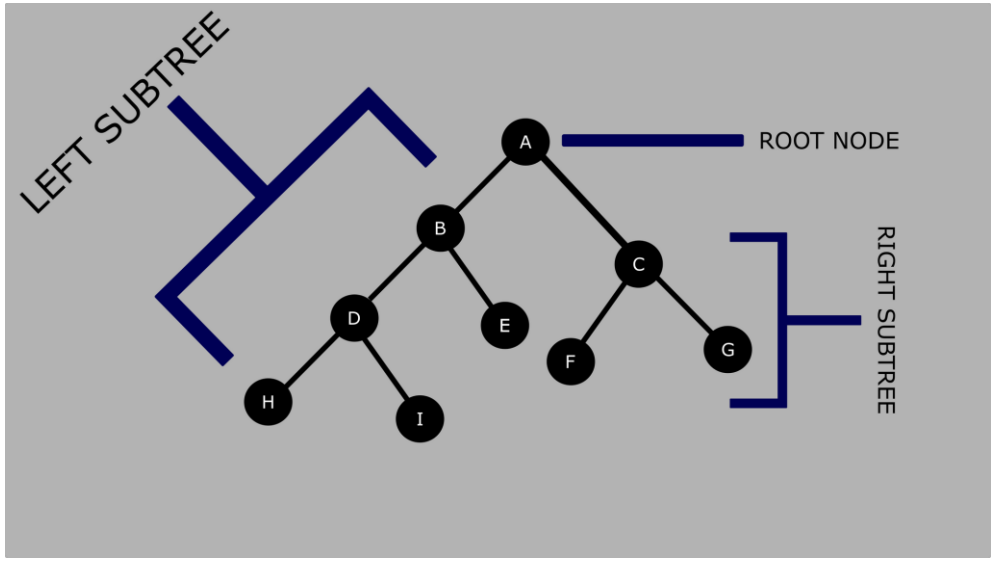
Data Structures and Algorithms

Lecture 28: Binary Search Tree

Binary Search Tree

A binary search tree is a binary tree made up of nodes. Each node has a key signifying its value.

- The value of the nodes on the left subtree are smaller than the value of the root node. And the value of the nodes on the right subtree are larger than the value of the root node.
- The root node is the parent node of both subtrees.



Binary Search Tree

- A is the root node.
- The left subtree begins at B while the right subtree begins at C.
- Node A has two child nodes – B and C.
- Node C is the parent node to F and G. F and G are siblings.
- Node F and G are know as leaf nodes because they do not have children.
- Node B is the parent node to D and E.
- Node D is the parent node to H and I.
- D and E are siblings as well as H and I.
- Node E is a leaf node.

Binary Search Tree

- **Root:** The topmost node in the tree.
- **Parent:** A node with a child or children.
- **Child:** A node extended from another node (parent node).
- **Leaf:** A node without a child.

Binary Search Tree

- Binary search trees help us speed up our binary search as we are able to find items faster.
- We can use the binary search tree for the addition and deletion of items in a tree.
- We can also represent data in a ranked order using a binary tree. And in some cases, it can be used as a chart to represent a collection of information.

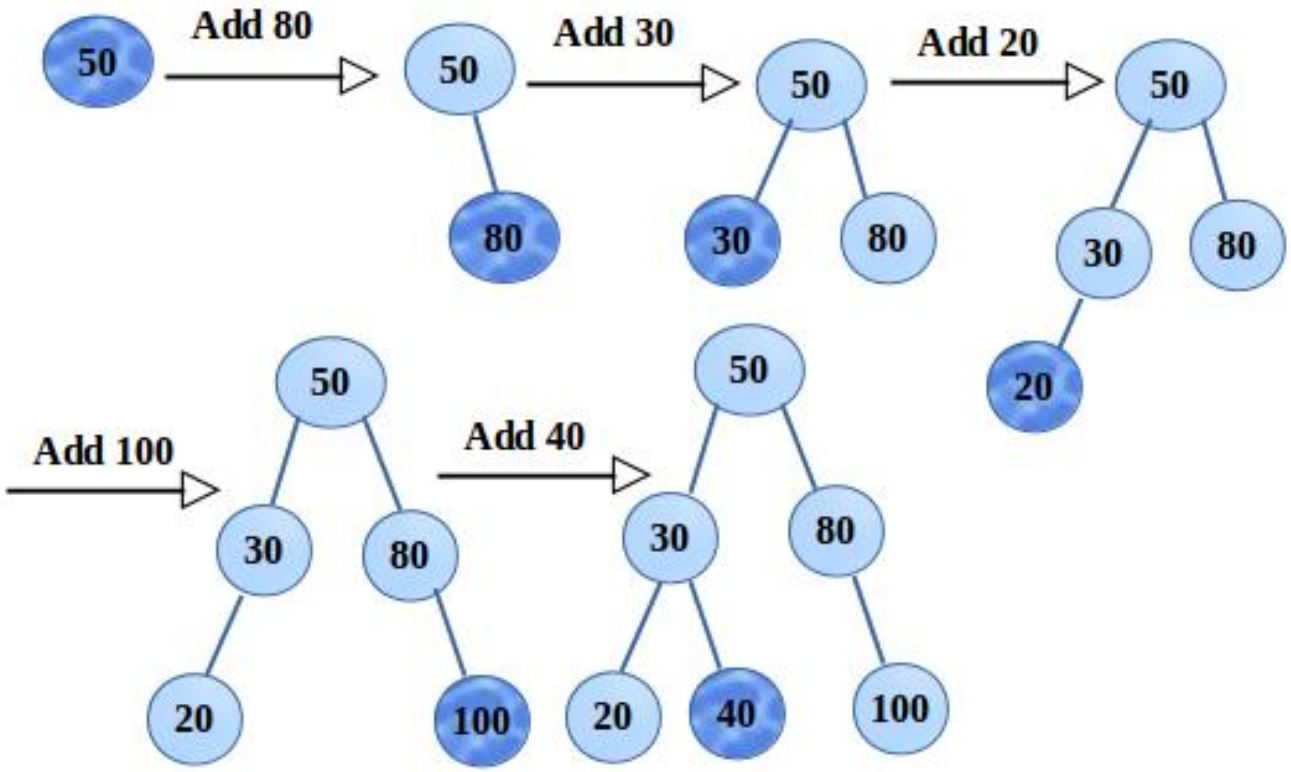
Binary Search Tree

Binary search trees support three main operations:

- **Searching:** For searching, we have to traverse all elements (assuming we do breadth-first traversal). Therefore, searching in a binary tree has worst-case complexity of O(n).
- **Insertion:** For inserting, we have to traverse all elements. Therefore, insertion in a binary tree has worst-case complexity of O(n).
- **Deletion:** For deletion, we have to traverse all elements to find 2 (assuming we do breadth first traversal). Therefore, deletion in binary tree has worst case complexity of O(n)

Binary Search Tree - Insertion

Have to build a BST of the keys, 50, 80, 30, 20, 100, 40. It can be clearly seen below, for inserting, first the key is compared is compared with the root, if smaller then goto Left subtree else Right subtree. The same step is repeated until all the keys are inserted.



Binary Search Tree - Insertion

Algorithm

1. Create a new BST node and assign values to it.

2. insert(node, key)

   i) If root == NULL,

      return the new node to the calling function.

   ii) if root=>data < key

      call the insert function with root=>right and assign the return value in root=>right.

      root->right = insert(root=>right,key)

   iii) if root=>data > key

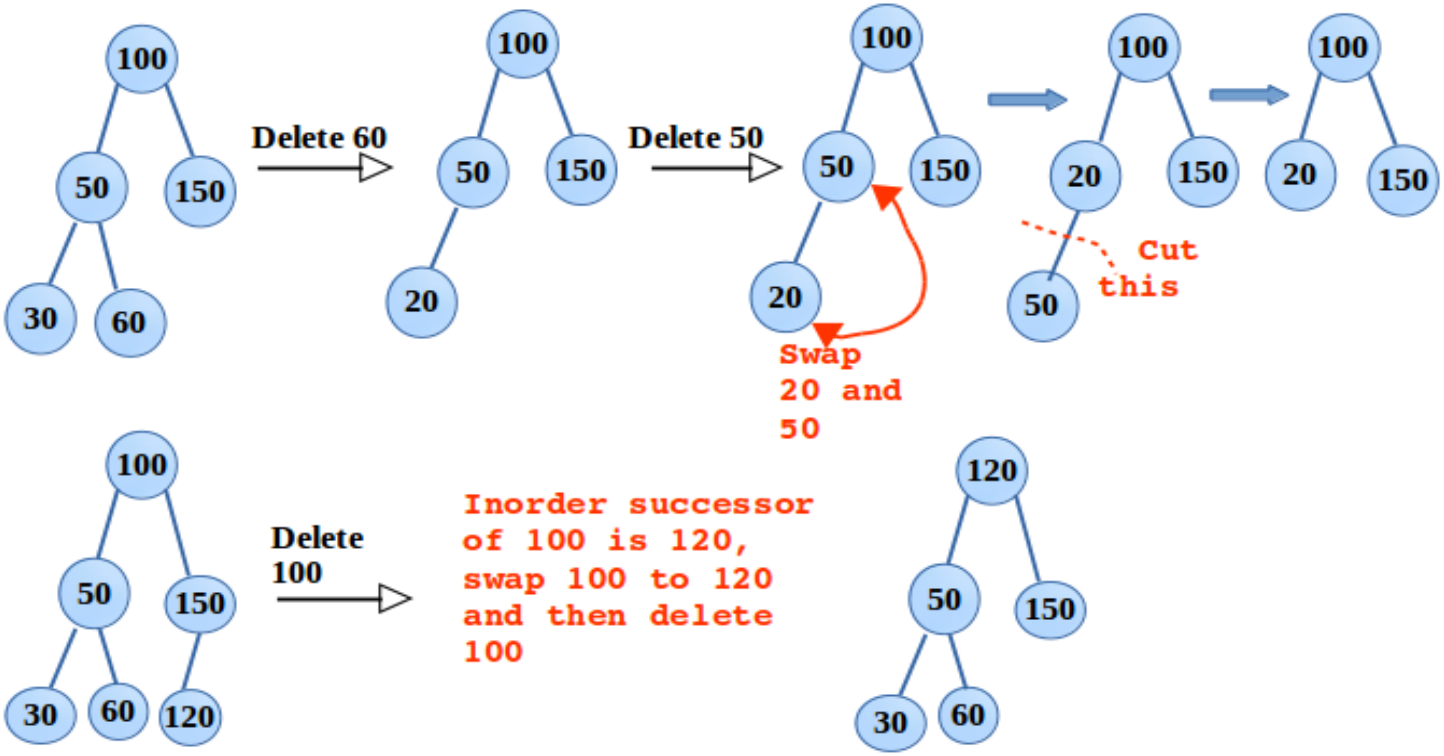
      call the insert function with root->left and assign the return value in root=>left.

      root=>left = insert(root=>left,key)

3. Finally, return the original root pointer to the calling function.

Binary Search Tree - Deletion

- Deleting node has no child, therefore just delete the node(made to point NULL)
- Deleting node has 1 child, swap the key with the child and delete the child.
- Deleting node has 2 children, in this case swap the key with inorder successor of the deleting node. It should be noted, inorder successor will be the minimum key in the right subtree (of the deleting node).



Binary Search Tree - Deletion

1.**Leaf Node**

- If the node is leaf (both left and right will be NULL), remove the node directly and free its memory.

2. **Node with Right Child**

If the node has only right child (left will be NULL), make the node points to the right node and free the node.

3. **Node with Left Child**

If the node has only left child (right will be NULL), make the node points to the left node and free the node.

4. Node has both left and right child

If the node has both left and right child,

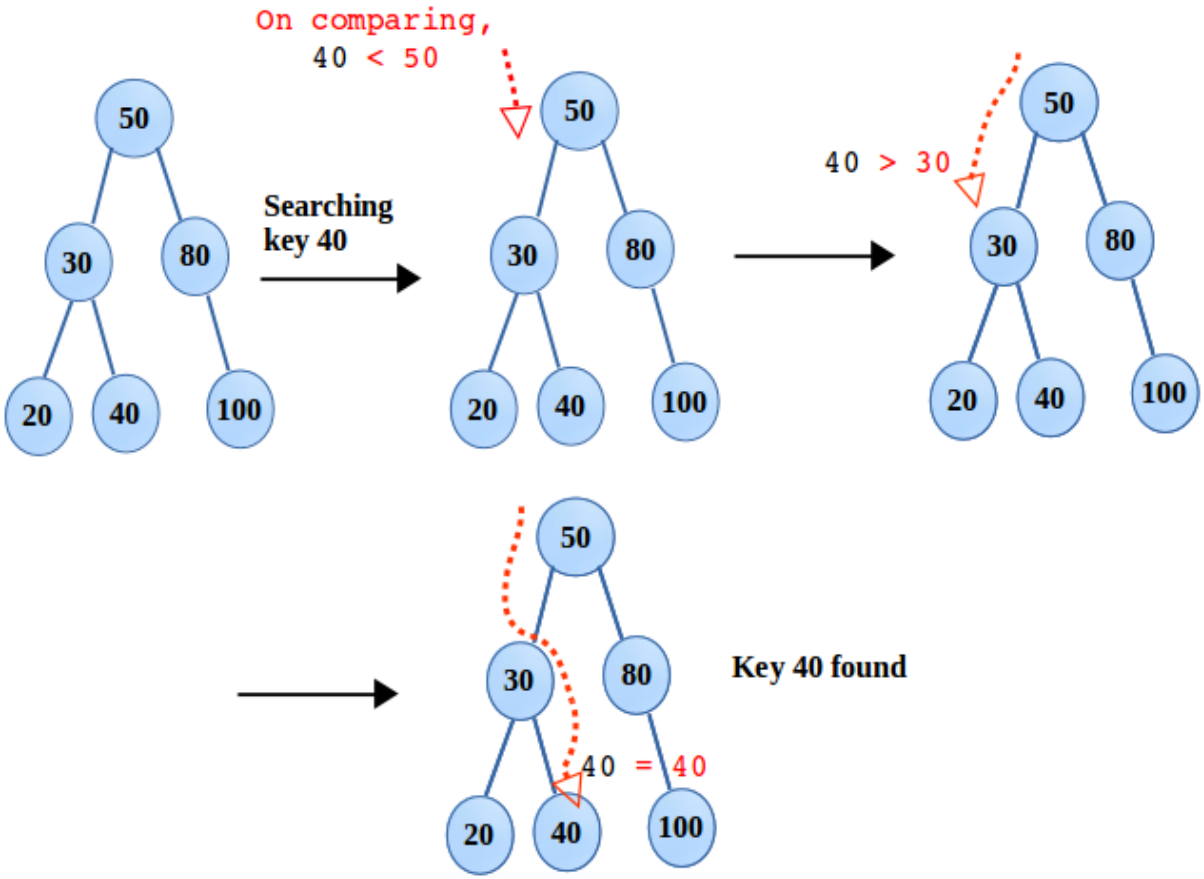
1.find the smallest node in the right subtree. say min

2.make node->data = min

3.Again delete the min node.

Binary Search Tree - Searching

- The steps follow in the insertion, are same followed here. Only difference is in comparison, if the key is not matched, repeat the steps till reached NULL. That says, desired key is not available in the BST.



Binary Search Tree - Searching

- We compare the value to be searched with the value of the root.
- If it’s equal we are done with the search if it’s smaller we know that we need to go to the left subtree because in a binary search tree all the elements in the left subtree are smaller and all the elements in the right subtree are larger.
- Repeat the above step till no more traversal is possible
- If at any iteration, key is found, return True. Else False.