Diameter 1.

diameter : 0 2 4 6 7

13

```java
public static int diameter = 0;

public static int HeightForDia(Node node) {
    if(node == null) return -1;

    int lh = HeightForDia(node.left);
    int rh = heightForDia(node.right);

    diameter = Math.max(diameter, lh + rh + 2);

    return Math.max(lh, rh) + 1;
}
```

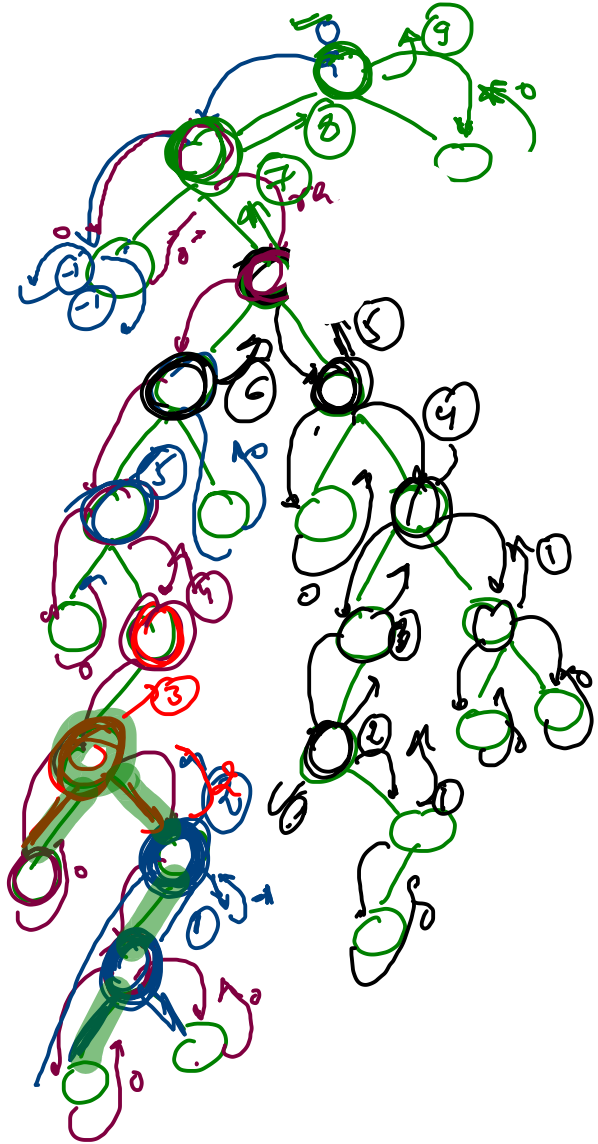
9

lh = 6

rh = 5

diamet = 6 + 5 + 2

= 13

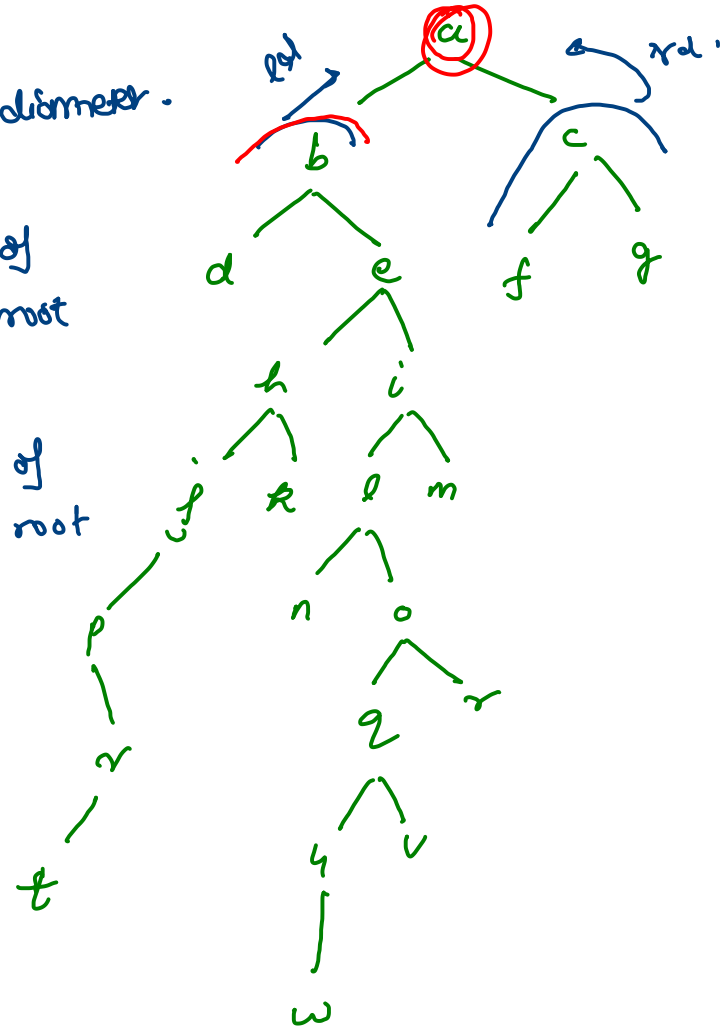


Diameter2.

Expectation → diameter (a) → overall max. diameter.

faith → ld = diameter(b) → overall diameter of subtree having root is b.

rd: diameter (c) → overall diameter of subtree having root is c.



Merging of faith & Expectation;

```
lh = height(node.left);
rh = height(node.right);
d = lh + rh + 2;

return → ld vs. rd vs d;
```

Complexity wise //-

```java
// Diameter 2
public static int diameter2(Node node) {
    if(node == null) return -1;

    int ld = diameter2(node.left);
    int rd = diameter2(node.right);

    int lh = height(node.left);
    int rh = height(node.right);
    int d = lh + rh + 2;

    return Math.max(d, Math.max(ld, rd));
}
```

$= O(n) =$

$\to O(n^2)$

$O(n)$

$O(n^2)$

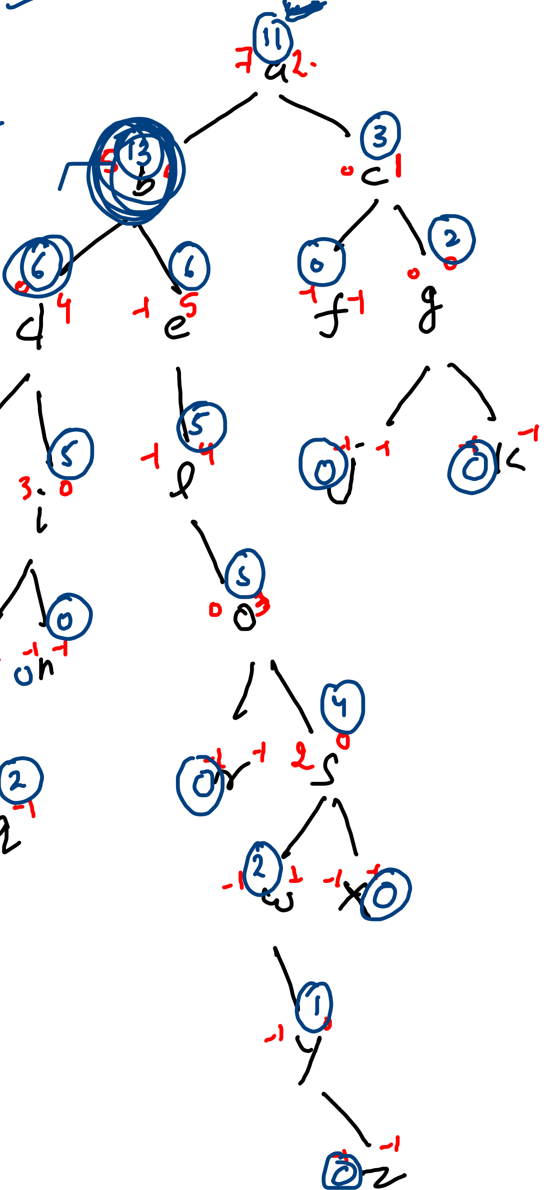
max = 13

Nadeem: 12.

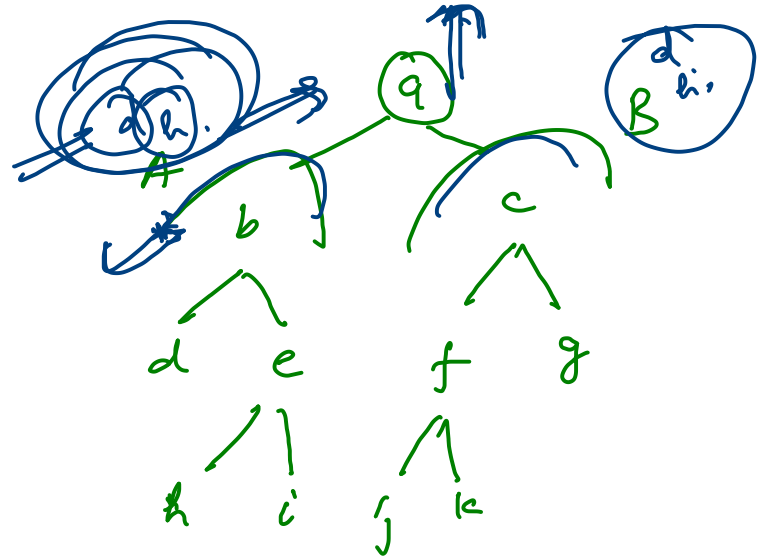
Alekhya: 17

Devender =

Amruth = 15.



Diameter .3.



A → left height
left diameter.

B → Right height
Right Diameter.

on Node 'a'.

lf =

rd =

lh =

rh =

d = lh + rh + 2;

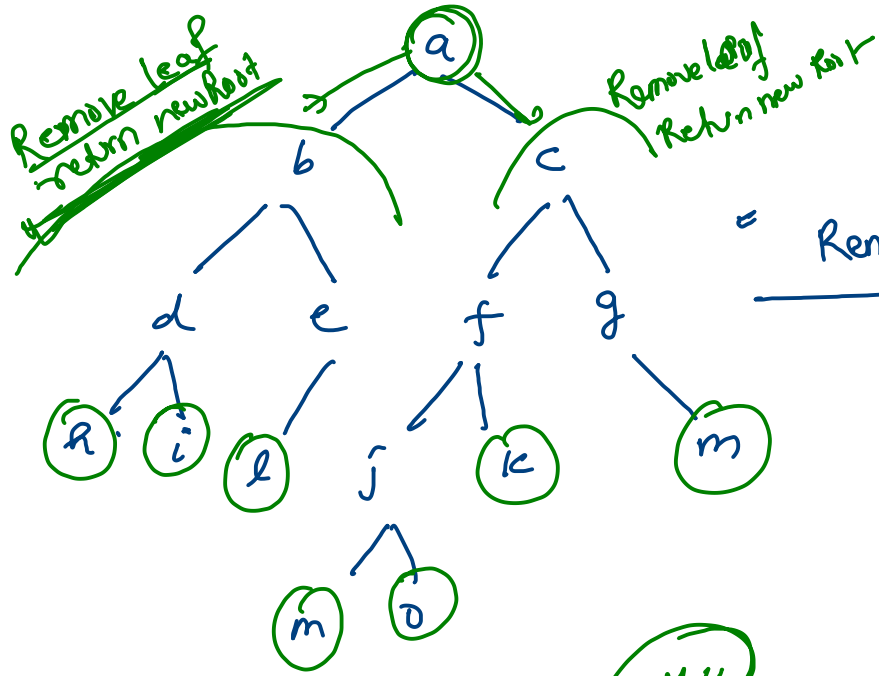
Math.max(lh, rh) + 1;

ld vs rd vs (lh + rh + 2)

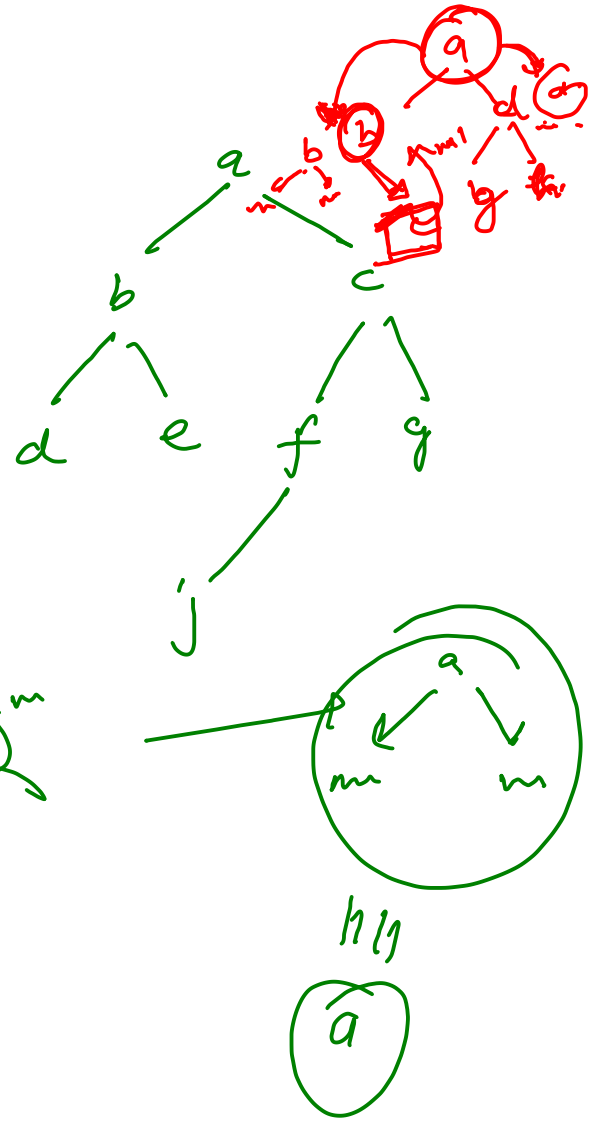
return (height, diameter)

Remove leaves.

leaf → [left == NULL, Right == Null] . I am leaf

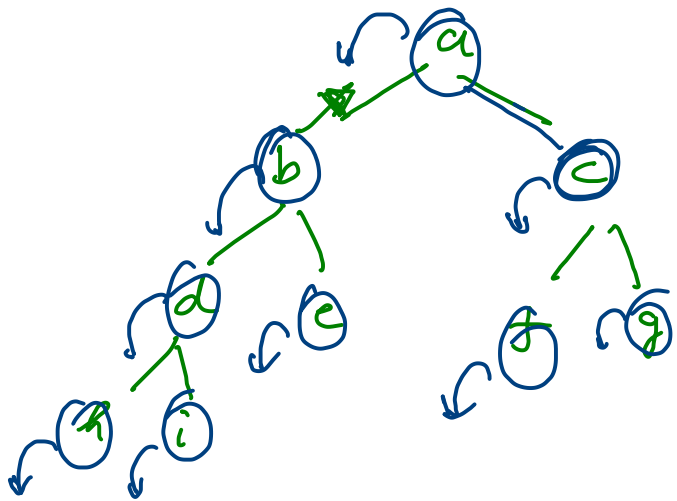


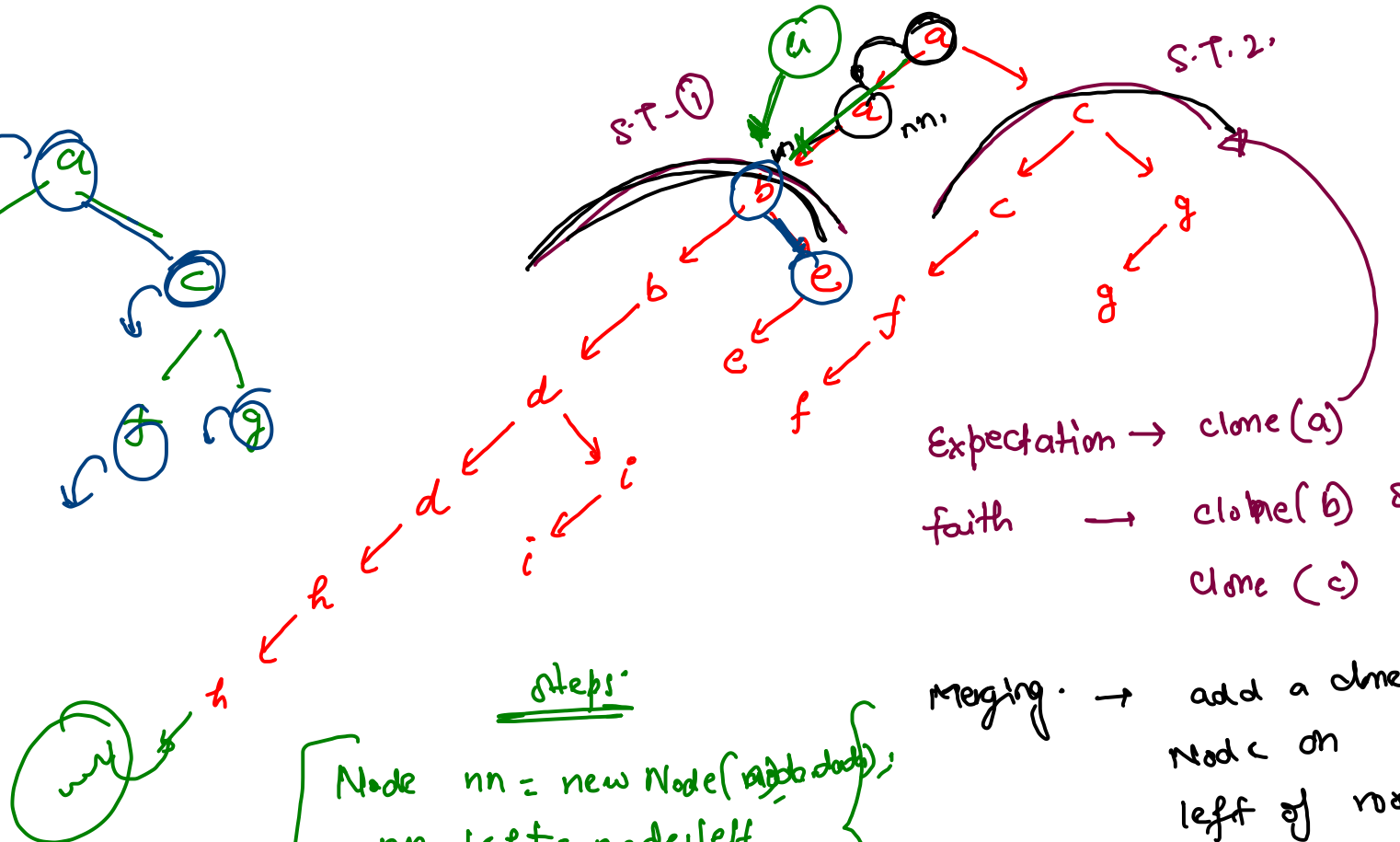
= Remove leaf →



a.left = null; Remove leaf (a.left);

left Clone tree



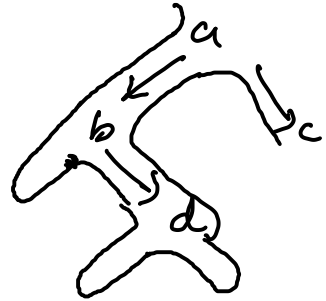


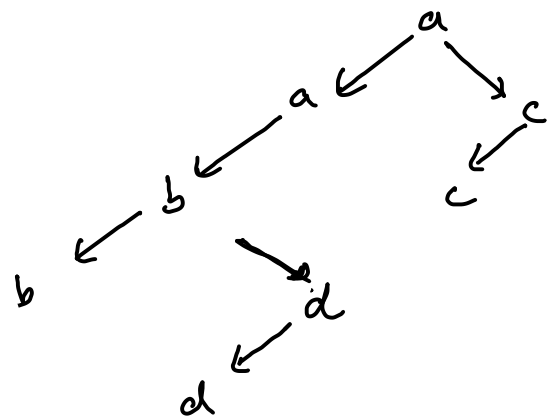
steps:

```
Node nn = new Node(node.data);
nn.left = node.left
node.left = nn;
return nn   (a)
```

merging → add a clone Node on left of root & make nn.left = actual left of Root

```java
public static Node createLeftCloneTree(Node node
{
    // write your code here
    if(node == null) return null;

    node.left = createLeftCloneTree(node.left);
    node.right = createLeftCloneTree(node.right);

    // steps of clonning
    Node nn = new Node(node.data, null, null);
    nn.left = node.left;
    node.left = nn;
    return node;
}
```





Expectation → clone(a) ⟶ clone complete tree.

faith → clone(a.left) → clone complete subtree ①

clone(a.right) → clone complete subtree ②

Method 1

a.left = nn;

nn.left = (b)

??

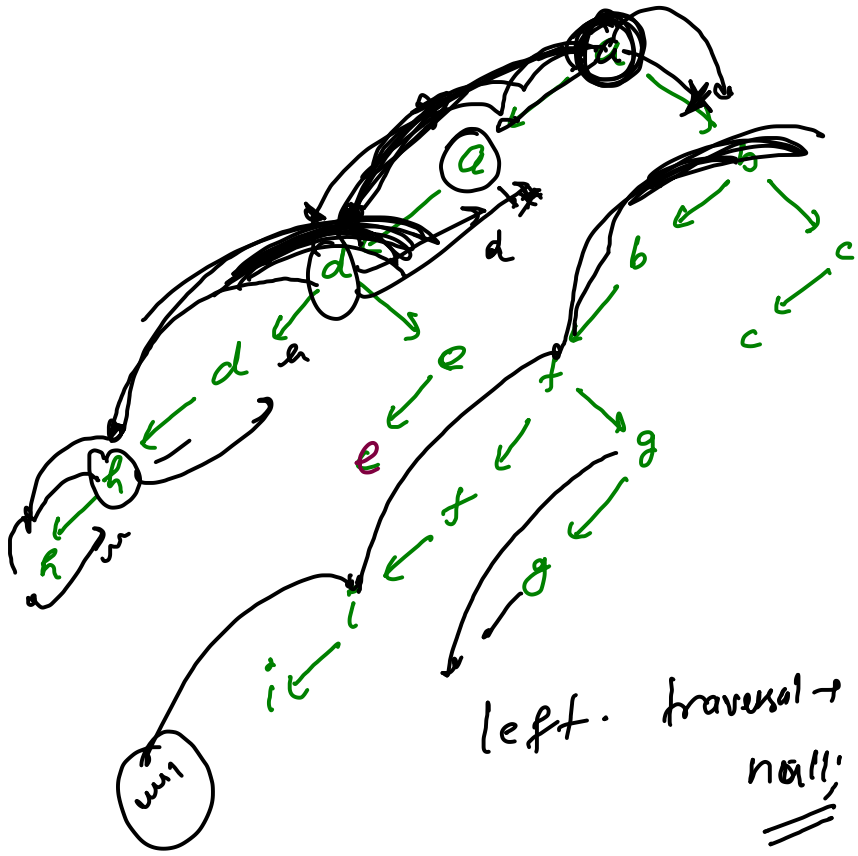
=

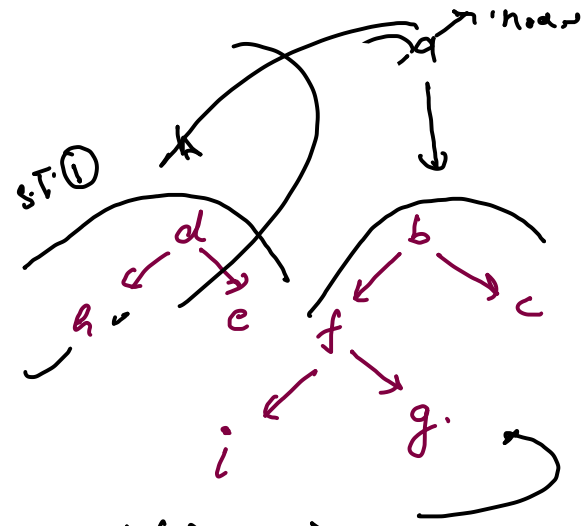
Method 2

nn.left = a.left

a.left = nn.

Transform a left Cloned Tree to Original form.





Expectation → reset (a) →

faith → reset (a.left.left); ST. ① root

reset (a.right) → ST. ② root

left. traversal → Even jump.

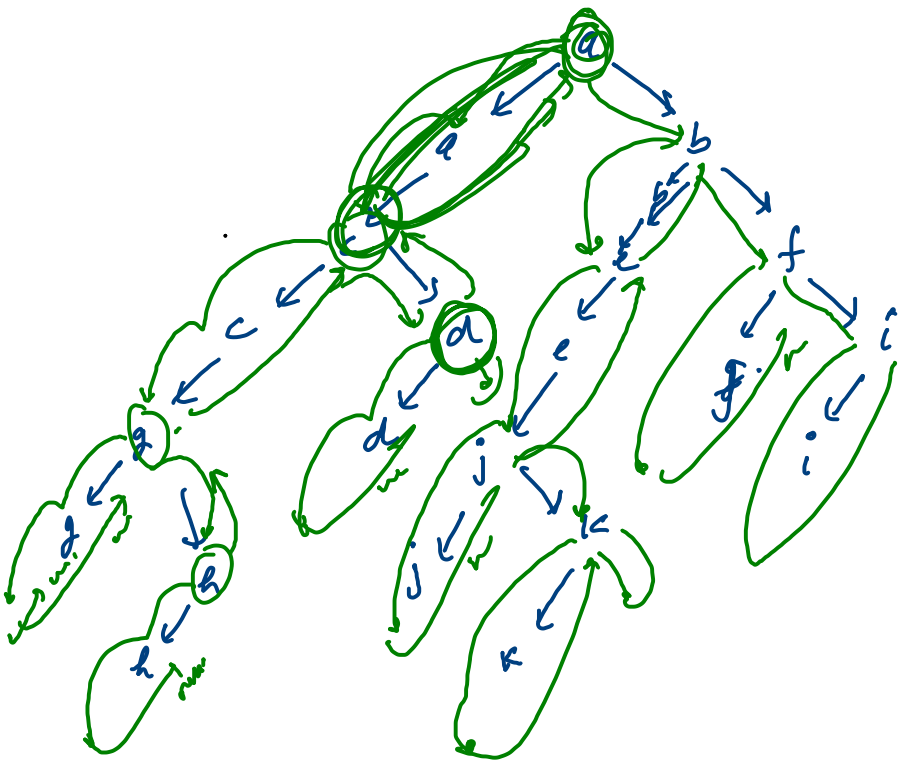
node.left = node.left.left;

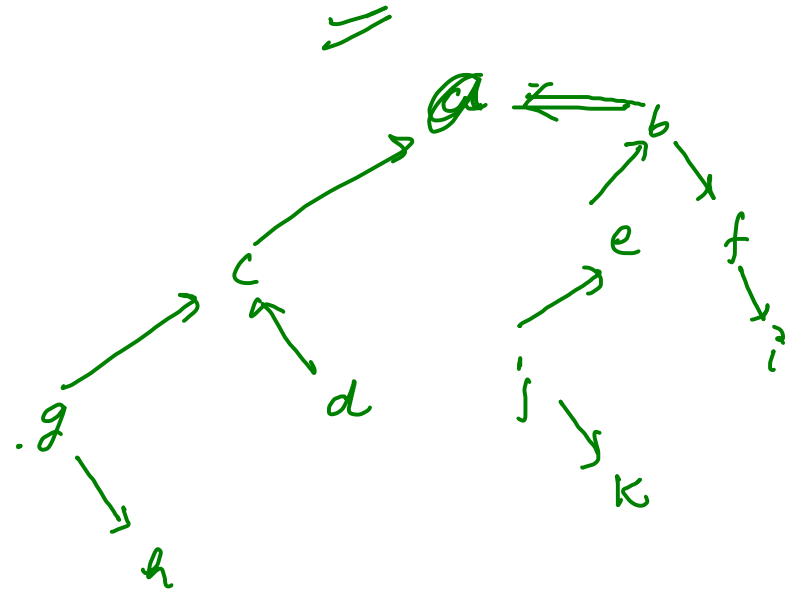
return node;

null;

a →

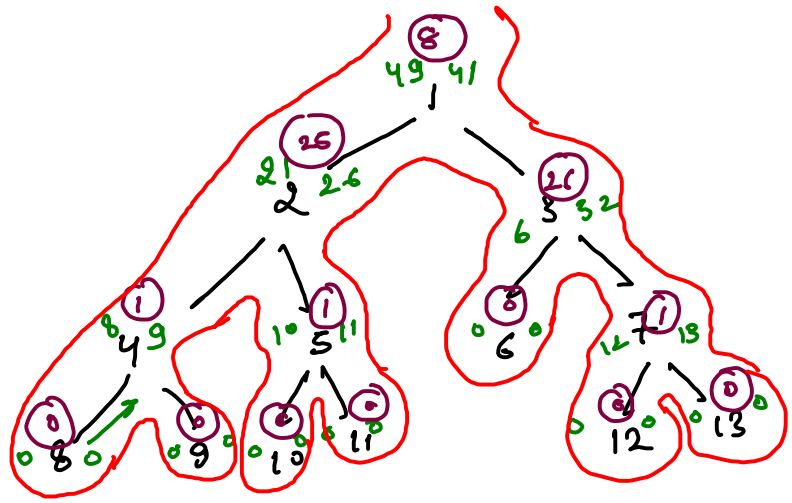
```java
public static Node transBackFromLeftClonedTree(Node node){
  // write your code here
  if(node == null) return null;

  node.left = transBackFromLeftClonedTree(node.left.left);
  node.right = transBackFromLeftClonedTree(node.right);

  return node;
}
```





Tilt of Binary Tree

Tilt = | left subtree sum − Right subtree sum |



Node- Lsum Rsum tilt

Tilt = overall sum of all tilts

0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 0 + 1 + 26 + 8.

= 37 ⇒

Travel & change strategy.

Travel → for sum

change → tilt variable.

Diameter

- Travel for height
- change - Diameter

tilt

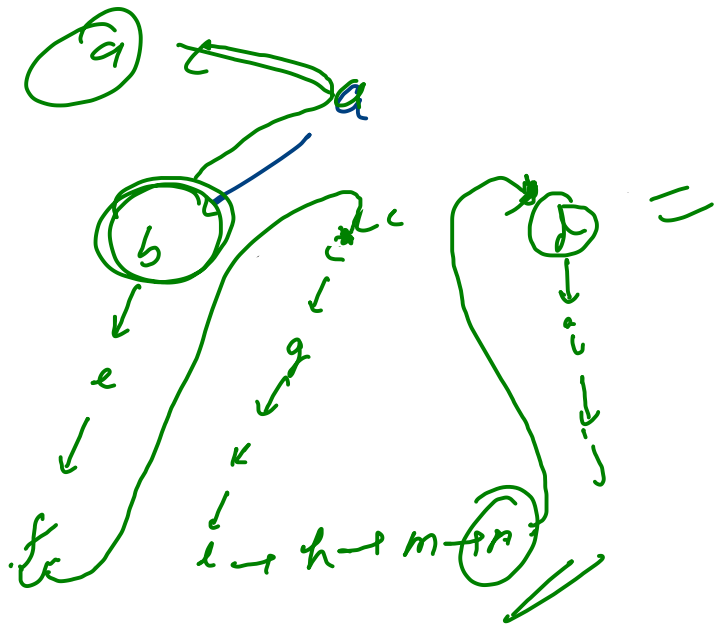
- Travel - sum
- change → tilt

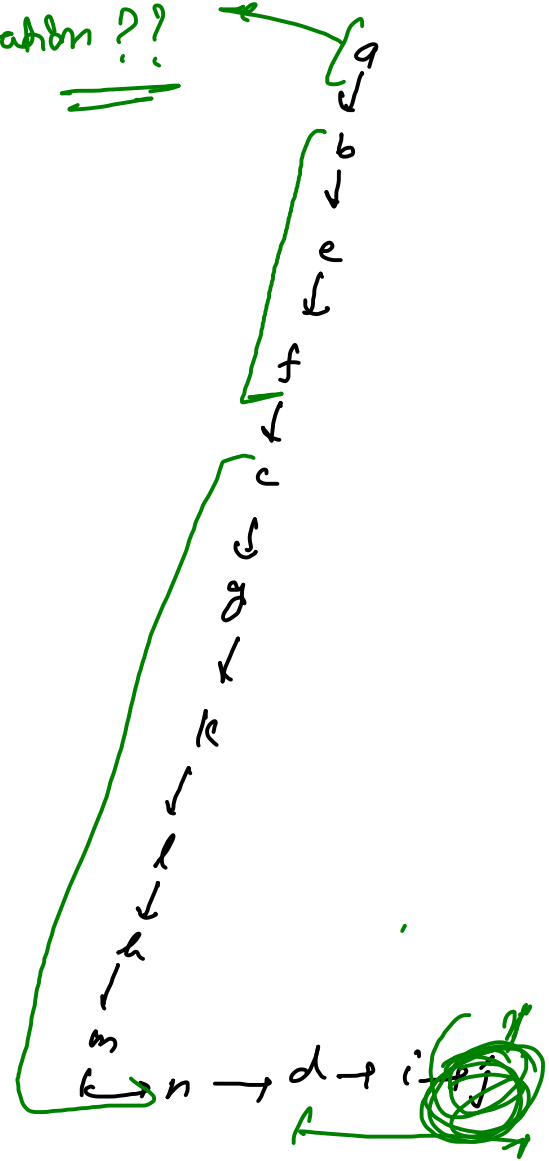
size, sum, max, min, height, dia, filt, → single iteration ??

fail

Class

No static var. allowed





Linearise -

Expectation → Linearise (a)

faith → Linearise (b)

children of a → Linearise (c)

→ Linearise (d)

~~get last Child~~ → d

remove last Child

get last Child - (c)

get Tail of last Child. = n

n.child (c).