Department of Electrical Engineering and Computer Science
Lassonde School of Engineering
York University, Winter 2017

EECS 2011: Assignment 2

Due Date - Tuesday, Mar 28, in class!!!

Question 1 **Property of Binary Trees** [15 points]

Assume a proper and complete binary tree (T) with n>1 nodes. Let **E(T)** represent the sum of the depths of all external nodes in T (see Figure1), and **I(T)** represent the sum of the depths of all internal nodes in T. Prove that:

(a) **E(T) = O(n·log(n))**
(b) **E(T) – I(T) = 2i**, where i represents the actual number of internal nodes in T.

(Hint: for part (b) use mathematical induction.)

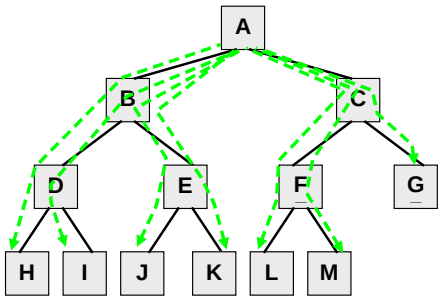

Figure 1 External Path Length

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

Given a BST and two numbers – a and b (where a≤b) – propose an algorithm that prints all the elements k of the BST that satisfy: k≥a and k≤b. Your algorithm should return the found keys in ascending order. You can assume that there are no duplicate keys in the tree.
(You can also assume that a, b, and all the keys stored in the BST are integers.)

In addition to writing a brief description of the algorithm, you also need to implement the algorithm in Java (i.e., provide respective Java code), as well as justify the algorithm's running time.

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: 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 (extends LinkedBinaryTree), 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 variables and methods to this class, as needed.

```
public class BinarySearchTree extends LinkedBinaryTree {
```

[1] Here BT stands for Binary Tree.

```
/* 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  */
    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.cse.yorku.ca/course/2011/WisdomForRoad.txt