Dynamic Programming

Content

1. Coin Change - Combinations

You are given an array of n numbers, which represents n different types of denominations of coins. You are given a target 'amt', you need to count the number of combinations of the coins possible that sum up to the given target amount, i.e. in how many ways you can pay 'amt' using the available denominations.

Note 1: You can consider that you are having an infinite supply of each coin denomination i.e. the same coin denomination can be used for many installments in payment of "amt".

Note 2: You are required to find the count of combinations and not permutations. For eg, 2 + 2 + 3 = 7 and 2 + 3 + 2 = 7 and 3 + 2 + 2 = 7 are different permutations of the same combination. You should treat them as 1 and not 3.

Example, to pay the amount = 7 using coins {2, 3, 5, 6}, there are two combinations of coins possible: {2, 5} and {2, 2, 3}. Hence answer is 2.

So, as I said, this problem is a slight variation of the 'Target Sum Subsets' problem. Can you figure out the similarities and differences between the two questions?

Similarity: We have to find a subset of the array (coins in this case) whose sum is equal to the target (amount in this case).

Difference: In the target sum subset, we can take each element at most one time. But in this problem, we can take each coin any number of times (infinite supply).

Let us define our DP state on a 1d array. We will create an array of sizes (amount + 1) all filled with 0 initially.

Value at index i in the DP array represents the count of ways to have combinations of coins such that their sum of values is i.

Trivial Case: What is the number of ways to select coins so that their sum = 0. Interestingly, there is 1 way to have sum = 0, and that way is to select no coins. Hence, we will initialize dp[0] as 1.

Algorithm

We will run a loop one by one for picking a coin denomination, i.e. the outer loop i will iterate from 0 to n-1.

After picking a coin, we will run a nested loop through the dp array, i.e. the inner loop j will iterate from 1 to amount.

If we pick the coin arr[i], then the remaining sum becomes j - arr[i]. Hence, the number of ways to form sum = j with the last coin as arr[i] is the same as the number of ways to form sum = j - arr[i].

Thus, we will add the number of ways of a combination of coins to form sum = j - arr[i] to the current DP state, i.e. dp[i].

```
dp[j] += dp[j - arr[i]] (if arr[i] <= j)
```

At last, when all coins are considered and the entire DP array is filled, we will return the value of dp[amt] as it contains the number of ways of coin change combinations.

Note:

Before accessing the value of j - arr[i], we will check if arr[i] >= j or not. This is because, if we take the current coin arr[i] which is greater than the sum j required, then the remaining sum will become negative. We do not have coins with negative denominations, and also checking dp[j - arr[i]] when j < arr[i] will give a run-time error (due to out of bound index).

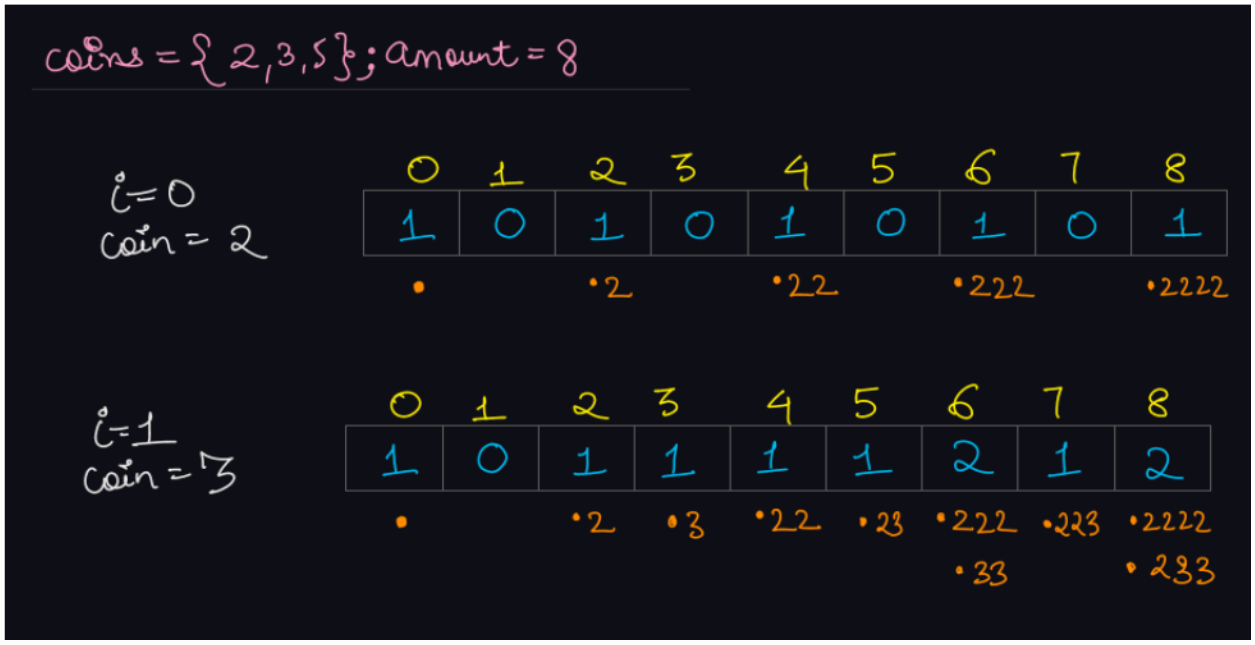
Q. How are we restricting permutations?

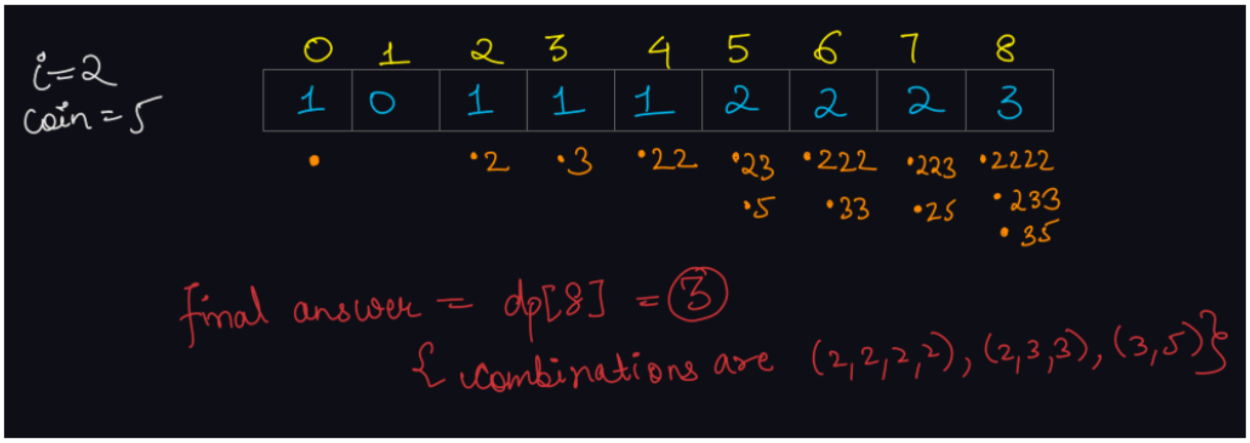
Please look carefully, that we run the outer loop for coin denominations, and then the inner loop through the DP array.

Hence, first, all ways will get added to the dp array (for all amounts = 1 to amt), using the current coin. Then, in the second iteration, all ways will get added to the dp array using the second coin, and so on.

Since the first coin will not come again after the second coin for a given dp state, we are successfully able to prevent permutations of coins.

Let us try to fill the DP table using the algorithm stated above:





Implementation - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Dynamic%20Programming/Codes/Coin_change.cpp

2. Coin Change - Permutations

Problem

You are given an array of n numbers, which represents n different types of denominations of coins. You are given a target 'amt', you need to count the number of permutations of the coins possible that sum up to the given target amount, i.e. in how many ways you can pay 'amt' using the available denominations.

Note 1: You can consider that you are having infinite supply of each coin denomination i.e. the same coin denomination can be used for many installments in payment of "amt".

Note 2: You are required to find the count of distinct permutations. For eg, 2 + 2 + 3 = 7 and 2 + 3 + 2 = 7 and 3 + 2 + 2 = 7 are different permutations, hence their count will be 3. However, we cannot swap coin value = 2 with other coin value = 2 only to form another permutation, as they will not form distinct permutations.

Example, to pay the amount = 7 using coins {2, 3, 5, 6}, there are five coin permutations possible: (2, 5), (5, 2), (2, 2, 3), (2, 3, 2) and (3, 2, 2). Hence the answer is 5.

Note: If you have not tried enough to come up with logic, then we recommend you to first spend an hour or so doing it, else read only the logic used, take it as a hint and try the problem again with the same logic.

Approach

So, as I said, this problem is a slight variation of the 'Target Sum Subsets' problem. Can you figure out the similarities and differences between the two questions?

Similarity: We have to find a subset of the array (coins in this case) whose sum is equal to the target (amount in this case).

Difference: In the target sum subset, we can take each element at most one time. But in this problem, we can take each coin any number of times (infinite supply).

Let us define our dp state on a 1d array. We will create an array of size (amount + 1) all filled with 0 initially.

Value at index i in the dp array represents the count of ways to have permutations of coins such that their sum of values is i.

Trivial Case: What is the number of ways to select coins so that their sum = 0. Interestingly, there is 1 way to have sum = 0, and that way is to select no coins. Hence, we will initialize dp[0] as 1.

Until now, whatever we have done is identical to what we did in coin change combinations.

Q) **But how did we stop coin permutations in the previous solution?**

We ran the loop for the first coin for the entire array at once and then ran the loop for the second coin for the entire array, and so on. Summary is that in the previous solution, outer loop i went from 0 to n and inner loop went from 1 to amount. By doing so, we stopped the occurrence of the first coin after the second coin.

Q) How can we bring back those permutations? Try to modify the code a little bit, so that first all denominations of coins are analyzed for smaller amounts, then for larger amounts.

So, in this case, what we all need to do is first find ways of permuting coins for amount 1 using all coins, then for amount 2, and so on. Hence we need the amount loop to be the outer one and the loop over coins to become the inner one.Voila! Only thing we need to change is interchange the loops. That's All.

Algorithm

1. We will run outer loop i from 1 to amount + 1, i.e. through the dp array.
2. Now for amount i, we will run a nested loop through all coin denominations, i.e. the inner loop j will iterate from 0 to n-1.
3. If we pick the coin arr[j], then the remaining sum becomes i - arr[j]. Hence, the number of ways to form sum = i with the last coin as arr[j] is the same as the number of ways to form sum = i - arr[j].

4. Thus, we will add the number of ways of combination of coins to form sum = i - arr[j] to the current dp state, i.e. dp[i].

   dp[i] += dp[i - arr[j]] (if arr[j] <= i)

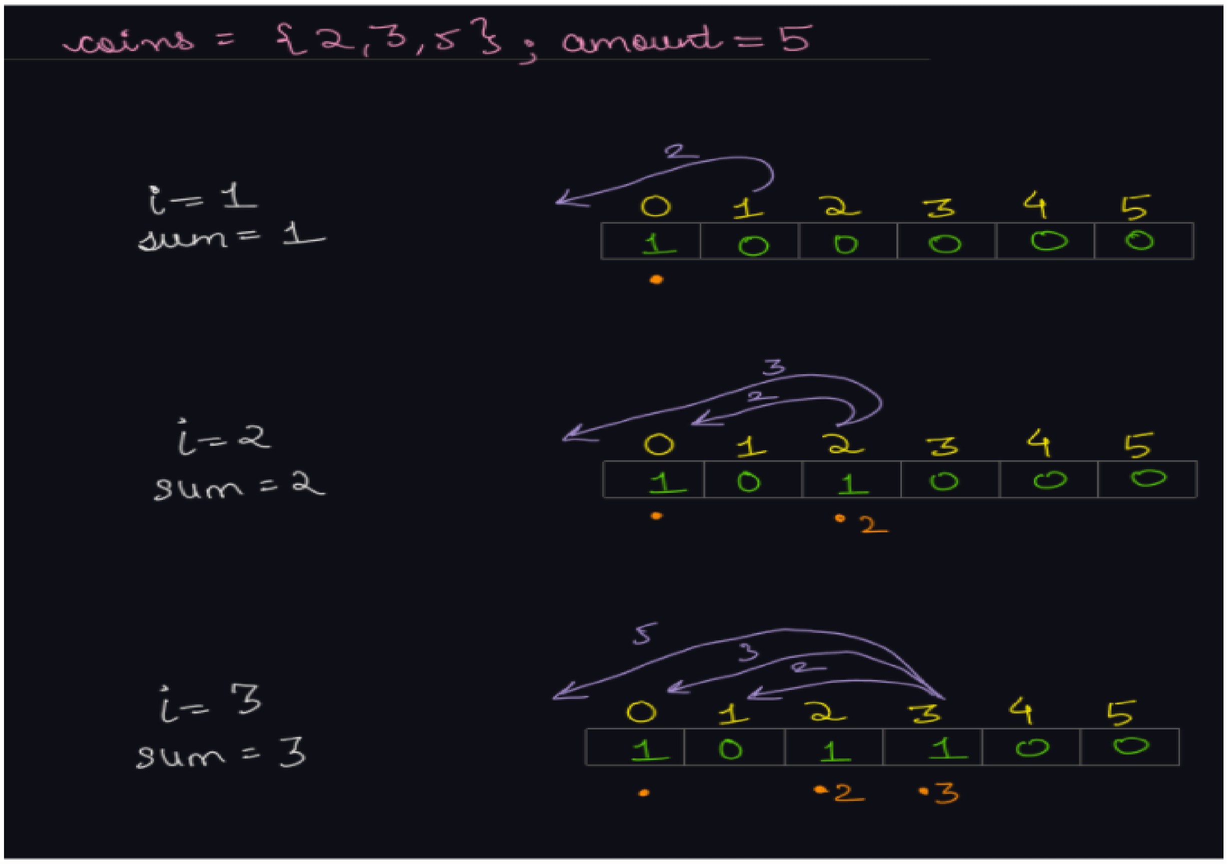
5. At last, when all coins are considered and the entire dp array is filled, we will return the value of dp[amt] as it contains the number of ways of coin change permutations.

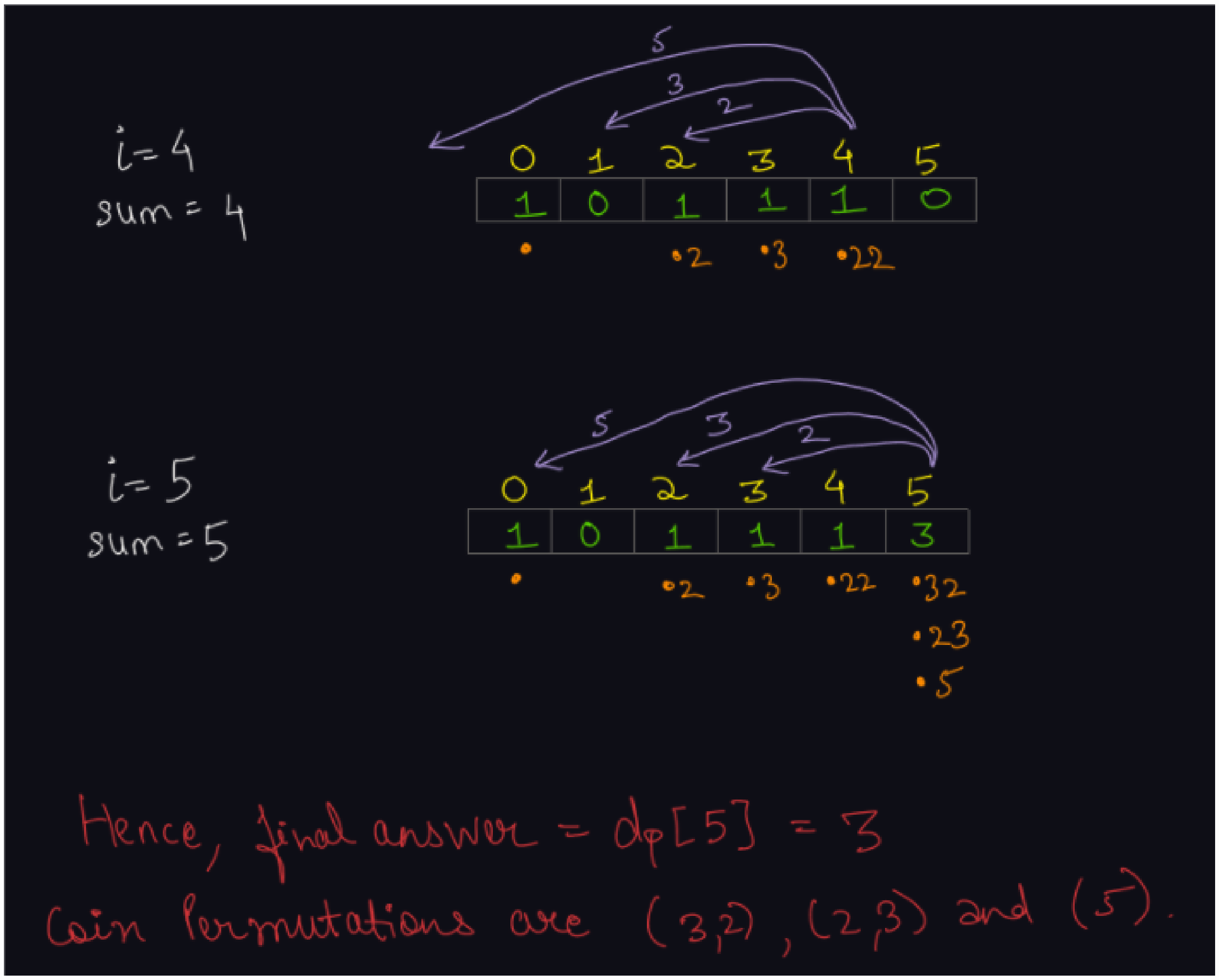
Note:

Before accessing the value of i - arr[j], we will check if arr[j] >= i or not. This is because, if we take the current coin arr[j] which is greater than the sum i required, then the remaining sum will become negative.

We do not have coins with negative denominations, and also checking dp[i - arr[j]] when i < arr[j] will give a run-time error (due to out of bound index).

Let us try to fill the dp table using the algorithm stated above:


coins = {2,3,5} ; amount = 5
i = 1
sum = 1
i = 2
sum = 2
i = 3
sum = 3
0 1 2 3 4 5
1 0 0 0 0 0
1 0 1 0 0 0
1 0 1 1 0 0
2
3
5



Hence, final answer = dp[5] = 3

Coin Permutations are (3,2), (2,3) and (5).

Implementation - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Dynamic%20Programming/Codes/coin_change_problem.cpp