一、 01硬币找零问题（01背包）

给定不同面额的硬币 `coins` 和总金额 `m`。每个硬币最多选择一次。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。

状态表示

- `f[i][j]` 表示只看前 `i` 个物品，总价值是 `j` 的情况下的最小硬币数目。 **状态转移**
- `f[i, j] = min(f[i-1, j], f[i-1, j-ci] + 1)` 分别对应了不拿和拿第 `i` 个硬币两种情况。
- 因为 `f[i, j]` 只和上一层的两个状态有关，所以可以将状态优化为一维数组。

  `f[j] = min(f[j], f[j-ci] + 1)`，

  因为 `j-ci < j`，所以如果从小到大枚举金额的话，`j-ci` 已经变成了第 `i` 层的状态。

  所以这一步可以从大到小枚举金额，确保计算到 `j` 时，`j-ci` 还是上一层的状态。

边界情况

- `f[0] = 0` 表示凑出金额为0的最小个数是0个。
- 初始化，因为题目要求的是恰好凑到金额m，**所以状态要初始化为** `inf` 因为有些状态可能达不到.

```
def func_2(coins, m):
    f = [float('inf')] * (m + 1)
    f[0] = 0
    for c in coins:  # 枚举硬币总数
        for j in range(m, c-1, -1):  # 从大到小枚举金额，确保j-c >= 0.
            f[j] = min(f[j], f[j - c] + 1)
    return f[m] if f[m] != float('inf') else -1  # 如果为inf说明状态不可达，返回-1即
可。
```

二、完全硬币找零问题（完全背包）

给定不同面额的硬币 `coins` 和总金额 `m`。每个硬币可以选择无数次。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。

状态表示

- `f[i][j]` 为考虑前 `i` 种硬币，凑出金额为 `j` 的最少数目。

状态转移

- 考第 `i` 种硬币，我们可以不拿，或者拿 `1...k` 个，直到把金额拿爆。
- `f[i][j] = min(f[i-1]f[j], f[i-1][j-c]+1, f[i-1][j-2*c]+2, ..., f[i-1][j-k*c]+k)`
- 又因为其中包含了大量的冗余计算

  例如：`f[i][j-c] = min(f[i-1][j-c], f[i-1][j-2*c]+1, ..., f[i-1][j-k*c]+k-1)`
- 两者合并得到：`f[i][j] = min(f[i-1]f[j], f[i][j-c]+1)`

  又因为 `f[i][j]` 只和上一层一个状态 (`f[i-1]f[j]`) 和这一层的一个状态 `(f[i][j-c]+1)` 有关。可以将状态优化为一维数组。

- `f[j] = min(f[j], f[j-c]+1)`

  因为金额从小到大枚举，`j-c < j`，所以计算 `j` 时， `j-c` 的状态已经在这一层计算好了，可以直接替换。这里与01背包问题相反。

边界情况

- `f[0] = 0` 表示金额为 0 时，最小硬币凑法为 0;
- 其余要初始化为 `inf`，因为此题要求的是恰好金额为m时的最小硬币数，所以有些状态可能达不到。

```
  class Solution:
      def coinChange(self, coins, m):
          f = [float('inf')]*(m+1)
          f[0] = 0
          for c in coins:  # 枚举硬币种数
              for j in range(c, m+1):  # 从小到大枚举金额，确保j-c >= 0.
                      f[j] = min(f[j], f[j - c] + 1)
          return f[m] if f[m] != float('inf') else -1  # 如果为inf说明状态不可达，返回-1即可。
```

可以看到，此题解法与**01问题解法几乎完全相同！！！**只是枚举金额时变为由小到大了。

三、多重硬币找零问题（多重背包）

给定不同面额的硬币 `coins` 和总金额 `m`。每个硬币选择的次数有限制为 `s`。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。

状态表示

- 这里和完全硬币问题的的初始状态表态表示很相似。
- 考第i种硬币，我们可以不拿，或者拿1...k个，直到拿到个数的限制。
- `f[i][j] = min(f[i-1]f[j], f[i-1][j-c]+1, f[i-1][j-2*c]+2, ..., f[i-1][j-k*c]+k)`

所以在**01问题的代码的基础上添加一层枚举硬币个数的循环即可**

这里可以使用二进制优化，转化为01背包问题求解。这里不扩展了。

```
def func_3(coins, m, s):
    f = [float('inf')] * (m + 1)
    f[0] = 0
    for i in range(len(coins)):
        for j in range(m, coins[i]-1, -1):
            for k in range(1, s[i]+1):  # 枚举每个硬币的个数 [1, s[i]]
                if j >= k*coins[i]:  # 确保不超过金额 j
                    f[j] = min(f[j], f[j - k*coins[i]] + k)
    print(f)
    return -1 if f[m] > m else f[m]
```