322. 零钱兑换

题目地址(322. 零钱兑换)

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

标签

- 动态规划

难度

- 中等

题目描述

给定不同面额的硬币 coins 和一个总金额 amount。编写一个函数来计算可以凑成总金额所需的最少的硬币个数。如果没有任何一种硬币组合能组成总金额，返回 -1。

你可以认为每种硬币的数量是无限的。

```
示例 1:

输入: coins = [1, 2, 5], amount = 11
输出: 3
解释: 11 = 5 + 5 + 1
示例 2:

输入: coins = [2], amount = 3
输出: -1
示例 3:

输入: coins = [1], amount = 0
输出: 0
示例 4:

输入: coins = [1], amount = 1
输出: 1
示例 5:

输入: coins = [1], amount = 2
输出: 2
```

提示：

```
1 <= coins.length <= 12
1 <= coins[i] <= 231 - 1
0 <= amount <= 104
```

思路

零钱系列是很经典的背包问题的变形，读题可以发现，每种硬币是没有数量限制的，硬币就是物品，amount 就是背包的大小，因此该题抽象出来就是个**完全背包问题**，只不过专题讲义用的是获得的最大价值，该题是求最小价值，所谓**背包中的价值就是装硬币的个数**。

需要注意的是由于专题问题定义为最大价值，因此 dp 初始化为 0。而该题需求最小价值，因此 dp 初始化为 max_value 且 dp[0] = 0

按照上述思路分析+专题给出的模板，可以很轻松地写出如下动态规划代码。

代码

代码支持： Java，Python3，CPP

Java Code：

```java
public int coinChange(int[] coins, int amount) {

    if (coins == null || coins.length == 0 || amount <= 0)
        return 0;

    int[] dp = new int[amount + 1];


    Arrays.fill(dp, amount + 1);
    dp[0] = 0;

    for (int coin : coins) {

        for (int i = coin; i <= amount; i++) {

            dp[i] = Math.min(dp[i], 1 + dp[i - coin]);
        }
    }

    return dp[amount] == amount + 1 ? -1 : dp[amount];
}
```

Python3 Code：

```python
class Solution(object):
    def coinChange(self, coins, amount):
        dp = [amount + 1] * (amount+1)
        dp[0] = 0
        for i in range(1, amount+1):
            for coin in coins:
                if i >= coin:
                    dp[i] = min(dp[i], dp[i-coin]+1)
        return -1 if dp[amount] == amount + 1 else dp[amount]
```

CPP Code：

```cpp
class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        vector<int> dp(amount + 1, amount+1);
        dp[0] = 0;
        for (int i = 1; i <= amount; i++) /* 填充 dp[1] ~ dp[amount] */
        {
            for (auto& coin : coins)
            {
                if (i < coin) continue; // 当前面值大于要凑的总额
                dp[i] = min(dp[i], dp[i - coin] + 1);
            }
        }
        return dp[amount] == amount+1 ? -1 : dp[amount];
    }
};
```

复杂度分析

令 N 为物品个数即硬币种类， amount 为总金额也即背包大小。

- 时间复杂度：$O(N * \text{amount})$
- 空间复杂度：$O(\text{amount})$