Jake Goldstein

HW 09 Written

1. Pseudo-Code
   1. Contains
      1. If the element that im checking is null, return false
      2. If the element that im checking is what I am looking for return true
      3. If the element im looking for is smaller than what im looking at, call contains on the left side
      4. If the element im looking for is larger than what im looking at, call contains on the right side
   2. rangedPrint
      1. If the element that im checking is null, return
      2. If the element im checking is larger than the smallest element, move to the left subtree
      3. If the element that im checking is within the range that I want to print, print that element
      4. If the element im checking is smaller than the largest element, move to the right subtree
   3. Stringy
      1. Find the min of the tree
      2. Recrusive call left side
      3. Find the largest node in the left subtree
      4. Have the largest node in left subtree point to myRoot
      5. Find the smallest node in the right subtree
      6. Recrusive call on right subtree
      7. Have myRoot point to the smallest Node in the right subtree
      8. Have the root of the tree become the smallest element in the tree
   4. Average Node Depth
      1. Find the depth of the left and right subtrees
         1. If the element is null, return
         2. Else: Add one to the total size of the tree and current depth
         3. Find the total depth of the right and left subtrees
         4. Pop out a level and subtract from the depth
      2. Divide the total depth by the total size of the tree
2. O(n)
3. This code should work correctly.
4. This code should work correctly.
5. Red-Black tree

(Insert 50):

22

17 30

14 18 50

(Insert 44):

22

17 30 Case 2

14 18 50

44

22

17 30 Case 3

14 18 44

50

22

17 44

14 18 30 50

1. Do not turn in…
2. Right rotate:

template <class Comparable>

void RedBlackTree<Comparable>::rightRotateRecolor( Node \* & k2 ){

auto k1 = k2->left;

k2->left = k1->right;

k1->right = k2;

k2 = k1;

}

1. A case two is when you make the greatest number of changes to pointers (case one your only changing the color of the tree, not the pointers). In a case two, first you add an element which I will count as one change. Then in a left-rotate, there are another two pointer changes. Then in a right-rotate there are another two pointer changes, for a total of 5 pointer changes.