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

Burak Yıldırım
1901042609

1. System Requirements

a. Functional Requirements

Part1

- **NavigableSet implementation using SkipList**
- add method adds the given element to the set using SkipList's add method.
- remove method removes the given element from the set using SkipList's remove method.
- descendingIterator method returns an iterator over the elements of the set, in descending order.
- **NavigableSet implementation using AVLTree**
- add method adds the given element to the set using AVLTree's add method.
- iterator method returns an iterator over the elements of the set, in increasing order.
- headSet method returns a NavigableSet consists of elements less than the given element.
- tailSet method returns a NavigableSet consists of elements greater than or equal to the given element.

Part2

- checkType method of TypeChecker class takes a binary search tree and checks its type.
- There are 4 possible outputs of this method:
- 1 if the given tree is both red-black tree and avl tree.
- 2 if the given tree is only red-black tree.
- 3 if the given tree is only avl tree.
- 4 if the given tree is neither red-black nor avl tree.

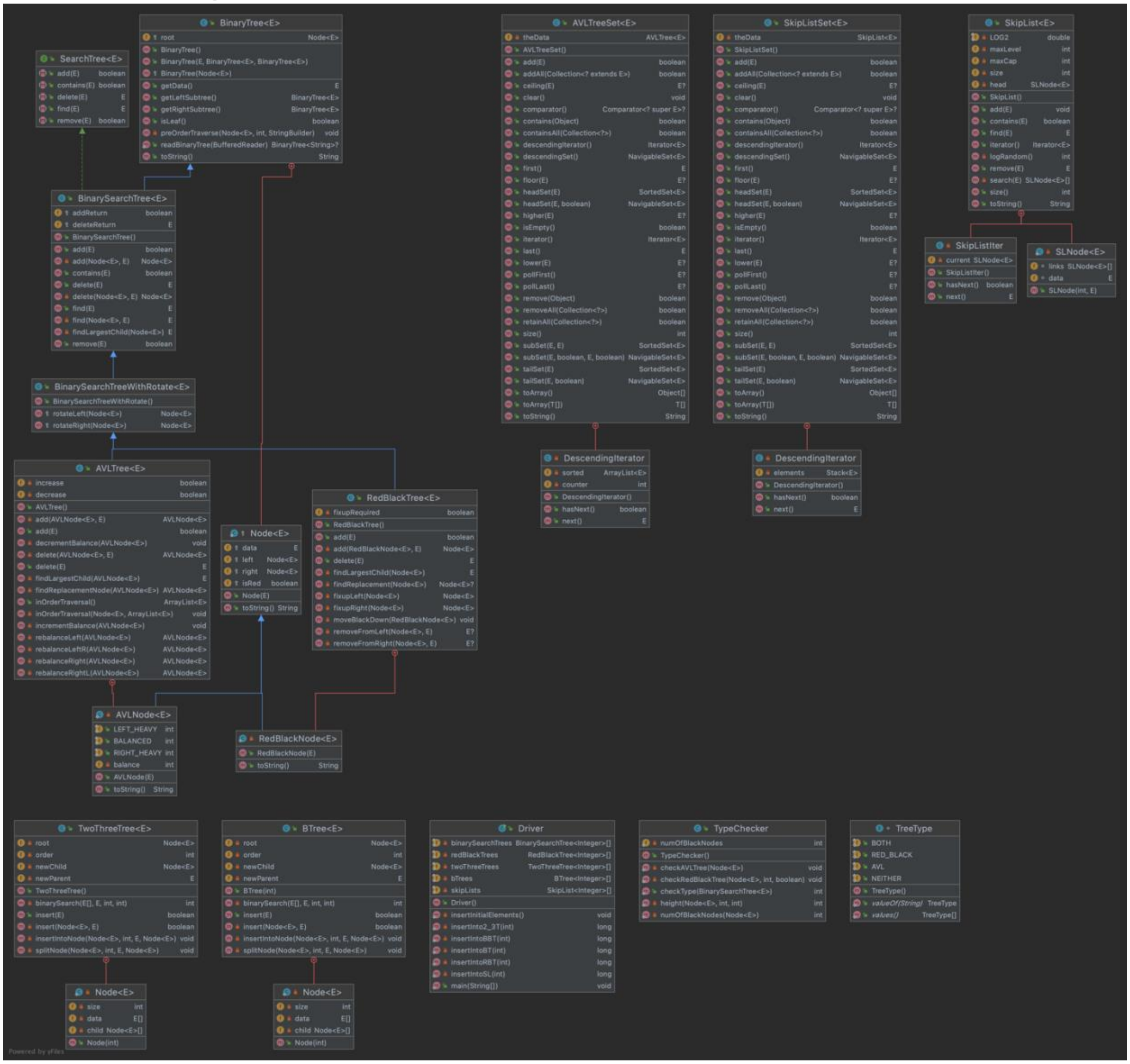
Part3

- There is no requirement needed.
- Implementations in the course book are done.

b. Non-Functional Requirements

User must have Java Runtime Environment to run this project.

2. Class Diagram



3. Problem Solution Approach

a. Part1

I implemented SkipList and AVLTree in the course book and used them to implement NavigableSet interface.

b. Part2

I wrote a method which first checks the color condition, then checks the height condition to determine the type of given tree.

c. Part3

I implemented the required data structures and record the average addition times.

4. Test Cases

a. Part1

```java
System.out.println("-------------SkipListSet Methods-------------");
System.out.println("---------Adding---------");
ArrayList<Integer> arrayList = new ArrayList<>();
SkipListSet<Integer> skipListSet = new SkipListSet<>();
for(int i = 0; i < 10; i++){
    arrayList.add(random.nextInt( bound: 1000));
}
System.out.println("Elements To Add");
System.out.println(arrayList);
System.out.println("Before Adding");
System.out.println(skipListSet);
System.out.println("After Adding");
for(Integer i : arrayList){
    skipListSet.add(i);
}
System.out.println(skipListSet);

System.out.println("\n--------Removing--------");
System.out.println("Element To Remove: " + arrayList.get(0));
System.out.println("Before Removing");
System.out.println(skipListSet);
System.out.println("After Removing");
skipListSet.remove(arrayList.get(0));
System.out.println(skipListSet);

System.out.println("\n---descendingIterator---");
Iterator<Integer> iterator = skipListSet.descendingIterator();
while (iterator.hasNext()){
    System.out.print(iterator.next() + " ");
}
```

```java
System.out.println("\n\n-------------AVLTreeSet Methods-------------");
System.out.println("---------Adding--------");
arrayList.clear();
AVLTreeSet<Integer> avlTreeSet = new AVLTreeSet<>();
for(int i = 0; i < 10; i++){
    arrayList.add(random.nextInt( bound: 1000));
}
System.out.println("Elements To Add");
System.out.println(arrayList);
System.out.println("Before Adding");
System.out.println(avlTreeSet);
System.out.println("After Adding");
for(Integer i : arrayList){
    avlTreeSet.add(i);
}
System.out.println(avlTreeSet);

System.out.println("\n--------Iterator--------");
Iterator<Integer> iterator1 = avlTreeSet.iterator();
while (iterator1.hasNext()){
    System.out.print(iterator1.next() + " ");
}

System.out.println("\n\n--------headSet--------");
System.out.println("toElement: " + arrayList.get(0));
System.out.println(avlTreeSet.headSet(arrayList.get(0)));

System.out.println("\n--------tailSet--------");
System.out.println("fromElement: " + arrayList.get(arrayList.size() - 1));
System.out.println(avlTreeSet.tailSet(arrayList.get(arrayList.size() - 1)) + "\n");
```

b. Part2

```java
System.out.println("Trying With A Red-Black Tree");
RedBlackTree<Integer> redBlackTree = new RedBlackTree<>();
redBlackTree.add(993);
redBlackTree.add(211);
redBlackTree.add(342);
redBlackTree.add(867);
redBlackTree.add(777);
redBlackTree.add(311);
redBlackTree.add(580);
redBlackTree.add(318);
redBlackTree.add(951);
redBlackTree.add(948);
redBlackTree.add(778);
redBlackTree.add(86);
redBlackTree.add(262);
redBlackTree.add(259);
redBlackTree.add(194);
System.out.println(redBlackTree);
System.out.println("Result: " + TreeType.values()[TypeChecker.checkType(redBlackTree)] + "\n");
```

```java
System.out.println("Trying With An AVL Tree");
AVLTree<Integer> avlTree = new AVLTree<>();
avlTree.add(615);
avlTree.add(836);
avlTree.add(510);
avlTree.add(258);
avlTree.add(699);
avlTree.add(116);
avlTree.add(286);
avlTree.add(343);
avlTree.add(178);
avlTree.add(772);
avlTree.add(713);
avlTree.add(287);
avlTree.add(548);
avlTree.add(932);
System.out.println(avlTree);
System.out.println("Result: " + TreeType.values()[TypeChecker.checkType(avlTree)] + "\n");
```

```java
System.out.println("Trying With A Binary Search Tree Which Is Neither One Of Them");
BinarySearchTree<Integer> binarySearchTree = new BinarySearchTree<>();
for(int i = 0; i < 10; i++){
    binarySearchTree.add(i);
}
System.out.println(binarySearchTree);
System.out.println("Result: " + TreeType.values()[TypeChecker.checkType(binarySearchTree)] + "\n");

System.out.println("Trying With A Red-Black Tree Which Is Also An AVL Tree");
RedBlackTree<Integer> redBlackTree1 = new RedBlackTree<>();
for(int i = 0; i < 10; i++){
    redBlackTree1.add(random.nextInt( bound: 1000));
}
System.out.println(redBlackTree1);
System.out.println("Result: " + TreeType.values()[TypeChecker.checkType(redBlackTree1)] + "\n");
```

c. Part3

```java
insertInitialElements();
for(int i = 0; i < 4; i++){
    System.out.println("Average running times of adding 100 randomly generated elements to " + 10*((int)Math.pow(2, i)) + "K-sized data structures");
    System.out.printf("%-19s%s%d%s", "Binary Search Tree", ": ", insertIntoBBT(i), "ns\n");
    System.out.printf("%-19s%s%d%s", "Red-Black Tree", ": ", insertIntoRBT(i), "ns\n");
    System.out.printf("%-19s%s%d%s", "2-3 Tree", ": ", insertInto2_3T(i), "ns\n");
    System.out.printf("%-19s%s%d%s", "B-Tree", ": ", insertIntoBT(i), "ns\n");
    System.out.printf("%-19s%s%d%s", "SkipList", ": ", insertIntoSL(i), "ns\n\n");
}
```

5. Running Command And Results

```
----------------------------------------PART 1----------------------------------------
--------------SkipListSet Methods--------------
---------Adding---------
Elements To Add
[753, 104, 742, 127, 435, 607, 457, 367, 655, 271]
Before Adding
[]
After Adding
[104, 127, 271, 367, 435, 457, 607, 655, 742, 753]

--------Removing--------
Element To Remove: 753
Before Removing
[104, 127, 271, 367, 435, 457, 607, 655, 742, 753]
After Removing
[104, 127, 271, 367, 435, 457, 607, 655, 742]

---descendingIterator---
742 655 607 457 435 367 271 127 104
```

```
--------------AVLTreeSet Methods--------------
---------Adding---------
Elements To Add
[149, 392, 936, 948, 775, 619, 210, 206, 73, 148]
Before Adding
[]
After Adding
[73, 148, 149, 206, 210, 392, 619, 775, 936, 948]

--------Iterator--------
73 148 149 206 210 392 619 775 936 948

--------headSet--------
toElement: 149
[73, 148]

--------tailSet--------
fromElement: 148
[148, 149, 206, 210, 392, 619, 775, 936, 948]
```

```
-------------------------------------------PART 2-------------------------------------------
Trying With A Red-Black Tree
Black: 342
  Black: 311
    Red  : 211
      Black: 86
        null
        Red  : 194
          null
          null
      Black: 262
        Red  : 259
          null
          null
        null
    Black: 318
      null
      null
  Black: 867
    Black: 777
      Red  : 580
        null
        null
      Red  : 778
        null
        null
    Black: 951
      Red  : 948
        null
        null
      Red  : 993
        null
        null

Result: RED_BLACK
```

```
Trying With An AVL Tree
1: 343
  -1: 258
    1: 116
      null
      0: 178
        null
        null
    0: 286
      null
      0: 287
        null
        null
  1: 615
    1: 510
      null
      0: 548
        null
        null
    1: 772
      0: 699
        null
        0: 713
          null
          null
      1: 836
        null
        0: 932
          null
          null

Result: AVL
```

```
Trying With A Binary Search Tree Which Is Neither One Of Them
0
  null
  1
    null
    2
      null
      3
        null
        4
          null
          5
            null
            6
              null
              7
                null
                8
                  null
                  9
                    null
                    null

Result: NEITHER
```

```
Trying With A Red-Black Tree Which Is Also An AVL Tree
Black: 695
  Red  : 470
    Black: 111
      Red  : 2
        null
        null
      Red  : 182
        null
        null
    Black: 526
      null
      Red  : 557
        null
        null
  Black: 969
    Red  : 730
      null
      null
    Red  : 974
      null
      null

Result: BOTH
```

```
------------------------------------------PART 3------------------------------------------
Average running times of adding 100 randomly generated elements to 10K-sized data structures
Binary Search Tree : 68023ns
Red-Black Tree     : 50821ns
2-3 Tree           : 83636ns
B-Tree             : 43663ns
SkipList           : 69100ns

Average running times of adding 100 randomly generated elements to 20K-sized data structures
Binary Search Tree : 75809ns
Red-Black Tree     : 61641ns
2-3 Tree           : 88785ns
B-Tree             : 47116ns
SkipList           : 83448ns

Average running times of adding 100 randomly generated elements to 40K-sized data structures
Binary Search Tree : 81231ns
Red-Black Tree     : 79101ns
2-3 Tree           : 109623ns
B-Tree             : 73080ns
SkipList           : 118916ns

Average running times of adding 100 randomly generated elements to 80K-sized data structures
Binary Search Tree : 87007ns
Red-Black Tree     : 131252ns
2-3 Tree           : 98825ns
B-Tree             : 91059ns
SkipList           : 254381ns
```

NOTE: Comparisons and graph below are made according to the part 3 result above

Comparison of running times:

B-Tree < Binary Search Tree < Red-Black Tree < 2-3 Tree < SkipList

Comparison of increase rates:

2-3 Tree < Binary Search Tree < B-Tree < Red-Black Tree < SkipList

Graph (running-time vs problem size)