Dynamic Programming

Video - 74

Note :- This playlist is only for explanation of dns & solutions.

See my "DP Concepts & dns" playlist for understanding DP from scratch...

Facebook, Instagram → codestorywithMIK

Twitter → cswithMIK

→ codestorywithMIK

Leetcode - 2920 Hard

Maximum Points

After Collecting Coins From All Nodes

2920. Maximum Points After Collecting Coins From All Nodes

Hint

Hard 108 10

Companies

→ options

There exists an undirected tree rooted at node `0` with `n` nodes labeled from `0` to `n - 1`. You are given a 2D **integer** array `edges` of length `n - 1`, where `edges[i] = [a_i, b_i]` indicates that there is an edge between nodes `a_i` and `b_i` in the tree. You are also given a **0-indexed** array `coins` of size `n` where `coins[i]` indicates the number of coins in the vertex `i`, and an integer `k`.

k = ?

Starting from the root, you have to collect all the coins such that the coins at a node can only be collected if the coins of its ancestors have been already collected.

2 10 → (10-k) → (10/2)

Coins at `node_i` can be collected in one of the following ways:

- Collect all the coins, but you will get `coins[i] - k` points. If `coins[i] - k` is negative then you will lose `abs(coins[i] - k)` points.
- Collect all the coins, but you will get `floor(coins[i] / 2)` points. If this way is used, then for all the `node_j` present in the subtree of `node_i`, `coins[j]` will get reduced to `floor(coins[j] / 2)`.

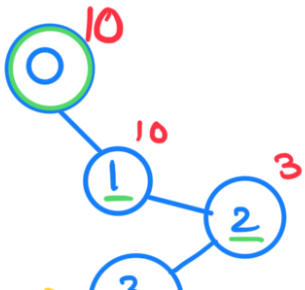
Return *the* ***maximum points*** *you can get after collecting the coins from* ***all*** *the tree nodes.*

Example :-

edges = [{0,1}, {1,2}, {2,3}]

Coins = [10, 10, 3, 3]

K = 5



Points = $\frac{(10-K)}{}$ + (10-K) + 3/2 → 3/2

+ (3/2)/2

= 5 + 5 + 1 + 0

= 11

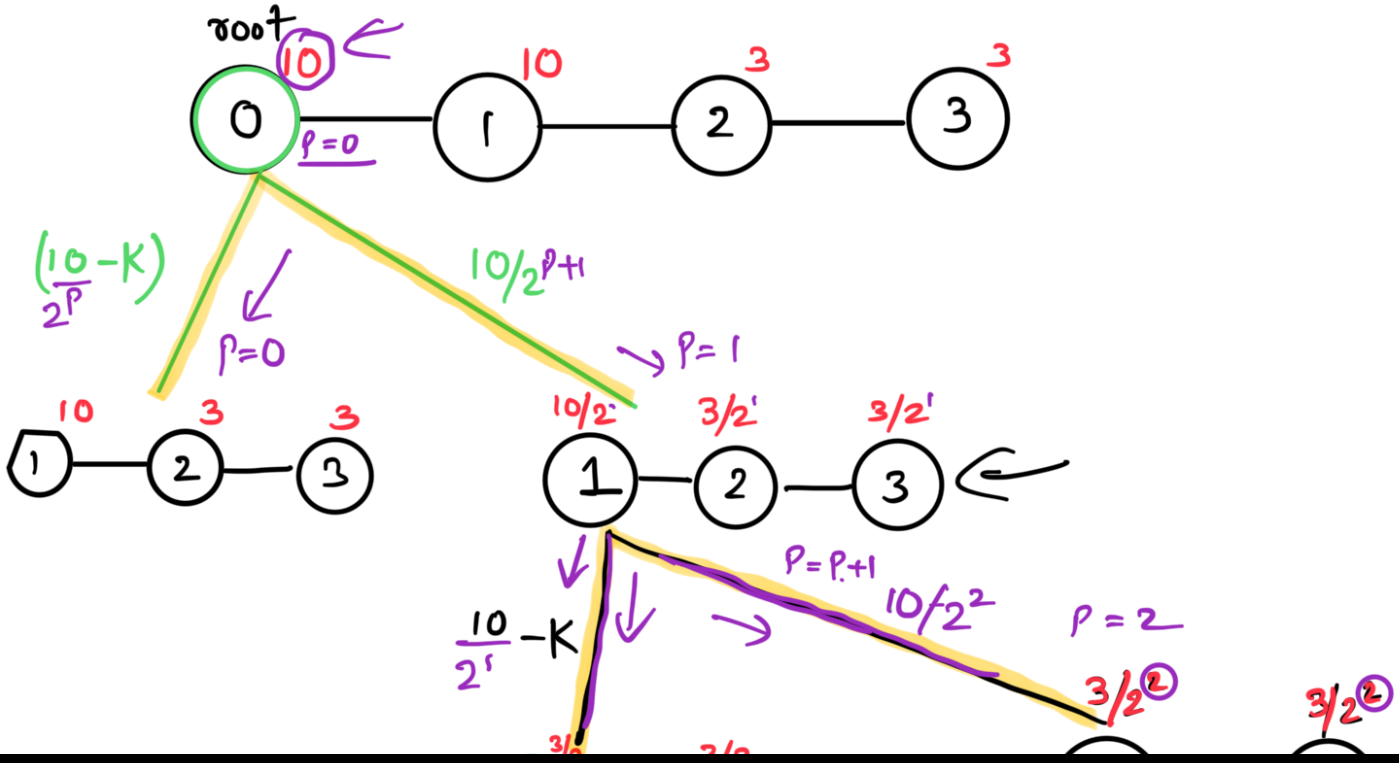
Options :- Recursion

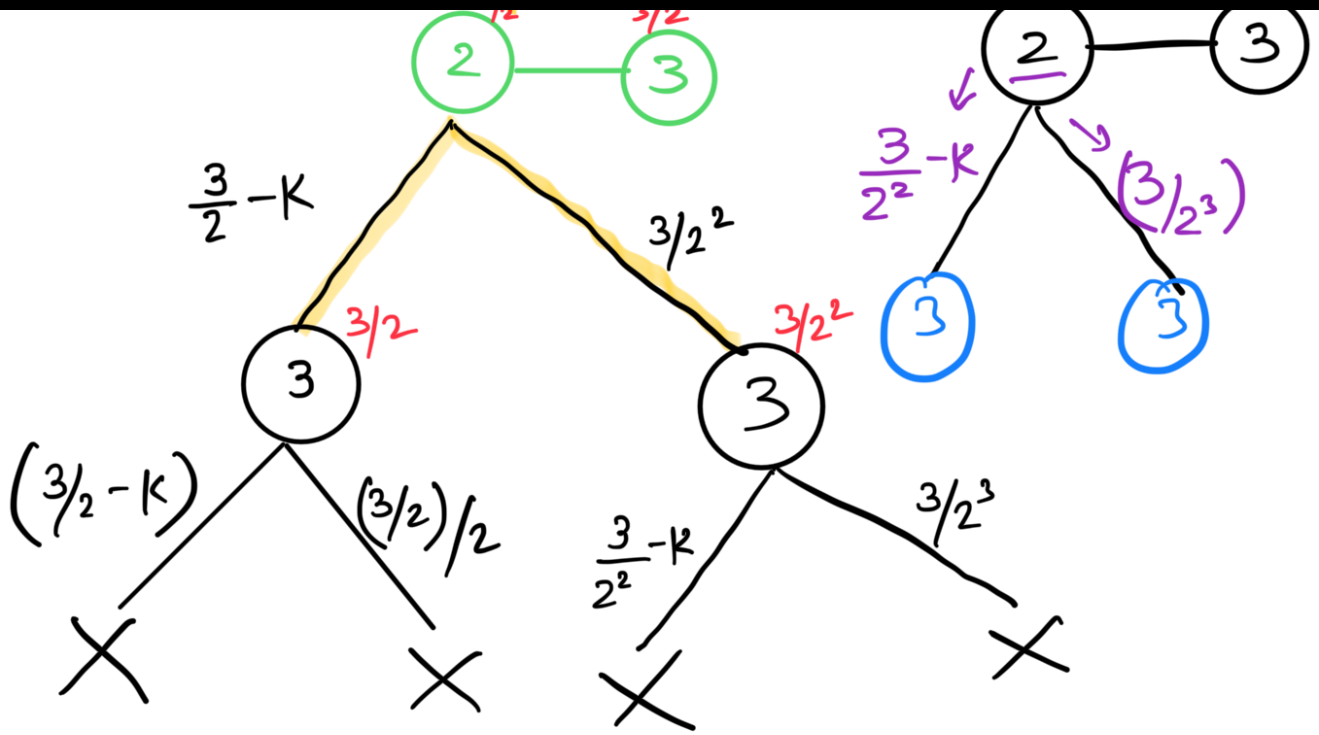
↓

Tree Diagram.

K = 5

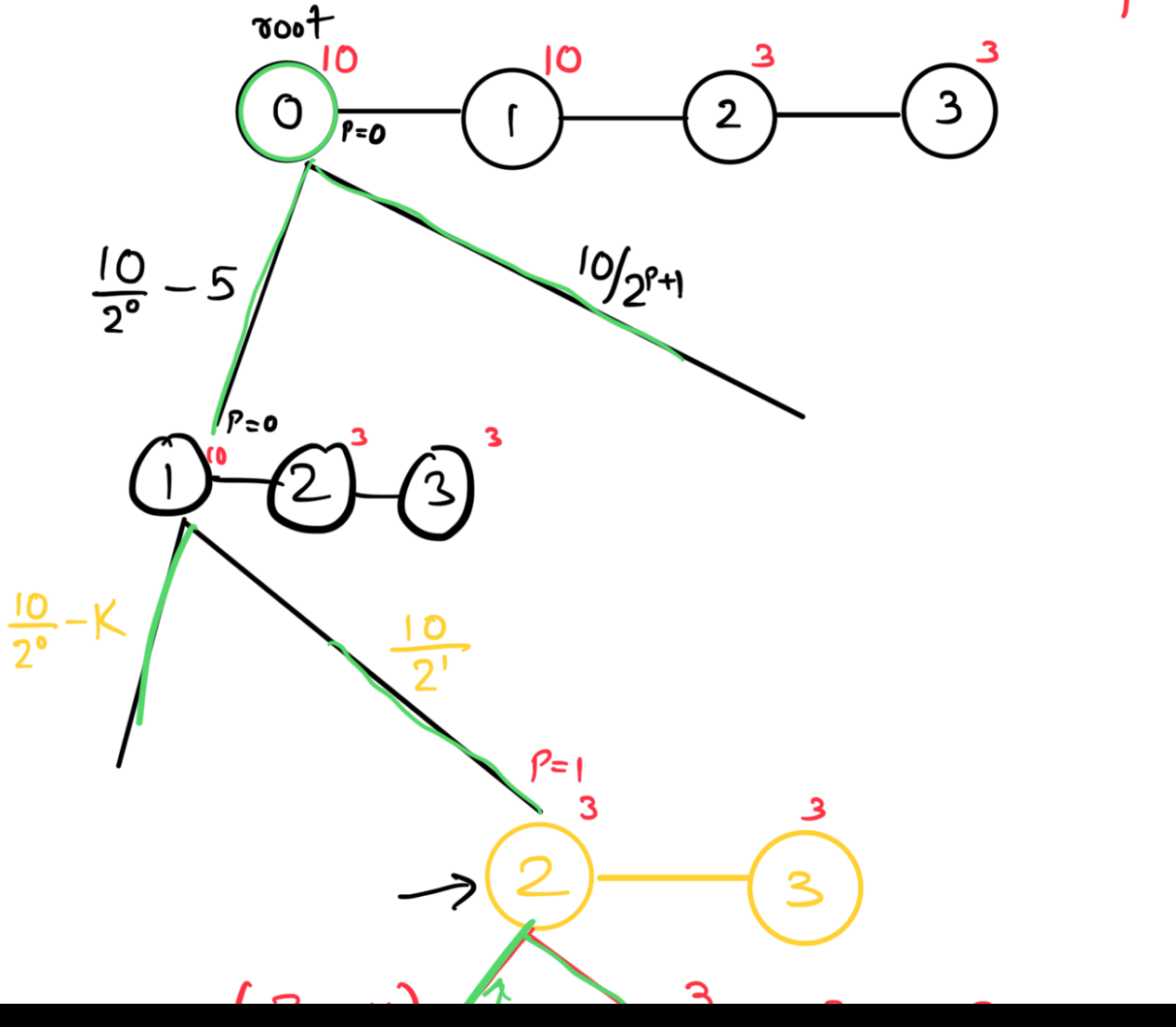
coins = {10, 10, 3, 3} (0, 1, 2, 3)

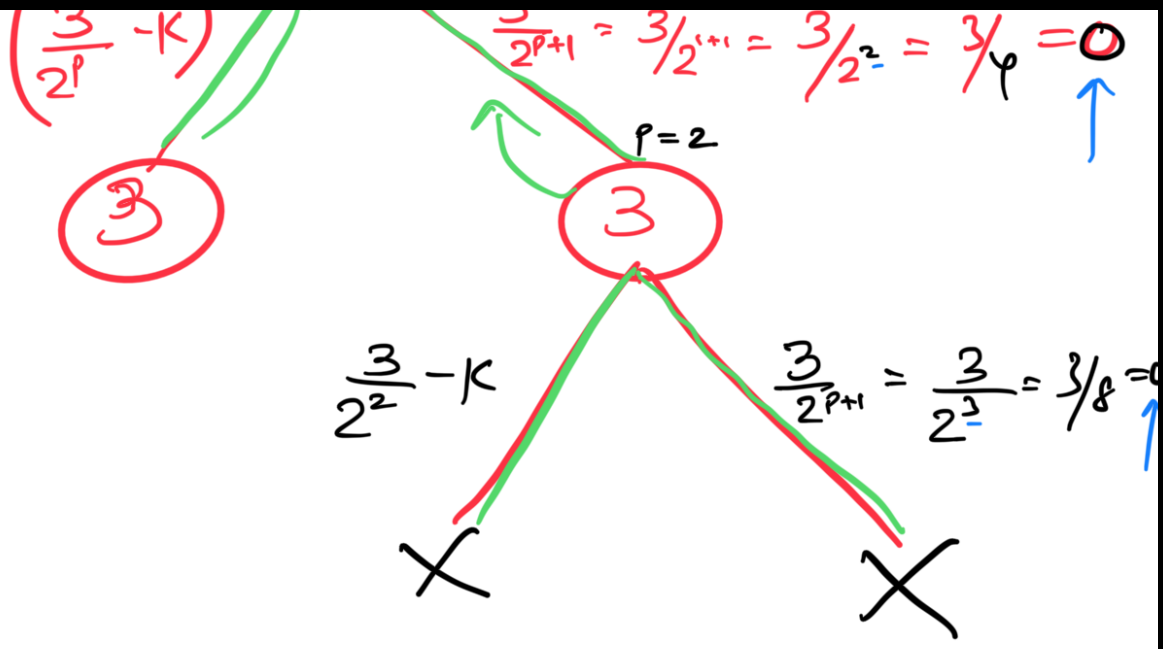




Coins = {10, 10, 3, 3}

K = 5





Story → code...

① adjacency list. (adj)

② DFS → p

③ case1 = (coins[i]/2^p − K) ; //(coins[i] >> p) − k

case2 = coins[i]/2^{p+1} ; //(coins[i] >> (p+1)),

```
for ( int & j : adj[i]) {
```

```
        case1 += dfs(j, i, p);
        case2 += dfs(j, i, p+1);
    }
}
return max(case1, case2);
}
```


Memoize :-

3D → 3 variables are changing

DFS(i , parent , p) ;

$2 <= n <= 10^5$

[100001] [100001] [14]

0 <= coins[i] <= 10

P = 14

(P >= 14)

return 0;

State = (i, Parent, P) = 3D → 2D.

C++

Github :- 3D memoization.

2D Memoization

DFS (i , parent, power) ;

-

```
          i  Parent  P
State =   0,   -1,   0
State =   1,    0,   0
State =   2,    1,   0
State =   3,    2,   0
```

```
State = 3,  2, 1 <-------
State = 2,  1, 1
State = 3,  2, 1 <-------
State = 3,  2, 2 <-------
State = 1,  0, 1
State = 2,  1, 1
State = 2,  1, 2
State = 3,  2, 2 <-------
State = 3,  2, 3
```