Assignment 4 (Maximum marks: 20) – BINARY SEARCH TREE- INSERT DATA FROM FILE

CIS 265 - Spring 2018

Due Date: April 15, 2018 (Sunday)

This assignment will help you get familiar with Binary Search Trees and the concepts of traversing the tree using techniques of Inorder, Preorder and Postorder, and searching, adding and removing nodes. The class lecture slides will also help you complete the assignment.

Problem Description:

city.txt has a list of cities which are added using Binary search tree insert algorithm.

- Cities are to be traversed in **Inorder, Preorder and Postorder** from the root **WITHOUT USING RECURSION**. You need to use Stack push() and pop() to accomplish your task.
- Then you need to **search** whether a city exists in the tree or not. If the city exist, its path from the root is displayed for you, else display false.

1) Write the code for the following methods to implement traversal BST in *BST.java*:

- protected void inorder(TreeNode<E> root)

- protected void postorder(TreeNode<E> root)

- protected void preorder(TreeNode<E> root)

- public boolean search(E e)

2) Run the program for city.txt already provided and see if the traversal output is displayed correctly.

Below is the file in which students will have to write their code wherever they find a comment: */* YOUR CODE HERE */*

BST.java

```
package bst;

import java.util.Stack;

public class BST<E extends Comparable<E>> implements Tree<E> {
        protected TreeNode<E> root;
        protected int size = 0;

        /** Create a default binary tree */
        public BST() {
        }

        /** Create a binary tree from an array of objects */
        public BST(E[] objects) {
          for (int i = 0; i < objects.length; i++)
            add(objects[i]);
        }

        @Override /** Returns true if the element is in the tree */
```

```java
  public boolean search(E e) {
    TreeNode<E> current = root; // Start from the root

    /* YOUR CODE HERE */

    return false;
  }

  @Override /** Insert element e into the binary tree
   * Return true if the element is inserted successfully */
  public void insert(E e) {
        root = insertRecord(root, e);
  }
  @SuppressWarnings("unchecked")
public TreeNode insertRecord(TreeNode root, E e){
        if (root == null) {
            root = new TreeNode(e);
            System.out.print(root.element + " ");
            size++;
            return root;
        }
        if (e.compareTo((E) root.element) < 0)
            root.left = insertRecord(root.left, e);
        else if (e.compareTo((E) root.element) > 0)
            root.right = insertRecord(root.right, e);
        return root; // Element inserted successfully
  }

  protected TreeNode<E> createNewNode(E e) {
    return new TreeNode<>(e);
  }

  @Override /** Inorder traversal from the root */
  public void inorder() {
    inorder(root);
  }

  /** Inorder traversal from a subtree */
  protected void inorder(TreeNode<E> root) {

      /* YOUR CODE HERE */
  }

  @Override /** Postorder traversal from the root */
  public void postorder() {
    postorder(root);
  }

  /** Postorder traversal from a subtree */
  protected void postorder(TreeNode<E> root) {

      /* YOUR CODE HERE */
  }
```

```java
@Override /** Preorder t7raversal from the root */
public void preorder() {
  preorder(root);
}

/** Preorder traversal from a subtree */
protected void preorder(TreeNode<E> root) {

    /* YOUR CODE HERE */
}

@Override /** Get the number of nodes in the tree */
public int getSize() {
  return size;
}

/** Returns the root of the tree */
public TreeNode<E> getRoot() {
  return root;
}

/** Returns a path from the root leading to the specified element */
public java.util.ArrayList<TreeNode<E>> path(E e) {
  java.util.ArrayList<TreeNode<E>> list =
    new java.util.ArrayList<>();
  TreeNode<E> current = root; // Start from the root

  while (current != null) {
    list.add(current); // Add the node to the list
    if (e.compareTo(current.element) < 0) {
      current = current.left;
    }
    else if (e.compareTo(current.element) > 0) {
      current = current.right;
    }
    else
      break;
  }

  return list; // Return an array list of nodes
}

@Override /** Delete an element from the binary tree.
 * Return true if the element is deleted successfully
 * Return false if the element is not in the tree */
public boolean delete(E e) {
  // Locate the node to be deleted and also locate its parent node
  TreeNode<E> parent = null;
  TreeNode<E> current = root;
  while (current != null) {
    if (e.compareTo(current.element) < 0) {
      parent = current;
      current = current.left;
    }
    else if (e.compareTo(current.element) > 0) {
```

```
      parent = current;
      current = current.right;
    }
    else
      break; // Element is in the tree pointed at by current
  }

  if (current == null)
    return false; // Element is not in the tree

  // Case 1: current has no left child
  if (current.left == null) {
    // Connect the parent with the right child of the current node
    if (parent == null) {
      root = current.right;
    }
    else {
      if (e.compareTo(parent.element) < 0)
        parent.left = current.right;
      else
        parent.right = current.right;
    }
  }
  else {
    // Case 2: The current node has a left child
    // Locate the rightmost node in the left subtree of
    // the current node and also its parent
    TreeNode<E> parentOfRightMost = current;
    TreeNode<E> rightMost = current.left;

    while (rightMost.right != null) {
      parentOfRightMost = rightMost;
      rightMost = rightMost.right; // Keep going to the right
    }
    // Replace the element in current by the element in rightMost
    current.element = rightMost.element;

    // Eliminate rightmost node
    if (parentOfRightMost.right == rightMost)
      parentOfRightMost.right = rightMost.left;
    else
      // Special case: parentOfRightMost == current
      parentOfRightMost.left = rightMost.left;
  }
  size--;
  return true; // Element deleted successfully
}

@Override /** Obtain an iterator. Use inorder. */
public java.util.Iterator<E> iterator() {
  return new InorderIterator();
}

@Override /** Remove all elements from the tree */
public void clear() {
```

```
            root = null;
            size = 0;
        }
    }
```

Java helper files (Students DO NOT need to change them):

TestBST.java

```
package bst;

import java.io.BufferedReader;

import java.io.File;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.FileInputStream;

import java.util.Scanner;

public class TestBST {

	public static void main(String[] args) {

		// Create a BST

		BST<String> tree = new BST<>();

		try { // reads in words from a file

			String word;

			BufferedReader diskInput = new BufferedReader(new InputStreamReader(new
FileInputStream(new File(args[0]))));

			Scanner input = new Scanner(diskInput);

			while (input.hasNext()) {

				word = input.next();

				word = word.toLowerCase();

				tree.insert(word);

			}

		} catch (IOException e) {

			System.out.println("io exception");

		}
```

```
                System.out.println();
                // Traverse tree
                System.out.print("Inorder: ");
                tree.inorder();
                System.out.print("\nPostorder: ");
                tree.postorder();
                System.out.print("\nPreorder: ");
                tree.preorder();
                System.out.print("\nThe number of nodes is " + tree.getSize());
                // Search for an element
                boolean found = false;
                found = tree.search("athens");
                System.out.print("\nIs athens in the tree? " + found);
                if(found){
                        // Get a path from the root to athens
                        System.out.print("\nA path from the root to athens is: ");
                        java.util.ArrayList<TreeNode<String>> path = tree.path("athens");
                        for (int i = 0; path != null && i < path.size(); i++)
                                System.out.print(path.get(i).element + " ");
                }
                found = false;
                found = tree.search("assignment4");
                System.out.print("\nIs assignment4 in the tree? " + found);
        }
}
```

InorderIterator.java

```
package bst;
import bst.TreeNode;

// Inner class InorderIterator
```

```
public class InorderIterator<E extends Comparable<E>> implements
java.util.Iterator<E> {
	// Store the elements in a list
	private java.util.ArrayList<E> list = new java.util.ArrayList<>();
	private int current = 0; // Point to the current element in list
	protected TreeNode<E> root;
	public InorderIterator() {
		inorder(); // Traverse binary tree and store elements in list
	}
	/** Inorder traversal from the root */
	private void inorder() {
		inorder(root);
	}
	/** Inorder traversal from a subtree */
	private void inorder(TreeNode<E> root) {
		if (root == null)
			return;
		inorder(root.left);
		list.add(root.element);
		inorder(root.right);
	}
	@Override /** More elements for traversing? */
	public boolean hasNext() {
		if (current < list.size())
			return true;
		return false;
	}

	@Override /** Get the current element and move to the next */
	public E next() {
		return list.get(current++);
	}
}
```

Tree.java

```
package bst;

import java.util.Collection;

public interface Tree<E> extends Collection<E> {
  /** Return true if the element is in the tree */
  public boolean search(E e);

  /** Insert element e into the binary tree
   * Return true if the element is inserted successfully */
  public void insert(E e);

  /** Delete the specified element from the tree
   * Return true if the element is deleted successfully */
  public boolean delete(E e);

  /** Get the number of nodes in the tree */
  public int getSize();
```

```java
/** Inorder traversal from the root*/
public default void inorder() {
}

/** Postorder traversal from the root */
public default void postorder() {
}

/** Preorder traversal from the root */
public default void preorder() {
}

@Override /** Return true if the tree is empty */
public default boolean isEmpty() {
  return this.size() == 0;
};

@Override
public default boolean contains(Object e) {
  return search((E)e);
}

@Override
public default boolean add(E e) {
  return true;
}

@Override
public default boolean remove(Object e) {
  return delete((E)e);
}

@Override
public default int size() {
  return getSize();
}

@Override
public default boolean containsAll(Collection<?> c) {
  // Left as an exercise
  return false;
}

@Override
public default boolean addAll(Collection<? extends E> c) {
  // Left as an exercise
  return false;
}

@Override
public default boolean removeAll(Collection<?> c) {
  // Left as an exercise
  return false;
}
```

```
  @Override
  public default boolean retainAll(Collection<?> c) {
    // Left as an exercise
    return false;
  }

  @Override
  public default Object[] toArray() {
    // Left as an exercise
    return null;
  }

  @Override
  public default <T> T[] toArray(T[] array) {
    // Left as an exercise
    return null;
  }
}
```

TreeNode.java

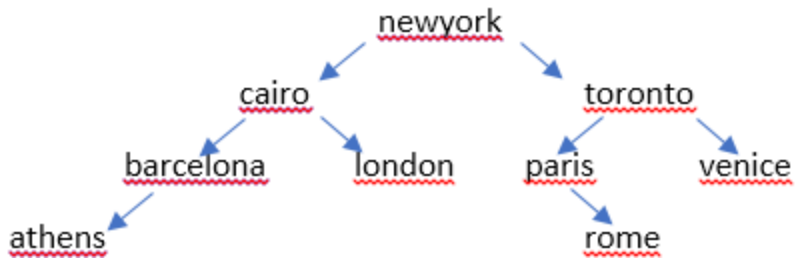
```
package bst;
/**
 * This inner class is static, because it does not access any instance members
 * defined in its outer class
 */
public class TreeNode<E> {
	protected E element;
	protected TreeNode<E> left;
	protected TreeNode<E> right;

	public TreeNode(E e) {
		element = e;
	}
}
```

city.txt

```
newyork
cairo
toronto
barcelona
london
paris
rome
athens
venice
```

SAMPLE OUTPUT:

Inserted tree to traverse:




```
newyork cairo toronto barcelona london paris rome athens venice
Inorder: athens barcelona cairo london newyork paris rome toronto venice
Postorder: athens barcelona london cairo rome paris venice toronto newyork
Preorder: newyork cairo barcelona athens london toronto paris rome venice
The number of nodes is 9
Is athens in the tree? true
A path from the root to athens is: newyork cairo barcelona athens
Is assignment4 in the tree? false
```

Please note a few points:

1. Create 5 separate java files for the above code and place them in a package named bst.
2. Create city.txt file and place it anywhere and pass it as argument while running TestBST.java
3. Name the files exactly as written above and modify the code only where expected. Do not modify any other piece of code.
4. Write your code in BST.java wherever you find the comment */* YOUR CODE HERE */*
5. DO NOT USE RECURSION for the implementation. Students are expected to use Stack push() and pop() technique.
6. Students can be asked to give a code walkthrough or rewrite the code in the Lab if the code is found to be plagiarized.
7. Correct traversal output is expected.
8. Source code should be well formatted as discussed in previous assignments.
9. Please ask all your questions to Professor/TA at least 2 days before submission deadline to expect a quick response.

What to turn in:

1. Attach the java file (BST.java ONLY) and output screenshot and send it on TA's email ID: **cis265assignment@gmail.com** and upload the same submission on Blackboard course message. No need to zip them in the email unless really required.
2. Do not send any additional file(s) in the email.
3. In the body of the email, please write a few lines about your approach of implementation.
4. Please keep your partner in 'cc' of the email.
5. Please keep email subject line as: **Assignment-<assignment number> <Section number> (First name last name & first name Last name)**
   For Example: Assignment-4 Section 1 (Sanchita Mal-Sarkar & Labhesh Popli)