Lab 12: Binary search trees

Professor: Ronaldo Menezes

TA: Ivan Bogun

Department of Computer Science
Florida Institute of Technology

April 2, 2014

1 General description

1.1 Legend

In this lab you are asked to implement a generic data structure representing binary search tree (BST).

2 Implementation

Create the class *BST.java* which implements interface *Tree.java*. Interface is given below.

```
//Tree.java
public interface Tree<Key extends Comparable<Key>>{

        public void insert(Node<Key> node);
        // insert the node into the tree

        public Node<Key> delete(Node<Key> node);
        // delete the node from the tree,

        public void insert(Key key);
        // insert the node into the tree

        public Node<Key> delete(Key key);
        // delete the node from the tree, if key is not present null should be
            returned

        public boolean search(Key key);
        //return true if the key is present or false otherwise

        public void inorderTreeWalk(Node<Key> x);
        // print inorder tree traversal of the tree

        public Node<Key> findMinimum(Node<Key> node);
        // find the minimum of the tree;

        public Node<Key> findMaximum(Node<Key> node);
        // find the maximum of the tree

        public String showTree();
        // return a String representation of a level-order traversal of a tree

}
```

Class *Node.java* is given below

```
public class Node<Key extends Comparable<Key>> {
```

```
private Node<Key> left;
private Node<Key> right;
private Key key;
private Node<Key> parent;
private int height;


public String toString(){
// return a string representing level-order traversal of a tree rooted
    at this node
}

public void inorderTreeWalk(){
// print inorder tree walk of the tree rooted at this node (elements to
    be printed should be in sorted order)
}
}
```

3 Extra-credit(+25)

Implement a class called *BalancedBST.java* which will be a balanced binary search tree (AVL or Red-Black tree).

4 Sample input-output

Use the following main for testing.

4.1 Input

```
public static void main(String[] args) {

// get random object
Random rgen = new Random();
rgen.setSeed(15);
BST<Integer> integerTree=new BST<Integer>();

// create array of characters
Integer[] integers={9,5,2,3,6,10,57,8};

// shuffle it
Collections.shuffle(Arrays.asList(integers),rgen);

System.out.print("To insert: ");
for (int i = 0; i < integers.length; i++) {
       System.out.print(integers[i]+" ");
```

```
            integerTree.insert(integers[i]);
        }
        System.out.println();
        // printout level-order traversal
        System.out.println(integerTree.showTree());

        // get random object
        BST<Character> letterTree=new BST<Character>();

        // create array of characters
        Character[] letters={'A','B','C','D','E','F','G','H','I','X'};

        // shuffle it
        Collections.shuffle(Arrays.asList(letters),rgen);
        System.out.print("To insert: ");
        for (int i = 0; i < letters.length; i++) {
            System.out.print(letters[i]+" ");
            letterTree.insert(letters[i]);
        }
        System.out.println();
        // printout level-order traversal
        System.out.println(letterTree.showTree());

        System.out.println("Inorder tree walk:");
        letterTree.getRoot().inorderTreeWalk();
        System.out.println();

        // delete some characters from a tree
        letterTree.delete('A');
        letterTree.delete('X');
        letterTree.delete('I');

        System.out.println("Tree after few elements were deleted:");
        System.out.println(letterTree.showTree());

        // search for a subtree rooted at 'G'
        Node<Character> nodeWithLetter=letterTree.searchNode('G');

        System.out.println("Subtree rooted at 'G':");

        // print level-order traversal of a tree rooted at nodeWithLetter
        System.out.println(nodeWithLetter.toString());

    }
```

4.2 Output

```
To insert: 5  9  57  3  2  6  8  10
```

```
                              [5 ]
                [3 ]                                  [9 ]
          [2 ]                --                [6 ]                [57 ]
      --      --      --      --      --      [8 ]       [10 ]       --

To insert: I  D  E  C  G  B  H  X  F  A
                              [I ]
                [D ]                                  [X ]
          [C ]                [E ]                --                --
     [B ]       --      --      [G ]       --      --      --      --
   [A ]   --  --  --  --  --  [F ]   [H ]   --  --  --  --  --  --  --  --

Inorder tree walk:
A B C D E F G H I X
Tree after few elements were deleted:
                              [D ]
                [C ]                                  [E ]
          [B ]                --                --                [G ]
      --      --      --      --      --      --      [F ]       [H ]

Subtree rooted at 'G':
                              [G ]
                [F ]                                  [H ]
```

5 Grade breakdown

basis	grade
Implementation	(60)
test cases	30
tree functions	30
Comments	(20)
Javadocs	10
General	10
Overall	(20)
Compiled	5
Style	5
Runtime	10
Total	100