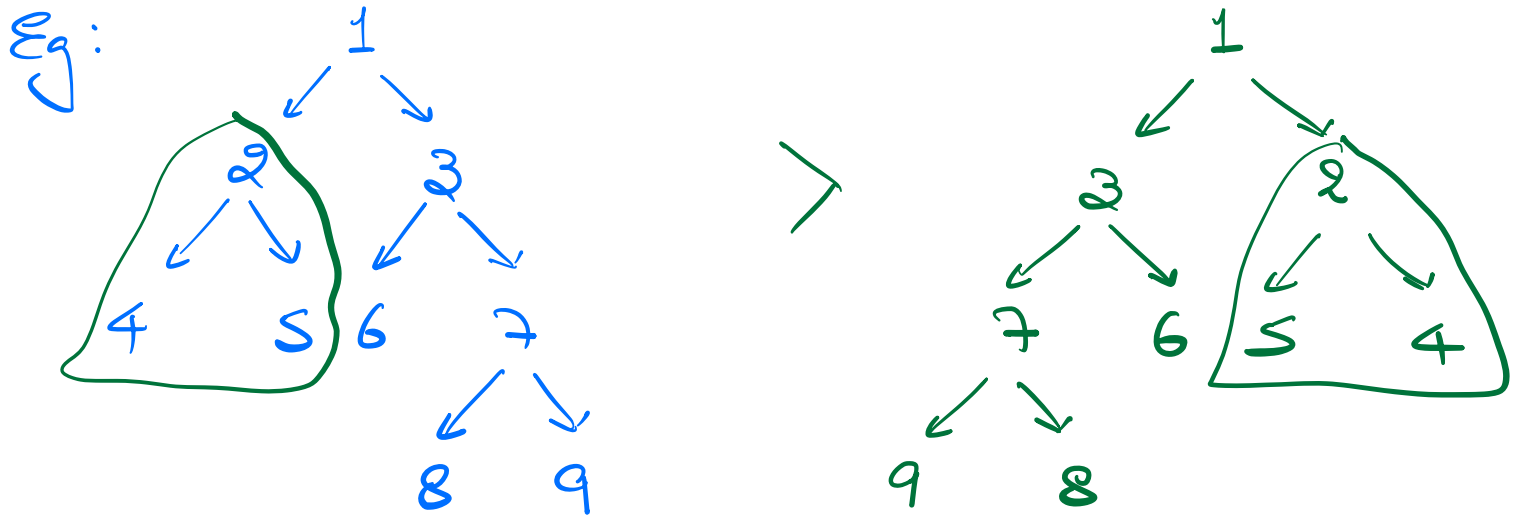
Agenda

1) Invert Binary Tree
2) Equal Tree Partition
3) Next pointer in Binary Tree
4) Root to leaf path sum = K
5) Diameter of Binary Tree.

Given the root node of a Binary Tree, write a function to invert the tree

Eg:



Inversion

Steps

1) Invert the LST
2) Invert the RST
3) Make left as right & vice versa.

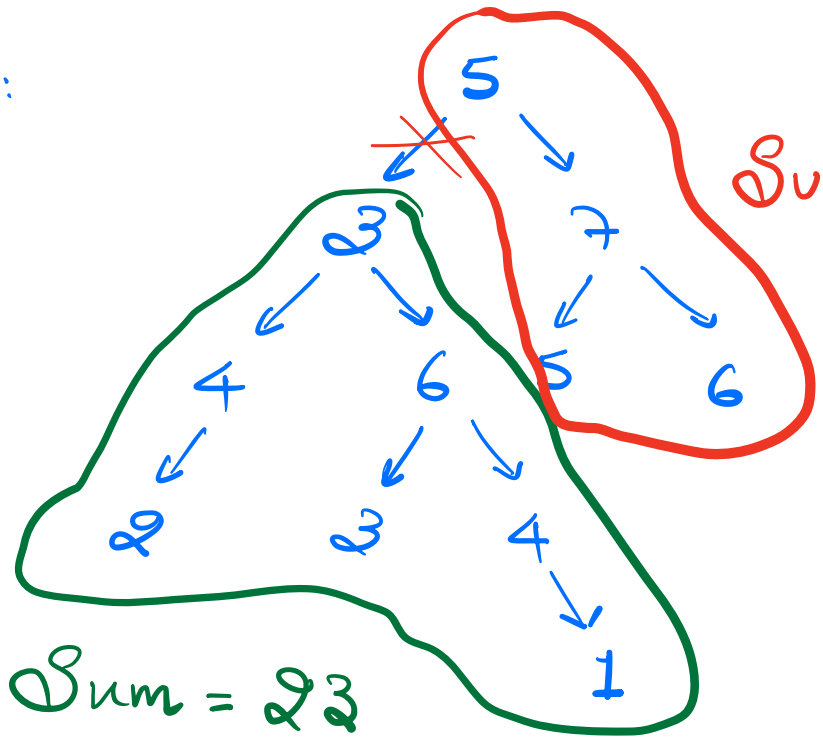
Code

```
void invertTree( Node root) {
    if (root == NULL) { return; }

    Node temp = root.left;
    root.left = root.right;
    root.right = temp;

    invertTree (root.left);
    invertTree (root.right);
}
```

T.C. = O(N)
S.C. = O(Height);

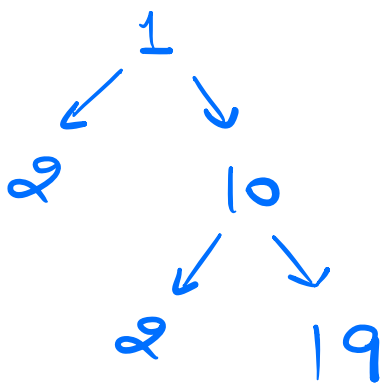
Q2 Given the root node of a Binary Tree,
Return True if the tree can be split into two non-empty subtrees with equal sums. false otherwise.

Eg 1:



True

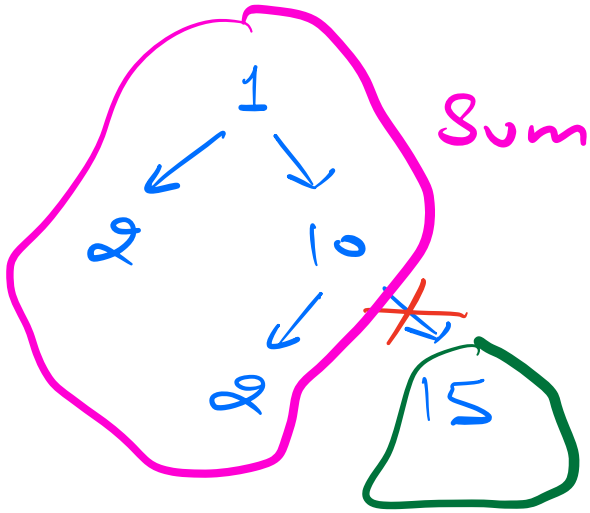
2)



X

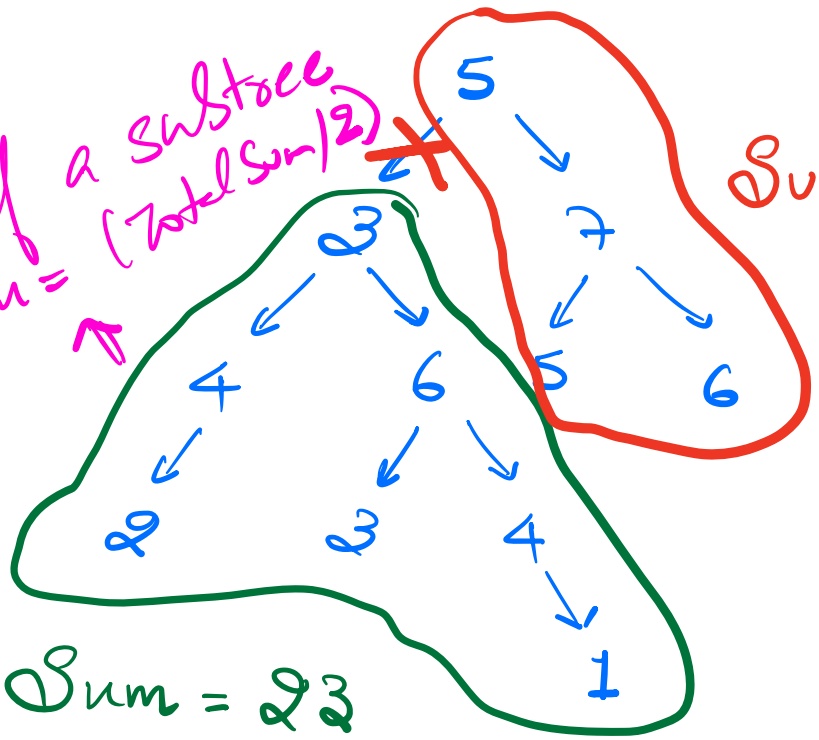
false

3)



Soln

find if a subtree with sum = (Total Sum/2) exists.



True

Total Sum = 46

Steps

1) Calculate Total Sum. (Return false if Odd)

2) ∀ subtree ⇒ Calculate sum & check if it is equal to half of Total Sum.

Code

totalSum (Calculate it first)
ans = false;

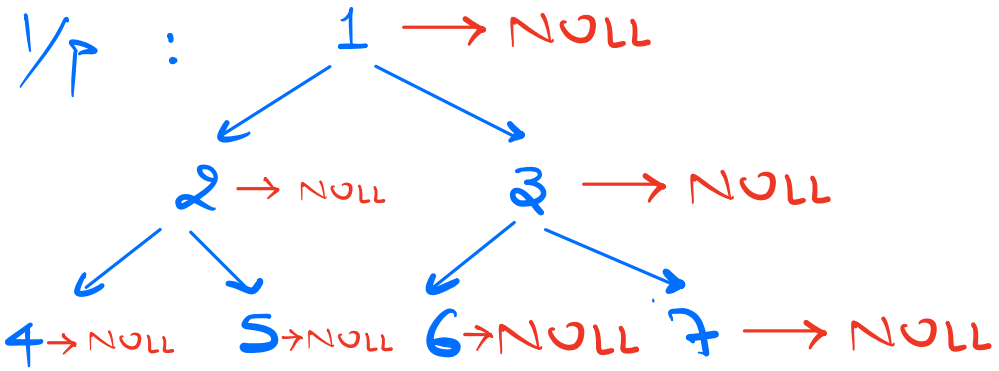
```
int sum (Node root) {
    if (root == NULL) { return 0; }

    int sum_left = sum (root.left);
    int sum_right = sum (root.right);

    if ( sum_left == totalSum/2 ||
         sum_right == totalSum/2 ) {

        ans = true;
    }

    return (sum_left + sum_right +
            root.data);
}
```

T.C. = O(N)

S.C. = O(Height)

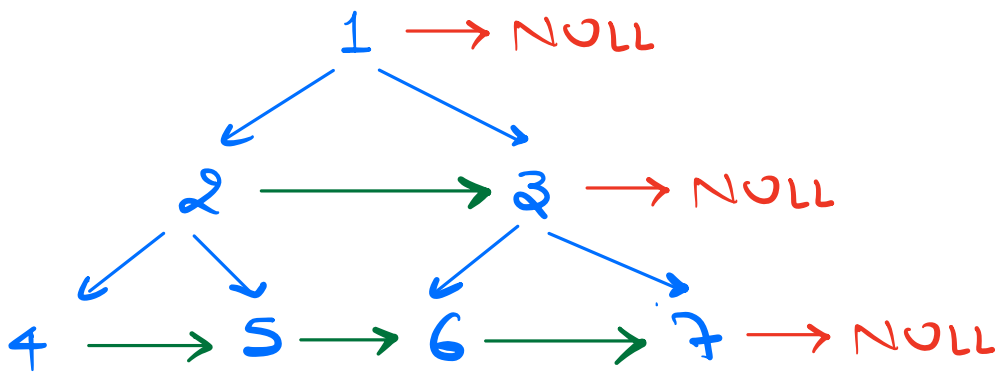
3 Given a perfect binary tree with all next pointers set to NULL,

Modify the tree in-place (S.C. = O(1)) to the next node in the same level from left to right.




```
Node {
    int data
    Node left
    Node right
    Node next;
}
```

O/p :



Soln ⇒ Brute force → level Order Traversal.

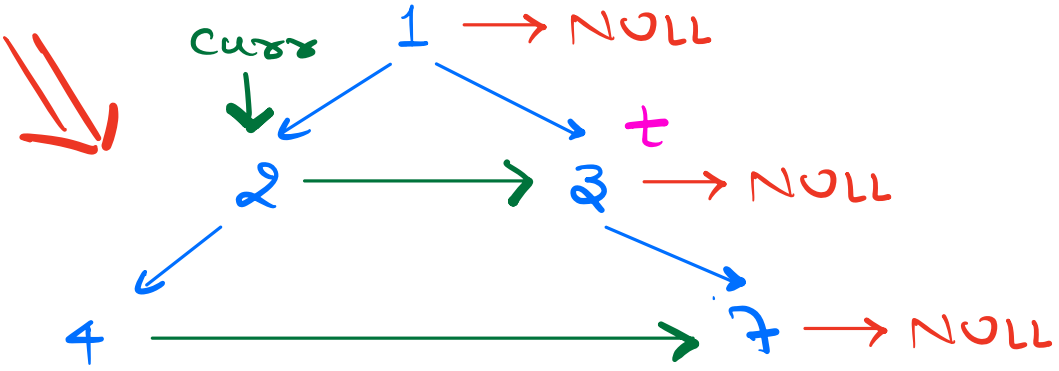
```
Node connect (Node root) {
    if (root == NULL) {
        return NULL;
    }

    Queue<Node> q;
    q.enqueue(root);
    while (!q.isEmpty()) {
        int levelSize = q.size();
        for (i=0; i<levelSize; i++) {
            Node temp = q.dequeue();
            if (i < (levelSize-1)) {
                temp.next = q.front;
            }
            if (temp.left != NULL) {
                q.enqueue(temp.left);
            }
            if (temp.right != NULL) {
                q.enqueue(temp.right);
            }
```

```
    }
  }
  return root;
}
```

T.C. = O(N)

S.C. = O(N)

H.W.



Code

```
curr = root;

while (curr.left != NULL) {
    temp = curr;
    while (temp != NULL) {
        temp.left.next = temp.right;
        if (temp.next != NULL) {
            temp.right.next = temp.next.left;
        }
        temp = temp.next;
    }
```

curr = curr.left;

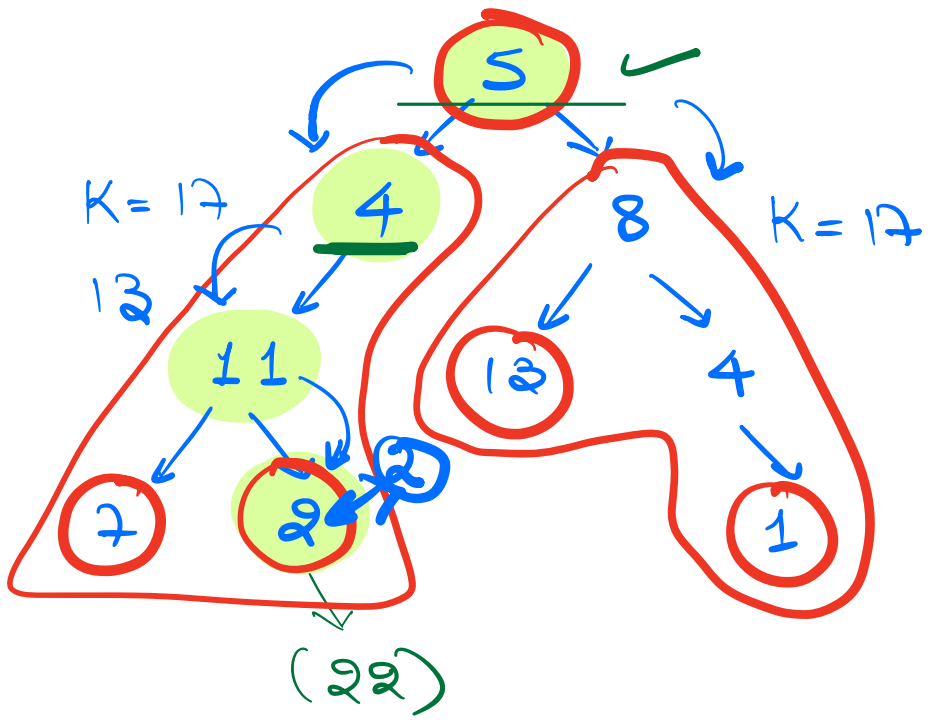
}

T.C. = O(N)

S.C. = O(1)

Q4 Given a Binary Tree. Check if there exists a root to leaf path in the tree such that adding all the node values along the path has sum = K. (Given).

I/P :



K = 22

True

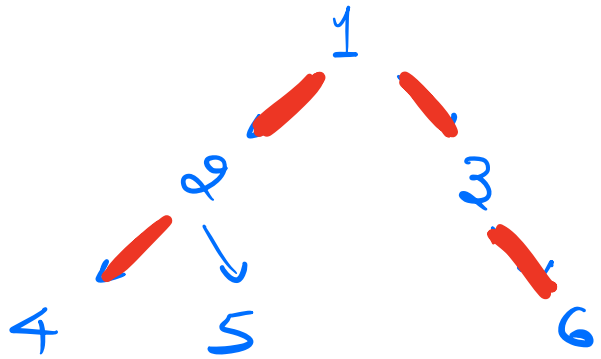
Soln

Code

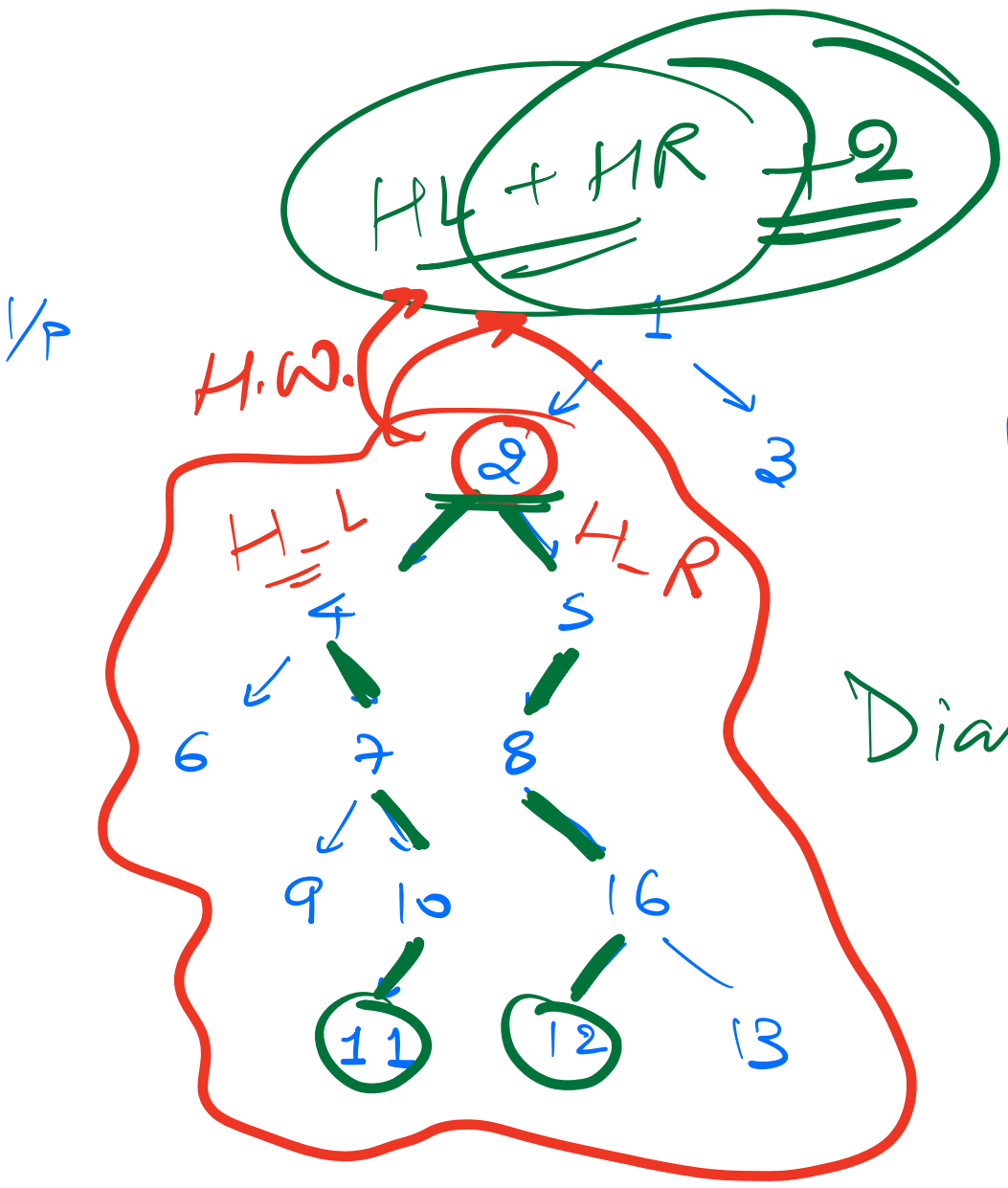
```
boolean hasPathSum (root, K) {
    if (root == NULL) {
        return false;
    }

    if (root.left == NULL && root.right == NULL) {
        return (K == root.data);
    }

    return hasPathSum(root.left, K - root.data)
        || hasPathSum(root.right, K - root.data);
}
```

Q5 Given a Binary Tree. find the length of the longest path between any two nodes! (diameter)

I/P :

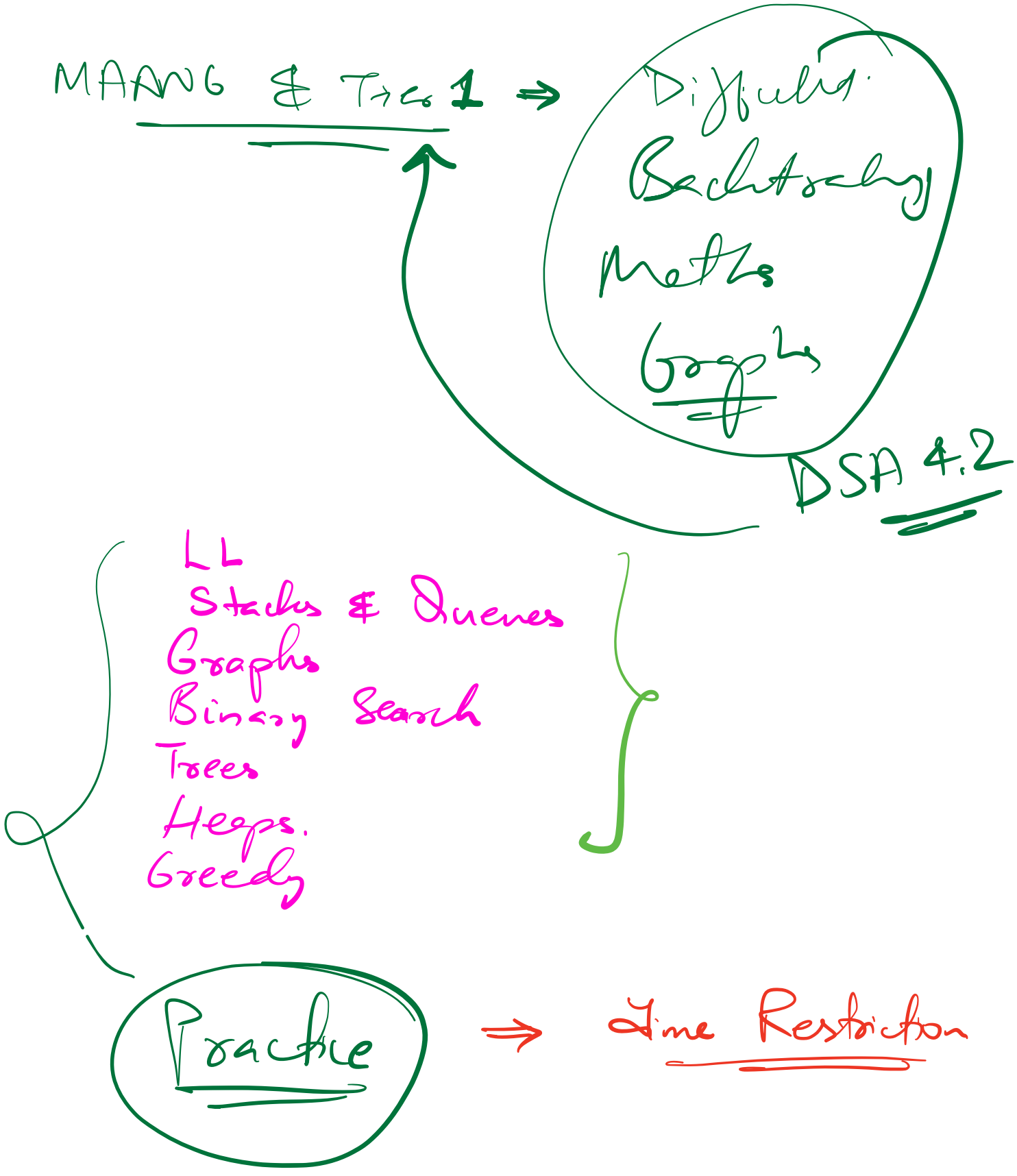


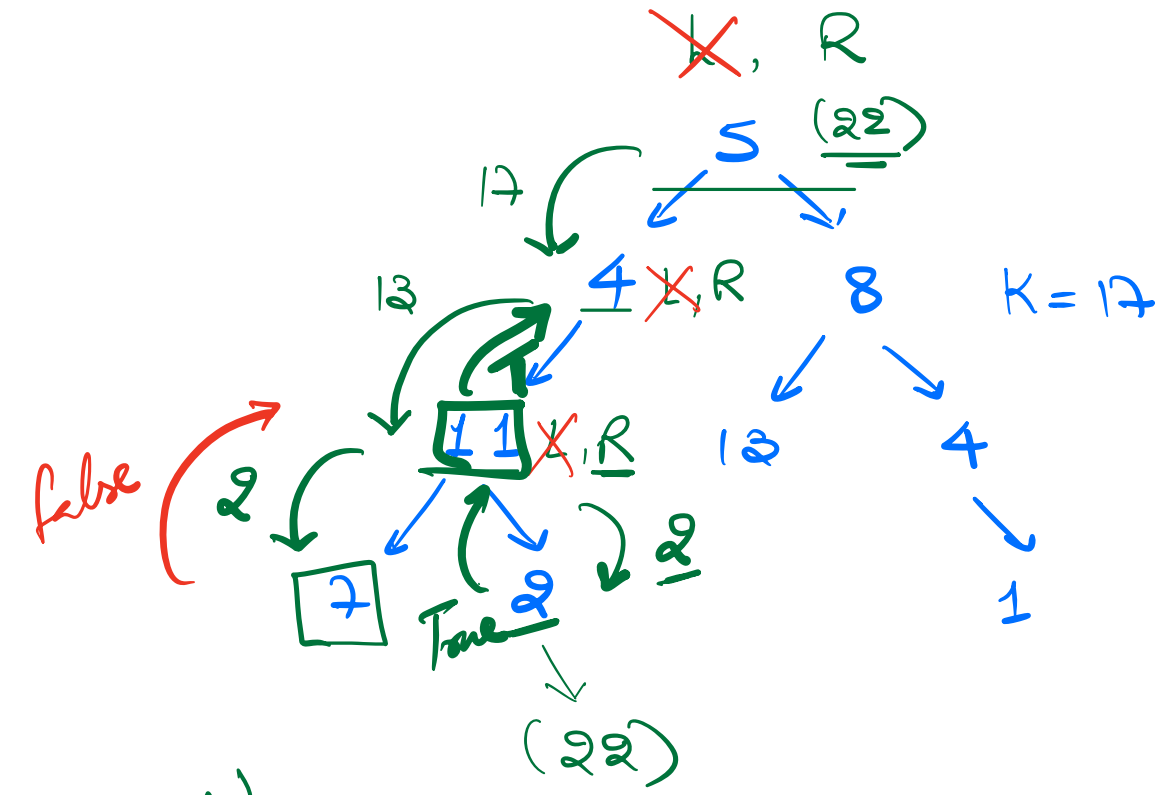
Diameter = 4



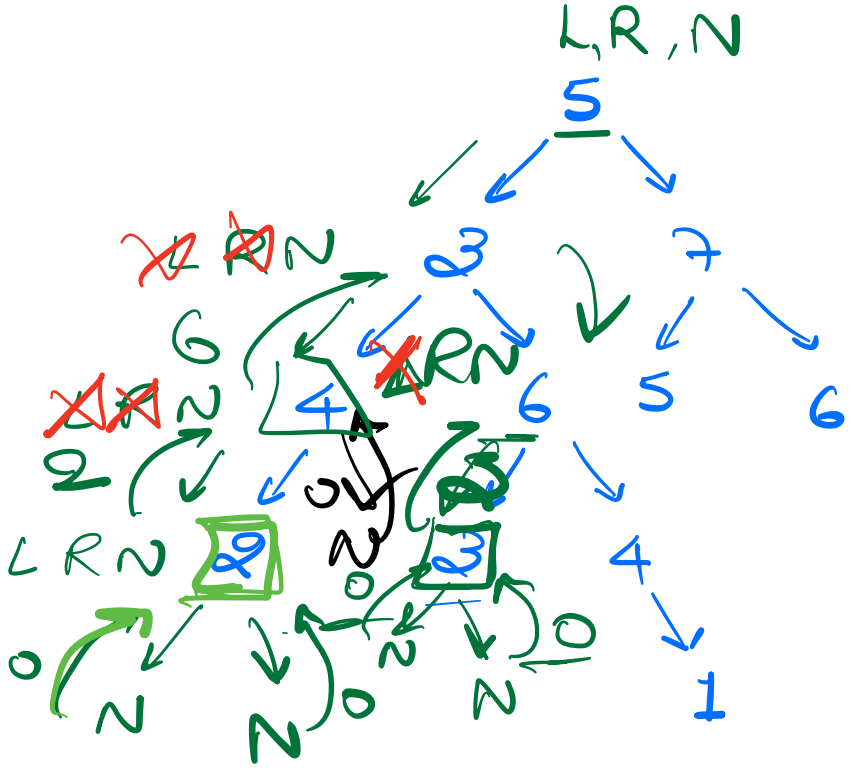
Post Order

Diameter = 8

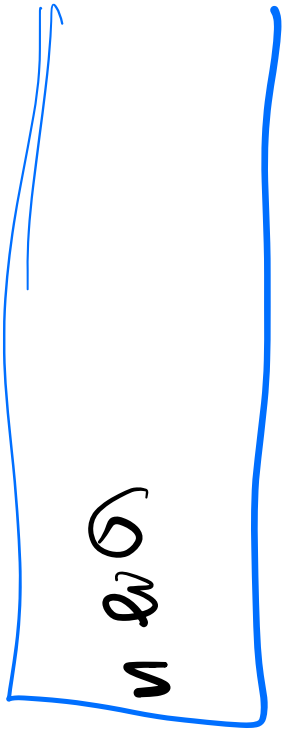

MANG & Tier 1 ⇒
Difficult
Backtracking
Maths
Graphs
DSA 4.2
LL
Stacks & Queues
Graphs
Binary Search
Trees
Heaps.
Greedy
Practice
⇒
Time Restriction



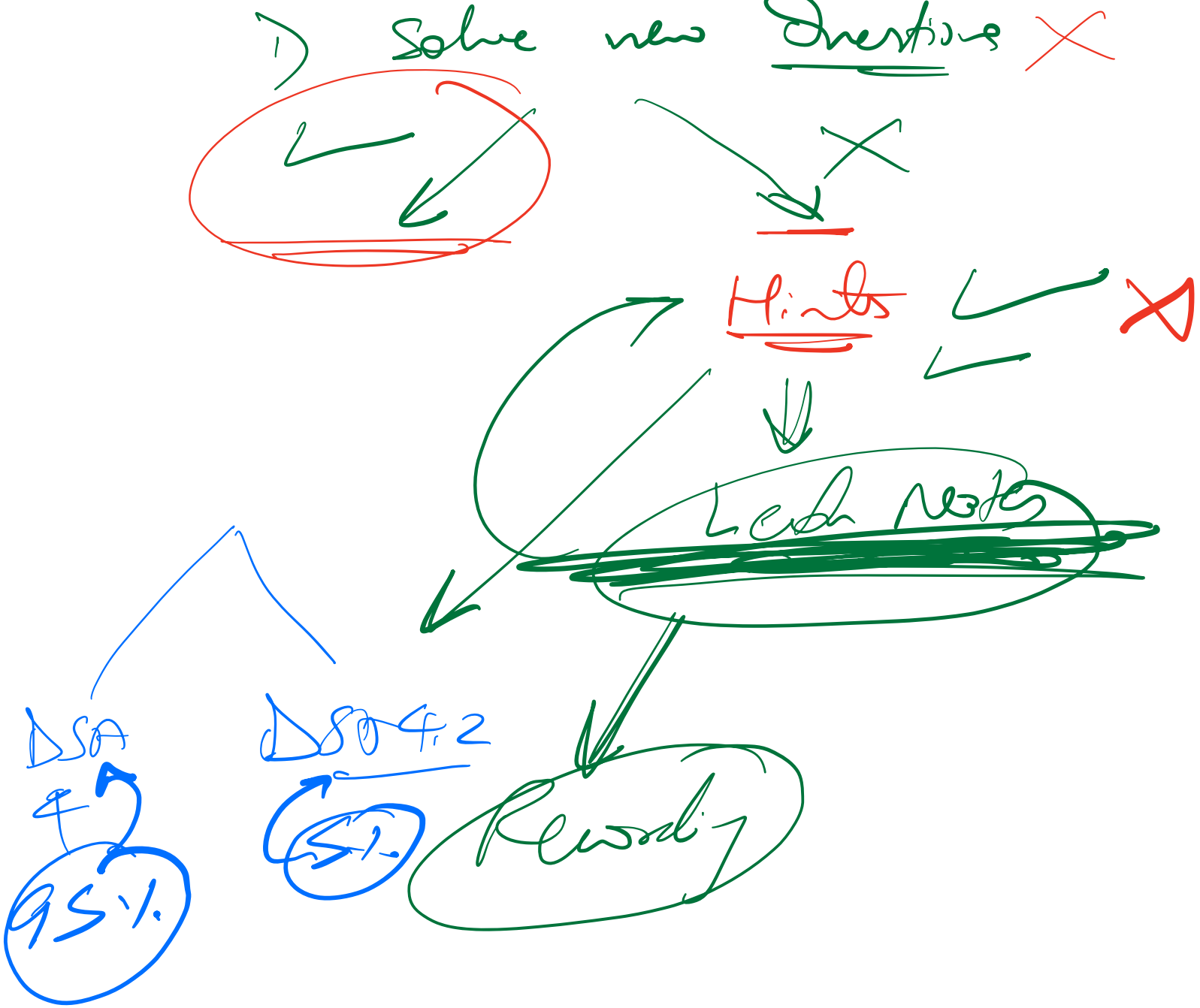
K = 22



46



Revision →


Solve new Questions
Hints
Leetcode
DSA
DSA 2
Revision