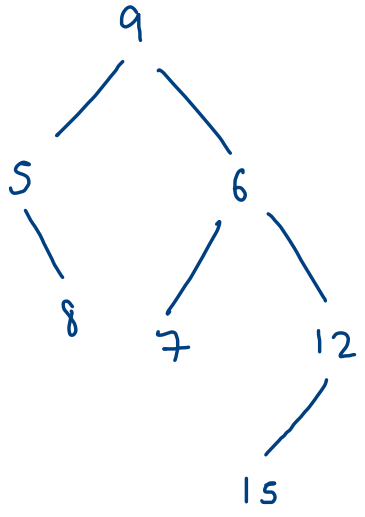
105. Construct Binary Tree from Preorder and Inorder Traversal



ps ↑ ps+cls pe

pre : 9 5 8 6 7 12 15

0 1 2 3 4 5 6

in : 5 8 9 7 6 15 12

is k ie

(ps, pe, is, ie)

ps, pe, is, ie

9

ps+1, ps+cls, is, k-1 ps+cls+1, pe, k+1, ie

pre: NLR

in: LNR

cls = k-is

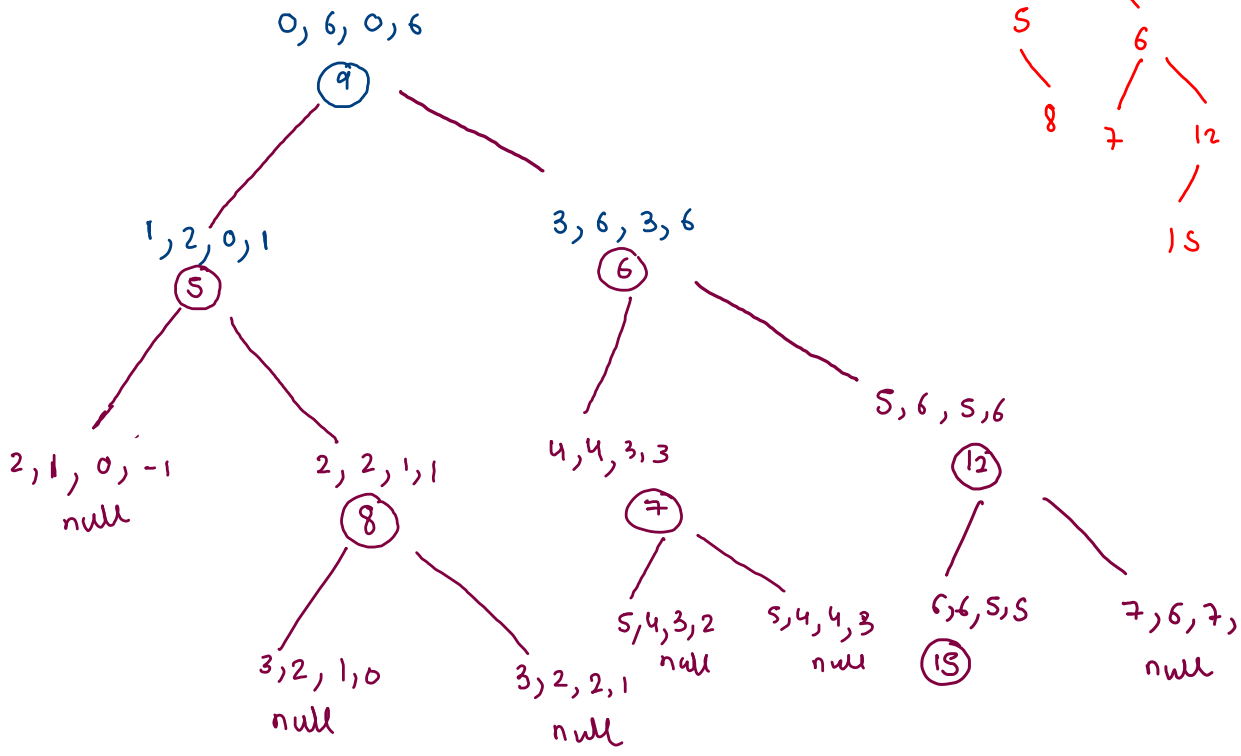
pre : 9 5 8 6 7 12 15

0 1 2 3 4 5 6

in : 5 8 9 7 6 15 12

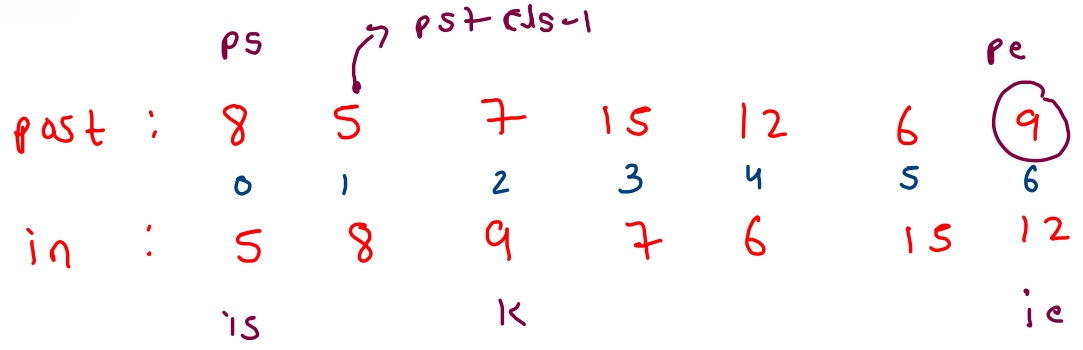
(ps, pe, is, ie)

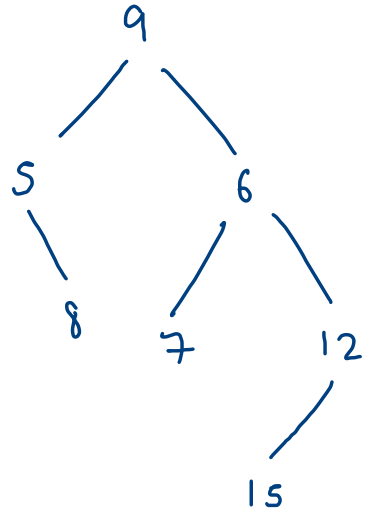
```
public TreeNode helper(int[]pre,int[]in,int ps,int pe,int is,int ie) {

    TreeNode node = new TreeNode(pre[ps]);

    int k = -1;
    for(int i=is; i <= ie;i++) {
        if(in[i] == node.val) {
            k = i;
            break;
        }
    }

    int cls = k - is; //count of left subtree elements(wrt node)

    node.left = helper(pre,in,ps + 1,ps + cls,is,k-1);
    node.right = helper(pre,in,ps + cls + 1,pe,k+1,ie);

    return node;
}
```



106. Construct Binary Tree from Inorder and Postorder Traversal

LRN (post) LNR (in)





ps, pe, is, ie

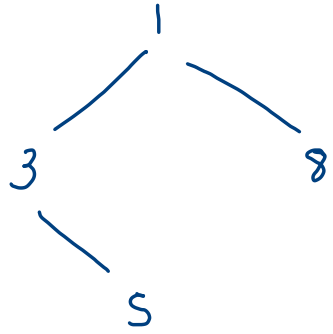
ps, pe, is, ie

9 → post[pe]

cls = ik - is

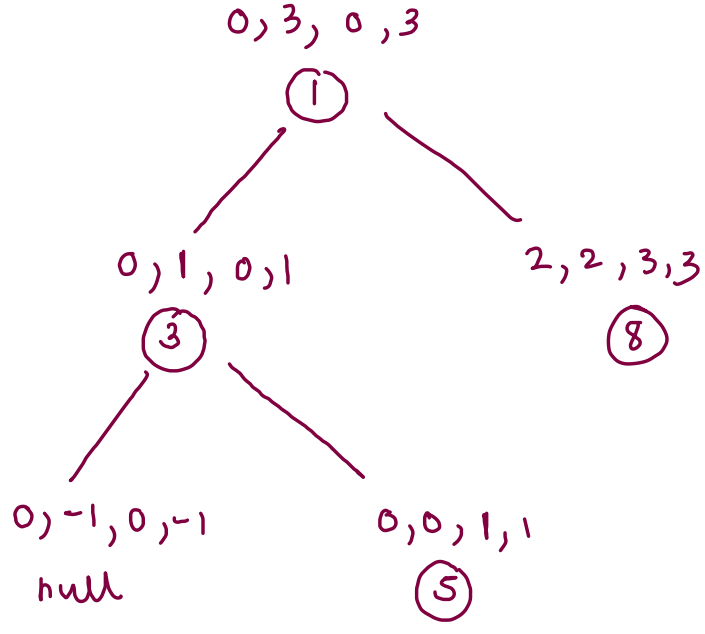
ps, pst+cls-1, is, ik-1

pst+cls, pe-1, ik+1, ie



post : 5 3 8 1
0 1 2 3
in : 3 5 1 8

```java
public TreeNode helper(int[]post,int[]in,int ps,int pe,int is,int ie) {

    if(ps > pe) {
        return null;
    }

    TreeNode node = new TreeNode(post[pe]);

    int k = -1;

    for(int i = is; i <= ie;i++) {
        if(in[i] == node.val) {
            k = i;
            break;
        }
    }

    int cls = k - is;

    node.left = helper(post,in,ps,ps + cls - 1,is, k - 1);
    node.right = helper(post,in,ps + cls,pe - 1,k + 1,ie);

    return node;
}
```

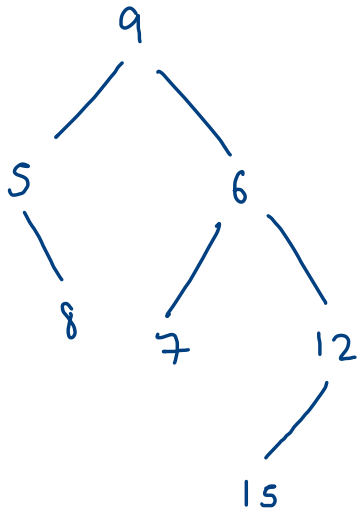


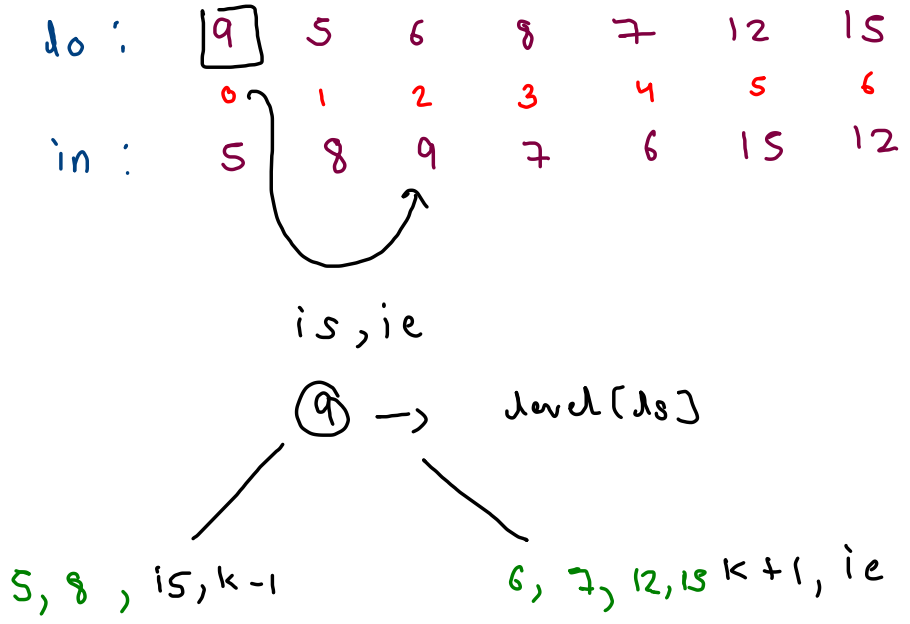
ps, pe, is, ie

k = 0

cls = 0

Construct tree from Inorder and LevelOrder





po:	9	5	6	8	7	12	15
	0	1	2	3	4	5	6
in:	5	8	9	7	6	15	12

inorder -> virtually shorten

levelorder -> reality shorten

Inorder mapping

5 -> 0

8 -> 1

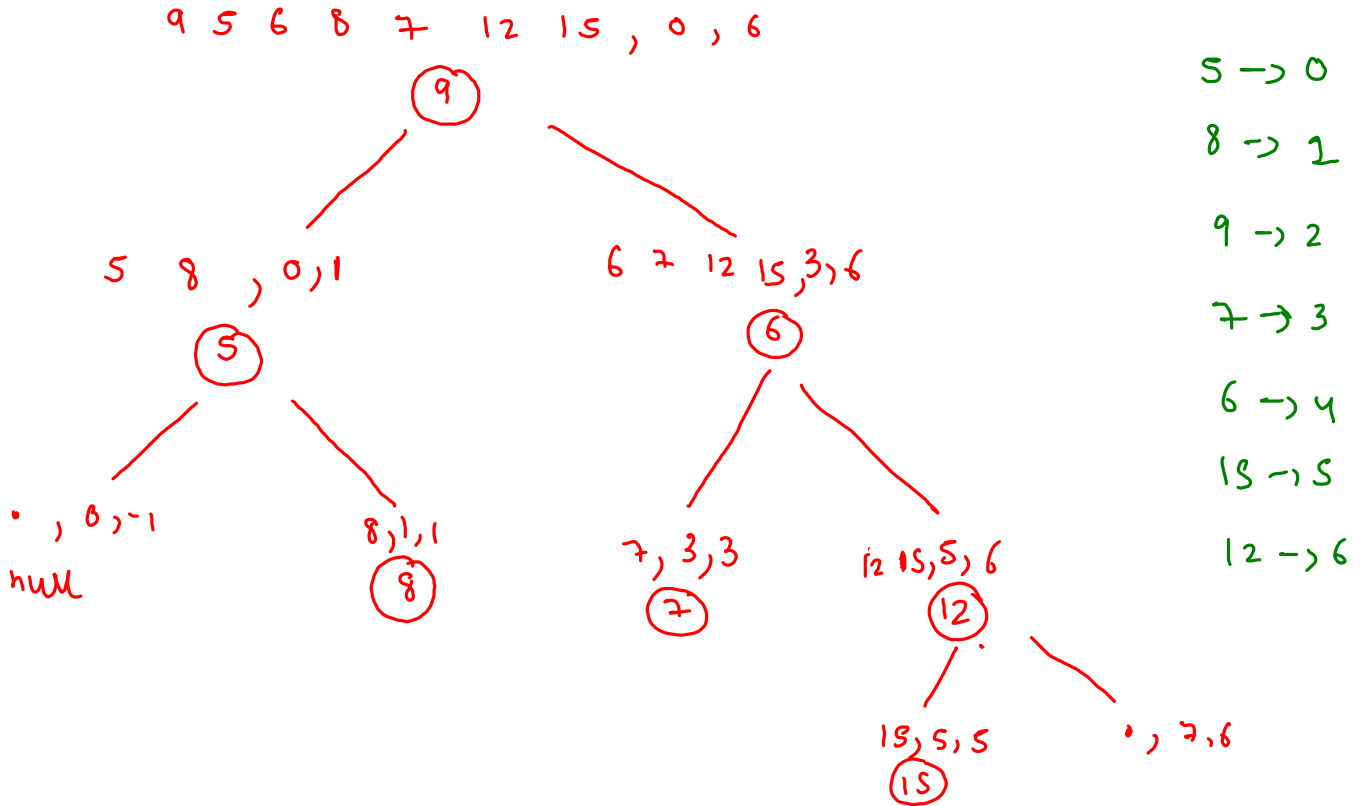
9 -> 2

7 -> 3

6 -> 4

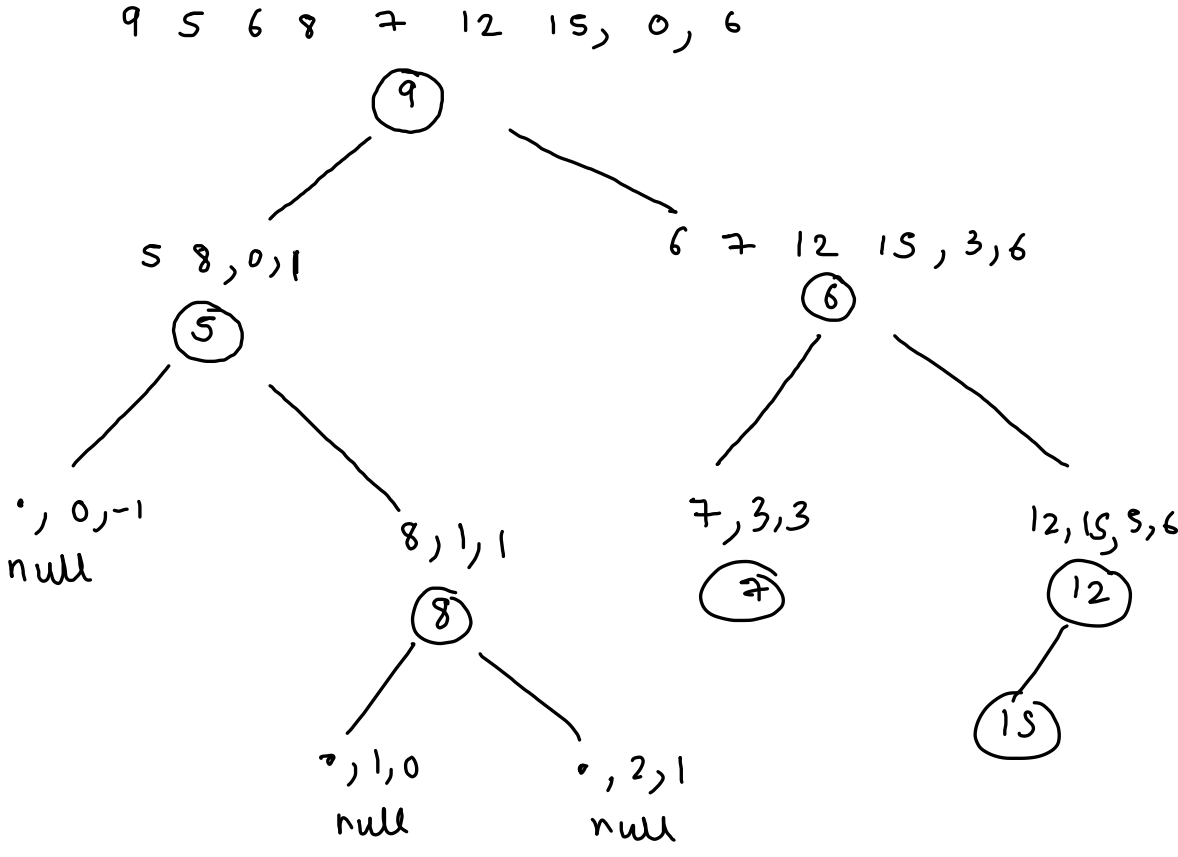
15 -> 5

12 -> 6



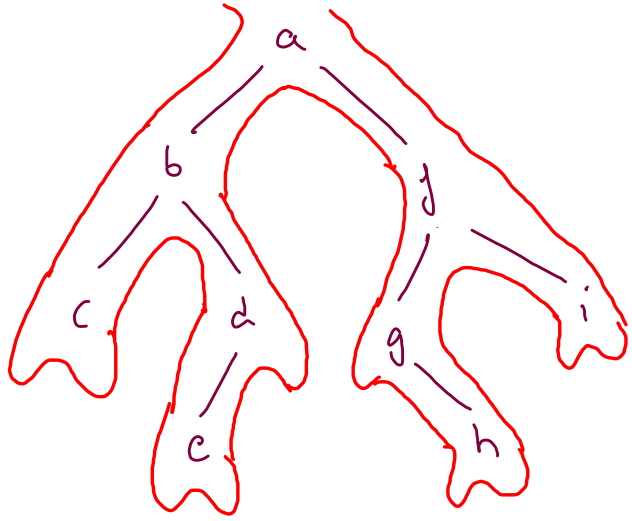
lo :	9	5	6	8	7	12	15
	0	1	2	3	4	5	6
in :	5	8	9	7	6	15	12

```
Node helper(int[]level,int[]inorder,int is,int ie) {
    if(is > ie) {
        return null;
    }

    Node node = new Node(level[0]);
    int k = map.get(node.data);

    int ls = k - is;
    int rs = level.length - ls - 1;

    int[]llo = new int[ls]; //left level order
    int[]rlo = new int[rs]; //right level order

    int p = 0,q = 0;

    for(int i = 1; i < level.length;i++) {
        int idx = map.get(level[i]);

        if(idx < k) {
            llo[p++] = level[i];
        }
        else if(idx > k){
            rlo[q++] = level[i];
        }
    }

    node.left = helper(llo,inorder,is,k-1);
    node.right = helper(rlo,inorder,k+1,ie);

    return node;

}
```



297. Serialize and Deserialize Binary Tree

```
// Encodes a tree to a single string.
public String serialize(TreeNode root) {

}

// Decodes your encoded data to tree.
public TreeNode deserialize(String data) {

}
```



preorder :

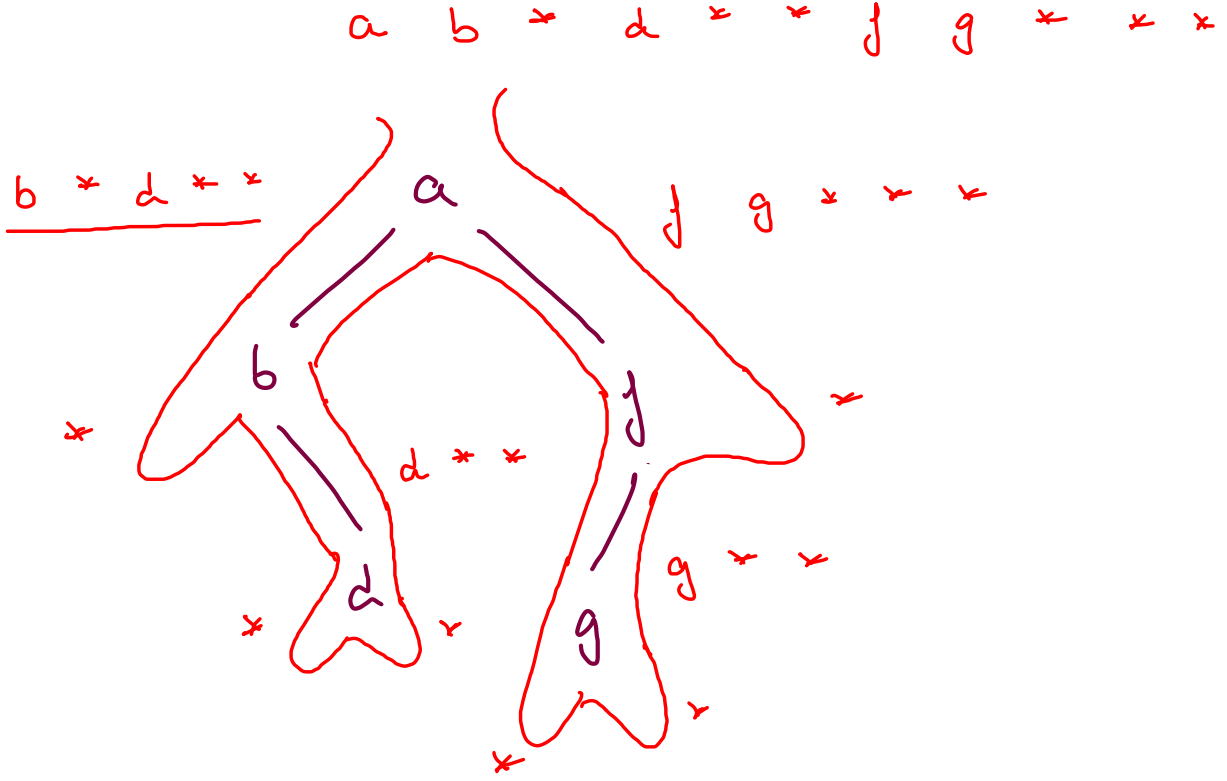
a b c * * d e * * *

f g * h * * i * *

then desiarilse

(L1 Binary tree construction code)

```java
// Encodes a tree to a single string.
public String serialize(TreeNode root) {
    if(root == null) {
        return "*";
    }

    String ls = serialize(root.left);
    String rs = serialize(root.right);

    return root.val + " " + ls + " " + rs;
}
```



a b * d * * f g * * *

```java
// Decodes your encoded data to tree.
public TreeNode deserialize(String data) {
    k = 0;
    String[]arr = data.split(" ");
    return helper(arr);
}

int k;
public TreeNode helper(String[]arr) {
    if(arr[k].equals("*") == true) {
        k++;
        return null;
    }
    else {
        int data = Integer.parseInt(arr[k]);
        k++;

        TreeNode node = new TreeNode(data);

        node.left = helper(arr);
        node.right = helper(arr);

        return node;
    }

}
```

arr : [a, b, *, d, *, *, f, g, *, *, *]