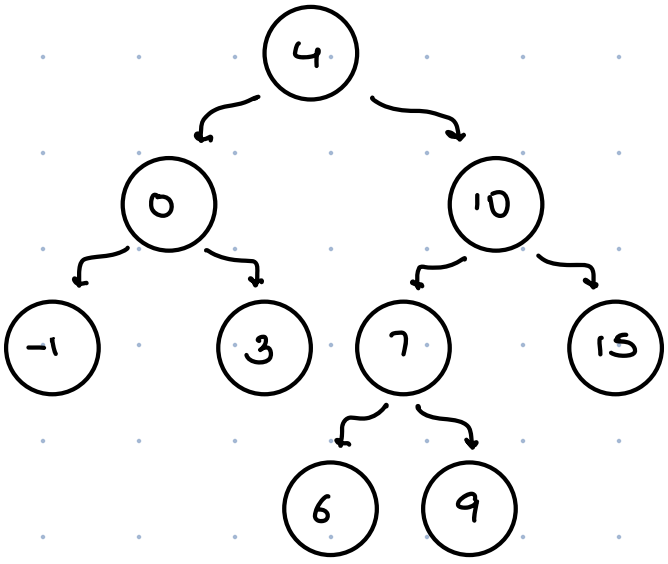
Trees 4 : LCA

Good Evening

Content

01. K^th Smallest Element in BST
02. Morris Inorder Traversal
03. LCA in Binary Tree
04. LCA in BST

01. Find K^{th} smallest element in BST



K = 3 ⟶ 3

K = 5 ⟶ 6

K = 8 ⟶ 10

Idea → Do an inorder traversal on BST & store all elements in AL & return al.get(k-1)

```
main(){
    AL<I> ans = new AL<>();
    inorder (root, ans);
    return ans.get(k-1);
}

void inorder (root, AL<I> ans){
    if (root == null) return;
    inorder (root.left, ans);
    ans.add (root.val);
    inorder (root.right, ans);
}
```

TC: O(n)

SC: O(n)

Approach 2 → while doing inorder traversal, maintain a count

```
int count = 0
int ans = -∞

void inorder (root, int k) {
    if (root == null) return;
    inorder (root.left, k)
    count = count + 1;
    if (count == k) {
        ans = root.val;
        return;
    }
    if (ans == -∞) inorder (root.right, k)
}
```

Tc : O(n)

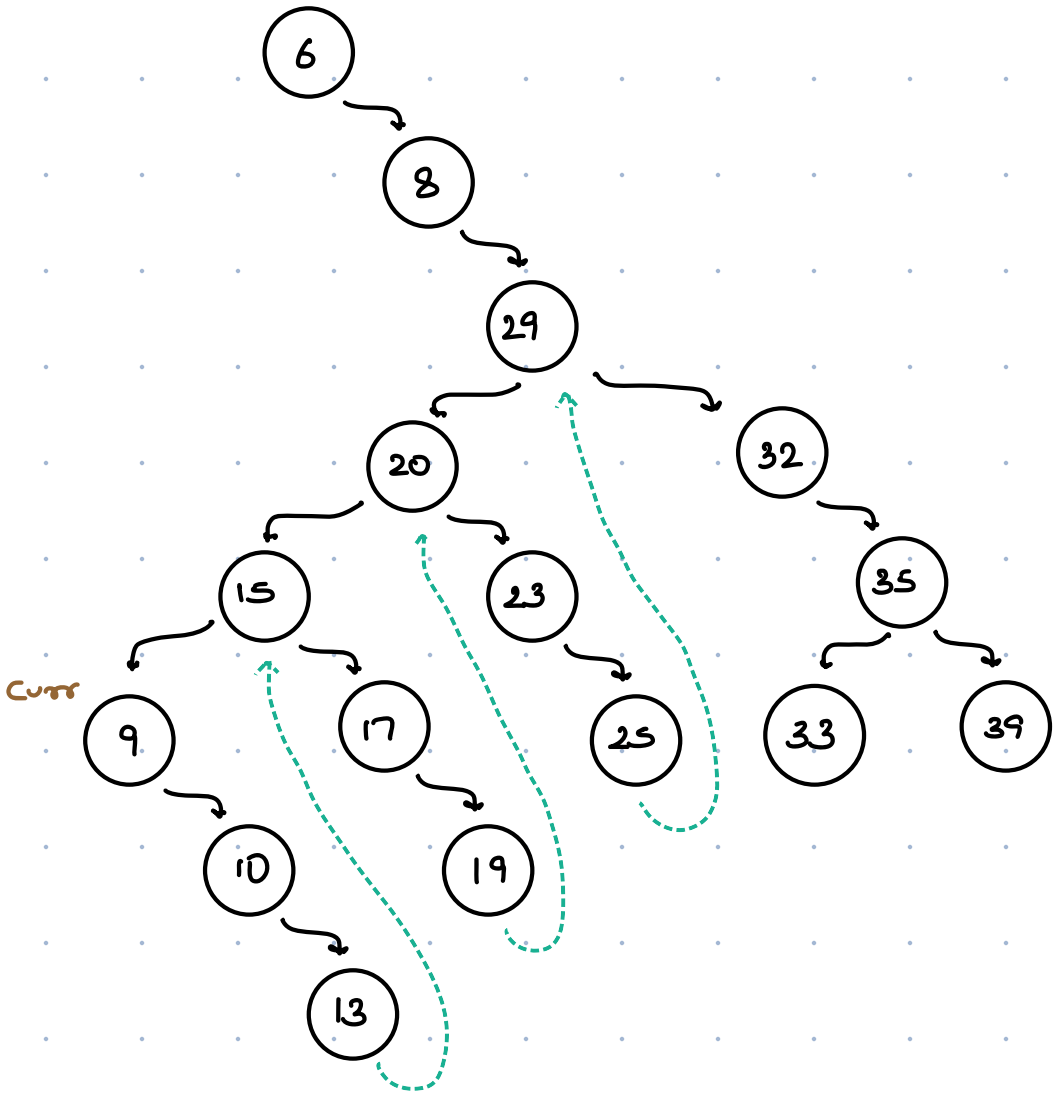
Sc : O(ht)

* Inorder Traversal

print left subtree
print root
print right subtree

- Recursion — Sc : O(h)
- Iterative — Sc : O(n)
- Morris Traversal — Sc : O(1)

Inorder = 6 8



Obs = If (curr.left == null) → print curr;

→ Then move to RHS

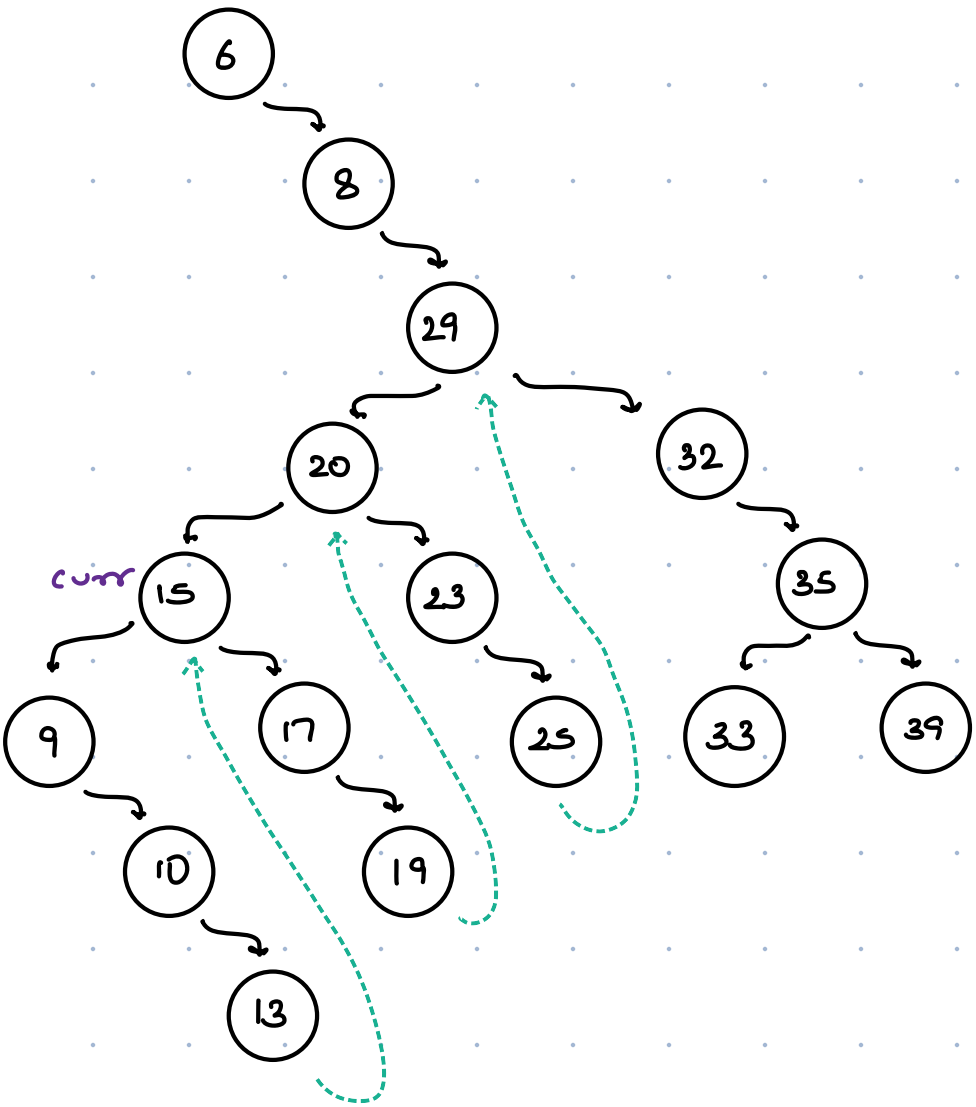
Inorder predecessor

→ Last node of left subtree to be printed OR

→ right most child of left subtree OR

→ Node in LST that doesn't right child

Inorder Traversal = 6 8 9 10 13



```
Node curr = root;

while (curr != null){

    if (curr.left == null){

        print (curr.data);

        curr= curr.right;

    }

    else{

        Node temp = curr.left;

        while (temp.right != null && temp.right != curr ){

            temp = temp.right;

        }

        if (temp.right == null){

            temp.right = curr;

            curr = curr.left;

        }

        if (temp.right == curr){

            temp.right = null;

            print( curr.data)

            curr= curr.right;

        }

    }

}
```

Tc : O(n)

Sc : O(1)

create connection

break connection

9:58 pm → 10:08 pm

We are all connected

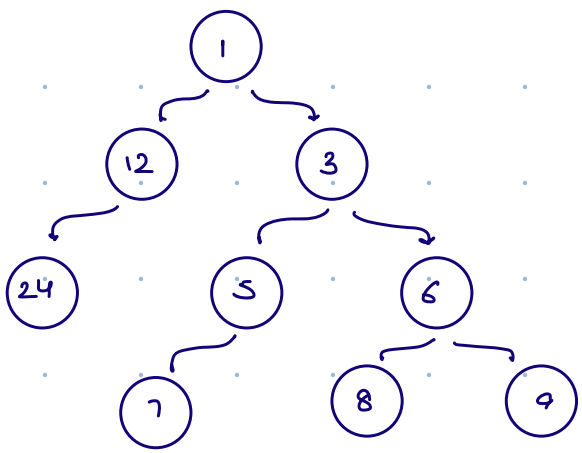
Problem Statement

It is said that we all **humans** are related through some **common ancestor** at some point of time. Assume that a person can have **0, 1 or 2 children only**.

Given the Binary tree **A** representing the family tree, discover the earliest common family member who connects two given people **B and C** in a family tree.

Example

```
       Ram
      /   \
   Shyam    Bob
    /   \      \
Dholu  Bholu   Satish

-> Find LCA of Dholu and Bob
-> Answer = Ram
```



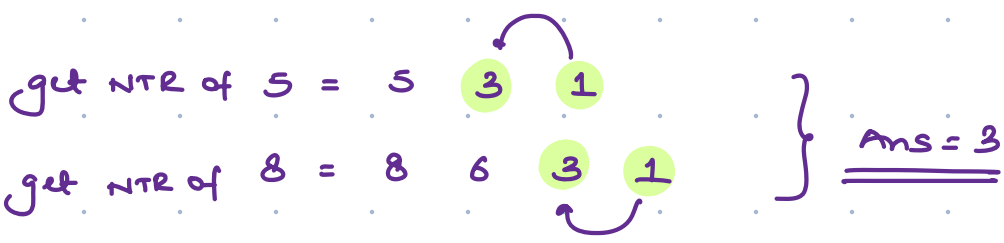
LCA (12, 5) ⇒ 1

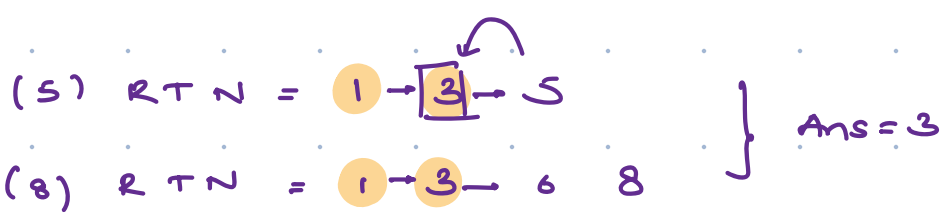
LCA (8, 9) ⇒ 6

LCA (3, 8) ⇒ 3

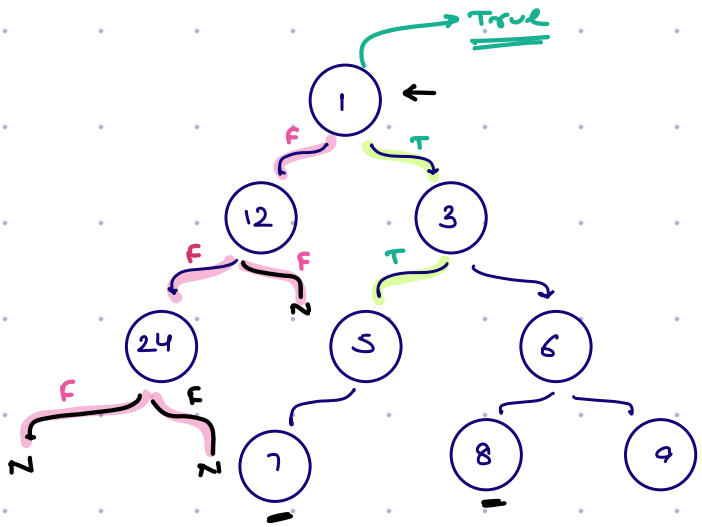
Idea → Node to Root Path

LCA (5, 8)





* Search for an element in Binary Tree




```
boolean search ( Node root, int k)
|
|   if (root == null) return false;
|
|   if (root.data == k)   return true;
|
|   if ( search (root.left, k) == true){
|   |
|   |   return true;
|   |
|   }
|
|   if ( search (root.right, k) == true){
|   |
|   |   return true;
|   |
|   }
|
|   return false;
|
}
```

* Node to Root Path

```
boolean search ( Node root, int k, AL<I> ans )
{
    if (root == null) return false;

    if (root.data == k){
        ans.add(root.data);
        return true;
    }

    if ( search(root.left, k) == true){
        ans.add(root.data)
        return true;
    }

    if (search(root.right, k) == true){
        ans.add(root.data)
        return true;
    }

    return false;
}
```

* LCA in Binary Tree for k_1 & k_2

01. Get NTR for k_1 ⟶ Reverse it
02. Get NTR for k_2 ⟶ Reverse it

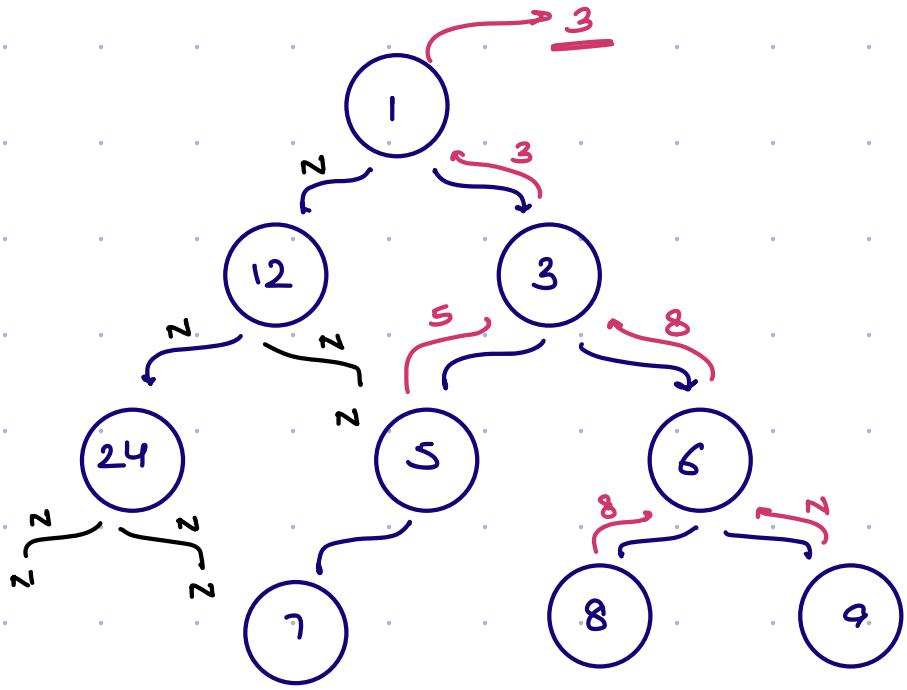
TC: O(n)
SC: O(h)
O(ht of tree)

$k_1 = 7$ RTN = 1 3 5 7

$k_2 = 8$ RTN = 1 3 6 8

Ans = 3

03. Traverse from start & return last common node.

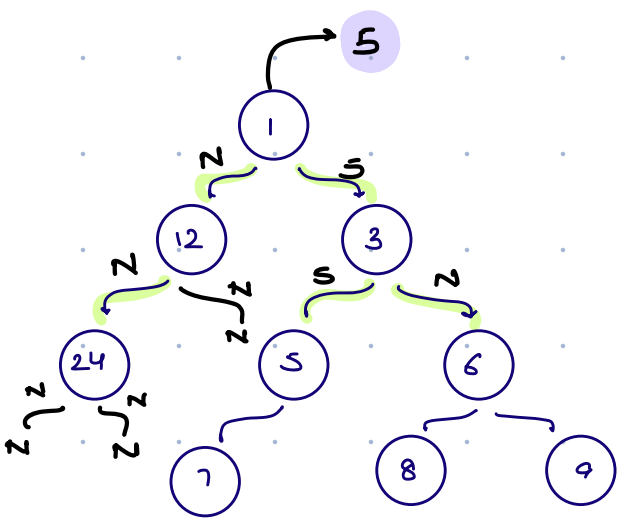


LCA (5, 8)

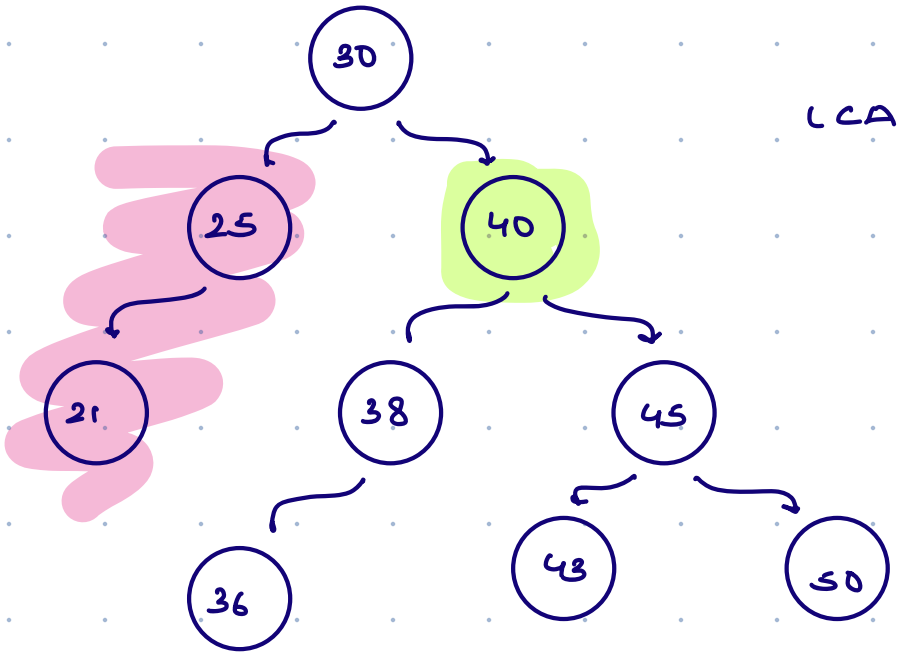
```
Node  LCA ( root, p, q ){

    if (root == null) return null;

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

    l =  LCA ( root.left , p, q );

    r =  LCA (root.right, p, q);

    if ( l != null && r != null) return root;

    return  l == null ? r : l;

}
```

TC : O(n)
SC : O(ht)

p = 5
q = 7

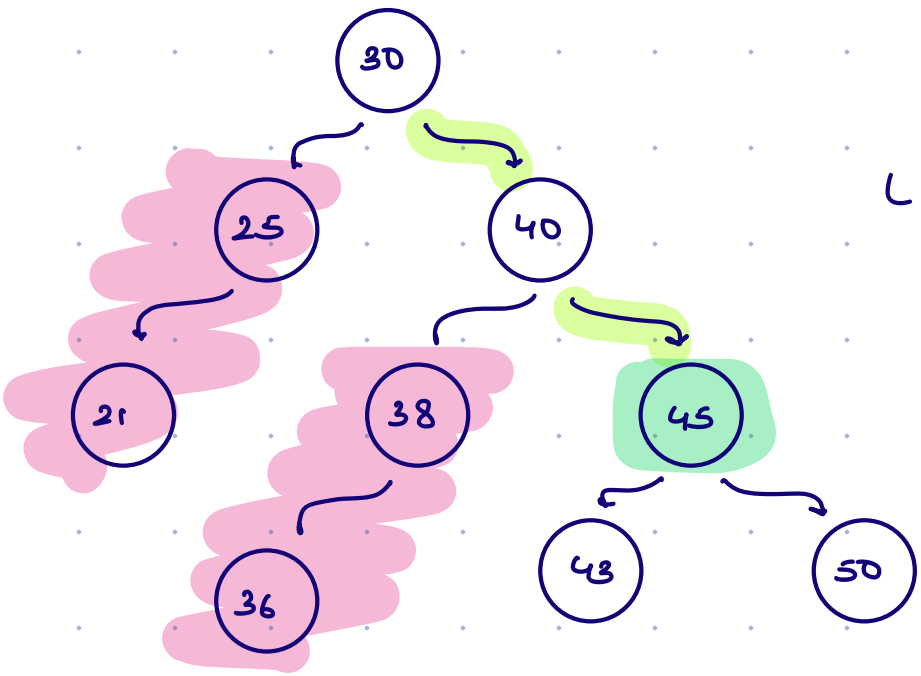


* Lowest common Ancestor in Binary Search tree



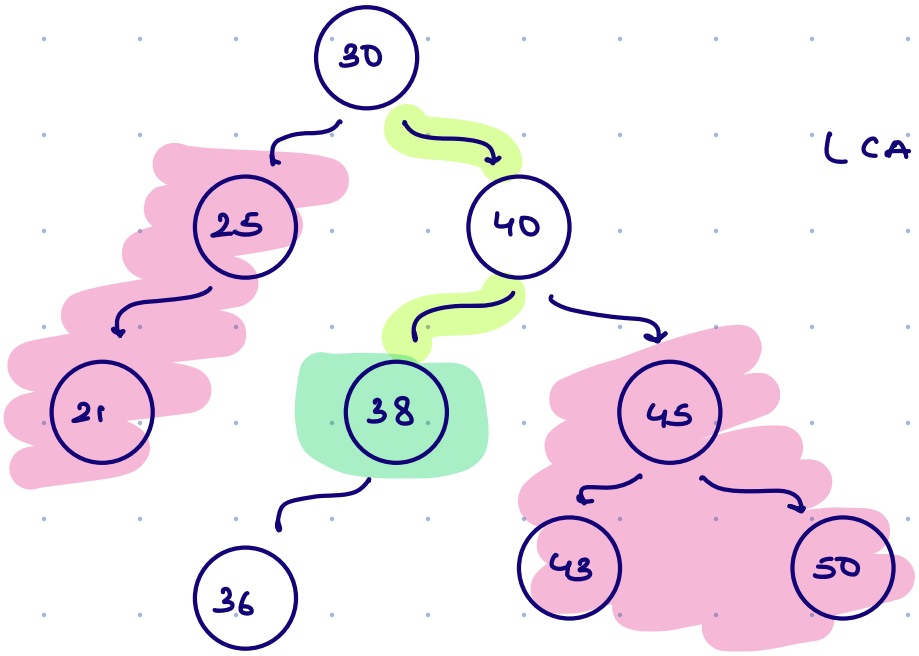
LCA (36, 43)

Ans = 40



LCA (43, 50)

Ans = 45

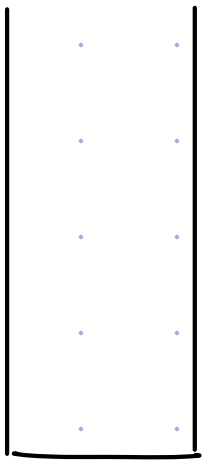


LCA (36, 38)

* Code in whatsapp group (very easy)

---

longest substring of valid parathesis



cnt : 1

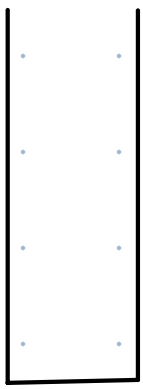
Valid parentheses count

Problem Description

Alice is a mathematician and she loves to solve puzzles. One day, her friend Bob gave her a string **A** of size N consisting of opening and closing parentheses and asked her to find the length of the **longest** valid parentheses **substring**.

Note: A valid parentheses string means that each opening symbol has a corresponding closing symbol and the pairs of parentheses are properly nested.

```
    public int solve(final String A) {
        int n = A.length();
        int ans = 0;
        Stack < Integer > st = new Stack < > ();
        st.push(-1);

        for (int i = 0; i < n; i++) {
            if (A.charAt(i) == '(') {
                st.push(i);
            }
            else {
                if (!st.empty()) {
                    st.pop();
                }
                if (!st.empty()) {
                    ans = Math.max(ans, i - st.peek());
                }
                if (st.empty()) {
                    st.push(i);
                }
            }
        }
        return ans;
    }
}
```



(( ) ( )
0 1 2 3 4

(( ))) (( ) ( ) ( ))

(( )) ( )