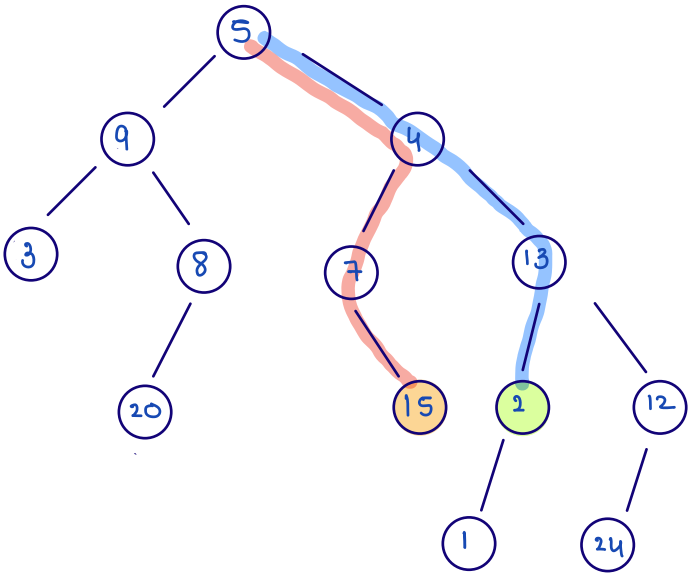
Agenda

1) Node to root path

2) LCA (lowest common ancestor)

3) K-far | K-away

Q.1 Given root of BT and a val, find node to root path

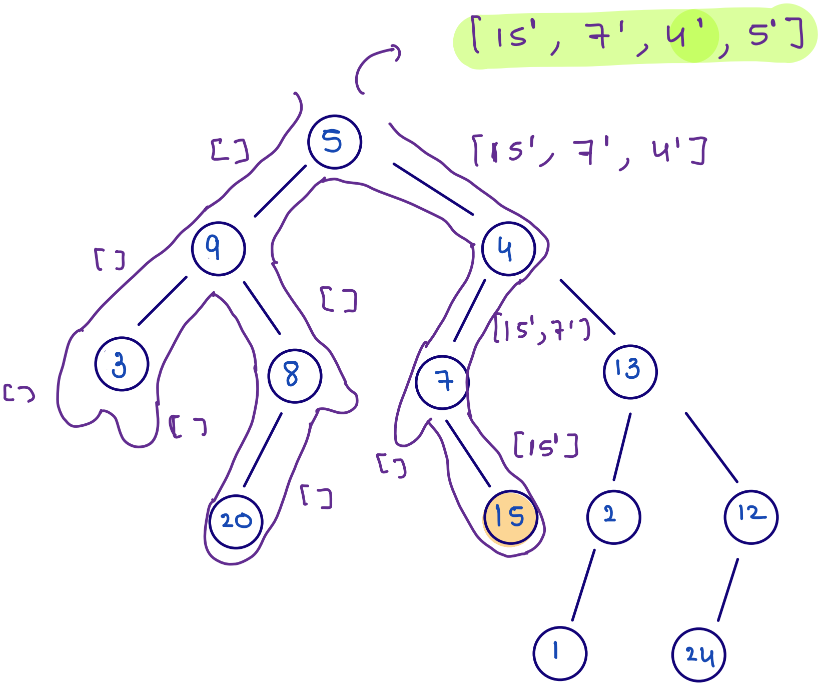


val = 2

ans = [2, 13, 4, 5]

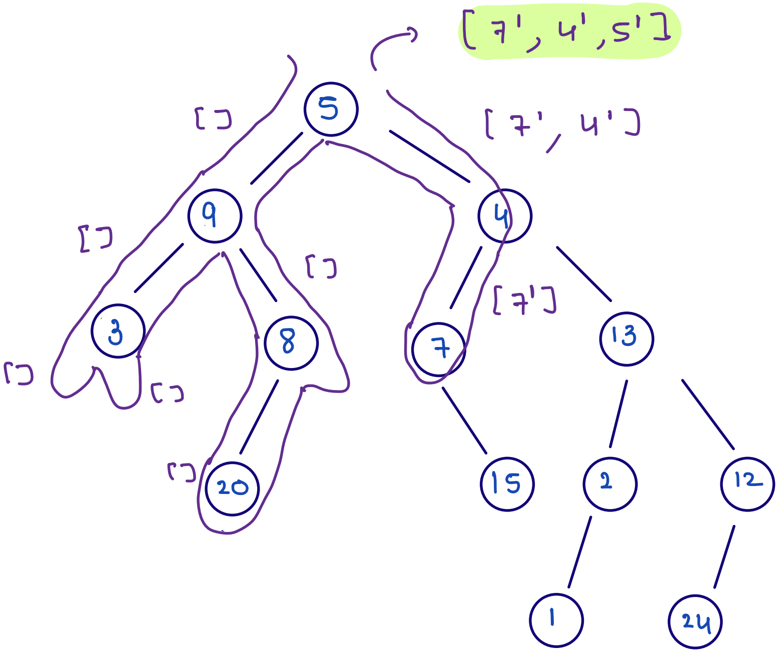
val = 15

ans = [15, 7, 4, 5]



val=15

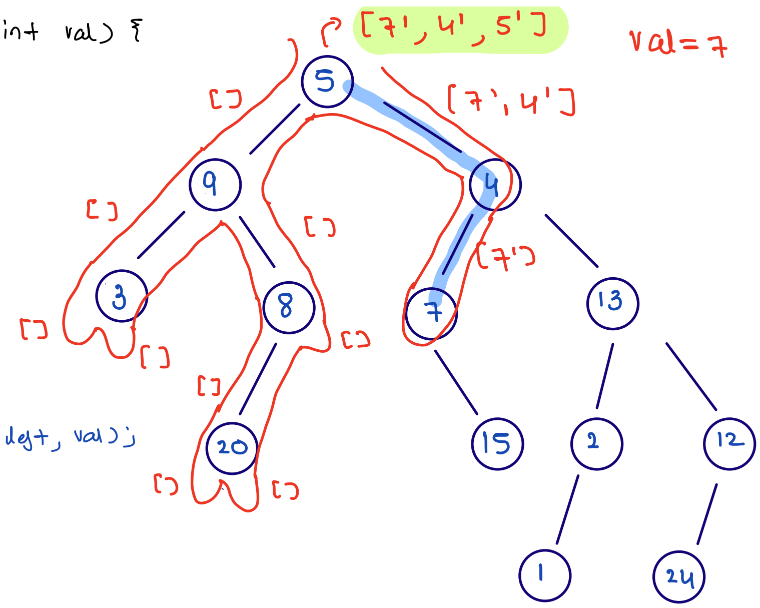
AL<Node>



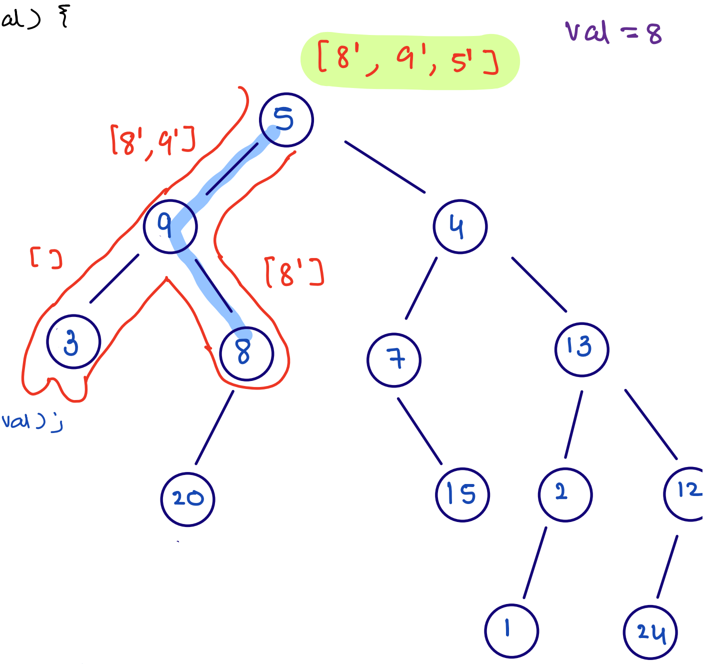
val = 7

```
AL < Node >   nodeToroot Path ( Node root, int val) {
    if (root == null) {
        return new AL<>();
    }

    if (root.val == val) {
        AL< Node > temp = new AL<>();
        temp.add (root);
        return temp;
    }

    AL<Node> la = nodeToroot path(root.left, val);
    if (la.size() >0) {
        la.add (root);
        return la;
    }

    AL<Node> ra = nodeToroot path(root.right, val);
    if (ra.size() >0) {
        ra.add (root);
        return ra;
    }

    return new AL<>();
}
```

```
AL<Node>   nodeToRootPath ( Node root, int val) {
    if (root == null) {
        return new AL<>();
    }

    if (root.val == val) {
        AL<Node> temp = new AL<>();
        temp.add(root);
        return temp;
    }

    AL<Node> la =  nodeToRootPath(root.left, val);
    if (la.size() >0) {
        la.add(root);
        return la;
    }

    AL<Node> ra =  nodeToRootPath(root.right, val);
    if (ra.size() >0) {
        ra.add(root);
        return ra;
    }
    return new AL<>();
}
```

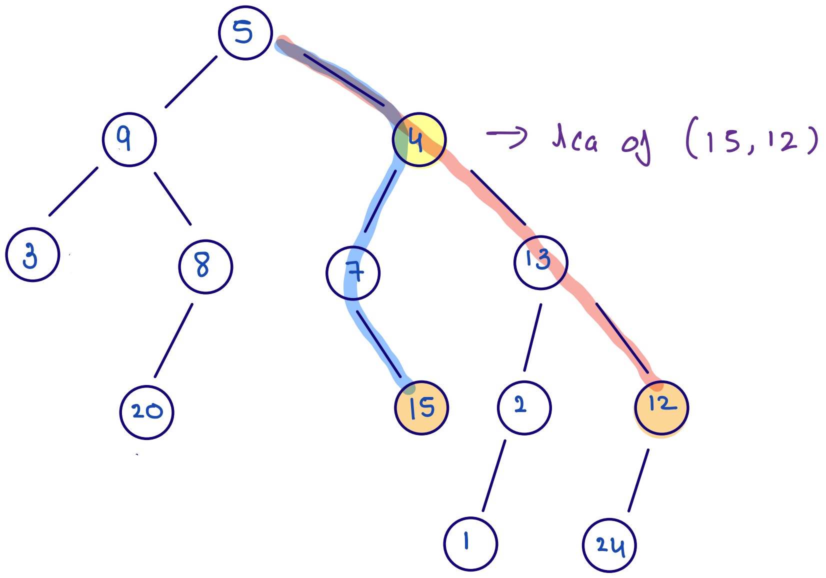


```
AL<Node>   nodeToRootPath ( Node root, int val) {
    if (root == null) {
        return new AL<>();
    }

    if (root.val == val) {
        AL<Node> temp = new AL<>();
        temp.add(root);
        return temp;
    }

    AL<Node> la = nodeToRootPath(root.left, val);
    if (la.size() >0) {
        la.add(root);
        return la;
    }

    AL<Node> ra = nodeToRootPath(root.right, val);
    if (ra.size() >0) {
        ra.add(root);
        return ra;
    }
    return new AL<>();
}
```



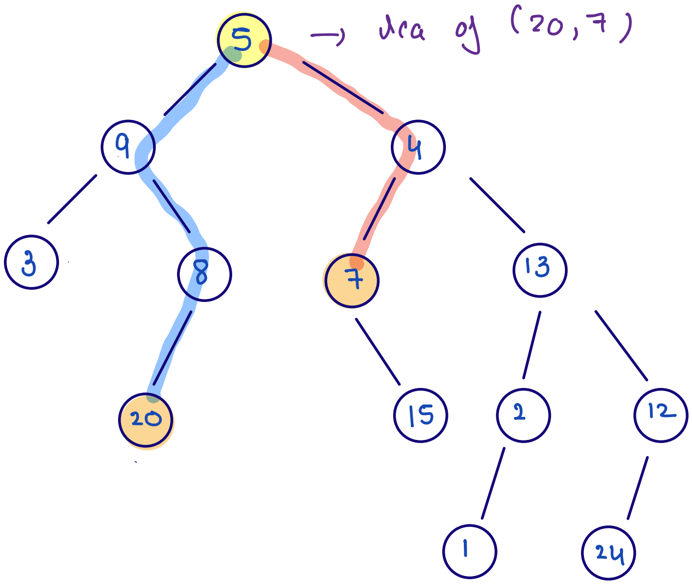
todo : 2 Dry runs

Q.2 Given root of a BT and two nodes (their values). Find LCA (lowest common ancestor) of these two nodes.



v1 = 15

v2 = 12



v1 = 20

v2 = 7

d1 → nodeToRootPath (root, v1)

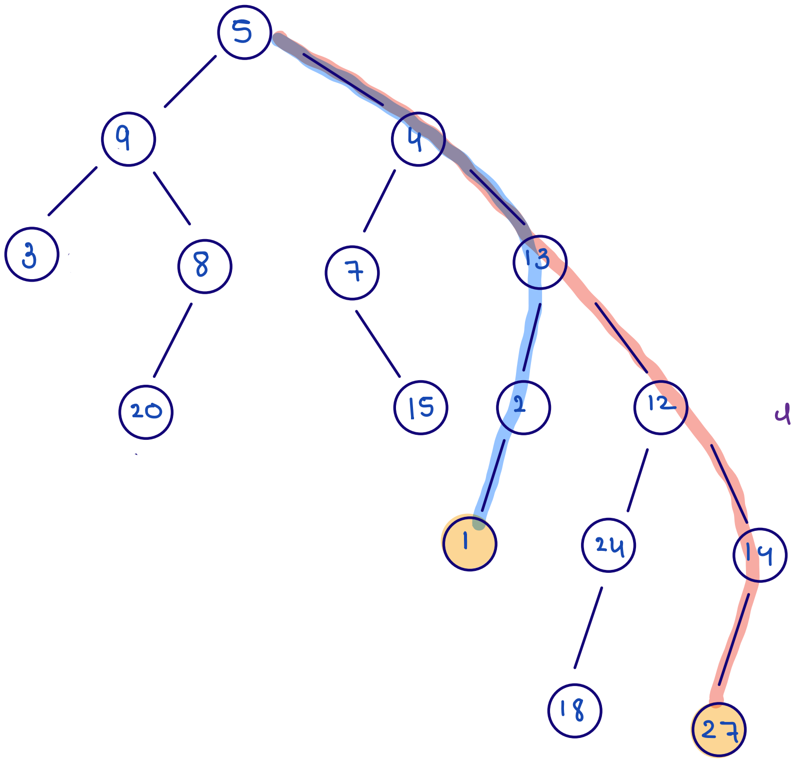
d2 → nodeToRootPath (root, v2)

i

d1 = [20 8 9 5]

v2 = [7 4 5]

j



v1 = 1

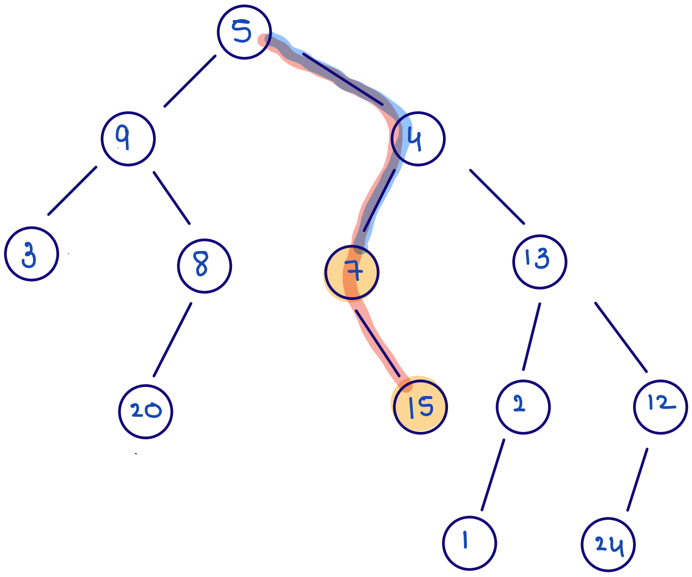
v2 = 27

i

u1 = [1 , 2 , 13 , 4 , 5]

u2 = [27 , 14 , 12 , 13 , 4 , 5]

j



v1 = 7

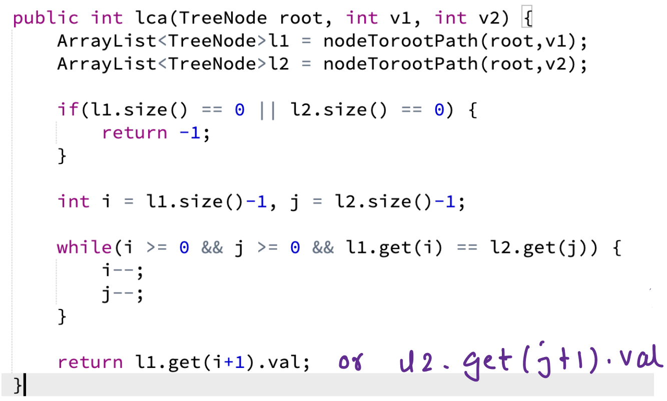
v2 = 15

i

u1 = [7 , 4 , 5]

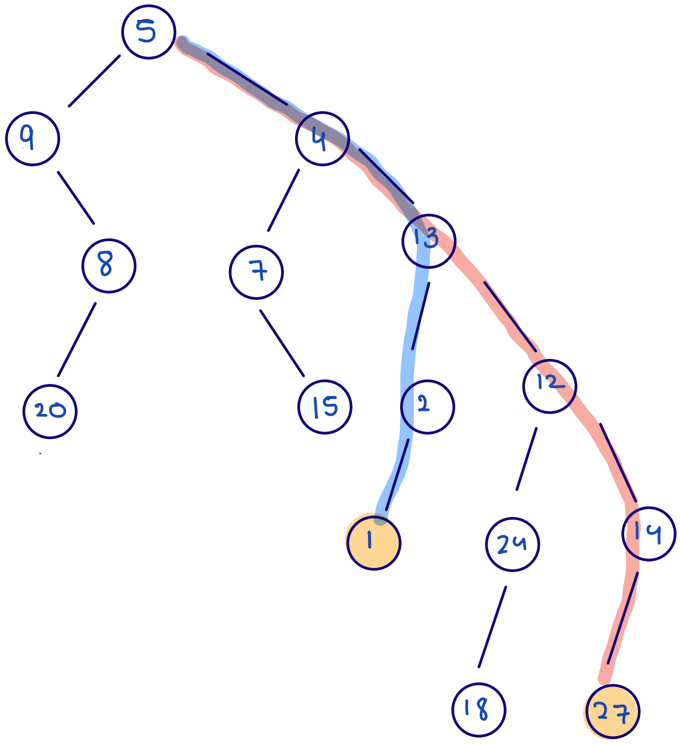
u2 = [15 , 7 , 4 , 5]

j



```
public int lca(TreeNode root, int v1, int v2) {
    ArrayList<TreeNode>l1 = nodeTorootPath(root,v1);
    ArrayList<TreeNode>l2 = nodeTorootPath(root,v2);

    if(l1.size() == 0 || l2.size() == 0) {
        return -1;
    }

    int i = l1.size()-1, j = l2.size()-1;

    while(i >= 0 && j >= 0 && l1.get(i) == l2.get(j)) {
        i--;
        j--;
    }

    return l1.get(i+1).val;
}
```

or l2.get(j+1).val



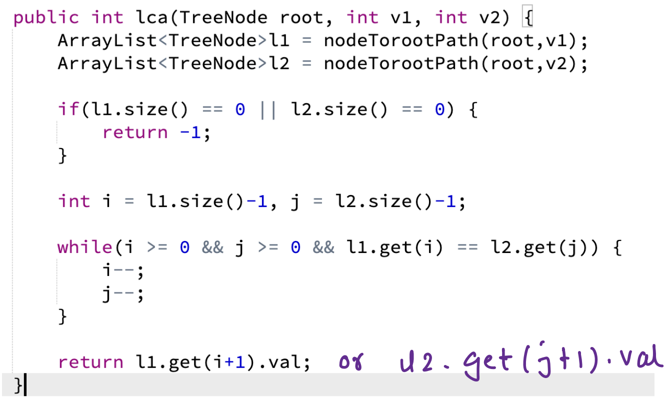
v1 = 1 v2 = 27

i

l1 = [1, 2, 13, 4, 5]

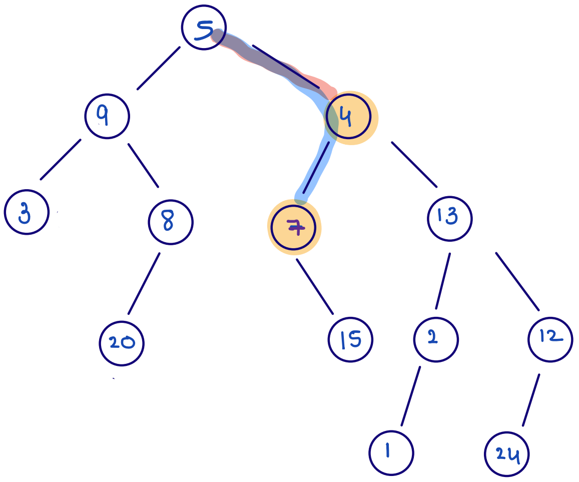
l2 = [27, 14, 12, 13, 4, 5]

j



```
public int lca(TreeNode root, int v1, int v2) {
    ArrayList<TreeNode>l1 = nodeTorootPath(root,v1);
    ArrayList<TreeNode>l2 = nodeTorootPath(root,v2);

    if(l1.size() == 0 || l2.size() == 0) {
        return -1;
    }

    int i = l1.size()-1, j = l2.size()-1;

    while(i >= 0 && j >= 0 && l1.get(i) == l2.get(j)) {
        i--;
        j--;
    }

    return l1.get(i+1).val;
}
```

or l2.get(j+1).val



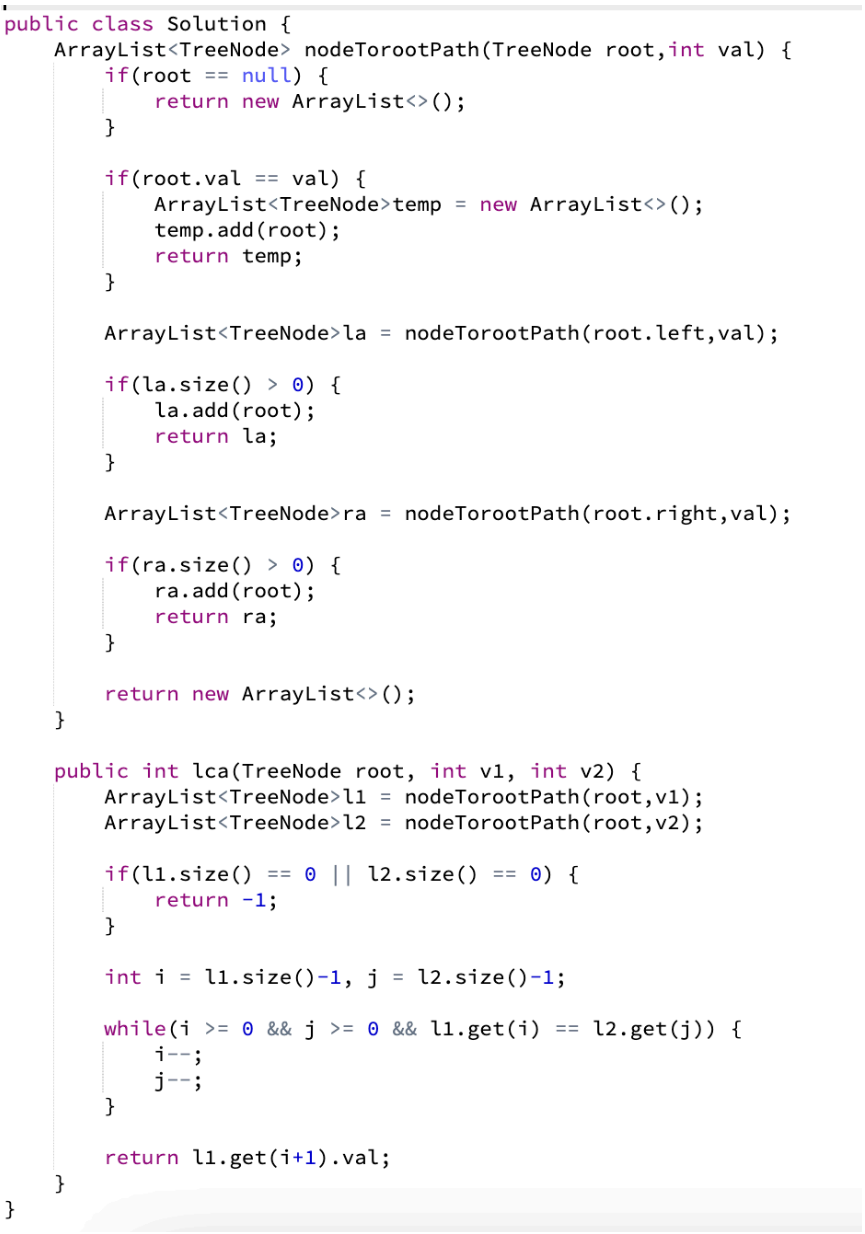
v1 = 7 v2 = 4

i

l1 = [7 4 5]

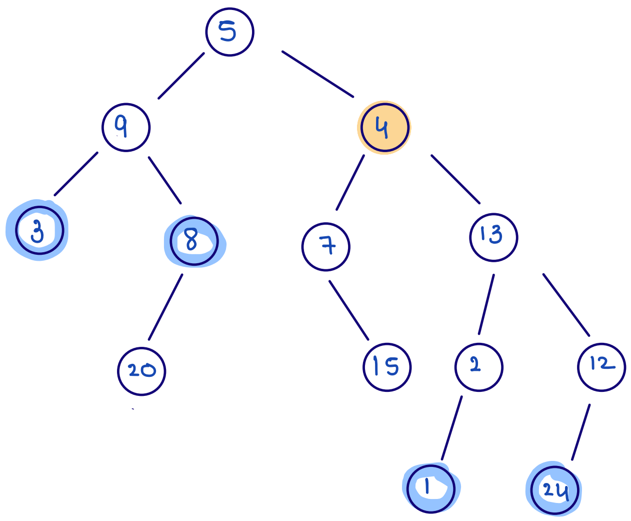
l2 = [4 5]

j



```java
public class Solution {
    ArrayList<TreeNode> nodeTorootPath(TreeNode root,int val) {
        if(root == null) {
            return new ArrayList<>();
        }

        if(root.val == val) {
            ArrayList<TreeNode>temp = new ArrayList<>();
            temp.add(root);
            return temp;
        }

        ArrayList<TreeNode>la = nodeTorootPath(root.left,val);

        if(la.size() > 0) {
            la.add(root);
            return la;
        }

        ArrayList<TreeNode>ra = nodeTorootPath(root.right,val);

        if(ra.size() > 0) {
            ra.add(root);
            return ra;
        }

        return new ArrayList<>();
    }

    public int lca(TreeNode root, int v1, int v2) {
        ArrayList<TreeNode>l1 = nodeTorootPath(root,v1);
        ArrayList<TreeNode>l2 = nodeTorootPath(root,v2);

        if(l1.size() == 0 || l2.size() == 0) {
            return -1;
        }

        int i = l1.size()-1, j = l2.size()-1;

        while(i >= 0 && j >= 0 && l1.get(i) == l2.get(j)) {
            i--;
            j--;
        }

        return l1.get(i+1).val;
    }
}
```

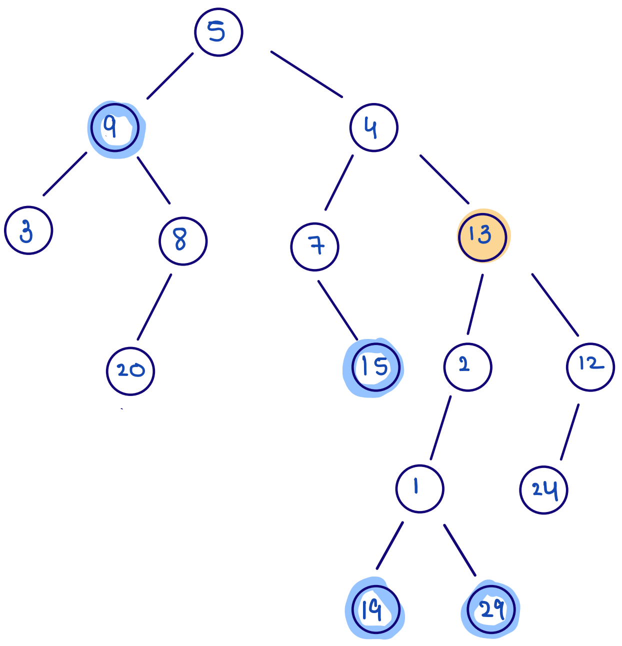
Q.3 Given root of a Binary tree, a val and a distance k. Return AL<Integers> consisting all the nodes that are k distance away from node containing the given val.



val = 4

k = 3 (dist)

ans = [1, 24, 3, 8]

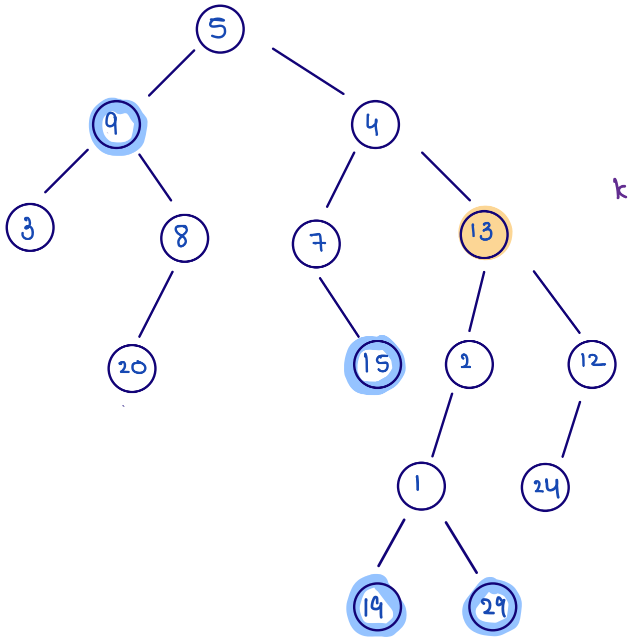


val = 13

k = 3 (dist)

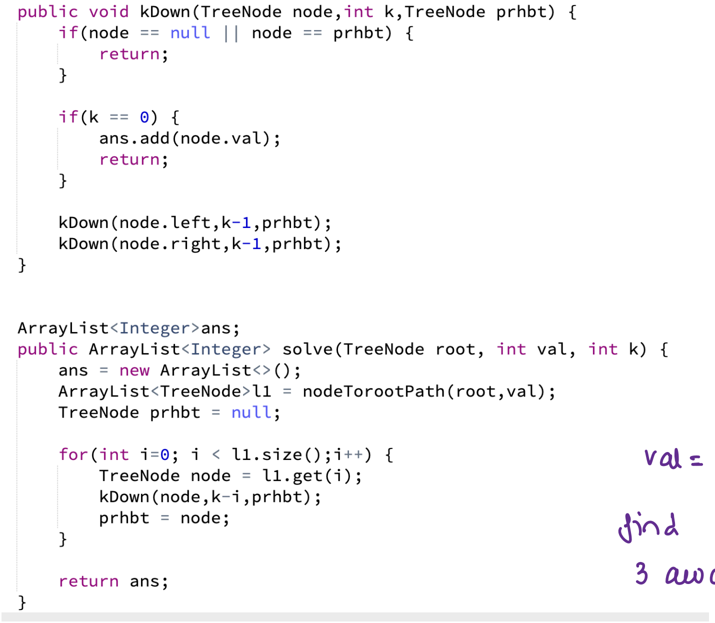
from 13 all nodes that 3 distance away

ans = [19, 29, 15, 9]

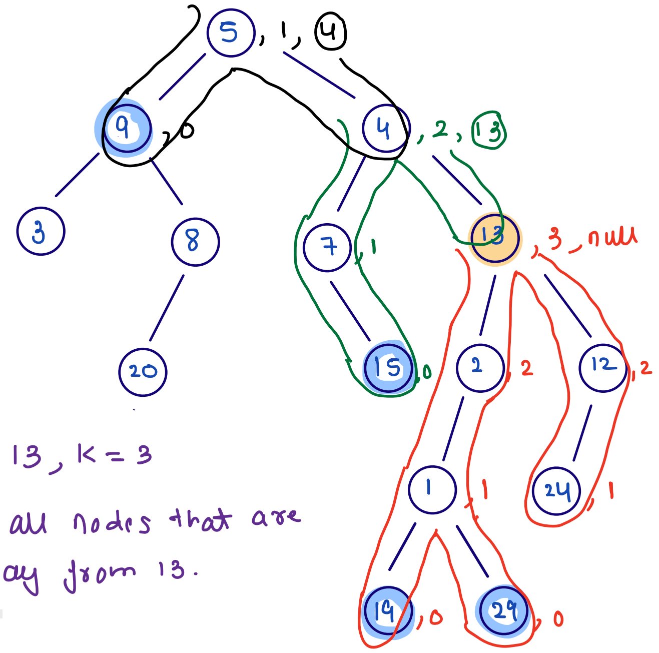


val = 13   k = 3

d1 =	[13	4	5]
kdown (k)	3	2	1
prhbt	null	13	4
	↓	↓	↓
	19,29	15	9



```
public void kDown(TreeNode node,int k,TreeNode prhbt) {
    if(node == null || node == prhbt) {
        return;
    }

    if(k == 0) {
        ans.add(node.val);
        return;
    }

    kDown(node.left,k-1,prhbt);
    kDown(node.right,k-1,prhbt);
}


ArrayList<Integer>ans;
public ArrayList<Integer> solve(TreeNode root, int val, int k) {
    ans = new ArrayList<>();
    ArrayList<TreeNode>l1 = nodeTorootPath(root,val);
    TreeNode prhbt = null;

    for(int i=0; i < l1.size();i++) {
        TreeNode node = l1.get(i);
        kDown(node,k-i,prhbt);
        prhbt = node;
    }

    return ans;
}
```



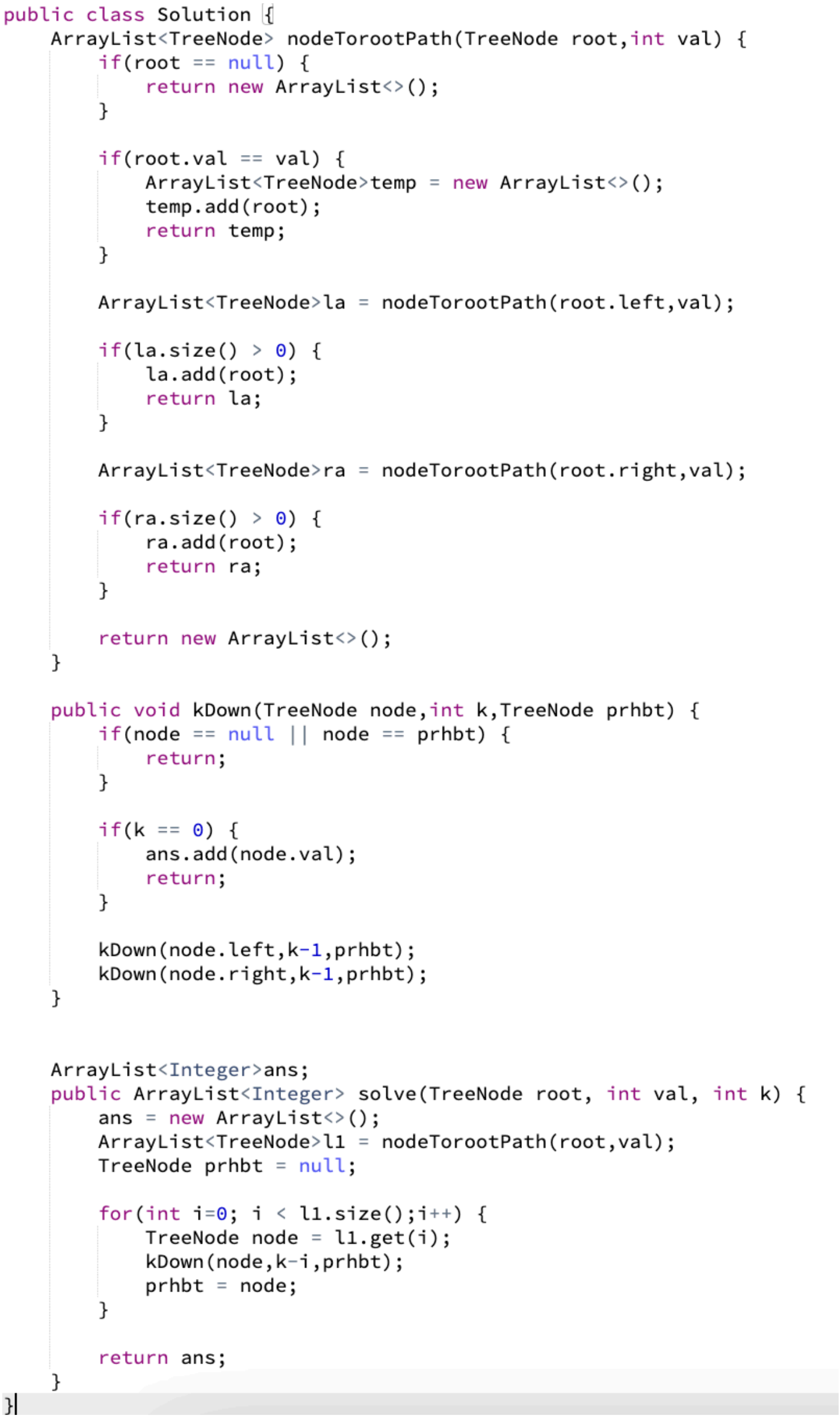
val = 13, k = 3

find all nodes that are 3 away from 13.

$$l1 = [\ \overset{0}{13}\quad \overset{1}{4}\quad \overset{2}{5}\ ]$$

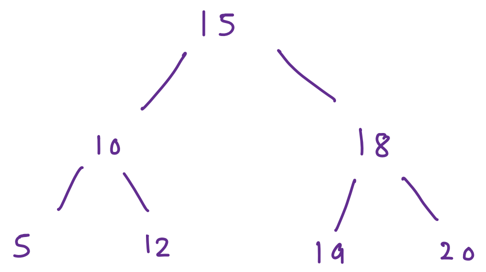
kdown (13, 3, null)   kdown (4, 2, 13)   kdown (5, 1, 4)

ans = [19, 29 , 15, 9]



```
public class Solution {
    ArrayList<TreeNode> nodeTorootPath(TreeNode root,int val) {
        if(root == null) {
            return new ArrayList<>();
        }

        if(root.val == val) {
            ArrayList<TreeNode>temp = new ArrayList<>();
            temp.add(root);
            return temp;
        }

        ArrayList<TreeNode>la = nodeTorootPath(root.left,val);

        if(la.size() > 0) {
            la.add(root);
            return la;
        }

        ArrayList<TreeNode>ra = nodeTorootPath(root.right,val);

        if(ra.size() > 0) {
            ra.add(root);
            return ra;
        }

        return new ArrayList<>();
    }

    public void kDown(TreeNode node,int k,TreeNode prhbt) {
        if(node == null || node == prhbt) {
            return;
        }

        if(k == 0) {
            ans.add(node.val);
            return;
        }

        kDown(node.left,k-1,prhbt);
        kDown(node.right,k-1,prhbt);
    }


    ArrayList<Integer>ans;
    public ArrayList<Integer> solve(TreeNode root, int val, int k) {
        ans = new ArrayList<>();
        ArrayList<TreeNode>l1 = nodeTorootPath(root,val);
        TreeNode prhbt = null;

        for(int i=0; i < l1.size();i++) {
            TreeNode node = l1.get(i);
            kDown(node,k-i,prhbt);
            prhbt = node;
        }

        return ans;
    }
}
```

Doubts



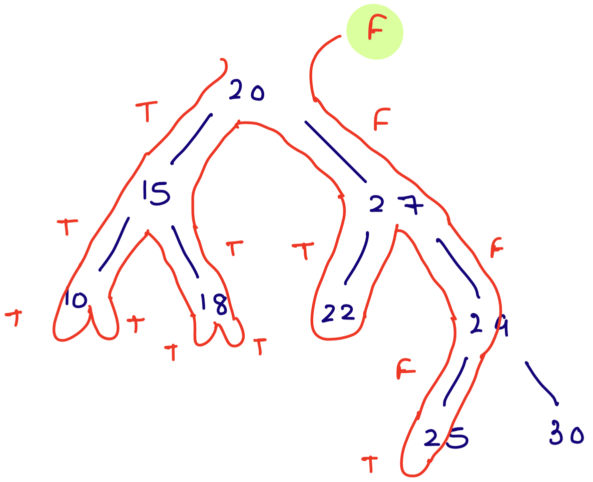
node, min, max

```
travel (node.left, min, node.val);

travel (node.right, node.val, max);
```

```
int prev = -∞;

boolean helper (Node node) {
    if (node == null) {
        return true;
    }

    boolean la = helper (node.left);
    if (la == false) { return false; }
    if (prev >= node.val) {
        return false;
    }
    prev = node.val;

    boolean ra = helper (node.right);

    return ra;
}
```

```
boolean isBST (Node node)
        prev = -∞;
        return helper (node);
}
```

```
boolean helper (Node  node) {
    if( node == null ) {
        return true;
    }
    boolean la = helper (node.left) ;
    if (la == false) { return false;}
    if( prev >= node.val ) {
        return false;
    }
    prev = node.val ;

    boolean ra = helper (node.right);

    return ra;
}
```



prev = ~~-∞~~ ~~10~~ ~~15~~ ~~18~~ ~~20~~ ~~22~~
27