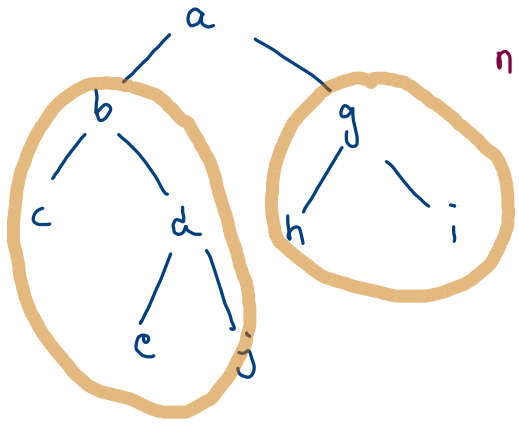
Maximum Path Sum In Between Two Leaves Of Binary Tree

n2l: [bc, bde, bdj]

u2l: [cbde, cbdj, edj]

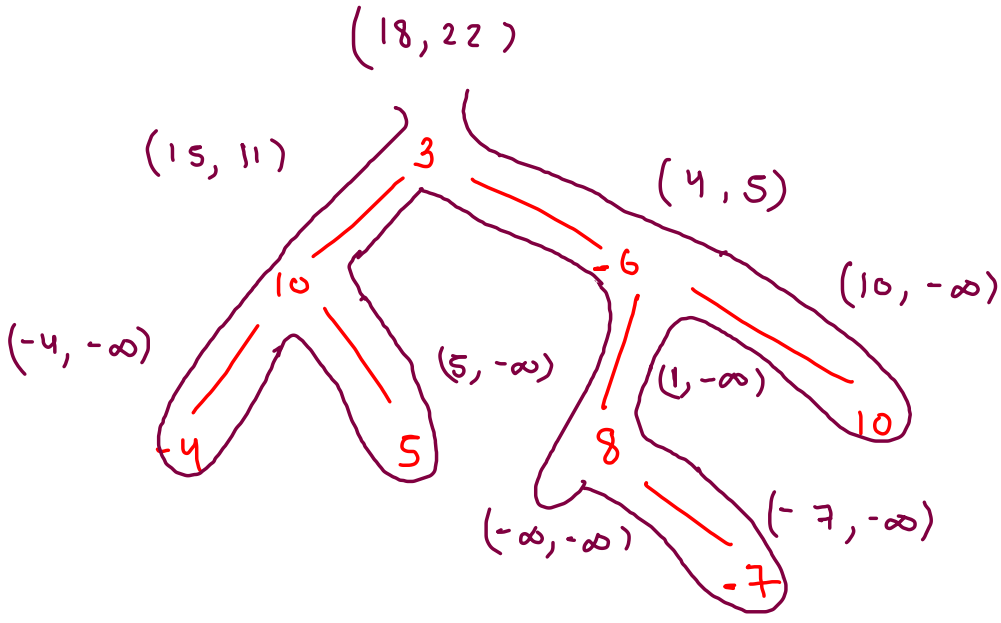


n2l: [gh, gi]

u2l: [hgi]

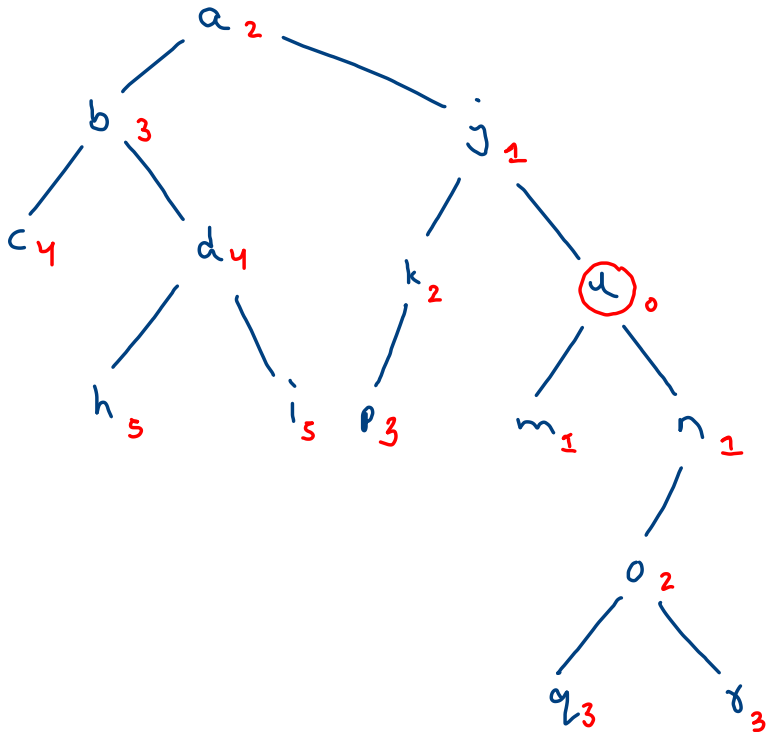
```
lp = solve(node.left);
rp = solve(node.right);

n2l = max(lp.n2l, rp.n2l)
      + node.val;

l2l = max(lp.l2l, rp.l2l,
      lp.n2l + node.val +
      rp.n2l)
```

```java
public  Pair helper(Node node) {
  if(node == null) {
      return new Pair(Integer.MIN_VALUE,Integer.MIN_VALUE);
  }
  if(node.left == null && node.right == null) {
      return new Pair(node.data,Integer.MIN_VALUE);
  }

  Pair lp = helper(node.left);
  Pair rp = helper(node.right);

  int n2l = Math.max(lp.n2l,rp.n2l) + node.data;

  int l2l = Math.max(lp.l2l,rp.l2l);

  if(node.left != null && node.right != null) {
      l2l = Math.max(l2l,lp.n2l + node.data + rp.n2l);
  }

  return new Pair(n2l,l2l);
}
```



n2l, l2l

Burning Tree



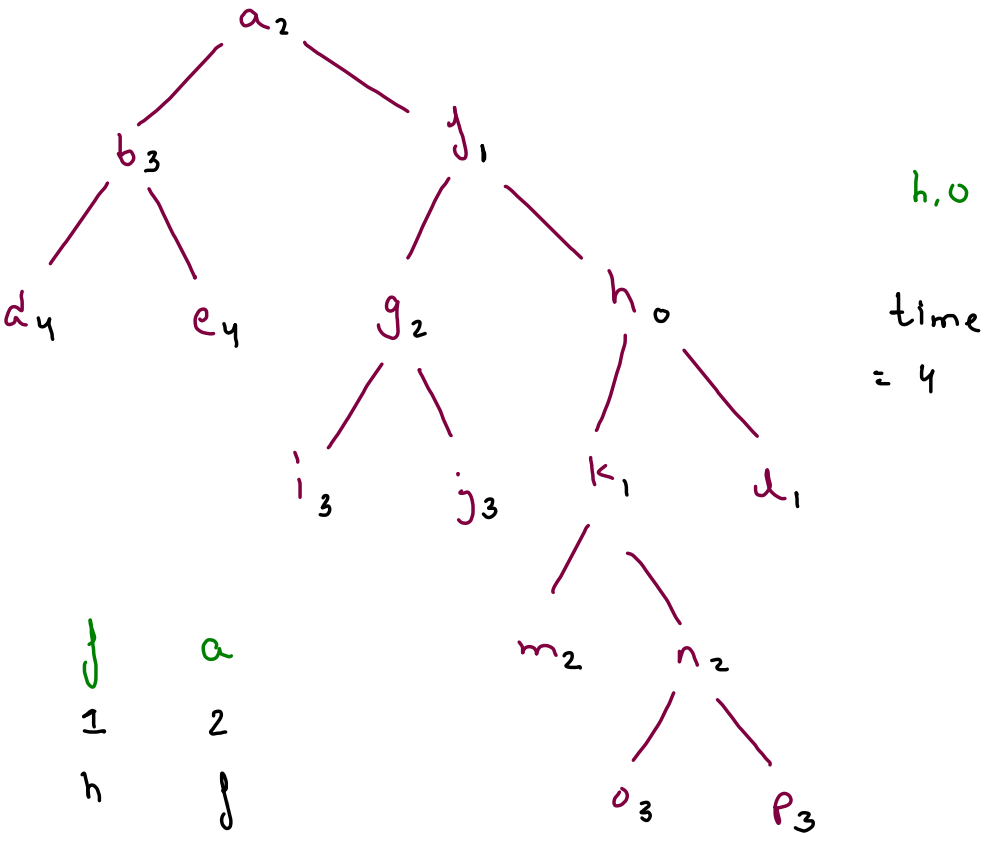
1, 0

	1	j	a
time	0	1	2
bn	null	1	j

time = 5

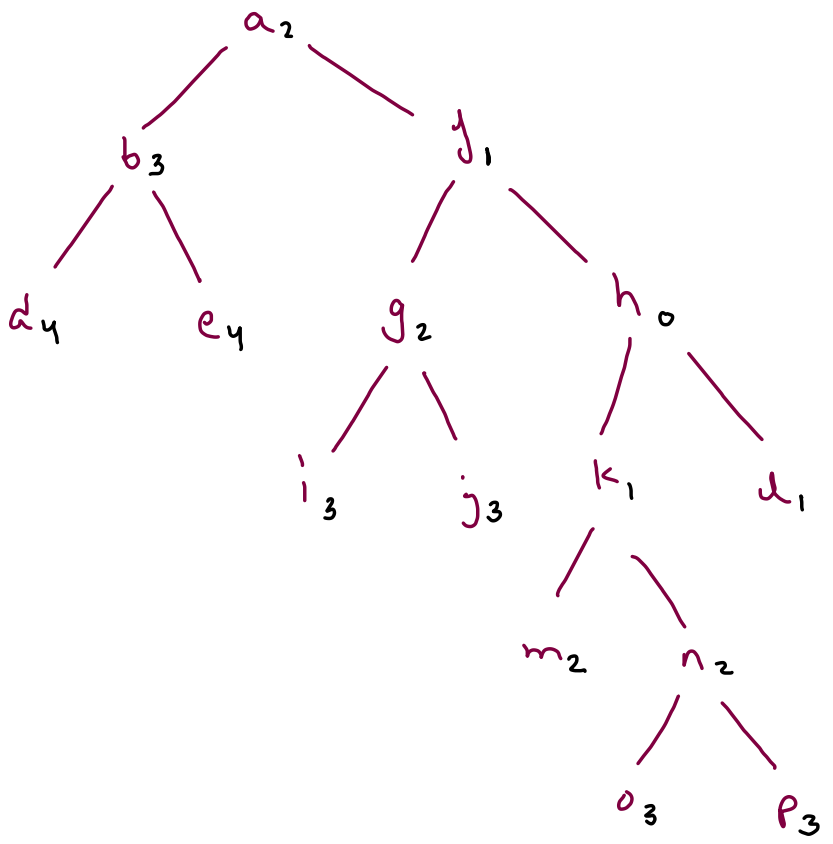
```java
public static int minTime(Node root, int target)
{
    time = 0;
    ArrayList<Node>n2rp = nodeToRootPath(root,target);

    int t = 0;
    Node bn = null;

    for(int i=0; i < n2rp.size();i++) {
        Kdown(n2rp.get(i),t,bn);
        t++;
        bn = n2rp.get(i);
    }

    return time;
}
```

```java
static int time; //total time taken to burn the complete tree

public static void Kdown(Node node,int t,Node bn) {
    if(node == null || node == bn) {
        return;
    }

    time = Math.max(time,t);

    Kdown(node.left,t+1,bn);
    Kdown(node.right,t+1,bn);
}
```



	h	j	a
t	0	1	2
bn	null	h	j

Burning tree 2



0 → h

1 → k, l, f

2 → m, n, g, a

3 → o, p, i, j, b

4 → d, e