题目：最少硬币找零问题

给定一组硬币数，找出一组最少的硬币数，来找换零钱N。

比如，可用来找零的硬币为: 1、3、4   待找的钱数为 6。用两个面值为3的硬币找零，最少硬币数为2。而不是 4，1，1

这题很典型是动态规划问题，为什么呢？

首先我们可以看一下什么问题可以被看做动态规划问题。

How should I explain dynamic programming to a 4-year-old?

底下有个42K赞同的答案，是这样说的：

\*writes down "1+1+1+1+1+1+1+1 =" on a sheet of paper\*

"What's that equal to?"

\*counting\* "Eight!"

\*writes down another "1+" on the left\*

"What about that?"

\*quickly\* "Nine!"

"How do you know it was nine so fast?"

"You just added one more"

"So you didn't need to recount because you remembered there were eight!Dynamic Programming is just a fancy way to say 'remembering stuff to save time later'"

因此动态规划的精髓在于，充分利用上一次的计算结果来达成本次快速计算的目的

OK， 一般我们解决动态规划问题可分为三步走；

1. 表示状态
2. 找出状态转移方程
3. 边界处理
4. 表示状态

分析问题的状态时，不要分析整体，只分析最后一个阶段即可！因为动态规划问题都是划分为多个阶段的，各个阶段的状态表示都是一样，而我们的最终答案在就是在最后一个阶段。

对于这道题，最后一个阶段是什么呢？

我在思考的这道题的时候就觉得本题最重要在于 1，1，1，1 怎么过渡到 5，它是怎么从

Coin[1, 5, 10, 25] 逐个阶层进行跳跃的，

经过我的反复看书本的解题思路发现，首先无论你要找的数额是多少，先把它和Coin[0] = 1

相减这是个不断的递归的过程，然后就进入到了递归的最深层，我们开始返回结果，返回的结果会正式进入我们对Coin[1, 5, 10, 25]的循环，返回的结果会直接从Coin[1]开始做一个减法，减法的结果进行判断，小于零会进入Coin[2]继续，大于等于零我们就装入我们的缓存数组里面，就以数5为例，5 - 5 == 0，第一次引用缓存中的 [ ], 后面10 - 5也会引用我们的[ 5 ], min = [coin].concat(newMin); 从此开始每一次对缓存数组中的值的引用得到的都是之前最优解和当前最优解的合并，通过缓存数组我们就能直接获取值不用再像判断出第一个5时进行重复的计算，而这也是明显的动态规划的体现。

if(cache[value]) {

return cache[value]

}

据此我们数组就将会保存到 n的最佳解