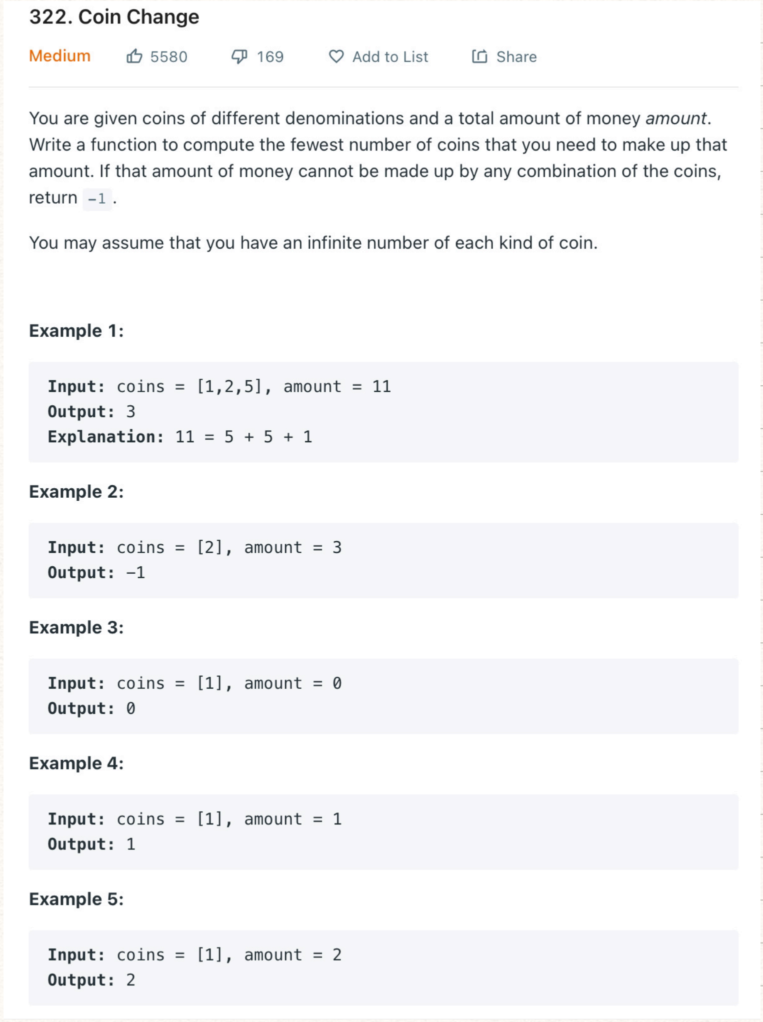


322. Coin Change

Medium 5580 169 Add to List Share

You are given coins of different denominations and a total amount of money *amount*. Write a function to compute 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
```

Example 4:

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

Example 5:

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

Def: dp[i] = min coins to make up i amount

Init: dp[0] = 0   dp[i] = -1 , i < 0

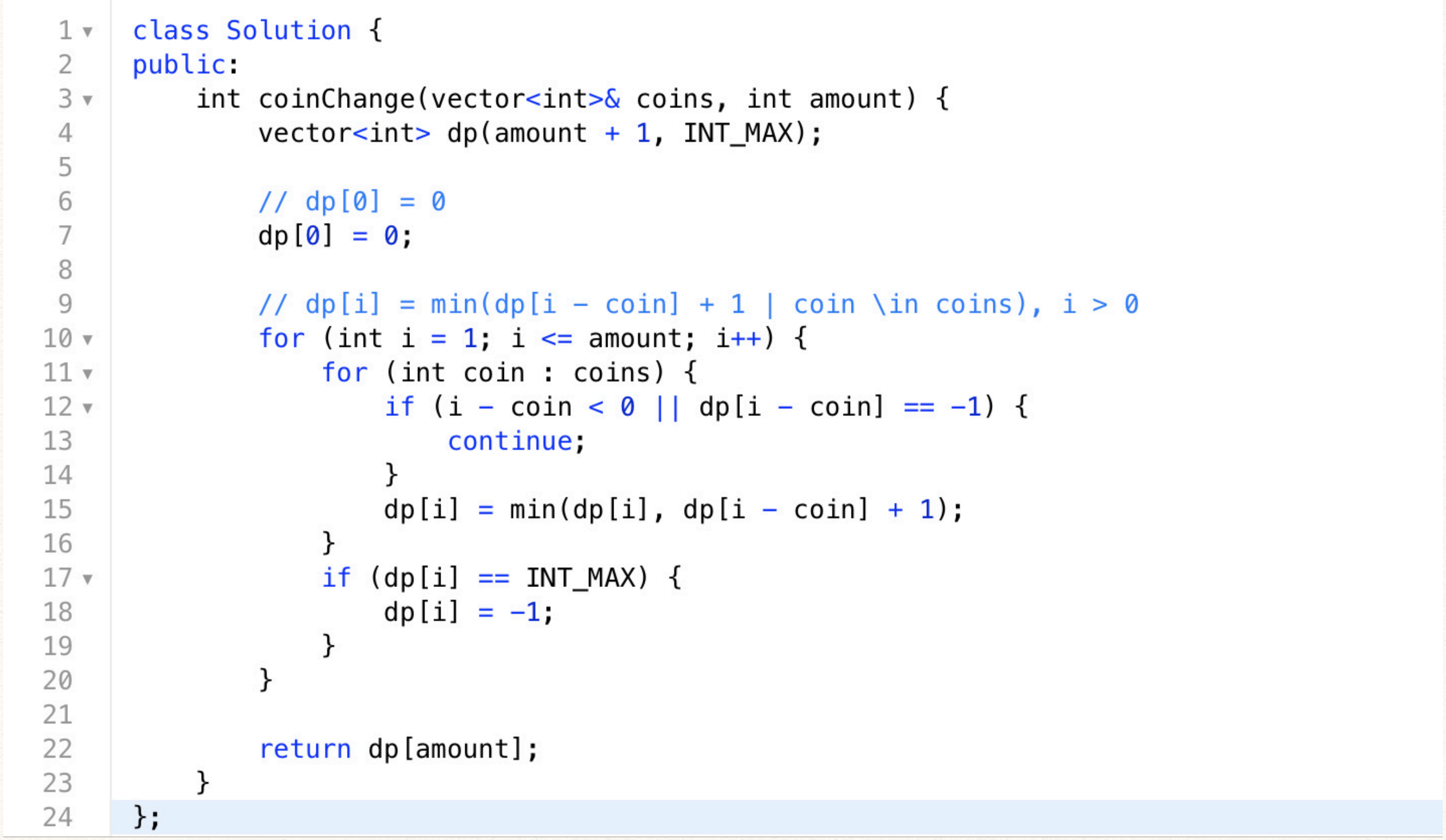
Transition:

$$dp[i] = \min_{coin \in coins}(dp[i - coin] + 1), \quad i > 0$$

Answer: dp[n]

Time Complexity: $O(n \times \#coins)$

Space Complexity: $O(n \times \#coins)$



```cpp
class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        vector<int> dp(amount + 1, INT_MAX);

        // dp[0] = 0
        dp[0] = 0;

        // dp[i] = min(dp[i - coin] + 1 | coin \in coins), i > 0
        for (int i = 1; i <= amount; i++) {
            for (int coin : coins) {
                if (i - coin < 0 || dp[i - coin] == -1) {
                    continue;
                }
                dp[i] = min(dp[i], dp[i - coin] + 1);
            }
            if (dp[i] == INT_MAX) {
                dp[i] = -1;
            }
        }

        return dp[amount];
    }
};
```