# 题目

给你一个整数数组coins，表示不同面额的硬币；以及一个整数amount，表示总金额。

计算并返回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

# 分析

## 方法一：记忆化搜索

## 方法二：动态规划

class Solution {

public:

int coinChange(vector<int>& coins, int amount) {

int Max = amount + 1;

vector<int> dp(amount + 1, Max);

dp[0] = 0;

for (int i = 1; i <= amount; ++i) {

for (int j = 0; j < (int)coins.size(); ++j) {

if (coins[j] <= i) {

dp[i] = min(dp[i], dp[i - coins[j]] + 1);

}

}

}

return dp[amount] > amount ? -1 : dp[amount];

}

};

**复杂度分析：**

时间复杂度：O(Sn)，其中S是金额，n是面额数。我们一共需要计算O(S)个状态，S为题目所给的总金额。对于每个状态，每次需要枚举n个面额来转移状态，所以一共需要O(Sn)的时间复杂度。

空间复杂度：O(S)。数组dp需要开长度为总金额S的空间。