## 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

## Symmetric trees

1. isSameTree(p,q)

if(p->val == q->val&&isSameTree(p->left,q->right)&&isSameTree(p->right,q->left))

return true

Else

Return false

1. 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

## 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);

}

## 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;

}

## Path sum

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

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

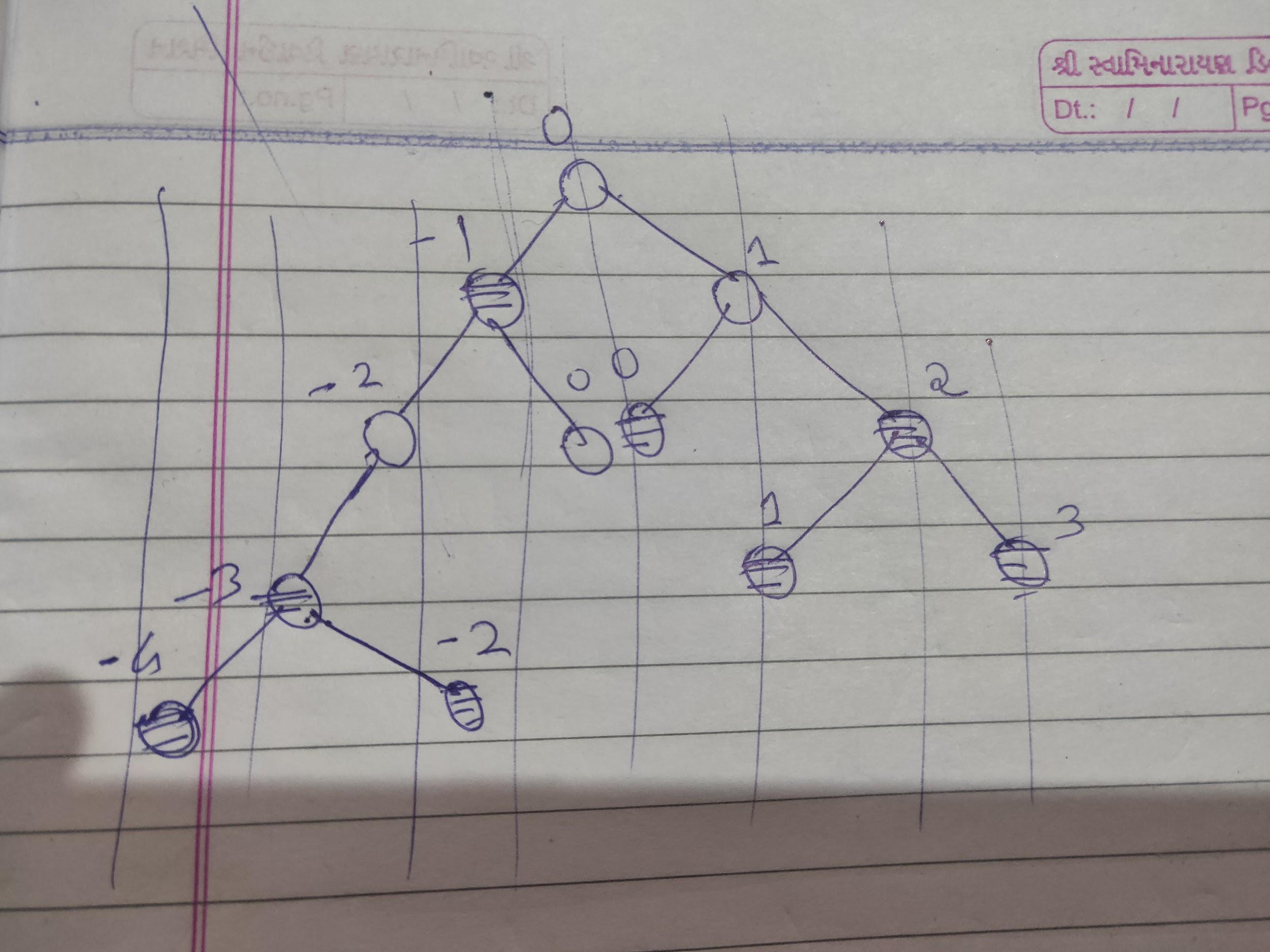
Return true

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

## 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

## 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

## 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;

}

## Zigzag order traversal

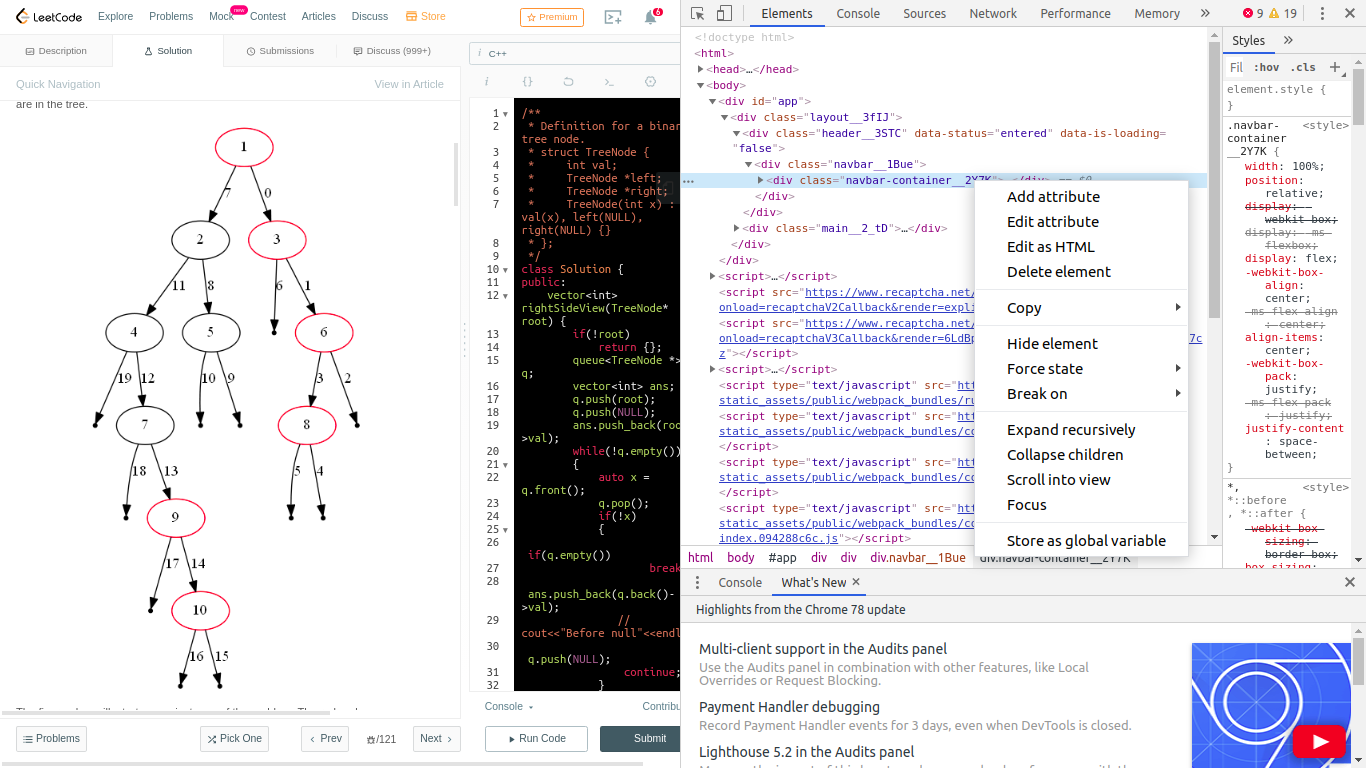
Use two stack concepts

<https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/>

## Right view of Binary tree

<https://leetcode.com/problems/binary-tree-right-side-view/solution/>

Using DFS

* 1. Call right child first then left child in recursion
  2. For each depth value keep track of first node visited at that level (using map or vec.size() == depth)
  3. When vec.size() == depth it is first time we visit node at that level

Using DFS

* 1. Call left child then right child in recursion or stack
  2. All nodes at depth d will be overwritten by last node

Using BFS

* 1. Take map of depth and node value
  2. For each depth value keep updating node value in map
  3. All nodes at depth d will be overwritten by last node

## Diagonal Traversal of Binary Tree

* 1. <https://www.geeksforgeeks.org/diagonal-traversal-of-binary-tree/>
  2. For root label = x

left child = x - 1

Right child = x

## Lowest common Ancestor of BST

* 1. <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;

## 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;

}

## 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;

}

## Inorder Traversal

* 1. Using Morris Traversal

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

* 1. 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;

}

}

* 1. 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

## PreOrder Traversal

* 1. Using Morris Traversal

<https://leetcode.com/problems/binary-tree-preorder-traversal/submissions/>

* 1. 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);

}

## PostOrder Traversal

* 1. 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);

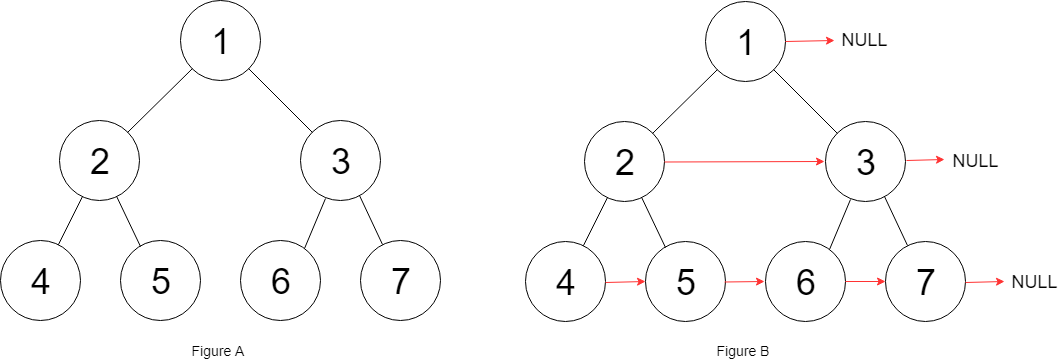
}

reverse(ans.begin(), ans.end());

return ans;

## Populate next right pointer

* 1. <https://leetcode.com/problems/populating-next-right-pointers-in-each-node/submissions/>
  2. Using Queue



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

* 1. 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;

}

}

## 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;

}

## Root to Leaf Number sum

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

## Construct tree from inorder and preorder

* 1. <https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/>
  2. 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);

## 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();

}

## 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

## <https://www.geeksforgeeks.org/perfect-binary-tree-specific-level-order-traversal/>

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

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

## <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

## <https://www.geeksforgeeks.org/reverse-alternate-levels-binary-tree/>

* 1. User 24. 2 pointer approach
  2. 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/>
   1. BST iterator and reverse iterator
   2. <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/>

1. Serialization and Deserialization of Tree

<https://leetcode.com/problems/serialize-and-deserialize-binary-tree/>

1. Binary Tree Camera

<https://leetcode.com/problems/binary-tree-cameras/>