二维费用01背包：879. 盈利计划

问题描述

- 你有 n 个员工和 m 的最低利润目标。
- g[i] 表示执行第 i 个计划所需的员工数量，p[i] 表示第 i 个计划能够带来的利润。
- 需要计算有多少种不同的计划组合方式，使得组合中的利润总和大于等于 m，且参与计划的人数不超过 n。答案需要对 1e9 + 7 取模。

代码解析

1. 定义动态规划数组 dp

```cpp
vector<vector<vector<int>>> dp(len+1,vector<vector<int>> (n+1,vector<int>(m+1)));
```

- dp[i][j][k] 表示在前 i 个计划中，使用了 j 个员工，并且获得至少 k 的利润时，有多少种选择方式。
- len 是计划的数量。
- n 是员工的最大数量，m 是最低利润目标。
- dp[i][j][k] 的含义：前 i 个计划中使用 j 个员工，并获得至少 k 利润的方案数。

2. 初始化状态

```cpp
for(int j = 0;j <= n;j++) dp[0][j][0] = 1;
```

- 初始化 dp[0][j][0] = 1，表示没有任何计划时，使用任何数量的员工（从 0 到 n），并且利润为 0 时，有 1 种方案（即什么也不做）。

3. 动态规划的状态转移

```cpp
for(int i = 1;i <= len;i++)
{
    for(int j = 0;j <= n;j++)
    {
        for(int k = 0;k <= m;k++)
        {
            dp[i][j][k] = dp[i-1][j][k];  // 不选择当前计划 i
            if(j >= g[i-1])
            {
                // 选择当前计划 i
                dp[i][j][k] += dp[i-1][j-g[i-1]][max(0, k-p[i-1])];
                dp[i][j][k] %= MOD;
            }
        }
    }
}
```

- 外层循环 i：遍历所有计划 1 到 len。
- 中层循环 j：表示使用员工的数量，从 0 到 n。
- 内层循环 k：表示利润，从 0 到 m。
- 状态转移方程：
  - **不选择当前计划** i：dp[i][j][k] = dp[i-1][j][k]，表示不选计划 i，那么前 i 个计划的状态与前 i-1 个计划相同。
  - **选择当前计划** i：
    - 首先要求当前计划所需的员工数量满足条件 j >= g[i-1]，即员工数量 j 足够选中当前计划。
    - 然后从前 i-1 个计划中，减去当前计划所需的员工 g[i-1]，并保证总利润至少为 max(0, k - p[i-1])，即当前利润应至少是 k - p[i-1]。
    - 最后，将方案数 dp[i][j][k] 对 MOD 取模。

4. 返回结果

```cpp
return dp[len][n][m];
```

- 返回 dp[len][n][m]，表示在 n 个员工和至少 m 的利润条件下，能够选择的所有不同的方案数。

举例说明

假设有以下输入：

```plaintext
```

```
n = 5, m = 3
g = [2, 2], p = [2, 3]
```

解释

- n = 5 表示总共有 5 个员工可以使用。
- m = 3 表示最低利润要求是 3。
- 有两个计划：
  - 第一个计划需要 2 个员工，带来 2 的利润。
  - 第二个计划需要 2 个员工，带来 3 的利润。

我们需要找出多少种方式，使用不超过 5 个员工，得到至少 3 的利润。

动态规划表更新

1. **初始化**：
   - dp[0][j][0] = 1，表示没有选择任何计划时，利润为 0 时总有 1 种方案（不做任何事情）。
2. **处理第一个计划 (i = 1)**：
   - **不选择第一个计划**：dp[1][j][k] = dp[0][j][k]，即不选择任何计划时，状态与上一步相同。
   - **选择第一个计划**：
     - 如果员工数 j >= 2 且利润至少为 2：
       - dp[1][j][k] = dp[0][j-2][max(0, k-2)] + 1。
   - 结果：

plaintext

```
dp[1] = [[1, 0, 0, 0],
         [1, 0, 0, 0],
         [1, 0, 1, 0],  // 选择了第一个计划
         [1, 0, 1, 0],
         [1, 0, 1, 0],
         [1, 0, 1, 0]]
```

3. 处理第二个计划 (i = 2)：

最终结果：

- dp[2][5][3] = 1，表示在使用 5 个员工，最低利润为 3 的情况下，有 1 种方式选择计划。

时间复杂度

- 外层循环遍历 len 个计划，时间复杂度是 O(len)。
- 中层和内层循环分别遍历 n 和 m，时间复杂度是 O(n * m)。
- 因此，整体时间复杂度为 **O(len * n * m)**，即所有状态的遍历次数。

总结

这段代码使用了动态规划的思想，逐步根据当前可用的计划、员工数和利润要求更新状态，最终找到所有符合条件的方案数，并对结果取模。

```cpp
class Solution {
public:
    int profitableSchemes(int n, int m, vector<int>& g, vector<int>& p) {
        const int MOD = 1e9 + 7;
        int len = g.size();
        vector<vector<vector<int>>> dp(len+1,vector<vector<int>>(n+1,vector<int>(m+1)));
        for(int j = 0;j <= n;j++) dp[0][j][0] = 1;
        for(int i = 1;i <= len;i++)
        {
            for(int j = 0;j <= n;j++)
            {
                for(int k = 0;k <=m;k++)
                {
                    dp[i][j][k] = dp[i-1][j][k];
                    if(j >= g[i-1])
                    {
                        dp[i][j][k] += dp[i-1][j-g[i-1]][max(0,k-p[i-1])];
                        dp[i][j][k] %= MOD;
                    }
                }
            }
        }
        return dp[len][n][m];
    }
};
```