

COP-3530 Data Structures

Programming Assignment 3: Balanced/Unbalanced Trees

Due Date: July 1 at 11:59 PM

This assignment has two parts: 1) implementing a tree map (UnbalancedTreeMap) based on the unbalanced binary search tree discussed in class, 2) and comparing the efficiency of the implemented map class with the standard `java.util.TreeMap` in a given scenario.

1 Constructing UnbalancedTreeMap

In this section, you need to first implement a class `OrderedKeyValue` to store an ordered pair of a `String` key and its `Integer` value. Then, you should implement the class `UnbalancedTreeMap` which stores objects of class `OrderedKeyValue` in the unbalanced binary search tree.

Class `OrderedKeyValue` which implements `Comparable` interface of Java must include the following members:

1. *String key;*
2. *int value;*
3. A constructor for creating a new object with key and value initialized by the input parameters.
4. *int compareTo(Object o)* that implements `Comparable` interface by comparing the keys of two `OrderedKeyValue`s based on the alphabetical order and in a case-insensitive fashion.

You can add more members to it if you would like to.

Class `UnbalancedTreeMap` must include the following members:

1. *BinaryNode root;* which is the root of your binary search tree and `BinaryNode` is a class (subclass of `UnbalancedTreeMap` preferably) with the following instance fields: *OrderedKeyValue keyValue*, *BinaryNode leftChild*, and *BinaryNode rightChild*. The binary search tree that you're supposed to construct uses the instances of `BinaryNode` as its nodes.
2. A constructor to create an empty tree map;

3. *public int get(String key)*: searches through the binary search tree to see whether any node stores an object of `OrderedKeyValue` which matches the input key. If such `OrderedKeyValue` is found, the method returns its value; otherwise, it returns 0. Obviously, your method should perform $O(\log n)$ comparisons for each search operation as you use a BST.
4. *public int put(String key, int value)*: searches through the tree to see whether any node stores an object of `OrderedKeyValue` which matches the input key. If such `OrderedKeyValue` is found, the method sets its value to the one given by the second input parameter and returns the previous value as the method output; otherwise, it inserts a new node to the tree to store the given key and value and returns 0. Again, your method should execute $O(\log n)$ instructions for each insert operation as you use a BST.
5. *public String[] keySet()*: By visiting the nodes of BST using in-order traversal, you should store the keys in an array of strings and return the array at the end. The returned array contains all the keys in alphabetical order.

You can add more members to it if you would like to.

2 Efficiency Analysis of the Implemented TreeMap

In this part, you need to write a program that compares the efficiency of `UnbalancedTreeMap` in part 1 with the standard `java.util.TreeMap<String, Integer>`. To do so, you need to find and compare the running times of the following scenario for both tree maps:

Test Scenario: Calculating the Frequency of Words appeared in H. C. Andersen Fairy Tales

1. `long startTime = System.nanoTime();`
2. `long totalTime = 0;`
3. `for(int i = 1; i<=77;i++)`
 - (a) Create a new Scanner (`new File(". . . your local address . . . \"+i+ ".okpuncs")`) (Files are available on Canvas).
 - (b) Read the i^{th} file and tokenize its sentences into words. For every observed word in the document, do the following:
 - i. `startTime = System.nanoTime();`
 - ii. update the frequency of every word in the tree map using `put` and `get` methods: `put(word, 1+get(word));`
 - iii. `totalTime+= (System.nanoTime()-startTime);`
4. `System.out.println(n + "," + totalTime);`

3 Submissions

You need to submit a *.zip* file compressing a folder containing the following files:

- all the Java source file(s).
- A readme.txt file containing the running times of the test scenario when Unbalance-TreeMap and java.util.TreeMap are used and short paragraph comparing and justifying the efficiency of your treemap with the standard one.