1273：货币系统

题目描述

给你一个n种面值的货币系统，求组成面值为m的货币有多少种方案。

输入

第一行为n和m。

输出

一行，方案数。

输入样例

```
3 10        //3种面值组成面值为10的方案
1           //面值1
2           //面值2
5           //面值5
```

输出样例

```
10          //有10种方案
```

解析

求什么，我们就定义什么。

首先，定义dp[i][j]的含义是前i种货币组成面值j的方案数。因此dp[0][0]=1，即使用不存在的货币组成面值为0的方案，只有1种。

接下来就是求解状态转移方程！

思路是这样，对于第i种货币，想组成面值为j的组合。一定存在两种情况，即i**加入**和**不加入**！

如果i不加入，那么方案数就是dp[i-1][j]，即前i-1种货币能够组成面值为j的方案数。

如果i加入，那么这个思路就非常类似《上楼梯》了。上楼梯的转移方程为：dp[i] = dp[i-1] + dp[i-2]，即到达第i层的方案数等于到达i-1层和i-2层的方案数之和。

对于i加入的情况，我们也可以把i看成上楼梯的最后一步，那么方案数就等于dp[i][j-v[i]]。

最终的状态转移方程为：

dp[i][j] = dp[i-1][j] + dp[i][j-[v[i]]。

对于这里，有些同学可能还会质疑，第二个式子为什么不是dp[i-1][j-[v[i]]，大家可以回忆一下完全背包，因为这里存在货币i可以被多次放置，因此，此处套用的就是完全背包的解法。

编码

```
#include<bits/stdc++.h>

using namespace std;

int MaxValue, n;  //目标面值和货币种类

long long v[9005];              //商品的价值
//动态规划表 f[0][0] = 1,用不存在面值的货币组成0元，只有一种方法
long long f[9005][9005] = {1};

int main() {
    //物品数量和目标面值
    cin >> n >> MaxValue;
    //记录每个物品价值
    for (long long i = 1; i <= n; i++) {
        cin >> v[i];
    }
    //遍历当前货币的每一种面值
    for (int i = 1; i <= n; ++i) {
        //从1开始遍历每一个目标面值，注意，一定是从0开始;
        for (int j = 0; j <= MaxValue; ++j) {
            //当前货币的面值大于目标面值，无法装入
```

```
            if (v[i] > j) {
                f[i][j] = f[i - 1][j];
            } else {
                //状态转移方程
                f[i][j] = f[i - 1][j] + f[i][j - v[i]];
            }
        }
    }
    //目标面值的最多组合数在最后一个格子
    cout << f[n][MaxValue];
    return 0;
}
```

一维优化

在完全背包中，我们能够将代码进行一维优化，在这里，我们同样可以将上面的代码进行优化。

```
#include<bits/stdc++.h>

using namespace std;

int MaxValue, n;  //目标面值和货币种类

//一维数组优化

//本题没有给出具体范围
long long v[9005];          //商品的价值
//动态规划表 f[0] = 1,用不存在面值的货币组成0元，只有一种方法
long long f[9005] = {1};

int main() {
    //物品数量和目标面值
    cin >> n >> MaxValue;
    //记录每个物品价值
    for (long long i = 1; i <= n; i++) {
        cin >> v[i];
    }
    //遍历当前货币的每一种面值
    for (int i = 1; i <= n; ++i) {
        //从1开始遍历每一个目标面值，注意，一定是从0开始;
        for (int j = 0; j <= MaxValue; ++j) {
            //当前货币的面值大于目标面值，无法装入
            if (v[i] > j) {
                f[j] = f[j];
            } else {
                //转移方程
                f[j] = f[j] + f[j - v[i]];
            }
```

```
            }
        }
        //目标面值的最多组合数在最后一个格子
        cout << f[MaxValue];
        return 0;
}
```