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

Emre SEZER
1901042640

SYSTEM REQUIREMENTS:

I created this homework on Windows 10 using terminal (Java Development Kit).My java version is 11.0.8 .You need to compile “driver.java” in order to test my homework.There are java files, a folder named “Javadoc” that includes javadoc files and report file in my homework. Compiling part 3 may take 1 minute to execute.

PROBLEM SOLUTION APPROACH:

PART1:

I implemented AVLTree class. AVLTree extends BinarySearchTreeWithRotate. BinarySearchTreeWithRotate extends BinarySearchTree.AVLNode extends BinarySearchTree’s inner Node class.

NavigableSetAVLTree:

I used AVLTree as a private data field in NavigableSetAVLTree. Insert method uses AVLTree’s add method. Inner NavigableSetIterator class uses BinarySearchTree’s iterator as private data field. iterator() method returns a NavigableSetIterator. headset method returns a NavigableSetAVLTree with elements smaller or equal(depends on inclusive parameter) to input. tailset method returns a NavigableSetAVLTree with elements larger or equal(depends on inclusive parameter) to input. Inside headset and tailSet I am using NavigableSetIterator.

NavigableSetSkipList**:**

I implemented a SkipList class.At NavigableSetSkipList class I used SkipList as a private data field.For insert method I used SkipList’s add method. For delete method I used SkipList’s remove method. For descendingIterator I used SkipList’s iterator method(There is an inner class named SkipListIterator inside the SkipList).

PART2: I implemented methods for this part on BinarySearchTree class.

AVLTree: I wrote 2 recursive methods and 2 starter methods for this part. height() method calculated height of the given node. isAVLTree() method checks if any node’s balance is greater than 1 or smaller than -1.

Red-Black Tree: I wrote 3 methods for this part. countBlackNodes() is a recursive method. Checks if any route from root to any leaf has equal Black Nodes.checkChildren() method checks if any Red Node's child id Red.If finds such condition returns false.isRedBlackTree() method checks if root node is Black and uses the countBlackNodes() and checkChildren() methods to decide if given BinarySearchTree is RedBlackTree or not.

PART3:

I completed the implementations on the course book.

TEST CASES:

PART1(AVL Tree):

- Inserting elements to the NavigableSetAVLTree. Returns true.
- Inserting an element that already exist in the NavigableSetAVLTree. Returns false
- Printing elements in the NavigableSetAVLTree with iterator
- Trying HeadSet Method Without inclusive.Returns NavigableSetAVLTree.
- Trying HeadSet Method With inclusive .Returns NavigableSetAVLTree.
- Trying TailSet Method Without inclusive. Returns NavigableSetAVLTree.
- Trying TailSet Method With inclusive. Returns NavigableSetAVLTree.
- Trying HeadSet Method With Element that Smaller than Minimum Element of NavigableSetAVLTree. Returns NavigableSetAVLTree.
- Trying TailSet Method With Element that Larger than Maximum Element of NavigableSetAVLTree. Returns NavigableSetAVLTree.
- Trying Iterator with NavigableSet has no elements
- Trying HeadSet Method with NavigableSet has no elements. NavigableSetAVLTree with 0 elements.
- Trying TailSet Method with NavigableSet has no elements. Returns NavigableSetAVLTree with 0 elements.

PART1(Skip List):

- Inserting elements to the NavigableSeSkipList.Returns true.
- Inserting an element that already exist in the NavigableSeSkipList. Returns false
- Trying Descending Iterator
- Trying To Delete an Element That Exists on NavigableSet. Returns true.
- Trying To Delete an Element That Doesn't Exist on NavigableSet. Returns false
- Trying Descending Iterator

PART2(AVL Tree):

- Trying isAVLTree with Tree that has one element. Returns true.
- Trying isAVLTree with tree that left heavy(left-left). Returns false.
- Trying isAVLTree with tree that left heavy(left-right). Returns false.
- Trying isAVLTree with tree that right heavy(right-right). Returns false.
- Trying isAVLTree with tree that right heavy(right-left). Returns false.
- Trying isAVLTree with empty tree. Returns false.
-

PART2(Red-BlackTree):

- Trying isRedBlackTree with 1 element and it's root is RED. Return false.
- Trying isRedBlackTree with Order of Red-Black Tree. Returns true.
- Trying isRedBlackTree with tree that any route from root to leaf is not equal for every leaf. Returns false
- Trying isRedBlackTree with RED Node has RED Child. Returns false.
- Trying isRedBlackTree with 1 element and it's root is BLACK. Returns false.
- Trying isRedBlackTree with empty tree. Returns false.

SCREENSHOTS:

```
TESTING NavigableSet With AVLTree:

15 is added
20 is added
5 is added
0 is added
25 is added
45 is added
50 is added
100 is added
65 is added
35 is added
10 is added

Trying To Insert Same Item To to NavigableSet With AVLTree:
10 is already in the NavigableSet

Trying Iterator
0
5
10
15
20
25
35
45
50
65
100
Trying HeadSet Method Without inclusive
0
5
10
15
20
25
35
Trying HeadSet Method With inclusive
0
5
10
15
20
25
35
45
Trying TailSet Method Without inclusive
50
65
100
Trying TailSet Method With inclusive
45
50
65
100
Trying HeadSet Method With Element that Smaller than Minimum Element of NavigableSet
As you can see nothing is printed

Trying TailSet Method With Element that Larger than Maximum Element of NavigableSet
As you can see nothing is printed

Trying Iterator with NavigableSet has no elements
As you can see nothing is printed

Trying HeadSet Method with NavigableSet has no elements
As you can see nothing is printed

Trying TailSet Method with NavigableSet has no elements
As you can see nothing is printed
```

```
100
Trying TailSet Method With inclusive
45
50
65
100
Trying HeadSet Method With Element that Smaller than Minimum Element of NavigableSet
As you can see nothing is printed

Trying TailSet Method With Element that Larger than Maximum Element of NavigableSet
As you can see nothing is printed

Trying Iterator with NavigableSet has no elements
As you can see nothing is printed

Trying HeadSet Method with NavigableSet has no elements
As you can see nothing is printed

Trying TailSet Method with NavigableSet has no elements
As you can see nothing is printed


TESTING NavigableSet With Skip List:

15 is added
20 is added
5 is added
0 is added
25 is added
45 is added
50 is added
100 is added
65 is added
35 is added
10 is added

Trying To Insert Same Item To to NavigableSet With Skip List:
10 is already in the NavigableSet

Trying Descending Iterator:
100
65
50
45
35
25
20
15
10
5

Trying To Delete an Element That Exists on NavigableSet:
10 is deleted

Trying To Delete an Element That Exists on NavigableSet:
25 is deleted

Trying To Delete an Element That Doesn't Exist on NavigableSet:
41 is not deleted
Trying Descending Iterator After Removals:
100
65
50
45
35
20
15
5
As you can see 10 and 25 is not printed:
Trying Delete at the Empty NavigableSet:
4545 is not deleted
```

~/java/cse222/hw7

```
35
20
15
5
As you can see 10 and 25 is not printed:
Trying Delete at the Empty NavigableSet:
4545 is not deleted

TESTING PART2:


'\' means right child and '|' means left child in prints

Trying isAVLTree with Tree That Has 1 Element:
\-- 40 B
TREE IS AVL TREE
Trying isAVLTree with perfect tree:
\-- 40 B
    |-- 10 B
    |   |-- 0 B
    |   \-- 20 B
    \-- 60 B
        |-- 50 B
        \-- 70 B
TREE IS AVL TREE
Trying isAVLTree with tree that left heavy(left-left):
\-- 40 B
    |-- 10 B
    |   |-- 0 B
    |   |   |-- -10 B
    |   |   |   |-- -15 B
    |   \-- 20 B
    \-- 60 B
        |-- 50 B
        \-- 70 B
TREE IS NOT AVL TREE
Trying isAVLTree with tree that left heavy(left-right):
\-- 40 B
    |-- 10 B
    |   |-- 0 B
    |   |   |-- -10 B
    |   |   |   \-- -5 B
    |   \-- 20 B
    \-- 60 B
        |-- 50 B
        \-- 70 B
TREE IS NOT AVL TREE
Trying isAVLTree with tree that right heavy(right-right):
\-- 40 B
    |-- 10 B
    |   |-- 0 B
    |   \-- 20 B
    \-- 60 B
        |-- 50 B
        \-- 70 B
            \-- 80 B
                \-- 90 B
TREE IS NOT AVL TREE
Trying isAVLTree with tree that right heavy(right-left):
\-- 40 B
    |-- 10 B
    |   |-- 0 B
    |   \-- 20 B
    \-- 60 B
        |-- 50 B
        \-- 70 B
            \-- 90 B
                |-- 80 B
TREE IS NOT AVL TREE
Trying isAVLTree with empty tree:
TREE IS NOT AVL TREE
```

```
Trying isAVLTree with tree that right heavy(right-right):
\-- 40 B
    |-- 10 B
    |   |-- 0 B
    |   \-- 20 B
    \-- 60 B
        |-- 50 B
        \-- 70 B
            \-- 80 B
                \-- 90 B
TREE IS NOT AVL TREE
Trying isAVLTree with tree that right heavy(right-left):
\-- 40 B
    |-- 10 B
    |   |-- 0 B
    |   \-- 20 B
    \-- 60 B
        |-- 50 B
        \-- 70 B
            \-- 90 B
                |-- 80 B
TREE IS NOT AVL TREE
Trying isAVLTree with empty tree:
TREE IS NOT AVL TREE


'\' means right child and '|' means left child in prints

Trying isRedBlackTree with 1 element and it's root is RED:
\-- 40 R
TREE IS NOT RED-BLACK TREE
Trying isRedBlackTree with Tree Order of Red-Black Tree:
\-- 40 B
    |-- 10 R
    |   |-- 0 B
    |   \-- 20 B
    \-- 60 R
        |-- 50 B
        \-- 70 B
TREE IS RED-BLACK TREE
Trying isRedBlackTree with tree that any route from root to leaf is not equal for every leaf:
\-- 40 B
    |-- 10 B
    |   |-- 5 B
    \-- 60 R
        |-- 50 B
        \-- 70 B
TREE IS NOT RED-BLACK TREE
Trying isRedBlackTree with RED Node has RED Child:
\-- 40 B
    |-- 10 R
    |   |-- 0 B
    |   \-- 20 R
    \-- 60 R
        |-- 50 B
        \-- 70 B
TREE IS NOT RED-BLACK TREE
Trying isRedBlackTree with 1 element and it's root is BLACK:
\-- 40 B
    |-- 10 R
    |   |-- 0 B
    |   \-- 20 R
    \-- 60 R
        |-- 50 B
        \-- 70 B
TREE IS NOT RED-BLACK TREE
Trying isRedBlackTree Empty Tree:
TREE IS NOT RED-BLACK TREE
TESTING PART3:
IT MAY TAKE APPROXIMATELY 1 MINUTES TO COMPILE(Since, it is generating a lot of non repeating numbers)
```

```
    |   |-- 0 B
    |   \-- 20 R
    \-- 60 R
        |-- 50 B
        \-- 70 B
TREE IS NOT RED-BLACK TREE
Trying isRedBlackTree with 1 element and it's root is BLACK:
\-- 40 B
    |-- 10 R
    |   |-- 0 B
    |   \-- 20 R
    \-- 60 R
        |-- 50 B
        \-- 70 B
TREE IS NOT RED-BLACK TREE
Trying isRedBlackTree Empty Tree:
TREE IS NOT RED-BLACK TREE
TESTING PART3:
IT MAY TAKE APPROXIMATELY 1 MINUTES TO COMPILE(Since, it is generating a lot of non repeating numbers)

BINARY SEARCH TREE:
PROBLEM SIZE: 10000:
AVG TIME(ns): 668.0
PROBLEM SIZE: 20000:
AVG TIME(ns): 1128.0
PROBLEM SIZE: 40000:
AVG TIME(ns): 896.0
PROBLEM SIZE: 80000:
AVG TIME(ns): 1156.0

RED-BLACK TREE:
PROBLEM SIZE: 10000:
AVG TIME(ns): 666.0
PROBLEM SIZE: 20000:
AVG TIME(ns): 721.0
PROBLEM SIZE: 40000:
AVG TIME(ns): 857.0
PROBLEM SIZE: 80000:
AVG TIME(ns): 933.0

2-3 TREE:
PROBLEM SIZE: 10000:
AVG TIME(ns): 642.0
PROBLEM SIZE: 20000:
AVG TIME(ns): 823.0
PROBLEM SIZE: 40000:
AVG TIME(ns): 1127.0
PROBLEM SIZE: 80000:
AVG TIME(ns): 1994.0

B-TREE(ORDER 10):
PROBLEM SIZE: 10000:
AVG TIME(ns): 515.0
PROBLEM SIZE: 20000:
AVG TIME(ns): 591.0
PROBLEM SIZE: 40000:
AVG TIME(ns): 1092.0
PROBLEM SIZE: 80000:
AVG TIME(ns): 1091.0

SKIP LIST:
PROBLEM SIZE: 10000:
AVG TIME(ns): 1189.0
PROBLEM SIZE: 20000:
AVG TIME(ns): 1271.0
PROBLEM SIZE: 40000:
AVG TIME(ns): 1681.0
PROBLEM SIZE: 80000:
AVG TIME(ns): 1763.0

emr3s@DESKTOP-POGAK7B ~/java/cse222/hw7
$
```

DIAGRAM:

NavigableSetSkipListInterface<E>

NavigableSetSkipList<E>
- data : ConcurrentSkipListSet <E>

+ insert(E item) : boolean
+ delete(E item) : boolean
+ descendingIterator() : Iterator

BinarySearchTree<E>
data : E
- size : int
addReturn : boolean

+ size() : int
+ add(E item, boolean red) : boolean
+ find(E item) : E
+ search(E item) : Node<E>
- find(Node<E> localRoot, E item) : Node <E>
+ height()
- height(E item) : int
+ isAVLTree(Node <E> node) : boolean
+ print() : void
- print() : void
+ isRed() : boolean
- countBlackNodes(Node <E> node, ArrayList<Integer> list, int counter) : void
+ checkChildren(Node <E> node, boolean isParentRed) : boolean
+ isRedBlackTree() : boolean
+ iterator() : Iterator
+ BinarySearchTree()
+ add(E item) : boolean
- add(Node<E> localRoot, E item, boolean red): boolean
- height(Node<E> localRoot) : int

Node<E>
data : E
left : Node<E>
right : Node<E>
isRed : boolean

+ toString() : String

BSTIterator<E>
-stack : Stack<E>

+hasNext() : boolean
+ next() : E

SkipList<E>
- head : SLNode <E>
- maxLevel : int
- maxCap : int
- size : int

+ SkipList()
+ search (E target) : SLNode<E>[]
+ find(E target) : E
+ add (E item) : void
- logRandom () : void
- computeMaxCap (int maxLevel) : int

SLNode <E>
- links : SLNode<E>[]
- data : E

+ SLNode (int m, E data)

BTree <E>
- root : Node<E>
- newChild : Node <E>
- newParent : E
- order : int

+operation()
+ add(E item) : boolean
- insert(Node<E> root, E item) : boolean
- insertIntoNode(Node<E> node, int index, E obj, Node<E> child)
- splitNode(Node<E> node, int index, E item, Node<E> child)
- binarySearch(E item, E[] data, int firstIndex, int size) : int

TwoThreeTree
- data : BTree <E>

+ TwoThreeTree()
+ add(E item)

Node<E>
- size : int
- data : E[]
- child : Node<E>[]

+ Node(int order)

NavigableSetAVLTreeInterface<E>

BinarySearchTreeWithRotate<E>

+ rotateRight() : void
+ rotateLeft() : void

SkipListIterator<E>

+ hasNext() : boolean
+ next : E

NavigableSetAVLTree<E>
- data : AVLTree <E>
- size : int

+ NavigableSetAVLTree()
+ insert(E item) : boolean
+ headSet(E toElement, boolean inclusive) : NavigableSetAVLTree
+ headSet(E toElement) : NavigableSetAVLTree
+ tailSet(E toElement, boolean inclusive) : NavigableSetAVLTree
+ tailSet(E toElement) : NavigableSetAVLTree
+ iterator : Iterator

AVLTree<E>
- increase : boolean
- root : AVLNode<E>

- incrementBalance(AVLNode<E> node) : void
- decrementBalance(AVLNode<E> node) : void
- rebalanceRight(AVLNode<E> localRoot) : AVLNode<E>
- rebalanceLeft(AVLNode<E> localRoot) : AVLNode<E>
- add(AVLNode<E> localRoot, E item) : AVLNode<E>
+ add(E entry) : boolean
+ print() : void
- print(Node <E> node) : void
+ iterator() : Iterator

RedBlackTree<E>

+RedBlackTree()
+ add(E item) : boolean
+ add(Node<E> localRoot, E item) : Node <E>
- moveBlackDown(Node<E> tempRoot) : boolean

AVLNode<E>
- LEFT_HEAVY : int
- BALANCED : int
- RIGHT_HEAVY : int

+ AVLNode(E item)
+ toString() : String

NavigableSetIterator<E>
- iter : Iterator

+ NavigableSetIterator()
+operation()
+ hasNext() : boolean
+ next : E

PART 3 TEST RESULTS:

(Time Unit is nano seconds)

	BinarySearchTree	Red-Black Tree	2-3 Tree	BTree(Order 10)	Skip List
10000	668	666	642	515	1189
20000	1128	721	823	591	1271
40000	896	857	1127	1092	1681
80000	1156	933	1994	1091	1763

Comparisons:

Problem Size = 10000:

BTree < 2-3 Tree < Red-Black Tree < BinarySearchTree < Skip List

Problem Size = 20000:

BTree < Red-Black Tree < 2-3 Tree < BinarySearchTree < Skip List

Problem Size = 40000:

Red-Black Tree < BinarySearchTree < BTree < 2-3 Tree < Skip List

Problem Size = 80000:

Red-Black Tree < BTree < BinarySearchTree < Skip List < 2-3 Tree

At the Next Graph:

Green Lines Represent: BTree (Order 10)

White Lines Represent: Red-Black Tree

Red Lines Represent: BinarySearchTree

Yellow Lines Represent: 2-3 Tree

Purple Line Represents: Skip List

TIME-PROBLEM SIZE GRAPH FOR DATA STRUCTURES