Your Ultimate Guide To Landing Top AI roles

DECODE AiML

2.16.1

Binary Search Tree

→ A binary Search Tree (BST) is a binary tree such that
① all left subtree elements should be less than root data
② all right subtree elements should be greater than root data.

→ This property should be satisfied at every node in the Tree.

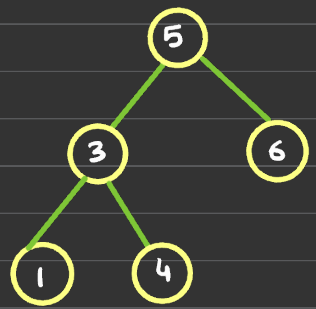
Properties of BST

→ If we perform Inorder Traversal of a BST, it will get all the elements in increasing order

→ In a BST, leftmost element will be the least and rightmost element will be the greatest element.

→ No of BST possible with n distinct keys

$$\boxed{\#BST = \frac{{}^{2n}C_n}{n+1}}$$



Inorder : 1, 3, 4, 5, 6

BST Implementation in Python

```
Class TreeNode:
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None
```

Value	
Left R	right R

Node structure

Key operations in a BST

① **Insertion**

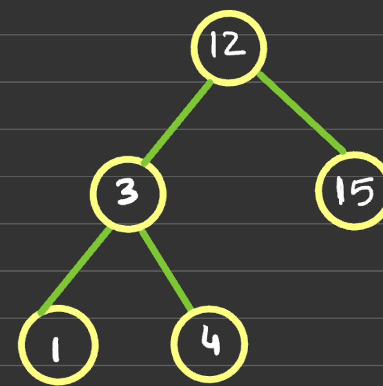
→ Insert a key in a BST

② **Search**

→ Search for a key in a BST

③ **Deletion**

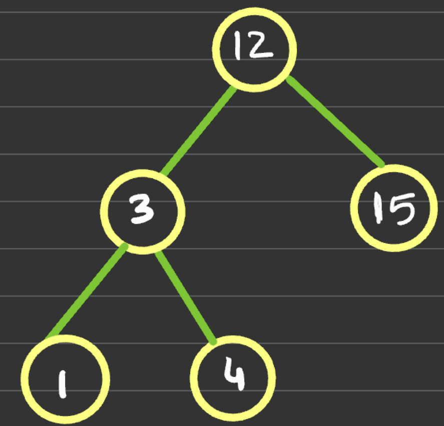
→ Delete a key in a BST



Search in a BST

```
def search(root, key):
    if root is None:
        return root
    if root.key == key:
        return root
    if key > root.key:
        return search(root.right, key)
    return search(root.left, key)
```

→ Search key → 4



* Time & Space Complexity

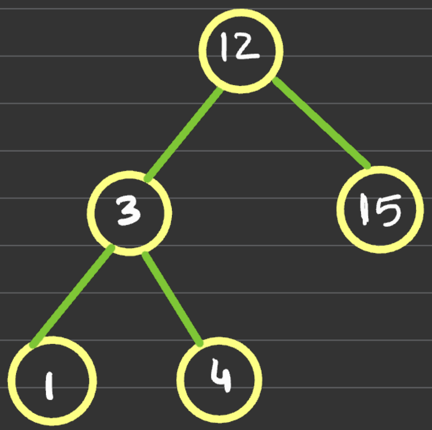
Time Complexity = $O(n)$

Space Complexity = $O(n)$

Insertion in a BST

```
def insert(root, key):
    if root is None:
        return Node(key)
    if key < root.key:
        root.left = insert(root.left, key)
    elif key > root.key:
        root.right = insert(root.right, key)
    return root
```

→ Insert key → 8



* Time & Space Complexity

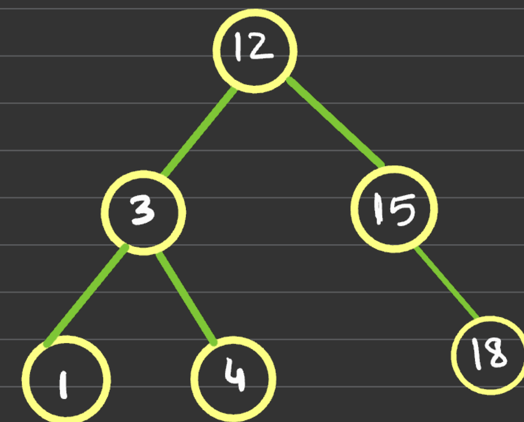
Time Complexity = $O(n)$

Space Complexity = $O(n)$

Deletion in a BST

→ When deleting a node, there are 3 cases

→ Delete key → 18



① Node has no children (leaf node)

→ Just remove it

② Node has 1 child

→ Replace the node with it's child

③ Node has 2 children

→ Find the inorder successor (smallest node in right subtree) or inorder predecessor (largest in left subtree)

→ Swap the value with the target Node.

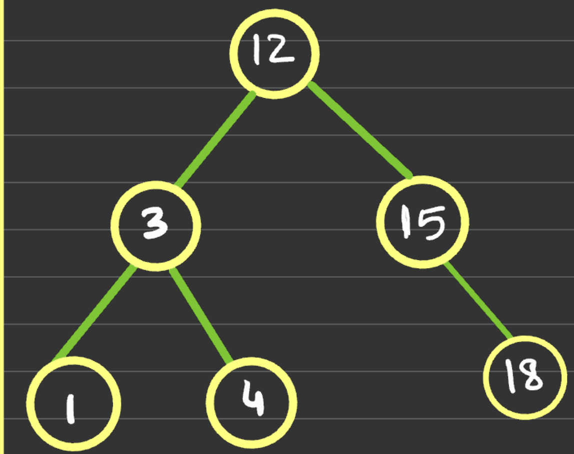
→ Delete that successor / Predecessor.

Deletion in a BST

```
def deleteNode(root, key):
    if root is None:
        return root
    if key < root.key:
        root.left = deleteNode(root.left, key)
    elif key > root.key:
        root.right = deleteNode(root.right, key)
    else: #node found
        if root.left==None:
            return root.right
        elif root.right == None:
            return root.left
        temp = inorderPredecessor(root.left)
        root.key = temp.key
        root.left = deleteNode(root.left, temp.key)
    return root
```

Case I or II

Case III



* Time & Space Complexity

Time Complexity = $O(n)$

Space Complexity = $O(n)$

Binary Tree Vs Binary search Tree

→ If Tree is balanced. means height (h) = $\log n$.

	Binary Tree	BST
Insert	$O(\log n)$	$\log(n)$
Search	$O(n)$	$\log(n)$
Delete	$O(\log n)$	$\log(n)$

Next Lecture

└→ AVL Tree

Like Subscribe