Programming Assignment 4:
Paint it Black

Due date: November 16 by 11:59 PM.

Overview

In this lab, you will implement the Map ADT as a red-black tree, supporting only the add and find operations.

Tree Design

You will implement a RedBlackTreeMap class using notes from lecture and features appropriate to the language of your choice. General implementation details:

1. Implement the *add* and *find* ADT operations. You will not implement *remove*.
   (a) Add: given a key and value, insert a new Node into the tree and maintain balance as necessary. If the key is already present, set its value to the parameter and do not modify anything else.
   (b) Find: given a key, return the value for the key. If the key does not exist in the tree, throw an exception/error.
2. Implement these auxiliary methods:
   (a) containsKey: given a key, return true if the key is in the tree, and false otherwise.
   (b) count: return the number of data elements in the tree.
   (c) printStructure: prints (with `System.out` / `cout`) a **pre-order traversal** of the tree's nodes, one per line, printing three things: the node's key, its value, and an indication of whether the node is **red** or **black**. For example, if the root of the tree is key "Neal Terrell" with value 500 in a black node, the first print from this method would be `Neal Terrell :  500 (black)`.
3. You will need a "helper" class to implement the tee:
   (a) A "node" class containing a key and value; plus links to the parent, left, and right child of the node; and a boolean for the color of the node.
   (b) You may want a static Node to represent "`NIL`", so that all "nil" links can instead point to a single real Node object that is colored black.
4. You will also need helper methods to handle the special insertion cases:
   (a) getGrandparent and getUncle, to find special relatives of nodes
   (b) singleRotateRight and singleRotateLeft, to rotate the tree in certain insertion cases

I have partially implemented some of these methods for you in a Java or C++ class I have put on BeachBoard. You are welcome to use that class as a starting point, or you may implement the tree in your own way as long as you following the requirements above. The provided implementation will handle the "normal BST insertion" rules without accounting for any of the special red-black insertion cases.

Program

I don't want a "real" program to demonstrate this structure. I just want you to thoroughly test your code to make sure it is correct. To that end, you **must** perform **and turn in** the following tests on your tree code:

1. First, download the file `players_homeruns.csv` from BeachBoard's Lab content. Construct a tree map to map from string (player name) to int (home run total).
2. Read the **first five lines of the file** and insert the corresponding players into the map.

3. Print the structure of the tree with the `printStructure` method.
4. **By hand, insert the first five names of the .csv file into a red/black tree. Draw the tree on a piece of paper, and turn in that paper.** Verify that the pre-order traversal of your hand-drawn tree **exactly matches** the output of `printStructure`.
5. Add the next five lines of the file and insert them into the tree. **Draw the resulting red/black tree by hand**. Verify that `printStructure` gives the correct structure.
6. Once steps 1-5 are correct, call the `find` method on **four** different keys in the tree. Verify that the results are correct. The keys should be:
   (a) one key that is a leaf (has two NIL children)
   (b) one key that is a root
   (c) one key that has one NIL child and one non-NIL child.
   (d) one key that is in a red node.
7. Now add all remaining lines of the file to the tree. **Do not attempt to draw the resulting red/black tree by hand.** Instread, call `find` on the same four keys as in step 6, and verify that the answers are still correct.

Deliverables

Turn in:

1. A printed copy of your program's code.
2. Your handwritten trees for steps 4 and 5 in "Program".
3. The results of calling `find` on the four keys you chose in step 6 of "Program".