Tree & Trie In-class/Warmup Questions

Your name: ___________________

Other Group members present:

___________________     ___________________

___________________

Resources used:

Tree to List Recursion Problem

Thanks to Nick Parlante for this problem.

In a binary search tree, each node contains a single data element and "small" and "large" pointers to sub-trees (i.e. left and right). A binary tree may be defined as follows: Below is a binary search tree with the numbers 1 through 5.

[Diagram of a binary search tree]

Now, consider the data as a circular doubly linked list.

[Diagram of a circular doubly linked list]
The challenge is to take an ordered binary tree and rearrange the internal pointers to make a circular doubly linked list out of it. The left pointer should play the role of prev and the right pointer should play the role of next. The list should be arranged so that the nodes are in increasing order.

1. Given the following definition of the class TreeList. Fill in the blank methods.

```java
public class Node {
    int data;
    Node left;
    Node right;

    Node (int v, Node lt, Node rt) {
        data = v;
        left = lt;
        right = rt;
    }

    /*
     * helper function -- given two list nodes, join them
     * together so the second immediately follow the first.
     * Sets the .next of the first and the .prev of the second.
     */
    public static void join(Node a, Node b) {
    }
}
```

```java
/*
 * helper function -- given two circular doubly linked
 * lists, append them and return the new list.
 */
public static Node append(Node a, Node b) {
    }
}
```

2. Write a recursive method `treeToList(Node root)` that takes an ordered binary tree and rearranges the internal pointers to make a circular doubly linked list out of the tree nodes. The prev pointers should be stored in the left field and the next pointers should be stored in the right field. The list should be arranged so that the nodes are in increasing order. Return the head pointer to the new list. The operation can be done in $O(n)$ time -- essentially operating on each node once.
public static Node treeToList(Node root) {

Tries

A trie stores words and supports add/search in $O(w)$ time, where $w$ is the length of the word. The number of total words stored in the trie doesn't matter, so looking up the word “apple” requires basically 5 operations regardless of whether the trie stores 100 words or 1,000,000 words.

The code in Trie.java shows how to add a word and how to look up a word. The picture below shows a trie containing the words that follow.

do, dog, dogma, dot, dote,
ha, we, wed, wept

The black dot in each node indicates a isWord field having value true in some node; if there is no black dot the isWord field is false.
1. If the word "hopefully" is inserted into an empty trie then nine nodes are created. If after this insertion the words "hop", "hopeful", and "hope" are inserted, in that order, how many new nodes are created? Why?

2. If after inserting the words in the previous problem the words "hopes" and "hoping" are inserted, how many new nodes are created? Why?

3. Write a Trie method prefixList that stores in an ArrayList all the words in a trie for which prefix is a prefix, e.g., if prefix is tid, the list returned might contain:
   
   tidal, tide, tides, tidied, tidiness, tidy, tidying

   You can use any existing code in writing the method.

   public void prefixList(ArrayList list, String prefix)

4. The alphabet is likely to have gaps, i.e. stretches of encodings that do not correspond to any String we insert. Instead of wastefully using 128 pointers for each node, how could the Trie children pointers be compressed?