CS146 Data Structures and Algorithms
Instructor: K. Potika SJSU CS Department
Programming Project 4

Important: Do individually (each student alone is NOT a group assignment), each student has to turn in one project.

Red Black Tree: Dictionary and Spell Checker.

In this project you will create a Dictionary, aka a Red Black Tree. Using the provided link to a dictionary, insert all the words in a Red Black Tree. Create a new file with a poem and use the dictionary as a spell checker for your poem. A spell checker just calls lookups to the dictionary to see if a word exists. Count the time to create the dictionary and the time for spell checking. Note: To count the time use system.currentTimeMillis() or a similar method.

You can go for the generic implementation of a Red Black Tree [5pts extra] or just for Strings.

In a Red Black tree the longest path from the root to a leaf cannot be more than twice of the shortest path from the root to a leaf. This means that the tree is always balanced and the operations are always O(lgn).

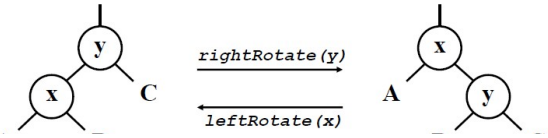
Since a Red Black Tree is a binary search tree, the following property must be true: the value in every node is larger than the value of the left child (or any value in the left subtree) and smaller than the value of the right child (or any value in the right subtree). Additionally, a Red Black Tree has these properties:

1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-black trees consist of nodes, which are instances of the class that is given in RBNode. Create a Red Black Tree (RBtree) class. First you will have to give the RBtree all the functionality of a binary search tree. Don't use Collections. Use your own code. Use the provided template.

Here are some of the instances/methods:

- an instance variable that points to the root RBNode.
- lookup(String): searches for a key.
- printTree(): Start at the root node and traverse the tree using preorder
- addNode(String): place a new node in the binary search tree with data the parameter and color it red.
- getSibling(RBNode): returns the sibling node of the parameter If the sibling does not exist, then return null.
- getAunt(RBNode): returns the aunt of the parameter or the sibling of the parent node. If the aunt node does not exist, then return null.
- getGrandparent(RBNode): Similar to getAunt() and getSibling(), returns the parent of your parent node, if it doesn't exist return null.
- rotateLeft(RBNode) and rotateRight(RBNode) functions: left, resp. right, rotate around the node parameter. See next figure that demonstrates the two rotations:



- fixTree(RBNode current) recursive function: recursively traverse the tree to make it a Red Black tree. Here is a description of all the cases for the current pointer:
  1) current is the root node. Make it black and quit.
  2) Parent is black. Quit, the tree is a Red Black Tree.

  BASE CASES?

  3) The current node is red and the parent node is red. The tree is unbalanced and you will have to modify it in the following way.

  I. If the aunt node is empty or black, then there are four sub cases that you have to process.

  A) grandparent –parent(is left child)— current (is right child) case. Solution: rotate the parent left and then continue recursively fixing the tree starting with the original parent.
  B) grandparent –parent (is right child)— current (is left child) case. Solution: rotate the parent right and then continue recursively fixing the tree starting with the original parent.
  C) grandparent –parent (is left child)— current (is left child) case. Solution: make the parent black, make the grandparent red, rotate the grandparent to the right and quit, tree is balanced.
  D) grandparent –parent (is right child)— current (is right child) case. Solution: make the parent black, make the grandparent red, rotate the grandparent to the left, quit tree is balanced.

  II. Else if the aunt is red, then make the parent black, make the aunt black, make the grandparent red and continue recursively fix up the tree starting with the grandparent.

Deliverables:

- Include a Tester program that creates a Dictionary using a RBtree. You read the words from a text file and insert the words in the RBtree. Use this Dictionary to lookup words of a document. Time the creation of the Dictionary and the loopkup calls. Use compareTo() for Strings.
- If you choose to do the generic implementation to avoid problems separate into different files: one interface file the Visitor, one class file the Node, one class file the Red Black Trees.
- Create a report with your findings and challenges you faced.
- Your main grade will be based on (a) how well your tests cover your own code, (b) how well your code does on your tests (create for all non-trivial methods), and (c) how well your code does on my tests (which you have to add to your test file). For JUnit tests check canvas.
- Use cs146S19.<lastname>.project4 as your package, and Test classes should be your main java file, along with your JUnit java tests.
- Export your project (LastNameFirstName.zip). Upload it with your report to canvas.
- All projects need to compile. If your program does not compile you will receive 0 points on this project.
- Do not use any fancy libraries. We should be able to compile it under standard installs. Include a readme file on how to you compile the project.