Homework Assignment #3

Thu, Feb 13, 2020 5:30 PM

Written by *Jiahong Zhai* 1005877561

Written by *Jiabao Shen* 1004297548

Question 2

a.

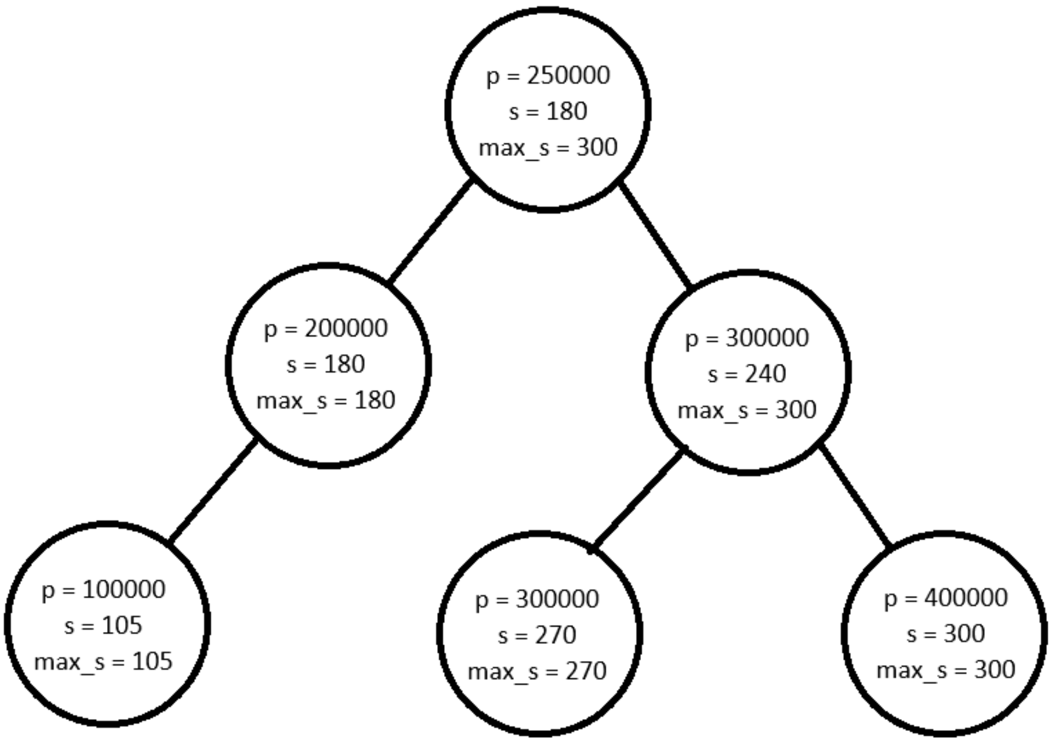
We can use augmented AVL tree as data structure. Each node represents a listing. Each node's key is bigger or equal to the key of its left sub-child, and smaller than the key of its right sub-child.

A node contains 3 information about listing

1. p = Price(key)
2. s = Floor Area
3. max_s = Biggest s of all nodes in the sub-tree rooted by this node

For leaves, max_s = s.

Example:



The height and the balance factor are not shown in the graph.

b.

Insert (D; x):

Step 1: Insert the node from the root of the tree, just like the usual BST tree insertion.

Step 2: Recursively update max_s for all nodes of x's predecessor. From bottom way up to the root (in some case we can stop at one nodes). When updating a node, compare it's max_s with its children's (the children which is x's predecessor or x itself). If the node's max_s is smaller than its children's max_s then update the max_s to be its children's, and do the same operation to its parent node. If the node's max_s is not smaller than its children's max_s, then we can stop here and move to step 3.

Step 3: Rebalance the AVL tree. For all nodes of x's predecessor from bottom way up to the root (in some case we can stop at one nodes). Also update the max_s value for all nodes whose children are changed.

Step 1, 2, 3 are recursive function, in the worst case, the method will operate once at each level of the tree, each single operation takes constant steps. In AVL tree the level of the tree is O(log n+log n+log n) = O(logn). So, worst case running time of Insert (D; x) is O(log n).

Delete (D; x):

Step 1: Delete. Delete the node that contains x in D, just like the usual BST tree deletion.

Step 2: Update the max_s. Find the starting node of this operation. If x is a leaf, then the starting node is x's predecessor. If x has one child, then the starting node is that child after it take the place of x in D. If x has two children, then the starting node is x's right sub-tree's left-most node's predecessor (before deletion). Recursively update max_s for starting node's predecessors and starting node from bottom way up to the root (in some case we can stop at one nodes). In each update, max_s = Max{s, left.max_s, right.max_s}. If one node's max_s is not changed after the update, then we stop here and move to step 3.

Step 3: Rebalance the AVL tree. For all nodes of x's predecessor from bottom way up to the root. Also update the max_s value for all nodes in the path of rebalance.

In worst case, step1, 2 ,3 recursively operate the number of levels of the tree times, each time it takes constant steps. In AVL tree the number of level is O(log n+log n+log n) = O(logn). WC running time of Delete (D; x) is O(log n).

MaxArea (D; p):

Traverse the tree from up to down starts from root

pseudo-code:

```
Let CurrMax = -1
Let Curr = root
    While(Curr is not null):
        If Curr.p <= p
            CurrMax = Max(CurrMax, Curr.s, Curr.left.max_s)
            Curr = Curr.right
        Else Curr = Curr.left
Return CurrMax
```

In the worst case, this program will repeat number of the level of D times, each time it takes constant steps. The number of level of AVL tree is O(log n). So, WC running time is O(log n).