CSE 222 DATA STRUCTURES AND ALGORITHMS

HOMEWORK 7

REPORT 7

MERVE HORUZ
1801042651

1) Run time complexities

a)

```
public BinaryTree<E> generateBST(BinaryTree<E> bt, E [] items){

    if(bt != null) {          // O(1)
        BinaryTree.Node<E> temp = bt.root;     // O(1)
        sortArray(items);  // insertion sort algorithm is O(n)

        // If a tree has n nodes, then each node is visited only once in inorder traversal and hence the complexity is O(n).
        sortedArrayToBST(items, temp);

        temp = bt.root;    // O(1)
        System.out.println("\n\n----------After converting from binary tree to binary search tree(inorder traversal)---------\n");
        printInorder(temp);    // O(n)
        System.out.println("\n");
    }
    else
        System.out.println("\nThere is no structure to put elements...\n");

    return bt;
}
```

→ so run time complexity is O(n).

b)

```
public BinarySearchTree<E> rotateBSTtoAVL(BinarySearchTree<E> bst){

    if(bst != null) {    //  O(1)
        int leftHeight = findHeight(bst.root.left);   // O(n)
        int rightHeight = findHeight(bst.root.right);   // O(n)

        while (Math.abs(leftHeight - rightHeight) > 1) {   //  O(n)
            if (leftHeight < rightHeight)   //  O(1)
                bst.root = rotateLeft(bst.root);   //  O(1)
            else if (leftHeight > rightHeight)   //  O(1)
                bst.root = rotateRight(bst.root);   //  O(1)

            leftHeight = findHeight(bst.root.left);    // O(n)
            rightHeight = findHeight(bst.root.right);    // O(n)
        }
    }
    else
        System.out.println("\nThere is no tree to convert to avl tree...\n");
    return bst;
}
```

→ so run time complexity is O(n^2).

c)

```
public boolean add(E item){
    size++;     // O(1)
    SLNode<E>[] pred = search(item);     // O(logn)
    if(size % 10 == 0){   // O(1)
        maxLevel++;   // O(1)
        maxCap = computeMaxCap(maxLevel);   // O(1)
        head.links = Arrays.copyOf(head.links, maxLevel);   // O(n)
        pred = Arrays.copyOf(pred, maxLevel);     // O(n)
        pred[maxLevel - 1] = head;    // O(1)
    }
    SLNode<E> newNode = new SLNode<E>(logRandom(), item);    // O(1)
    for(int i = 0; i < newNode.links.length; i++){    // O(m)
        newNode.links[i] = pred[i].links[i];   // O(1)
        pred[i].links[i] = newNode;     // O(1)
    }
    /*SLNode<E> itr = head;      // O(1)
    while(itr.links[0] != null){     // O(n)
        if(itr.links.length > 1)    // O(1)
            increaseLevel(itr.data);    // O(n)
        itr = itr.links[0];      // O(1)
    }*/
    return true;
}
```

→ so run time complexity is O(n^2).

2) Problem solution approach

a) First, we mixed all the elements we wanted to put in a binary search tree in an array. I then added zeros to each node to determine the structure of the binary tree. We sent these two arguments as parameters to the method we wrote and the array was sorted using insertion sort, and then the items in the array were placed one by one using the inorder traversal algorithm to each node of the binary tree.

b) First, coding was started by finding the heights of the left and right subtrees. If the difference in heights of the left and right subtrees is greater than 1, we rotate the tree to the right. If the difference in heights of the left and right subtrees is less than -1, we rotate the tree to the left and loop it until the tree stabilizes.

c) First, skip list data structure is coded. After that the skip list has two level lists in default. A new level list is added to the skip list each time when the size of the first level list reaches powers of 10.

// This matter is coded but could not ran exactly so it is committed The tall items (contained in the more than one level) are appended to a one-level upper list when a new level is added to the skip list.

3) Detailed system requirements

3.1) non-functional requirements

openjdk 17.0.3 2022-04-19
OpenJDK Runtime Environment (build 17.0.3+7-Ubuntu-0ubuntu0.20.04.1)
OpenJDK 64-Bit Server VM (build 17.0.3+7-Ubuntu-0ubuntu0.20.04.1, mixed mode, sharing)

3.2) functioanal requirements

public BinaryTree<E> generateBST(BinaryTree<E> bt, E [] items) ----------

- This method requires an array that contains unique items.
- This method requires a structured binary tree to puts items.

public BinarySearchTree<E> rotateBSTtoAVL(BinarySearchTree<E> bst)----------

- This method requires a binary search tree to convert avl tree.

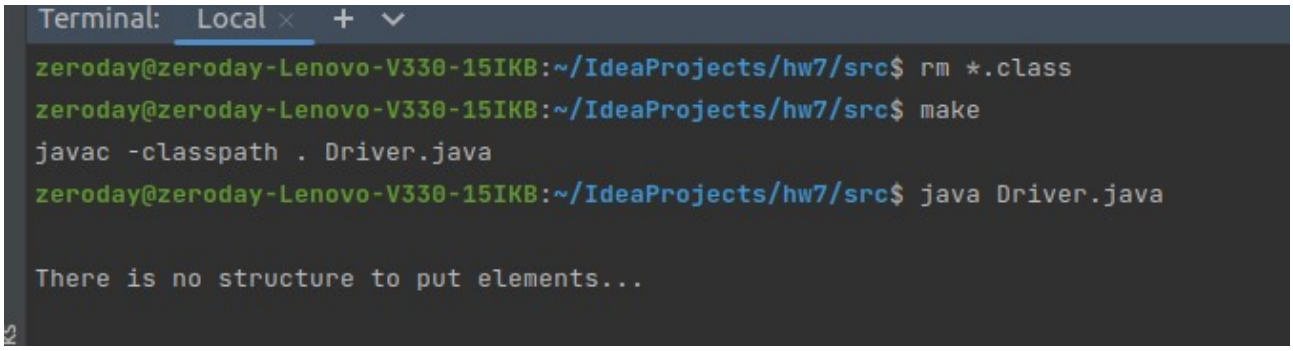
public boolean add(E item)----------

- This method requires a generic item to add skip list.

4) Class diagram

5) Test cases

if binary tree that will convert to binary search tree is null and binary search tree that will convert to avl tree is null, then that is resulted as warning.



6) Running commands

- make
- java Driver.java
- if you would like to run one more time remove .class file -rm *.class

7) Running results

```
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/hw7/src$ java Driver.java
CONVERTING BINARY TREE TO BINARY SEARCH TREE

--------------Array before converting from binary tree to binary search tree----------------

7 9 4 3 2 5 6 8 1 10



----------After converting from binary tree to binary search tree(inorder traversal)---------

1 2 3 4 5 6 7 8 9 10
```

```
CONVERTING BINARY SEARCH TREE TO AVL TREE

--------------Before converting from binary search tree to avl tree----------------

21
  15
    14
      9
        7
          5
            3
              null
              null
            null
          null
        null
      null
    17
      null
      null
  33
    30
      null
      null
    34
      null
      null
```

```
---------------After converting from binary search tree to avl tree----------------

14
  9
    7
      5
        3
          null
          null
        null
      null
    null
  15
    null
    21
      17
        null
        null
      33
        30
          null
          null
        34
          null
          null
```

```
IMPLEMENTING CUSTOM SKIP LIST

----------------------Adding 24 elements to skipList----------------------

Head: 4 --> 1 |3| --> 2 |1| --> 5 |1| --> 6 |1| --> 7 |1| --> 9 |2| --> 11 |1| --> 14 |1| --> 15 |1| --> 21 |2|
 --> 28 |1| --> 33 |2| --> 35 |3| --> 63 |1| --> 67 |2| --> 68 |1| --> 79 |4| --> 115 |2| --> 116 |1| --> 121 |1|
 --> 147 |3| --> 213 |2| --> 435 |1| --> 1271 |2|
----------------------Adding 13 elements to skipList----------------------

Head: 3 --> 1 |1| --> 2 |1| --> 5 |1| --> 6 |1| --> 7 |1| --> 14 |1| --> 15 |1| --> 21 |1| --> 33 |2| --> 67 |1|
 --> 121 |1| --> 213 |1| --> 435 |1|
```