背包专题之完全背包

完全背包是指在有限的条件下，每样物品能无限放入的情况。

完全背包的状态转移方程有两种，我们逐一来进行详细的讲解。

三重循环方案

所谓的朴素方法，实际上就是将这些无限量的物品中的每一件，都当成01背包的独立物品进行放置，唯一的约束就是不能超过背包上限。

举例，对于物品1，我们可以不停的放入1件、2件……直到背包无法装入。然后来尝试第二件物品，装入1件、2件、3件……以此类推。

原01背包的状态转移方程如下：

```
//从放入第一件物品开始
for (int i = 1; i <= n; i++) {
   //从第一个格子开始尝试
      for (int j = 1; j <= bagV; j++) {
       //如果当前的格子的重量小于目标物品的重量，则价值等于前一个物品的价值
       if (j < w[i])
           dp[i][j] = dp[i - 1][j];
       else
           dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);
   }
}
```

我们对其进行简单的修改：

```
//从放入第一件物品开始
for (int i = 1; i <= n; i++) {
   //从第一个格子开始尝试
      for (int j = 1; j <= bagV; j++) {
       //如果当前的格子的重量小于目标物品的重量，则价值等于前一个物品的价值
            for(int k=0; k*w[i]<=j; k++) {
             dp[i][j] = max(dp[i][j], dp[i-1][j - k*w[i]] + k*v[i]);
       }
   }
}
```

我们在01背包的基础上增加了第三个循环k，代表的是每个物品装载的数量，需要注意的是，所能装载的最大重量不能超过当前背包的重量。

这个方法最大的问题就是三重循环，复杂度太高。所以，基本上我们不会使用这个方法。

二重循环方案

下面的表格是一个我们按照三重循环方案填好的表格，仔细思考。

物品信息			背包容量							
索引	重量	价值	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	2	2	3	3
2	3	4	0	0	1	4	4	5	8	8

观察表格中的蓝色方块，我们想得到这个位置的最大值，使用的想法依然和01背包是完全相同的，即

放置物品i后的最大价值 = 放置物品i前的最大价值 + 物品i的价值

其中物品的价值是来源于输入，我们无需理会。但是，放置物品前的最大价值该如何求解呢？

大家很容易想到dp[2][5] = dp[1][2] + 3 = 4，即dp[i][j] = dp[i-1][j-w[i]]+v[i]。这就是我们01背包的状态转移方程。

看起来一切正常，但是，这样真的对吗？当然不对！

继续观察后面的绿色方块，正确的结果应该是8，即在容量为6时，我们可以放入3个2号物品。但是，如果我们按照上面的方程计算，得出来的结果却是5。这是因为01背包的转移方程会遗漏某件物品被多次放入的情况。

继续思考，放置物品i后，描述最大价值的数据存在哪里了呢？很明显，就是第i行嘛。在我们的背包问题中，每一行代表的含义都是第i个物品做了选择后的最大价值。

现在，新的递推公式应该是这个样子的：

```
//如果当前的格子的重量小于目标物品的重量，则价值等于前一个物品的价值
if (j < w[i])
    dp[i][j] = dp[i - 1][j];
else
    dp[i][j] = max(dp[i - 1][j], dp[i][j - w[i]] + v[i]);
```

仔细对比，我们可以发现完全背包的二维循环方案与01背包的公式就差了一个i-1。正是这一点点的细微差异，就导致了二者结果的完全不同，是不是很有趣呢？

真题讲解

一本通 1268：完全背包问题

题目描述

设有n种物品，每种物品有一个重量及一个价值。但每种物品的数量是无限的，同时有一个背包，最大载重量为M，今从n种物品中选取若干件(同一种物品可以多次选取)，使其重量的和小于等于M，而价值的和为最大。

输入

第一行：两个整数，M(背包容量，M≤200)和N(物品数量，N≤30)；

第2..N+1行：每行二个整数Wi,Ci，表示每个物品的重量和价值。

输出

仅一行，一个数，表示最大总价值。

输入样例

```
10 4
2 1
3 3
4 5
7 9
```

输出样例

```
max=12
```

编码

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

using namespace std;

int bagV, n;

int w[31];          //商品的体积
int v[31];          //商品的价值

int dp[31][201] = {{0}};                //动态规划表

int main() {

   //记录最大承重和物品数量
      cin >> bagV >> n;
   //记录每个物品的重量和价值
      for (int i = 1; i <= n; i++) {
       cin >> w[i] >> v[i];
   }

   //从放入第一件物品开始
      for (int i = 1; i <= n; i++) {
       //从第一个格子开始尝试
```

```
            for (int j = 1; j <= bagV; j++) {
            //如果当前的格子的重量小于目标物品的重量，则价值等于前一个物品的价值
                    for (int k = 0; k * w[i] <= j; k++) {
                dp[i][j] = max(dp[i][j], dp[i - 1][j - k * w[i]] + k * v[i]);
                }
            }
        }

    //01背包的最大值在最后一个格子中
        cout << "max=" << dp[n][bagV];

    return 0;
}
```