Dynamic Programming - Lecture (9)

9 AM - 12 PM

Sunday, 1 May

count of ways
- → Coin Change Permutations
- → Coin change Combinations

minimum steps
- → Minimum Coin change
- → Indian Coin change

Coin change → minimum coins (Indian)

eg1 {1, 2, 5, 10}

target = 79

Why greedy is working?

(10 * 7) + (5 * 1) + (2 * 2)

7 + 1 + 2 = 10 coins

eg2 uniformity coins multiples of 2, 5

{1, 2, 5, 10, 20, 50, 100, 500, 2000} ← Indian denomination ⇒ greedy will also work

target: 2499

2000 * 1 + 100 * 4 + 50 * 1 + 20 * 2 + 5 * 1 + 2 * 2

1 + 4 + 1 + 2 + 1 + 2 = 11 coins

```java
static List<Integer> minPartition(int target)
{
    List<Integer> res = new ArrayList<>();
    int[] coins = {1, 2, 5, 10, 20, 50, 100, 200, 500, 2000};
    int count = 0;

    for(int i=coins.length-1; i>=0; i--){

        while(target - coins[i] >= 0){
            res.add(coins[i]);
            target -= coins[i];
        }

        if(target == 0) break;
    }

    return res;
}
```

Time Comp

↳ $\frac{N}{2000} + \frac{N}{500} + \frac{N}{200} + \cdots \frac{N}{1}$

$\Rightarrow O(N + N + N + \cdots N) = O(N*10)$

$\approx O(N)$

Space Complexity → O(1) extra space

O(n) output space

322. Coin Change → Minimum

Coins = {2, 3, 7} target = 8

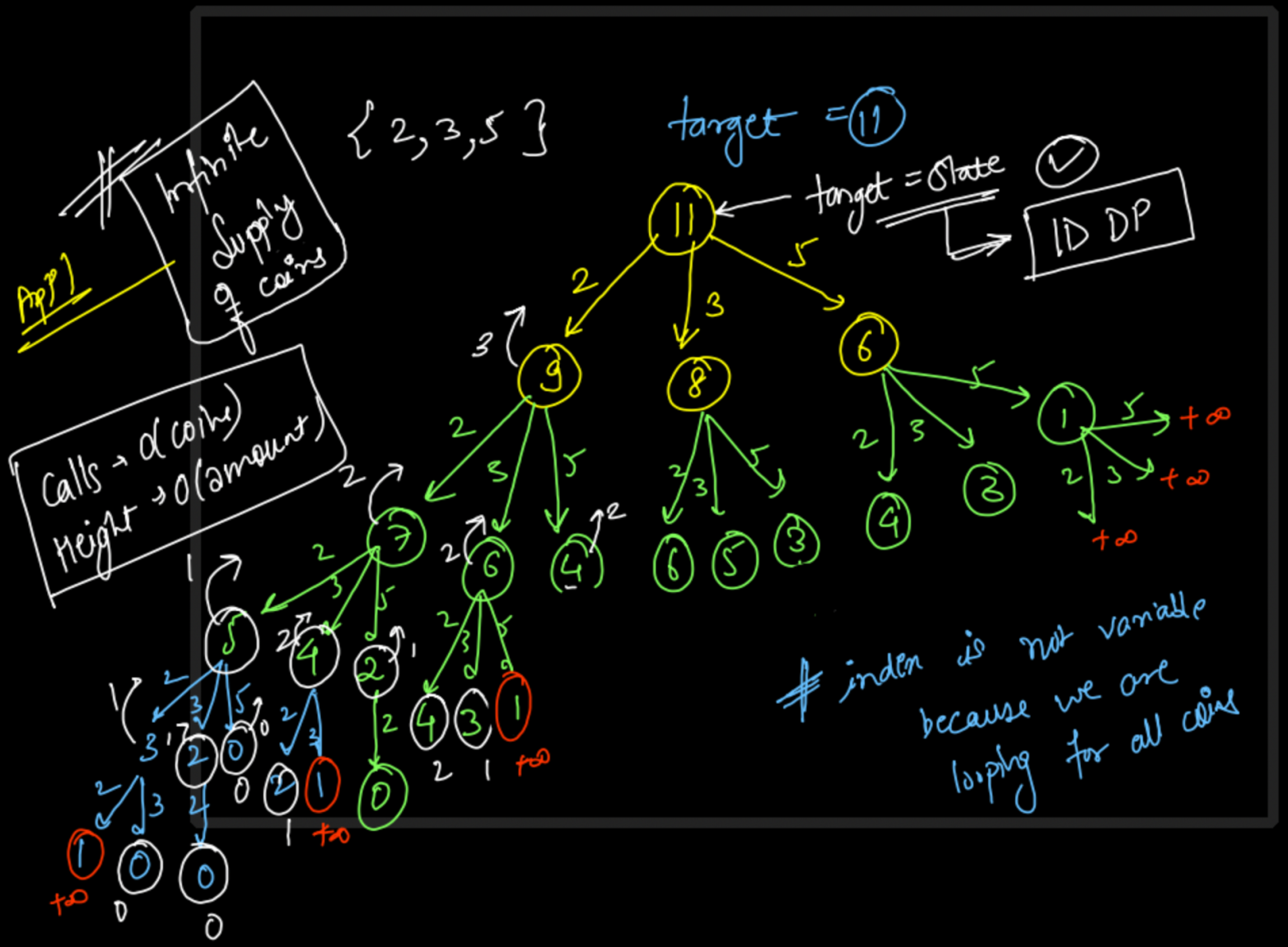
Greedy: target = 8 − 7 = 1 → target. } no answer possible

Actual res: target = 8 − 3 = 5 (not the biggest coin)

5 − 3 = 2

2 − 2 = 0 → target

ans = 3


Infinite supply of coins
{2,3,5}
target = 11
target = State
1D DP
Calls → O(coins)
Height → O(amount)
index is not variable
because we are
looping for all coins
0.7 x

```java
// Time Complexity: O(Amount * Coins), Space Complexity: O(Amount)
public int memo(int amount, int[] coins, int[] dp){
    if(amount == 0) return 0;
    if(dp[amount] != -1) return dp[amount];

    int minCoins = Integer.MAX_VALUE;
    for(int i=0; i<coins.length; i++){
        if(amount - coins[i] >= 0){
            minCoins = Math.min(minCoins, memo(amount - coins[i], coins, dp));
        }
    }

    if(minCoins < Integer.MAX_VALUE) minCoins += 1;
    return dp[amount]  = minCoins;
}

public int coinChange(int[] coins, int amount) {
    int[] dp = new int[amount + 1];
    Arrays.fill(dp, -1);

    int ans = memo(amount, coins, dp);
    return (ans == Integer.MAX_VALUE) ? -1 : ans;
}
```

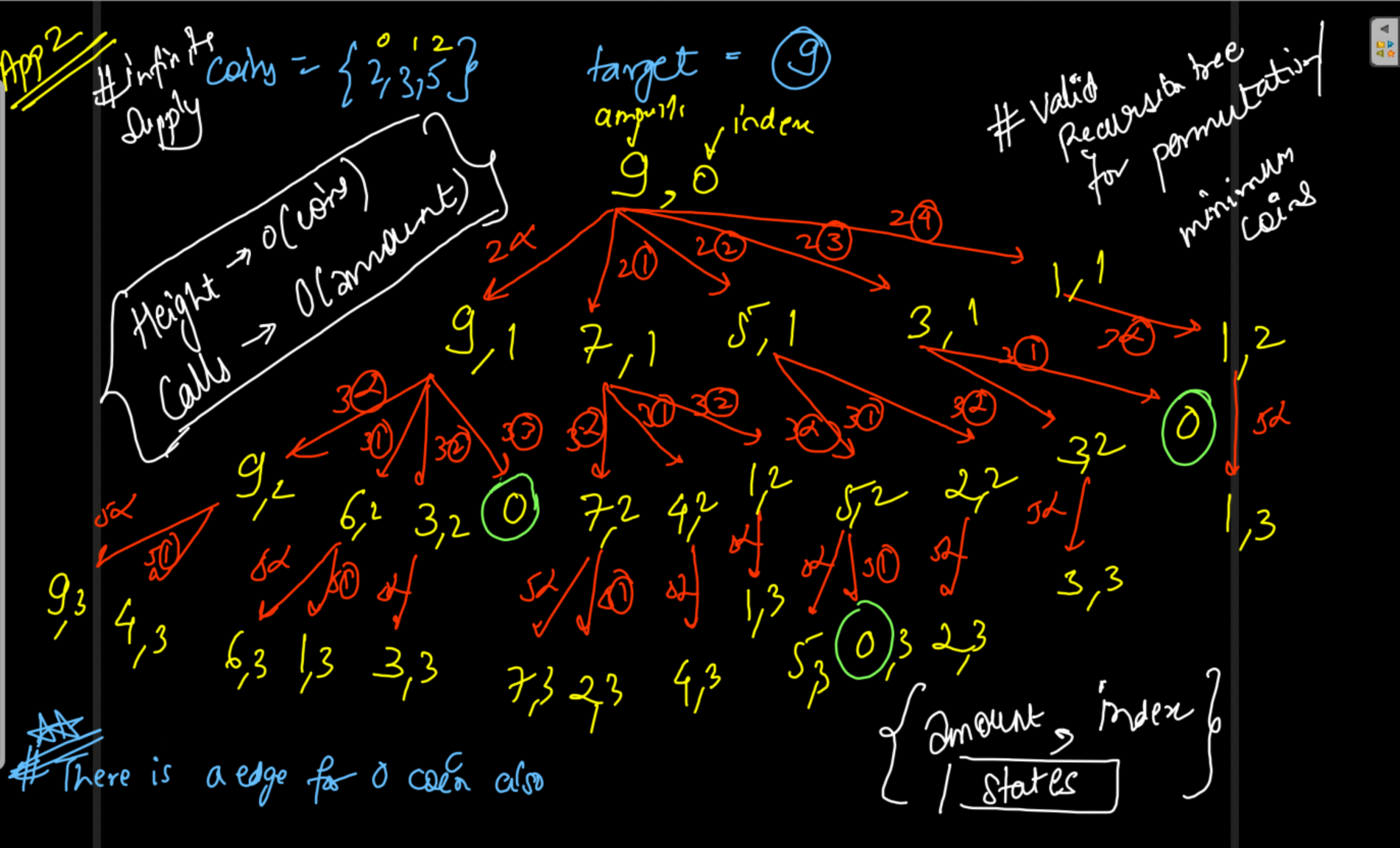
App 2

infinite supply

coins = {2, 3, 5}

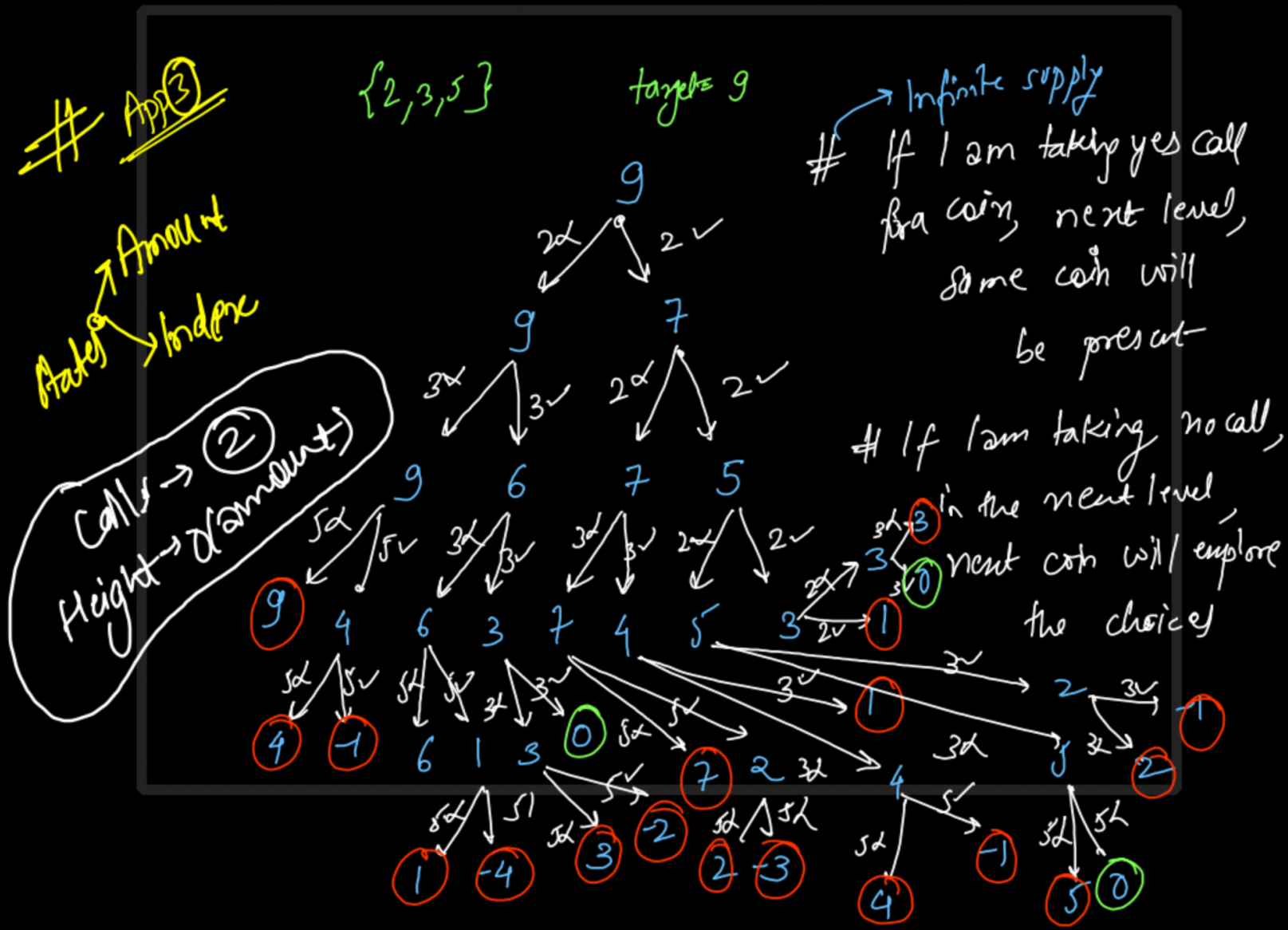
target = 9

amount, index



```java
// Time Complexity: O(Amount * Coins), Space Complexity: O(Amount * Coins)
public int memo(int amount, int idx, int[] coins, int[][] dp){
    if(amount == 0) return 0;
    if(idx == coins.length) return Integer.MAX_VALUE;
    if(dp[amount][idx] != -1) return dp[amount][idx];

    int minCoins = Integer.MAX_VALUE;
    for(int coin=0; amount >= coins[idx]*coin; coin++){
        int ans =  memo(amount - coins[idx] * coin, idx + 1, coins, dp);
        if(ans < Integer.MAX_VALUE) ans += coin;
        minCoins = Math.min(minCoins, ans);
    }

    return dp[amount][idx] = minCoins;
}

public int coinChange(int[] coins, int amount) {
    int[][] dp = new int[amount + 1][coins.length];
    for(int i=0; i<=amount; i++){
        for(int j=0; j<coins.length; j++){
            dp[i][j] = -1;
        }
    }

    int ans = memo(amount, 0, coins, dp);
    if(ans == Integer.MAX_VALUE) return -1;
    return ans;
}
```

0.5 x


App 3
{2,3,5}
target 9
States → Amount, Index
Calls → (2) Height → O(amount)
→ Infinite supply
If I am taking yes call for a coin, next level, same coin will be present
If I am taking no call, in the next level, next coin will explore the choices

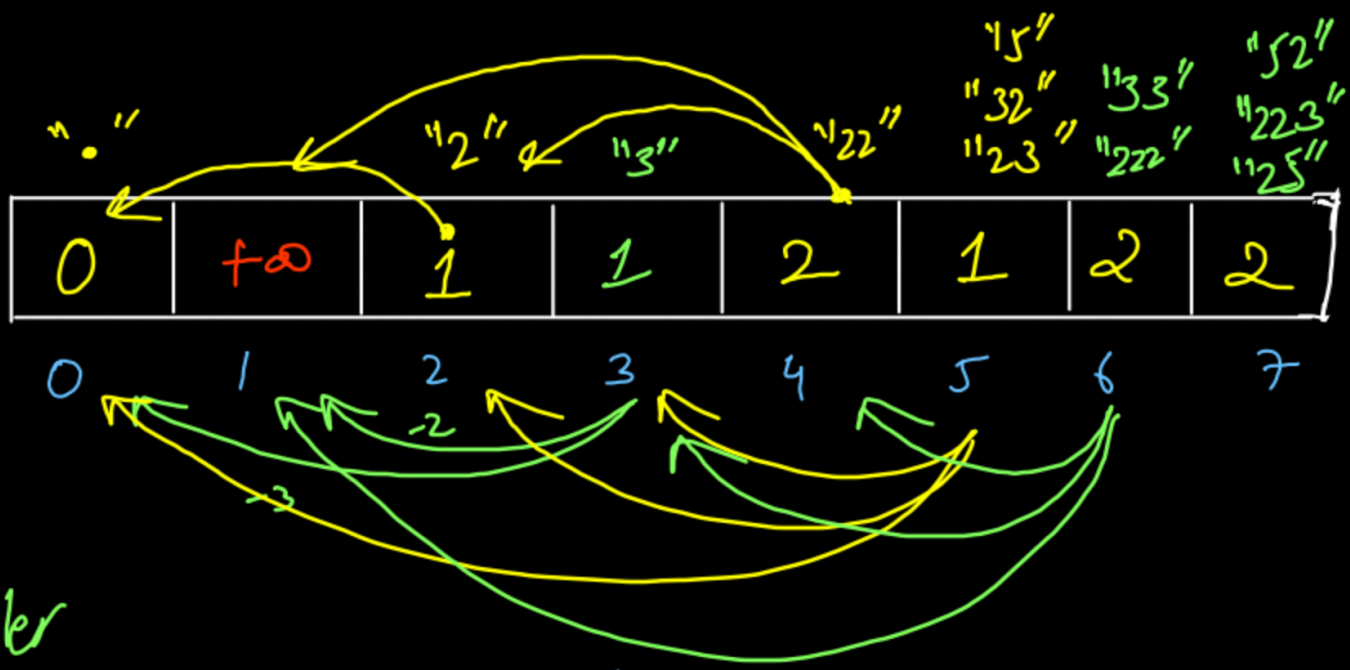
```java
// Time Complexity: O(Amount * Coins), Space Complexity: O(Amount * Coi
public int memo(int amount, int idx, int[] coins, int[][] dp){
    if(amount < 0) return Integer.MAX_VALUE;
    if(amount == 0) return 0;
    if(idx == coins.length) return Integer.MAX_VALUE;
    if(dp[amount][idx] != -1) return dp[amount][idx];

    int yes = memo(amount - coins[idx], idx, coins, dp);
    if(yes != Integer.MAX_VALUE) yes += 1;

    int no = memo(amount, idx + 1, coins, dp);

    return dp[amount][idx] = Math.min(yes, no);
}

public int coinChange(int[] coins, int amount) {
    int[][] dp = new int[amount + 1][coins.length];
    for(int i=0; i<=amount; i++){
        for(int j=0; j<coins.length; j++){
            dp[i][j] = -1;
        }
    }

    int ans = memo(amount, 0, coins, dp);
    if(ans == Integer.MAX_VALUE) return -1;
    return ans;
}
```

0.5 x

Tabulation

{2, 3, 5}

target = 7



1D array (amount) is a must

space opt
↓
Not possible

smaller problem

larger problem

dp[i] = min coins to form target i

```java
// Time -> O(Amount * Coin), Space -> O(Amount)
public int coinChange(int[] coins, int amount) {
    int[] dp = new int[amount + 1];
    Arrays.fill(dp, Integer.MAX_VALUE);
    dp[0] = 0;

    for(int i=1; i<=amount; i++){
        for(int coin: coins){
            if(i - coin >= 0 && dp[i - coin] != Integer.MAX_VALUE)
                dp[i] = Math.min(dp[i], dp[i - coin] + 1);
        }
    }

    return (dp[amount] == Integer.MAX_VALUE) ? -1 : dp[amount];
}
```

Coin change Permutations vs Combinations

coins = {2, 3, 5}

target = 10

Permutations

14

{2,2,2,2,2} {5,5}

{2,2,3,3} {2,3,3,2} {2,3,3,2}

{3,2,2,3} {3,2,3,2} {3,3,2,2}

{2,3,5} {2,5,3} {3,2,5}

{3,5,2} {5,2,3} {5,3,2}

Combinations

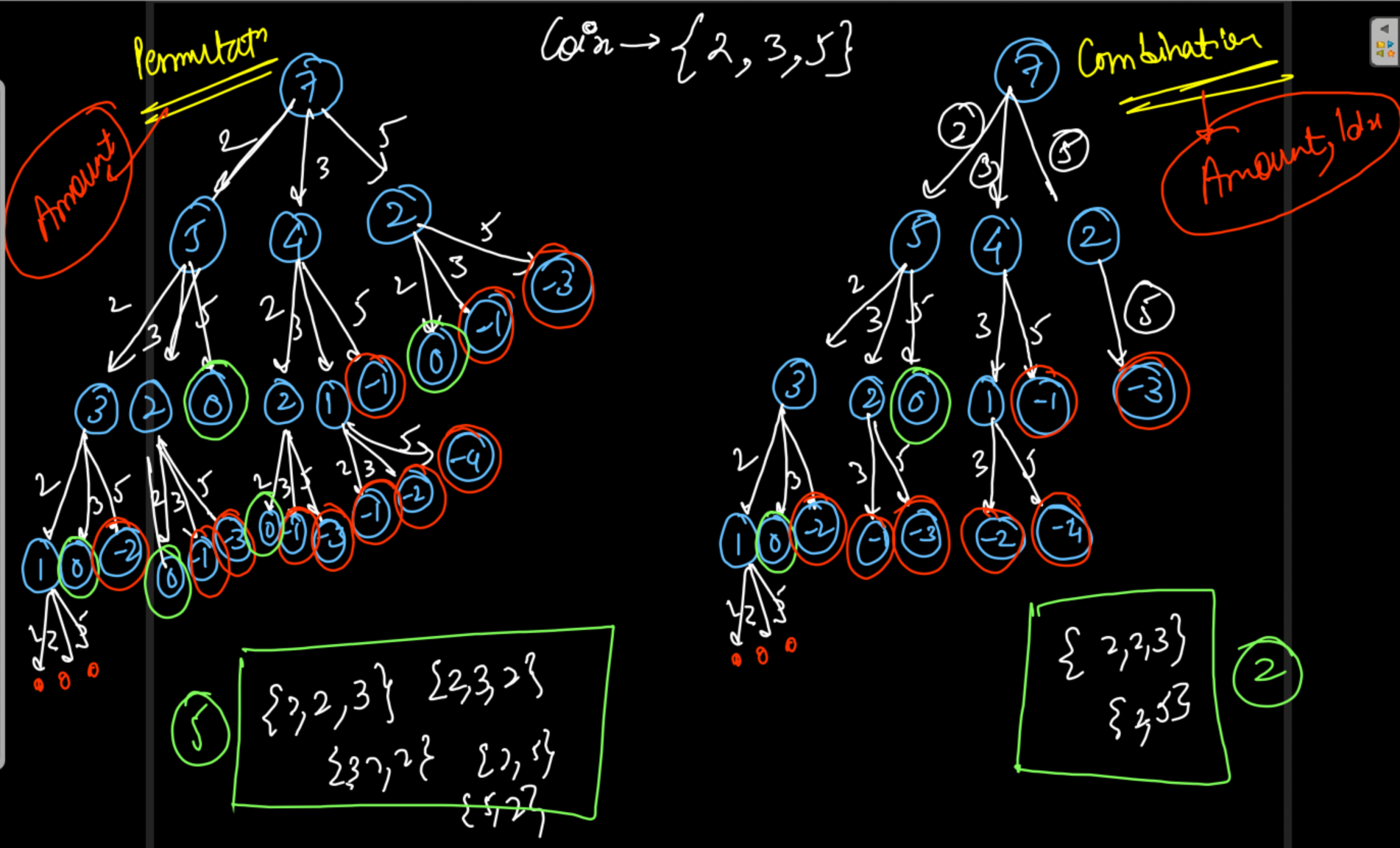
5

{2,2,2,2,2} {2,2,3,3}

{5,5} {2,3,5}

3! = 6

$\frac{4!}{2!2!} = \frac{24}{4} = 6$


Permutation
Coin → {2, 3, 5}
Amount
Combination
Amount, Idx
{2,2,3} {2,3,2}
{3,2,2} {2,5}
{5,2}
{2,2,3}
{3,5}
2