# 一、回顾

## 多重背包 一

上一个文档我们讲到了01背包的变形之多重背包

大家应该还记得，如果不记得就去翻看一下。

下面我们看一下多重背包 二

# 二、多重背包 二

## 题目

有 N 种物品和一个容量是 V 的背包。

第 ii 种物品最多有 s【i】 件，每件体积是 v【i】，价值是 w【i】。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。
输出最大价值。

## 输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 v【i】,w【i】,s【i】用空格隔开，分别表示第 ii 种物品的体积、价值和数量。

## 输出格式

输出一个整数，表示最大价值。

## 数据范围

$$0 < N \leq 10000 < V \leq 20000 < vi, wi, si \leq 2000$$

## 输入样例

```
4 5
1 2 3
2 4 1
3 4 3
4 5 2
```

## 输出样例：

```
10
```

## 提示

该题使用位运算优化

# 与多重背包一作比较

我们可以发现，其实都一样，唯一不一样的就是数据范围

> 我们可以发现，多种背包的数据范围 都在100 之内，最大复杂度也就是100 * 100 * 100也就是一百万，为什么这么计算呢，你们想一下代码，我们要遍历容量，我们要展开每一种背包的数量，我们也要遍历容量的可能性。
> 而这个多重背包二呢？他的复杂度就是 1000 * 2000 * 2000 也就是约为 4 * 1e9 多必然会超时

# 解题思路

比如 物品1 我们有10个 那么我们就将物品1 分解成 0 1 2 4 2 三种情况。因为这三种情况已经可以包括0到 10 了

# 核心二进制换算代码

```
        for(int i = 1; i <= s; i *= 2)
        {
            s -= i;
            v[t] = v1*i;
            w[t] = w1*i;
            t++;
         }
        if(s > 0)
        {
            v[t] = s * v1;
            w[t] = w1 * s;
            t ++;
        }
```

其实就相当于，本来的把有n种把每一种拆分成n个，优化成现在的把n个拆分成logn向上取整(就是 log 10 向上取整为4)

看了这些应该都能理解这道题了。那么我们就开始上整体的代码了，仅供参考哦，有错误记得给我说

```
#include<iostream>
using namespace std;

const int N1 = 2010;
const int N2 = 20100;
int dp[N1];
int v[N2],w[N2];
int v1,m,w1,s,n,t;

int main()
{
    cin >> n >> m;
    while(n --)
    {
        cin >> v1 >> w1  >> s;
        // 二维转换
        for(int i = 1; i <= s; i *= 2)
        {
            s -= i;
            v[t] = v1*i;
```

```
            w[t] = w1*i;
            t++;
         }
        if(s > 0)
        {
            v[t] = s * v1;
            w[t] = w1 * s;
            t ++;
        }
    }
    // 套用01背包模板
    for(int i = 0; i < t; i++)
    {
        for(int j = m; j >=v[i]; j--)
        {
            dp[j] = max(dp[j], dp[j - v[i]] + w[i]);
        }
     }
     cout << dp[m] << endl;

    return 0;
}
```

# 三、多重背包 三 《究极版》

## 题目

有 N 种物品和一个容量是 V 的背包。

第 ii 种物品最多有 s【i 】件，每件体积是 v【i】，价值是 w【i】。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。
输出最大价值。

## 输入格式

第一行两个整数，N，V (0<N≤1000(0<N≤1000, 0<V≤20000)0<V≤20000)，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 v【i】,w【i】,s【i】，用空格隔开，分别表示第 ii 种物品的体积、价值和数量。

## 输出格式

输出一个整数，表示最大价值。

## 数据范围

$$0 < N \leq 10000 < V \leq 200000 < vi, wi, si \leq 20000$$

## 提示

该题使用队列优化、

**在讲解这个题之前 我要讲解一道该题利用到一个基础思想的题目**

# 嵌套一个题

## 题目

给定一个大小为 n≤106n≤106 的数组。

有一个大小为 kk 的滑动窗口，它从数组的最左边移动到最右边。

你只能在窗口中看到 kk 个数字。

每次滑动窗口向右移动一个位置。

以下是一个例子：

该数组为 `[1 3 -1 -3 5 3 6 7]`，kk 为 33。

| 窗口位置 | 最小值 | 最大值 |
| --- | --- | --- |
| [1 3 -1] -3 5 3 6 7 | -1 | 3 |
| 1 [3 -1 -3] 5 3 6 7 | -3 | 3 |
| 1 3 [-1 -3 5] 3 6 7 | -3 | 5 |
| 1 3 -1 [-3 5 3] 6 7 | -3 | 5 |
| 1 3 -1 -3 [5 3 6] 7 | 3 | 6 |
| 1 3 -1 -3 5 [3 6 7] | 3 | 7 |

你的任务是确定滑动窗口位于每个位置时，窗口中的最大值和最小值。

## 输入格式

输入包含两行。

第一行包含两个整数 nn 和 kk，分别代表数组长度和滑动窗口的长度。

第二行有 nn 个整数，代表数组的具体数值。

同行数据之间用空格隔开。

## 输出格式

输出包含两个。

第一行输出，从左至右，每个位置滑动窗口中的最小值。

第二行输出，从左至右，每个位置滑动窗口中的最大值。

## 输入样例

```
8 3
1 3 -1 -3 5 3 6 7
```

## 输出样例

```
-1 -3 -3 -3 3 3
3 3 5 5 6 7
```

## 思路

首先我们要思考一下 假如让你用队列题做的话，你会怎么去思考呢?

1. 用普通的队列做该题时，怎么做呢?
2. 普通队列的时间复杂度时NK 我们怎么优化呢?
3. 优化方法：在求最小值时，假如队尾值比要入队的值大了，那么我们就可以直接舍弃队尾。 这样的话我们的队列就是一个单调递增的队列，我们求解的时候，这样的话 我们在拿最小值时就只需要直接从队头取值即可

## 代码

看代码理解该题

```
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
const int N = 1e6 + 10;
int q[N];
int a[N];
int n,k;
int main()
{
    cin >> n >> k;
    for(int i = 0; i < n; i++) scanf("%d",&a[i]);
    int hh = 0, tt = -1;
    for(int i = 0; i < n; i++)
    {
        if(hh <= tt && q[hh] < i - k + 1) hh ++;

        while(hh <= tt && a[q[tt]] >= a[i]) tt--;
        q[++tt] = i;// 入队
        if(i >= k -  1) printf("%d ",a[q[hh]]);
    }
    printf("\n");

    hh = 0, tt = -1;
    for(int i = 0; i < n; i++)
    {
        if(hh <= tt && q[hh] < i - k + 1) hh ++;
        while(hh <= tt && a[q[tt]] <= a[i]) tt--;// 直接删除队尾元素
        q[++tt] = i;// 入队
        if(i >= k -  1) printf("%d ",a[q[hh]]); //打印队头元素
    }
    return 0;
}
```

## 核心代码

```
    for(int i = 0; i < n; i++)
    {
        if(hh <= tt && q[hh] < i - k + 1) hh ++;
        // 在保证队列有元素的情况下，只要窗口太大了，我们将删除对头 hh <= tt 可以不加，因为后面的判断就已经可以保证 hh <= tt了
        while(hh <= tt && a[q[tt]] >= a[i]) tt--;
        // 在保证队列有元素的情况下，如果队尾的值大于入队元素的值 我们将删除队尾
        q[++tt] = i;// 入队
        if(i >= k -  1) printf("%d ",a[q[hh]]);
    }
```

我们先总体讲解一下代码的意思

首先 如果队列种的元素 超过窗口要求的数量，那么我们将直接删除对头元素，如果队尾大于要插入的元素的值，我们将直接删除队尾即可

具体每一句代码的意思直接看代码注释即可

# 下面我们继续看多重背包

经过我们的滑动窗口题 我们已经对单调队列优化滑动窗口有了简单的理解，那么我们下面将直接写出当前的背包问题，那里比较难理解我将写在注释上

```
#include <iostream>

using namespace std;

const int N = 1010, M = 20010;

int n, m;
int v[N], w[N], s[N];
int f[N][M];
int q[M];

int main()
{

    cin >> n >> m;
    for (int i = 1; i <= n; ++ i) cin >> v[i] >> w[i] >> s[i];
    for (int i = 1; i <= n; ++ i)
    {
        for (int r = 0; r < v[i]; ++ r)
        {
            int hh = 0, tt = -1;
            for (int j = r; j <= m; j += v[i])
            {
                if (hh <= tt && j - q[hh] > s[i] * v[i]) hh ++ ;
                while (hh <= tt && f[i - 1][q[tt]] + (j - q[tt]) / v[i] * w[i] <= f[i - 1][j]) -- tt;
                q[ ++ tt] = j;
                f[i][j] = f[i - 1][q[hh]] + (j - q[hh]) / v[i] * w[i];  // j - q[hh] 为 剩余的背包空间，/ v[i] 表示有多少个，*w[i]表示价值增加了多少
            }
        }
```

```
    }
    cout << f[n][m] << endl;
    return 0;
}
```