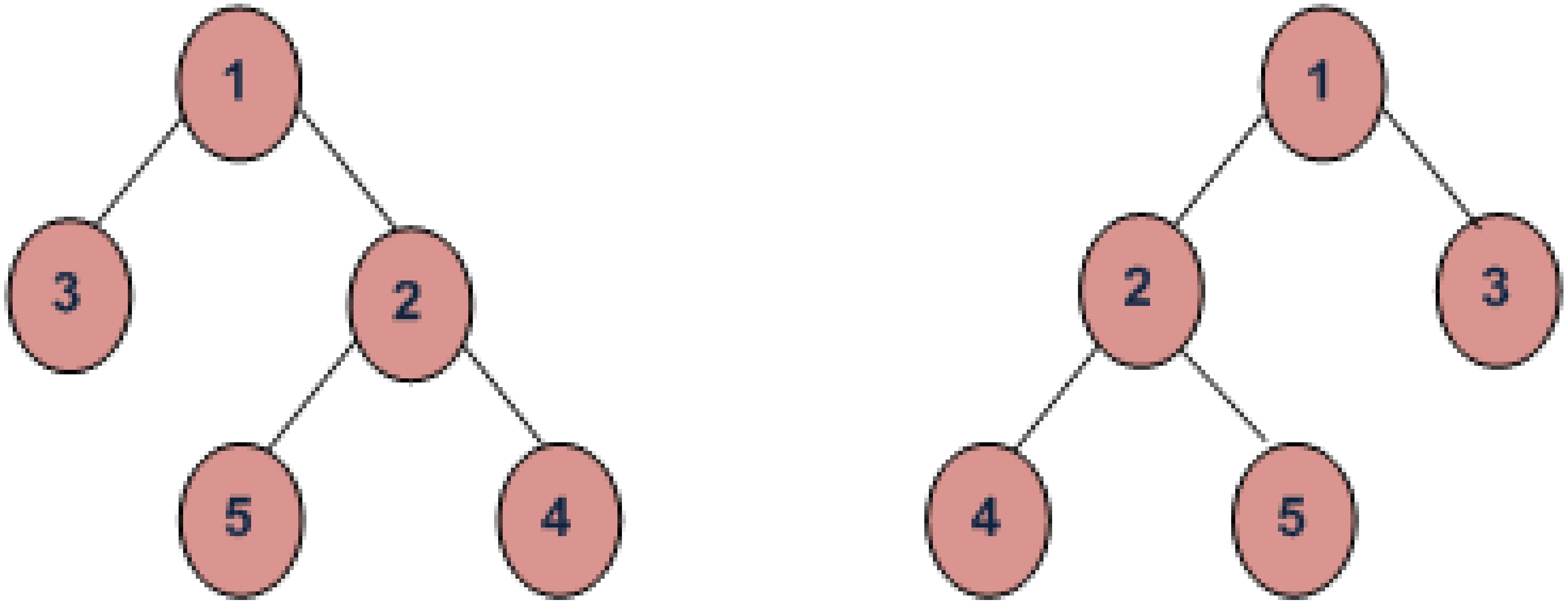
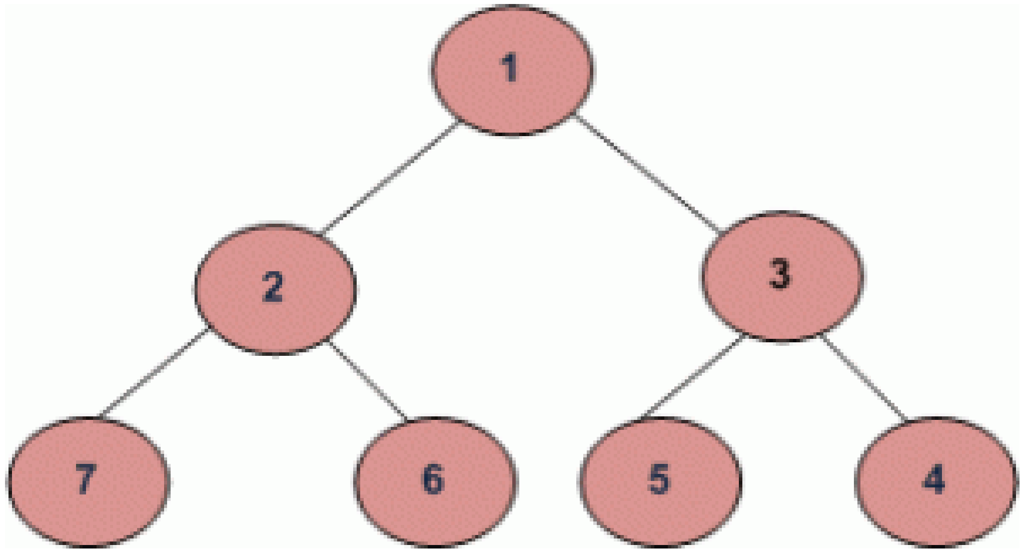
Trees

1. Given a binary tree, convert it to its mirror.



Mirror Trees

2. Given a binary tree, your task is to complete the function zigZagTraversal(), that prints the nodes of binary tree in ZigZag manner.



ZigZag traversal for the above tree will be:

1 3 2 7 6 5 4

3. Check for Balanced Tree.

Given a binary tree, find if it is height balanced or not. A tree is height balanced if difference between heights of left and right sub-trees is not more than one for all nodes of tree.

A height balanced tree

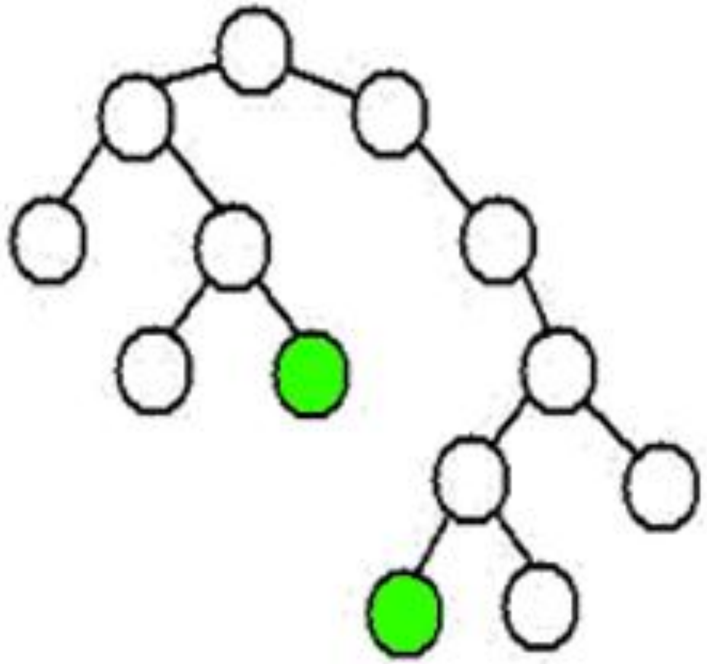
```
      1
    /   \
   10    39
  /
 5
```

An unbalanced tree

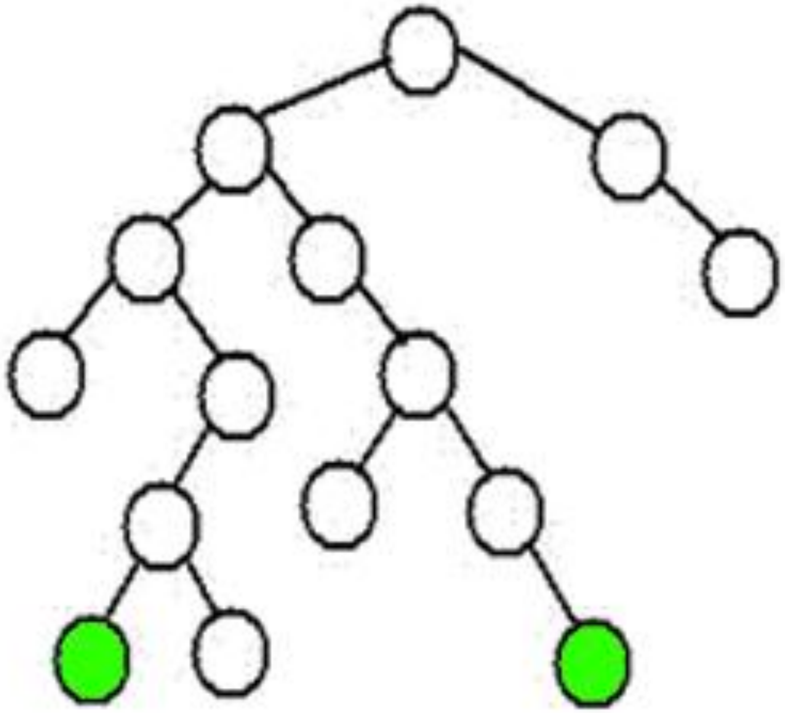
```
      1
    /
   10
  /
 5
```

4. Diameter of a Binary Tree.

Given a Binary Tree, find diameter of it. The diameter of a tree is the number of nodes on the longest path between two leaves in the tree. The diagram below shows two trees each with diameter nine, the leaves that form the ends of a longest path are shaded (note that there is more than one path in each tree of length nine, but no path longer than nine nodes).



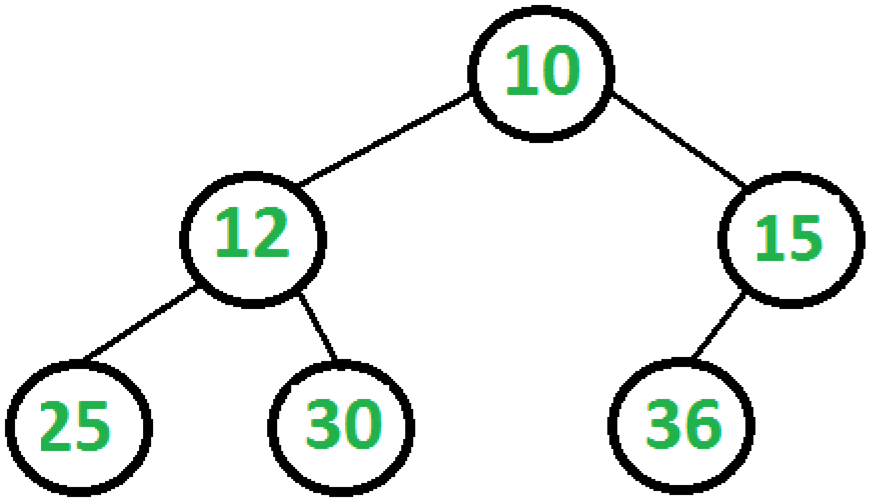
Diameter, 9 nodes, through root



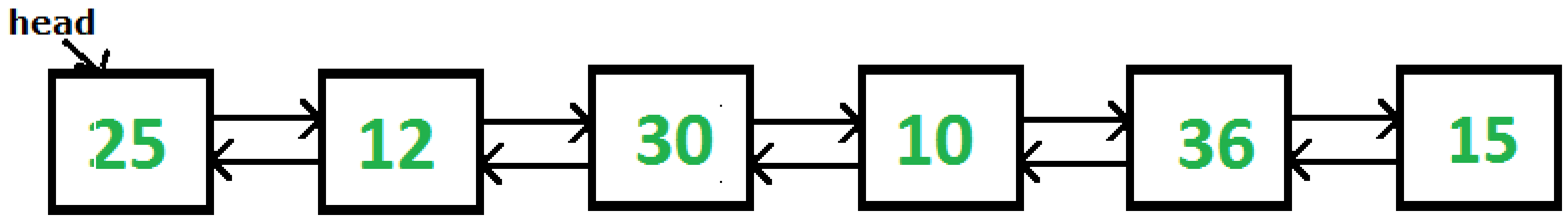
Diameter, 9 nodes, NOT through root

5. Binary Tree to DLL

Given a Binary Tree (BT), convert it to a Doubly Linked List(DLL) In-Place. The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be same as Inorder of the given Binary Tree. The first node of Inorder traversal (left most node in BT) must be head node of the DLL.



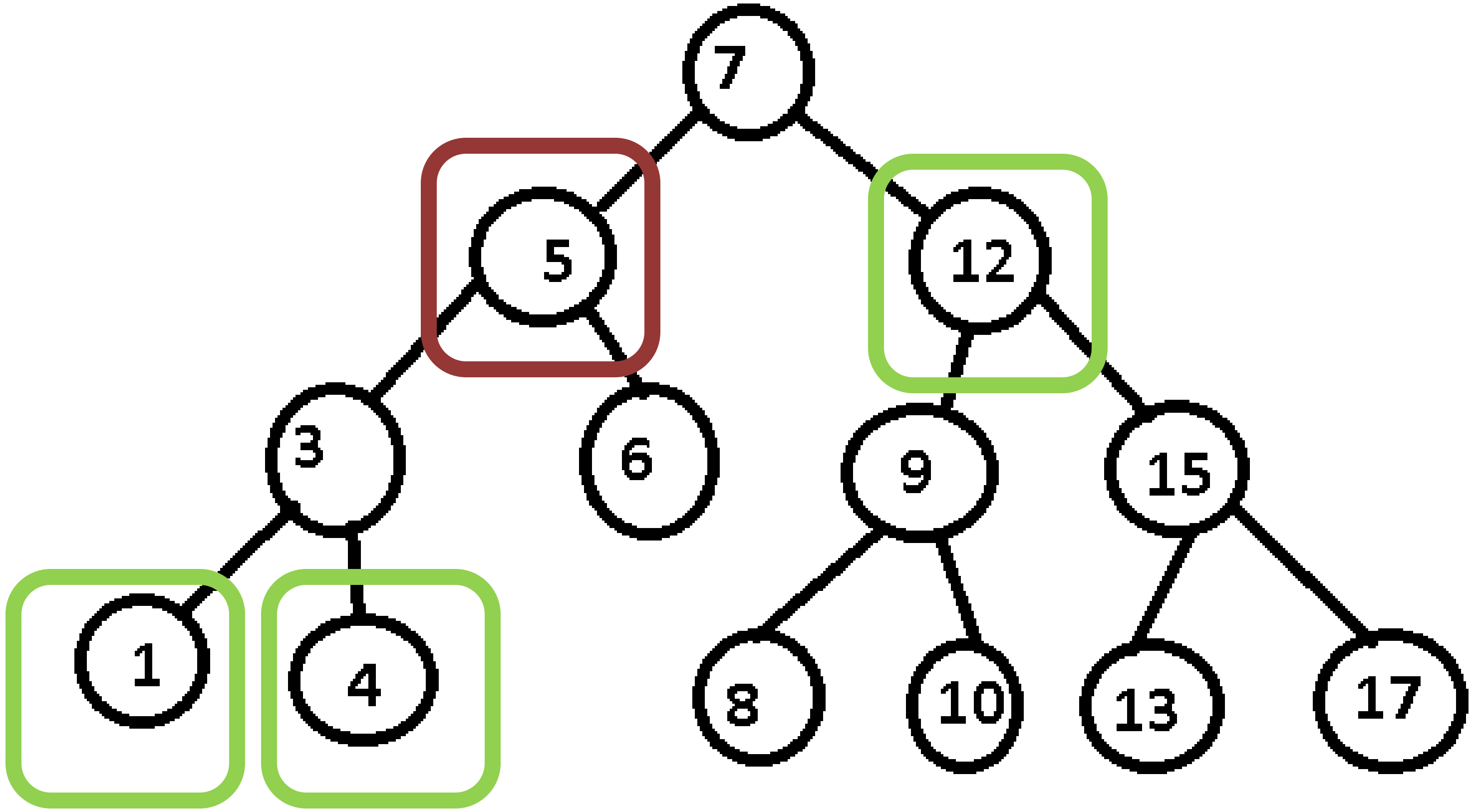
The above tree should be in-place converted to following Doubly Linked List(DLL).



6. Nodes at given distance in Binary Tree

Given a binary tree, a target node in the binary tree, and an integer value k, print all the nodes that are at distance k from the given target node. No parent pointers are available.

K = 2



```
public static int printkdistanceNode(Node root, Node target , int k) {
        if(root == null)
                return -1;
        if(root == target) {
                printAtK(root, k);
                return 1;
        }
        int leftDist = printkdistanceNode(root.left, target, k);
        int rightDist = printkdistanceNode(root.right, target, k);
        if(leftDist == k || rightDist == k)
                System.out.print(root.data + " ");
        if(leftDist < k && leftDist != -1)
                printAtK(root.right, k - leftDist - 1);
        if(rightDist < k && rightDist != -1)
                printAtK(root.left, k - rightDist - 1);
        if(leftDist != -1)
                return leftDist + 1;
        if(rightDist != -1)
                return rightDist + 1;
        return -1;
}
```

```
public static void printAtK(Node root, int k)
{
          if(root == null)
                   return;
         if(k == 0)
                    System.out.print(root.data + " ");
          printAtK(root.left, k-1);          printAtK(root.right, k-1);
}
```