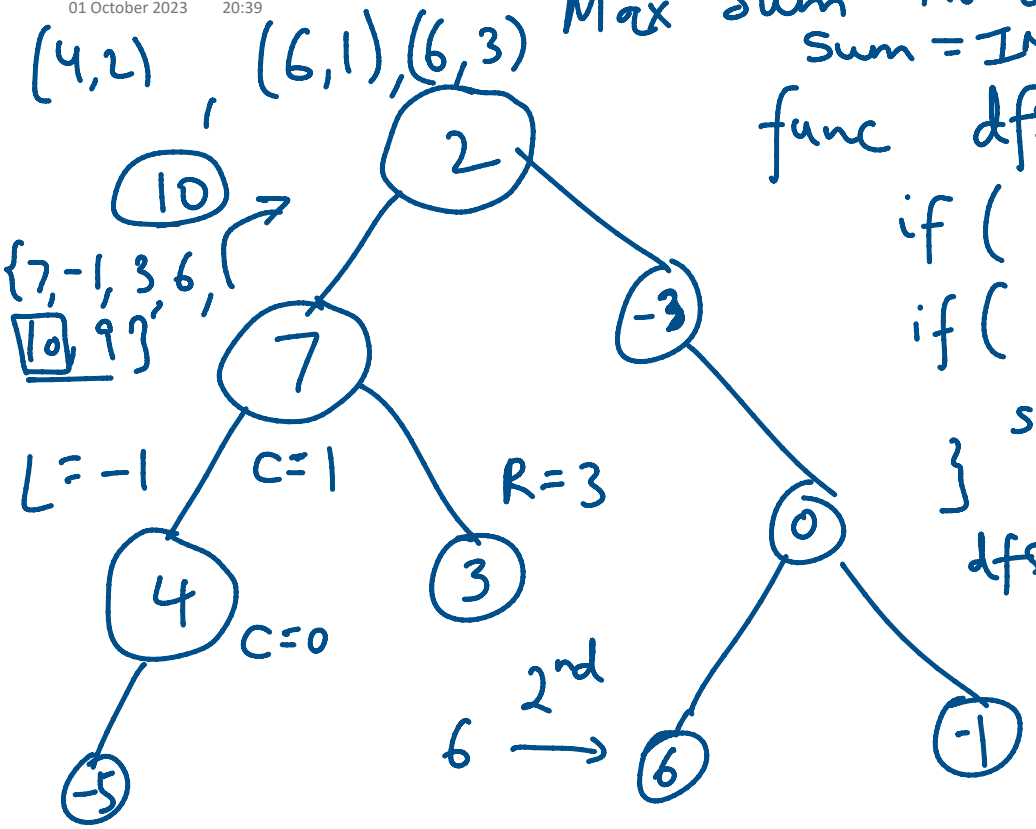
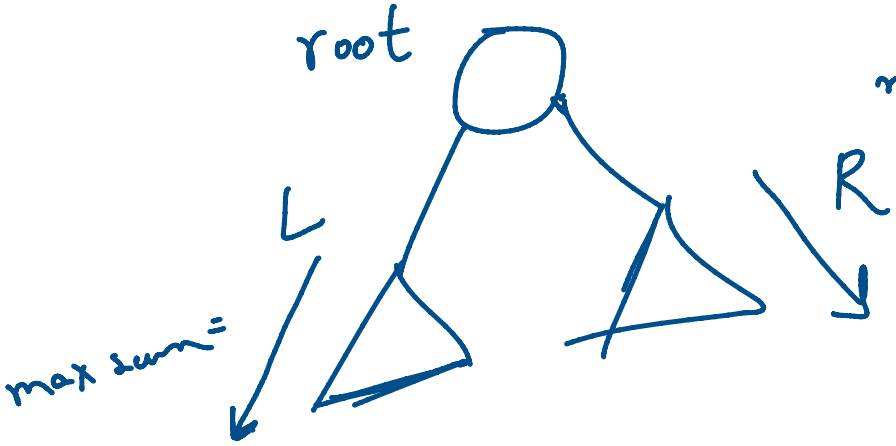
01 October 2023 20:39

Max Sum Root to leaf




```
Sum = INT_MIN
func dfs( root, cur=0){
  if (!root) {return}
  if ( leaf ){
    sum = max(sum, cur)
  }
  dfs( root → left,
       cur + root → data
  "  right
```

3



max Sum BW Any 2 Nodes

L+R ✗

(L, R, L + R + root→data, L + root→data, R+root→d

, root → data, max_sum)

maxSum = INT_MIN

```
int f( root ) {
  if( !root)   return   INT_MIN
  if( !root → left && !root → right)
```

if(!root → left ++ ...)

return root → data

int L = f(root → left)

int R = f(root → right)

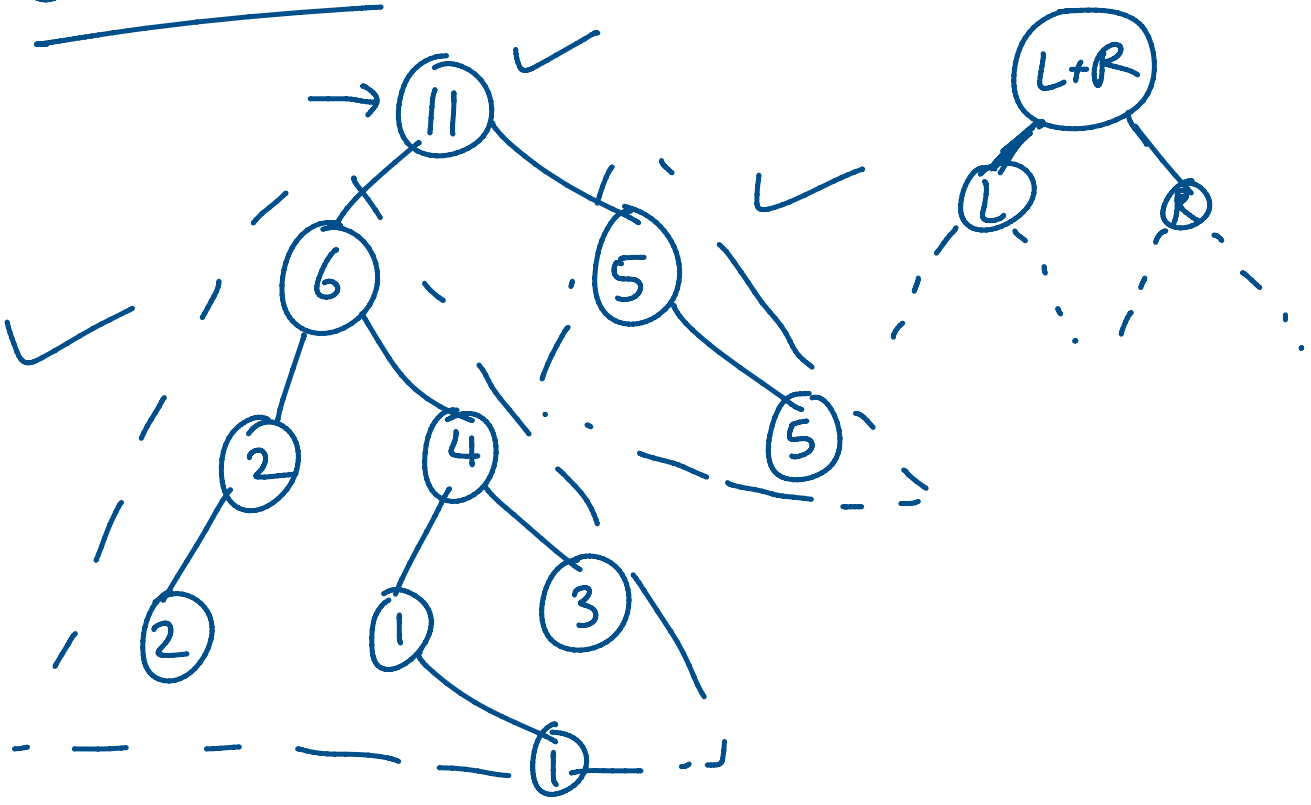
⟹ maxSum = max({maxSum, L, R, ...})

return max({0, L, R}) + root → data

}

maxSum from root to any leaf node

Sum tree →



bool isSumTree(root) {

if(!root) {

```
if( !root ) {
    return true
}

if(!isSumTree(root → left)) {
    return false }
if (!isSumTree(root → right)){
    return false }

int sum = 0
if (root → left)   sum += root → left → data
if (root → right)  sum += root → right → data

return sum == root → data
}
```

→ K^th ancestor

c = -1

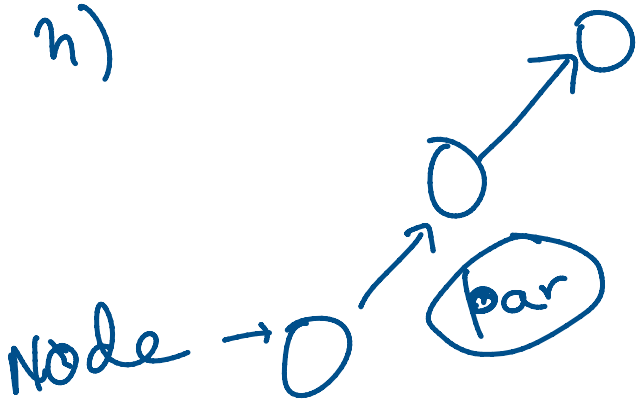
```
void f( root, val, K) {
    if ( !root )  return
    if ( c == K)  print ( root → data)

    if (root → data == val )  c = 0
```

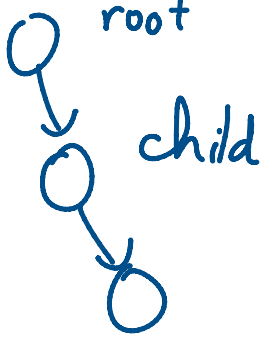
```
if( c == -1) {
    f( root → left, val, K)
}
if ( c == -1) {
    f( root → right, val, K)
}
}
```

⇒ Tree ✓ , pair (Node , K)

m * O(n) — O(mn)

O(m + n)





Map , map[Node] → parent node

depth[root] = 0

dfs(root) {

```
dfs( root ) {
    if( ! root ) return
    if( ! root → left) {
        par[ root → left] = root → data
map →   depth[root → left] = depth[root] + 1
    }
    if( ! root → right )
        par[root → right] = root
    dfs ( root → left)
    dfs (root → right)
}
```

→ loop → (val, K)

```
while ( K > 0 ) {
    Node = par[Node]
    K--
}
print (Node)
```

LCA (Lowest Common Ancestor)

(15)

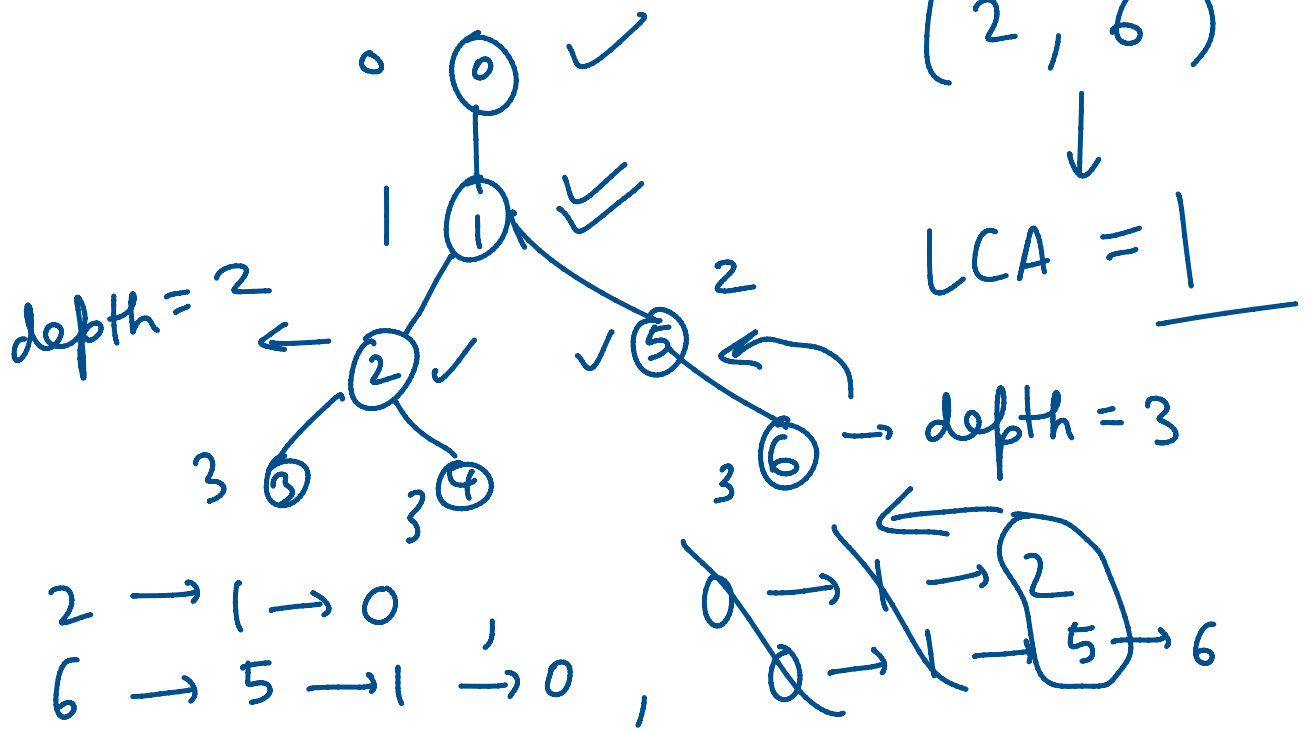
LCA (Lowest Common Ancestor)

(1,5)

(2,6)

↓

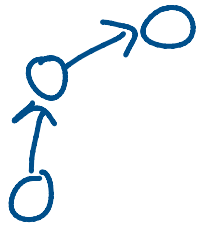
LCA = 1



2 → 1 → 0 ,

6 → 5 → 1 → 0 ,

```
int LCA( n1 , n2 , par , depth ) {
    if( depth[n2] > depth[n1] ) {
            swap(n1, n2)
    }
    K = depth[n1] - depth[n2]
    while (K--)
        n1 = par[n1]
    if( n1 == n2)
        return n1
```



```
        return  n1
    while ( n1 != n2 )
        n1 = par[n1]
        n2 = par[n2]
    }
    return n1
}
```