

## Homework -9

Aswin Sampath  
[saswin@uci.edu](mailto:saswin@uci.edu)

53844684

### Question 1)

#### 322. Coin Change

Medium Topics Companies

You are given an integer array `coins` representing coins of different denominations and an integer `amount` representing a total amount of money.

Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return `-1`.

You may assume that you have an infinite number of each kind of coin.

##### Example 1:

**Input:** `coins = [1,2,5]`, `amount = 11`  
**Output:** 3  
**Explanation:**  $11 = 5 + 5 + 1$

##### Example 2:

**Input:** `coins = [2]`, `amount = 3`  
**Output:** -1

##### Example 3:

**Input:** `coins = [1]`, `amount = 0`  
**Output:** 0

##### Constraints:

- $1 \leq \text{coins.length} \leq 12$
- $1 \leq \text{coins}[i] \leq 2^{31} - 1$
- $0 \leq \text{amount} \leq 10^4$

### Understanding the problem:

Given a set of coins with different denominations and a total amount, the task is to find the fewest number of coins needed to make up that amount.

If it's impossible to achieve the amount with the given coins, return -1.

## **Identified Edge Cases:**

Empty coin set: When there are no coins provided, it's impossible to make up any amount, returning -1 would be appropriate.

Amount = 0: The output should be 0 because no coins are needed to achieve an amount of 0.

Test Cases:

A scenario with multiple denominations and a specific amount that requires a combination of coins.

A case where the given amount cannot be achieved using any combination of coins.

A case with only one coin denomination that needs to make up a non-zero amount.

## **Algorithmic Solution:**

The code uses dynamic programming to find the minimum number of coins needed to make up each amount from 0 to the target amount.

It initializes a DP array `dp` to store the fewest number of coins required for each amount.

It iterates through each coin denomination and checks for each amount whether using the current coin minimizes the number of coins needed to make that amount.

Finally, it returns either the fewest number of coins required for the given amount or -1 if the amount cannot be achieved.

## **Time and Space Complexity:**

Time Complexity:  $O(\text{amount} * n)$ , where 'amount' is the target amount and 'n' is the number of coins. It iterates through the DP array for each coin denomination.

Space Complexity:  $O(\text{amount})$ , for storing the DP array.

## Code:

```
1 class Solution {
2 public:
3     int coinChange(vector<int>& coins, int amount) {
4         // Given an amount of money , what is the min nm of coins that makes up that
5         vector<long>dp(amount+1,pow(2,31)-1);
6         dp[0]=0;
7         sort(coins.begin(),coins.end());
8         for(int c:coins){
9             for(int i=c;i<=amount;i++){
10                 dp[i] = min(dp[i-c]+1,dp[i]);
11             }
12         }
13         return dp[amount]==pow(2,31)-1 ? -1 : dp[amount];
14     }
15 };
```

## Output:

Aswin submitted at Dec 06, 2023 09:53



Accepted

Editorial

+ Solution

Runtime

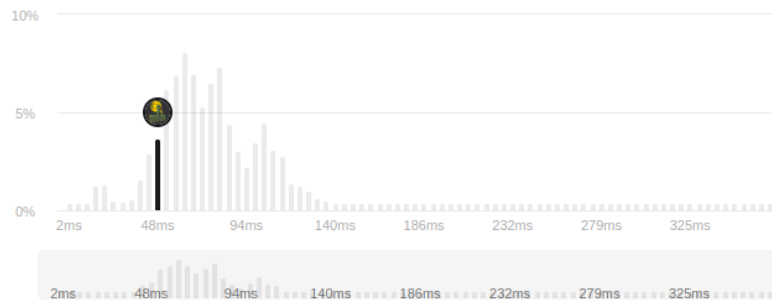
52 ms

Beats 87.12% of users with C++

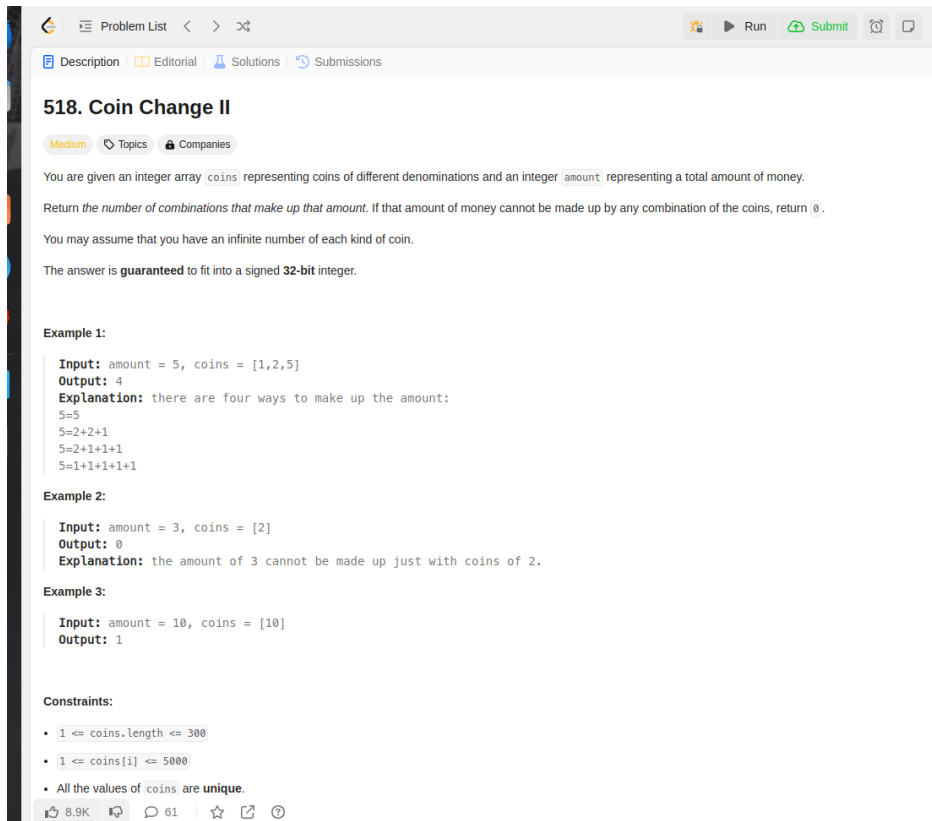
Memory

18.68 MB

Beats 40.66% of users with C++



## Question 2)



The screenshot shows the LeetCode interface for the problem '518. Coin Change II'. At the top, there's a navigation bar with 'Problem List', 'Run', 'Submit', and other icons. Below the navigation bar, the problem title '518. Coin Change II' is displayed, followed by a 'Medium' difficulty tag and 'Topics' and 'Companies' links. The problem description states: 'You are given an integer array `coins` representing coins of different denominations and an integer `amount` representing a total amount of money. Return the number of combinations that make up that amount. If that amount of money cannot be made up by any combination of the coins, return 0. You may assume that you have an infinite number of each kind of coin. The answer is guaranteed to fit into a signed 32-bit integer.'

Below the description, there are three examples:

**Example 1:**  
`Input:` amount = 5, coins = [1,2,5]  
`Output:` 4  
`Explanation:` there are four ways to make up the amount:  
5=5  
5=2+2+1  
5=2+1+1+1  
5=1+1+1+1+1

**Example 2:**  
`Input:` amount = 3, coins = [2]  
`Output:` 0  
`Explanation:` the amount of 3 cannot be made up just with coins of 2.

**Example 3:**  
`Input:` amount = 10, coins = [10]  
`Output:` 1

**Constraints:**

- `1 <= coins.length <= 300`
- `1 <= coins[i] <= 5000`
- All the values of `coins` are unique.

At the bottom, there are statistics: 8.9K votes, 61 comments, and a star icon.

## Problem Understanding:

Given a total amount and a set of coins with distinct denominations, the task is to find the number of combinations that can form the given amount using the provided coins. If it's impossible to achieve the amount using any combination of coins, return 0.

## Identified Edge Cases:

Empty coin set: When there are no coins provided, it's impossible to make up any amount, and the output should be 0.

Amount = 0: If the given amount is 0, there's only one way to make up the amount, which is using no coins.

Test Cases:

Multiple denominations and a specific amount that requires various coin combinations to form.

A scenario where the given amount cannot be achieved using any combination of coins.

A case with only one coin denomination, validating whether it can make up the target amount.

## **Algorithmic Solution:**

The code employs dynamic programming (DP) to calculate the number of combinations to form each amount from 0 to the target amount.

Initializes a DP array `dp` to count the number of combinations for each amount.

It iterates through each coin denomination and updates the DP array for each amount that can be formed using the current coin denomination.

The final value in `dp[amount]` represents the total number of combinations that can form the target amount.

## **Time and Space Complexity:**

Time Complexity:  $O(\text{amount} * n)$ , where 'amount' is the target amount and 'n' is the number of coins. It iterates through the DP array for each coin denomination.

Space Complexity:  $O(\text{amount})$ , for storing the DP array.

## **Code:**

```
1  class Solution {
2  public:
3      int change(int amount, vector<int>& coins) {
4          vector<int> dp(amount+1, 0);
5          dp[0] = 1;
6          for(int c: coins){
7              for(int i=c; i<=amount; i++){
8                  dp[i] += dp[i-c];
9              }
10         }
11         return dp[amount];
12     }
13 };
```

# Output:

Accepted

Editorial

Solution

Runtime

4 ms

Beats 93.85% of users with C++

Memory

7.52 MB

Beats 83.14% of users with C++

