# 背包专题之多重背包

多重背包是指在给定的背包容量下，物品拥有不同的数量和价值，如何对其进行组合，使背包的价值达到最大。

## 1269：庆功会

### 题目描述

为了庆贺班级在校运动会上取得全校第一名成绩，班主任决定开一场庆功会，为此拨款购买奖品犒劳运动员。期望拨款金额能购买最大价值的奖品，可以补充他们的精力和体力。

### 输入

第一行二个数n(n≤500)，m(m≤6000)，其中n代表希望购买的奖品的种数，m表示拨款金额。

接下来n行，每行3个数，v、w、s，分别表示第I种奖品的价格、价值（价格与价值是不同的概念）和能购买的最大数量（买0件到s件均可），其中v≤100，w≤1000，s≤10。

### 输出

一行：一个数，表示此次购买能获得的最大的价值（注意！不是价格）。

### 输入样例

```
5 1000
80 20 4
40 50 9
30 50 7
40 30 6
20 20 1
```

### 输出样例

```
1040
```

### 解析

### 朴素算法

最直接的朴素算法就是把每类物品中的**s件**看成**s类**，每类只有1个。这样变化以后，整个题目就变成了最原始的01背包，然后我们就可以直接套用01背包。

代码实现如下：

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

using namespace std;

//dp[i][j]  购买前i件物品达到最大价值j
//第一维：物品的数量500*10+1，500物品的种类，10购买的最大件数
//第二维：预算的最大费用6000
int dp[5001][6001];
int w[5001];  //单个物品的最大价值
int v[5001];  //单个物品的价格

int n, m; //物品的种类和预算
int wi, vi, si;  //临时变量：物品i的价值、价格、数量;
int k;  //转化为01背包后的物品索引

int main() {
    //读入  物品的种类和预算
    cin >> n >> m;
    for (int i = 0; i < n; ++i) {
        //输入  价格     价值    数量
        cin >> vi >> wi >> si;
        //将多重背包转换为01背包
        for (int j = 0; j < si; ++j) {
            //从1开始计数
            k++;
            v[k] = vi;
            w[k] = wi;
        }
    }
    //调用01背包
    //注意，此处物品的种类已经不在是n，而转变成了k
    for (int i = 1; i <= k; ++i) {
        for (int j = 0; j <= m; ++j) {
            if (j >= v[i]) {
                dp[i][j] = max(dp[i][j], dp[i - 1][j - v[i]] + w[i]);
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }

    cout << dp[k][m];

    return 0;
}
```

一维数组优化

当然，我们可以将以上的代码进行二进制优化。代码如下：

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

using namespace std;

//dp[i][j]  购买前i件物品达到最大价值j
//第一维：物品的数量500*10+1，500物品的种类，10购买的最大件数
//第二维：预算的最大费用6000
int dp[6001];
int w[5001];  //单个物品的最大价值
int v[5001];  //单个物品的价格

int n, m; //物品的种类和预算
int wi, vi, si;  //临时变量：物品i的价值、价格、数量;
int k;  //转化为01背包后的物品索引

int main() {
    //读入  物品的种类和预算
    cin >> n >> m;
    for (int i = 0; i < n; ++i) {
        //输入  价格     价值    数量
            cin >> vi >> wi >> si;
        //将多重背包转换为01背包
            for (int j = 0; j < si; ++j) {
            //从1开始计数
                k++;
            v[k] = vi;
            w[k] = wi;
        }
    }
    //调用01背包
      //注意，此处物品的种类已经不在是n，而转变成了k
    for (int i = 1; i <= k; ++i) {
        for (int j = m; j >= v[i]; --j) {
            dp[j] = max(dp[j], dp[j - v[i]] + w[i]);
        }
    }

    cout << dp[m];

    return 0;
}
```

二进制优化

在朴素的算法中，由于我们将s件物品看成了s个。虽然便于理解，但是我们无形中增加了算法的复杂度——变成了三重循环，这样就导致我们在一些高标准的比赛中，无法通过。因此，二进制解法应运而生。

二进制解法的根本思想：

1、将每类物品的分组尽可能的拆分较少的数量

2、二进制可以表达出任意的数字

举例，数字7可以被拆分成1,2,4，它们分别对应$2^0$，$2^1$，$2^2$，同时，这些数字任意的组合也能够表达出1-7全部的数字，而不存在任何遗漏。这样就能够保证，我们确确实实的尝试过1-7中各种组合。具体拆分代码如下：

```
for (int i = 0; i < n; ++i) {
   //输入  价格    价值   数量
    cin >> vi >> wi >> si;
   //开始进行二进制拆分
    for (int j = 1; j <= si; j <<= 1) {
    //将所有的物品数量按照 1,2,4……进行拆分
         v[k] = j * vi;
      w[k] = j * wi;
      si -= j;
      k++;
   }
   //不够2进制数的，直接保留下来
     if (si > 0) {
      v[k] = si * vi;
      w[k] = si * wi;
      k++;
   }
}
```

完整AC代码

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

using namespace std;

//dp[i][j]  购买前i件物品达到最大价值j
//第一维：物品的数量500*10+1，500物品的种类，10购买的最大件数
//第二维：预算的最大费用6000
int dp[6001];
int w[5001];  //单个物品的最大价值
int v[5001];  //单个物品的价格

int n, m; //物品的种类和预算
int wi, vi, si;  //临时变量：物品i的价值、价格、数量;
int k = 1;  //将物品数量按照二进制拆分后的索引数
```

```cpp
int main() {
    //读入 物品的种类和预算
      cin >> n >> m;
    for (int i = 0; i < n; ++i) {
      //输入 价格    价值   数量
           cin >> vi >> wi >> si;
      //开始进行二进制拆分
           for (int j = 1; j <= si; j <<= 1) {
        //将所有的物品数量按照 1,2,4……进行拆分
                v[k] = j * vi;
        w[k] = j * wi;
        si -= j;
        k++;
      }
      //不够2进制数的，直接保留下来
       if (si > 0) {
        v[k] = si * vi;
        w[k] = si * wi;
        k++;
      }
    }
    //调用01背包
      //注意，此处物品的种类已经不在是n，而转变成了k
    for (int i = 1; i < k; ++i) {
      for (int j = m; j >= v[i]; --j) {
        dp[j] = max(dp[j], dp[j - v[i]] + w[i]);
      }
    }

    cout << dp[m];

    return 0;
}
```