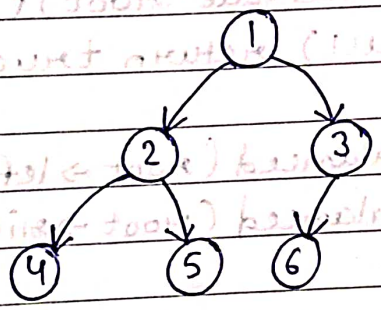
01/12/2023

* Balanced Binary Tree - (leetcode - 110)

Tree is height balanced if tree ki har node ke left subtree and right subtree ki height ka absolute difference atmost 1 hai.
If this condition is true at every node, then tree is height-balanced.

I/P - [1, 2, 3, 4, 5, 6]

O/P - true



At every node, 3 things must be true -

(1) Left subtree should be balanced.
(2) Right subtree should be balanced.
(3) abs (leftHeight - rightHeight) <= 1

Code -

```
class Solution{
public:
    int height (TreeNode * root) {
        if (root == NULL) return 0;

        int leftHeight = height (root -> left);
        int rightHeight = height (root -> right);
        return max (leftHeight, rightHeight) + 1;
    }

    bool isBalanced (TreeNode* root) {
        if (root == NULL) return true;

        bool leftAns = isBalanced (root -> left);
        bool rightAns = isBalanced (root -> right);

        bool diff = (abs (height(root->left) - height(root->right))
                                                    <= 1);

        return diff && leftAns && rightAns;
    }
};
```

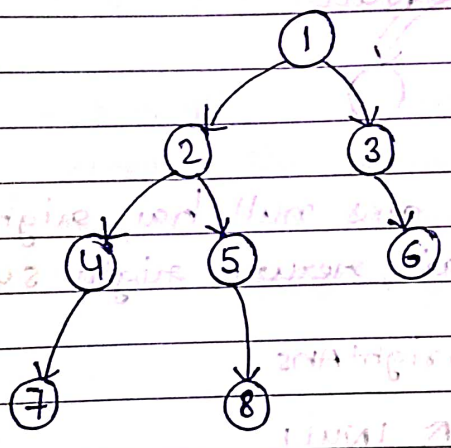
* Lowest Common Ancestor of a Binary Tree (LCA) (Leetcode - 236)

Lowest common ancestor is the lowest node that has both p and q as its descendants.

I/P - [1, 2, 3, 4, 5, null, 6, 7, null, null, 8]

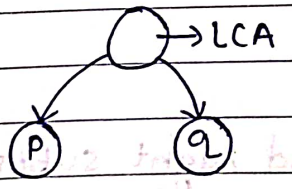
p = 4 q = 8

O/P - 2

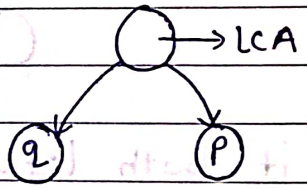


Here, if p = 2 and q = 8. O/P will be 2 i.e. lowest common Ancestor of node 2 and 8 is 2.

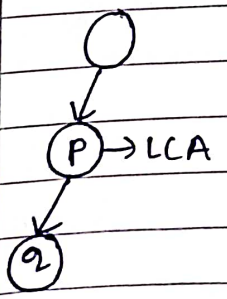
Cases - (1)



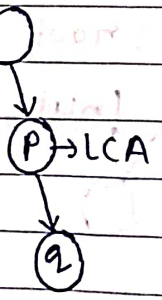
(2)



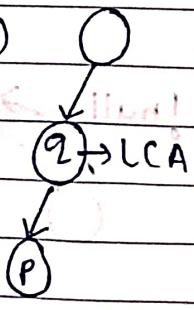
(3)



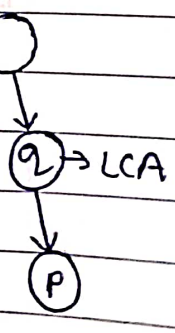
(4)



(5)

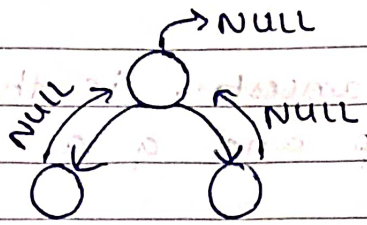


(6)

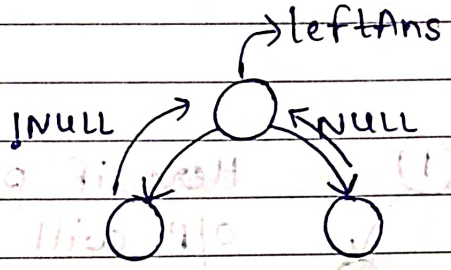


Answer below 4 ways se aa sakta hai— / /

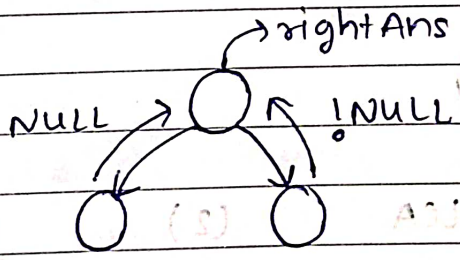
(1) if left subtree ka ans null hai, right subtree ka ans null hai, return null.



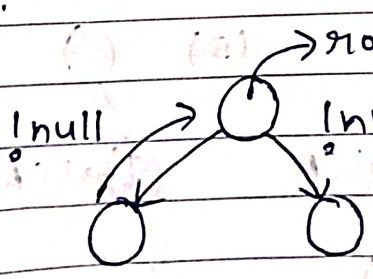
(2) if left subtree ka ans (!null) hai, right subtree ka ans null hai, return left subtree ka ans.



(3) if left subtree ka ans null hai, right subtree ka ans (!null) hai, return right subtree ka ans.



(4) if both left subtree and right subtree ka ans (!null) hai, return root because that root is LCA.



Code -

```
class Solution {
public:

TreeNode* lowestCommonAncestor(TreeNode* root,
    TreeNode* p, TreeNode* q)
{

    if (root == NULL) return NULL;
    if (root == p) return p;
    if (root == q) return q;

TreeNode* leftAns = lowestCommonAncestor(root->left, p, q);
TreeNode* rightAns = lowestCommonAncestor(root->right, p, q);

if (leftAns == NULL && rightAns == NULL) return NULL;
else if (leftAns != NULL && rightAns == NULL) return leftAns;
else if (leftAns == NULL && rightAns != NULL) return rightAns;
else return root;
    }
};
```

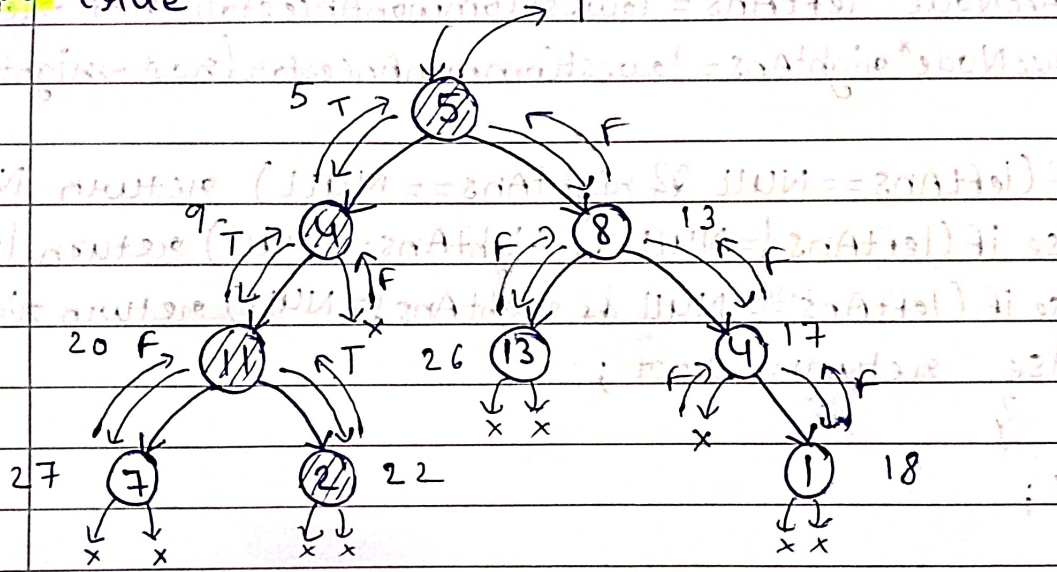
__/__/____

* **Path sum (leetcode - 112)**

Given root of binary tree and an integer targetSum, return true if tree has a root to leaf path such that adding up all the values along the path equals targetSum.

I/P - root = [5, 4, 8, 11, null, 13, 4, 7, 2, null, null, null, 1]
targetSum = 22.

O/P - true



Approach -

= **Base cases -**

- If root = NULL hai, return false.
- If root is a leaf node, then check if - targetSum reached, return true else return false.

= Make recursive calls for left & right subtree.

= return leftAns || rightAns.

Code -

```
class Solution {
public:
    bool solve(TreeNode* root, int &targetSum, int Sum)
    {
        if(root == NULL) return false;

        Sum += root->val;
        if(root->left == NULL && root->right == NULL){
            if (targetSum == sum ) return true;
            return false;
        }

        ~~bool leftAns = solve(root, targetSum, sum);~~

        bool leftAns = solve(root->left, targetSum, Sum);
        bool rightAns = solve(root->right, targetSum, Sum);
        return leftAns || rightAns;
    }

    bool hasPathSum(TreeNode* root, int targetSum){
        int sum = 0;
        return solve(root, targetsum, sum);
    }
};
```

→ Code mei sum ko pass by value kiya hai, if by reference krenge to backtracking wala step bhi add krna pdega.

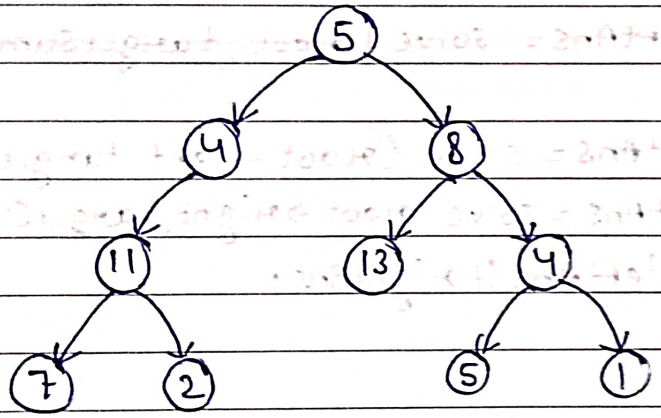
Note - Base case bnane ke liye 2 chize dhyan mei rakhni hai, i.e. rukna khahai.
Other is kya extra kaam kr rhe hai jisme __/__/__ return jana pdd rha hai.

* **Path Sum II (leetcode - 113)**

Given root of binary tree and targetSum, return all root-to leaf path where sum of node values in the path equals targetSum

I/P - root = [5, 4, 8, 11, null, 13, 4, 7, 2, null, null, 5, 1]
targetSum = 22

O/P - [[5, 4, 11, 2], [5, 8, 4, 5]]



Approach -

- Base cases - If root = null, return.
  If root is a leaf node, store temp in ans.
- At every node → sum += root->val;
  temp.push_back(root->val);
- Make recursive calls for left and right subtree.

Code -

```
class Solution {
public:
void solve (TreeNode* root, int &targetSum ,int sum,
            vector<int> temp, vector< vector <int>> &ans){

    if (root == NULL) return;

    sum += root->val;
    temp.push_back(root->val);

    if (root->left == NULL && root->right == NULL){
        if (sum == targetSum) ans.push_back (temp);
        return;
    }
    solve (root->left, targetSum, sum, temp, ans);
    solve (root->right, targetSum, sum, temp, ans);
}

vector <vector <int>> pathSum(TreeNode* root, int targetSum)
{
    int sum= 0;
    vector<int >temp;
    vector <vector<int >> ans;
    solve (root, targetSum, sum, temp, ans);
    return ans;
    }
};
```

→ Code mei sum and temp ko pass by value kr rhe so that backtracking vala step add na krna pde.

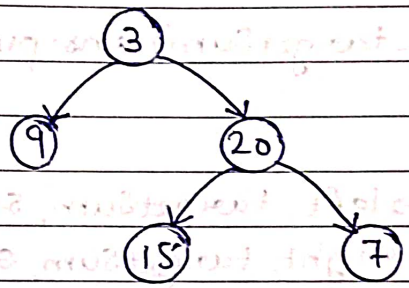
Important -

* **Construct Binary Tree from Preorder and Inorder Traversal (leetcode - 105)**

Given preorder and inorder of a binary tree, construct a tree from it.

I/P - preorder = [3, 9, 20, 15, 7]
inorder = [9, 3, 15, 20, 7]

O/P - [3, 9, 20, null, null, 15, 7]



Approach -

Preorder - NLR

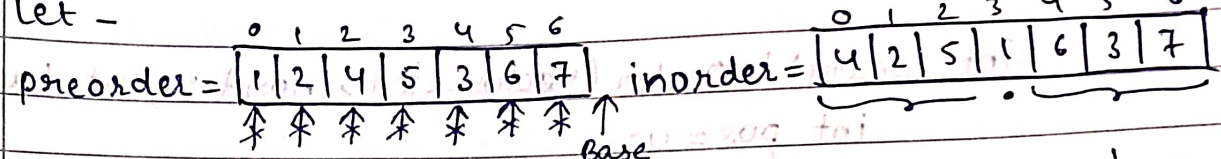
- Preorder array ka leftmost element i.e. 0th index element hamesha root node hoga tree ka.

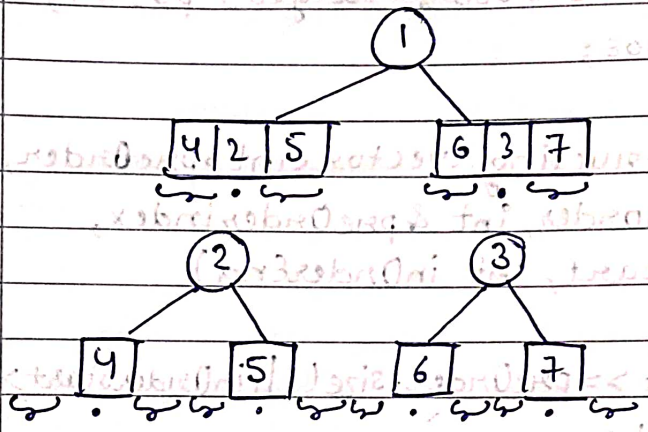
- Inorder - LNR
Inorder array me vo element search kro. Us element ke left ke sare elements left subtree me aayenge and right ke sare elements right subtree me aayenge..

- Iske baad preorder array me index ko increment kr do and repeat above process until preorderIndex < preorder.size()

and InorderStartIndex > InorderEndIndex

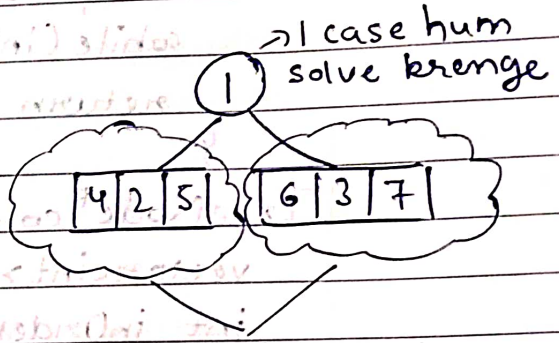
Let -



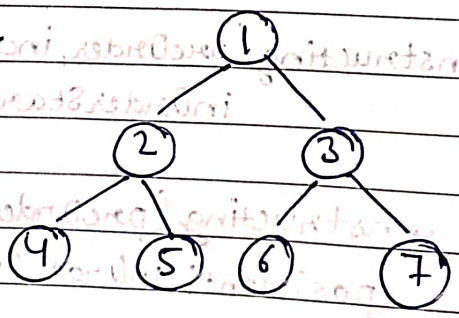


Nothing to solve in left & right, return

Nothing to solve in left & right, return



So, tree is -



Note - Isme preOrderIndex ko hamesha by reference bhejna hai ni to call se return krte time iska index change ho jayega which we don't want i.e. preorder array mei ek index pr ek baar hi jana hai isliye.

Code -

```
class Solution{
public:

int searchInOrder(vector<int>inorder, int target){
        int pos = 0;
        while(inorder[pos] != target) pos ++;
        return pos;
    }
TreeNode* constructing(vector<int>preOrder,
  vector<int>inorder, int &preOrderIndex,
  int inOrderStart, int inOrderEnd)
 {
  if(preOrderIndex >= preOrder.size() || inOrderStart>inOrderEnd)
    return NULL;

  int element= preOrder[preOrderIndex ++];
  TreeNode* Root= new TreeNode(element);

  int position = searchInOrder(inorder, element);

Root->left = constructing(preOrder, inorder, preOrderIndex,
                          inOrderStart, ~~inOrderEnd~~ position-1);

Root->right= constructing(preOrder, inorder,preOrderIndex,
                          position+1, inOrderEnd);

  return Root;
}
```

```
TreeNode* buildTree (vector<int>&preorder,
                     vector<int>&inorder)
{
    int preOrderIndex = 0;
    int inOrderStart = 0;
    int inOrderEnd = inorder.size()-1;
    TreeNode* root = constructing(preorder, inorder,
                     preOrderIndex, inOrderStart, inOrderEnd);

    return root;
}
};
```

→ Is code mei inorder array mei element ko search krne ke har baar O(n) time complexity jari hai.
So, instead of using this, we can use map here. Ye map har element ke corresponding jo index hoga vo btayega.

* **Construct Binary Tree from Inorder and postorder traversal (leetcode-106)**

I/P - inorder = [9, 3, 15, 20, 7]
postorder = [9, 15, 7, 20, 3]

O/P - [3, 9, 20, null, null, 15, 7]

Approach -

- Postorder - LRN
postorder traversal mei root node always rightmost element i.e. last index of array hoga. So, isme postOrderIndex ko postorder.size()-1 se start krna hai.

- Postorder array mei right to left aate time pehle right subtree ke elements aayenge then left subtree ke. That's why pehle right subtree ke liye recursive call lgani hai, then left subtree ke liye.

- Same idhr bhi, postOrderIndex ko by reference pass krna pdega so that jis index pr ja chuke hai uspe fir se na jaye.

Code -

```
class Solution{
public:

void createMapping(map<int,int>valueToIndexMap,
                   vector<int>inorder)
{
   for(int i=0; i<inorder.size(); i++){
        int element= inorder[i];
        valueToMapIndex[element]= i;
       }
}

TreeNode* constructing(map<int,int>&valueToIndexMap,
vector<int>&inorder, vector<int>&postorder,
int &postOrderIndex, int inOrderStart, int inOrderEnd)
{
 if(postOrderIndex<0 || inOrderStart>inOrderEnd)
   return NULL;

 int element= postorder[postOrderIndex--];
 TreeNode* Root= new TreeNode(element);

 int position= valueToIndexMap[element];

Root->right= constructing(valueToIndexMap, inorder,
postorder, postOrderIndex, position+1, inOrderEnd);

Root->left= constructing(valueToIndexMap, inorder,
postorder, postOrderIndex, inOrderStart, position-1);

  return Root;
}
```

```
TreeNode * buildTree (vector<int> &inorder,
                      vector <int> & postorder)
{
    int postOrderIndex = postOrder.size() - 1;
    int inOrderStart = 0;
    int inOrderEnd = inorder.size() - 1;
    map<int, int> valueToIndexMap;

    createMapping (valueToIndexMap, inorder);

TreeNode* root = constructing (valueToIndexMap,
inorder, postorder, postOrderIndex, inOrderStart,
inOrderEnd);

    return root;
  }
};
```