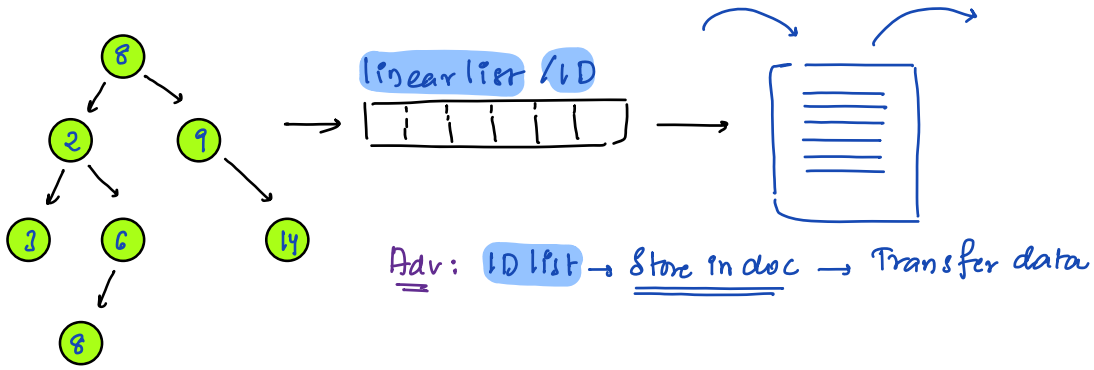
Todays Content:

- → Sereialigatim
- → deserialigatim
- → Max sum path
- → diameter

Serialization: We will store every node, & child information as well



Adv: 1D list → Store in doc → Transfer data

Idea: Inorder[] & preorder[] construct tree?

preorder[]: 8 2 3 6 8 9 14

Inorder[]: 3 2 8 6 8 9 14

Note: If data repeats, we cannot construct tree using pre[] & in[]

Serialization: Converting Tree → 1D list : Note:

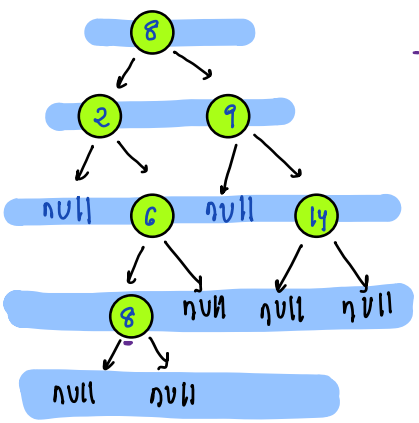
↳ preorder / postorder / levelorder / Inorder * ?

[TODO]

↳ Explain in comms

Serialization level order: store info about nodes & children

8 2 9 null 6 null 14 8 null null null null null



→ Say store list<int> : Assignments / Interviewbit

: Storing null is not possible

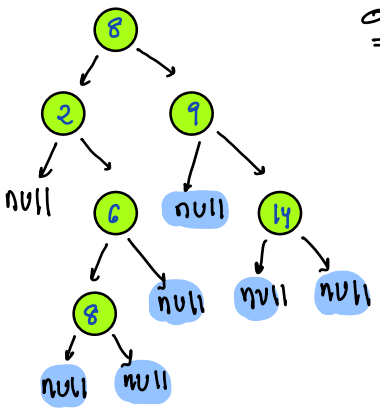
: Instead of storing null, store -1

↳ If data is +ve

→ Say store list<string> : Real world

: Storing "null" is possible

: Storing number as strings is possible



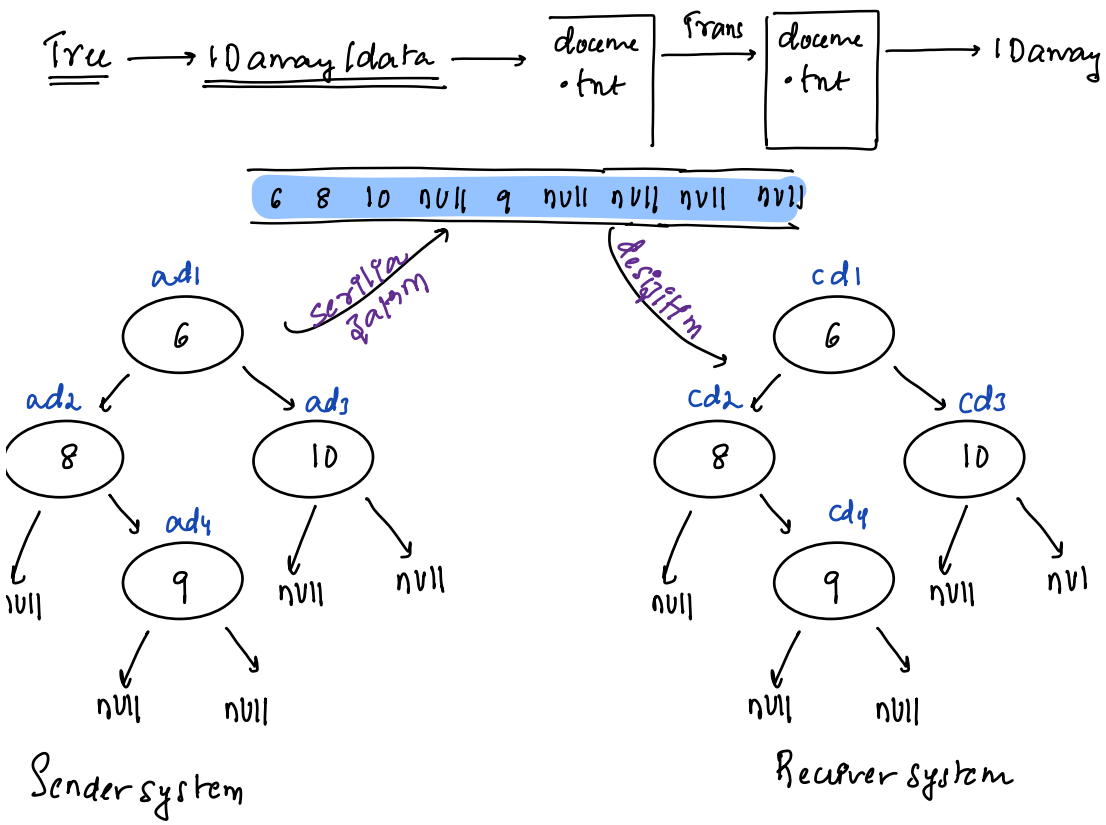
obs1: When we pop a node
→ if node not null
a) push data in list
b) Children in Queue
→ if node is null
a) push -1 in list

q: 8 | 2 9 | null 8 | null 14 | 8 null | null null | null null | null null }

li 8 2 9 -1 6 -1 14 8 -1 -1 -1 -1 -1 } <List<Int>>

List<Int> Serialization levelorder(Node root){ TC:O(N) SC:O(N)

```
List<Int> li
Queue<Node> q;
q.enque(root)
while( q.size() > 0){
    Node tmp = q.front()
    q.deque()
    if( tmp != null){
        li.insert( tmp.data)
        q.enque( tmp.left)
        q.enque( tmp.right)
    }
    else{
        li.insert(-1) // instead of null we are pushing -1
    }
}
return li
}
```


Tree
1D array [data
docume
• fnt
Trans
docume
• fnt
1Darray
6 8 10 null 9 null null null null
serializ
zation
desiglitm
ad1
6
ad2
8
ad3
10
ad4
9
null
null
null
null
null
cd1
6
cd2
8
cd3
10
cd4
9
null
null
null
null
null
Sender system
Receiver system

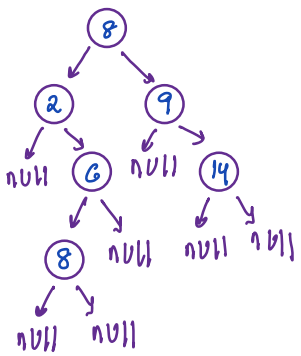
deserialization Using level order:

	0	1	2	3	4	5	6	7	8	9	10	11	12
li	8	2	9	-1	6	-1	14	8	-1	-1	-1	-1	-1
i = 13												↑	

level | ~~8~~ | ~~2~~ | ~~9~~ | ~~6~~ | ~~14~~ | ~~8~~ |

Steps:

- → Pop node from Queue say: temp
- → for temp left child is at index i
  - : insert leftchild in queue if not null
- → for temp right child is at index i+1
  - : insert rightchild in queue if not null
- → i = i+2

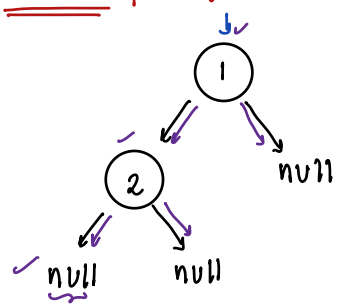


Node deserialize levelorder(List<int> l) TC:O(N) SC:O(N)

```
Node root = new Node(l[0])
Queue<node> que
que.enque(root)
i=1;
while(que.size()>0){
    Node tmp = q.front()
    q.deque()
    if(l[i] != -1){ // left child of tmp at index i
        tmp.left = new node(l[i])
        q.enque(tmp.left)
    }
    if(l[i+1] !=-1){ //right child of tmp at index i+1
        tmp.right = new node(l[i+1])
        q.enque(tmp.right)
    }
    i = i+2
}
return root;
}
```

10:23 → 10:33pm LG

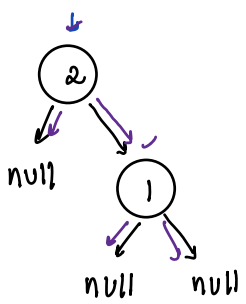
Why inorder fails?



inorder Serialization:

null 2 null 1 null

Issue: different tree can have same inorder serialization, de-serialization not possible

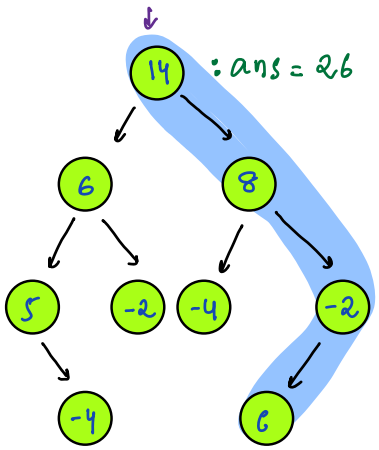


inorder Serialization:

null 2 null 1 null

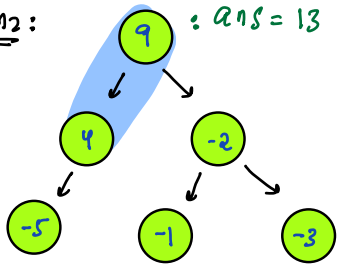
Q1: Max path sum starting from root ? : {Can end any where}

Eg1:



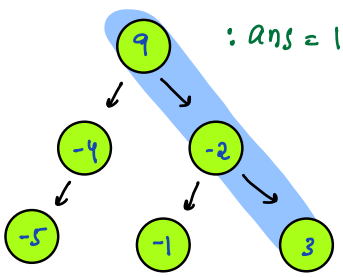
: ans = 26

Eg2:



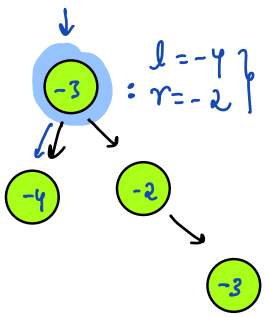
: ans = 13

Eg3:



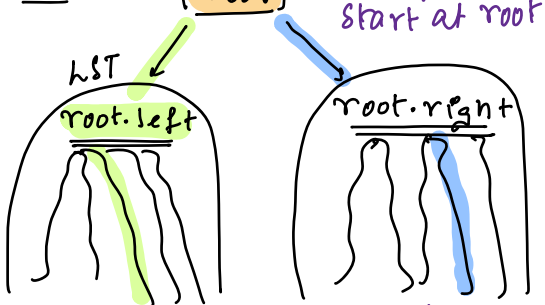
: ans = 10

Eg4:



l = -4
r = -2
ans = 3

Idea:



max path sum start at root node → root.data + max(l, r, 0)

// if l<0, r<0 ignore

l = max path sum in LST start at root.left

r = max path sum in RST start at root.right

max path in LST start at root.left

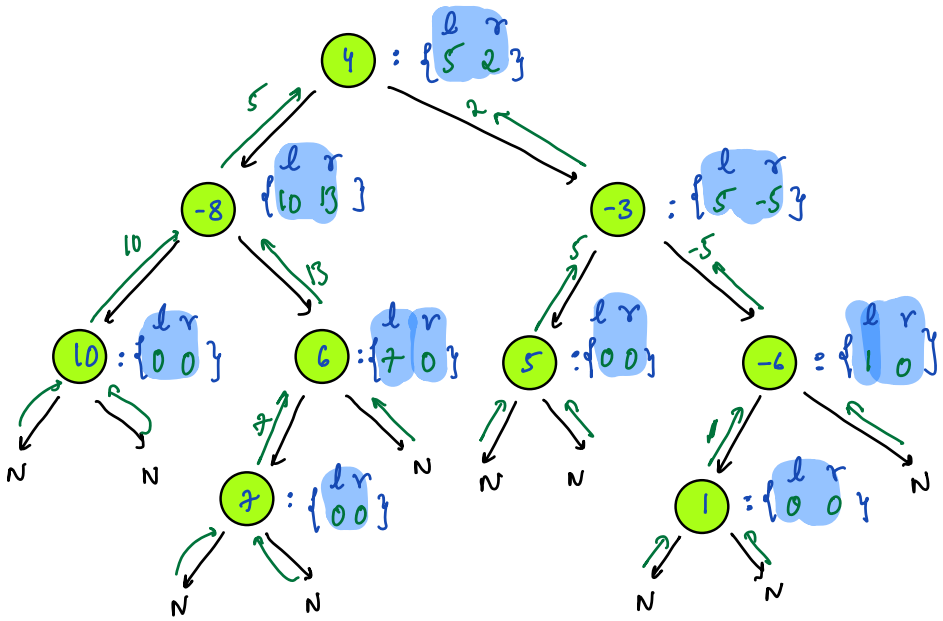
max path in RST start at root.right

Ass: Given tree, return max path sum starting at root node

```
int maxPathSumStartRoot(Node root){ TC: O(N) SC: O(H)
    if(root==null){return 0}
    int l = maxPathSumStartRoot(root.left)
    int r = maxPathSumStartRoot(root.right)
    return root.data + max3(l, r, 0)
                      // will return max of 3 numbers
}
```

Tracing: l = maxpathsum in LST start at root.left

r = maxpathsum in RST start at root.right



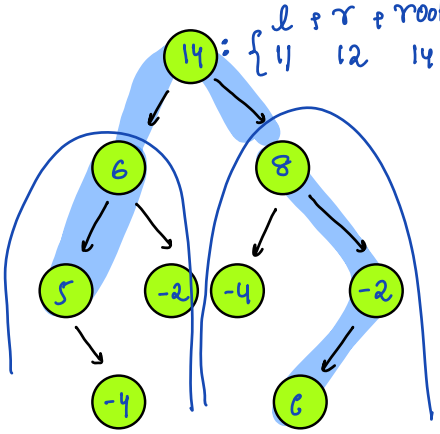
Q2) Max path sum **passing rootnode** in given Tree

→ path containg root node

Eg1: ans = 37

l + r + root.data

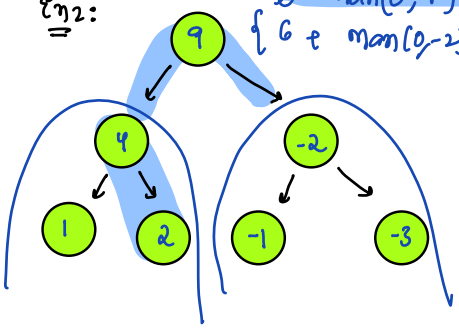
{11 12 14}



Eg2: ans = 15

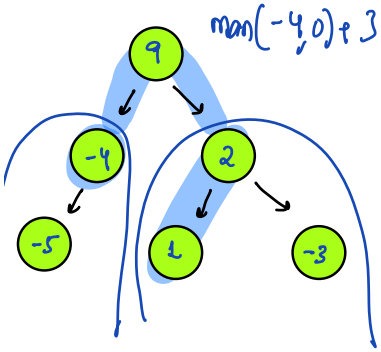
l max(0,r) + root.data

{6 + max(0,-2) + 9 = 15



Eg3: ans = 12 max(0,l) r root.data

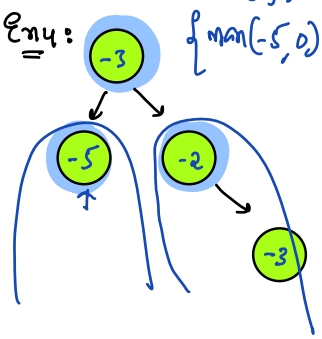
max(-4,0) + 3 + 9 = 12



ans = -3

max(l,0) + max(r,0) + root.data

Eg4: {max(-5,0) + max(-2,0) + -3} = -3

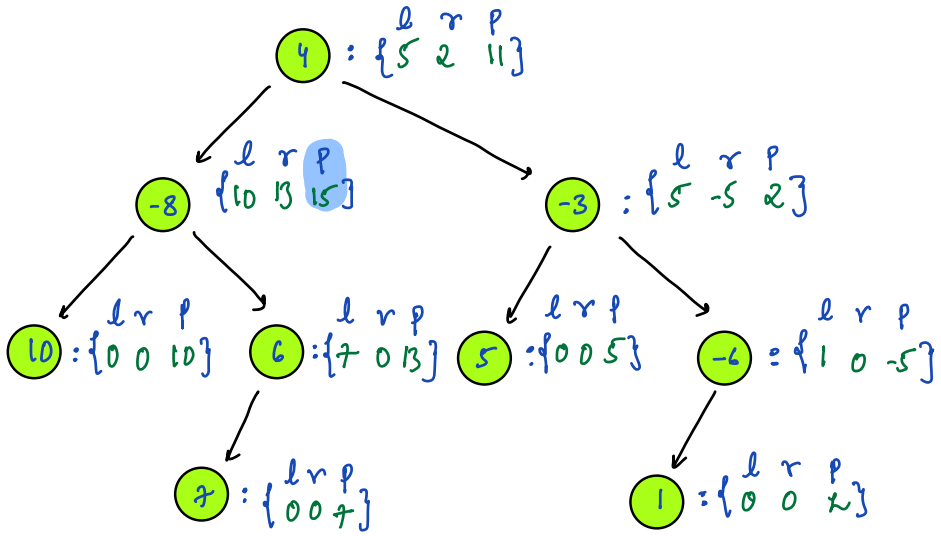


// Given root node:

```
if (root == null) {return 0}
l = maxpathsum startRoot(root.left) // max sum path start at root of LST
r = maxpathsum startRoot(root.right) // max sum path start at root of RST
val = root.data + max(0,l) + max(0,r)
//val: max path sum containg root node
}
```

Last Observation : man sum path passing via root:

= man(l, 0) + man(r, 0) + root.data



//When we call below function for every node we get man sum path start in LST & start in RST

{ Man path sum in BT }

```
int ans = INT_MIN
int manpathsum startRoot (Node root){ TC: O(N) SC: O(H)
    if (root == null) { return 0 }
    int l = manpathsum startRoot (root.left)
    int r = manpathsum startRoot (root.right)
    int p = man(l,0) + man(r,0) + root.data // man sum path passing across the root.node
    ans = man(ans, p) // man path sum in BT
    return root.data + man3(l, r, 0)
}
// ans code is return ans;
```

Obs: If all data of node, we say it is 1, and calculate man Sum path, man path in Tree - 1 = diameter of Tree // longest path in Tree