Assignment 9

R-3.11 Consider the following sequence of keys:

(5, 16, 22, 45, 2, 10, 18, 30, 50, 12, 13, 33)

Draw *T’* and *T’’* after each insertion.

At each step, you can check your diagram using the following simulators:

http://cs.armstrong.edu/liang/animation/web/24Tree.html https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Consider the insertion of items with this set of keys, in the order given, into:

1. an initially empty (2,4) tree *T’*.

A picture containing circle, diagram, line, sketch

Description automatically generated

b. an initially empty red-black tree *T’’.*

A picture containing circle

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

Answer: **False**

For example, the R-B tree T’’ of R-3.11 contains a subtree that violates **Root Property** that says **the root is black** such as node 45 is Red as below:

A picture containing circle

Description automatically generated

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

Answer: **False** when the parent node is red

For example, the R-B tree T’’ of R-3.11 follows **Internal Property** that **says the children of a red node are black** as below:

A picture containing circle

Description automatically generated

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

Answer:

A picture containing circle

Description automatically generated

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

Answer:

A picture containing diagram, line, sketch, white

Description automatically generated

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?

Answer:

Algorithm **isValidAVL**(T)

(h, B) 🡨 **validHelper**(T, T.root())

return B

Algorithm **validHelper**(T, p)

if T.isExternal(p) then

return (0, T)

(lh, LB) 🡨 **validHelper** (T, T.leftChild(p))

(rh, RB) 🡨 **validHelper** (T, T.rightChild(p))

return (MAX(lh, rh) + 1, LB ^ RB ^ ABS(lh – rh) < 2)

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.

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

Answer: The worst case when both A and B do not contain duplicates and A is permutation B

Algorithm **isPernutation**(A, B)

if A.isEmpty() or B.isEmpty() then

return F

if A.size() != B.size()

return F

D 🡨 newDictionary(HT)

**countElements**(A, D)

return **reduceElements**(B, D)

Algorithm **countElements**(S, D)

p 🡨 S.first()

D.insertItem(p.element(), 1)

while !S.isLast(p) do

p 🡨 S.after(p)

k 🡨 p.element()

cnt 🡨 D.findElement(k)

if cnt = NO\_SUCH\_KEY then

cnt 🡨 0

cnt = cnt + 1

D.insertItem(k, cnt)

Algorithm **reduceElements**(S, D)

p 🡨 S.first()

k 🡨 p.element()

cnt 🡨 D.findElement(k)

if cnt = NO\_SUCH\_KEY or cnt = 0 then

return F

cnt = cnt -1

D.replaceElement(k, cnt)

while !S.isLast(p) do

p 🡨 S.after(p)

k 🡨 p.element()

cnt 🡨 D.findElement(k)

if cnt = NO\_SUCH\_KEY or cnt = 0 then

return F

cnt = cnt -1

D.replaceElement(k, cnt)  
 return T

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.

Answer:

Algorithm **allInRange**(k1, k2)

S 🡨 new Sequence

**findHelper**(T, k1, k2, T.root(), S)

return S

Algorithm **findHelper**(T, k1, k2, p, S)

if T.isExternal(p) then

return

e 🡨 p.element()

if k2 <= e then

**findHelper**(T, k1, k2, T.leftChild(p), S)

if k1 <= e and e <= k2 then

S.insertLast(e)

if e <= k1 then

**findHelper**(T, k1, k2, T.rightChild(p), S)