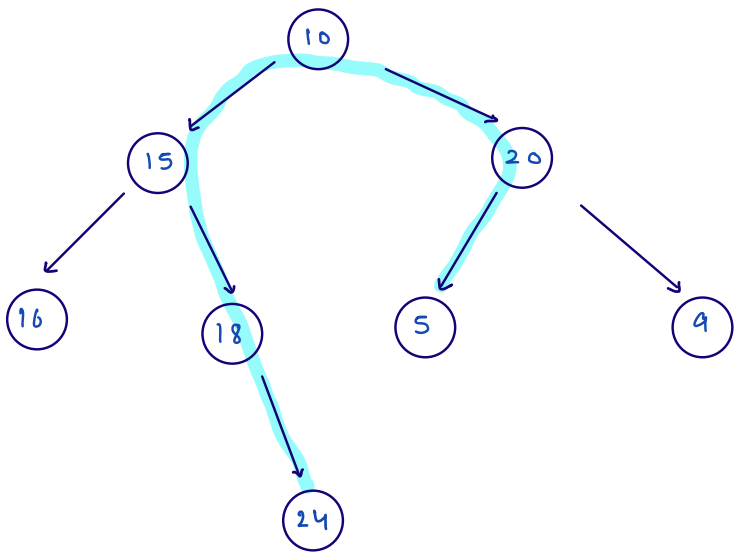
## Agenda

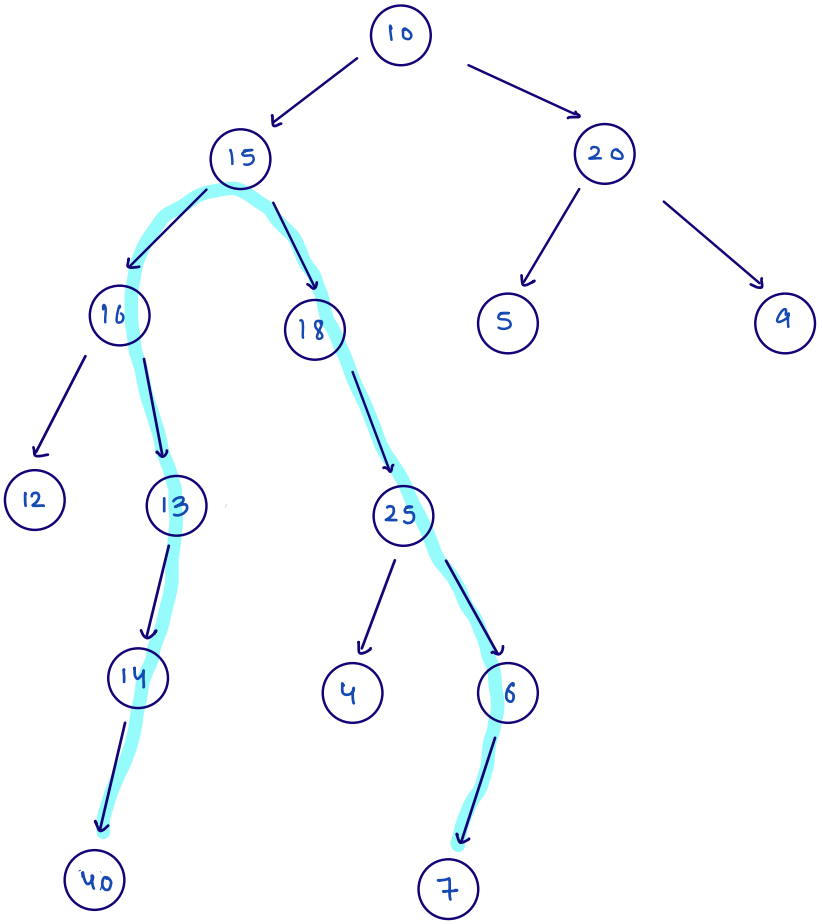
1) Diameter of Binary tree
2) Serialize & Deserialize Binary tree
3) TreeMap Introduction & usage

Q.1 Given root of a binary tree, find its diameter.

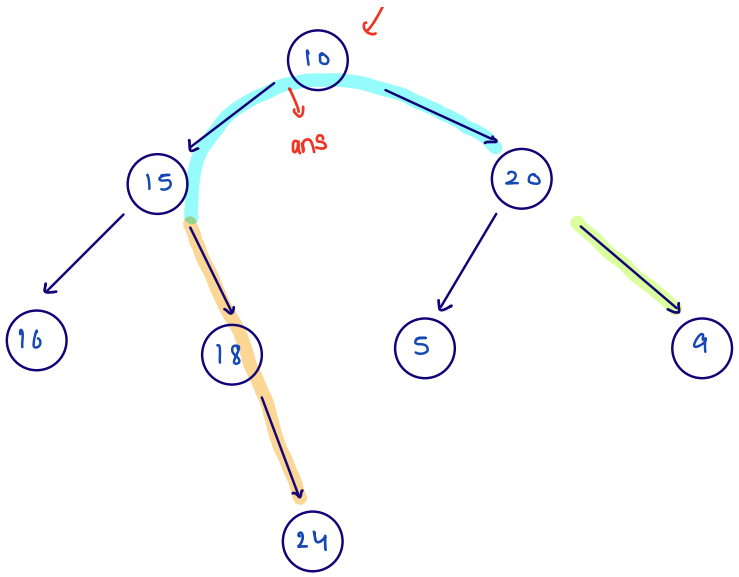
Note: The diameter of binary tree is the length of the longest path blw any two nodes. This path may or may not pass through the root.




diameter = 5



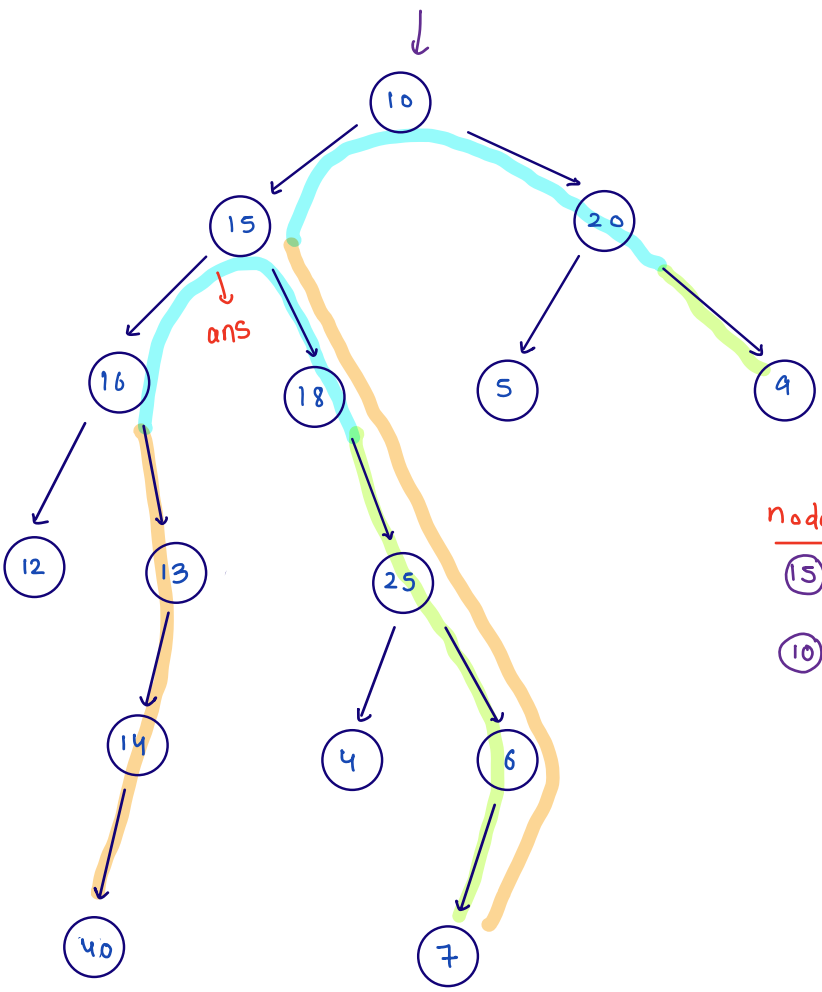

diameter = 8




lh = distance b/w left child and deepest node in left subtree

rh = distance b/w right child and deepest node in right subtree.

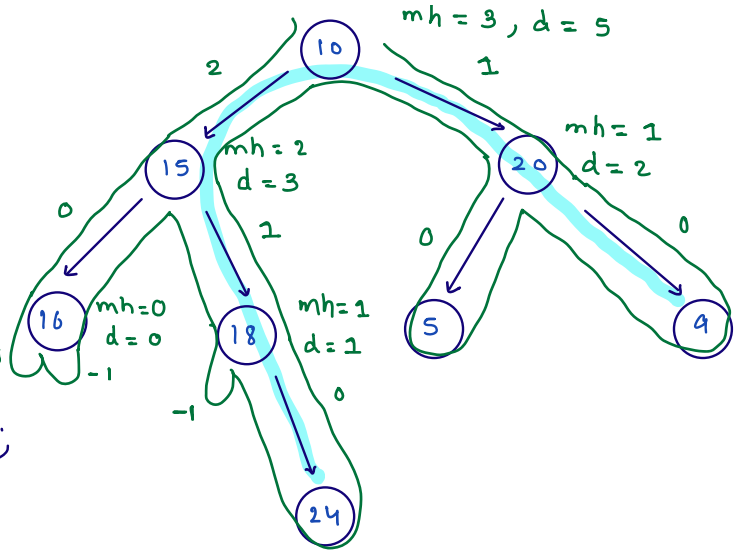
$d = lh + rh + 2$




| node | lh | rh | d |
|---|---|---|---|
| (15) | 3 | 3 | 8 |
| (10) | 4 | 1 | 7 |

```
int maxDist;
int diameter (Node root) {
    maxDist = 0;
    height (root);
    return maxDist;
}
int height (Node root) {
    if (root == null) {
        return -1;
    }
    int lh = height (root.left);
    int rh = height (root.right);
    int mh = Math.max (lh, rh) + 1;
    int d = lh + rh + 2;
    if (d > maxDist) {
        maxDist = d;
    }
    return mh;
}
```

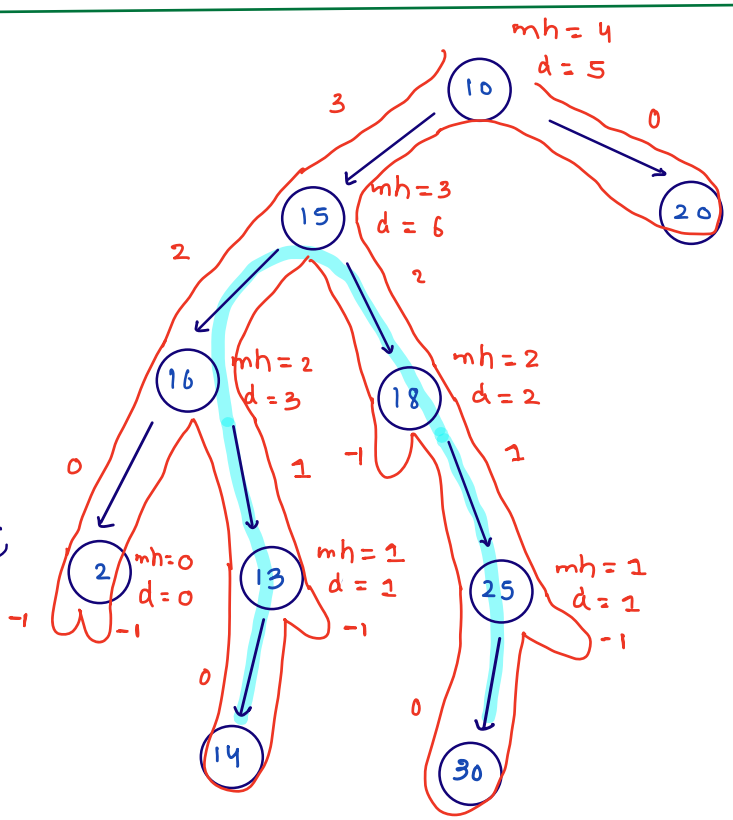
```
int height(Node root) {
    if (root == null) {
        return -1;
    }
    int lh = height(root.left);
    int rh = height(root.right);
    int mh = Math.max(lh, rh) + 1;
    int d = lh + rh + 2;
    if (d > maxDist) {
        maxDist = d;
    }
    return mh;
}
```




maxDist = ~~0 1 3~~ 5

---

```
int height(Node root) {
    if (root == null) {
        return -1;
    }
    int lh = height(root.left);
    int rh = height(root.right);
    int mh = Math.max(lh, rh) + 1;
    int d = lh + rh + 2;
    if (d > maxDist) {
        maxDist = d;
    }
    return mh;
}
```

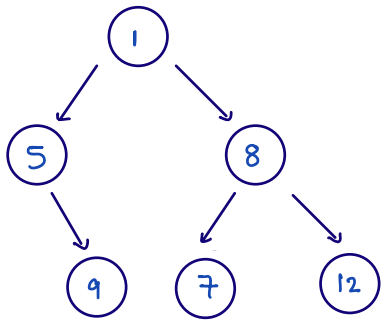



maxDist = ~~0 1 3~~ 6

https://leetcode.com/problems/diameter-of-binary-tree/description/

```java
class Solution {
    int maxDist;
    public int diameterOfBinaryTree(TreeNode root) {
        maxDist = 0;
        height(root);
        return maxDist;
    }

    public int height(TreeNode root) {
        if(root == null) {
            return -1;
        }

        int lh = height(root.left);
        int rh = height(root.right);
        int mh = Math.max(lh,rh) + 1;

        int d = lh + rh + 2;
        if(d > maxDist) {
            maxDist = d;
        }

        return mh;
    }
}
```

Q. 2 Serialize and Deserialize a Binary tree.




Serialization: convert Binary tree into a string.

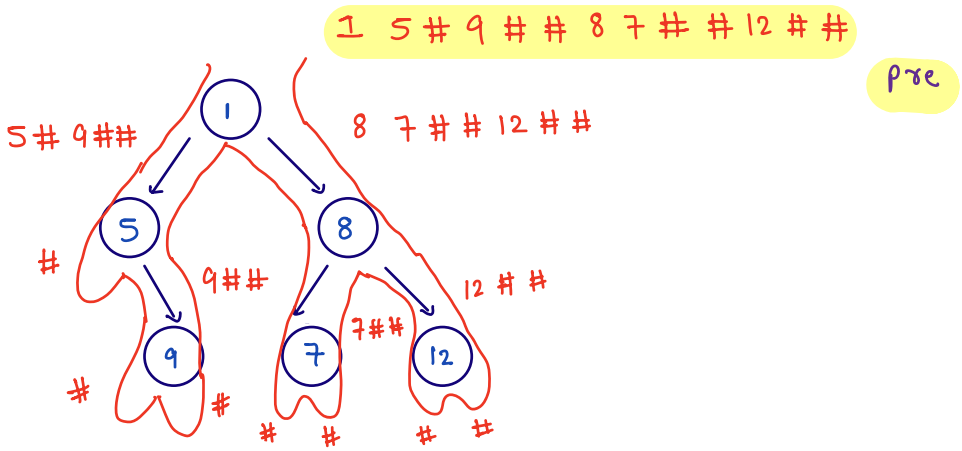
Deserialization: take your serialized string and from this string convert back to binary tree.

```
String Serialization (Node root) {
|
}
```

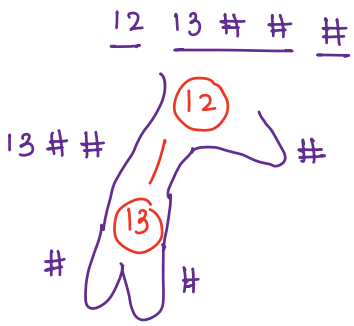
```
Node Deserialization (String data) {
|
}
```

your serialized string

**Serialization** (convert binary tree to string)

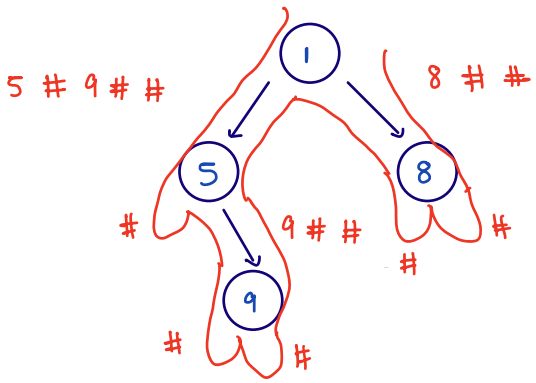



```
String Serialization (Node root) {
    if(root == null) {
        return "#";
    }
    String la = Serialization(root.left);
    String ra = Serialization(root.right);
    String ma = root.val + " " + la + " " + ra;
    String ma;
}
```



Dry run

1 5 # 9 # # 8 # #




```
String Serialization (Node root) {
    if (root == null) {
        return "#";
    }
    String la = serialization (root.left);
    String ra = Serialization (root.right);
    String ma = root.val + " " + la + " " + ra;
    String ma;
}
```

Deserialization : from serialized string, construct the tree

1 5 # 9 # # 8 # #

"1 5 # 9 # # 8 # #"

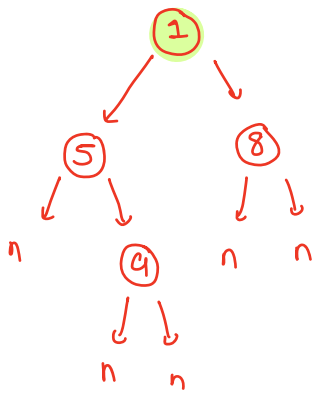
["1", "5", "#", "9", "#", "#", "8", "#", "#"]

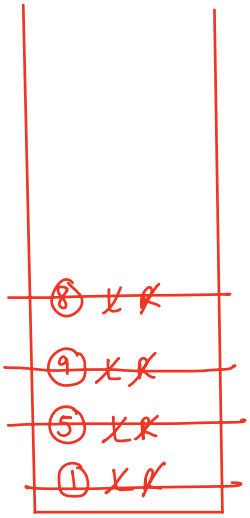
idx

["1", "5", "#", "9", "#", "#", "8", "#", "#"]

if arr[idx] is # → return null

else
↳ create node
build left subtree
build right subtree




int idx;

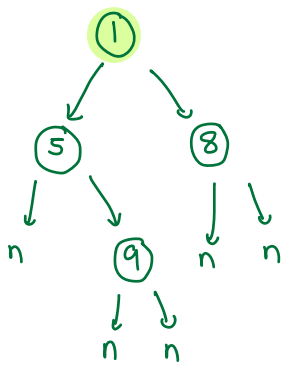
```
Node Deserialize(String data) {
    String[] arr = data.split(" ");
    idx = 0;
    return construct(arr);
}
Node construct(String[] arr) {
    if(arr[idx].equals("#")) {
        idx++;
        return null;
    }
    else {
        int val = Integer.parseInt(arr[idx]);
        Node nn = new Node(val);
        idx++;
        nn.left = construct(arr);
        nn.right = construct(arr);
        return nn;
    }
}
```

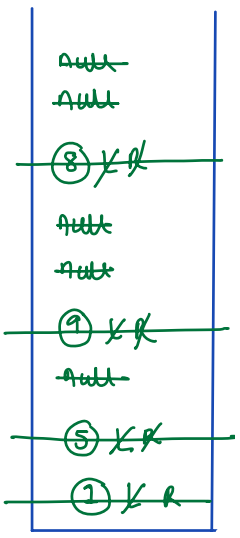
↳ serialized string

dry run

["1", "5", "#", "9", "#", "#", "8", "#", "#"]

```
Node construct (String[] arr) {
    if (arr[idx].equals("#")) {
        idx++;
        return null;
    }
    else {
        int val = Integer.parseInt(arr[idx]);
        Node nn = new Node(val);
        idx++;
        nn.left = construct(arr);
        nn.right = construct(arr);
        return nn;
    }
}
```




todo: complete dry run

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/description/

```java
public class Codec {

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

        String la = serialize(root.left);
        String ra = serialize(root.right);
        String ma = root.val + " " + la + " " + ra;

        return ma;
    }

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

    int idx;
    public TreeNode construct(String[]arr) {
        if(arr[idx].equals("#")) {
            idx++;
            return null;
        }
        else {
            int val = Integer.parseInt(arr[idx]);
            TreeNode nn = new TreeNode(val);
            idx++;

            nn.left = construct(arr);
            nn.right = construct(arr);

            return nn;
        }
    }
}
```

## Introduction to TreeMap

↳ TreeMap is a **sorted** HashMap

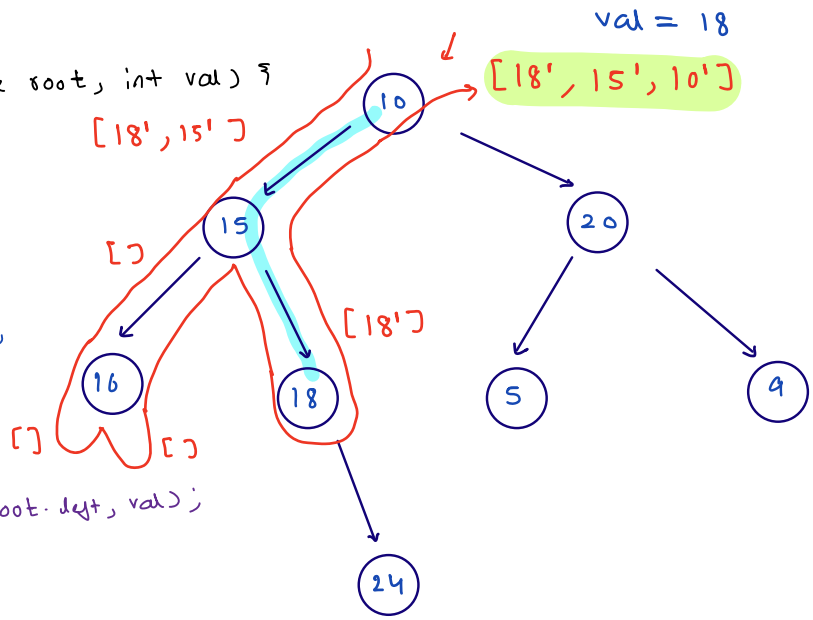
└ based on key

| HashMap | TreeMap |
| --- | --- |
| put(key, val) | put(key, val) |
| get(key) | get(key) |
| containsKey(key) | containsKey(key) |
| remove(key) | remove(key) |
| TC: O(1) | ceilingKey(key) |
| | floorKey(key) |
| | TC: O(log n) |

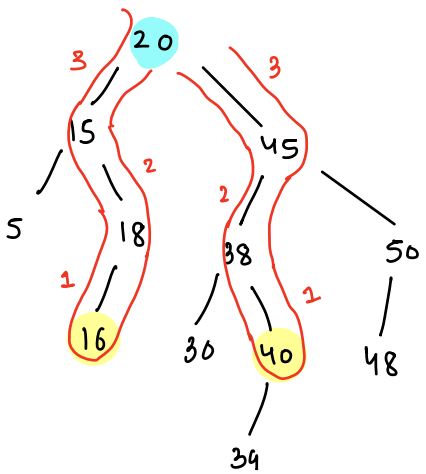
JDE link https://www.interviewbit.com/snippet/bf89ca9f2d856e24c1b9/

Doubts

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



# Distance blw two nodes




lca(16, 40) => (20)

first call

→ search((20), 16) = 3

second call

→ search((20), 40) = 3