在LeetCode上有两道题目非常类似，分别是

- 70.爬楼梯
- 518. 零钱兑换 II

如果我们把每次可走步数/零钱面额限制为[1,2], 把楼梯高度/总金额限制为3. 那么这两道题目就可以抽象成"给定[1,2], 求组合成3的组合数和排列数"。

接下来引出本文的核心两段代码，虽然是Cpp写的，但是都是最基本的语法，对于可能看不懂的地方，我加了注释。

```
class Solution1 {
public:
    int change(int amount, vector<int>& coins) {
        int dp[amount+1];
        memset(dp, 0, sizeof(dp)); //初始化数组为0
        dp[0] = 1;
        for (int j = 1; j <= amount; j++){ //枚举金额
            for (int coin : coins){ //枚举硬币
                if (j < coin) continue; // coin不能大于amount
                dp[j] += dp[j-coin];
            }
        }
        return dp[amount];
    }
};
class Solution2 {
public:
    int change(int amount, vector<int>& coins) {
        int dp[amount+1];
        memset(dp, 0, sizeof(dp)); //初始化数组为0
        dp[0] = 1;
        for (int coin : coins){ //枚举硬币
            for (int j = 1; j <= amount; j++){ //枚举金额
                if (j < coin) continue; // coin不能大于amount
                dp[j] += dp[j-coin];
            }
        }
        return dp[amount];
    }
};
```

如果不仔细看，你会觉得这两个Solution似乎是一模一样的代码，但细心一点你会发现他们在嵌套循环上存在了差异。这个差异使得一个求解结果是**排列数**，一个求解结果是**组合数**。

因此在不看后面的分析之前，你能分辨出哪个Solution是得到排列，哪个Solution是得到组合吗?

在揭晓答案之前，让我们先分别用DP的方法解决爬楼梯和零钱兑换II的问题。每个解题步骤都按照DP三部曲，a.定义子问题，b. 定义状态数组，c. 定义状态转移方程。

70. 爬楼梯

问题描述如下:

> 假设你正在爬楼梯。需要 *n* 阶你才能到达楼顶。
>
> 每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢?

这道题目子问题是，problem(i) = sub(i-1) + sub(i-2), 即求解第i阶楼梯等于求解第i-1阶楼梯和第i-2阶楼梯之和。

状态数组是 DP[i], 状态转移方程是DP[i] = DP[i-1] = DP[i-2]

那么代码也就可以写出来了。

```
class Solution {
public:
    int climbStairs(int n) {
        int DP[n+1];
        memset(DP, 0, sizeof(DP));
        DP[0] = 1;
        DP[1] = 1;
        for (int i = 2; i <= n; i++){
            DP[i] = DP[i-1] + DP[i-2] ;
        }
        return DP[n];

    }
};
```

由于每次我们只关注DP[i-1]和DP[i-2]，所以代码中能把数组替换成2个变量，降低空间复杂度，可以认为是**将一维数组降维成点。**

如果我们把问题泛化，不再是固定的1，2，而是任意给定台阶数，例如1,2,5呢?

我们只需要修改我们的DP方程`DP[i] = DP[i-1] + DP[i-2] + DP[i-5]`,也就是`DP[i] = DP[i] + DP[i-j] ,j=1,2,5`

在原来的基础上，我们的代码可以做这样子修改

```
class Solution {
public:
    int climbStairs(int n) {
        int DP[n+1];
        memset(DP, 0, sizeof(DP));
        DP[0] = 1;
        int steps[2] = {1,2};
        for (int i = 1; i <= n; i++){
            for (int j = 0; j < 2; j++){
                int step = steps[j];
                if ( i < step ) continue;// 台阶少于跨越的步数
                DP[i] = DP[i] + DP[i-step];
            }
        }
        return DP[n];
```

```
    }
};
```

后续修改`steps`数组，就实现了原来问题的泛化。

那么这个代码是不是看起来很眼熟呢？我们能不能交换内外的循环呢？也就是下面的代码

```
for (int j = 0; j < 2; j++){
    int step = steps[j];
    for (int i = 1; i <= n; i++){
        if ( i < step ) continue;// 台阶少于跨越的步数
         DP[i] = DP[i] + DP[i-step];
    }
}
```

大家可以尝试思考下这个问题，嵌套循环是否能够调换，调换之后的DP方程的含义有没有改变？

零钱兑换II

问题描述如下:

> 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。

定义子问题: problem(i) = sum(problem(i-j)), j =1,2,5. 含义为凑成总金额i的硬币组合数等于凑成总金额硬币i-1, i-2, i-5,...的子问题之和。

我们发现这个子问题定义居然和我们之前泛化的爬楼梯问题居然是一样的，那后面的状态数组和状态转移方程也是一样的，所以当前问题的代码可以在之前的泛化爬楼梯问题中进行修改而得。

```
class Solution {
public:
    int change(int amount, vector<int>& coins) {
        int dp[amount+1];
        memset(dp, 0, sizeof(dp)); //初始化数组为0
        dp[0] = 1;
        for (int j = 1; j <= amount; j++){ //枚举金额
            for (int i = 0; i < coins.size(): i++){
                int coin = coins[i]; //枚举硬币
                if (j < coin) continue; // coin不能大于amount
                dp[j] += dp[j-coin];
            }
        }
        return dp[amount];
    }
};
```

这就是我们之前的Solution1代码。

但是当你运行之后，却发现这个代码并不正确，得到的结果比预期的大。究其原因，该代码计算的结果是**排列数**，而不是**组合数**，也就是代码会把1,2和2,1当做两种情况。**但**更加根本的原因是我们子问题定义出现了错误。

正确的**子问题**定义应该是，problem(k,i) = problem(k-1, i) + problem(k, i-k)

即前**k**个硬币凑齐金额**i**的组合数**等于**前**k-1**个硬币凑齐金额**i**的组合数**加上**在原来**i-k**的基础上使用硬币的组合数。说的更加直白一点，那就是用前k的硬币凑齐金额i，要分为两种情况开率，一种是没有用前k-1个硬币就凑齐了，一种是前面已经凑到了i-k，现在就差第k个硬币了。

状态数组就是DP[k][i], 即前k个硬币凑齐金额i的组合数。

这里不再是一维数组，而是**二维**数组。第一个维度用于记录当前组合有没有用到硬币k，第二个维度记录现在凑的金额是多少？如果没有第一个维度信息，当我们凑到金额i的时候，我们不知道之前有没有用到硬币k。

因为这是个**组合**问题，**我们不关心硬币使用的顺序，而是硬币有没有被用到。**是否使用第k个硬币受到之前情况的影响。

状态转移方程如下

```
if 金额数大于硬币
    DP[k][i] = DP[k-1][i] + DP[k][i-k]
else
    DP[k][i] = DP[k-1][i]
```

因此正确代码如下:

```
class Solution {
public:
    int change(int amount, vector<int>& coins) {
        int K = coins.size() + 1;
        int I = amount + 1;
        int DP[K][I];
        //初始化数组
        for (int k = 0; k < K; k++){
            for (int i = 0; i < I; i++){
                DP[k][i] = 0;
            }
        }
        //初始化基本状态
        for (int k = 0; k < coins.size() + 1; k++){
            DP[k][0] = 1;
        }
        for (int k = 1; k <= coins.size() ; k++){
            for (int i = 1; i <= amount; i++){
                if ( i >= coins[k-1]) {
                    DP[k][i] = DP[k][i-coins[k-1]] + DP[k-1][i];
                } else{
                    DP[k][i] = DP[k-1][k];
                }
            }
        }
        return DP[coins.size()][amount];
    }
};
```

我们初始化的数组大小为`coins.size()+1* (amount+1)`，这是因为第一列是硬币为0的基本情况。

此时，交换这里面的循环不会影响最终的结果。也就是

```
for (int i = 1; i <= amount; i++){
    for (int k = 1; k <= coins.size() ; k++){
        if ( i >= coins[k-1]) {
            DP[k][i] = DP[k][i-coins[k-1]] + DP[k-1][i];
         } else{
            DP[k][i] = DP[k-1][k];
         }
     }
}
```

之前爬楼梯问题中，我们将一维数组降维成点。这里问题能不能也试着降低一个维度，只用一个数组进行表示呢?

这个时候，我们就需要重新定义我们的子问题了。

此时的子问题是，对于硬币从0到k，我们必须使用第k个硬币的时候，当前金额的组合数。

因**此状态数组** `DP[i]` 表示的是对于第k个硬币能凑的组合数

状态转移方程如下

```
  DP[[i] = DP[i] + DP[i-k]
```

于是得到我们开头的第二个Solution。

```
class Solution {
public:
    int change(int amount, vector<int>& coins) {
        int dp[amount+1];
        memset(dp, 0, sizeof(dp)); //初始化数组为0
        dp[0] = 1;
        for (int coin : coins){ //枚举硬币
            for (int i = 1; i <= amount; i++){ //枚举金额
                if (i < coin) continue; // coin不能大于amount
                dp[i] += dp[i-coin];
            }
        }
        return dp[amount];
    }
};
```

好了，继续之前的问题，这里的内外循环能换吗?

显然不能，因为我们这里定义的子问题是，必须选择第k个硬币时，凑成金额i的方案。如果交换了，我们的子问题就变了，那就是对于金额i, 我们选择硬币的方案。

同样的，我们回答之前爬楼梯的留下的问题，原循环结构对应的子问题是，对于楼梯数i, 我们的爬楼梯方案。第二种循环结构则是，固定爬楼梯的顺序，我们爬楼梯的方案。也就是第一种循环下，对于楼梯3，你可以先2再1，或者先1再2，但是对于第二种循环

作者：徐洲更hoptop

链接：https://www.jianshu.com/p/42fe8e5192af

来源：简书