Must Do **Binary Tree** problems for **Placement**

Binary Trees	**level order traversal**
Binary Trees	Reverse Level Order traversal
Binary Trees	Height of a tree
Binary Trees	Diameter of a tree
Binary Trees	Mirror of a tree
Binary Trees	Inorder Traversal of a tree both using recursion and Iteration
Binary Trees	Preorder Traversal of a tree both using recursion and Iteration
Binary Trees	Postorder Traversal of a tree both using recursion and Iteration
Binary Trees	Left View of a tree
Binary Trees	Right View of Tree
Binary Trees	Top View of a tree
Binary Trees	Bottom View of a tree
Binary Trees	Zig-Zag traversal of a binary tree
Binary Trees	Check if a tree is balanced or not
Binary Trees	Diagnol Traversal of a Binary tree
Binary Trees	Boundary traversal of a Binary tree
Binary Trees	Construct Binary Tree from String with Bracket Representation
Binary Trees	Convert Binary tree into Doubly Linked List
Binary Trees	Convert Binary tree into Sum tree
Binary Trees	Construct Binary tree from Inorder and preorder traversal
Binary Trees	Find minimum swaps required to convert a Binary tree into BST
Binary Trees	Check if Binary tree is Sum tree or not
Binary Trees	Check if all leaf nodes are at same level or not
Binary Trees	Check if a Binary Tree contains duplicate subtrees of size 2 or more [IMP]
Binary Trees	Check if 2 trees are mirror or not
Binary Trees	Sum of Nodes on the Longest path from root to leaf node

Binary Trees	Check if given graph is tree or not. [IMP]
Binary Trees	Find Largest subtree sum in a tree
Binary Trees	Maximum Sum of nodes in Binary tree such that no two are adjacent
Binary Trees	Print all "K" Sum paths in a Binary tree
Binary Trees	Find LCA in a Binary tree
Binary Trees	Find distance between 2 nodes in a Binary tree
Binary Trees	Kth Ancestor of node in a Binary tree
Binary Trees	Find all Duplicate subtrees in a Binary tree [IMP]
Binary Trees	Tree Isomorphism Problem

Ques → First non-repeating character in a Stream

Given input stream of 'A' on n lower case alphabet. Task is to find non-repeating char., each time a char. is inserted to the stream. if there is no such char then append '#'.

* Example :-

Stream

| a a b C

a a b c

ans = a # b b . // ∵ 1st non repeating ele.

* Intution :-

→ ∵ we have to maintain count of each alphabet we'll use a 'counter' variable.

→ Now, here we've to maintain freq. as well as its order of alphabet

→ so we'll tak a vector. to maint ain order. Or map.

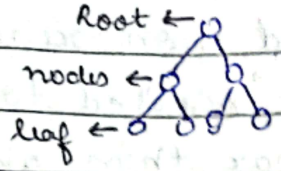
* Implementation :-

```
string NonRepeat (string a)
int n = s.size();   unordered_map<char, int> map;
string ans = "";    queue<char> q;
    for (int i=0; i<n; i++)
    if (!map[s[i]]) q.push(s[i]);
  map[s[i]]++;
while(!q.empty() && map[q.front()] > 1) q.pop();
    if (q.empty()) ans += "#";
    else ans += q.front();
return ans;
```

9/3/22

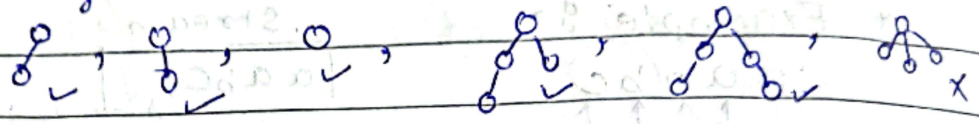
Binary Tree

* Tree :-

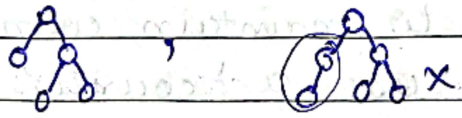


we don't have loop / cycle in tree.
structure : Root, Internal node, leaf.

→ Binary tree : Atmost 2 children (0, 1, 2)



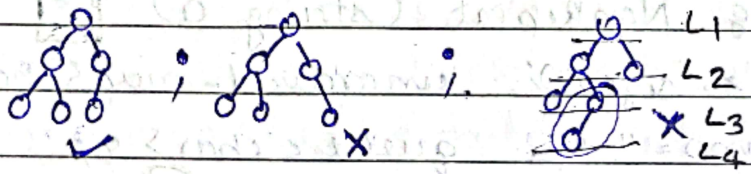
→ Full / strict Binary tree :-
either 0 or 2 children.



→ ACBT (Almost comp. BT) :

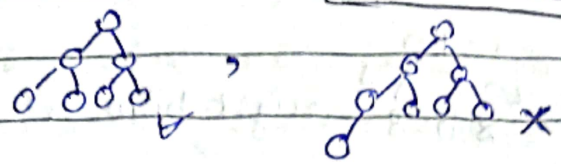
 → not ACBT,

 → Yes, Bcoz alway left after right then goto next level.



→ perfect / Complete Binary Tree :-

NO holes



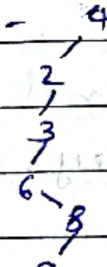
(it is ACBT, but not CBT)

→ Binary Search tree :-

⊙ create BST, with following keys.

4, 2, 3, 6, 5, 7, 1

in BST, there is no systematic order,

eg:- 

, after storing the data, we have to search for particular data afterward

And, if we don't put data in systematic order, then we'll definately face complexity in searching particular data.

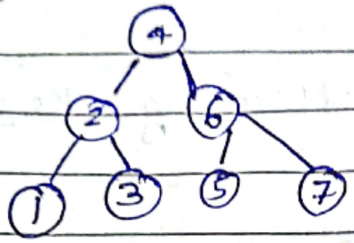
average TC - O(n).

Now, BST says, while entering data left child should be less than root, and right child should be greater than root.

* If in case keys are equal, then you can put either side.

eg - 10 (≤ left bias, ≥ right bias) 8 10 8 — 10 8 10

Sol"



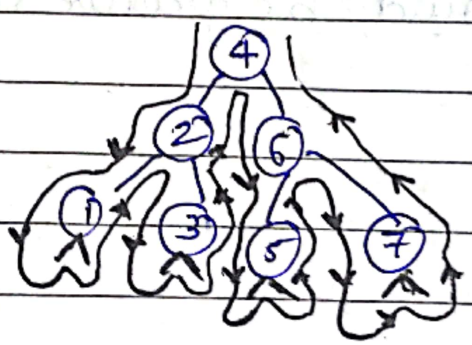
(∵ 2 < 3 < 4)

Binary search tree.

→ Least element in the end of left side will be smallest key.

→ Least element at the end of right side will be biggest element in key.

→ Inorder : always give sorted array.
make 2 child of each node (dummy)
check; if we reach any element 2nd/more time, is its inorder.



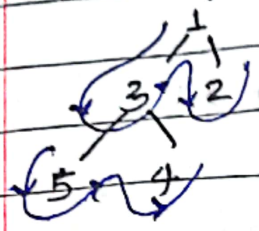
1, 2, 3, 4, 5, 6, 7

(Binary Tree)

Ques → Level Order traversal

Given BT, find its level order traversal.
LOT → nothing but breadth-first traval of tree.

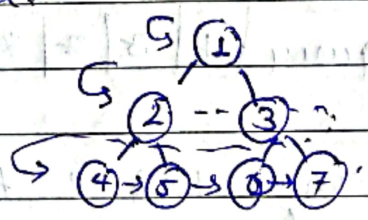
Example :-



Ans - 1, 3, 2, 5, 4.

* Intution :-

1) Solve it using queue.
2) store elemenent in queue, and push back queue element in a vector 'ans'.
3) check if element's left node exist, if yes push in queue, and pop root.
4) same with right node.
5) Repeat process for entire tree, for every level.



queue - | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
pop

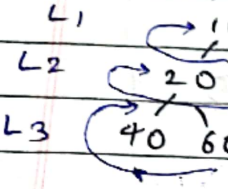
Ans = 1, 2, 3, 4, 5, 6, 7

at the end queue = empty,
Break the loop.

Ques → Reverse Level Order Traversal

Given binary tree of size N, find its Reverse level order traversal.
(traverse must begin from last level)

Example :-



40, 60, 20, 30, 10

Level => L3 → L2 → L1

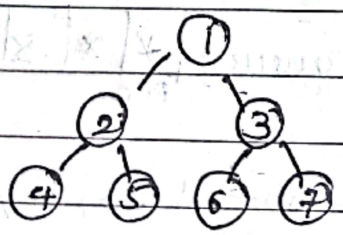
(*) Intution :-

→ Maintain queue and ans vector.
firstly enter root, then check if right exist, push in queue and check left if exist push in queue.

→ pop, root and push back in the vector.

→ do the same for every element in respective level.

(when we reverse ans we'll get the o/p)



queue | 1 | 3 | 2 | 7 | 6 | 5 | 4 |

ans - 1 3 2 7 6 5 4

↓ Reverse

Ans => 4 5 6 7 2 3 1

Ques→ Height of Binary Tree

Given BT, find its height.

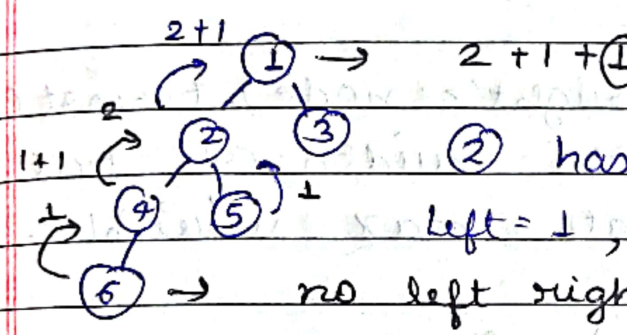
Example :-

L1 ① i) Ans = 2

L2 ② ③ (ii) (max. depth = height).

* Intution :- (Recursion

→ We'll check for height in both left and right side, max. will be returned.



② has 2 & 1 ⇒ 2 max

Left = 1, righ = 0, left max.

→ no left right, retur 1 to 4.

Ans = 4

* Implementation :- int height (struct Node* node)

{ if (node == NULL) return 0;

int left = height (node → left);

int right = height (node → right);

int height;

if (left > right)

height = 1 + left;

else

height = 1 + right;

return height;

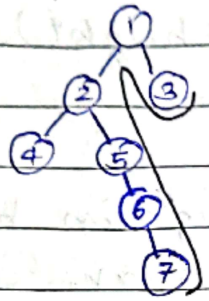
}

10/3/22

Ques- Diameter of binary tree.

Diameter/width is no. of nodes on the longest path b/w two end nodes.

⊛ Example :-



ans = 6 . (longest width)

⊛ Intution :-

(left's nod + right's node) + root(1) will give the width of tree and update max variable.

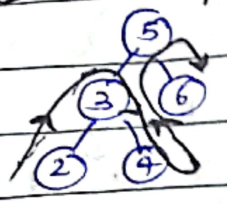
⊛ Implementation :-

```
int ma;
int func(Node * root) {
    if (!root) return 0;
    int x = func(root -> left);
    int y = func(root -> right);
ma = max(ma, x+y+1);
return (max(x,y)+1);
}

int diameter(Node * root)
{
    ma = INT_MIN;
int x = func(root)
return ma;
}
```

Ques → Create a mirror tree from the given binary tree

Given a BT, create a new binary tree which is a mirror image of given BT.

(*) Example :-



Inorder of original T : 2 3 4 5 6
Inorder of Mirror T : 6 5 4 3 2

(*) Intution :-

In order to change original tree in its mirror tree, then we simply swap the left and right link of each node.

If the node is leaf node then do nothing.

(*) Code :-

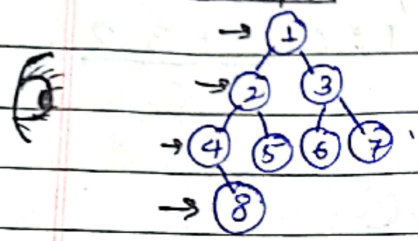
```
treenode * mirrorTree (node *root)
{
    if (root == NULL)
        return root;
node * t = root → left;   root → left = root → right;
root → right = t;

if (root → left) mirrorTree (root → left);

if (root → right) mirrorTree (root → right);

return root;
}
```

Date 11/3/22

Ques- Left view of Binary Tree

Given BT, print left view of tree.
Left view : node visible, when tree visited from left side.

⊛ Example :-



ans → 1, 2, 4, 8
∴, 8 is right but no right exist

Intution :-

→ take vector 'ans' :- put root a default value in the ans.
→ Do the level order traversal, at level 1 enter the 1st element and change the level.
→ Do the above step for entire tree.

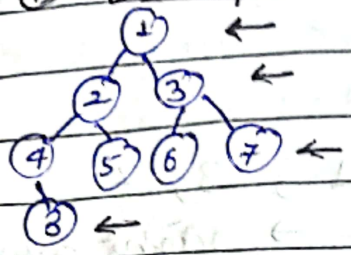
* Implementation :-

```
left view (Node* root) {
  vector<int> ans; queue<node*> q;
  if (!root) return ans;
    q.push(root);
  while (!q.empty()) {        int sz = q.size();
    ans.push_back(q.front()->data);
      while(sz--) {           node* t = q.front();
        q.pop();
  if (t->left) q.push(t->left);
  if (t->right) q.push(t->left }
  }
    return ans;
}
```

Ques → Right view of tree

Given BT, find Right view of tree.

(*) Example :-



ans = 1, 3, 7, 8

(*) Intution :-

→ take vector ans, and push root into it.

→ then traverse level by level and push the last element of that level.

→ do the above process for entire tree

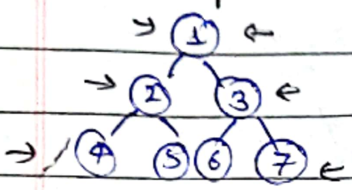
(*) Implement :-

```
void RightView(Node* root)
{
    queue<node*> q;
    if(!root) return;
    q.push(root);
    while(!q.empty())
    {
        int sz = q.size();
        Node* t;
        while(sz--)
        {
            t = q.front();
            q.pop();
            if(t->left) q.push(t->left);
            if(t->right) q.push(t->right);
        }
        cout << t->data << " ";
    }
}
```

Ques- Top View of tree

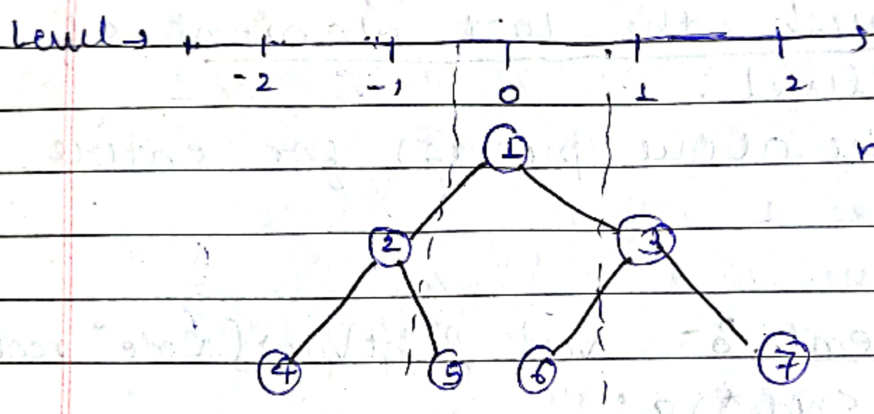
Given BT, find its top view.
Top view = left mos + right most + root.

Example :-

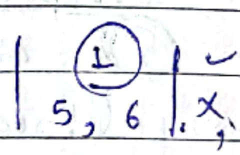


4, 2, 1, 3, 7
(leftmost → right most)

Intution :-



map(~~height (x)~~, ~~value~~) ↗ vertical
map(value, heigh)



at level 0, element on
∴, 5, 6 covered & not visible so ans = 1.

map :- (int, t)

0 → ~~0~~ 1
1 → ~~0~~ 3
2 → ~~0~~ 7
-1 → ~~0~~ 2
2 → ~~0~~ 4

queue (node* h) NULL
| ~~(1, 0)~~ ~~(2, -1)~~ ~~(4, -2)~~ ~~(3, 1)~~ ~~(7, 2)~~

Ans => 1, 3, 7, 2 4,

print in level order -2 → -1 → 0 → 1 → 2
=> 4, 2, 1, 3, 7

* Implement :-

```
struct Void TopView (node * root)
{
    map < int, int > m;
    queue < pair < Node*, int > > q;

    if (! root) return;
    q.push ( { root, 0 } );
while ( ! q.empty ()) {
    Node * t = q.front () . first;
int h = q.front () . second;
    q.pop ();
if ( ! m[h] ) m[h] = t -> data;
    if ( t -> left ) q.push ( { t -> left, h-1 } );
if ( t -> right ) q.push ( { t -> right, h+1 } );
}
for (auto x : m) cout << x.second << " ";
}
```

TC- $O(N) \times \log n \Rightarrow n \log n$.

SC- $O(N)$ // extra space of queue

- Implement :-

```
start  Void TopView (node * root)
(1)    map < int, int > m ;
       queue < pair < Node*, int >> q ;

       if (! root)  return ;
          q . push ( { root, 0 } ) ;
while ( ! q . empty ( ) ) {2}
       Node * t = q . front ( ) . first ;
int    h = q . front ( ) . second ;
          q . pop ( ) ;
if ( ! m [ h ] )  m [ h ] = t → data ;
       if ( t → left )  q . push ( { t → left, h - 1 } ) ;
if ( t → right )  q . push ( { t → right, h + 1 } ) ;

[3]

for ( auto  x : m )  cout << x . second << " " ;

(3)
```

TC- $O(N) \times \log n \Rightarrow n \log n$.

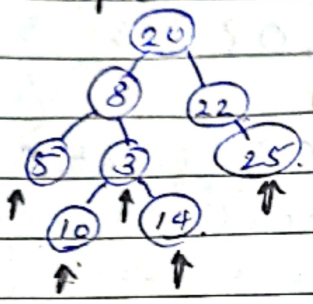
SC- $O(N)$ // extra space of queue

12/3/22

Ques→ Bottom View of Binary Tree

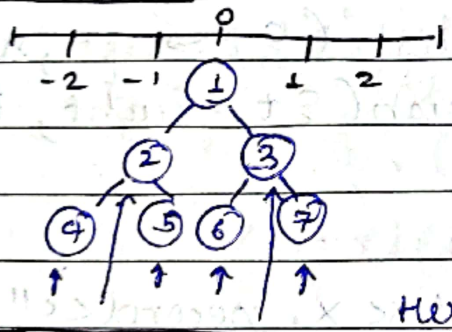
Given BT, print bottom view from left to right.
A node is included in bottom view if it can be seen when we look at tree from bottom.

* Example :-



5, 10, 3, 14, 25

* Intution :-



Here 2 is visible b/w 4 & 5
3 is visible b/w 6 & 7

Here, 5 & 6 both at 0 Level, the print which comes later.

Ans→ 4, 2, 6, 3, 7

map<val, h>, queue<node* h> required

Pick each and every element, and put into map and the element which come in the end or last will be the answere.

queue

map :- (1,0) (2,-1) (3,1) (4,-2), (5,0) (6,0) (7,2)

update update

0 → 0 x 6

1 → 0 3

2 → 7

-1 → 0 2

-2 → 0 4

when queue end, break.

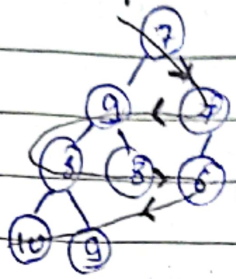
and our ans. will be stored in the map.

Ans : 4, 2, 0, 3, 7.

Ques→ Zig zag traversal of binary tree

Given a BT. find zig-zag level order traversal of BT.

⊛ Example :-

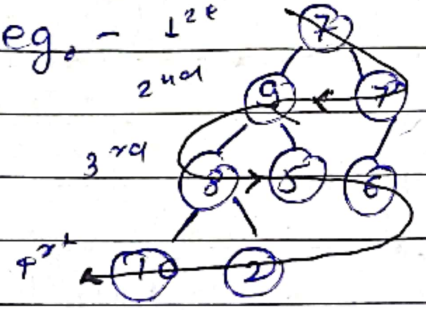


Ans : 7, 7, 9, 8, 8, 6, 9, 10

Intution :-

Logic :- take root as 1st element, do the level order traversal, and print 2nd alternative level reversely.

eg. -



→ here, 2nd & 4th will print in reverse (R) (even)

→ 1st & 3rd will print in forward (F) direction (odd)

Ans - 7, 7, 9, 8, 5, 6, 2, 10

F(odd) R R(even) R

we'll use vector to print all the all the element of that particular level and if level become odd or even, we'll print accordingly.

Code :-

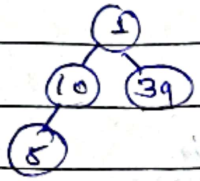
```
vector <int> zigZag(Node* root) {
    vector <int> ans;
    queue <node*> q;
    q.push(root);
    int f=1; // to check for odd even
    while(!q.empty()) {
        vector <int> temp;
        int sz = q.size();
        while(sz--) {
            Node* t = q.front();
            temp.push_back(t->data);
            q.pop();
            if(t->left) q.push(t->left);
            if(t->right) q.push(t->right);
        }
        if(f%2==0)
            reverse(temp.begin(), temp.end());
        for(int i=0; i<temp.size(); i++)
            ans.push_back(temp[i]);
        f = !f;
    }
    return ans;
}
```

Ques- Check if tree is balanced or not

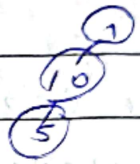
Given BT, find if it is height balanced or not.

Balanced : if height diff- b/w left and right subtree is not more than one for all nodes.

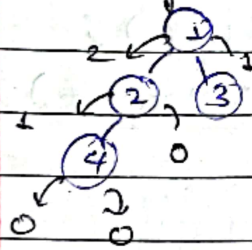
Balanced



Unbalanced



Example :-



Ans - True.

(*) Intution :- (using Recursion)

→ we have to find the height of every node

→ And the check if diffrunce b/w left & right is ≤ 1.

→ then true else false & break

Implementation :-

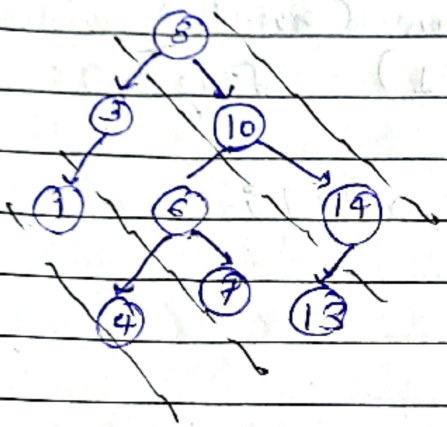
```
int flag = 1;
int solve( node * root ) {
    if ( !root ) return 0;
    int l = solve (root → left);
    int r = solve (root → right);
    if ( abs(l-r) > 1 )  flag = 0;

    return max (l, r) + 1;
```

Date 13/3/22 Page no.

Ques→ Digonal Traversal of BT

Consider lines of slope -1, passing b/w nodes. Given BT, print all elements of bT belong to same line.

Example :-



Ans → 8, 10, 14
3, 6, 7, 13
1, 4.

* Intution :-

→ The idea is use map.
→ we use diff. slope dist, and use it as key of map.
→ Value will be store in map as vector
→ Traverse tree to store values in the map. Once map built, then print its content.

TC - N log n
SC - O(N)

Implementation :-

```
void digonalPrint (Node* root).
{
    map<int, vector<int>> diag;
    diagprintUntil(root, 0, digonalPrint);
    cout << " digonal Transal of BT";
for(auto it : diagonalPrint)
    {
        vector<int> v = it.second;
    for(auto it : v)
        {
            cout << " ";
            cout << endl;
        }
    }
```

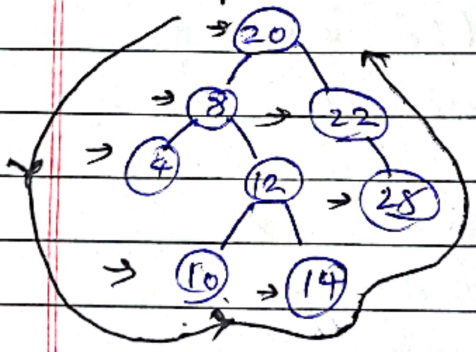
Date: / / Page no:

Ques → Boundry Traversal of BT

Given BT, find its boundry traversal.

- Left boundry node: path from root to left most node. i.e. leaf node you could reach, when you always travel prefering left subtree over the right.
- Leaf node: all leaf node, except for the one who is part of left or right Boundry.
- Reverse right boundry node:- path from right most node of the root. Right most node is leaf node you could reach when you always travel prefering right subtree over the left.

Example:-



Ans = 20, 8, 4, 10, 14, 25, 22.

(*) Intuition :-

(1) we'll break problem in 3 parts.

1) print left-boundry in top down manner.

2) print all leaf node from L to R, which can be sub-divided.

2.1) print all leaf node of left sub tree from L to R

2.2) " " " " Right sub tree " " "

3) Print the right boundry in bottom up manner.

NOTE- We take care 1 thing that node are not printing again.

eg.- left most node is also leaf node of tree.

Date 14/3/22 Page no.

Ques → Construct BT from string with bracket representation

Construct bt from a string consisting of parenthesis and integers. The whole input represent. The whole input represent it contain an integer followed by zero, one or two pair of parenthesis.

Integer represents root's value & a pair of parenthesis contains a child binary tree with same structure.

⊛ Example :-

I/P : 1(2)(3) O/P : 1 2 3.

→ 1st pair of parenthesis contains left subtree and second one contains the right subtree.

(preorder → 1 2 3).

4(2(3)(1))(6(5))

Root Left subtree Right subtree

We need to find the substring corresponding to left substring & substring " to right subtree, and then call recursively call on both the substring.

(*) Recursive Approach

(1) The very first element is of string is root.

(2) The next two consecutive elements are "(" & ")", this means their is no left child otherwise we will create and add the left child to the parent node recursively.

(3) Once the left child added recursively we will look for consecutive "(" and add the right child to parent node.

(4) Encountering ")" means the end of either left or right node and we will use start index.

(5) The recursion end when the start index is greater than equal to the end index.

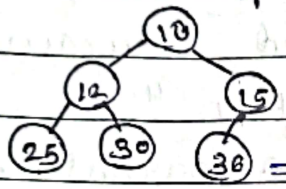
```
* Preorder (node* root)
{
    if (root == NULL)
    cout << root -> data << " ";
    preorder ( root -> left);
    preorder ( root -> Right);
}
```

Ques→ Binary Tree to DLL

Given BT, convert it to DLL in place. left and right pointers in node are to be used as previous and next pointer respectively in converted DLL.

Order of node in DLL is same as inorder of given BT.

Example :-



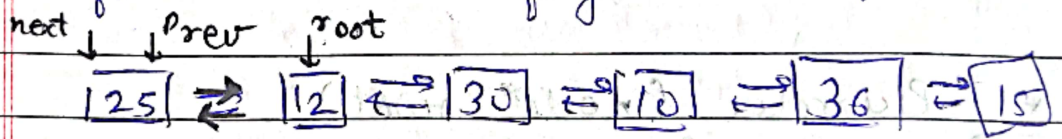
The above tree should be in-place convert to dll,

⇒ 25 ⇄ 12 ⇄ 30 ⇄ 10 ⇄ 36 ⇄ 15

(*) Intution :-

→ Right ptr = next ptr. and left ptr = prev. ptr

→ firstly (head == NULL) ; and taking forward with a flag variable.



① now, at the next node, (prev → ~~next~~ right = root)

② then, prev → right → left = prev.

③ then, prev = prev → right

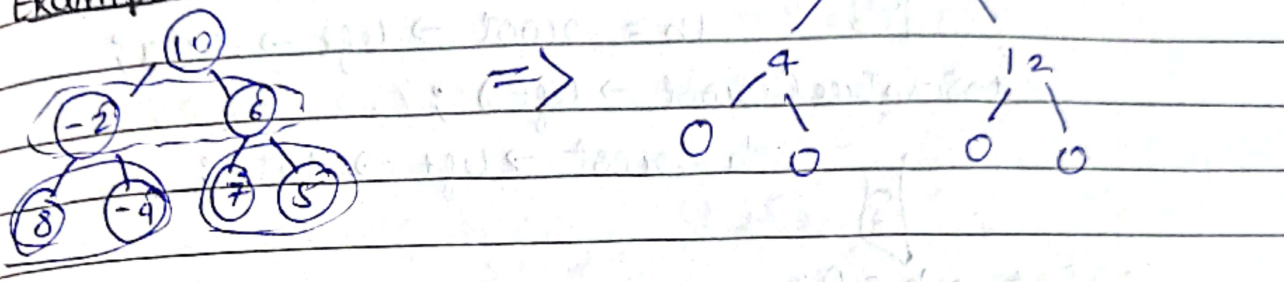
→ Repeat this step for entire inorder traversal

15/3/22

Ques - Transform to Sum Tree

Given BT of size N, where each node can have +ve or -ve values. Convert this to tree where each node contain the sum of left and right sub tree of the original tree.

Example :-



⊛ Intution :-

Traverse given tree. In traversal store old value of current node, recursively call for left and right subtree and change the value of current node as sum of values returned by recursive call.

In the end, return sum of new value which is in subtree rooted with this node.

Implementation :-

```
int ToSumTree (node * Node)
{
    if (node == NULL)
        return 0;
    int lb = 0;
    int la = 0;
    if (root -> left)
    {
        lb = root -> left -> data;
    }
    toSumTree (root -> left);
    {
        la = root -> left -> data;
    }
    int rb = 0;
    int ra = 0;
    if (root -> right)
    {
        rb = root -> right -> data;
        toSumTree (root -> right);
        ra = root -> right -> data;
    }
    root -> data = ra + rb + la + lb;
    return;
}
```

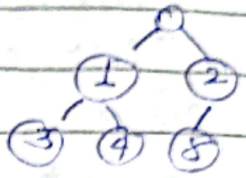
Ques → Construct tree from Inorder & preorder.

Given 2 array of inorder & preorder traversal construct tree and print postorder traversal

* Example :-

Inorder - 3, 1, 4, 0, 5, 2

preorder - 0, 1, 3, 4, 2, 5



Intuition :-

→ Traverse in preorder, make ~~node~~ element as a node.

→ Search node in inorder, ~~once~~ once if present in ~~inord~~ inorder,

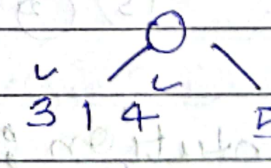
→ all the element present in left, ~~m~~ send to the left side of tree.

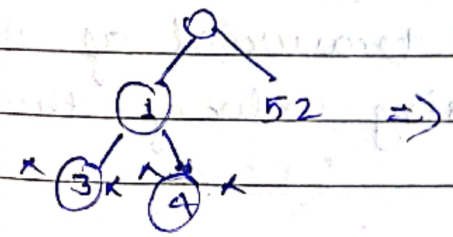
→ and all the element present in right of node send to right side of tree.

② Dry Run :-

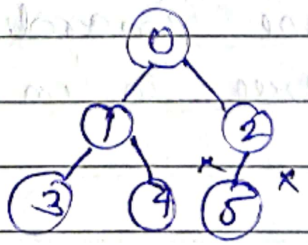
3 1 4 0 5 2 - In

0 1 3 4 2 5 - pre

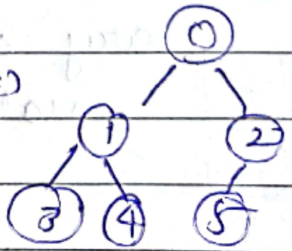




=>



=>



Date 16/3/22 Page no:

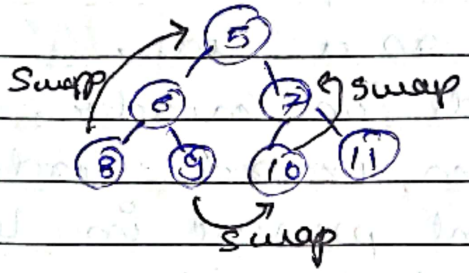
Ques- Find min. swap require to convert BT to BST.

Given the array, represents complete BT, i.e. if index i is parent, index 2*i+1 is left child and 2*i+2 is right child. find min no. of swap require to convert into BST.

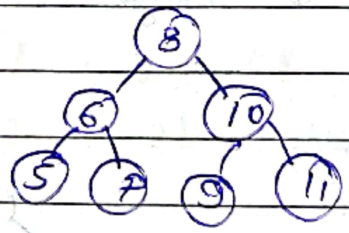
Example :-

Input : arr[] = {5, 6, 7, 8, 9, 10, 11}

O/P = 3.



Swap 8 to 5, 9 to 10, and 10 to 7.



∴ 3 swap required

→ Intution :-

we use inorder traversal of Binary search tree is in rising order of their values.

We find inorder traversal of BT and store it in array.
They sort the array.
Then, min. no. of swap required to get array sorted will be answere.

Implementation :-

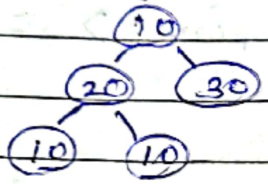
```
int minSwap (std :: vector<int>& v)
{
    std :: vector < pair<int, int> > t(v.size());
int ans = 0;
for (i=0 → i<v.size)
t[i].first = v[i], t[i].second = i;

sort ( t.begin(), t.end() );
for(int i=0; i< t.size(); i++)
{
    if ( i== t[i].second)
        continue;
else
{
swap (t[i].first, t[t[i].second].first);
swap (t[i].second, t[t[i].second].second);
}
    if (i != t[i].second)
        --i;
ans++;
}
return ans;
}
```

Date: / / Page no:

Ques → Check if BT is sum tree or not.

Given a BT, return true if, for every node X in the tree other than the leaves, its value is equal to the sum of its left subtree's value and its right subtree's value. else return false.

Example :-



o/p = false

for root node sum of elem. in left subtree is 40. and sum of ele. in right sub tree is 30.
Root element = 10, i.e not = 30 + 40.

Intuition :-

1. Same as postorder traversal, iteratively find the sum in each step.
2. Return left + right + current data, if left + right = current node data
3. else return -1.

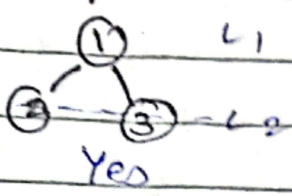
Implementation

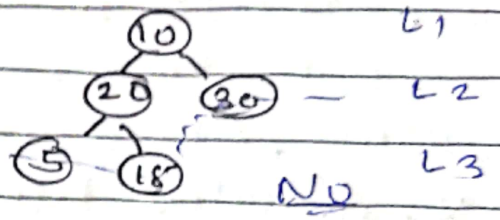
```
bool isSumTree(node * node)
{
    if (!root)
        return true;
}
if (!root->left && !root->right)
{
    return true;
}
int count = 1;
help(root, count);
if (count == 1)
{
    return true;
}
else
    return false;
}
```

Ques → Leaf at same level

Given a BT, check if all leaves are at same level or not.

Example :-





Intution :-

(1) The idea is iteratively traverse tree, & when encounter 1st leaf node, store its level in result variable.

(2) Now whenever you encounter any leaf node, compare its level with prev. stored result, they are the same then proceed for rest of the tree, else return false

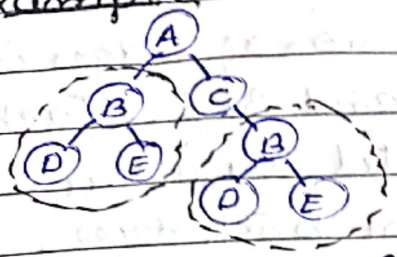
Snippet :-

```
bool check(Node* node)
{
if (!root) return true;
if (!root->left && !root->right)
return true;
vector<int> v;
levelLeaf(root, v, 0);
for(auto x : v)
    if (x != v[0])
        return false;
return true;
}
```

Ques → Duplicate Subtree in Binary Tree

Given BT, check if BT contain any sub tree of size 2 or more.

Example :-



Yes

Intution :- (Serialization & Hashing)

Firstly we serialize the subtree as string.

Store the string in hash table.

Once we find a serialized tree (which is not a leaf) already existing in hash-table, we return true

(#) Implement :- string Duplicate (node * root)

(2) string s = ""; if (root == NULL) return s+M;

string lstr = duplicate (root → left);

left → if (lstr.compare (s) == 0)

return s;

right → string rstr = duplicate (root → right);

if (rstr.compare (s) == 0) return s;

s = s + root → key + lstr + rstr;

subtree insert (s);

return s;

}

(3)

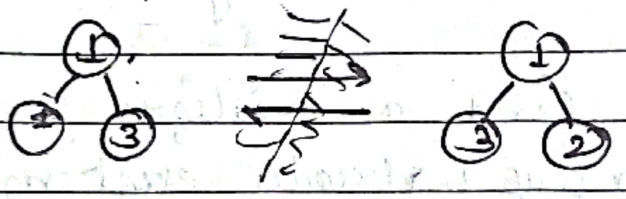
Ques→ Check if 2 tree are mirror or not

Given 2 n-array trees. Check if they are mirroring of each other or not, also given e denote no. of edges in both tree, & 2 array A[] and B[]. Each array has 2*e space seperated values. u, v denoting edge from u to v from both tree.

⊛ Example :-

n=3, e=2

A[]= {1,2,1,3} B[]= {1,3,1,2} => O/P = 1 (true)



⊛ Intution :-

The main approch is to use list of stack and one list of queue to store to value of node given in form of 2 array.

(8) Approach (algo) :-

→ Init both lists with empty stack and empty queue.

→ Now, iterate over list, push all connected node of each node first tree in list of stack and second tree list of queue.

→ Now iterate over the array & pop item from both stack & queue & check if they are same, if not the return 0.

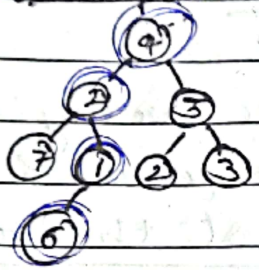
Date 28/3/22 Page no:

Ques:- Sum of longest bloodline of tree

Given BT, find sum of all nodes on the longest path from root of leaf node.

If 2 or more paths complete for longest path, then the path having max. sum of node is being considered.

Example :-



Highest node => 4, 2, 1, 6.

Sum = 4 + 6 + 6 + 1 = 13

O/P => 13

Intution :- (We use BFS)

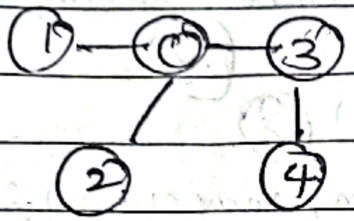
1. Create structure containing current node, level & sum in the path.
2. push root element with level 0 & sum as root's data
3. Pop front data/element & update max. level sum & max. level if needed.
4. Push left & right node if exist
5. Do same for all node in tree

Implement :-

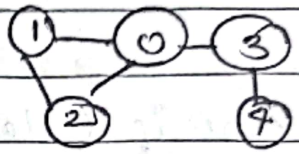
```
void solve (node* root, int len, int& maxlen,
int sum, int& maxsum)
    if (root == NULL)
        if (len > maxlen)
            maxlen = len;
            maxsum = max (sum, maxsum);
    else if (len == maxlen)
        maxsum = max (sum, maxsum);

sum = sum + root -> data;
solve (root -> left, len+1, maxlen, sum, maxsum);
solve (root -> right, len+1, maxlen, sum, maxsum);
```

Ques → Check if given graph is tree or not

Write func" that returns true if given undirected graph is tree and false otherwise.
An undirected graph is tree has :-
(1) no cycle (2) graph is connected
Example :-



Yes



NO

(*) Intution :-

How to detect cycle ?

we can use either BFS or DFS, for every visited vertex 'v', if there is and adjacent 'u' such that 'u' is already visited and u is not parent of v, then their is cycle in graph..

How to check connectivity ?

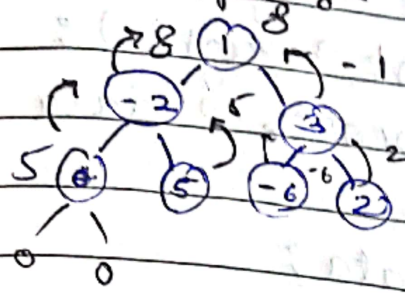
:- the graph is undirected, we can star BFS or DFS from any vertex. and check if all vertices are reachable or not. if all vertices are reachable, then graph is connected..

19/3/22

Ques → Find largest subtree sum in tree

Given BT, task is find subtree with maximum sum in tree.

Example :-



(5) => 0 + 0 + 5 => 5

max = ∅ & 8

node (2) = 5 + 5 + (-2) = 8

node (3) = -6 + 2 + 3 = -1

node (1) => 8 + (-1) + 1 = 8

return max; (Ans = 8)

(*) Intution :-

(1) Take left data, take right data.

(2) maintain 'max' variable.

(3) now add all 3 data i.e Sum = left + right + curr_data.

(4) if sum > max, update max else return max in end.

Date: / / Page no:

(*) Intution algo :-

```
int funcn (node * root)
{   if (!root) return 0;
int l = funcn (root → left);
int r = funcn (root → right);
ma = max (ma, 1 + r + root → data);

return l + root → data;
}
```

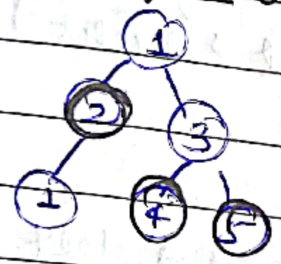
TC - O(n)

SC - O(height)

Ques → Maximum sum of nodes in BT such that no two are adjacent.

Given BT, with node associated values, we need to choose subset of these nodes such that sum of chosen node is max under a constraint that no two choose nodes in subset should be directly connected that is, if we've taken node in our sum then we can't take any of its children in consider recursion.

* Example :-



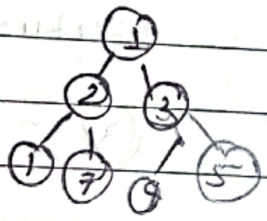
If we pick 2, then can't pick 1, coz they are adjacent.
So, we choose 2, 4, 5
=> 2 + 4 + 5 = 11 (Ans)

* Intution :- (memorization)

2 possible soln, (1st) we include 1 (root) (2nd) doesn't include 1.

(1st) if we include 1 then can't pick 2 & 3, rest we can pick them all.

inc => root → left → left
=> root → left → right
=> root → right → right
=> root → right → left



simply add these stuff in ans.

Date: / / Page no:

Implement :-

```
unordered_map<Node*, int> dp;

int func(node* root) {
    if (!root) return 0;

if (dp[root]) return dp[root]; // memoization

int inc = root->data;
if (root->left) {
    inc += func(root->left->left);
    inc += func(root->left->right);
}
if (root->right) {
    inc += func(root->right->left);
    inc += func(root->right->right);
}

int exc = func(root->left) + func(root->right);

dp[root] max(inc, exc);

return dp[root];
}
```

Date: 20/3/22

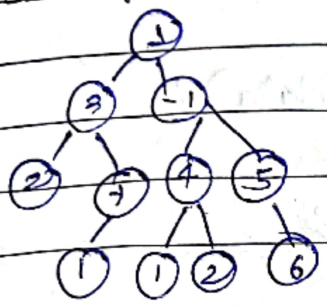
Ques: Print all K-sum path in BT.

Q BT & no. k is given, print every path in tree with sum of node in path as k.

Path can start from any node & end at any node & must be downward only.

i.e., they need not be root node & leaf node;

(*) Example :-



K = 5

3,2 ; 3,1,1 ; 1 3 1 ;
4 1 ; 1 -1 +1 ; -1 4 2 ;
5 ; 1 -1 5.

(All possible ans)

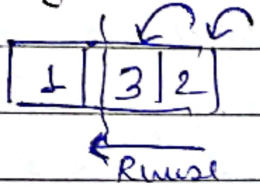
(*) Intuition :-

As we traverse in the tree downward, we'll store all the ~~post~~ path in a vector.

Then when we reach at any leaf node,

we'll start a loop in that path which is stored in vector, whether it is = k or not (loop in reverse).

Dry Run



sum = 2 + 3 = 5 ⇒ break

print 3 2

Now 2's work is over & we'll backtrack to 3, again do the same steps above.

| 1 | 3 | 1 | 1 |

← sum = 1 + 1 + 3 = 5

[3, 1, 1]

Back track 1.

(*) Implement :-

```
void func (node * root, vector<int> & path, int K)
{
  if (!root) return;
      path.push_back(root->data);
func (root->left, path, K);
func ( root->right, path, K);

  int f=0;
for (int j= path.size()-1; j>=0; j--)
{
      f += path[j];
  if (f==k) {
      for( int m=j; m<path.size(); m++)
  }     cout<<"path[m]<<" ";
      cout<<endl;
}

path.pop_back();
}
```

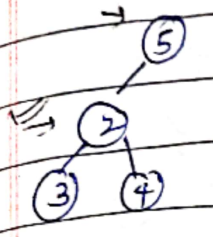
Ques → Longest common Ancestors in a BT

Given BT with all unique value & two nodes value n_1 & n_2. Find LCA of BT.

Examples :-

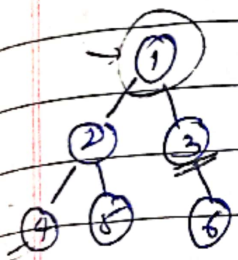
LCA = 1st parent node of any node is its ancestor,

② → 1 & ③ → 1 = True

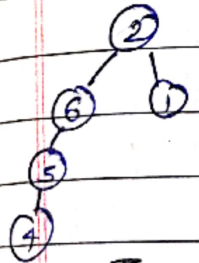


$n_1 = 3$ $n_2 = 4$

Ans => 2,



$n_1 = 4$, $n_2 = 3$ Ans = 1



$n_1 = 6$, $n_2 = 4$

∵, 4 is comming from 6

So Ans = 6

(*) Intution :-

① firstly we'll find n_1 & n_2, if any of them encounter, return them to backstack

② For any node, we are getting value from left & right which is not null, i.e LCA confirm.

③ Above step follow for entire tree till we find n_1 & n_2.

Date: / / Page no:

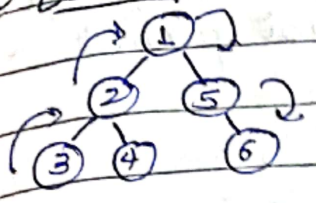
* Code :-

```
node * lca (node * root, int n1, int n2)
{
    if(!root) return root;
    if(root -> data == n1 or root -> data == n2)
        return root;

    node * l = lca(root -> left -> n1 -> n2)
        r = lca(root -> right -> n1 -> n2)
    if (l and r) return root;
    if (l) return l;

    else return r;
}
```

Ques → Find dist b/w 2 nodes in BT

Given BT, & 2 node values, find min. dist. b/w them

⊛ Example :-



3 to 6 => 4 = ans

3 to 5 => 3 (ans)

⊛ Intution :-

→ we've to count the edges b/w nodes. n1 & n2.

→ ∴ we've to find the common point b/w n1 & n2, so we use LCA.

→ firstly find the LCA, then find / calculate left distance and right distance.

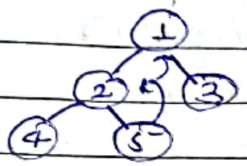
→ Add both left and right and return. the answere.

Date 21/3/22 Page no:

Ques → K^{th} ancestor of node in BT

Given BT, in which nodes are numbered from 1 to n. Given node & a +ve K.
We've to print K^{th} ancestor of given node in BT. If doesn't exist print -1.

Example :-



2^{nd} ancestor of 5 = 1
3^{rd} ancestor of 5 = -1

Intuition :-

We can solve with the help of DFS.
First find given node in tree & then backtrack K times to reach to K^{th} ancestor.

Once we've find/reached K^{th} ancestor parent,
we'll simply print the node & return NULL.

(2) Implement :-

```
Node * kthAncestor(Node *root, int node, int &k)
{
    if(!root) return NULL;
    node* temp = NULL;
    if( root -> data == Node ||
    (temp = kthAncestor(root -> left, node, k) ||
    (temp = kthAncestor(root -> right, node, k)))
    {
        if( k>0 )      k--;
        else if (k==0)
        { cout << "kth ancestor is: " << root -> data;
          return NULL; }
        return root;
    }
}
```

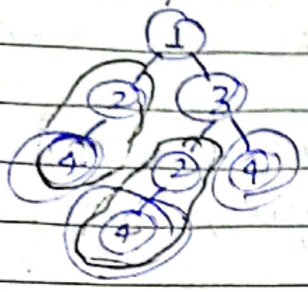
(3)

Date: / / Page no:

Ques → find all duplicate subtree in BT

Given BT of size N, task is find all duplicate subtree from given BT.

* Example :-



⇒ 2 — 4,
⇒ 4.

* Intution :-

→ We'll use hashing. we store
→ inorder traversal of subtree in a hash.
∵, simple inorder traversal can't uniquely identify a tree we can
→ use symbol like '(' and ')' to represent NULL node.

→ We pass unordered map as argument to helper func^n, which recursively calculates inorder string & Ise its count in map.
→ If str get repeated, then it will imply duplication of subtree rooted at that node so push that node in final result and return vector of this node.

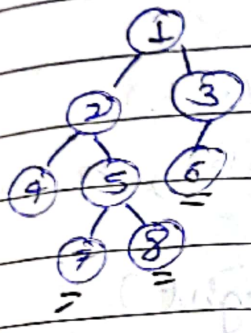
Ques → Check if tree is isomorphic

Check whether tree is isomorphic or not

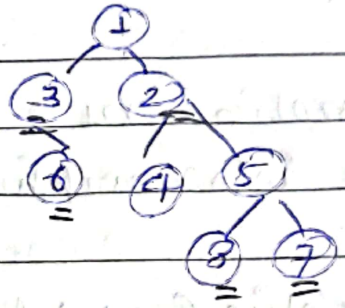
or, isomorphic : 2 tree are called isomorphic if one can be obtained from another by series of flip. i.e by swaping left and right children of several node.

Example :-

Tree 1



Tree 2



flip : 2 & 3, NULL and 6, 7 & 8.
(isomorphic)

Intution :-

we simultaneously traverse both tree. Let current internal node of two tree being traversed be n1 and n2 respect. following 2 condtn for subtree rooted with n1 & n2 to be isomorphic :

1) Data on n1 & n2 same.
2) one of the following two condtn for children of n1 and n2.

(a) Left child of n_1 is isomorphic to left child of n_2 and right child of n_1 is isomorphic to right child of n_2.

(b) Left child of n_1 is isomorphic to right child of n_2 & right child of n_1 is isomorphic to left child of n_2.

(*) Snippet :-

```
(isIsomorphic(n1→left, n2→left)
    && isIsomorphic(n1→right, n2→right))
||
(isIsomorphic(n1→left, n2→right)
&& isIsomorphic(n1→right, n2→left)).
```