518. 零钱兑换 II

入选理由

1. 昨天题目的系列题目。一起做更好理解背包的变种

题目地址(518. 零钱兑换 II)

https://leetcode-cn.com/problems/coin-change-2/

题目描述

给定不同面额的硬币和一个总金额。写出函数来计算可以凑成总金额的硬币组合数。假设每一种面额的硬币有无限个。

示例 1:

```
输入: amount = 5, coins = [1, 2, 5]
输出: 4
解释: 有四种方式可以凑成总金额:
5=5
5=2+2+1
5=2+1+1+1
5=1+1+1+1+1
示例 2:

输入: amount = 3, coins = [2]
输出: 0
解释: 只用面额2的硬币不能凑成总金额3。
示例 3:

输入: amount = 10, coins = [10]
输出: 1


注意:

你可以假设:

0 <= amount (总金额) <= 5000
1 <= coin (硬币面额) <= 5000
硬币种类不超过 500 种
结果符合 32 位符号整数
```

标签

- 动态规划

难度

- 中等

思路

定义状态 `dp[i][j]` 为使用前 i 个硬币组成金额 j 的组合数。

则有状态转移方程为：

dp[i][j] = dp[i-1][j] + dp[i]j - coins[i]]

其中 dp[i-1][j] 为不选择 coins[i] 的组合数， dp[i]j - coins[i]] 为选择 coins[i] 的组合数。

由于 dp[i][j] 仅仅依赖 dp[i-1][...] 因此使用滚动数组可以进行空间优化。优化后的转移方程为：

dp[i] = dp[i] + dp[i - coins[j]]

代码

代码支持：Python，JS，CPP

Python Code:

```python
class Solution:
    def change(self, amount: int, coins: List[int]) -> int:
        dp = [0] * (amount + 1)
        dp[0] = 1

        for j in range(len(coins)):
            for i in range(1, amount + 1):
                if i >= coins[j]:
                    dp[i] += dp[i - coins[j]]

        return dp[-1]
```

JS Code：

```javascript
var change = function (amount, coins) {
  const dp = Array.from({ length: amount + 1 }).fill(0);
  dp[0] = 1;
  for (let coin of coins) {
```

```
    for (let i = coin; i <= amount; i++) {
      dp[i] += dp[i - coin];
    }
  }
  return dp[amount];
};
```

CPP Code：

```
class Solution {
public:
    int change(int amount, vector<int>& coins) {
        vector<int> dp(amount + 1); // dp[i]: 凑成金额 i 的组合方式的总数量
        dp[0] = 1; // 可以使用一个tets case测试出来
        for (int& coin : coins)
        {
            for (int i = coin; i <= amount; i++)
            {
                dp[i] = dp[i - coin] + dp[i]; /* 当前面值的硬币，如果选它接下来处理总金额 i-coin，如果不选它继续处理总金额i */
            }
        }
        return dp[amount];
    }
};
```

复杂度分析

令n是coins的数量, m是amount

- 时间复杂度: $O(m * n)$
- 空间复杂度: $O(m)$

上一页 下一页