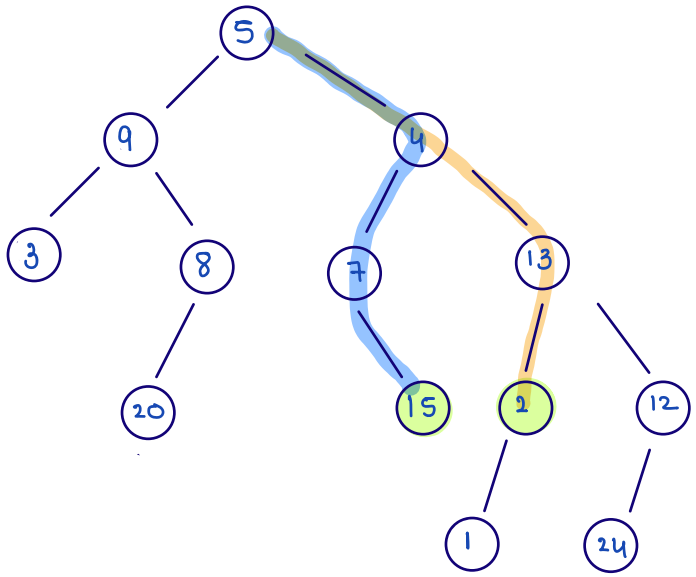
Agenda

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.

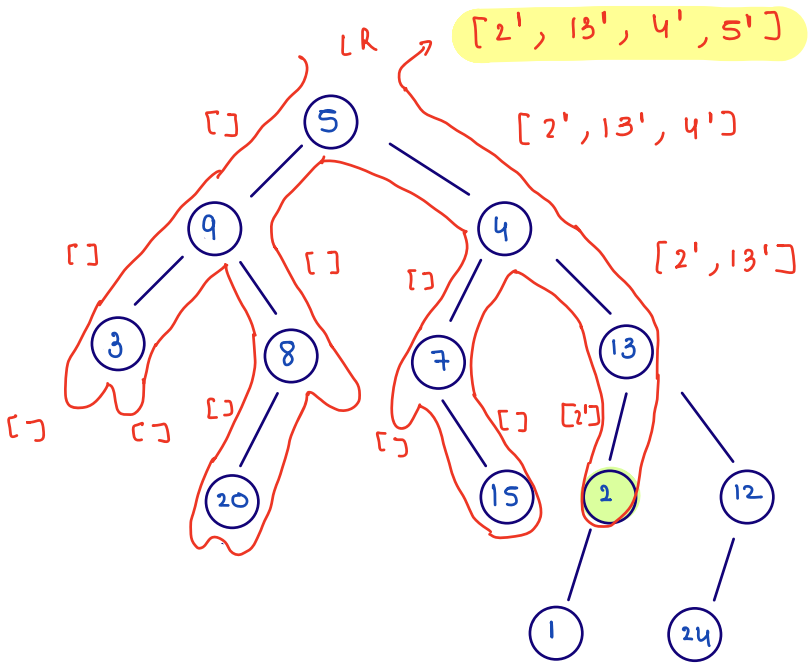


val = 2

ans = 2 13 4 5

val = 15

ans = 15 7 4 5



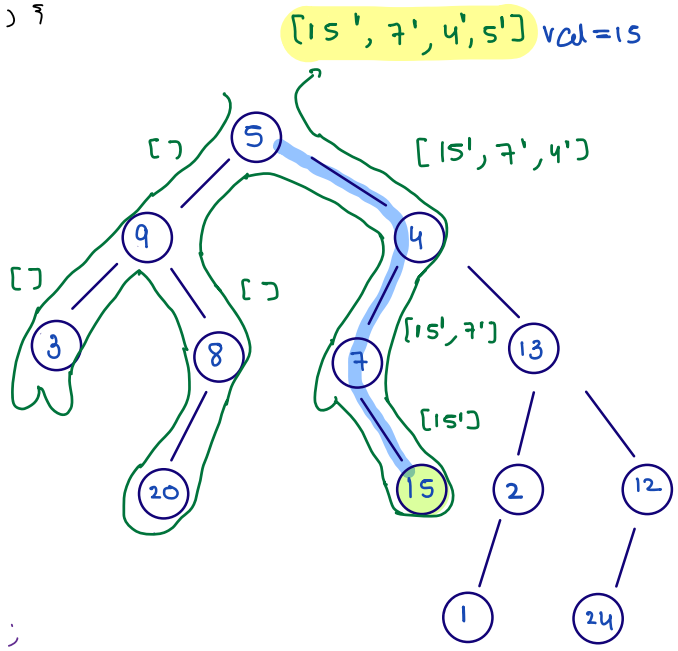
val = 2

ans =

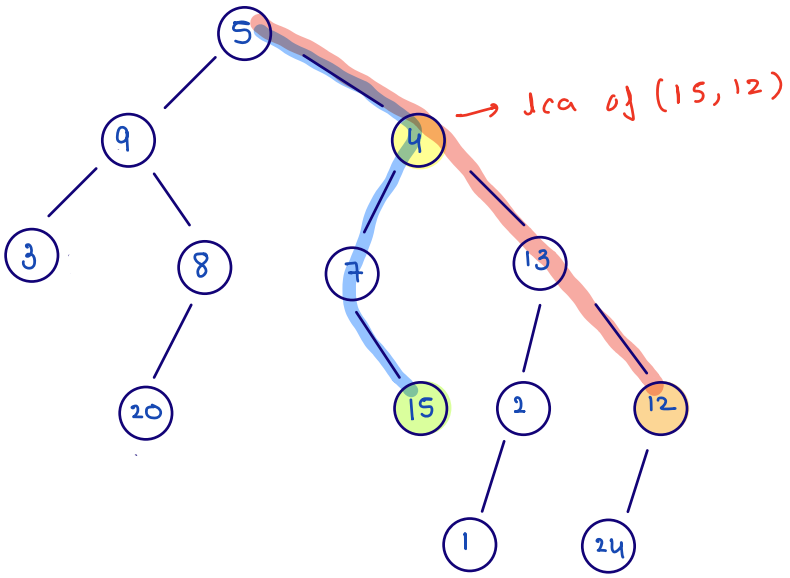
AL <Node>

```
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<>();
}
```

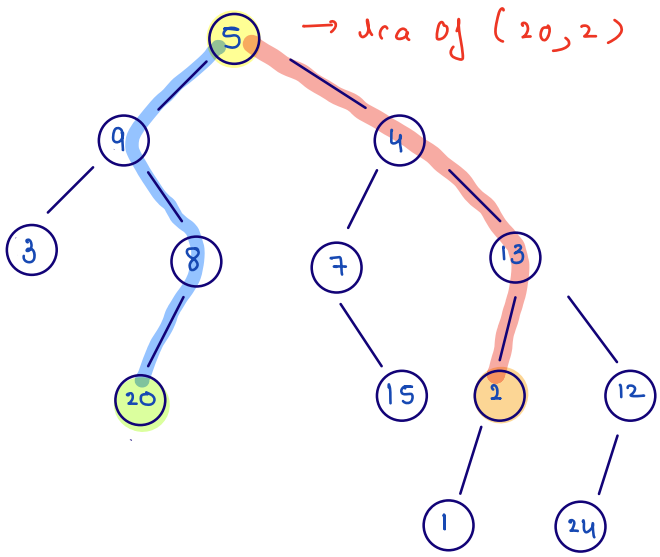


Q.2 Given root of a BT and two nodes (their values). Find LCA (lowest common ancestor) of these two nodes.



v1 = 15

v2 = 12



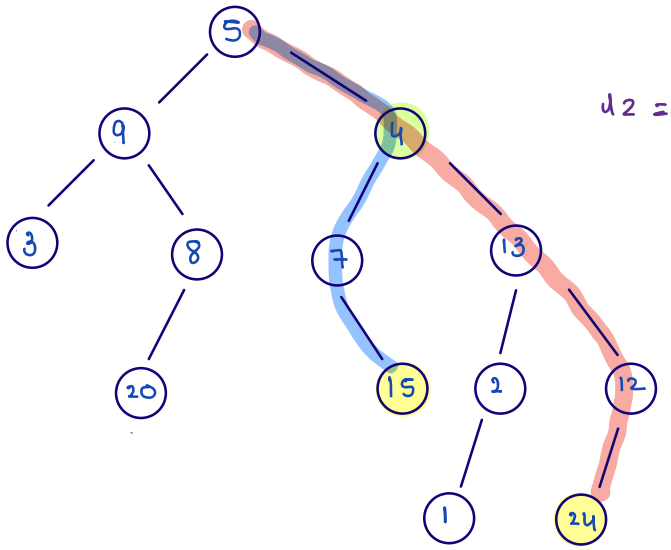
v1 = 20

v2 = 2

```
u1 = nodeToRootPath(root, v1);
u2 = nodeToRootPath(root, v2);
```

u1 = [20, 8, 9, 5]

u2 = [2, 13, 4, 5]



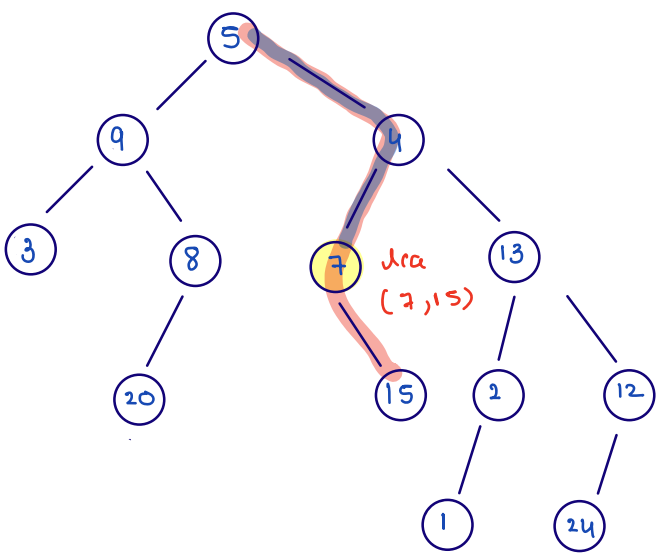
l1 = [15, 7, 4, 5]

i

l2 = [24, 12, 13, 4, 5]

j

ans =



v1 = 7

v2 = 15

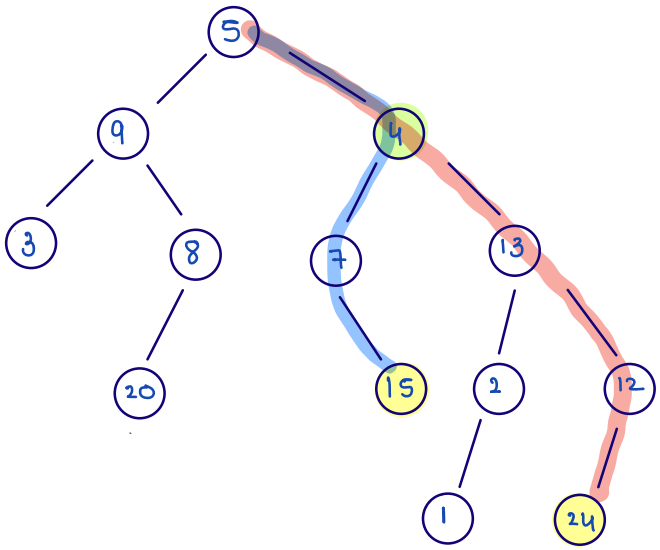
i

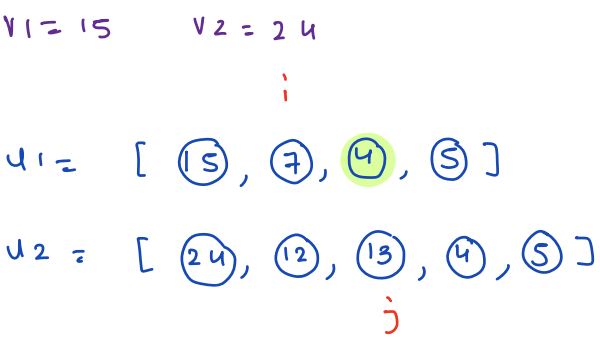
l1 = [7, 4, 5]

l2 = [15, 7, 4, 5]

j

```java
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
}
```

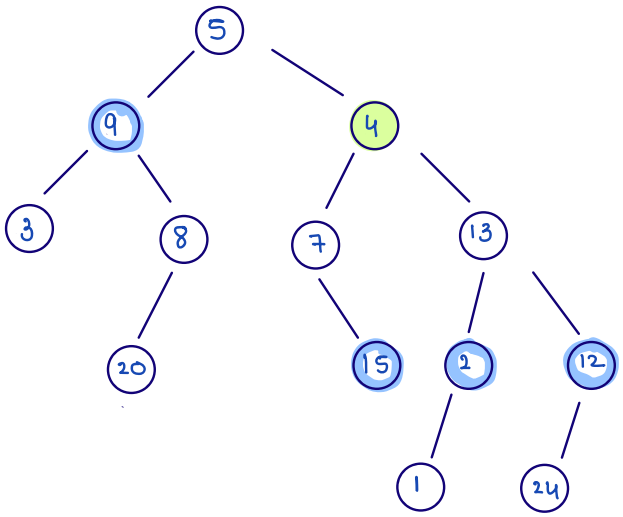




```java
public class Solution {
    public ArrayList<TreeNode> nodeTorootPath(TreeNode root,int val) {
        if(root == null) {
            return new ArrayList<>();
        }

        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; //or l2.get(j+1).val
    }
}
```

LCA complete code

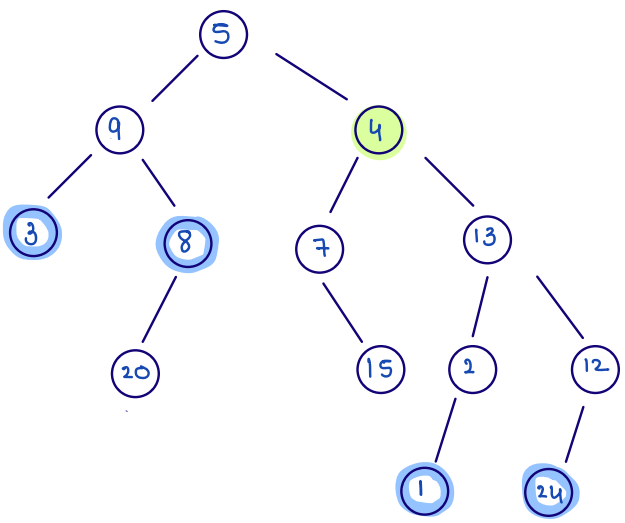
Q.3 Given root of a Binary tree, a val and a distance k. Return AL<Integers> consisting all the nodes that are k distance away from node containing the given val.



val = 4

k = 2 (dist)

ans = [15, 2, 12, 9]



val = 4

k = 3 (dist)

ans = [1, 24, 3, 8]

kdown(node, k)

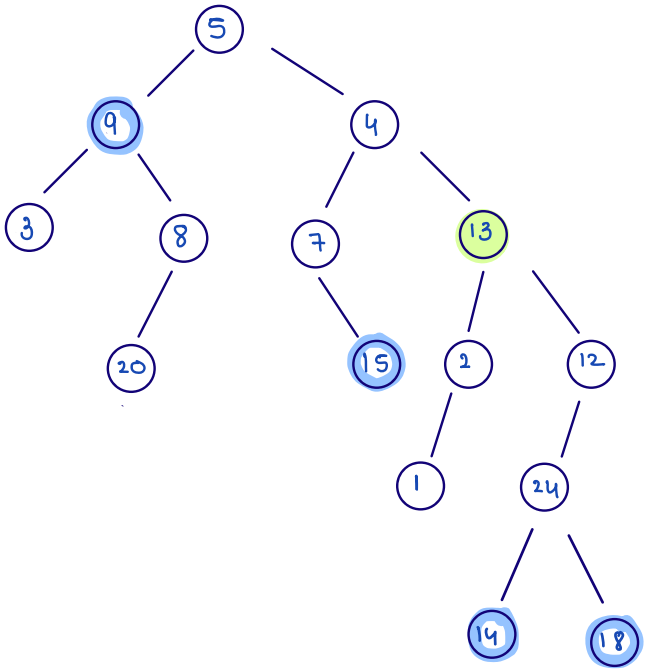
↳ all the nodes k down from node

dl = [④, ⑤]

↓ ↓

kdown (④, 3) => 1, 24

kdown (⑤, 2) => 3, 8, ~~7, 13~~



val = 13

k = 3 (dist)

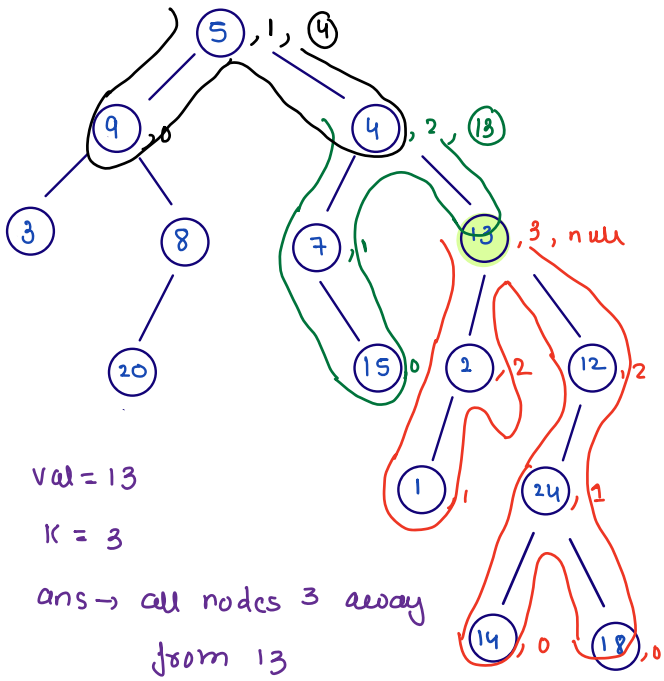
ans = all nodes 3 away from (13)

↳ [14, 18, 15, 9]

ul = [(13) (4) (5)]

	(13)	(4)	(5)
Kdown(k)	3	2	1
prhbt	null	(13)	(4)
ans	14, 18	15	9

```java
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

ans → all nodes 3 away from 13

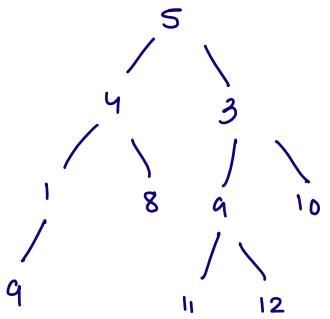
	0	1	2
l1 =	[13	4	5]
prhbt	null	13	4
	kDown (13, 3, null)	kDown (4, 2, 13)	kDown (5, 1, 4)

ans = [14, 18, 15, 9]

```java
public class Solution {
    public 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;
    }
}
```

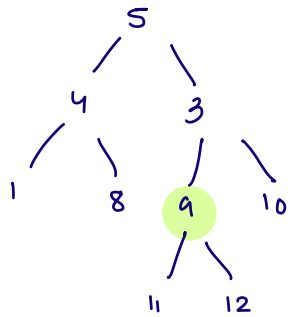
K- away complete code

Doubts



max = 3

[9 , 11 , 12]



max = 3

[11 , 12]

LCA ⇒ 9