Assignment 9

R-3.11 Consider the following sequence of keys: (5, 16, 22, 45, 2, 10, 18, 30, 50, 12, 1) Consider the insertion of items with this set of keys, in the order given, into: a. an initially empty (2,4) tree T’. b. an initially empty red-black tree T’’. Draw T’ and T’’ after each insertion.

A close up of text on a whiteboard

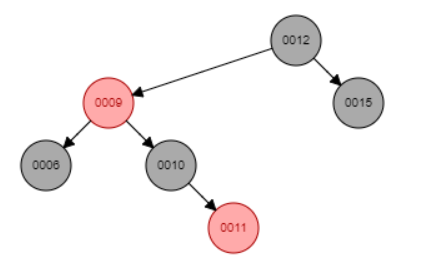
Description automatically generatedA picture containing text, whiteboard

Description automatically generated

R-3.14 For each of the following statements about red-black trees, determine whether it is true or false. If you think it is true, provide a justification. If you think it is false, give a counterexample.

a. a subtree of a red-black tree is itself a red-black tree.

False. For example:



The subtree 0009 is not a red-black tree when the node 0009 is red

12, 9, 6, 10, 11, 15

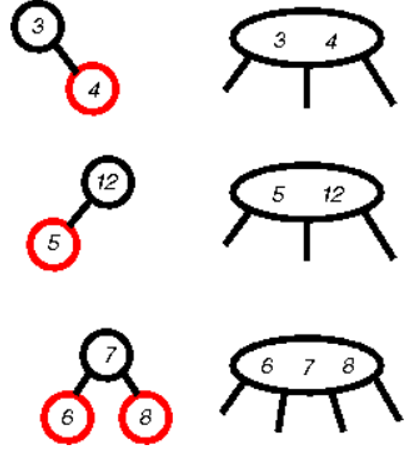
Findindex(9, 11) -> [1, 2, 3]

b. the sibling of an external node is either external or it is red.

False. As the example above, 0010 is sibling node of an external node of the node 0015, but the node 0010 is black.

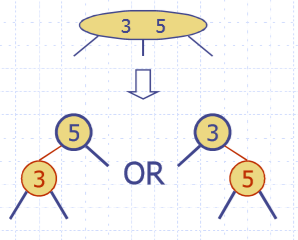
c. given a red-black tree T, there is an unique (2,4) tree T’ associated with T.

True because a red node always belongs to a black parent node



d. given a (2,4) tree T, there is an unique red-black tree T’ associated with T.

False. For example: the node (3,5) as below can perform to 2 types of red-black tree



Design a pseudo code algorithm isValidAVL(T) that decides whether or not a binary tree is a valid AVL tree. For this problem, we define valid to mean that the height of the left and right sub-trees of every node do not differ by more than one.

What is the time complexity of your algorithm?

|  |  |
| --- | --- |
| Algorithm isAVLTree(T)  Input: T is BST  Output: true of the tree T is AVL tree  if T.isEmpty() = true then  return false    \_isAVLSubTree(T, T.root())    \_isAVLSubTree(T, v)  l <- T.leftChild(v)  r <- T.rightChild(v)    if T.isExternal(l) = true /\ T.isExternal() = true then  return true  if T.isExternal(l) then  return \_isNotParent(r)  else  if T.isExternal(r) then  return \_isNotParent(l)    return \_isAVLSubTree(l) /\ \_isAVLSubTree(r)    \_isNotParent(T, v)  l <- T.leftChild()  r <- T.rightChild()  return (T.isExternal(l) /\ T.isExternal(r)) | Total running time: O(2n)  O(1)  O(1)  O(2n)  Running time: O(2n)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(2n)  Running time: O(1)  O(1)  O(1)  O(1) |

Design an algorithm, isPermutation(A,B) that takes two sequences A and B and determines whether or not they are permutations of each other, i.e., they contain same elements but possibly occurring in a different order. Assume the elements in A and B cannot be sorted. Hint: A and B may contain duplicates. Same problem as in previous homework, but this time use a dictionary to solve the problem.

|  |  |
| --- | --- |
| Algorithm isPermutation(A, B)  D <- new Dictionary(HT)  for each a in A.elements() do  D.insertElement(a, a)    for each b in B.elements() do  p <- D.findElement(b)  if p = NO\_SUCH\_KEY then  return false  else  D.removeElement(b)    return true | O(1)  O(n)  O(n)  O(n)  O(n) because using HT, O(nlogn) if using BST  O(n)  O(1)  O(1)  O(1)  Total running time: O(n) |

What is the worst case time complexity of your algorithm? Justify your answer.

* Calculate theta?

C-3.10 Let D be an ordered dictionary with n items implemented by means of an AVL tree (or a Red-Black tree). Show how to implement the following operation on D in time O(log n + s), where s is the size of the iterator returned:

FindAllInRange(k1, k2)

Return an iterator of all the elements in D with key k such that k1 < k < k2.

|  |  |
| --- | --- |
| Algorithm findAllInRange(D, k1, k2)  Iterator iter <- new Iterator  if D.isEmpty() = true then  return iter    Dr <- new Dictionary(BST)    Iterator dIter <- D.keys()    while dIter.hasNext() do  p <- dIter.nextObject()  if p.key() > k1 /\ p.key() < k2 then  Dr.insertItem(p.key(), p.element())  else  if p.key() >= k2 then  break    rIter <- Dr.keys()  return iter | O(1)  O(1)  O(1)  O(1)  O(n) |