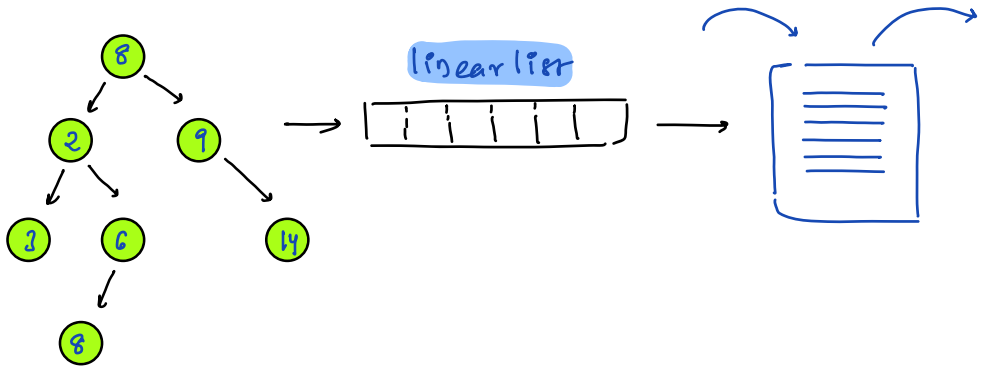
Todays Content:

- → Sereialigatim
- → deserialigatim
- → Max sum path
- → diameter {TODO}

Serialization:



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

preorder[]: 8 2 3 6 8 9 14

Inorder[]: 3 2 8 6 8 9 14

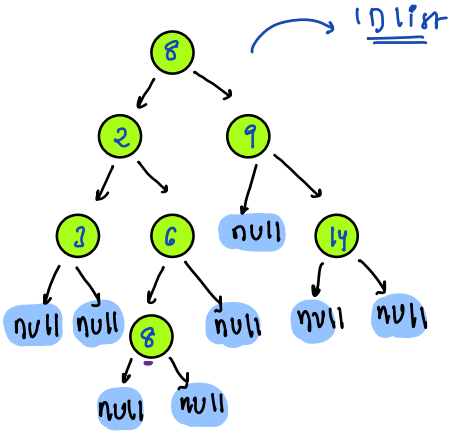
Note: If data repeats, getting tree is not possible

Serialization: Converting Tree → ID list : { For every node, store itself & its children using traversal }

TODO

→ preorder / postorder / levelorder / Inorder* ?

Seriligation level order:



ID list obs: list<int> lev; {assignments}

: We cannot store Null in list

: Instead of Null use -1,

-1 is indicating null here

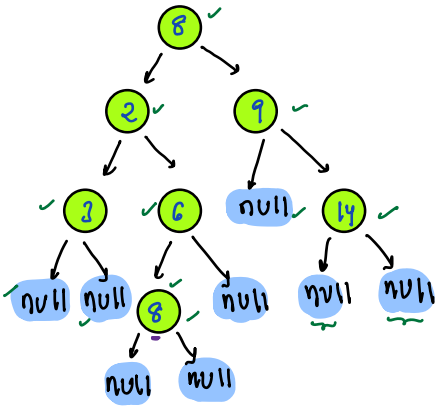
→ # of all value +ve

list<string> lev; {TODO}

: data in string is possible

: "NULL" string, indicates null

8	2	9	3	6	N	14	N	N	8	N	N	N	N	N



obs1: → We push null in Queue

→ When we delete null it won't have any children

↳ Instead of Null push -1 in list

Q: ~~8~~ | ~~2~~ | ~~9~~ | ~~3~~ | ~~6~~ | ~~N~~ | ~~14~~ | ~~N~~ | ~~N~~ | ~~8~~ | ~~N~~ | ~~N~~ | ~~N~~ | ~~N~~ | ~~N~~

li: 8 | 2 | 9 | 3 | 6 | -1 | 14 | -1 | -1 | 8 | -1 | -1 | -1 | -1 | -1

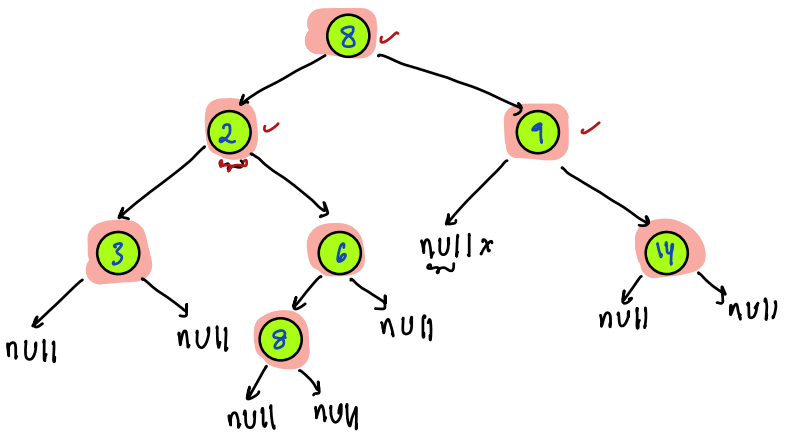
```
List<Int> Serialization (Node root){   TC: O(N)  SC: O(N)
    List<Int> l
    Queue<Node> q;
    q.enque(root)
    while(q.size() > 0){
        Node t = q.front()
        q.deque()
        if(t != null){
            l.add(t.data)
            q.enque(t.left)
            q.enque(t.right)        } → // They can be null
        }
        else{ // t is null
            l.add(-1)
            // no children to push inside queue
        }
    }
    return l;
}
```

deserialization Using level order:

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

i = 15 ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

→ Obs: Construct Tree & fill its level at same time



level: ~~8~~ ~~2~~ ~~9~~ 3 6 14 ~~8~~

```
Node deserialize(List<int> l) // returning root node of Tree
    Node root = new Node(l[0])
    Queue<node> que;
    q.enque(root)
    int i=1
    while(q.size()>0){
        Node tmp = q.front()
        q.deque()
        // fill children for tmp.
        if(l[i] != -1){
            tmp.left = new node(l[i])
            q.enque(tmp.left)
        }
        if(l[i+1] != -1){
            tmp.right = new node(l[i+1])
            q.enque(tmp.right)
        }
        i = i+2
    }
    return root;
}
```

TC: O(N) SC: O(N)

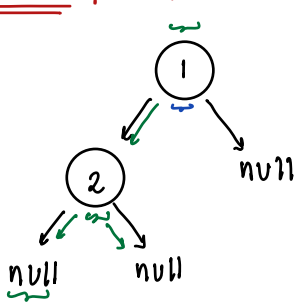
index

left child of tmp = i

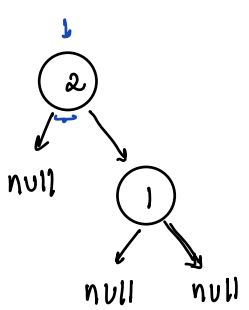
right child of tmp = i+1

8:30 → 8:40

Why inorder fails?



Issue: 2 Trees have same inorder Serialization



inorder Serialization:

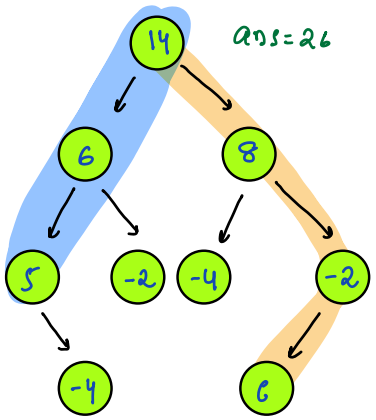
-1 2 -1 1 -1

inorder Serialization:

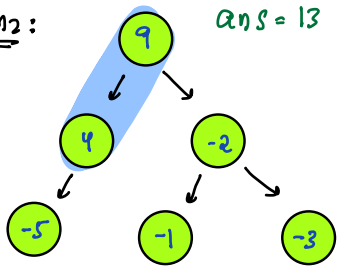
-1 2 -1 1 -1

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

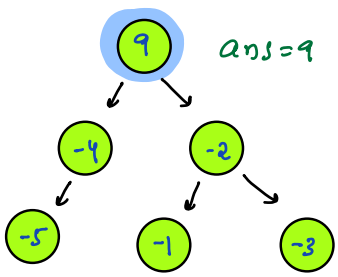
Eg1:



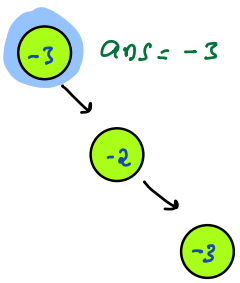
Eg2:



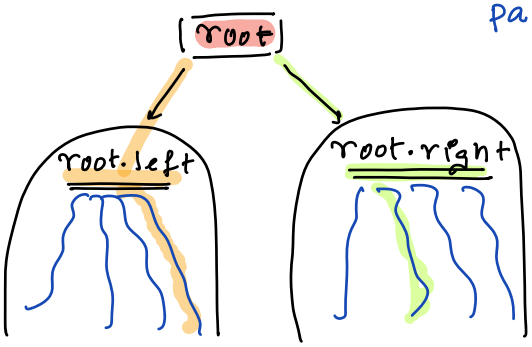
Eg3:



Eg4:



Idea :



path: start at root {root.data}

→ left: max path in in LST = {l}

→ right: max path in RST = {r}

→ don't goto left r right = {0}

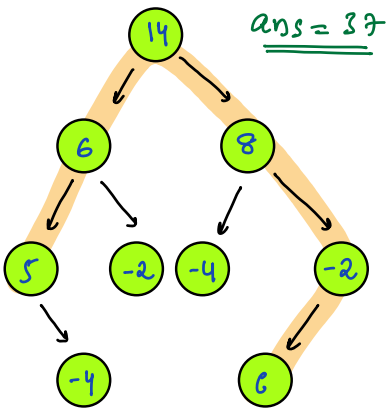
→ pick max of {l, r, 0} + root.data

```
int maxpathsum root(Node root){ TC: O(N)  SC: O(H)
    if(root==null) { return 0;}
    int l = maxpathsum root(root.left) // max path sum in LST
    int r = maxpath sum root(root.right) // max path sum in RST
    return  max(l, r, 0) + root.data
}
```

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

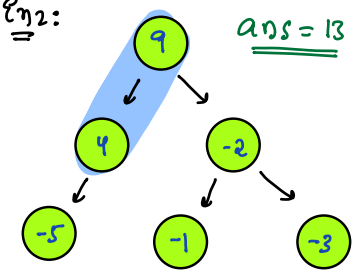
- → path passing rootnode
- → path containg rootnode

Eg1:



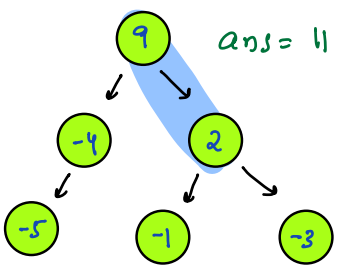
ans = 37

Eg2:



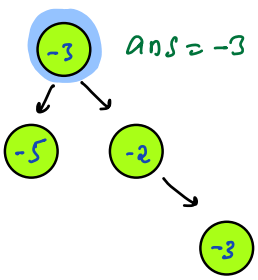
ans = 13

Eg3:



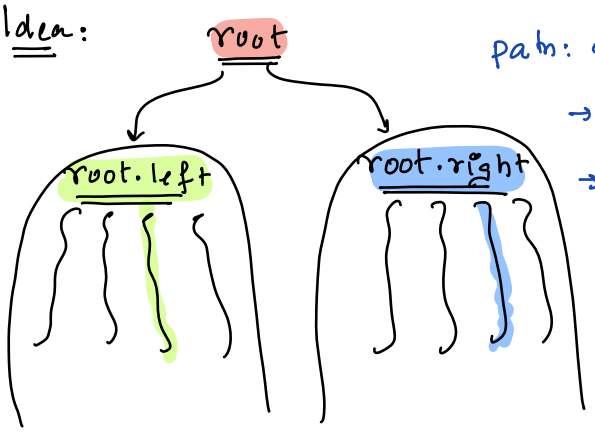
ans = 11

Eg4:



ans = -3

Idea:



path: contain rootnode

- → left: max path in in LST = l
- → right: max path in RST = r

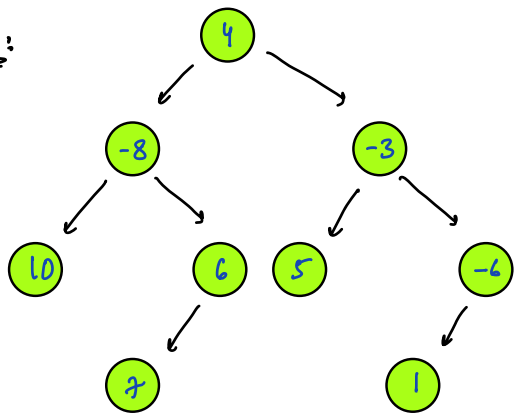
ans = root.data + max(l, 0) + max(r, 0)

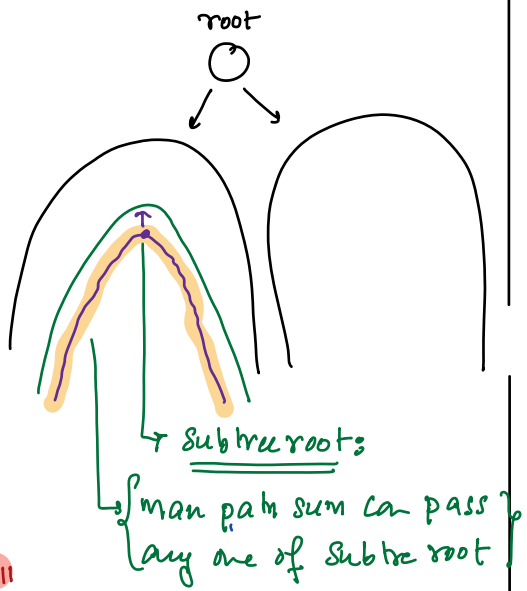
```
— manpath root(Node root){ TC: O(N)  SC: O(H)
    int l = manpathsum(root.left)
    int r = manpathsum(root.right)
    return root.data + man(l,0) + man(r,0)
}

int manpathsum(Node root){
    if(root==null) { return 0;
    int l = manpathsumroot(root.left) // man path sum in LST
    int r = manpathsumroot(root.right) // man path sum in RST
    return man(l,r,0) + root.data
}
```

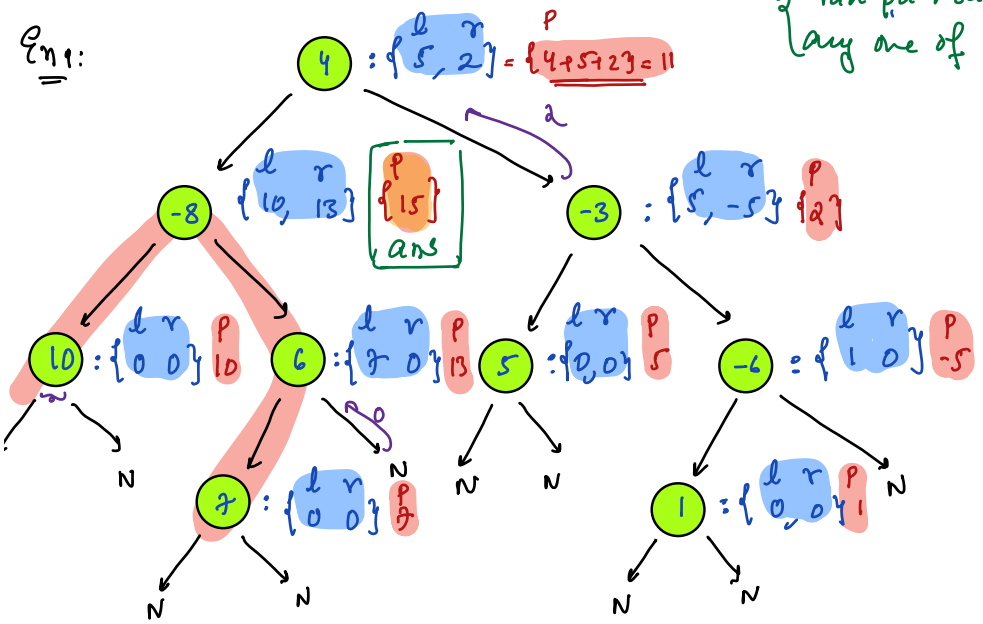
Q3) Max sum path in Tree

Eg:





Eg:



Idea: For every node get max sum path passing through it & get overall max

```
int ans = INT_MIN
int maxpathsum root(Node root){ TC: O(N)  SC: O(H)
    if(root == null) { return 0;
    int l = maxpathsum root(root.left) // max path sum in LST
    int r = maxpathsum root(root.right) // max path sum in RST
     ans = max (ans, root.data + max(l,0) + max(r,0))
    return  max(l, r, 0) + root.data
}
```