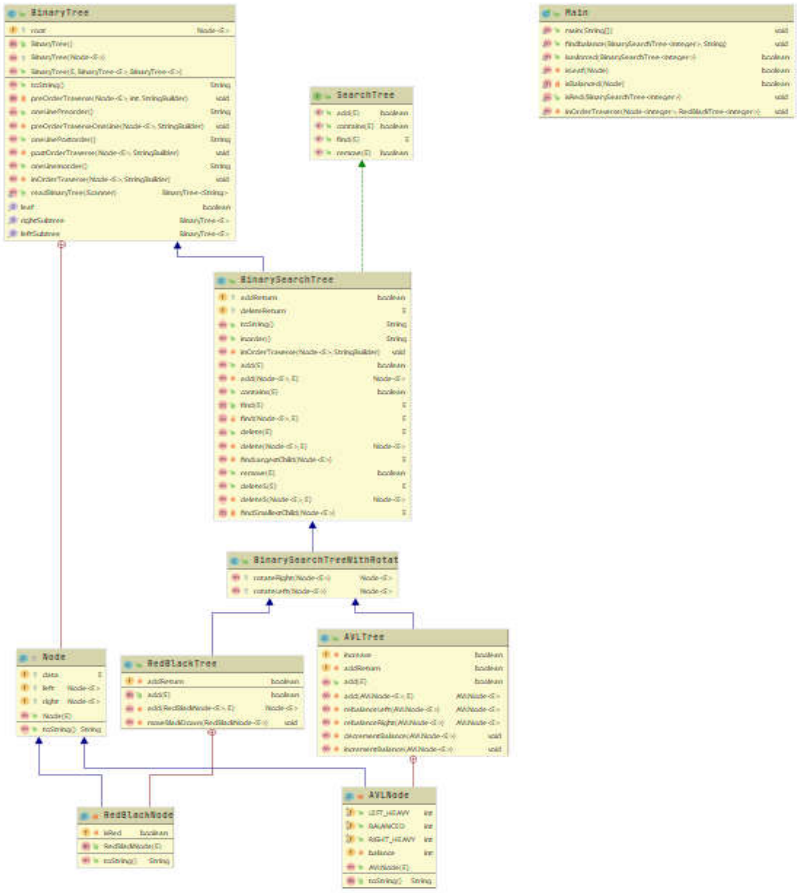
1.1)Problem Definition

The binarysearch tree given in the problem asks whether it is an avl tree and the red black tree asks for the root node color.

1.2- System Requirements

Starting from the definition of the problem, the solution of this problem is solved by writing the AVL tree and the Red-Black tree structure and applying the necessary functions to them.

2)Class Diagram



3) Problem Solution Approach

The given binary search tree is analyzed and checked if it is a balanced tree. If it is balanced, it fits the avl tree and red black tree structure. Then the root node color is determined with the isRed function.

4)Test Case

```java
public static void main(String[] args) {
    BinarySearchTree<Integer> binarySearchTree=new BinarySearchTree<Integer>();
    BinarySearchTree<Integer> binarySearchTree1=new BinarySearchTree<Integer>();
    System.out.println("*****AVLTREE******");
    binarySearchTree.add(10);
    binarySearchTree.add(5);
    binarySearchTree.add(3);
    binarySearchTree.add(20);
    binarySearchTree.add(30);
    binarySearchTree.add(40);
    binarySearchTree.add(50);
    findbalance(binarySearchTree, name: "AVLTREE");
    System.out.println("\n*****RedBlackTree****");
    binarySearchTree1.add(20);
    binarySearchTree1.add(30);
    binarySearchTree1.add(40);
    binarySearchTree1.add(50);
    isRed(binarySearchTree1);
```

Run: Main

```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrain
*****AVLTREE******
This Tree isnot AVLTREE

*****RedBlackTree****
Black: 30

Process finished with exit code 0
```

Part3:

1)INTRODUCTİON

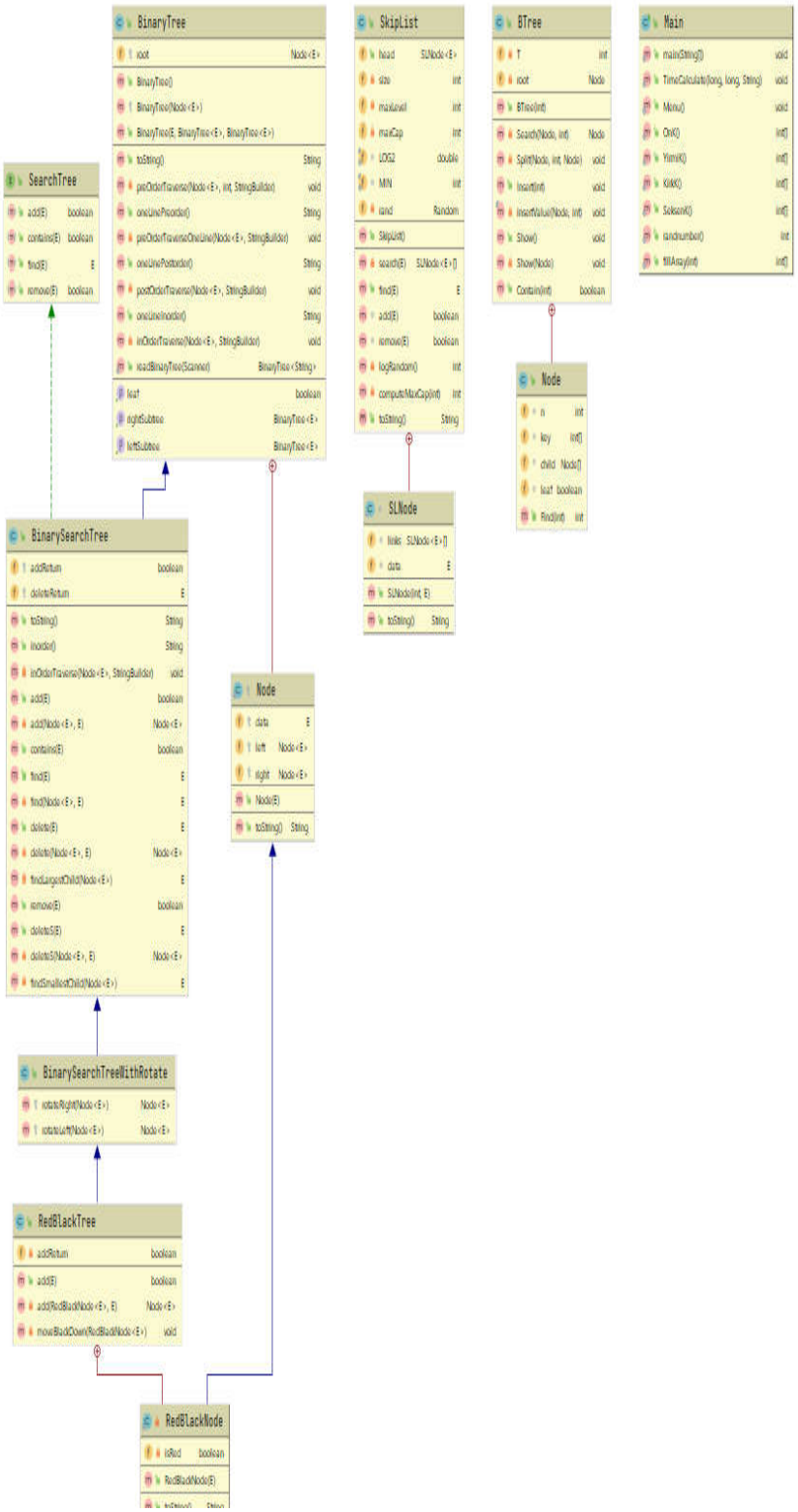
1.1)Problem Definition

In this homework , we are asked to implement the Binary search tree , Red-Black tree, B-tree , SkipList add some functions and write them.

1.2- System Requirements

Based on the definition of the problem, all the structures required for the solution of this problem were defined, and a very different number of data was added to these structures as a driver function, and their running times were measured.

2) Class Diagram



3) Problem Solution Approach

The structures with algorithms in the book were implemented with Java. 10k, 20k, 40k, 80k values were added to these structures and their running times were measured.

4)Test Case

a) 10k, 20k, 40k, 80k values were added to these structures and their running times were measured.

BinarySearchTree:

```
####BinarySearchTree10K-Time-Seconds####
0.005
**************************
####BinarySearchTree20K-Time-Seconds####
0.008
**************************
####BinarySearchTree40K-Time-Seconds####
0.011
**************************
####BinarySearchTree80K-Time-Seconds####
0.007
**************************
```

Red-BlackTree:

```
####RedBlackTree10K-Time-Seconds####
0.009
**************************
####RedBlackTree20K-Time-Seconds####
0.014
**************************
####RedBlackTree40K-Time-Seconds####
0.013
**************************
####RedBlackTree80K-Time-Seconds####
0.015
**************************
```

BTree:

```
####BTree10K-Time-Seconds####
0.003
**************************
####BTree20K-Time-Seconds####
0.006
**************************
####BTree40K-Time-Seconds####
0.013
**************************
####BTree80K-Time-Seconds####
0.008
**************************
```

SkipList:

```
####SkipList10K-Time-Seconds####
0.015
**************************
####SkipList20K-Time-Seconds####
0.012
**************************
####SkipList40K-Time-Seconds####
0.013
**************************
####SkipList80K-Time-Seconds####
0.017
**************************
```

Graphic: