# 518: Coin Change II (Number of ways)

amount = 5 , coins = [1, 2, 5]

answer = 4

Explanation:

1 + 1 + 1 + 1 + 1

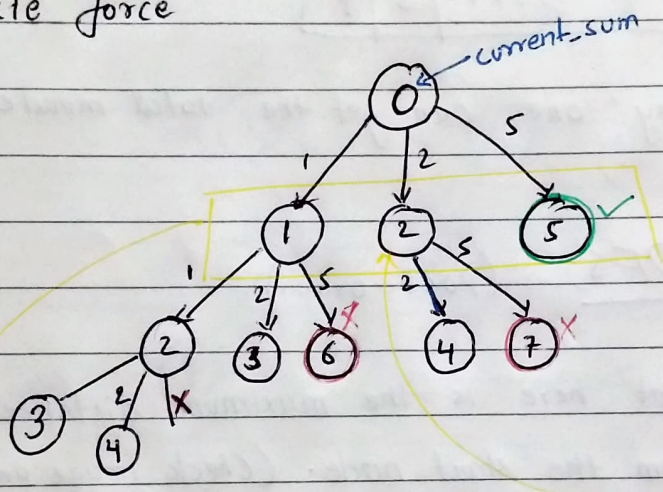
2 + 2 + 1

2 + 1 + 1 + 1

5

## (1) Approach 1 (My approach)

Brute force




→ At every step we add all coin denominations to the current_sum.

→ One point to note here: we only add coins greater than equal the previous denomination

→ we only added [2,5] here as previous coin was 2.

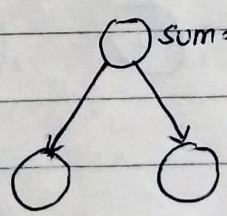
1 + 2 is same as 2 + 1

→ No. of ways would be the distinct paths ~~each~~ ~~he~~ that will lead to sum 5.

This means we have to use "for" loop in our recursion to traverse through the coins list.

Imp:- Doing this not a good, as it makes difficult to convert the solution to tabulation.

## (2) Approach 2 (Brute force but optimized)




sum, pos + 1

don't use the coin & move to next position.

sum - coin[pos], pos

use the coin, and stay in same position. (Since same coin be used again)

- Base case would be when sum == 0

```
if coins[pos] > sum:
    count += func(pos+1, sum)
else:
    count += func(pos+1, sum) + func(pos, sum - coins[pos])
```

## (3) Approach 3 : Converting to tabulation

pos ~~→~~ n → 0

sum → 0 → n

the column sum = 0 will have all values to 1.

coins = [1, 2, 5] amount = 5

| pos | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 3 | 4 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 2 | 1 | 0 | 0 | 0 | 0 | 1 |
| 3 | 1 | 0 | 0 | 0 | 0 | 0 |

sum = 0

```
if coins[pos] > sum:
    dp[pos][sum] = dp[pos+1][sum]
else:
    dp[pos][sum] =
        dp[pos+1][sum] +
        dp[pos][sum - coins[pos]]
```

## (4) Approach 4 : Space optimized

to calculate row "pos" we need only "pos+1"

we don't need other rows.

At iteration pos = 2, we use pos = 3

∴ We can keep one array (instead of 2D) and update in-place in the next iteration.

```
dp[pos][sum] = dp[pos+1][sum]
             + dp[pos][sum - coins[pos]]
```

this is not required anymore

```
dp[sum] += dp[sum] + dp[sum - coins[pos]]
```