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

Coşkun Hasan ŞALTU
1801042631

1. PROBLEM SOLUTION APPROACH

Using skip-list, navigableset was implemented. Insert, delete and descendingiterator functions were written. Also, navigableset was implemented using avl-tree. Insert, iterator, headset and tailset functions were written. In the second part, it was tested whether a tree created was an avl tree. In third part, the insert functions of some data structures were tested incrementally. Also, which one was faster was compared.

2. SYSTEM REQUIRMENTS

1.Functional Requirments

PART 1:

- Insert() : insert operation
- Delete(): delete operation

-descending iterator(): this iterator sort elements to descending order.
-iterator(): returns iterator
-headSet(): used to return a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement.
-tailSet(): returns the element present in the set, which is greater than or equal to fromElement.

PART 2:

-is_AVL():This function check the tree is AVL tree or not.

PART 3:

-testing some data structers insert operation and compare them.

2.Unfunctional Requirments

-Hardware should be able to run at least JAVA SE13.

3. TEST CASE

A. Navigable set with AVL tree

```
System.out.println("-----Define navigableset with avl tree-----");

NavigableSetwithAVLTree<Integer> tree = new NavigableSetwithAVLTree<Integer>();
```

INSERT AVL TREE

```
System.out.println("-----Insert operation avl tree-----");
System.out.println("----- 1 2 3 4 5 added-----");
tree.insert(1);
tree.insert(2);
tree.insert(3);
tree.insert(4);
tree.insert(5);
```

DEFINE ITERATOR AND TEST

```
System.out.println("-----Define iterator avl tree-----");
iter = tree.Iterator();

System.out.println("-----Testing Iterator-----");
while(iter.hasNext() != false) {
    System.out.println(iter.next());
}
```

HEAD SET FUNCTİON TESTED

```
System.out.println("-----Testing HeadSet operation until 4-----");
tree2 = tree.headSet(4, false);


System.out.println();
iter = tree2.Iterator();
System.out.println("-----Printing HeadSet Tree-----");
while(iter.hasNext() != false) {
    System.out.println(iter.next());
}
```

B. Navigable set with Skip List

```
System.out.println("-----Define navigableset with skiplist-----");

NavigableSetwithSkipList<Integer> list = new NavigableSetwithSkipList<>();
```

INSERT SKİP LİST

```
System.out.println("-----Insert operation skip list-----");
System.out.println("----- 1 2 3 4 5 added-----");
list.insert(1);
list.insert(2);
list.insert(3);
list.insert(4);
list.insert(5);
```

DELETE SKİP LİST

```
System.out.println("-----Delete operation skip list - delete 4 ");
System.out.println(list.delete(4));

System.out.println("-----Delete operation skip list - delete 100 ");
System.out.println(list.delete(100));
```

DEFİNE AND TEST DESCENDİNG İTERATOR

```
System.out.println("-----Define descending iterator skip list-----");
iter2 = list.descendingIterator();
System.out.println("-----Testing Iterator-----");
while(iter2.hasNext() != false) {
    System.out.println(iter2.next());
}
```

C. UNDERSTANDİNG İS AVL TREE OR NOT

DEFINE AVL TREE

```
AVLTree<Integer> avl = new AVLTree<>();
```

```
avl.add(1);
avl.add(2);
avl.add(3);
avl.add(4);
avl.add(5);
```

DEFINE NOT AVL TREE

```
BinarySearchTree<Integer> search = new BinarySearchTree<>();

search.add(1);
search.add(2);
search.add(3);
search.add(4);
search.add(5);
```

TESTING AVL TREE FUNCTIONS

```
System.out.println(is_AVL(avl));
System.out.println(is_AVL(search));
```

D. COMPARE DATA STRUCTERS

TESTING 100 EXTRA ELEMENTS TO 10.000 ELEMENTS

```
BinarySearchTree<Integer> searchTree = new BinarySearchTree<>();
RedBlackTree<Integer> redBlack = new RedBlackTree<>();
TwoThreeTree<Integer> twoThreeTree = new TwoThreeTree<>();
SkipList<Integer> skipList = new SkipList<>();
BTree<Integer> bTree = new BTree<>(5);
List<Integer> range = IntStream.range(0, 10000).boxed()
        .collect(Collectors.toCollection(ArrayList::new));
Collections.shuffle(range);

for(int j=0;j<10000;j++) {
    searchTree.add(range.get(j));
    redBlack.add(range.get(j));
    twoThreeTree.add(range.get(j));
    skipList.add(range.get(j));
    bTree.add(range.get(j));
}
```

```
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    searchTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_bst += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    redBlack.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_redBlack += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    twoThreeTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_twoSecond += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    skipList.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_skiplist += estimatedTime;
startTime = System.nanoTime();
```

```
System.out.println("BST add operation with 10000 items time: " + average_bst/10);
System.out.println("Red-Black Tree add operation with 10000 items time: " + average_redBlack/10);
System.out.println("Two-Theree Tree add operation with 10000 items time: " + average_twoSecond/10);
System.out.println("Skip-List add operation with 10000 items time: " + average_skiplist/10);
System.out.println("B-tree add operation with 10000 items time: " + average_btree/10);
```

TESTING 100 EXTRA ELEMENTS TO 20.000 ELEMENTS

```
BinarySearchTree<Integer> searchTree = new BinarySearchTree<>();
RedBlackTree<Integer> redBlackTree = new RedBlackTree<Integer>();
TwoThreeTree<Integer> twoThreeTree = new TwoThreeTree<>();
SkipList<Integer> skipList = new SkipList<>();
BTree<Integer> bTree = new BTree<>(5);
List<Integer> range = IntStream.range(0, 20000).boxed()
        .collect(Collectors.toCollection(ArrayList::new));
Collections.shuffle(range);

for(int j=0;j<20000;j++) {
    searchTree.add(range.get(j));
    redBlackTree.add(range.get(j));
    twoThreeTree.add(range.get(j));
    skipList.add(range.get(j));
    bTree.add(range.get(j));
}
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    searchTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_bst += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    redBlackTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_redBlack += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    twoThreeTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_twoSecond += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    skipList.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_skiplist += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    bTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;

System.out.println("BST add operation with 20000 items time: " + average_bst/10);
System.out.println("Red-Black Tree add operation with 20000 items time: " + average_redBlack/10);
System.out.println("Two-Theree Tree add operation with 20000 items time: " + average_twoSecond/10);
System.out.println("Skip-List add operation with 20000 items time: " + average_skiplist/10);
System.out.println("B-tree add operation with 20000 items time: " + average_btree/10);
```

TESTING 100 EXTRA ELEMENTS TO 40.000 ELEMENTS

```
BinarySearchTree<Integer> searchTree = new BinarySearchTree<>();
RedBlackTree<Integer> redBlackTree = new RedBlackTree<>();
TwoThreeTree<Integer> twoThreeTree = new TwoThreeTree<>();
SkipList<Integer> skipList = new SkipList<>();
BTree<Integer> bTree = new BTree<>(5);
List<Integer> range = IntStream.range(0, 40000).boxed()
        .collect(Collectors.toCollection(ArrayList::new));
Collections.shuffle(range);

for(int j=0;j<40000;j++) {
    searchTree.add(range.get(j));
    redBlackTree.add(range.get(j));
    twoThreeTree.add(range.get(j));
    skipList.add(range.get(j));
    bTree.add(range.get(j));
}
```

```
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    searchTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_bst += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    redBlackTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_redBlack += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    twoThreeTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_twoSecond += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    skipList.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_skiplist += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    bTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
```

```
System.out.println("BST add operation with 40000 items time: " + average_bst/10);
System.out.println("Red-Black Tree add operation with 40000 items time: " + average_redBlack/10);
System.out.println("Two-Theree Tree add operation with 40000 items time: " + average_twoSecond/10);
System.out.println("Skip-List add operation with 40000 items time: " + average_skiplist/10);
System.out.println("B-tree add operation with 40000 items time: " + average_btree/10);
```

TESTING 100 EXTRA ELEMENTS TO 80.000 ELEMENTS

```
BinarySearchTree<Integer> searchTree = new BinarySearchTree<>();
RedBlackTree<Integer> redBlackTree = new RedBlackTree<>();
TwoThreeTree<Integer> twoThreeTree = new TwoThreeTree<>();
SkipList<Integer> skipList = new SkipList<>();
BTree<Integer> bTree = new BTree<>(5);
List<Integer> range = IntStream.range(0, 80000).boxed()
        .collect(Collectors.toCollection(ArrayList::new));
Collections.shuffle(range);

for(int j=0;j<80000;j++) {
    searchTree.add(range.get(j));
    redBlackTree.add(range.get(j));
    twoThreeTree.add(range.get(j));
    skipList.add(range.get(j));
    bTree.add(range.get(j));
}
```

```
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    searchTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_bst += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    redBlackTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_redBlack += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    twoThreeTree.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_twoSecond += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    skipList.add(range.get(k));
}
endTime = System.nanoTime();
estimatedTime = endTime - startTime;
average_skiplist += estimatedTime;
startTime = System.nanoTime();
for(int k=0;k<100;k++) {
    bTree.add(range.get(k));
}
endTime = System.nanoTime();
```

```
System.out.println("BST add operation with 80000 items time: " + average_bst/10);
System.out.println("Red-Black Tree add operation with 80000 items time: " + average_redBlack/10);
System.out.println("Two-Theree Tree add operation with 80000 items time: " + average_twoSecond/10);
System.out.println("Skip-List add operation with 80000 items time: " + average_skiplist/10);
System.out.println("B-tree add operation with 80000 items time: " + average_btree/10);
```

4. RUNNING COMMAND AND RESULTS

A. AVL TREE AND SKIP LIST

```
-----Define navigableset with avl tree-----
-----Insert operation avl tree-----
----- 1 2 3 4 5 added-----
-----Define iterator avl tree-----
-----Testing Iterator-----
1
2
3
4
5
-----Testing HeadSet operation until 4-----

-----Printing HeadSet Tree-----
1
2
3
-----Define navigableset with skiplist-----
-----Insert operation skip list-----
----- 1 2 3 4 5 added-----
-----Delete operation skip list - delete 4
4
-----Delete operation skip list - delete 100
null
-----Define descending iterator skip list-----
-----Testing Iterator-----
5
3
2
1
```

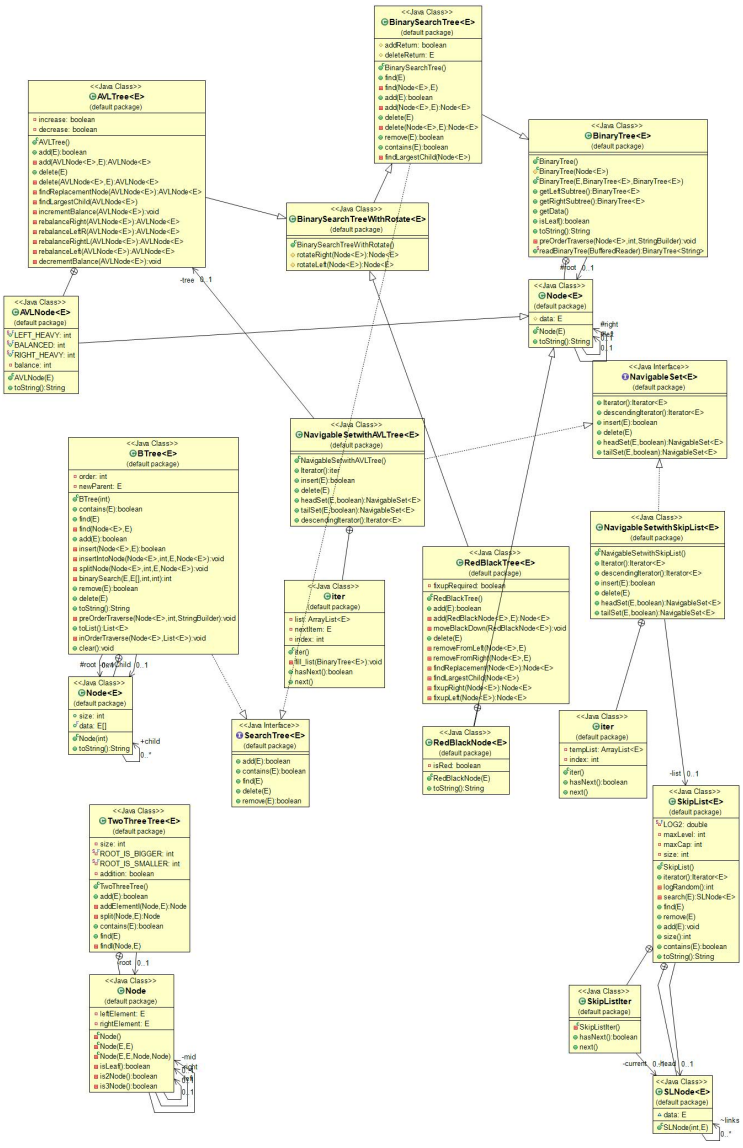
B. IS AVL TREE OR NOT

```
-----Define AVL Tree-----
-----Insert elements AVL Tree-----
-----Define BST Tree-----
-----Insert elements BST Tree-----
Is avl tree is avl tree ?
true
Is BST tree is avl tree ?
false
```

C. TEST AND COMPARE DATA STRUCTERS

```
--------------TEST WITH  SIZE (10000)-------------------
BST add operation with 10000 items time: 19660.1
Red-Black Tree add operation with 10000 items time: 25930.2
Two-Theree Tree add operation with 10000 items time: 25069.7
Skip-List add operation with 10000 items time: 37540.3
B-tree add operation with 10000 items time: 32420.0
--------------TEST WITH  SIZE (20000)-------------------
BST add operation with 20000 items time: 16589.8
Red-Black Tree add operation with 20000 items time: 24030.1
Two-Theree Tree add operation with 20000 items time: 19169.9
Skip-List add operation with 20000 items time: 31940.3
B-tree add operation with 20000 items time: 26639.8
--------------TEST WITH  SIZE (40000)-------------------
BST add operation with 40000 items time: 12390.3
Red-Black Tree add operation with 40000 items time: 19580.0
Two-Theree Tree add operation with 40000 items time: 14379.9
Skip-List add operation with 40000 items time: 27099.7
B-tree add operation with 40000 items time: 19920.2
--------------TEST WITH SIZE (80000)-------------------
BST add operation with 80000 items time: 15159.7
Red-Black Tree add operation with 80000 items time: 19439.9
Two-Theree Tree add operation with 80000 items time: 12950.3
Skip-List add operation with 80000 items time: 27590.1
B-tree add operation with 80000 items time: 17379.9
```

5. CLASS DIAGRAMS



6. GRAPH FOR COMPARİNG DATA STRUCTERS