CSE 2011: Assignment 2

Due Date - Monday, May 11, by Noon !

Question 1 **Binary Tree Traversal** [15 points]

The recursive implementations of Preorder and Inorder Traversal on a binary tree are provided below.

```
void recursivePreorder(Node t) {
        if (t != null) {
                processNode(t);
                recursivePreorder(t.getLeft());
                recursivePreorder(t.getRight()); }
}

void recursiveInorder(Node t) {
        if (t != null) {
                recursiveInorder(t.getLeft());
                processNode(t);
                recursiveInorder(t.getRight()); }
}
```

Write a corresponding non-recursive version of both above methods. (Hint: you are allowed to use an additional data structure, e.g. a stack or a queue.)

Question 2 **Binary Search Tree** [15 points]

In a binary search tree, in addition to the standard operations, we want to support an operation called *findLargest*(K), that takes integer K as an argument and returns the K-th largest key in the tree. In order to implement this method, we are allowed to add new attributes to BTNode class (see textbook, pp. 288).
In your answer to this question, the following should be clearly explained:

1) what new attribute(s) we need to store at each node;
2) how these attributes should be used to perform the *findLargest*(K) operation, and
3) how these attributes should be updated during insertion and deletion of new nodes in the tree.

In order to obtain full credit, your findLargest(K) should run in O(h) time (where h is the height of the given binary search tree), and the complexities of insert and delete should be the same as in the standard binary search tree implementation.

.

Question 3 BST-based Text Analyzer [70 points]

In this project, you are required to write a program that builds a binary search tree (BST) of distinct words from an input text file. Each time a new word occurs in the text, it is inserted into the tree; if the word has previously occurred, a count associated with the word is incremented. (Note that you will have to parse the text obtained from the file to find the words before inserting them into the tree. Furthermore, you will have to replace all capital letters with their lower case equivalents and remove punctuation.)

Once all words are inserted into the BST, the following three experiments should be performed.

1) Compute the maximum length of all search paths in the BST.
2) Print all distinct words found in the text (in sorted/alphabetic order).
3) Find the ten most common words in the text. You should output both the word and the number of times it occurred.

Your program design should be based on the following guidelines:

(1) Create BTNode[1] class, which will be used to store each distinct word together with is occurrence-counter (wordCounter). The outline of this class is given below.

```
public class  BTNode implements Position {
        String word;
        int wordCounter=0;
        BTNode left, right, parent;
        public BTNode(String s, BTNode u, BTNode v, BTNode w) { … }
        public String element() { … }
        public int getWordCounter() { … }
        public void increaseWordCounter() { … }
        public BTNode getLeft() { … }
        public void setLeft(BTNode v) { … }
        public BTNode getRight() { … }
        public void setRight(BTNode v) { … }
        public BTNode getParent() { … }
        public void setParent(BTNode v) { … }
}
```

(2) Create LinkedBinaryTree class as outlined in the textbook and class-notes.

(3) Create BinarySearchTree class, which will be used to store all distinct words form a given file. The outline of this class is given below. You are allowed to add new methods to this class, as needed.

```
public class BinarySearchTree {

LinkedBinaryTree T;

  /* readIn performs the following opearations:
     (1)  reads from fileName word by word
     (2)  for every new word found in fileName a new BTNode is created and added to T
          according to the rules of BST insertion
     (3)  for every duplicate word wordCounter of the corresponding BTNode is incremented  */
```

[1] Here BT stands for Binary Tree.

```
    public void readIn(String fileName) { }

    /* maxSearchPath finds and returns the length of the longest search path in T */
    public int maxSearchPath() { }

    /* printWordsSorted prints all distinct words found in fileName, in sorted order */
    public void printWordsSorted() { }

    /* printTenMostCommonWords finds and prints ten most common words from fileName together
    with their respective wordCounter-s /
    public void printTenMostCommonWords() { }

}
```

(4) Create TextAnalyzer class, which will contain main() method and act as a tester class. The outline of this class is given below. You are allowed to add new methods to this class, as needed.

```
public class TextAnalyzer {

        public static void main(String[] args) {

                BinarySearchTree BST = new BinarySearchTree();
                BST.readIn("WisdomForRoad.txt");
                System.out.println(BST.maxSearchPath());
                BST.printWordsSorted();
                BST.printTenMostCommonWords();

        }
}
```

The file to be used to test your program is available at:
http://www.cs.yorku.ca/course/2011/WisdomForRoad.txt