Homework 3, Group Problem 7

Graded by: Group 1 (Ya Chen, Saku Kukkonen, Prabir Yadav)
Adopted from a solution by: Group 1, Group 3, Group 6

TREE-INSERT(T,z)

```
1  y ← NIL
2  x ← root[T]
3  while x ≠ NIL
4        do y ← x
5        if key[z] < key[x]
6           then x ← left[x]
7            else x ← right[x]
8  p[z] ← y
9  if y = NIL
10    then root[T] ← z
11     else if key[z] < key[y]
12             then left[y] ← z
13              else right[y] ← z
```

Problem Statement

Binary search trees with equal keys

Equal keys pose a problem for the implementation of binary search trees.

a. What is the asymtotic performance of TREE-INSERT when used to insert n items with identical keys into an initially empty binary search tree?

We propose to improve TREE-INSERT by testing before line 5 whether or not $key[z] = key[x]$ and by testing before line 11 whether or not $key[z] = key[y]$. If equality holds, we implement one of the following strategies. For each strategy, find the asymptotic performance of inserting n iteMps with identical keys into an initially empty binary search tree. (The strategies are described for line 5, in which we compare the keys of z and x. Substitute y for x to arrive at the strategies for line 11.)

b. Keep a boolean flag $b[x]$ at node x, and set x to either $left[x]$ or $right[x]$ based on the value of $b[x]$, which alternated between FALSE and TRUE each time the node is visited during TREE-INSERT.

c. Keep a list of nodes with equal keys at x, and insert z into the list.

d. Randomly set x to either $left[x]$ or $right[x]$. (Give the worst-case performance and informally derive the average-case performance.)

Solutions

Solution for a.

When inserting items with identical keys then boolean clause at the line 5 of TREE-INSERT is always FALSE and so the right child will always be chosen. Because

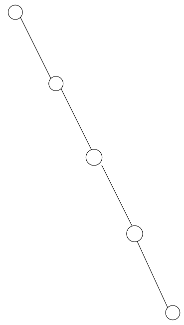

Figure 1: Tree using original TREE-INSERT and items with identical keys.

boolean clause at line 11 is also FALSE new node will be inserted as a right child of the rightmost node. After inserting n nodes to the tree, the height of the tree will be n and no node will have a left child (Figure 1). Inserting n items into an initially empty binary search tree will cost

$$T(n) = \sum_{i=1}^{n} i = \frac{1}{2}n(n+1) = O(n^2),$$

because the height of tree increases at every insertion and new element is inserted as a new leaf node. This is asymtotic performance of inserting n items initially empty binary search tree.

If a new element is inserted to this tree, cost is equal to the height of the tree i.e. $O(n)$ since height equal to n.

Solution for b.

Building binary search tree like this using identical keys will form a perfect binary tree with height $O(\lg n)$. This can be seen when we start to insert elements into an initially empty binary search tree. Lets say that boolean flag value 0 set x to $left[x]$ and value 1 set x to $right[x]$. Boolean flag of a node will be changed after visiting that node. Now, the first element will go as a left child of root node, second element will go as a right child of root node (boolean flag changes its value after first visit), third element will go as a left child of left child of root node, fourth element will go as a left child of right child of root node, etc.

Inserting seven items into an initially empty binary search tree is shown in figure 2. One should note that when one level in this tree is full, all the flags have value FALSE.

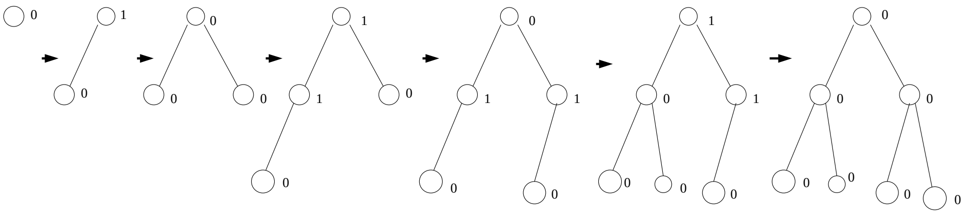

Figure 2: Tree using boolean flags in nodes and items with identical keys.

Alternating boolean flag value every time we visit a node makes sure that all paths in binary tree will be travelled before travelling an old path again and one level will be filled before moving to the next level. This can be proved by method of induction as follows:

Inductive Hypothesis: At any level n, a node at the level n with its flag equal to FALSE will have the same number of nodes in its left and right subtree, whereas if its flag equal to TRUE then it will have a left subtree with one more node than its right subtree.

Base Case, $n = 1$: We insert an element to an empty tree. After this the tree will contain a root node and its flag is set to FALSE. The root node has no subtrees and so the left subtree has 0 more nodes than the right subtree. Therefore the inductive hypothesis is true for $n = 1$.

Inductive Step: Assume that the inductive hypothesis holds for a level $n = k$. We have two possible cases:

a. If a node has its flag equal to FALSE, then both of its subtrees have the same number of nodes. We insert a new element to the left subtree of this node, and we set the node's flag value to TRUE. Now the left subtree has one more node than the right subtree and the inductive hypothesis is true.
b. If the node's flag were TRUE instead, then, by inductive hypothesis, its left subtree contains one more node than its right subtree. By definition, when we insert a new node, we insert it to the right subtree. Now, both subtrees have the same number of nodes again and the flag is set to FALSE again. Therefore the inductive hypothesis is true.

The inductive hypothesis holds for $n = k + 1$ and therefore it holds also any other level n.

It follows from the above proof that root node has left and right subtrees whose height can differ by at most one level. This means that each level of tree is filled up before moving to the next level.

Asymptotic performance of inserting n items into an initially empty binary search tree is same as with ideal binary search tree and it is

$$\begin{aligned}
T(n) &= 1+2+2+3+3+3+3+\cdots+(\lfloor \lg n \rfloor + 1) \\
&= \sum_{i=1}^{n} (\lfloor \lg i \rfloor + 1) \\
&\leq \sum_{i=1}^{n} (\lg i) + n \\
&= \lg \left(\prod_{i=1}^{n} i \right) + n \\
&= \lg(n!) + n \\
&\leq \lg(n^n) + n \\
&= n \lg(n) + n \\
&= O(n \lg n)
\end{aligned}$$

Cost for inserting one item is same as with ideal binary search tree i.e. $O(\lg n)$.

Solution for c.

When inserting items with identical keys, tree will have only the root node and rest of the items are inserted to a list of equal keys. It is of course reasonable to insert elements always at the begining of the list when each item inserted takes constant time. Inserting n identical items into an initially empty binary search tree will take n steps. Asymtotic performance of inserting n items with identical keys into an initially empty empty binary search tree is then $O(n)$ and cost for inserting one item is $O(1)$.

Solution for d.

Worst-case is when the algorithm sets the new node x to either the left child ($left[x]$) or the right child ($right[x]$) of the leaf node (probability for this is $1/2^n$) forming a linearly linked-list structure. Then the tree will contain only the left child or the right child at every level (i.e. every node has only one child) as shown in the figure 1.

In the average-case, x is set randomly to either $left[x]$ or $right[x]$. This corresponds to the case where we use items with random keys to build binary search tree (chapter 13.4 in CLR). In a random permutation of n distinct keys the next element to be inserted in the tree has an equal random chance of at any one of the child positions. Similarly, in a distribution of n identical keys, the next element to be inserted in the tree has an equal random chance of being placed at any one of the child positions. That is to say that, at every node there is 1 in 2 chance that the next key will be placed either to the left or to the right.

Average-case performance for inserting n items with identical keys into an initially empty binary search tree is then $O(n \lg n)$ and cost for inserting one item is $O(\lg n)$.

GradingPolicy

Points	
0–2	7 a.
0–3	7 b. (a formal proof was required)
0–2	7 c.
0–3	7 d.