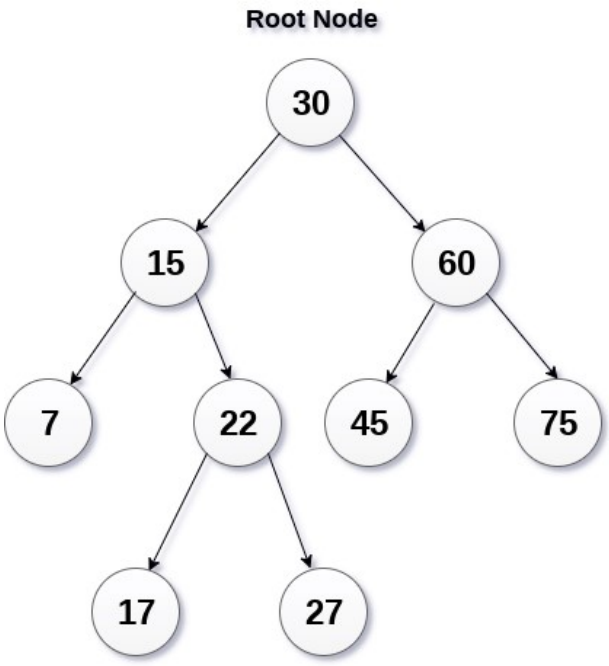
Discrete Mathematics

Scribed Notes 17 (22nd October)

Binary Search Tree

1. Binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order. This is also called ordered binary tree.
2. In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root.
3. Similarly, value of all the nodes in the right sub-tree is greater than or equal to the value of the root.
4. This rule will be recursively applied to all the left and right sub-trees of the root.



Binary Search Tree

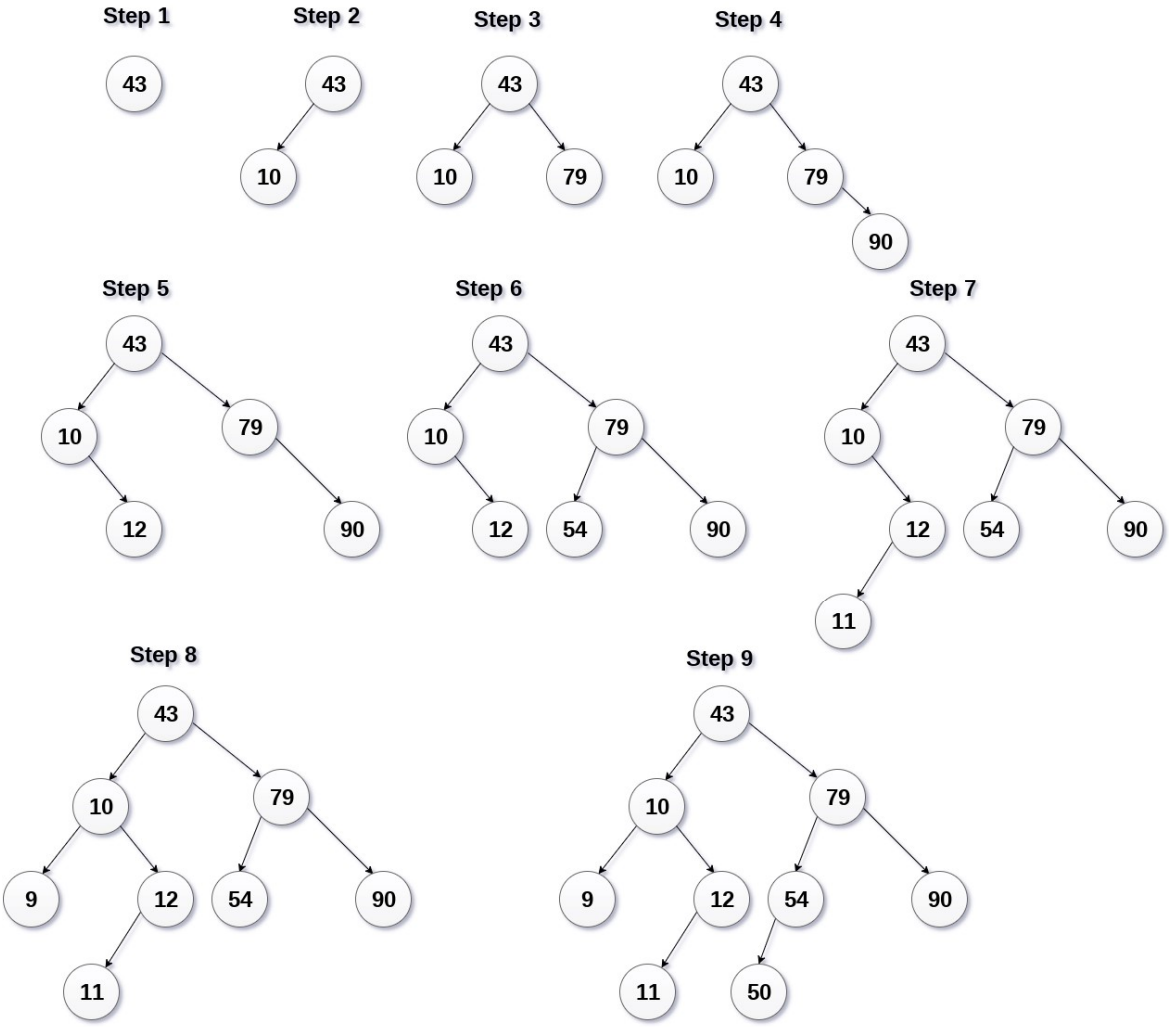
A Binary search tree is shown in the above figure. As the constraint applied on the BST, we can see that the root node 30 doesn't contain any value greater than or equal to 30 in its left sub-tree and it also doesn't contain any value less than 30 in its right sub-tree.

Create the binary search tree using the following data elements.

43, 10, 79, 90, 12, 54, 11, 9, 50

1. Insert 43 into the tree as the root of the tree.
2. Read the next element, if it is lesser than the root node element, insert it as the root of the left sub-tree.
3. Otherwise, insert it as the root of the right of the right sub-tree.

The process of creating BST by using the given elements, is shown in the image below.



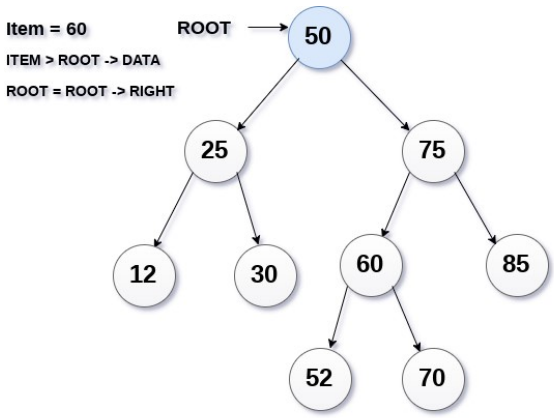
Binary search Tree Creation

Operations on Binary Search Tree

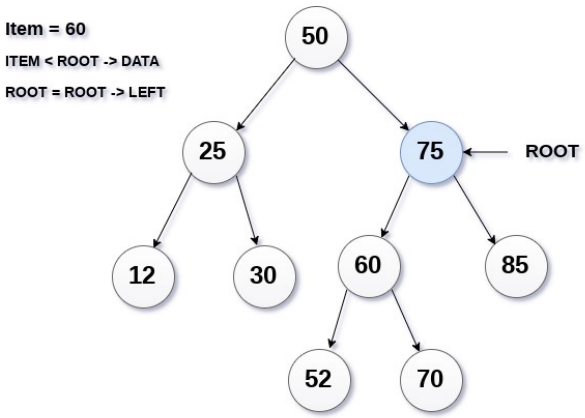
Searching

Searching means finding or locating some specific element or node within a data structure. However, searching for some specific node in binary search tree is pretty easy due to the fact that, element in BST are stored in a particular order.

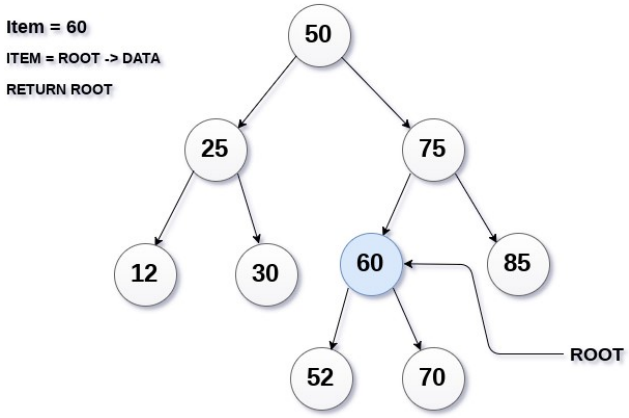
1. Compare the element with the root of the tree.
2. If the item is matched then return the location of the node.
3. Otherwise check if item is less than the element present on root, if so then move to the left sub-tree.
4. If not, then move to the right sub-tree.
5. Repeat this procedure recursively until match found.
6. If element is not found then return NULL.



STEP 1



STEP 2

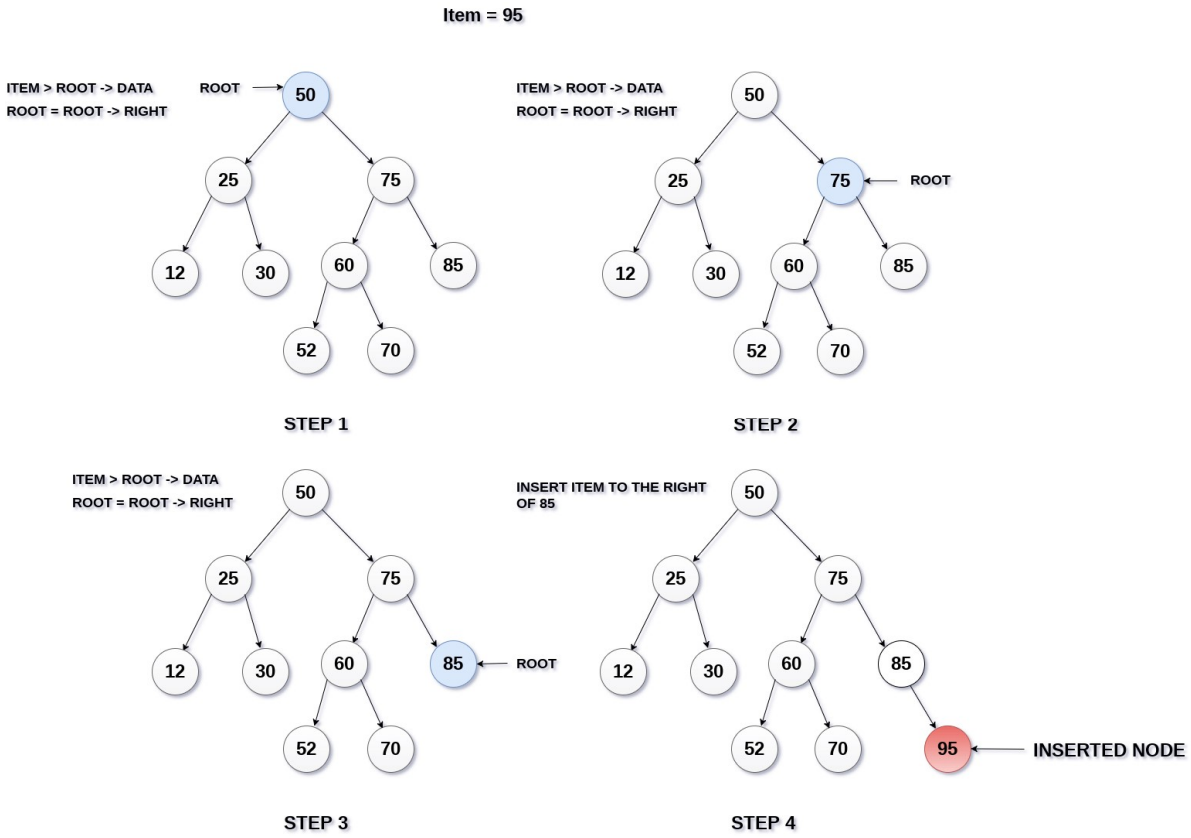


STEP 3

Insertion

Insert function is used to add a new element in a binary search tree at appropriate location. Insert function is to be designed in such a way that, it must node violate the property of binary search tree at each value.

1. Allocate the memory for tree.
2. Set the data part to the value and set the left and right pointer of tree, point to NULL.
3. If the item to be inserted, will be the first element of the tree, then the left and right of this node will point to NULL.
4. Else, check if the item is less than the root element of the tree, if this is true, then recursively perform this operation with the left of the root.
5. If this is false, then perform this operation recursively with the right sub-tree of the root.



Rank-Balanced Binary Search Trees

A ranked binary tree is a binary tree whose nodes have integer ranks, with leaves having rank zero and missing nodes having rank minus one. If x is a child, the rank difference of x, Δx, is the rank of its parent minus the rank of x. The rank of a ranked tree is the rank of its root.

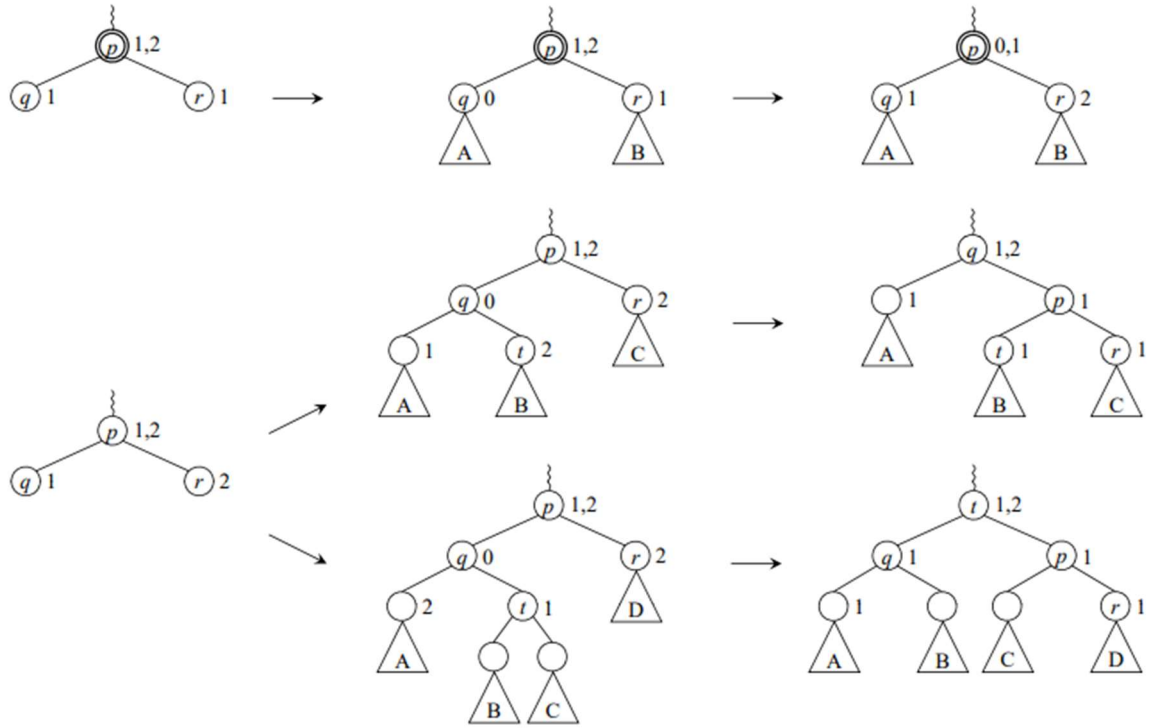
In general there is at most one violation: a node q has Δq = 0, and exactly one child of q has rank difference one. Let p and r be the parent and sibling of q, respectively. There are two cases :

• Δr = 1: Increase the rank of p by one. This repairs the violation at q but may create a new violation at p. The children of node p now have rank differences 1 (q) and 2 (r).

• Δr = 2: Assume q is the left child of p; the other case is symmetric. Exactly one child of q has rank difference one. Let t be the right child of q. There are two subcases:

~ Δt = 2: Do a right rotation at q and decrease the rank of p by one. This repairs the violation without creating a new one.

~ Δt = 1: Do a double rotation at t, making q its left child and p its right child. Increase the rank of t by one and decrease the ranks of p and q by one. This repairs the violation without creating a new one.



Permutation and Combinations

Permutation:

Any arrangement of a set of n objects in a given order is called Permutation of Object. Any arrangement of any r ≤ n of these objects in a given order is called an r-permutation or a permutation of n object taken r at a time.

It is denoted by P (n, r)

$$P(n, r) = \frac{n!}{(n-r)!}$$

Theorem: Prove that the number of permutations of n things taken all at a time is n!.

Proof: We know that

$$n_{P_n} = \frac{n!}{(n-n)!} = \frac{n!}{0!} = \frac{n!}{1} = n!$$

Combination:

A Combination is a selection of some or all, objects from a set of given objects, where the order of the objects does not matter. The number of combinations of n objects, taken r at a time represented by n_{Cr} or C (n, r).

$$n_{C_r} = \frac{n!}{r!\,(n-r)!}$$

Proof: The number of permutations of n different things, taken r at a time is given by

$$n_{P_r} = \frac{n!}{(n-r)!}$$

As there is no matter about the order of arrangement of the objects, therefore, to every combination of r things, there are r! arrangements i.e.,

$$n_{P_r} = r!\ n_{C_r} \quad \text{or} \quad n_{C_r} = \frac{n_{P_r}}{r!} = \frac{n!}{(n-r)!r!},\ n \geq r$$

Thus,

$$n_{C_r} = \frac{n!}{r!(n-r)!}$$