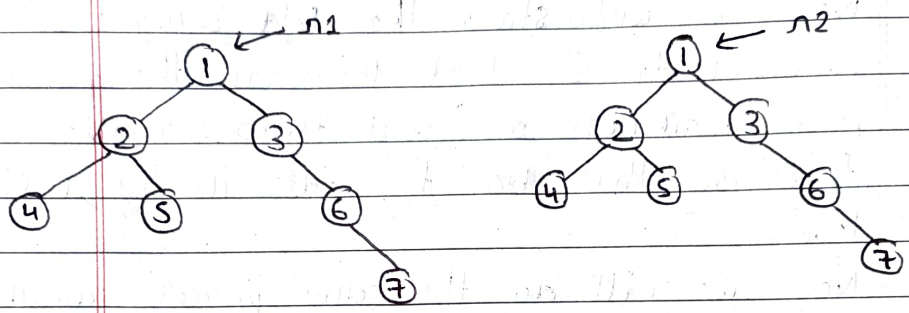
Date _____ Page _____

Day-154

Trees-4

* **Two trees are identical:**



⇒ We will check every node of both trees one by one i.e. we will check left part of both the trees then check node then right part.

⇒ If all part are same then we can say, both trees are identical. and return 1.

⇒ Otherwise we will return 0.

⇒ We will first check node values then check left and right.

⇒ And return 1 if both left & right is equal.

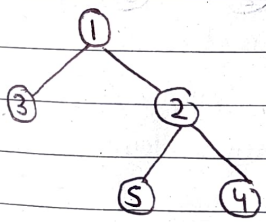
Code

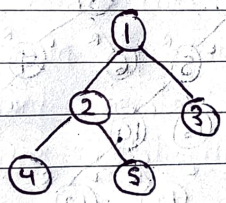
Date _____ Page _____

```
bool isIdentical(Node *n1, Node *n2){
    if(n1 == NULL && n2 == NULL)
        return 1;
    if(!n1 && n2) || (n1 && !n2)
        return 0;
    if(n1->data != n2->data)
        return 0;
return isIdentical(n1->left, n2->left) &&
        isIdentical(n2->right, n2->right);
```

T.C → O(n) , S.C. → O(n) or O(h)

* Mirror Tree:





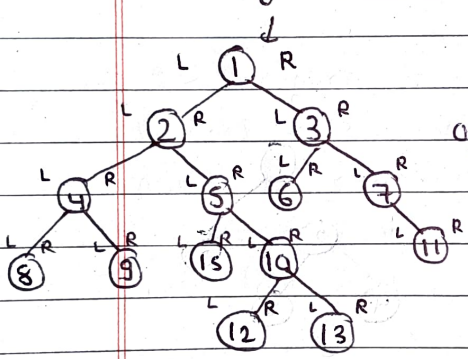
- Here left part is becoming right part & vice versa.
- We will change the left part by right part and right part by left part.

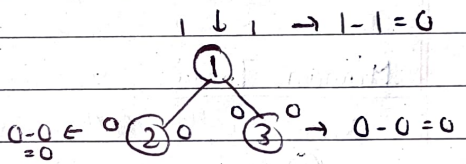
Code

Date _____ Page _____

```
void Mirror (Node * root){
    if(!root)
        return;
    Node * temp = root → left;
    root → left = root → right;
    root → right = temp;
    Mirror( root → left );
    Mirror ( root → right);
}
```

* **Check for Balance Tree:**





⇒ If the height from every node is equal to -1, 0 or 1.

⇒ Then that tree is called Balanced Tree.

$$-1 \leq \text{height(L)} - \text{height(R)} \leq 1$$
$$(-1, 0, 1)$$

⇒ So, we will take a valid variable and sets its value to 1.

Date _____ Page _____

⇒ So, that if we get any unbalanced node then we will change the value of valid to 0.

⇒ First we have to find the height of the left side & height of right side.

⇒ Then, we will for balanced condition.

⇒ If the node is balance then return the height to the above node.

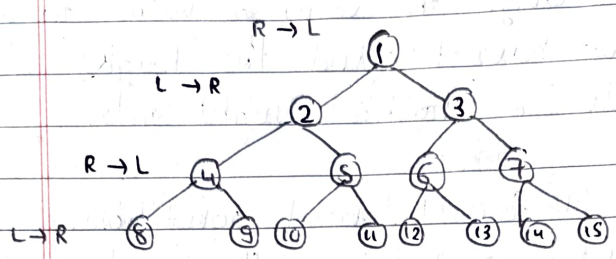
Code

```
bool isBalance (Node *root){
    bool valid;
    height (root, valid);
    return valid};
}

int height (Node *root, int &valid){
    if(root == NULL)
        return 0;
    int L = height (root→ left, valid);
    int R= height (root→ right, valid);
    if( abs(L-R) > 1 )
        valid =0;
    return 1 + max(L,R);
}
```

Date _____ Page _____

* Level Order Traversal in spiral Form:



nodes

⇉ We have to print ~~spiral~~ in the form of spiral i.e S.

⇉ So, 1 2 3 7 6 5 4 8 9
10 11 12 13 14 15 ....

⇉ 1st Approach

⇉ We can observe a pattern that we are printing root from right to left then next level from left to right and we will follow same cycle again and again.

⇉ So, we can use a queue and store the root in it.

⇉ Then pop the root on front & push its children in the order of that level printing i.e L to R or R to L.

⇉ Now, we have to reverse the elements. So, we will use a stack to do this.

⇉ After that again we will follow all these steps.

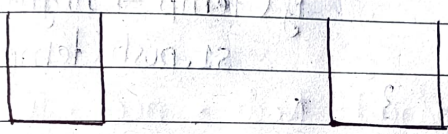
Date _____ Page _____

2nd Approach

=> As we have noticed, we are doing reverse of queue by using stack again & again.

=> So, to don't do this, we can use stack.

=> We will use two stack one for storing L to R & one for R to L elements.



L to R R to L

Code

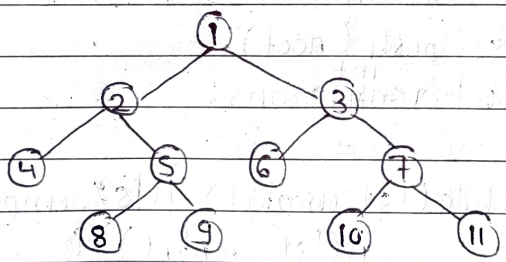
```
vector<int> levelOrder(Node * root){
    stack<Node *> s1; // R to L
    stack<Node *> s2; // L to R
    s1.push(root);
    vector<int> ans;

    while(!s1.empty() || !s2.empty()){
        if(!s1.empty()){
            while(!s1.empty()){
                Node * temp = s1.top();
                s1.pop();
                ans.push_back(temp->data);
                if(temp->right)
                    s2.push(temp->right);
                if(temp->left)
                    s2.push(temp->left);
            }
        }
    }
```

Date _____ Page _____

```
else{
    while(!s2.empty()){
        Node *temp = s2.top();
        s2.pop();
        ans.push_back(temp->data);
        if(temp->left)
            s1.push(temp->left);
        if(temp->right)
            s1.push(temp->right);
    }
} return ans;
}
```

* Check if 2 Nodes are cousin:



⇒ Two nodes are cousin if -

⇒ both the nodes are at same level but have different parents.

⇒ So, here, we will use level order traversal for finding the levels.

Date _____ Page _____

⇒ After that when we find the levels we will stop finding the levels.

⇒ Now, we will find that they are children of same parent or different.

⇒ For that, we will ask every node that those nodes are your children.

⇒ In the end, if we get answer no then we can return 1 as our actual ans.

Code

```
bool isCousin(Node* root, int a, int b){
    queue<Node*> q;
    q.push(root);
    int l1 = -1; l2 = -1;
    int level = 0;
    while(!q.empty()){
        int n = q.size();
        while(n--){
            Node* temp = q.front();
            q.pop();
            if(temp->data == a)
                l1 = level;
            if(temp->data == b)
                l2 = level;
            if(temp->left)
                q.push(temp->left);
            if(temp->right)
                q.push(temp->right);
```

Date _____ Page _____

```
        level++;
        if(l1 != l2)
            return 0;
        if(l1 != -1)
            break;
    }
return !parent(root, a, b);
}

bool parent(Node * root, int a, int b){
    if(root == NULL)
        return 0;
    if(root->left && root->right){
        if(root->left->data == a
        && root->right->data == b)
            return 1;
        if(root->left->data == b &&
            root->right->data == a)
            return 1;
    }
return (parent(root->left, a, b) || parent(
        root->right, a, b));
}
```