CS2010 Semester 1 2012/2013

Data Structures and Algorithms II

Tutorial 02 - Binary Search Trees

For Week 04 (03 September - 07 September 2012)

Released: Wednesday, August 29, 2012

Document is last modified on: September 5, 2012

1 Introduction and Objective

The purpose of this tutorial is to reinforce the concepts of Binary Search Tree (BST) and the importance of having balanced BST. We will also discuss Subtask 1 of PS1 during this tutorial.

2 Tutorial 02 Questions

Operations on BSTs

Q1. Find the successor of 33 in the BST (and show how to find it)

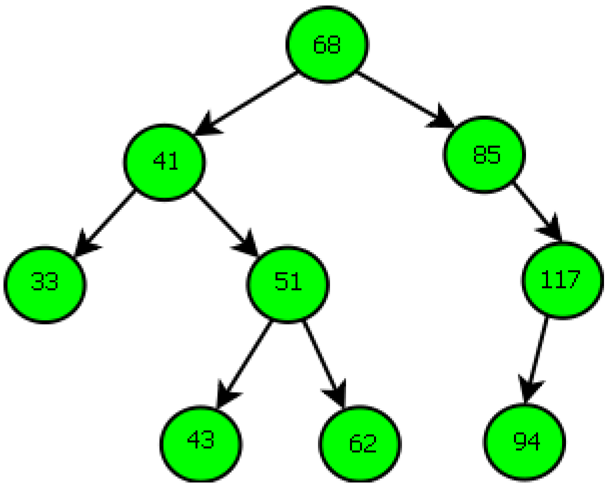

Figure 1: BST for Q1 and Q2

1. 68
2. 43
3. 85
4. 41
5. 62

Ans: **4. 41; as 33 has no right subtree, we move up from 33 to its parent(s) until we hit a right turn, 33 to 41 is a right turn so we report 41.**

Q2. Find the predecessor of 94 in the same BST in Figure 1 (and show how to find it)

1. 17
2. 41
3. 68
4. 33
5. 85

Ans: **5. 85; as 94 has no LEFT subtree, we move up from 94 to its parent(s) until we hit a left turn, 94 to 117 is a right turn, 117 to 85 is a left turn, so we report 85. Try doing this succ/pred exercise on other numbers to see if you have mastered this concept.**

Q3. What sequence does a preorder traversal of the BST in Figure 3 yield?

Preorder traversal is very similar to Inorder traversal that we have seen earlier in Lecture 2. Preorder traversal is just like this:

```
PreOrder(T)
  if T is null, stop
  Visit/Process T (see, the visitation is done first)
  PreOrder(T.left)
  PreOrder(T.right)
```

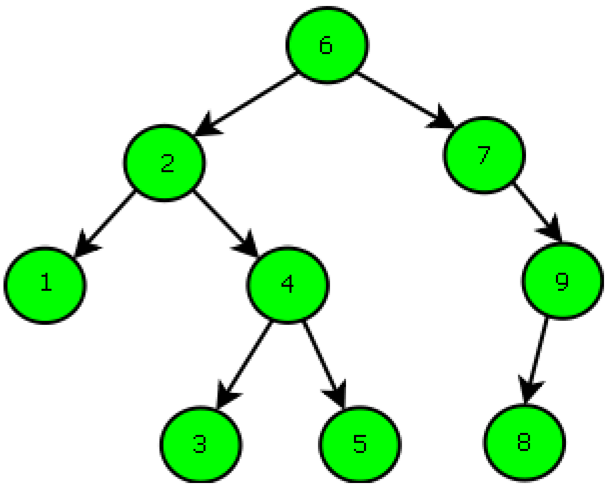

Figure 2: BST for Q3

1. 1, 2, 3, 4, 5, 6, 7, 8, 9
2. 9, 8, 7, 6, 5, 4, 3, 2, 1
3. 1, 3, 5, 4, 2, 8, 9, 7, 6
4. 6, 2, 1, 4, 3, 5, 7, 9, 8

Ans: **4. 6, 2, 1, 4, 3, 5, 7, 9, 8**.
Challenge, what about the **PostOrder** traversal of this tree?

Q4. Can you re-write the successor algorithm without using the pointer comparison?
(i.e without the use of `(cur == par.right)`)
Note: You can do the same for predecessor algorithm.

Ans: Instead of comparing the pointer, we can compare the value of the key in the parent. This is because a "right turn" occurs when we hit the first ancestor node with key > key of T (node we want to find the successor of).

Algorithm 1 int succssor(BSTNode T)

```
if (T.right != null) then                          ▷ case where T has a right subtree
    return findMin(T.right)
else
    par = T.parent
    cur = T
    while ((par != null) & (cur.key > par.key)) do
        par = par.parent
    end while
    if (par == null) then
        return -1                                  ▷ The successor of T is not found
    else
        return par.key
    end if
end if
```

Q5. Is it possible to enforce that a BST is always a full binary tree?
Full binary tree is defined as a tree where each node other then the leaf has exactly two children.

Ans: **No.**

Since a full binary tree (a.k.a a proper binary tree) always has an odd number of nodes in the tree. However in a series of insertions, we will alternate between even and odd number of nodes. For the case when we have even number of nodes there is no way to maintain the property of a full binary tree.

A simple (but maybe overkill) proof that a full binary tree has odd number of nodes:

1. Place two tokens on each internal node.
2. For each internal node (in any order), push one token to left children and another token to the right children.
3. Notice that now all nodes but the root have one token; the root has none.
4. Hence (#nodes) = (#internal) + (#leaves) = 2*(#internal) + 1 = an odd number.

AVL Trees

Q6. Is the following a valid AVL tree?

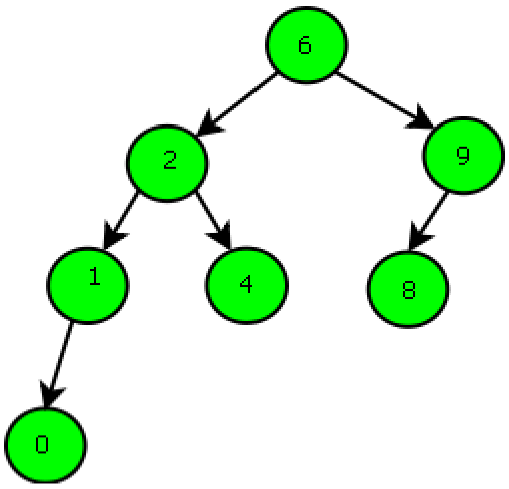

Figure 3: AVL Tree for Q6

Ans: **Yes**.

For every node, the difference in height between it's left and right subtree is ≤ 1.

Q7. Show the sequence of rotations required to re-balance the following AVL tree if 77 is inserted. (Indicate the balancing factor at each step).

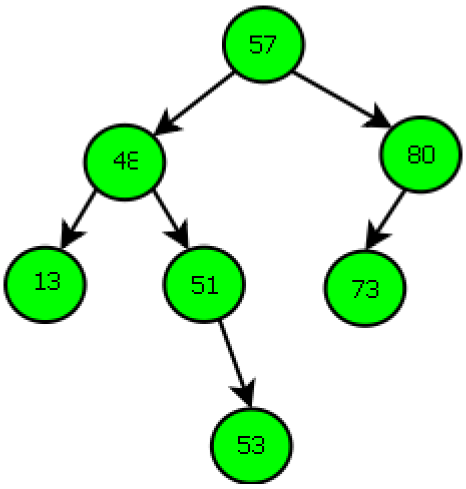

Figure 4: AVL Tree for Q7

Ans:

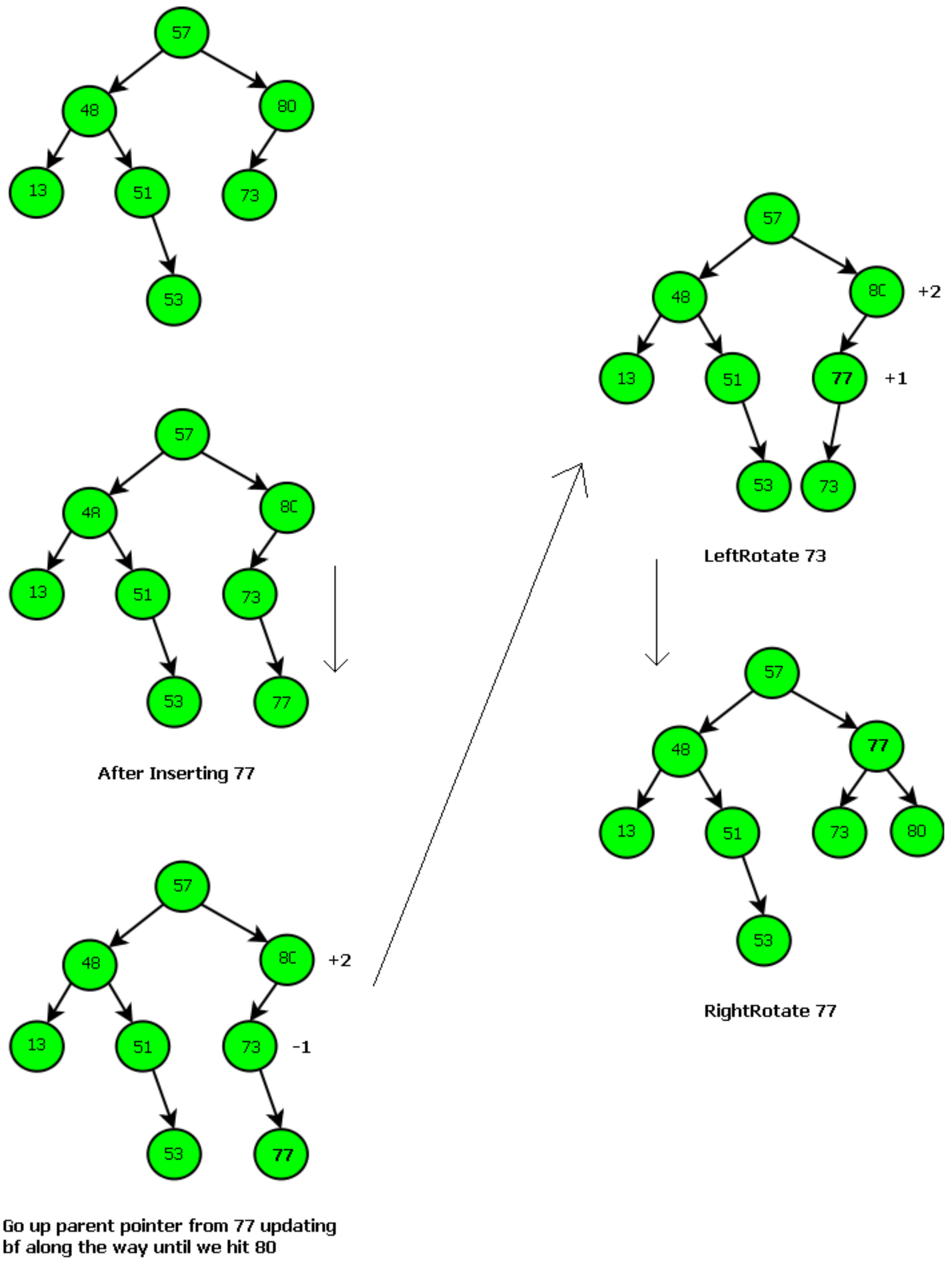
57
48
80
13
51
73
53
After Inserting 77
77
Go up parent pointer from 77 updating
bf along the way until we hit 80
+2
-1
LeftRotate 73
+1
RightRotate 77

Figure 5: Sequence of Rotations

Problem Set 1

Q8. Discussion of PS1 subtask 1.

Ans: **Solution 1. Just using Array**
Store records containing name and gender into an array of that record type. For each query, go through the entire array and check if the first letter of name start with START. If yes, check if the gender matches.

Storing record = O(N).
Each Query = O(N) at most
Time complexity = O($N + QN$)

Doable with $Q \leq 10$ and $N \leq 26$

Solution 2. Using Direct Addressing Table – optional
We use an array `A` that initially contains all zeroes. Use the first letter of Baby Name to know which index that has to be changed to one. For example, "Aaron" will set `A[0]` ('A'-'A') to be one, "Cindy" will set `A[1]` ('C'-'A') to be one, etc. We can then do a prefix sum (DP) strategy where `A[i] = A[i - 1] + A[i]` $\forall$ `i` $\in$ `[1..25]`. Then to check how many baby names are there within a range, we can just do an $O(1)$ lookup. This solution is overkill actually as the input constraint is too small.

Storing record = O(N)
Each Query = O(1)
Time complexity = O($N + Q$)

Doable with $Q \leq 10$ and $N \leq 26$

Solution 3 (can pass Subtask 3)
Store in 2 array, one for gender = male, another for gender = female. sort both array by alphabetical order of Name. Query by performing two binary searches (to get first index of START and last index before END) of either array depending on whether gender provided is male of female.

Store record = O(N).
Sort records = O($NM \log N$) //using merge sort or any other fast sorting method but to compare string, we need another $O(M)$
Each Query = 2*$M \log N$ = O($M \log N$). // 2 binary searches
Time Complexity = O($N + NM \log N + QM \log N$) = O(($N + Q$)$M \log N$)

But to solve Subtask 4, Steven put a requirement to use balanced BST. So this Solution 3 is not a valid solution for Subtask 4 although it may works just fine as Steven does not interleave 'AddSuggestion' with 'Query' (do you understand why this matters?). You will learn a trick for this in Lab Demo of this week (if using Java API), or you may end up using hours of coding/debugging time if you want to use your own balanced BST (e.g. AVL tree).