# 习题题解

# 合唱队形



题意：可以从队列里面淘汰学生，使得整个队列成为一个合唱队列，求最少淘汰的人数

草稿纸上，分析之后
问题可以转换为：求从左往右LIS和从右往左LIS之和的最大值ans，答案为n-ans+1

枚举队列中的每一个学生，以该学生i为核心
分别求出其左侧的最长递增子序列dp1[]和其右侧的最长递减子序列dp2[]
dp1[]+dp2[]-1就是以该同学为中心的合唱队的人数，-1是因为这个学生被算了两次

所以我们只需要把每个学生都作为中心遍历一遍，就能得出人数最多的合唱队形，再把总人数减去合唱人数就是需要剔除的人数

# 摆花



题意：有n种花,每种花都有它自己的数量a[i],现在要求将这些花按照编号从小到大排成m盆花的方案数
简而言之，求用前n种花摆m盆的方案数。

状态定义：故设f[i][j]:用前i种花摆前j盆的方案数,最后的答案就是f[n][m]。

对于第i种花可以使用0、1、2...a[i]盆,对应的前i-1种花摆放的盆数为j-0、j-1、j-2、...j-a[i]
即f[i][j]=f[i-1][j]+f[i-1][j-1]+f[i-1][j-2]+...+f[i-1][j-a[i]]

状态转移方程：f[i][j]= f[i-1][j-k](0<=k<=a[i],j>=k)

初始值：f[i][0]=1;

# 代码

```
for (int i=2;i<=n;i++)
  for(int j=1;j<=m;j++)
   for(int k=0;k<=a[i];k++)
    if (j>=k)
        f[i][j]=(f[i][j]+f[i-1][j-k])% 1000007;

cout<<f[n][m]<<endl;
```

# 石子合并



区间DP经典题目
f[i][j]表示i到j的最小值
枚举断点k把大区间划分为多个小区间求解

```
for (int i = 1; i <= n; i++) {  //预处理前缀和
        cin >> a[i];
        s[i] = s[i - 1] + a[i];
    }
    for (int i = 1; i <= n; i++) {  //初始化
        dp[i][i] = 0;
    }
    for (int l = 2; l <= n; l++) {   //以区间长度为阶段
        for (int i = 1; i <= n - l + 1; i++) {
            int j = i + l - 1;
            for (int k = i; k < j; k++) {
                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j] + s[j] - s[i - 1]);
            }
        }
    }
    cout << dp[1][n];
```

# 编辑距离

题意：提供删除、增加、替换三种操作，问把A串变B串的最小步数

状态定义：dp[i][j]表示子串A[0~i]与子串B[0~j]的编辑距离。

状态转移方程为

A[i]==B[j]时，

```
dp[i][j]=dp[i-1][j-1];
```

A[i]≠B[j]时，

```
dp[i][j]=min(dp[i-1][j],dp[i][j-1],dp[i-1][j-1])+1;
```

# 括号匹配



题意：问最少需要添加多少括号才能使得匹配?

状态定义：dp[i][j]表示i到j位置需要添加括号的数量

- 如果当前i和j位置的括号可以匹配，那么[i,j]之间需要的括号数和[i+1,j-1]的相同，所以dp[i][j]=dp[i+1][j-1]
- 如果i和j不匹配，找到断点k，k与i匹配，k+1与j匹配，由于不知道哪个k最优，需要枚举k

  dp[i][j]=min(dp[i][j],dp[i][k]+dp[k+1][j])

因为要找最小的次数，所以给dp[i][j]的初始值可以设为字符串的长度（足够大也可以），当i==j的时候，至多需要1个括号就能匹配了，所以初始值为1

# 参考代码



```
for (int i = 0; i < l; i++) {
    dp[i][i] = 1;
}
for (int i = l - 2; i >= 0; i--) {
    for (int j = i + 1; j < l; j++) {
        dp[i][j] = l;
        if (check(ch[i], ch[j]))  //check检验括号是否匹配
            dp[i][j] = dp[i + 1][j - 1];
        for (int k = i; k < j; k++)
            dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j]);
    }
}
cout << dp[0][l - 1] << endl;
```

# 背包问题

【问题描述】

一个旅行者有一个最多能用m公斤的背包，现在有n件物品，它们的重量分别是W1，W2，...,Wn,它们的价值分别为C1,C2,...,Cn.若每种物品只有一件求旅行者能获得最大总价值。

【输入格式】

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。

【输出格式】

仅一行，一个数，表示最大总价值。

【样例输入】

```
10 5
2 6
2 3
6 5
5 4
4 6
```

【样例输出】

```
15
```


信息学暑期

# 背包型DP

(概念与基本DP模型)

# 学习目标



需要掌握的背包模型：

1.01背包
2.完全背包
3.多重背包
4.混合背包(以上三个背包的结合体)
5.二维费用背包
6.分组背包
7.简单依赖背包

目前无需了解：泛化物品(树形DP)

# 1.01背包

贪心是否可以?

# 你能分别举出反例吗?

贪心方法:

方案一:
体积小的

方案二:
价值高的

方案三:
性价比（价值/体积）高的

| 容量 | 物品 |
|---|---|
| 3 | 3 |
| 体积 | 价值 |
| 1 | 1 |
| 1 | 1 |
| 2 | 4 |

【方案一反例】

| 容量 | 物品 |
|---|---|
| 4 | 3 |
| 体积 | 价值 |
| 3 | 4 |
| 2 | 3 |
| 2 | 3 |

【方案二反例】

| 容量 | 物品 |
|---|---|
| 4 | 3 |
| 体积 | 价值 |
| 3 | 5 |
| 2 | 3 |
| 2 | 3 |

【方案三反例】

物品可分割时，可以用贪心解决

# 01背包问题

例如，有5个物品，其重量分别是{2, 2, 6, 5, 4}，价值分别为{6, 3, 5, 4, 6}，背包的容量为10。

用一个(n+1)×(C+1)的二维表dp[ ][ ]

dp[i][j]表示把前i个物品装入容量为j的背包中获得的最大价值。

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | j |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | 0 | | | | | | | | | | | | |
| $w_1=2$ $v_1=6$ | 1 | | | | | | | | | | | | |
| $w_2=2$ $v_2=3$ | 2 | | | | | | | | | | | | |
| $w_3=6$ $v_3=5$ | 3 | | | | | | | | | | | | |
| $w_4=5$ $v_4=4$ | 4 | | | | | | | | | | | | |
| $w_5=4$ $v_5=6$ | 5 | | | | | | | | | | | | |

# 01背包问题



填表的过程，是按只放第1个物品、只放前2个、只放前3个……一直到放完，这样的顺序考虑。(从小问题扩展到大问题)

1、只装第1个物品。(横向是递增的背包容量)

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $w_1=2$ $v_1=6$ | 1 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $w_2=2$ $v_2=3$ | 2 | | | | | | | | | | | |
| $w_3=6$ $v_3=5$ | 3 | | | | | | | | | | | |
| $w_4=5$ $v_4=4$ | 4 | | | | | | | | | | | |
| $w_5=4$ $v_5=6$ | 5 | | | | | | | | | | | |

2、只装前2个物品。如果第2个物品重量比背包容量大，那么不能装第2个物品，情况和只装第1个一样。如果第2个物品重量小于背包容量，那么：

（1）如果把物品2装进去(重量是2)，那么相当于只把1装到(容量-2)的背包中。

（2）如果不装2，那么相当于只把1装到背包中。取（1）和（2）的最大值。

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $w_1=2$ $v_1=6$ | 1 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $w_2=2$ $v_2=3$ | 2 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| $w_3=6$ $v_3=5$ | 3 | | | | | | | | | | | |
| $w_4=5$ $v_4=4$ | 4 | | | | | | | | | | | |
| $w_5=4$ $v_5=6$ | 5 | | | | | | | | | | | |

**需要用到前面的结果，即已经解决的子问题的答案。**

# 01背包问题

3、只装前3个物品。如果第3个物品重量比背包大，那么不能装第3个物品，情况和只装第1、2个一样。如果第3个物品重量小于背包容量，那么：

（1）如果把物品3装进去(重量是2)，那么相当于只把1装到(容量-2)的背包中。

（2）如果不装3，那么相当于只把1、2装到背包中。 取（1）和（2）的最大值。

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $w_1=2$ $v_1=6$ | 1 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $w_2=2$ $v_2=3$ | 2 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| $w_3=6$ $v_3=5$ | 3 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 11 | 11 | 14 |
| $w_4=5$ $v_4=4$ | 4 | | | | | | | | | | | |
| $w_5=4$ $v_5=6$ | 5 | | | | | | | | | | | |

# 01背包问题

西南大学附属中学
High School Affiliated to Southwest University

最终填出来的表格：

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $w_1=2\ v_1=6$ | 1 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $w_2=2\ v_2=3$ | 2 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| $w_3=6\ v_3=5$ | 3 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 11 | 11 | 14 |
| $w_4=5\ v_4=4$ | 4 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 10 | 11 | 13 | 14 |
| $w_5=4\ v_5=6$ | 5 | 0 | 0 | 6 | 6 | 9 | 9 | 12 | 12 | 15 | 15 | 15 |

物品的件数就是阶段

# 01背包问题



## 1.划分阶段

物品件数

## 2.确定状态和状态变量

F[i,j] 表示前 i 件物品恰放入一个容量为 j 的背包可以获得的最大价值;

## 3.确定决策并写出状态转移方程

F[i,j] = max {F[i - 1,j],F[i - 1,j - W i ] + C i } (j>=Wi)

## 4.寻找边界条件

f[0][ ]=0;

# 01背包问题



```
for (int i=0; i<=m; i++)
        dp[0][i]=0;
for (int i=1; i<=n; i++) //物品种类数
    for (int v=1; v<=m; v++) //背包容量，m总容量
       if (w[i] <= v) //w[i] 体积 c[i]价值
             dp[i][v] = max(dp[i-1][v],dp[i-1][v-w[i]]+c[i]);
       else
             dp[i][v] = dp[i-1][v];
```

# 拓展:如何输出方案?



| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $x_1=1$ |
| $w_1=2$ $v_1=6$ | 1 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | $x_2=1$ |
| $w_2=2$ $v_2=3$ | 2 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | $x_3=0$ |
| $w_3=6$ $v_3=5$ | 3 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 11 | 11 | 14 | $x_4=0$ |
| $w_4=5$ $v_4=4$ | 4 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 10 | 11 | 13 | 14 | $x_5=1$ |
| $w_5=4$ $v_5=6$ | 5 | 0 | 0 | 6 | 6 | 9 | 9 | 12 | 12 | 15 | 15 | 15 | |

辅助数组记录下选择的情况

# 01背包问题:空间优化1

如果物品和空间均超过 8000 意味着。。。。
dp[8000][8000] =64000000 > 128M 内存可开int数 == 爆空间 （MLE）

【如何压缩？】
【分析一下？】

由于只用到了f[i-1][]和f[i][]这两层
所以我们可以利用滚动数组优化空间

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $w_1=2$ $v_1=6$ | 1 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $w_2=2$ $v_2=3$ | 2 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| $w_3=6$ $v_3=5$ | 3 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 11 | 11 | 14 |
| $w_4=5$ $v_4=4$ | 4 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 10 | 11 | 13 | 14 |
| $w_5=4$ $v_5=6$ | 5 | 0 | 0 | 6 | 6 | 9 | 9 | 12 | 12 | 15 | 15 | 15 |

```
F[i][j]=max(f[i^1][j-1],f[i^1][j-ci]+wi);
i^=1;
```

空间复杂度：O(n*m)变成了O(2*n)

# 01背包问题:空间优化2

西南大学附属中学
High School Affiliated to Southwest University

如果物品和空间均超过 8000 意味着。。。。
dp[8000][8000] =64000000 > 128M 内存可开int数 == 爆空间 （MLE）

递推过程只用到了
i-1阶段的 j 和 j-w[i] 位置

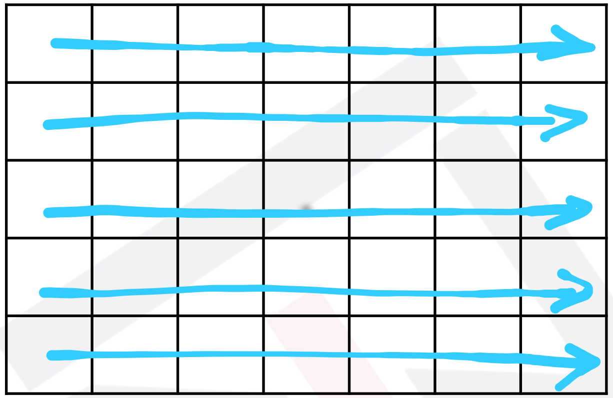
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $w_1=2\ v_1=6$ | 1 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $w_2=2\ v_2=3$ | 2 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| $w_3=6\ v_3=5$ | 3 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 9 | 11 | 11 | 14 |
| $w_4=5\ v_4=4$ | 4 | 0 | 0 | 6 | 6 | 9 | 9 | 9 | 10 | 11 | 13 | 14 |
| $w_5=4\ v_5=6$ | 5 | 0 | 0 | 6 | 6 | 9 | 9 | 12 | 12 | 15 | 15 | 15 |

**如果把n个阶段压到一维数组中…**

# 01背包问题:空间优化



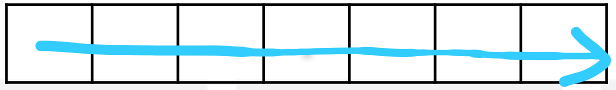
原来的计算过程：



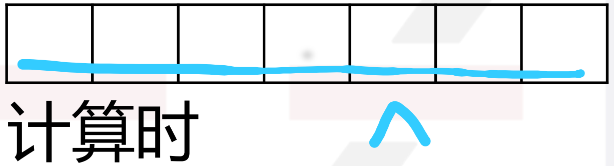
每次计算要用到上阶段的的 v 和 v-w[i] 位置

空间复杂度：O(n*m)变成了O(n)

现在的计算过程

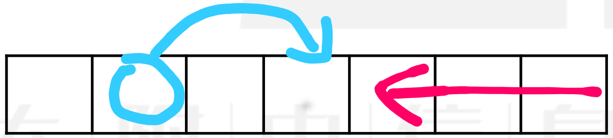


当你已经算好一排后



计算时

此点需要之前计算出的前面位置的点



逆序能够保证在调用dp[j-w[i]]是保存的是原来的dp[i-1][v-w[i]]的值

# 01背包问题:空间优化

```
for (int i=0; i<=m; i++)
        dp[0][i]=0;
for (int i=1; i<=n; i++) //物品种类数
    for (int v=1; v<=m; v++) //背包容量，m总容量
       if (w[i] <= v) //w[i] 体积 c[i]价值
             dp[i][v] = max(dp[i-1][v],dp[i-1][v-w[i]]+c[i]);
       else
             dp[i][v] = dp[i-1][v];
```

空间优化

```
for (int i = 1; i <= n; i++)
    for (int v = m; v >=1; v--)
        if (w[i] <= v)
            dp[v] = max(dp[v],dp[v-w[i]]+c[i]);
```

# 01背包问题:时间优化



```
for (int i = 1; i <= n; i++)
    for (int v = m; v >=1; v--)
        if (w[i] <= v)
            dp[v] = max(dp[v],dp[v-w[i]]+c[i]);
```

不必要判断

压缩
时间

```
for (int i = 1; i <= n; i++)
    for (int v = m; v >= w[i]; v--)
        dp[v] = max(dp[v], dp[v-w[i]]+c[i]);
```

# 2.完全背包

# 完全背包

有N种物品和一个容量为V的背包。第i种物品是w[i]，价值是c[i]，**每种物品数量不限**。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

```
for (int i = 1; i <= n; i++) //i种类
    for (int v = 1; v <=m; v++) //体积
        for(int k=0;k<=v;k++) //选k份
            if (k*w[i] <= v)
                dp[i][v] = max(dp[i-1][v],dp[i-1][v-k*w[i]]+k*c[i]);
               else
                dp[i][v] = dp[i-1][v];
```

## 解法二：

```
for (int i = 1; i <= n; i++)
        for (int v = w[i]; v <=m; v++)
                dp[v] = max(dp[v],dp[v-w[i]]+c[i]);
```

## 压缩空间的01背包

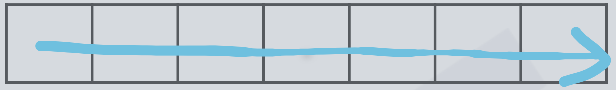
```
for (int i = 1; i <= n; i++)
        for (int v = m; v >= w[i]; v--)
                dp[v] = max(dp[v],dp[v-w[i]]+c[i]);
```

**思考：顺序不一样，为什么？**

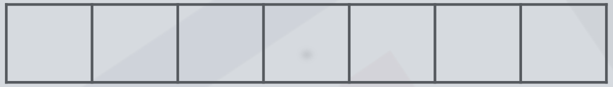
# 完全背包

现在的计算过程

01背包的选择

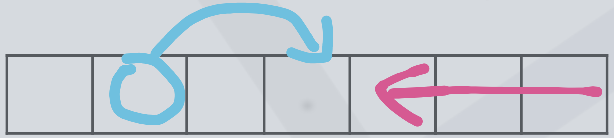


当你已经算好一排后



计算时

此点需要之前计算出的前面位置的点



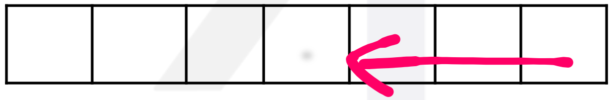
~~逆序能够保证在调用~~

dp[v-w[i]]是保存的是原

来的dp[i-1][v-w[i]]的值

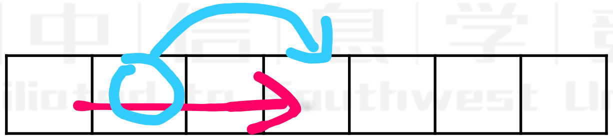
之前的逆序是为了满足

每件物品用0/1次这个条件



现在的顺推是为了满足

每个物品可重复用这个条件



在推新状态时，

旧状态就可能已经用过了此件物品

是谓 复用

【问题描述】

一个旅行者有一个最多能用m公斤的背包，现在有n件物品，它们的重量分别是W1，W2，...,Wn,它们的价值分别为C1,C2,...,Cn.若每种物品只有一件求旅行者**恰好装满背包**能获得最大总价值。

【输入格式】

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。

【输出格式】

仅一行，一个数，表示最大总价值。

【样例输入】

10 5
2 6
2 3
6 5
5 4
4 6

## 需要将背包装满的情况

F[0][0]

一件都不放也装满了背包（背包的剩余容量为0）是合法的，价值为0；

F[0][v]

一件都不放是不合法的，价值为-∞ ；

$F[0][0]=0;$

$F[0][1...v]= -\infty;$

# 3.多重背包

# 多重背包



有N种物品和一个容量为V的背包。第i种物品**最多有n[i]件可用**，每件费用是w[i]，价值是c[i]。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

解法一：

普通01背包解法：

F[i,j] = max {F[i – 1,j],F[i – 1,j –C i ] + W i }

多重背包问题解法

F[i,j] = max {F[i – 1,j],F[i – 1,j –k* C i ] + k*W i }

解法一代码：

```
for (int i = 1; i <= n; i++)
        for (int j = 1; j <=m; j++)
           for(k=0;k<=n[i];k++)
              if (w[i] <= j)
                  f[i][j] = max(f[i-1][j],f[i-1][j-k*w[i]]+ k* c[i]);
              else
                 f[i][j] = f[i-1][j];
```

解法二：

对于存在n[i]件的物品i，可将其转换为**n[i]件01背包**类的物体。

则整体转换为01背包问题。

# 解法一与解法二在时间复杂度上都比较大

# 有没有比较优化的方法?

二进制分组

$n[i]=2^0+2^1+\ldots+2^k+c$

```
for(int i=0;i<=W;i++)
    dp[i]=0;
for(int i=1;i<=n;i++){
    int tot=m[i];
    int now=1;
    while(tot!=0){//二进制分组
        int need=min(tot,now);//当前组有need个物品i
        for(int j=W;j>=need*c[i];j--){
            dp[j]=max(dp[j],dp[j-need*c[i]]+need*v[i]);
        }
        tot-=need;
        now*=2;
    }
}
printf("%d\n",dp[W]);
```

**拓展**：[多重背包问题的二进制优化]

# 小结



01背包的状态转移方程:

```
F[i,v] = max{F[i-1,v], F[i-1,v-Ci] + Wi}
```

完全背包的状态转移方程:

```
F[i,v] = max{F[i-1,v-kCi] + kWi | 0 <= kCi <= v}
```

多重背包的状态转移方程:

```
F[i,v] = max{F[i-1,v-kCi] + kWi | 0 <= k <= Mi}
```

# 小结

- 若每件物品的个数是1，那么就是普通**01背包问题**；
- 若规定每件物品都有多个，那么该问题就是**多重背包问题**；
- 若每件物品的个数是无限个（或者都超过v/w[i])则是**完全背包问题**；
- 若三种情况都有，则是**混合背包问题**。

都以**01背包**为基础实现

```
for (int i = 1; i <= n; i++){
    if(第i件物品为01背包物体)
        使用01背包遍历方式
    if(第i件物品为完全背包物体)
        使用完全背包遍历方式
    if(第i件物品为多重背包物体)
        使用多重背包遍历方式
}
```

# 4.二维背包

有N种物品和一个**容量为V**的背包，背包还**最多只能承受W的重量**。第i种物品最多有n[i]件可用，每件体积是v[i]，重量是w[i]，价值是c[i]。求解将哪些物品装入背包可使这些物品的体积和重量不超过背包容量，且价值总和最大。

# 二维费用背包



**大问题：**

求最多装M重量V体积的最大价值

**状态确定：**

dp[i][v][w] 表示 **前i个物品，**
**恰放入容量为v载重为w的背包，可以获得的最大价值**

**状态转移：**

$dp[i][v][w] = \max\{dp[i-1][v][w],\ dp[i-1][v-C_i][w-W_i]+C_i\}$ ($v>=C_i$且$w>=W_i$)

不放第i件物品　　放第i件物品

**边界条件：**

dp[0][][] = 0

# 5.分组背包

# 分组背包



【问题描述】

一个旅行者有一个最多能用V公斤的背包，现在有n件物品，它们的重量分别是W1，W2，...,Wn，它们的价值分别为C1,C2,...,Cn。这些物品被划分为若干组，每组中的物品互相冲突，最多选一件。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

【输入格式】

第一行：三个整数，V(背包容量，V<=200)，N(物品数量，N<=30)和T(最大组号，T<=10)；

第2..N+1行：每行三个整数Wi,Ci,P，表示每个物品的重量，价值，所属组号。

【输出格式】

仅一行，一个数，表示最大总价值。

【样例输入】

```
10 6 3
2 1 1
3 3 1
4 8 2
6 9 2
2 8 3
3 9 3
```

【样例输出】

```
20
```

这个问题变成了每组物品有若干种策略：是选择本组的某一件，还是一件都不选。也就是说设 F[i,j] 表示前 i 组物品花费费用 j 能取得的最大权值，则有：

$$F[i,j] = \max\{F[i-1,j], F[i-1,j-C_k]+W_k\}\ k \in i组$$

```
for i = 1 to T
  for j = V to 0
    for (k为第i组中物品)
      F[v] = max {F[v],F[v − C k ] + W k }
```

# 6.简单依赖背包

# 金明的预算方案



金明今天很开心，家里购置的新房就要领钥匙了，新房里有一间金明自己专用的很宽敞的房间。更让他高兴的是，妈妈昨天对他说："你的房间需要购买哪些物品，怎么布置，你说了算，只要不超过N元钱就行"。今天一早，金明就开始做预算了，他把想买的物品分为两类：主件与附件，附件是从属于某个主件的，下表就是一些主件与附件的例子：

| 主件 | 附件 |
|---|---|
| 电脑 | 打印机，扫描仪 |
| 书柜 | 图书 |
| 书桌 | 台灯，文具 |
| 工作椅 | 无 |

如果要买归类为附件的物品，必须先买该附件所属的主件。每个主件可以有0个、1个或2个附件。附件不再有从属于自己的附件。金明想买的东西很多，肯定会超过妈妈限定的N元。于是，他把每件物品规定了一个重要度，分为5等：用整数1~5表示，第5等最重要。他还从因特网上查到了每件物品的价格（都是10元的整数倍）。他希望在不超过N元（可以等于N元）的前提下，使每件物品的价格与重要度的乘积的总和最大。

设第j件物品的价格为v[j]，重要度为w[j]，共选中了k件物品，编号依次为j1，j2，……，jk，则所求的总和为：

v[j1]*w[j1]+v[j2]*w[j2]+ …+v[jk]*w[jk]。（其中*为乘号）

请你帮助金明设计一个满足要求的购物单。

# 金明的预算方案

【输入文件】

输入文件budget.in 的第1行，为两个正整数，用一个空格隔开：

N m （其中N（<32000）表示总钱数，m（<60）为希望购买物品的个数。）

从第2行到第m+1行，第j行给出了编号为j-1的物品的基本数据，每行有3个非负整数

v p q

（其中v表示该物品的价格（v<10000），p表示该物品的重要度（1~5），q表示该物品是主件还是附件。如果q=0，表示该物品为主件，如果q>0，表示该物品为附件，q是所属主件的编号）

【输出文件】

输出文件budget.out只有一个正整数，为不超过总钱数的物品的价格与重要度乘积的总和的最大值（<200000）。

【输入样例】

1000 5
800 2 0
400 5 1
300 5 1
400 3 0
500 2 0

【输出样例】

2200

# 金明的预算方案



| 主件 | 附件 |
| --- | --- |
| 电脑 | 打印机，扫描仪 |

有以下几种可能：

1）买电脑
2）买电脑+打印机
3）买电脑+扫描仪
4）买电脑+打印机+扫描仪

转换为四件物品
但这四件物品最
多只能选择一件

**分组背包问题**

# Thanks

## For Your Watching

FZOI

西|大|附|中|信|息|学|竞|赛

High School Affiliated to Southwest University