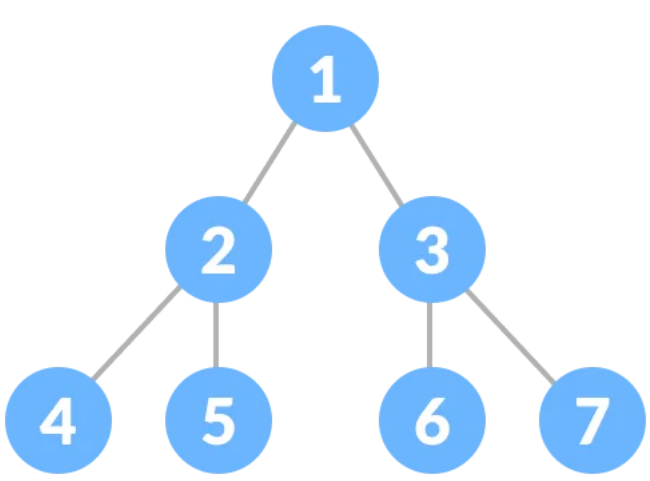
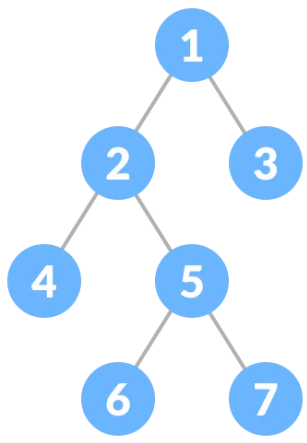
Binary Tree

What is Binary Tree?

A binary tree is a tree non-liner data structure in which every node (including root node) have at most 2 children nodes





A binary tree Supports 4 basic operations for interaction:

1. Insertion
2. Deletation
3. Traversal

Why & When Binary Trees?

Binary trees are used when we want to store data in a hierarchical order. Insertion and deletion in quicker in trees as compared to array and linked lists. And accessing a element is faster as compared to linked list and slower when compared to arrays.

Some Real life implementation of tree are listed below:

- Routing Tables: A Routing Table is used to link routers in a network. It is usually implemented using a binary tree data structure. A tree data structure will store the location of routers based on their IP addresses. Routers with similar address are grouped under a single sub tree.
- Indices for Databases: In database indexing, Binary trees are used to sort data for simplified searching, insertion, and deletion.
- Binary Space partition: Used in almost any 3D video game to determine which objects need to rendered.

Time Complexity

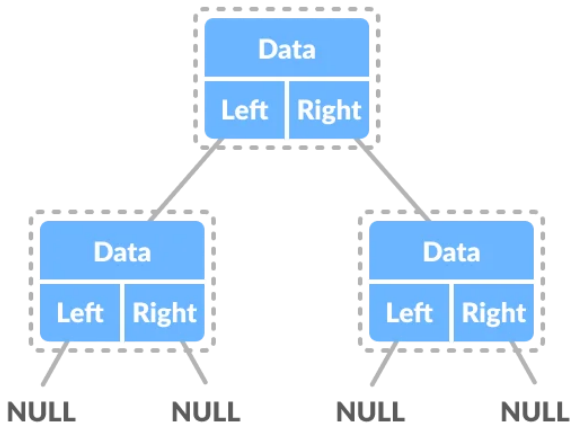
Aa Operation	Time-Complexity	Explanation
Untitled		
Searching	O(n)	For searching element which is at the bottom node, we have to traverse all elements (assuming we do breadth first traversal). Therefore, searching in binary tree has worst case complexity of O(n).
Insertion	O(n)	For inserting element at the most bottom node we have to traverse all elements (Therefore, insertion in binary tree has worst case complexity of O(n).
Deletion	O(n)	For Deleting element at the most bottom node we have to traverse all elements (Therefore, Deleting in binary tree has worst case complexity of O(n).
Untitled		

Code Implementation

- Constructing Node Class for Binary tree

Each Node of the Binary Tree consist of three items

1. Data item (value of the Node)
2. Address of left child
3. Address of right child




```
class Node
{
public:
   int data;
   Node* left;
   Node* right;


   Node(int d)
   {
      data = d;
      left = NULL;
      right = NULL;
   }

};
```

- Traversing a binary Tree

There are 4 types of traversal in a binary tree

1. InOrder Traversal
2. PreOrder Traversal
3. PostOrder Traversal
4. LevelOrder Traversal

    - InOrder Traversal

    `Left-Subtree -> Root -> Right-Subtree`

    In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order.

    $TimeComplexity = O(n)$

    $TimeComplexity = O(n)$

    Binary Tree Inorder Traversal - LeetCode

    Given the root of a binary tree, return the inorder traversal of its nodes' values. Example 1: Input: root = [1,null,2,3]Output: [1,3,2] Example 2: Example 3: Example 4: Example 5: Input: root = [1,null,2]Output: [1,2] Constraints: The number of nodes in the tree is in the range [0,

    https://leetcode.com/problems/binary-tree-inorder-traversal


    ```
    // Recursive Soloution
    void inorderPrint(Node* root)
    {
    ```

```
   // Base Case
   if(root == NULL){return;}
   else
   {
      // Calling on Left sub tree
      inorderPrint(root->left);

      // Printing Root Data
      cout<<root->data<<" ";

      // Calling on right sub tree
      preorderPrint(root->right);
   }
}
```

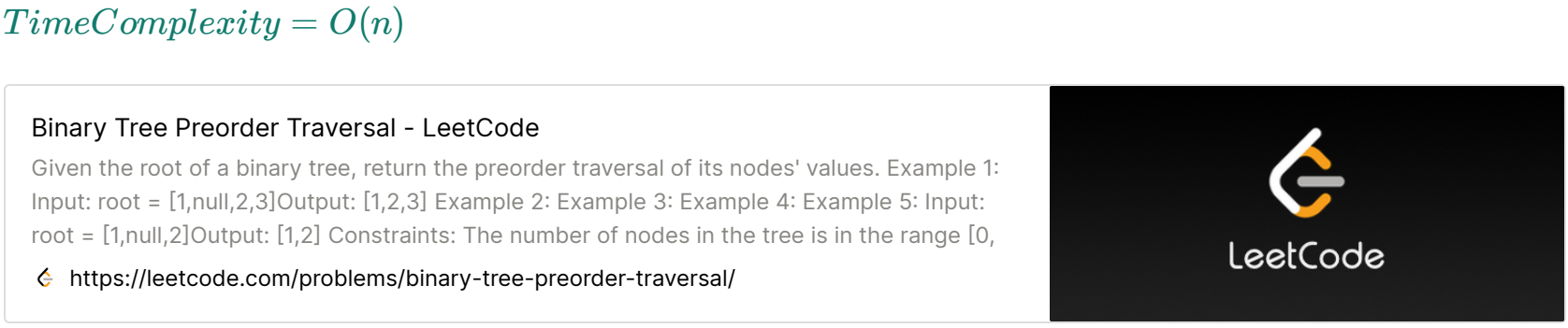
- PreOrder Traversal

`Root -> Left-Subtree -> Right-Subtree`

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree.

$$TimeComplexity = O(n)$$

$$TimeComplexity = O(n)$$




```
void preorderPrint(Node* root)
{

   // Base Case
   if(root==NULL){return;}
   else
   {
      cout<<root->data<<" ";

      // Calling on Left sub tree
      preorderPrint(root->left);

      // Calling on right sub tree
      preorderPrint(root->right);
   }
}
```

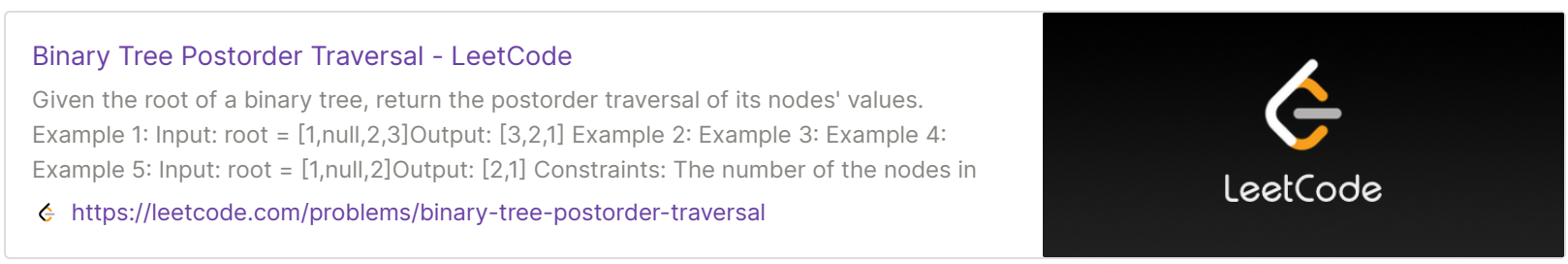
- PostOrder Traversal

`Left-Subtree -> Right-Subtree -> root`

Postorder traversal is used to delete the tree

$$TimeComplexity = O(n)$$

$$TimeComplexity = O(n)$$




```
// Recursive Method
void postorderPrint(Node* root)
{
   // Base case
   if(root == NULL){return;}
   else
   {
      // Calling on Left sub tree
      postorderPrint(root->left);

      // Calling on right sub tree
```

```
        postorderPrint(root->right);

        // Printing Root Data
        cout<<root->data<<" ";
    }
}
```

Algorithms or Techniques Used in Linked List

Count Total Number of Nodes in B-Tree

Count Complete Tree Nodes - LeetCode

Given the root of a complete binary tree, return the number of the nodes in the tree. According to , every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible.

https://leetcode.com/problems/count-complete-tree-nodes/

Approach: Recursive

- Base Case: if root is `NULL` return `0`
- Total Number of Nodes = No of nodes in left sub-tree + Number of nodes in right Sub tree + 1 (Root Node)`

```
// Recursive Method
int numberOfNodes(Node* root)
{
   // Base Case
   if(root==NULL){return 0;}
   else
   {
      // Number of nodes in Left Sub Tree
      int lef = numberOfNodes(root->left);

      // Number of Nodes in Right Sub Tree
      int rig = numberOfNodes(root->right);

      // Total Number of Nodes
      return (lef+ rig)+1;
   }
}
```

Max depth or Height Of B-Tree

Maximum Depth of Binary Tree - LeetCode

Level up your coding skills and quickly land a job. This is the best place to expand your knowledge and get prepared for your next interview.

https://leetcode.com/problems/maximum-depth-of-binary-tree/

Approach: Recursive

- Base Case: if root is NULL return 0
- Height of B-Tree = Max among left ans right sub tree + 1 (Root Node)

```
int height(Node* root)
{
   // Base Case
   if(root == NULL){return 0;}
   else
   {
      // Height of left Sub Tree
      int left = height(root->left);

      // Height of right Sub Tree
```

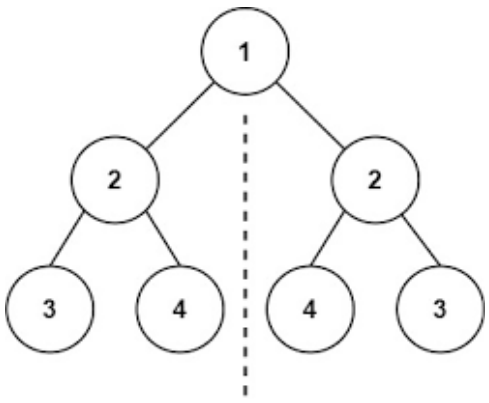
```
      int right = height(root->right);

      // Height of full Tree
      return max (left, right)+1;
   }
}
```

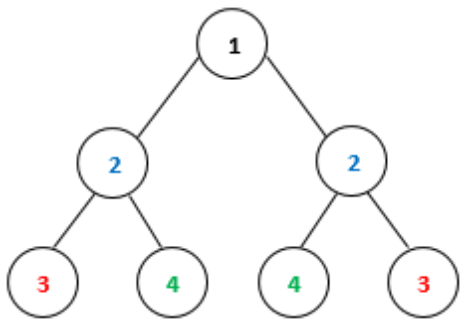
Symmetric Binary Tree

Symmetric Tree - LeetCode

Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center). Example 1: Input: root = [1,2,2,3,4,4,3]Output: true Example 2: Input: root = [1,2,2,null,3,null,3]Output: false Constraints: The number of nodes in the tree is in the range [1,

https://leetcode.com/problems/symmetric-tree





In the above picture every node is colour coded with its corresponding symmetric node

Approach: Recursive

Conditions for a B-Tree to be symmetric

- both of nodes of same level must be same, they both can also be null for eg: node with value 2 in above pic.
- If two nodes have parents of same level then left child of left node must be equal and symmetric to right child of right node eg: node with value 3 in above pic.

```
bool symmetricHelperFunction(Node* left, Node* right)
{
   // If both are NULL then its symmetric this is also the base case
   if(left==NULL and right==NULL){return true;}

   // If any of the one node is NULL while the other isn't then its not symmetric
   if(left==NULL and right!=NULL){return false;}
   if(left!=NULL and right==NULL){return false;}

   // If values of left and right aren't same then its not symmetric
   if(left->data!=right->data){return false;}

   // If any of the above conditions do not occour then these two nodes are symmetric now proceed to their childrens
   return symmetricHelperFunction(left->left, right->right) and symmetricHelperFunction(left->right, right->left);
}


bool isSymmetric(Node* root)
{
   // An empty tree is also an symmetric tree
   if(root==NULL){return true;}

   return symmetricHelperFunction(root->left, root->right);
}
```

Check if a Value is present in B-Tree

Approach: Recursive

```
bool isPresent(Node* root, int key)
{
   // If tree is empty return false
   if(root==NULL){return false;}

   // If key is found at root return true (Base Case)
   if(root->data == key){return true;}

   // If any of the above conditions dosent satisfy froceed to left and right childrens
   return isPresent(root->left, key) or isPresent(root->right, key);
}
```

Find Minimum value in B-Tree

Approach: Recursive

```
int minValue(Node* root, int &ans)
{
   // ans = INT_MAX
   // If the tree is empty
   if(root==NULL){return -1;}

   // If the current root's value is less than ans then update ans
   if(root->data <= ans){ans = root->data;}

   // return min of ans and left sub tree and right sub tree and ans
   return min(ans, min(minValue(root->left,ans), minValue(root->right,ans)));
}
```

Find Maximum vale in B-Tree

Approach: Recursive

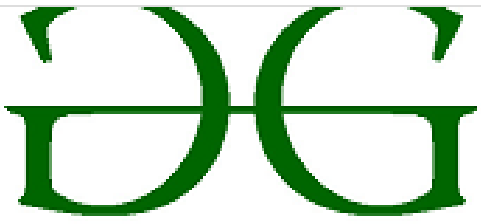
```
int maxValue(Node* root, int &ans)
{
   // ans = INT_MIN
   if(root==NULL){return -1;}
   if(root->data >= ans){ans = root->data;}

   // return max of ans and left sub tree and right sub tree and ans
   return max(ans, max(maxValue(root->left, ans), maxValue(root->right, ans)));
}
```

Count Total Number of Leaves in Binary Tree

Practice | GeeksforGeeks | A computer science portal for geeks

Platform to practice programming problems. Solve company interview questions and improve your coding intellect

https://practice.geeksforgeeks.org/problems/count-leaves-in-binary-tree/



Approach: Recursive

- In a leaf node both left and right children are NULL
- i.e (`root->left == NULL and root->right == NULL`)

```
int countLeaves(Node* root)
{
  // If tree is empty return NULL
  if (root==NULL){return 0;}
```

```
  // Condidtion for Leaf Node
  if(root->left == NULL and root->right == NULL){return -1;}

  // If the above condition does not satisfy then there exist left or right child

  // No of leaf Nodes in left subtree
  int lef = countLeaves(root->left);

  // No of leaf Nodes in right subtree
  int rig = countLeaves(root->right);

  // Total Leaves Nodes is sum of these two
  return lef+rig;
}
```

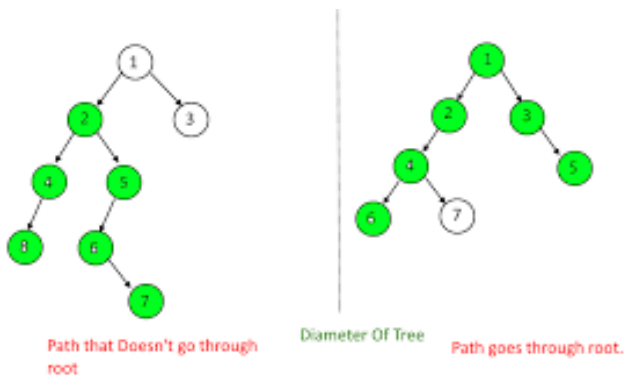
Diameter of a Binary Tree

Diameter of Binary Tree - LeetCode

Given the root of a binary tree, return the length of the diameter of the tree. The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

https://leetcode.com/problems/diameter-of-binary-tree/

Approach: Recursive



There are two possibilities

- The diameter will include both left and right sub-tree of root as shown in rightmost above image
- It will be composed in any of the sub-tree of root and will not include root as shown in leftmost above image

This approach can be optimised further

Time Complexity = O(No.Of Nodes*Height)

Worst Case Time Complexity = O(N^2)

```
// Function To calculate height of Binary Tree
    int height(TreeNode* root)
    {
        if(root==NULL){return 0;}
        int lef = height(root->left);
        int rig = height(root->right);
        return max(lef, rig)+1;
    }


    int diameterOfBinaryTree(TreeNode* root) {
        if(root==NULL){return 0;}

        // This condition represents scenerio on right of above pic
        int option1 = height(root->left)+height(root->right);

        // This condition represents scenerio on left of above pic
        int option2 = diameterOfBinaryTree(root->left);
        int option3 = diameterOfBinaryTree(root->right);

        // Return max of option 1, 2 ,3
        int max1 = max(option1, option2);
        return max(max1, option3);
    }
```