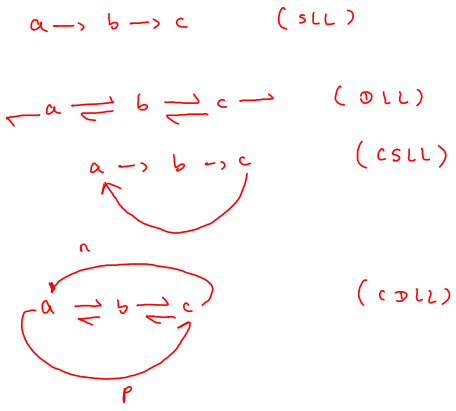
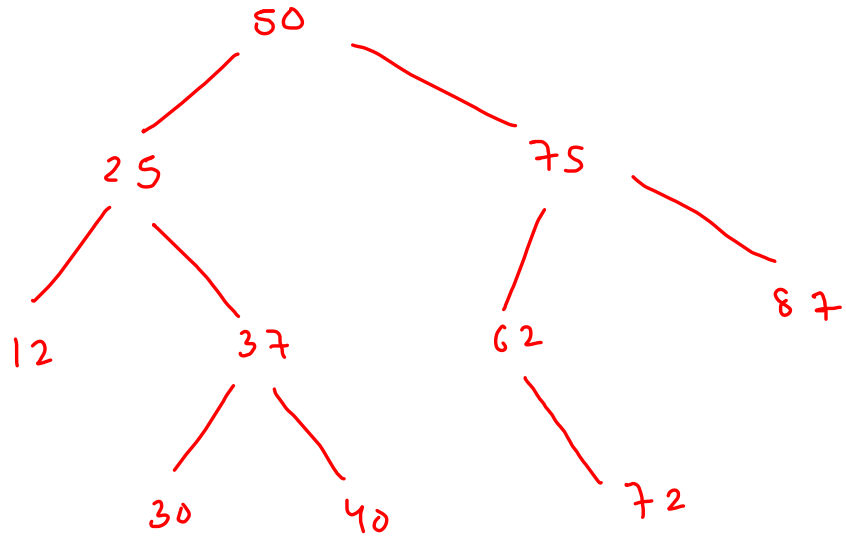


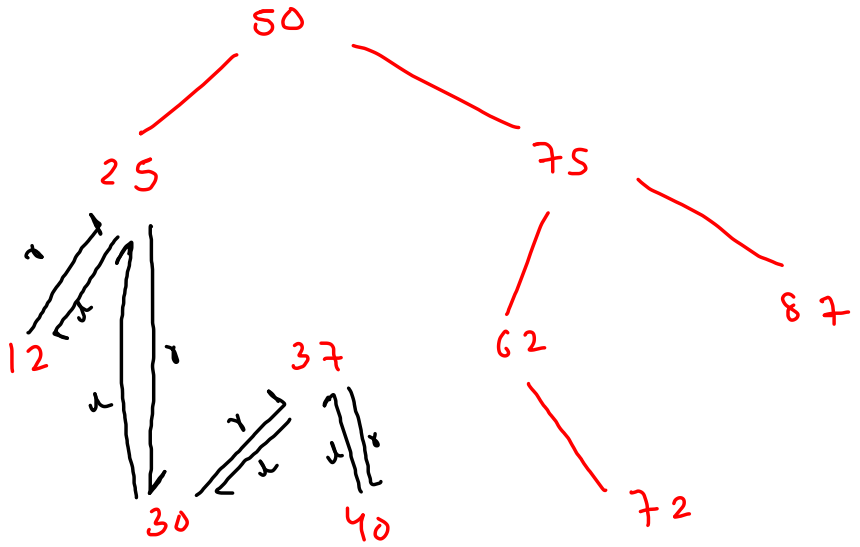




```
TreeNode {
    int data;
    TreeNode left;

    TreeNode right;

}
```

left → (prev)

right → (next)

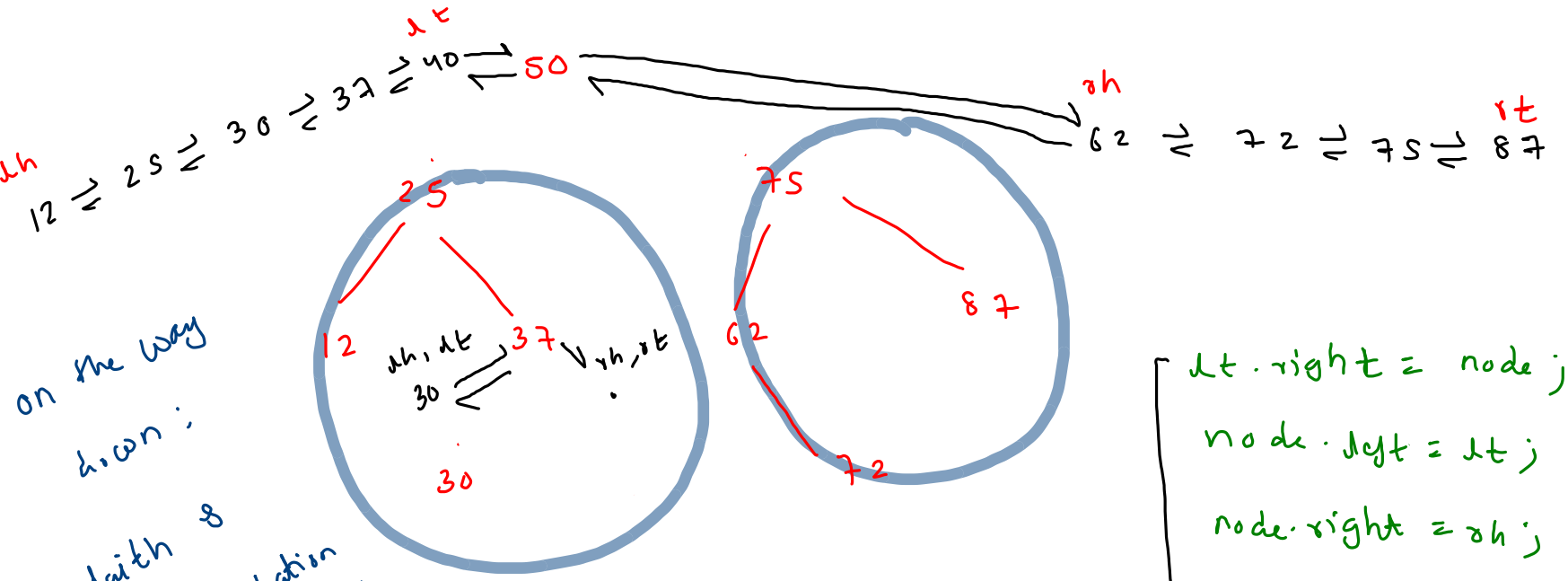
✗ extra space




inplace conversion

left : prev

right : next



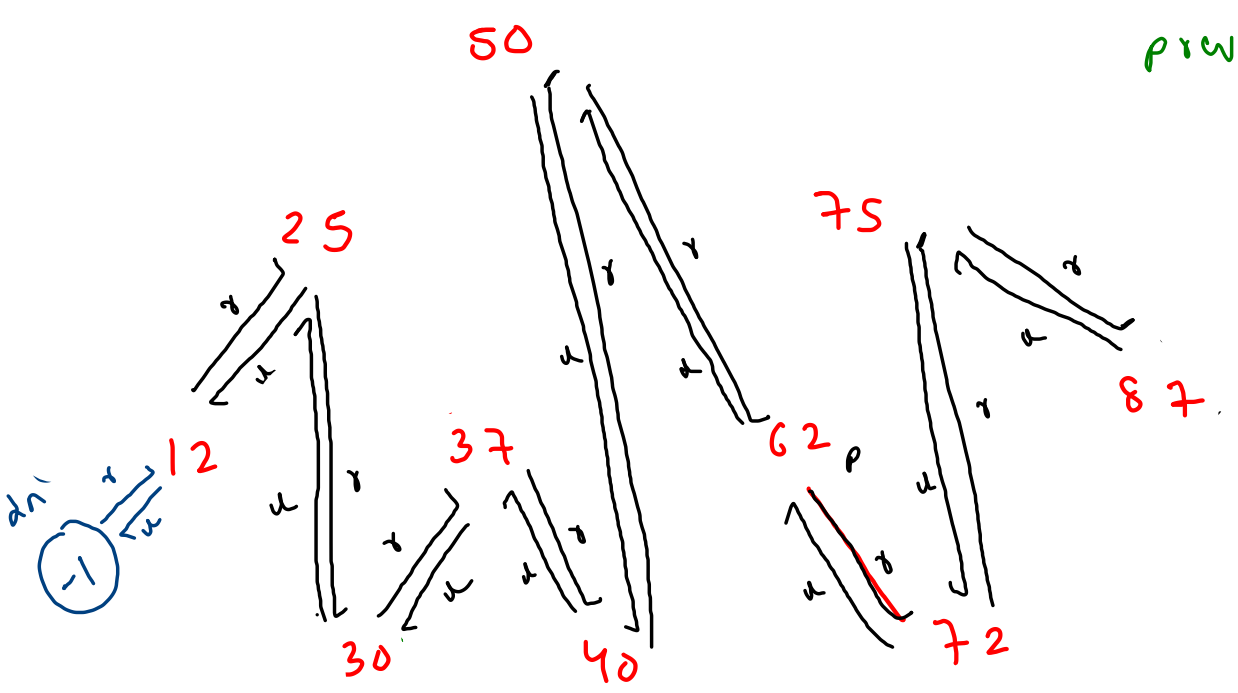

on the way down:

faith & expectation

```
lt.right = node;
node.left = lt;
node.right = rh;
rh.left = node;
```

head : lh

tail : rt



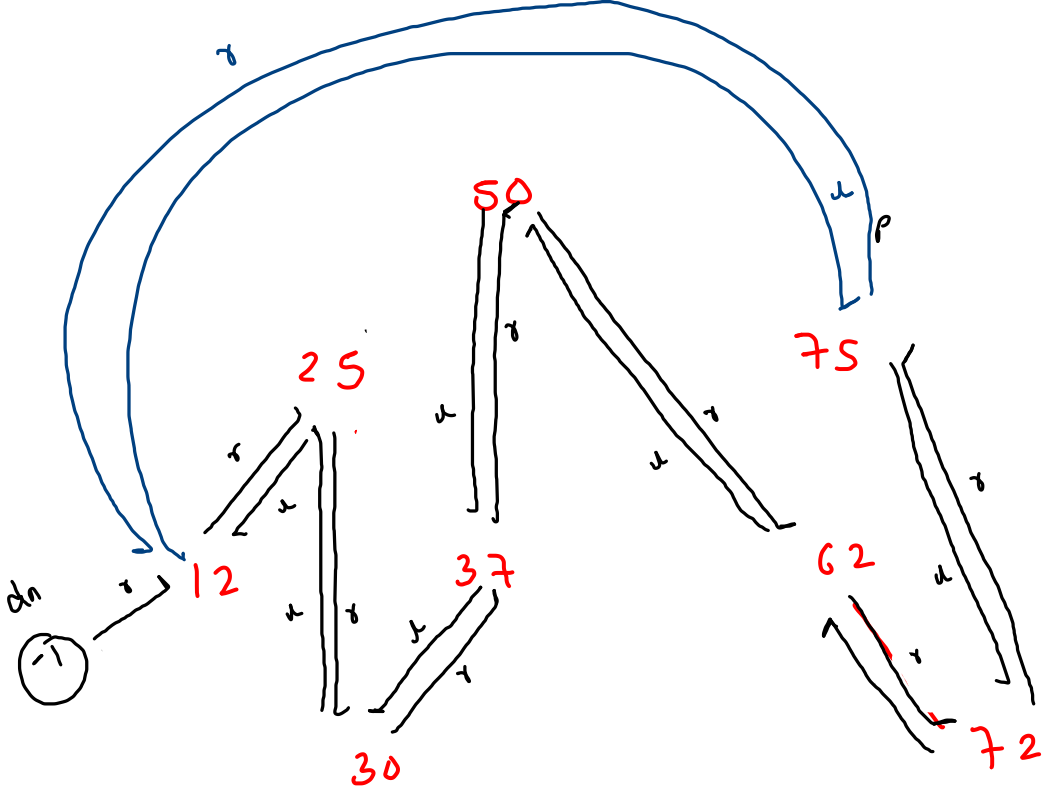

prev = ~~-1~~

~~12~~ ~~25~~ ~~30~~ ~~37~~ ~~40~~ ~~50~~ ~~62~~ ~~72~~ ~~75~~ 87

left : prev

right : next

```
call (node.lft);
prev.right = node;
node.lft = prev;
prev = node;
call (node.right);
```

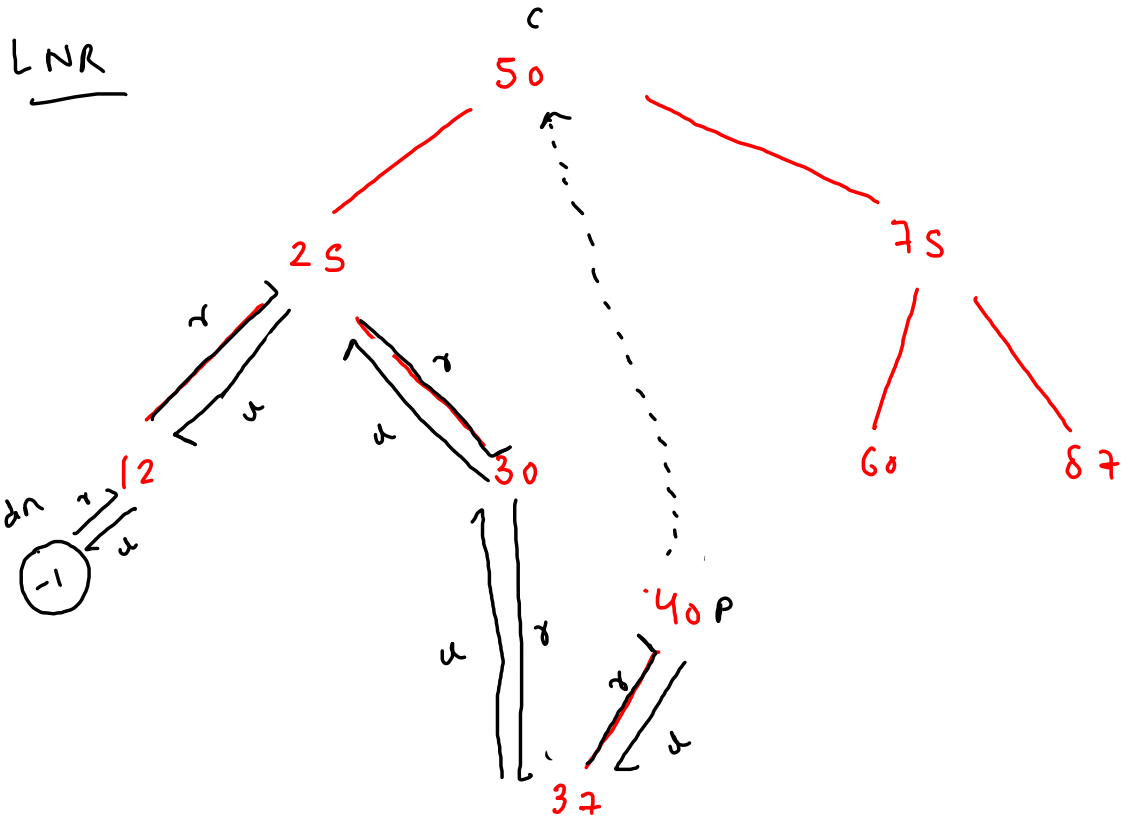


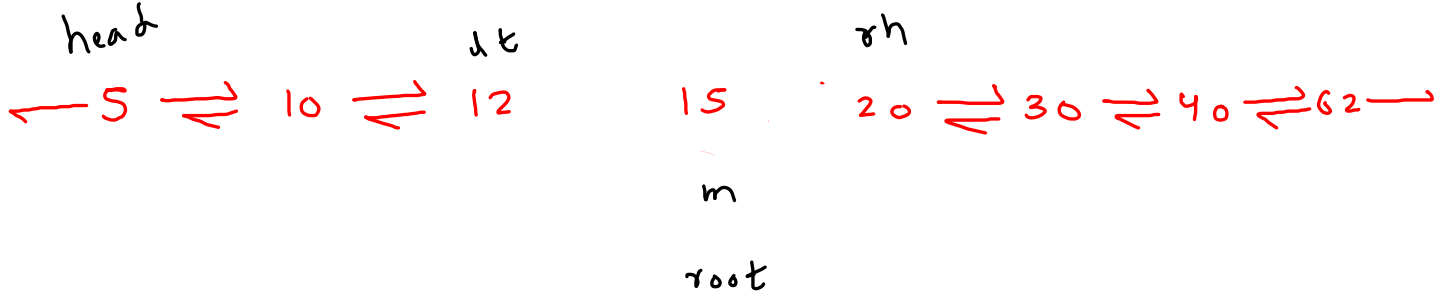

```
public static void helper(Node node) {
    if(node == null) {
        return;
    }

    helper(node.left);

    //work & update
    prev.right = node;
    node.left = prev;
    prev = node;

    helper(node.right);
```

head = 12 (dn.right)

tail = 75 (prev)

```java
public static Node bToDLL(Node root) {
    Node dummy = new Node(-1);
    Node prev = dummy;

    Node curr = root;

    while(curr != null) {
        Node ln = curr.left;

        if(ln == null) {
            //links
            prev.right = curr;
            curr.left = prev;

            //move
            prev = curr;
            curr = curr.right;
        }
        else {
            Node rmn = rightMostNode(ln,curr);

            if(rmn.right == null) {
                //create a thread then go to left subtree
                rmn.right = curr;
                curr = curr.left;
            }
            else if(rmn.right == curr) {
                //left subtree is done, break the link and then go to right
                rmn.right = null;

                //links
                prev.right = curr;
                curr.left = prev;

                //move
                prev = curr;
                curr = curr.right;
            }
        }
    }
```

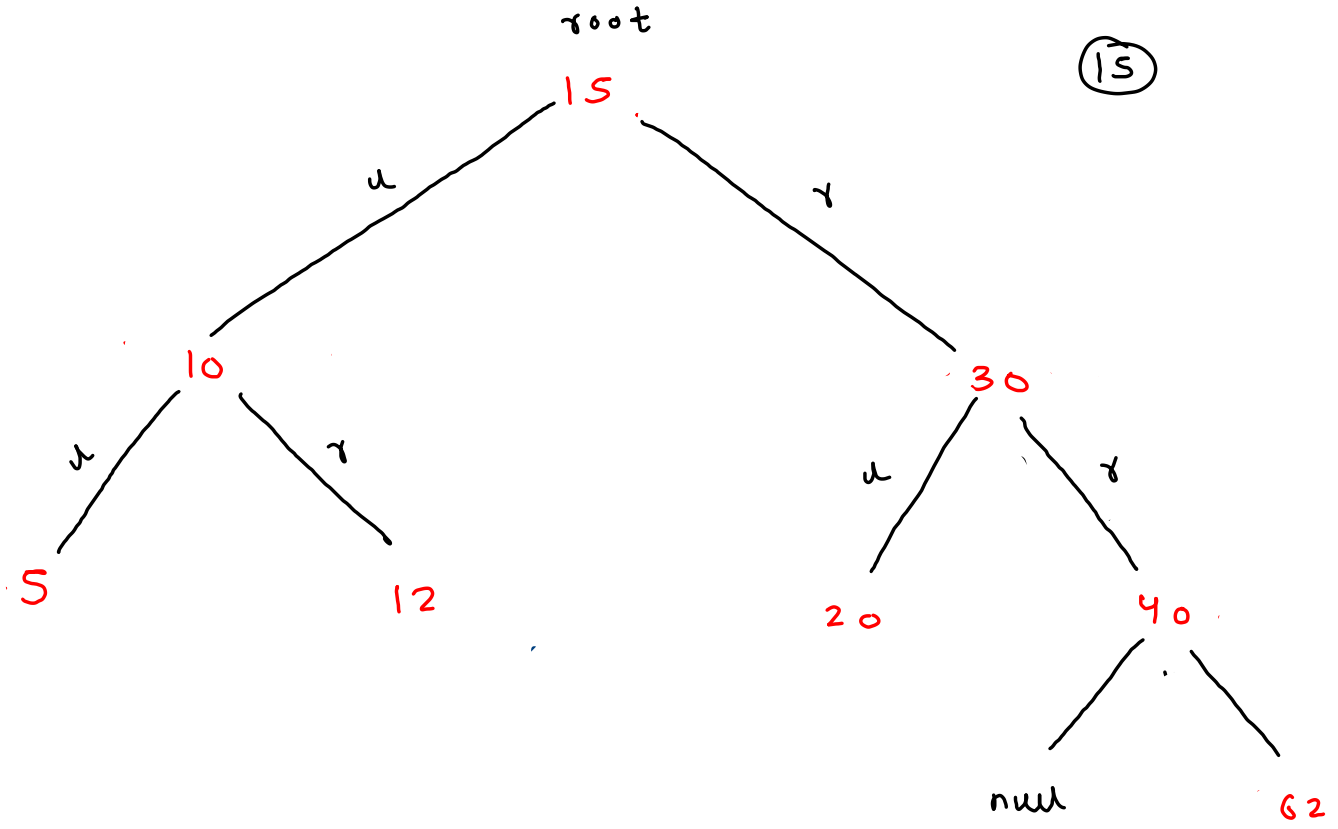





```
Node construct (Node head)

    Node m = mid(head);
    Node rh = m.right;
    Node lt = m.prev;

    connections break;

    root.left = construct(head);
    root.right = construct(rh);
}
```

```
m.right = null;
rh.left = null;
lt.right = null;
m.prev = null;
```

```java
public static Node SortedDLLToBST(Node head) {
    if(head == null || head.right == null) {
        return head;
    }

    Node mid = midNode(head);

    Node root = mid;

    Node rh = mid.right;
    Node lt = mid.left;


    //connections

    if(lt != null) {
        lt.right = null;
    }
    mid.left = null;

    rh.left = null;
    mid.right = null;


    root.left = SortedDLLToBST(lt == null ? null : head);
    root.right = SortedDLLToBST(rh);

    return root;
}
```

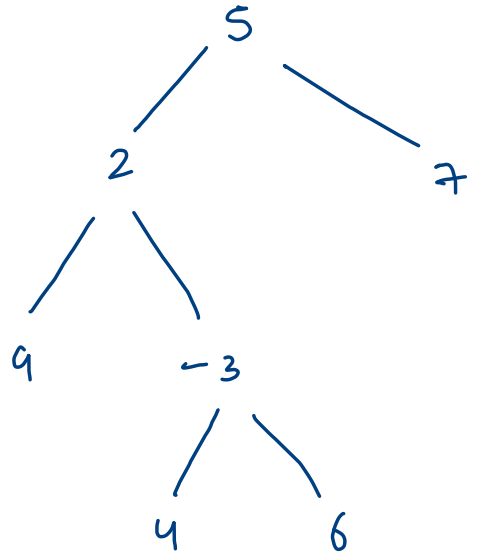


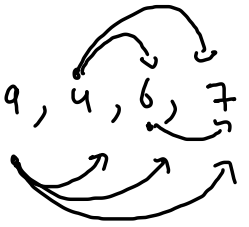

15

right (next)

left (prev)

max path sum
leaf to leaf



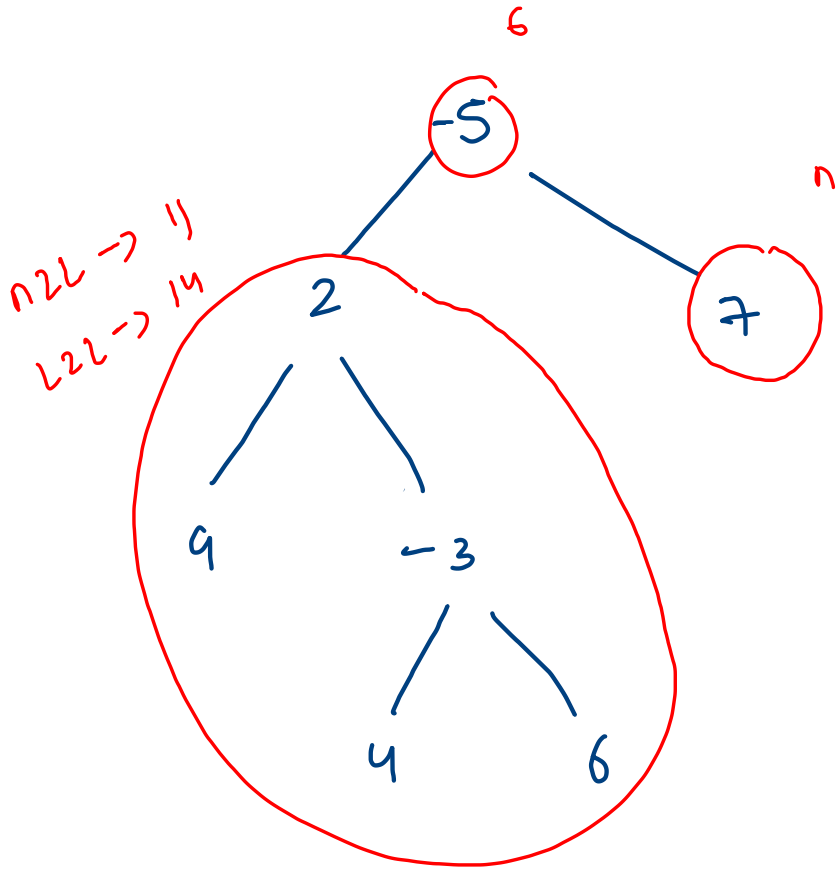




3 + 2 + 1

= 6

leaf to leaf

- 9 -> 7 : 9 2 5 7
- 9 -> 6 : 9 2 -3 6
- 9 -> 4 : 9 2 -3 4
  - 4 -> 6 : 4 -3 6
  - 4 -> 7 : 4 -3 2 5 7
- [ 6 -> 7 : 6 -3 2 5 7




n2L -> max(lcp.n2L , rcp.n2L) + node.data;

L2L -> max( lcp.L2L, rcp.L2L, lcp.n2L + rcp.n2L + node.data);