Multi-Binary Search Tree

Due 11:59 pm Mon, April 14, 2008

You may work with 1 partner of your choosing on this assignment. Both partners receive the same grade and cannot use late days.

1 Assignment

1. Complete the "Improving the BinarySearchTree" lab from *Java Structures*, Chapter 14.9, pages 367—368
2. Your classes must follow the interface described below
3. You may use any classes that you wish internally and add any private or protected helper methods or classes that you desire, so long as your data structure actually is a multi-binary tree.
4. Include detailed comments describing:
    a. Specifications, wherever the one below is ambiguous or undefined
    b. The merits of your design decisions
    c. The asymptotic behavior of your methods

Come to your lab session already having read the assignment and prepared to work on it. We will discuss the problem together in lab and then you will begin the assignment on your own.

When complete, clean up your directory and then submit your solution as a file named binary.tar

turnin –c 136 binary.tar

1.1 Interface

```
public class BinaryMultiTree<T> implements Iterable<T> {

    /** The comparator must have the property that if a.equals(b) then
        comparator.compare(a, b) == 0. */
    public BinaryMultiTree(Comparator<T> comparator);

    /** Number of values in the tree (not the number of nodes) */
    public int size();
    public void add(T x);

    /** Returns true if there exists some value y in the tree such that y.equals(x) */
    public boolean contains(T x);
     public void clear();

    /** Removes one value that is equals(x) and returns it; if
       there are no instances equals(x), then returns null. */
    public T remove(T x);

    /** Returns one value that is equals(x); if there are no
       instances equals(x), returns null. The value returned is
       the same one that remove would return.*/
    public T get(T x);

    /** An in-order iterator.  Assumes that the tree is not
        modified during iteration.*/
    public Iterator<T> iterator();
}

public class Name {
     public Name(String first, String last);
     public String getFirst();
     public String getLast();
     public boolean equals(Object obj);
     /** firstname lastname */
     public String toString();
}

public LastNameComparator implements Comparator<Name> {
    public int compare(Name a, Name b);
    public boolean equals(Object obj) { return false; }
}
```

2 Evaluation

The grading guidelines for this assignment are:

Correctness	**20**
Readability	**25**
Design	**20**
Efficiency	**20**
Total	**85**

3 Alternative

Instead of the assignment described above, you may instead do this alternative. Write a program that finds full binary trees (not necessarily binary *search* trees) of Strings containing at least 3 levels, whose nodes are not English words ("a", "I", and "it" excepted), but which spell out English words or a series of English words when traversed in-order, pre-order, post-order, and level order. For example, the tree:

Spells "C A T" in level order and pre-order, "ACT" in-order, and "ATC" (air traffic control?) in post-order. This is not a valid solution because the tree contains only two levels and ATC is kind of sketchy; but hopefully you get the idea.

The words are not required to form a meaningful sentence. Proper nouns that appear in the dictionary are allowed. Punctuation counts; if your word needs an apostrophe you must have an apostrophe in the tree. You don't have to have spaces between the words. You may have multiple nodes with the same letter or combination of letters.

Your program must periodically print status messages that give some indication of progress (e.g., "Checking combination 418122/21321833", and should print out any solutions that it encounters as "SOLUTION: " followed by the node values separated by commas in layer order.

You can access files containing the Webster's dictionary contents at /usr/share/dict on most Macintosh computers. These will be essential to the operation of your program! You do not need to actually implement any trees in your code—it is acceptable to hard-code the order of node traversal for each of the four traversal methods.

To receive full credit you must either submit your source code and:

1. one or more trees that satisfy the specification *or*
2. evidence that your code is correct and that no such tree of three levels exists *or*
3. argue that it would take more than a week of computation time to verify that no solution exists.

Making the second argument will probably involve tiny test dictionaries with fake words, just to demonstrate that you can find them in some circumstances.

Your program might need to run for a long time. Estimate the order of growth of your program, and consider whether different data structures would improve performance. Contact Mary Bailey or Morgan McGuire if you believe that your running time is as good as it can be and need to run a program overnight; we can help you execute on the Unix machines even after you've logged out.