CS 1203: Data Structures **Semester:** Monsoon 2023

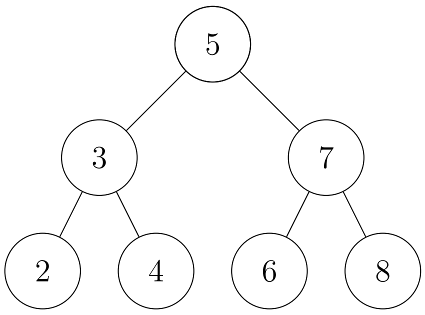
Assignment #5

Instructor: Subhamoy Mitra **Student Name:** Varsha Baisane
Due: September 29, 2023 varsha.baisane_ug25#@ashoka.edu.in

Broad topics discussed:

1. Binary tree: A binary search tree is a data structure composed of nodes each of which have a key, may also have an additional element value that provides info about that node. each node has a left child, a right child and a parent. each node in the tree contains pointers to the other nodes within the tree.

Enumeration: Binary search trees have keys and the nodes are positioned such that if x is a node, then the left subtree having node y has a key $\leq$ x.key and the right subtree node y has a key $\geq$ x.key. This thus maintains the binary tree structure as follows:



2. Binary search: The search algorithm thus proceeds using the key that we are looking for. We traverse the tree using a anode and if the key of this anode matches, the anode is returned. If not, it compares the key values with the node, if lesser, it goes to the left child if greater goes to the right child.

3. Inserting an element in BST: Inserting an element into a binary search tree occurs in a similar approach, initializing a anode, the int k to insert and the key the parameter based on what you traverse. If the anode is empty, I insert the element right there, if not compare and move left/right based on value of key. Also, of the key already exist, you return that it already exist in the tree.

Types of traversals:

A. Inorder traversal: Inorder traversal follows the Left-Root-Right pattern, such that:
The left subtree is traversed first
Then the root node for that subtree is traversed
Finally, the right subtree is traversed
This approach of traversing a binary tree gives us a sorted tree and proves useful in getting the elements in ascending order.

B. Postorder traversal:
The left subtree is traversed first
The right subtree is traversed next
The root node for that subtree is traversed last
Preorder traversal used when we want to create a copy or clone of a BST. Start by creating the root node and then recursively create the left and right subtrees.

C. Preorder traversal:
The root node for that subtree is traversed first
The left subtree is traversed next
The right subtree is traversed finally
Postorder traversal is used in deleting a BST or releasing memory occupied by the tree. You first delete the children and then the current node. It's also useful in calculating the height or balancing factor of a node.

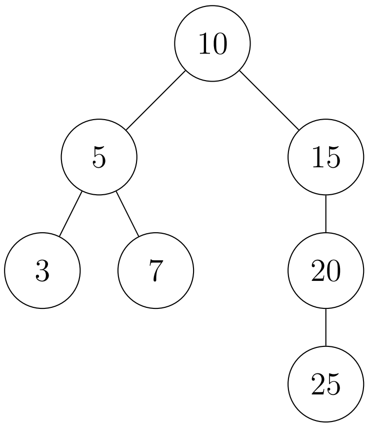
4. Deleting an element in a BST:
Deleting an element in a BST takes place as follows: If the node to be deleted has the key k which is of the current node anode and the only node, delete it.

If the node has only one child- left or right, you delete the node and then set the left and right children of the deleted node to the parent of the deleted node appropriately, i.e. taking care of the BST property. if there exists both child, then after deleting the node, their restructuring is done by obtaining the inorder successor of the tree.

1. If the condition $(\text{temp} \rightarrow \text{left} \neq \text{NULL})\&\&(\text{temp} \rightarrow \text{right} \neq \text{NULL})$ is true for the node with key 10 because it has both left and right children.

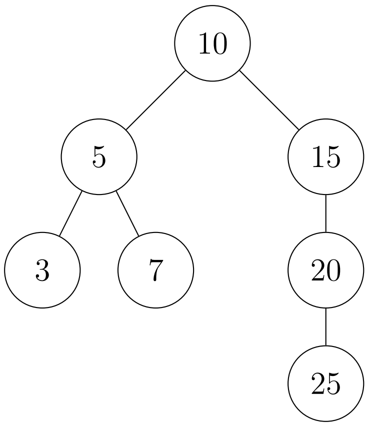
   We start with temp pointing to the node we want to delete (temp = 10), inparent is initialized to temp, and intemp is initialized to the right child (intemp = 15).

   We want to find the in-order successor, which is the leftmost node in the right subtree. In this case, intemp has a left child (20), so we continue to the left:

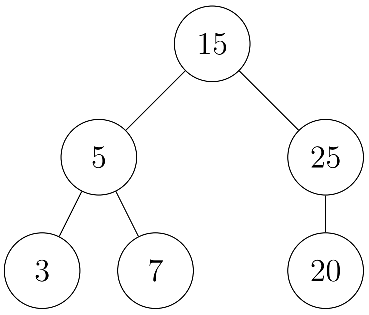
   intemp = 20 inparent = 15

   Since intemp has a left child (20 has a left child), we continue moving to the left:



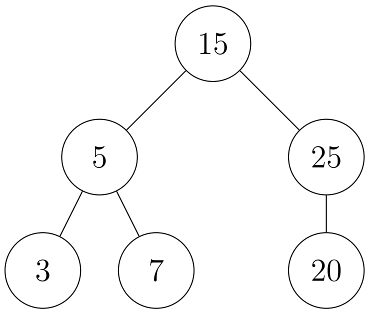
intemp = 25 inparent = 20

Now, intemp doesn't have a left child, so it's the in-order successor. We swap the keys of temp and intemp:



temp→ key $=10 is swapped with$ intemp → key =15

The node with key 10 has been effectively moved to the position of the node we wanted to delete. Now, temp points to the node with key 15 (the in-order successor), and inparent points to its parent (the right child of the original temp):



The in-order successor node with key 15 has replaced the original node with key 10. At this point, you can proceed to delete the node with key 10 (e.g., by making its parent's left or right child point to NULL, depending on the tree structure).

Time Complexity:
The depth of the binary tree may be O(n), and thus are the time complexities of the search, insert, delete operations.
The space complexity is stays constant O(n), moreover balanced binary tree will reduce the time complexities to O(log n).