GTU Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework 7 Report

AHMET OKUR
1801042655

1. SYSTEM REQUIREMENTS

Functional Requirement

Part 1

These methods need to be implemented

FOR PART 1.1

1) insert METHOD
2) delete METHOD
3) descendingIterator METHOD

FOR PART 1.2

1) insert
2) iterator
3) headSet
4) tailSet

Part 2

1) WE NEED METHODS TO CONTROL AVL TREE RULES
2) WE NEED METHODS TO CONTROL RED-BLACK TREE RULES

Part 3

1) Binary search tree implementation in the book

2) Red-Black tree implementation in the book

3) 2-3 tree implementation in the book

3) B-tree implementation in the book

5) Skip list implementation in the book

Non-Functional Requirement

1) Back-end software: Java 11

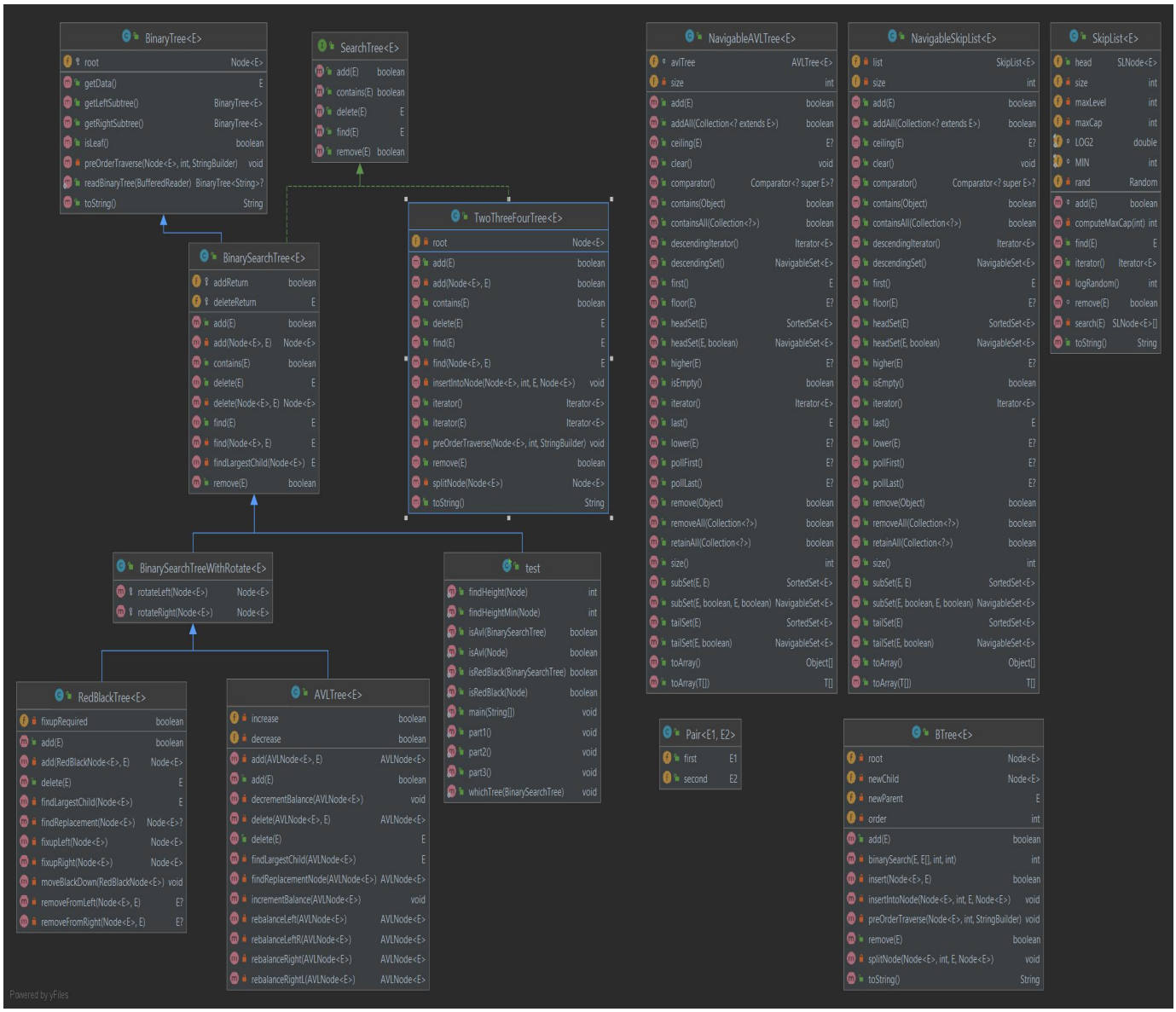
2) Hardware Interfaces: Mac, Linux and Windows operating systems

3) The system has an internal bug reporting system that keeps track of what types errors happened

and when

4) The system holds all the data to keep track of everything available in itself.

2. USE CASE AND CLASS DIAGRAMS



3. PROBLEM SOLUTION APPROACH

Part 1

I implemented skip list and avl tree for part 1

Part 2

I wrote a method that measures the depth of the tree for avl control.For red black tree control, I wrote a method that measures the depth of the tree and measures the step to the shortest leaf**.**

Part 3

I use

- Binary search tree implementation in the book

- Red-Black tree implementation in the book

- 2-3 tree implementation in the book

- B-tree implementation in the book

- Skip list implementation in the book

And then and I compared these structers

4.TEST CASES AND RESULTS

Part 1.1

In this section, we add elements to the skip list.

```
NavigableSet<Integer> slist = new NavigableSkipList<>();

slist.add(140);
slist.add(28);
slist.add(21);
slist.add(92);
slist.add(59);
slist.add(17);
slist.add(81);
slist.add(91);
slist.add(-1);
```

In this section, you can see the elements added to the skip list and the iterator.

```
System.out.println("\nTEST descendingIterator");
Iterator iter = slist.descendingIterator();
while (iter.hasNext()) {
    res1.append(iter.next());
    res1.append(" ");
}
System.out.println(res1);
```

```
TEST descendingIterator
140 92 91 81 59 28 21 17 -1
```

In this section, we are deleting elements from the skip list.

```
System.out.println("\nRemoving... 91");
slist.remove( o: 91);
iter = slist.descendingIterator();

while (iter.hasNext()) {
    res2.append(iter.next());
    res2.append(" ");
}
System.out.println(res2);
```

```
Removing... 91
140 92 81 59 28 21 17 -1
```

Part 1.2

In this section, we add elements to the AVL TREE.

```
NavigableSet<Integer> tree = new NavigableAVLTree<>();

res1 = new StringBuilder();
res2 = new StringBuilder();
tree.add(10);
tree.add(38);
tree.add(-128);
tree.add(-12);
tree.add(-29);
tree.add(147);
tree.add(241);
tree.add(92);
tree.add(51);
```

In this section, you can see the elements added to the avl tree and the iterator.

```
    iter = tree.iterator();

    while (iter.hasNext()) {
        res1.append(iter.next());
        res1.append(" ");
    }
    System.out.println(res1);
```

```
TEST Iterator
-128 -29 -12 10 38 51 92 147 241
```

This section contains the use and result of the headset method.

```
StringBuilder res = new StringBuilder();
NavigableSet newTree = tree.headSet( toElement: 51,  inclusive: false);
Iterator iter1 = newTree.iterator();

while (iter1.hasNext()) {
    res.append(iter1.next());
    res.append(" ");
}
System.out.println(res);
```

```
Testing headset->51->false
-128 -29 -12 10 38
```

This section contains the use and result of the tailset method.

```
    StringBuilder res = new StringBuilder();
    NavigableSet newTree = tree.tailSet( fromElement: 51,  inclusive: false);
    Iterator iter1 = newTree.iterator();

    while (iter1.hasNext()) {
        res.append(iter1.next());
        res.append(" ");
    }
    System.out.println(res);
```

result

```
Testing tailset->51->false
92 147 241
```

In this section, we are deleting elements from the AVL TREE.

```
tree.remove( o: -12);
iter = tree.iterator();
while(iter.hasNext()){
    res2.append(iter.next());
    res2.append(" ");
}
```

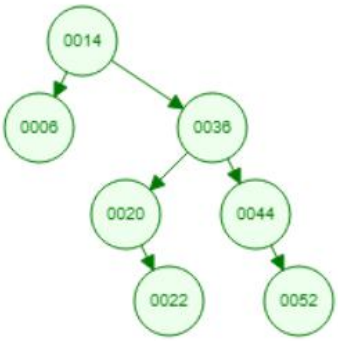
```
Removing -12
-128 -29 10 38 51 92 147 241
```

Part 2

In this section, we add elements to the BST.

```java
BinarySearchTree<Integer> tree1 = new BinarySearchTree<>();
tree1.add(14);
tree1.add(6);
tree1.add(36);
tree1.add(20);
tree1.add(44);
tree1.add(20);
tree1.add(22);
tree1.add(52);
whichTree(tree1);
```

this tree cannot be avl but can be red black tree

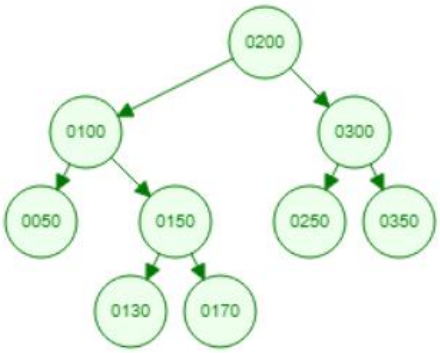



```
AVL TREE = NO
RED-BLACK TREE=YES
```

In this section, we add elements to the BST.

```
BinarySearchTree<Integer> tree2 = new BinarySearchTree<>();
tree2.add(200);
tree2.add(100);
tree2.add(300);
tree2.add(50);
tree2.add(150);
tree2.add(250);
tree2.add(350);
tree2.add(130);
tree2.add(170);

whichTree(tree2);
```

this tree can be both avl and red black tree




```
Testing with inserting items like AVL TREE

AVL TREE = YES
RED-BLACK TREE=YES
```

some methods i use to control the tree

```
public static void whichTree(BinarySearchTree Tree){...}

public static boolean isAvl(BinarySearchTree tree){...}
public static boolean isAvl(BinaryTree.Node Tree){...}
public static int findHeight(BinaryTree.Node aNode){...}
public static int findHeightMin(BinaryTree.Node aNode){...}


public static boolean isRedBlack(BinarySearchTree Tree){
    return isRedBlack(Tree.root);
}
public static boolean isRedBlack(BinaryTree.Node Tree){...}
```

Part 3

first i created 4 arraylists (10000,20000,40000,80000) each receives 10 BSTs

```
ArrayList<BinarySearchTree<Integer> > bSearchTrees10 = new ArrayList<>();
ArrayList<BinarySearchTree<Integer> > bSearchTrees20 = new ArrayList<>();
ArrayList<BinarySearchTree<Integer> > bSearchTrees40 = new ArrayList<>();
ArrayList<BinarySearchTree<Integer> > bSearchTrees80 = new ArrayList<>();

int count = 0;
double TotalTime_10=0,TotalTime_20=0,TotalTime_40=0,TotalTime_80=0;

for(int i=0; i<10; ++i){
    bSearchTrees10.add(new BinarySearchTree<>());
    bSearchTrees20.add(new BinarySearchTree<>());
    bSearchTrees40.add(new BinarySearchTree<>());
    bSearchTrees80.add(new BinarySearchTree<>());
}
```

I filled the trees inside the arraylist and summed their duration

```
for(int i=0; i<10; ++i) {
    double start = System.currentTimeMillis();
    while (count < 10000) {
        int num = rand.nextInt();
        if(bSearchTrees10.get(i).add(num))
            ++count;
    }
    double end = System.currentTimeMillis();
    TotalTime_10 += end - start;
    count = 0;
    start = System.currentTimeMillis();
    while(count < 20000){
        int num = rand.nextInt();
        if(bSearchTrees20.get(i).add(num))
            ++count;
    }
    end = System.currentTimeMillis();
    TotalTime_20 += end - start;
    count = 0;
    start = System.currentTimeMillis();
    while(count < 40000){
        int num = rand.nextInt();
        if(bSearchTrees40.get(i).add(num))
            ++count;
    }
    end = System.currentTimeMillis();
    TotalTime_40 += end - start;
    count = 0;
    start = System.currentTimeMillis();
    while(count < 80000){
        int num = rand.nextInt();
        if(bSearchTrees80.get(i).add(num))
            ++count;
    }
    end = System.currentTimeMillis();
    TotalTime_80 += end - start;
    count = 0;
}
```

then i calculated the average

```
System.out.println("\t\t### BINARY SEARCH TREE TIME TEST ###\n\n");
System.out.println("Average time of 10000 adding element Binary Search Tree " +  TotalTime_10/10 + " ms");
System.out.println("Average time of 20000 adding element Binary Search Tree " +  TotalTime_20/10 + " ms");
System.out.println("Average time of 40000 adding element Binary Search Tree " +  TotalTime_40/10 + " ms");
System.out.println("Average time of 80000 adding element Binary Search Tree " +  TotalTime_80/10 + " ms");
```

NOT!

I did the above operations for Red-Black tree, 2-3 tree, B-tree and Skip list, it is exactly the same.

RESULTS

```
        ### BINARY SEARCH TREE TIME TEST ###


Average time of 10000 adding element Binary Search Tree 3.7 ms
Average time of 20000 adding element Binary Search Tree 4.5 ms
Average time of 40000 adding element Binary Search Tree 11.2 ms
Average time of 80000 adding element Binary Search Tree 27.5 ms
```

```
        ### RED BLACK TREE TIME TEST ###


Average time of 10000 adding element to Red Black Tree 4.1 ms
Average time of 20000 adding element to Red Black Tree 8.2 ms
Average time of 40000 adding element to Red Black Tree 19.4 ms
Average time of 80000 adding element to Red Black Tree 35.9 ms
```

```
        ### TWO THREE TREE TIME TEST ###


Average time of 10000 adding element to two-three Tree 6.3 ms
Average time of 20000 adding element to two-three Tree 9.1 ms
Average time of 40000 adding element to two-three Tree 19.1 ms
Average time of 80000 adding element to two-three Tree 56.4 ms
```

```
        ###  B TREE TIME TEST ###


Average time of 10000 adding element to B-Tree Tree 4.1 ms
Average time of 20000 adding element to B-Tree Tree 8.8 ms
Average time of 40000 adding element to B-Tree Tree 20.4 ms
Average time of 80000 adding element to B-Tree Tree 52.5 ms
```

```
       ###  SKIP LIST TIME TEST ###


Average time of 10000 adding element to Skip List 4.6 ms
Average time of 20000 adding element to Skip List 11.2 ms
Average time of 40000 adding element to Skip List 22.5 ms
Average time of 80000 adding element to Skip List 71.8 ms
```

In this section, we add 100 elements to the trees filled in before and calculate their duration.

```
TotalTime_10=0;TotalTime_20=0;TotalTime_40=0;TotalTime_80=0;
for(int i=0; i<10; ++i) {
    double start = System.nanoTime();
    while (count < 100) {
        int num = rand.nextInt();
        if(bSearchTrees10.get(i).add(num))
            ++count;
    }
    double end = System.nanoTime();
    TotalTime_10 += (end - start) / Math.pow(10,6);
    count = 0;
    start = System.nanoTime();
    while(count < 100){
        int num = rand.nextInt();
        if(bSearchTrees20.get(i).add(num))
            ++count;
    }
    end = System.nanoTime();
    TotalTime_20 += (end - start) / Math.pow(10,6);
    count = 0;
    start = System.nanoTime();
    while(count < 100){
        int num = rand.nextInt();
        if(bSearchTrees40.get(i).add(num))
            ++count;
    }
    end = System.nanoTime();
    TotalTime_40 += (end - start) / Math.pow(10,6);
    count = 0;
    start = System.nanoTime();
    while(count < 100){
        int num = rand.nextInt();
        if(bSearchTrees80.get(i).add(num))
            ++count;
    }
    end = System.nanoTime();
    TotalTime_80 += (end - start) / Math.pow(10,6);
    count = 0;
}
System.out.printf("\nAfter inserting 100 item to Binary Search Tree(10000) %.4f ms\n",TotalTime_10);
System.out.printf("After inserting 100 item to Binary Search Tree(10000) %.4f ms\n",TotalTime_20);
System.out.printf("After inserting 100 item to Binary Search Tree(10000) %.4f ms\n",TotalTime_40);
System.out.printf("After inserting 100 item to Binary Search Tree(10000) %.4f ms\n",TotalTime_80);
```

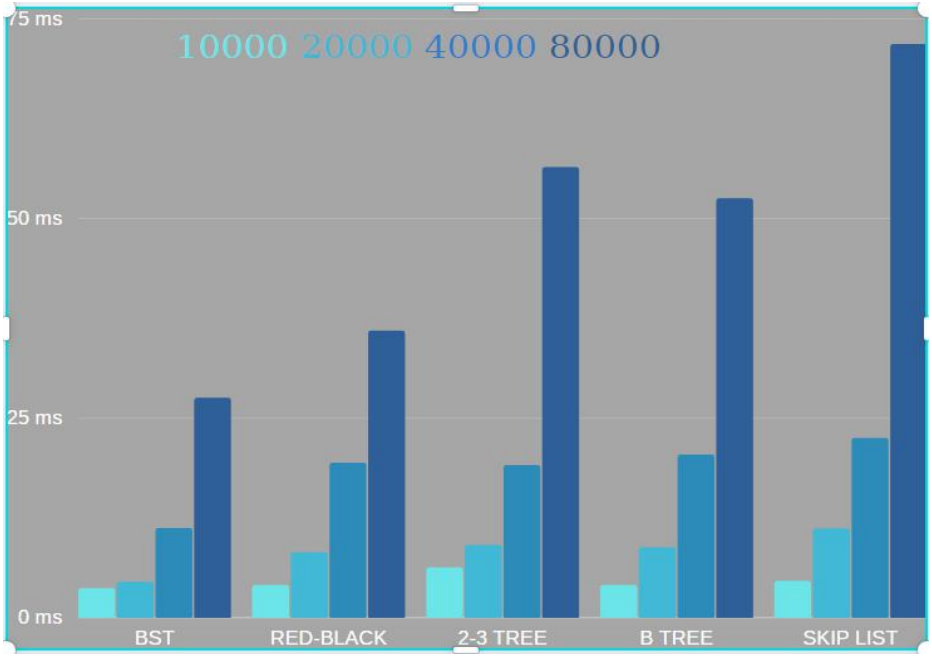
NOT!
I did the above operations for Red-Black tree, 2-3 tree, B-tree and Skip list, it is exactly the same.

RESULTS

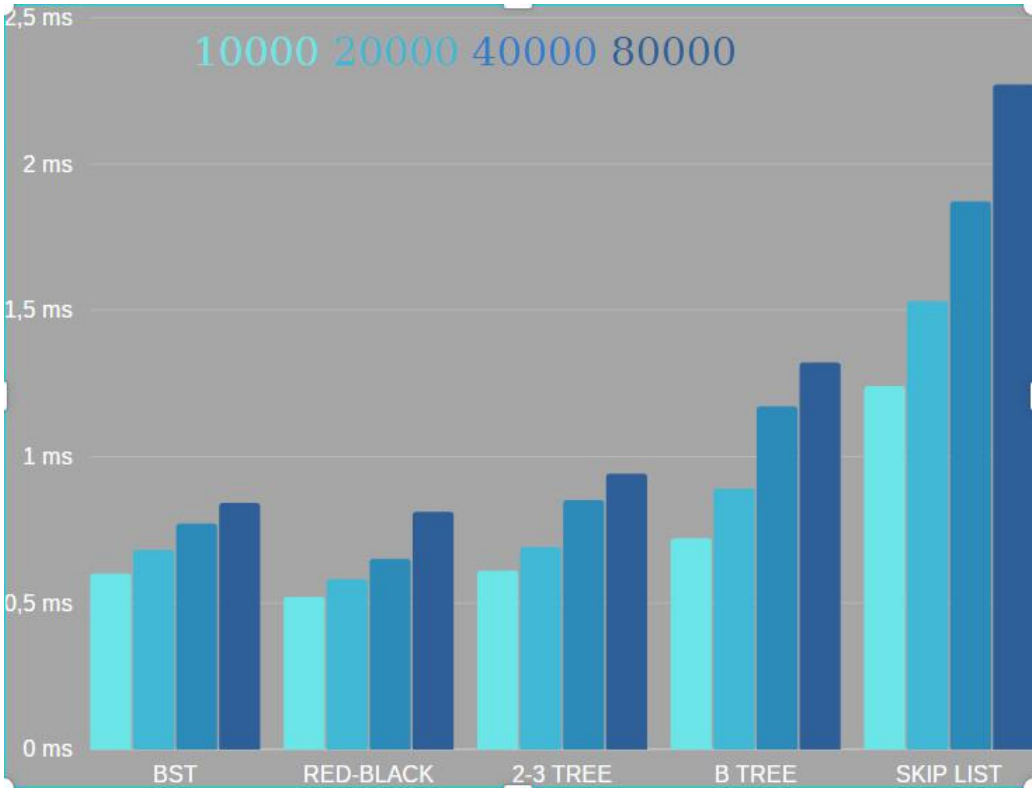
```
After inserting 100 item to Binary Search Tree(10000) 0,6006 ms
After inserting 100 item to Binary Search Tree(10000) 0,6836 ms
After inserting 100 item to Binary Search Tree(10000) 0,7740 ms
After inserting 100 item to Binary Search Tree(10000) 0,8476 ms

After inserting 100 item to Red Black Tree(10000) 0,5216 ms
After inserting 100 item to Red Black Tree(10000) 0,5806 ms
After inserting 100 item to Red Black Tree(10000) 0,6549 ms
After inserting 100 item to Red Black Tree(10000) 0,8129 ms

After inserting 100 item to Two-Three Tree(10000) 0,6113 ms
After inserting 100 item to Two-Three Tree(20000) 0,6950 ms
After inserting 100 item to Two-Three Tree(40000) 0,8541 ms
After inserting 100 item to Two-Three Tree(80000) 0,9497 ms

After inserting 100 item to B-Tree Tree(10000) 0,7280 ms
After inserting 100 item to B-Tree Tree(20000) 0,8915 ms
After inserting 100 item to B-Tree Tree(40000) 1,1778 ms
After inserting 100 item to B-Tree Tree(80000) 1,3235 ms

After inserting 100 item to Skip List(10000) 1,2469 ms
After inserting 100 item to Skip List(20000) 1,5376 ms
After inserting 100 item to Skip List(40000) 1,8744 ms
After inserting 100 item to Skip List(80000) 2,2724 ms
```

PART 3 GRAPH

ADDING 10000,20000,40000,80000



AFTER ADDING 100 ELEMENTS ALL PART


2,5 ms
10000 20000 40000 80000
2 ms
1,5 ms
1 ms
0,5 ms
0 ms
BST
RED-BLACK
2-3 TREE
B TREE
SKIP LIST