1. Identical trees

https://leetcode.com/problems/same-tree/submissions/

Recursion
isSameTree(p,q)
if(p->val == q->val&&isSameTree(p->left,q->left)&&isSameTree(p->right,q->right))
    return true
Else
    Return false

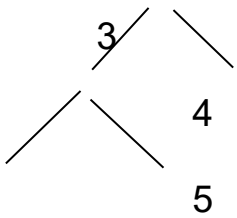
Iterative
Use queue and {push left and right pointer}
While Queue is not empty
    Pop pair and compare value
    Push {x->left, y->left}
    Push {x->right, y->right}
Return true

2. Symmetric trees

a. isSameTree(p,q)
if(p->val == q->val&&isSameTree(p->left,q->right)&&isSameTree(p->right,q->left))
    return true
Else
    Return false

b. Iterative
Use queue and {push left and right pointer}
While Queue is not empty
    Pop pair and compare value
    Push {x->left, y->right}
    Push {x->right, y->left}
Return true

3. Level Order traversal



Using queue queue : 3 Null 4 5 Null 6 7 Null
Using PreOrder :
https://leetcode.com/problems/binary-tree-level-order-traversal/

```
vector<vector<int>> ret;

void buildVector(TreeNode *root, int depth)
{
    if(root == NULL) return;
    if(ret.size() == depth)
        ret.push_back(vector<int>());

    ret[depth].push_back(root->val);
    buildVector(root->left, depth + 1);
    buildVector(root->right, depth + 1);
}
```

4. Min depth of tree

```
int minDepth(TreeNode* root) {
    if(!root)
        return 0;
    if(!root->right)
        return minDepth(root->left) + 1;
    if(!root->left)
        return minDepth(root->right) + 1;
    return min(minDepth(root->left), minDepth(root->right)) + 1;
}
```

5. Path sum

https://leetcode.com/problems/path-sum/solution/

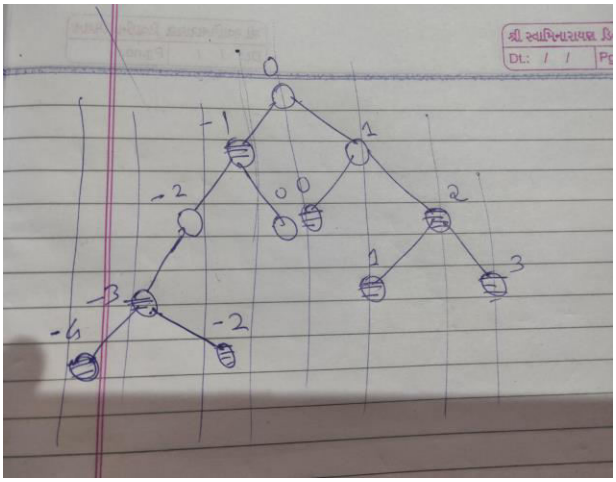
if(leaf node and sum == root->val)

    Return true

Return hasPathSum(root->left, sum - root->val) || hasPathSum(root->right, sum - root->val)

6. Bottom View of Binary Tree

Traverse using dfs or bfs and maintain vertical numer
Vertical number root = 0 , left = -1 and right -2



Do bfs or dfs and update map vertical number with node val
Traverse map in sorted order and print val

7. Vertical Order of Binary Tree

https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/
https://www.geeksforgeeks.org/print-a-binary-tree-in-vertical-order-set-3-using-level-order-traversal/
Use a queue and apply bfs and push in map as illustrate in above figure

8. Ds

```
int ans = -1;
  int findDiameter(TreeNode * root)
  {
      if(!root)
          return 0;
     int lh = findDiameter(root->left) ;
     int rh = findDiameter(root->right);
     if(lh + rh > ans)
        ans = lh + rh;
      return max(lh, rh) + 1;
  }
```

9. Zigzag order traversal

Use two stack concepts
https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/

10. Right view of Binary tree

https://leetcode.com/problems/binary-tree-right-side-view/solution/
Using DFS

i. Call right child first then left child in recursion

ii. For each depth value keep track of first node visited at that level (using map or vec.size() == depth)

iii. When vec.size() == depth it is first time we visit node at that level

Using DFS

iv. Call left child then right child in recursion or stack

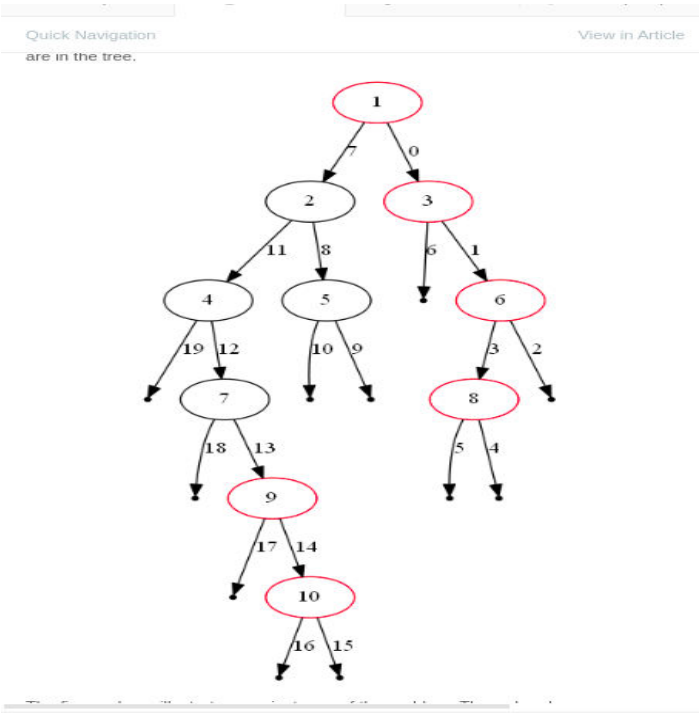
v. All nodes at depth d will be overwritten by last node

Using BFS

vi. Take map of depth and node value

vii. For each depth value keep updating node value in map

viii. All nodes at depth d will be overwritten by last node



11. Diagonal Traversal of Binary Tree

a. https://www.geeksforgeeks.org/diagonal-traversal-of-binary-tree/

b. For root label = x

left child = x - 1

Right child = x

12. Lowest common Ancestor of BST

a. https://www.geeksforgeeks.org/lowest-common-ancestor-in-a-binary-search-tree/

b.
```
if(root->val > p->val && root->val > q->val)
    return lowestCommonAncestor(root->left, p, q);

if(root->val < p->val && root->val < q->val)
    return lowestCommonAncestor(root->right, p, q);

return root;
```

13. Lowest Common Ancestor of BT

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/submissions/

```
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
    if (!root || root == p || root == q) return root;
    TreeNode* left = lowestCommonAncestor(root->left, p, q);
    TreeNode* right = lowestCommonAncestor(root->right, p, q);
    return !left ? right : !right ? left : root;
```

```
}
```

14. Flatten Binary tree to linked list

https://leetcode.com/problems/flatten-binary-tree-to-linked-list/

```
public void flatten(TreeNode root) {
    if (root == null)
        return;
    flatten(root.right);
    flatten(root.left);
    root.right = prev;
    root.left = null;
    prev = root;
}
```

15. Inorder Traversal

a. Using Morris Traversal
https://leetcode.com/problems/binary-tree-inorder-traversal/solution/

b. Using Stack

```
Stack<TreeNode> stack = new Stack<>();
    pushAllLeft(root, stack);
    while (!stack.isEmpty()) {
        TreeNode cur = stack.pop();
        res.add(cur.val);
        pushAllLeft(cur.right, stack);
    }
    return res;

public void pushAllLeft(TreeNode node, Stack stack){
    while (node != null) {
        stack.add(node);
        node = node.left;
    }
}
```

c. BST Iterator
https://leetcode.com/problems/binary-search-tree-iterator/solution/
Same logic as above
next() : returns top element
Push right child and all left diagonal child onto stack

16. PreOrder Traversal

a. Using Morris Traversal
https://leetcode.com/problems/binary-tree-preorder-traversal/submissions/

b. Using stack
st.push(root);
while(!st.empty())
{

```
        root = st.top();
        st.pop();
        ans.push_back(root->val);
          if(root->right)
           st.push(root->right);
        if(root->left)
           st.push(root->left);

      }
```

17. PostOrder Traversal

a. Using stack

```
        st.push(root);
        while(!st.empty())
        {
           TreeNode * root= st.top();
           st.pop();
           ans.push_back(root->val);
           if(root->left)
              st.push(root->left);
           if(root->right)
              st.push(root->right);
        }
        reverse(ans.begin(), ans.end());
        return ans;
```

18. Populate next right pointer

a. https://leetcode.com/problems/populating-next-right-pointers-in-each-node/submissions/

b. Using Queue

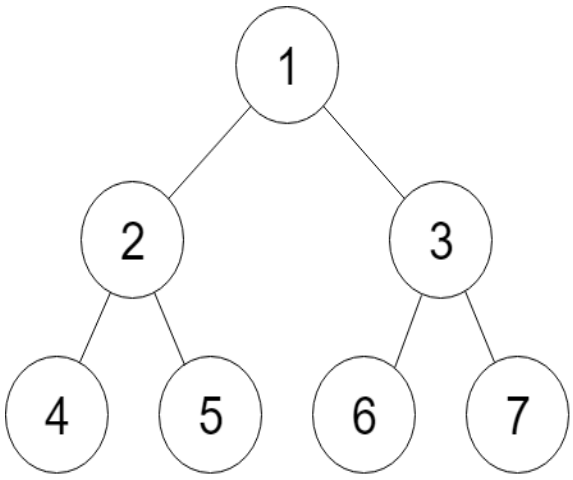


Figure A

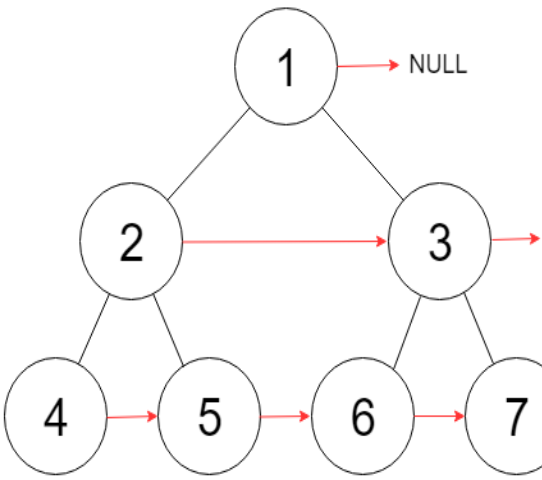


Figure B

Queue = 1 NULL 2 3 NULL 4 5 6 7 NULL

c. Using Constant Space

```
public void connect(TreeLinkNode root) {
    TreeLinkNode tempChild = new TreeLinkNode(0);
```

```
    while (root != null) {
        TreeLinkNode currentChild = tempChild;
        while (root != null) {
            if (root.left != null) {
                currentChild.next = root.left;
                currentChild = currentChild.next;
            }
            if (root.right != null) {
                currentChild.next = root.right;
                currentChild = currentChild.next;
            }
            root = root.next;
        }
        root = tempChild.next;
        tempChild.next = null;
    }
}
```

19. Balanced BST from sorted Array

```
TreeNode * makeTree(vector<int> &nums, int low, int high, TreeNode * root)
 {
    if(low > high)
       return NULL;

    int mid = (low + high)/2;
    root = new TreeNode(nums[mid]);

    root->left = makeTree(nums, low, mid-1, root);
    root->right = makeTree(nums, mid+1, high, root);

    return root;
 }
```

20. Root to Leaf Number sum

https://leetcode.com/problems/sum-root-to-leaf-numbers/submissions/

21. Construct tree from inorder and preorder

a. https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/
b. Use hashmap to store indices of element and index mapping of inorder
   S1 e1 = inorder s2 e2 preorder
   root->left = makeTree(root, inorder, s1, i-1, preorder, s2+1, s2+i-s1);
   root->right = makeTree(root, inorder, i+1, e1, preorder, s2+i-s1+1, e2);

22. Populate Inorder Successor for all nodes

https://www.geeksforgeeks.org/populate-inorder-successor-for-all-nodes/

```
while(temp){
    s.push(temp);
    temp=temp->left;
}

while(!s.empty()){
    auto x = s.top();
    s.pop();
    temp=x->right;
    while(temp){
        s.push(temp);
        temp=temp->left;
    }
    if(s.empty()) x->next=NULL;
    else x->next = s.top();
}
```

23. Reverse tree path

store the path in stack till data reached (stack of TreeNode *)
Store stack into vector a and reverse vector b
Now store value of b into a for each element

24. https://www.geeksforgeeks.org/perfect-binary-tree-specific-level-order-traversal/

first->left->data second->right->data
first->right->data second->left->Data

25. https://www.geeksforgeeks.org/level-order-traversal-direction-change-every-two-levels/

while queue is not empty
increment level++
    For i =0 to q.size()
        if bool var = false
            print pop value
        else push into stack
if(var == true) pop stack and print
if(level ==2) change direction

26. https://www.geeksforgeeks.org/reverse-alternate-levels-binary-tree/

a. User 24. 2 pointer approach
b. Use queue and do level order traversal
   Pop to first and second

If fg true swap first and seconds pointers values

---

1. https://leetcode.com/problems/two-sum-iv-input-is-a-bst/
    a. BST iterator and reverse iterator
    b. https://leetcode.com/problems/two-sum-iv-input-is-a-bst/discuss/106063/C%2B%2B-Clean-Code-O(n)-time-O(lg-n)-space-BinaryTree-Iterator

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

3. Find Duplicate Subtree
   https://leetcode.com/problems/find-duplicate-subtrees/

4. Serialization and Deserialization of Tree
   https://leetcode.com/problems/serialize-and-deserialize-binary-tree/

5. Binary Tree Camera
   https://leetcode.com/problems/binary-tree-cameras/