

Coin Change II

Count ways (not min coins) — combinations DP ■

● MEDIUM

■ Hook

How many combinations of coins sum to amount? Order doesn't matter: [1,2] and [2,1] = same combination. This is counting DP, closely related to 0/1 knapsack. **Iterate coins in outer loop** to avoid duplicates.

■ Core Intuition — Combinations vs Permutations

■ Outer loop order matters!

Coins in OUTER loop → count combinations (each subset counted once). Amount in OUTER loop → count permutations (each ordering counted separately). Coin Change II wants combinations → coins outer, amount inner.

■ DP Array — coins=[1,2,5], amount=5

```
dp = [1,0,0,0,0,0] (dp[0]=1: one way to make 0)
Process coin=1:
dp[1]+=dp[0]=1 dp[2]+=dp[1]=1 ... dp=[1,1,1,1,1,1]
Process coin=2:
dp[2]+=dp[0]=1→2 dp[3]+=dp[1]=1→2 dp[4]+=dp[2]=2→3 dp[5]+=dp[3]=2→3
dp=[1,1,2,2,3,3]
Process coin=5:
dp[5]+=dp[0]=1→4
dp=[1,1,2,2,3,4]
★ ANSWER = dp[5] = 4 ({1,1,1,1,1}, {1,1,1,2}, {1,2,2}, {5})
```

■ Code

```

public int change(int amount, int[] coins) {
    int[] dp = new int[amount+1];
    dp[0] = 1; // one way to make amount 0
    for (int c : coins) { // COINS in outer loop!
        for (int a=c; a<=amount; a++) {
            dp[a] += dp[a-c];
        }
    }
    return dp[amount];
}

```

■ TIME

$O(\text{amount} \times \text{coins})$

■ SPACE

$O(\text{amount})$

- ✓ $dp[a]$ = number of combinations to make amount a
- ✓ $dp[0]=1$ (empty combination), rest=0
- ✓ COINS in outer loop → combinations (no duplicates)
- ✓ $dp[a] += dp[a-c]$ for each coin $c \leq a$
- ✓ Time: $O(\text{amount} \times |\text{coins}|)$ | Space: $O(\text{amount})$

■ MENTAL MODEL — Remember This Forever

The coins-outer-loop trick: by processing all amounts for $\text{coin}[0]$ first, then all amounts for $\text{coin}[1]$, you ensure each coin's contribution is layered on top — never double-counting. If you flip the loops, $[1,2]$ and $[2,1]$ get counted as different.