浅谈几类背包题

浙江省温州中学　　徐持衡

指导老师：舒春平

2008 年 12 月

目录

摘要

背包问题作为一个经典问题在动态规划中是很基础的一个部分，然而以 0-1 背包问题为原题，衍生转变出的各类题目，可以说是千变万化，当然解法也各有不同，如此就有了继续探究的价值。

本文就 4 道背包变化的题做一些探讨研究，提出本人的一些做法，希望能起到抛砖引玉的作用。

关键字

动态规划 背包 优化

正文

一、 引言

背包问题是运筹学中的一个经典的优化难题，是一个 NP-完全问题，但其有着广泛的实际应用背景，是从生活中一个常见的问题出发展开的：

一个背包，和很多件物品，要在背包中放一些物品，以达到一定的目标。

在信息学中，把所有的数据都量化处理后，得到这样的一个问题：

0-1 背包问题：给定 n 件物品和一个背包。物品 i 的价值是 W_i ，其体积为 V_i，背包的容量为 C。可以任意选择装入背包中的物品，求装入背包中物品的最大总价值。

在选择装入背包的物品时，对每件物品 i ，要么装入背包，要么不装入背包。不能将物品 i 多次装入背包，也不能只装入部分物品 i （分割物品 i）。因此，该问题称为 0-1 背包问题。

用于求解 0-1 背包问题的方法主要有回溯算法、贪婪算法、遗传算法、禁忌搜索算法、模拟退火算法等。

在高中阶段，我们所谓的经典 0-1 背包问题，保证了所有量化后的数据均为正整数，即是一个特殊的整数规划问题，本文中如无特殊说明均以此为前提。其经典的 O(n*C)动规解法是：

状态是在前 i 件物品中，选取若干件物品其体积总和不大于 j，所能获得的最大价值为 $F_i[j]$，当前的决策是第 i 件物品放或者不放，最终得到转移方程：

$F_i[j] = F_{i-1}[j]$ $(V_i>j>=0)$

$F_i[j] = \max\{ F_{i-1}[j] , F_{i-1}[j-V_i]+W_i \}$ $(C>=j>=V_i)$

其中由于 F_i 只与 F_{i-1} 有关，可以用滚动数组来节省程序的空间复杂度。以下就是经典算法的伪代码。

```
1.  FOR i: = 1 TO n
2.   FOR j: = C DOWNTO Vi
3.    Max ( F[ j ] , F[ j-Vi ]  +  Wi ) → F[ j ]
4.   END FOR
5.  END FOR
```

二、 背包的基本变换

① 完全背包

完全背包问题：给定 n 种物品和一个背包。第 i 种物品的价值是 W_i ，其体积为 V_i，背包的容量为 C，同一种物品的数量无限多。可以任意选择装入背包中的物品，求装入背包中物品的最大总价值。

这个问题完全可以转化为 0-1 背包问题来解决，即把第 i 种物品分割成 (C div V_i)件物品，再用 0-1 背包问题的经典动规实现。

但是，这个算法的时间复杂度太高，并不能作为一种实用的方法来实现。

很容易注意到，这个问题对于 0-1 背包来说，是另一个极端，每种物品都可以无限制地取，只要改变转移方程，就可以构造出新的算法：

状态是在前 i 种物品中，选取若干件物品其体积总和不大于 j，所能获得的最大价值为 $F_i[j]$，当前的决策是第 i 件物品放（多件）或者不放，转移方程是

$F_i[j] = F_{i-1}[j]$ $(V_i>j>=0)$

$F_i[j] = \max\{ F_{i-1}[j] , F_i[j-V_i]+W_i \}$ $(C>=j>=V_i)$ //注意第二个是 F_i 而不是 F_{i-1}，与 0-1 背包区别仅在于此，因为允许在放过的基础上再增加一件。

这样，这个问题就有了与 0-1 背包一样时间复杂度 O(n*C)的解决方法，同样的可以用滚动数组来实现。

```
1. FOR i: = 1 TO n
2.  FOR j: = Vi TO C
3.   Max ( F[ j ] , F[ j-Vi ]  +  Wi ) → F[ j ]
4.  END FOR
5. END FOR
```

② 多次背包

多次背包问题：给定 n 种物品和一个背包。第 i 种物品 的价值是 W_i ，其体积为 V_i，数量是 K_i 件，背包的容量为 C。可以任意选择装入背包中的物品，求装入背包中物品的最大总价值。

和完全背包一样，可以直接套用 0-1 背包问题的经典动规实现，但是效率太低了，需要寻找更高效的算法。

首先对于第 i 种物品，不能确定放多少件才是最优的，因为并没有什么可以证明放一件或者全放一定会更优。换句话说，最优解所需要的件数，可能是 0 到 K_i 中的任何数。

在日常生活中，如果需要能拿得出 1 到 K_i 的任意整数数额的钱，往往

不会带 K_i 个一元钱，因为那实在是太不方便了，取而代之的是带一些 1 元和其他一些面值各不相同的非 1 数额的钱。

这种思想完全可以运用到这道题上！

不需要把一种物品拆分成 K_i 份，而是只要物品拆分到能凑出 1 到 K_i 之间任意数量的程度就可以了。

可以证明，按照二进制的拆分能使件数达到最小，把 K_i 拆分成 $1,2,4\ldots2^t,K_i-2^{t+1}+1$（$2^{t+2}>K_i>=2^{t+1}$），就一定可以满足最优解要求了。下一步，还是用 0-1 背包的经典算法。

如此，我们得到了一个时间复杂度为 $O(C*\Sigma([\log_2 K_i]))$的算法。

```
1. FOR i: = 1 TO n
2.  1 → m
3.  WHILE Ki > 0
4.   IF m > Ki THEN Ki → m
5.   Ki - m → Ki
6.   FOR j: = C DOWNTO Vi * m
7.    Max( F[ j ] , F[ j-Vi*m ]  +  Wi * m ) → F[ j ]
8.   END FOR
9.   m * 2 → m
10. WEND
11. END FOR
```

此算法还能加上一个优化，判断如果 K_i 大于 C/V_i，则多出的部分是没有意义的，可以舍去。时间复杂度就可以优化到 $O(n*C*\log_2 C)$。由于在高中阶段碰到的题中 C 的值很有限，所以这个算法在实际应用上的效果已经可能满足一般的需求了。

但是在下一节中，有一个更高效的解决方法。

③ 单调队列优化☆

长度限制最大连续和问题：给出长度为 n 的序列 X_i，求这个序列中长度不超过 l_{max} 的最大连续和。

首先考虑最简单的做法，就是直接用 $O(n*l_{max})$的二次循环求最大值：

```
1. FOR i: = 1 TO n
2.  0 → s
```

```
3.  FOR j: = i DOWNTO Max( i-l_max+1 ,1)
4.   s + X_j → s
5.   IF s > ans THEN s → ans
6.  END FOR
7. END FOR
```

用 S_i 记 X_1 到 X_i 的总和，就可以看到，如果确定一个端点后，要做的就是在 S 数组的一个连续区间取一个最值，区间最值问题完全可以用线段树来实现。

但是这个题目的另一个特性是区间的长度是固定的，而且每个区间都只需要取一次，所以我们可以用更简单的数据结构来实现——单调队列。

先来研究一下单调队列，以维护最大值为例，在满足序列中的编号递增以后，还要满足元素的值的递减。

	L	L+1	……	R-1	R
编号	A_L	A_{L+1}	……	A_{R-1}	A_R
值	B_L	B_{L+1}	……	B_{R-1}	B_R

满足 $A_{i+1}>A_i>A_{i-1}$ and $B_{i-1}>B_i>B_{i+1}$ $(R>i>L)$。

单调队列除队列首元素出队列外，还需要用一定的操作来维护队列的特殊性质。如果进入了一个新的元素(a,b)，其中 a 必然大于 A_R，但是 b 可能会大于等于 B_R。既然 b 大于等于 B_R，而元素 R 又是要先于新元素出队列，那么元素 R 就已经失去价值，因为接下来新元素必然都会比元素 R 更优。所以现在就可以删除元素 R 了。

重复上面的步骤，直到前面没有元素或者满足 $B_R>b$ 为止，再让新元素进队列。显然，当前的队列首元素，必定是这个区间的最大值。

```
1. PROCEDURE INSERT a , b
2.  WHILE R >= L AND b > B[ R ] DO R - 1 → R
3.  R + 1 → R
4.  a → A[ R ]
5.  b → B[ R ]
6. END
```

如此完成的单调队列，虽然不能保证每一次的操作是 O(1)，但是因为每个元素只进队列一次，并出队列一次，所以总效率是 O(n)。

当然，这道题其实就是单调队列的基本功能，而我们希望的是能把它

用来优化背包问题，所以现在重新考虑 多次背包问题 。

对于第 i 种物品来说，已知体积 v，价值 w，数量 k，那么可以按照当前枚举的体积 j 对 v 的余数把整个动规数组分成 v 份，以下是 v=3 的情况：

j	0	1	2	3	4	5	6	7	8	……
j mod v	0	1	2	0	1	2	0	1	2	……

我们可以把每一份分开处理，假设余数为 d。

编号 j	0	1	2	3	4	5	……
对应体积	d	d+v	d+2*v	d+3*v	d+4*v	d+5*v	……

现在看到分组以后，编号 j 可以从 j-k 到 j-1 中的任意一个编号转移而来（因为相邻的体积正好相差 v），这看上去已经和区间最大值有点相似了。但是注意到由于体积不一样，显然体积大的价值也会大于等于体积小的，直接比较是没有意义的，所以还需要把价值修正到同一体积的基础上。比如都退化到 d，也就是说用 F[j*v+d]- j*w 来代替原来的价值进入队列。

对于第 i 件物品，转移伪代码：

```
1. FOR d: = 0 TO v-1                          //枚举余数，分开处理
2.  清空队列
3.  FOR j: = 0 TO (C-d) div v                 //j 枚举标号，对应体积为 j*v+d
4.   INSERT j , F[ j*v+d ] – j * w            //插入队列
5.   IF A[ L ] < j - k THEN L + 1 → L         //如果队列的首元素已经失效
6.   B[ L ] + j * w → F[ j*v+d ]              //取队列头更新
7.  END FOR
8. END FOR
```

已知单调队列的效率是 O(n)，那么加上单调队列优化以后的多次背包，效率就是 O(n*C)了。

三、 其他几类背包问题

① 树形依赖背包（选课）☆

树形依赖背包问题：给定 n 件物品和一个背包。第 i 件物品 的价值是 W_i ，其体积为 V_i，但是依赖于第 X_i 件物品（必须选取 X_i 后才能取 i，如果无依赖则 X_i=0），依赖关系形成森林，背包的容量为 C。可以任意选择装入背包中的物品，求装入背包中物品的最大总价值。

这道题需要在 treedp 的基础上用背包实现。

[1]泛化物品——定义：

考虑这样一种物品，它并没有固定的费用（体积）和价值，而是它的价值随着你分配给它的费用（体积）变化而变化。

泛化物品可以用一个一维数组来表示体积与价值的关系 G_j 表示当体积为 j 的时候，相对应的价值为 $G_j(C>=j>=0)$。

显然，之前的背包动规数组 F_i，就是一件泛化物品，因为 $F_i[j]$表示的正是体积为 j 的时候的最大价值。同样的，多件物品也是可以合并成一件泛化物品。

泛化物品的和：

把两个泛化物品合并成一个泛化物品的运算，就是枚举体积分配给两个泛化物品，满足：

$G[j] = \max\{ G_1[j-k] + G_2[k] \}$ $(C>=j>=k>=0)$

把两个泛化物品合并的时间复杂度是 O(C^2)。

对于具有树形依赖关系的背包问题，我们可以把每棵子树看作是一个泛化物品，那么一棵子树的泛化物品就是子树根节点的这件物品的泛化物品与由根所连的所有子树的泛化物品的和。

整个动规过程就是从叶子进行到根，对于每一棵子树的操作就是：

```
1. PROCEDURE DEAL i , v , w      //第 i 个节点 体积为 v 价值为 w
2.  FOR j: = v TO C              //初始化这件泛化物品
3.   w → Fi[ j ]
4.  END FOR
5.  FOR s: = 1 TO n
6.    IF s 是 i 的儿子 THEN
7.     Fi 与 Fs 的和 → Fi
8.    END IF
9.  END FOR
10. END
```

一次只能把两个泛化物品合并，那么要把 n 个泛化物品合并成一个就需要 n-1 次合并，所以这个算法效率是 $O(n*C^2+n^2)$ ，当然，其中的 $O(n^2)$ 可以用邻接表的记边方法变成 O(n)，最终的效率就是 $O(n*C^2)$。

[1] “泛化物品”一词最初在 DDengi 的《背包九讲》中出现，下文中的“泛化物品的和”也引自《背包九讲》

回顾经典 0-1 背包问题，在那个经典算法中，求泛化物品与一件物品的和，只需要 O(C)的时间复杂度，推论出：

泛化物品与一件物品的和：

把一个泛化物品与一件物品合并成一个泛化物品，可以用类似于 0-1 背包经典动规的方法求出。

$G[j] = G_1[j]$ $(v>j>=0)$

$G[j] = \max\{ G_1[j] , G_1[j-v]+w \}$ $(C>=j>=v)$

这样的合并，时间复杂度仅为 O(C)，同样也是合并了一件物品，效率比求两件泛化物品的和快很多。那么有没有办法用这种 O(C)的合并方式来代替计算两个泛化物品的和来处理这道题呢？

泛化物品的并：

因为两个泛化物品之间存在交集，所以不能同时两者都取，那么我们就需要求泛化物品的并，对同一体积，我们需要选取两者中价值较大的一者，效率 O(C)。

$G[j] = \max\{ G_1[j] , G_2[j] \}$ $(C>=j>=0)$

重新考虑对以 i 为根的子树的处理，假设当前需要处理 i 的一个儿子 s。

如果我们在当前的 F_i 中强制放入物品 s 后作为以 s 为根的子树的初始状态的话，那么处理完以 s 为根的子树以后，F_s 就是与 F_i 有交集的泛化物品（实际上是 F_s 包含 F_i），同时，F_s 必须满足放了物品 s，即 $F_s[j]$ $(V_s>j>=0)$ 已经无意义了，而 $F_s[j]$$(C>=j>=V_s)$必然包含物品 s。为了方便，经过处理以后，在程序中规定只有 $F_s[j]$$(C-V_s>=j>=0)$是合法的。

接下来只需要把 F_s 与 F_i 的并赋给 F_i，就完成了对一个儿子的处理。如此，我们需要的总时间复杂度仅为 O(n*C)。

```
1. PROCEDURE DEAL i , C
2. FOR s: = 1 TO n
3.    IF s 是 i 的儿子 THEN
4.     Fi → Fs
5.     DEAL s , C - Vs            //背包容量减小 Vs
6.     FOR K: =Vs TO C            //求两者的并
7.      Max ( Fi[ k ] , Fs[ k-Vs ] + Ws ) → Fi[ k ]
8.     END FOR
9.    END IF
10. END FOR
11. END
```

[2]用这个算法，可以把《选课》这一类题优化到 O(n*C)，亦可以作为 noip06《金明的预算方案》的 O(n*C) treedp 解法。

② PKU3093☆

PKU3093：给定 n 件物品和一个背包，第 i 件物品的体积为 V_i，背包的容量为 C。要求把一些物品放入背包使得剩下的物品都放不下，求方案数。

暂时先不考虑“使剩下的物品都放不下”的条件，那就是求 0-1 背包的所有可行方案。

用 $F_i[j]$表示前 i 件物品中选若干件总体积为 j 的方案数，初始为 $F_0[0]=1$，转移方程是：

$F_i[j] = F_{i-1}[j]$ (Vi>j)

$F_i[j] = F_{i-1}[j] + F_{i-1}[j-Vi]$(j>=Vi)

显然这个算法的效率是 O(n*C)的，它计算了所有装放背包的方案数。

现在考虑“使剩下的物品都放不进去”的条件，如果剩下的物品中体积最小为 v，那么方案数就是 sum{ $F_n[j]$ }(C>=j>C-v)。前提是我们事先确定了剩下中体积最小的是哪个。

对体积排序后，下一步就是枚举 i 作为剩余物品中体积最小的一件。对于所有 s<i 的物品必须都要放入背包，对于 i 则不能放入背包，对于 s>i 的物品做 0-1 背包可行方案的统计，将 sum{ $F_n[j]$ }(C>=j>C-V_i)累加到 ans。

由于每次都需要对 n-i 件物品做统计，一共统计 n 次，效率是 $O(n^2*C)$。

```
1. 0 → sum                          //sum 记 1 到 i-1 的物品体积总和
2. FOR i: = 1 TO N
3.  F 数组清零
4.  1 → F[ sum ]                     //初始化
5.  FOR s: = i + 1 TO N             //统计可行方案数
6.   FOR j: = C DOWNTO Vs + sum
7.    F[ j ] + F[ j-Vs ] → F[ j ]
8.   END FOR
9.  END FOR
10. FOR k: = C DOWNTO C - Vi + 1    //累加总方案数
11.  IF k >= sum THEN ans + F[ k ] → ans
```

[2] 在附录中有两道原题以及我的程序。在论文完成之后，经 TKY 提醒发现，同样效率的算法已经在 07 年的《浅谈数据的合理组织》（何森）中被提及，所以这个算法只能算是另一种更简单的 O(n*C)实现方法。

```
12. END FOR
13. sum + Vi → sum
14. END FOR
```

可以发现，同一个物品多次被考虑放入背包，这样会造成时间的浪费。观察得到，第 i 件物品共考虑了 i-1 次。每一次循环都会少一件物品。如果把整个过程倒置，每件物品是否可以只考虑一次呢？

由于初始状态不一样，我们还需要把初始状态统一。可以让每次 F[0]=1，总容量为 C-sum。

但是只统一初始化状态还不够，因为每次的背包容量还是不同的，做背包统计的时候，背包容量不可以是一个变值，也必须要统一，所以每次考虑一件物品都要用最大容量 C 来更新背包。

一次操作之后要将 sum{ F[j] }(C-sum>=j>C-sum-V_i , j>=0)累加到 ans。

现在，每件物品都只考虑一次，背包体积统一是 C，那么效率就变成了 O(n*C)。

```
1. 0 → sum
2. 1 → F[ 0 ]
3. FOR i: = 1 TO n
4.  sum + Vi → sum
5. END FOR
6. FOR i: = n DOWNTO 1
7.  sum - Vi → sum
8.  FOR j: = C - sum DOWNTO  Max( C – sum – Vi + 1 , 0 )//累加总方案数
9.   ans + F[ k ] → ans
10. END FOR
11. FOR j: = C DOWNTO Vi            //考虑第 i 件物品放入背包
12.  F[ j ] + F[ j – Vi ] → F[ j ]
13. END FOR
14. END FOR
```

四、 总结

回顾全文的四道背包题：对于完全背包问题，用转化方程就解决了；对于多次背包，使用了单调队列优化来实现 O(n*C)的效率；在树形依赖背包问题中，

探索了新的概念，最终完成算法的转化；而在 PKU3093 一题中，通过合并相似操作来达到优化的效果。

虽然用到的方法各不相同，每个方法都不仅仅限于背包问题，完全可以灵活运用到其他问题中。

凡是文中提到的问题最后都用时间复杂度 O(n*C)的算法解决了，这并不是说所有背包题都可以优化到这个程度，但是，可以肯定的是不会有比这个更快的效率了。

就目前来说，背包类的题目还有很多没有得到很好的解决，等待着大家去继续探索研究。

附录

参考文献：

《背包九讲》——DDengi，ZJU

文中原题

来源：ctsc97

1．选课

大学里实行学分。每门课程都有一定的学分，学生只要选修了这门课并考核通过就能获得相应的学分。学生最后的学分是他选修的各门课的学分的总和。

每个学生都要选择规定数量的课程。其中有些课程可以直接选修，有些课程需要一定的基础知识，必须在选了其它的一些课程的基础上才能选修。例如，《数据结构》必须在选修了《高级语言程序设计》之后才能选修。我们称《高级语言程序设计》是《数据结构》的先修课。每门课的直接先修课最多只有一门。两门课也可能存在相同的先修课。为便于表述每门课都有一个课号，课号依次为 1，2，3，……。下面举例说明

课号	先修课号	学分
1	无	1
2	1	1
3	2	3
4	无	3
5	2	4

上例中 1 是 2 的先修课，即如果要选修 2，则 1 必定已被选过。同样，如果要选修 3，

那么 1 和 2 都一定已被选修过。

学生不可能学完大学所开设的所有课程，因此必须在入学时选定自己要学的课程。每个学生可选课程的总数是给定的。现在请你找出一种选课方案，使得你能得到学分最多，并且必须满足先修课优先的原则。假定课程之间不存在时间上的冲突。

输入

输入文件的第一行包括两个正整数 M、N（中间用一个空格隔开）其中 M 表示待选课程总数（1≤M≤1000)，N 表示学生可以选的课程总数（1≤N≤M)。

以下 M 行每行代表一门课，课号依次为 1，2……M。每行有两个数（用一个空格隔开），第一个数为这门课的先修课的课号（若不存在先修课则该项为 0），第二个数为这门课的学分。学分是不超过 10 的正整数。

输出

输出文件第一行只有一个数，即实际所选课程的学分总数。以下 N 行每行有一个数，表示学生所选课程的课号。

输入输出示例

INPUT.TXT

```
7 4
2 2
0 1
0 4
2 1
7 1
7 6
2 2
```

OUTPUT.TXT

```
13
2
6
7
3
```

《选课》的源代码

```
1. var
2. n,C,i:longint;
3. x,w:array[1..400]of longint;
4. f:array[0..400,0..400]of longint;
5.
6. procedure dfs(k,C:longint);
7. var
8. i,j:longint;
9. begin
10.    if C<=0 then exit;
11.    for i:=1 to n do if x[i]=k then begin
12.        for j:=0 to C-1 do f[i,j]:=f[k,j]+w[i];//强制放入物品 i
13.        dfs(i,C-1);
14.        for j:=1 to C do
15.            if f[i,j-1]>f[k,j] then
16.                f[k,j]:=f[i,j-1];             //求两者的并
17.    end;
18. end;
19.
20. begin
21.    readln(n,C);
22.    for i:=1 to n do begin
23.        readln(x[i],w[i]); //读入父节点标号 x[i] 和学分 w[i]
24.    end;
25.    dfs(0,C);             //以 0 作为所有没有父节点的点的父亲
26.    writeln(f[0,C]);
27. end.
```

来源：NOIP2006 第二题

2．金明的预算方案

(budget.pas/c/cpp)

【问题描述】

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 j_1，j_2，……，j_k，则所求的总和为：

$v[j_1]*w[j_1]+v[j_2]*w[j_2]+ \cdots+v[j_k]*w[j_k]$。（其中*为乘号）

请你帮助金明设计一个满足要求的购物单。

【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

```
1. var
2. n,C,i:longint;
3. x,w,v:array[1..60]of longint;
4. f:array[0..60,0..3200]of longint;
5. procedure dfs(k,C:longint);
6. var i,j:longint;
7. begin
8.     if C<=0 then exit;
9.     for i:=1 to n do if x[i]=k then begin
10.        for j:=0 to C-v[i] do f[i,j]:=f[k,j]+w[i]; //强制放入物品 j
11.        dfs(i,C-v[i]);
12.        for j:=v[i] to C do
13.            if f[i,j-v[i]]>f[k,j] thenf[k,j]:=f[i,j-v[i]];//求两者的并
14.    end;
15. end;
16. begin
17.    assign(input,'budget.in');reset(input);
18.    assign(output,'budget.out');rewrite(output);
19.    readln(C,n);
20.    c:=c div 10;
21.    for i:=1 to n do begin
22.        readln(v[i],w[i],x[i]); //读入费 v[i] 重要度 w[i] 父节点标号 x[i]
23.        w[i]:=w[i]*v[i];
24.        v[i]:=v[i] div 10;
25.    end;
26.    dfs(0,C);              //以 0 作为所有没有父节点的点的父亲
27.    writeln(f[0,C]);
28.    close(output);
29. end.
```

来源：Pku 3093

Margaritas on the River Walk

Time Limit: 1000MS Memory Limit: 65536K
Total Submissions: 694 Accepted: 271

Description

One of the more popular activities in San Antonio is to enjoy margaritas in the park along the river know as the *River Walk*. Margaritas may be purchased at many establishments along the River Walk from fancy hotels to *Joe's Taco and Margarita* stand. (The problem is not to find out how Joe got a liquor license. That involves Texas politics and thus is much too difficult for an ACM contest problem.) The prices of the margaritas vary depending on the amount and quality of the ingredients and the ambience of the establishment. You have allocated a certain amount of money to sampling different margaritas.

Given the price of a single margarita (including applicable taxes and gratuities) at each of the various establishments and the amount allocated to sampling the margaritas, find out how many different maximal combinations, choosing at most one margarita from each establishment, you can purchase. A valid combination must have a total price no more than the allocated amount and the unused amount (*allocated amount – total price*) must be less than the price of any establishment that was not selected. (Otherwise you could add that establishment to the combination.)

For example, suppose you have $25 to spend and the prices (whole dollar amounts) are:

Vendor	A	B	C	D	H	J
Price	8	9	8	7	16	5

Then possible combinations (with their prices) are:

ABC(25), ABD(24), ABJ(22), ACD(23), ACJ(21), ADJ(20), AH(24), BCD(24), BCJ(22), BDJ(21), BH(25), CDJ(20), CH(24), DH(23) and HJ(21).

Thus the total number of combinations is 15.

Input

The input begins with a line containing an integer value specifying the number of datasets that follow, N ($1 \leq N \leq 1000$). Each dataset starts with a line containing two

integer values ***V*** and ***D*** representing the number of vendors ($1 \leq V \leq 30$) and the dollar amount to spend ($1 \leq D \leq 1000$) respectively. The two values will be separated by one or more spaces. The remainder of each dataset consists of one or more lines, each containing one or more integer values representing the cost of a margarita for each vendor. There will be a total of ***V*** cost values specified. The cost of a margarita is always at least one (1). Input values will be chosen so the result will fit in a 32 bit unsigned integer.

Output

For each problem instance, the output will be a single line containing the dataset number, followed by a single space and then the number of combinations for that problem instance.

Sample Input

```
2
6 25
8 9 8 7 16 5
30 250
1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
```

Sample Output

```
1 15
2 16509438
```

Hint

Note: Some solution methods for this problem may be exponential in the number of vendors. For these methods, the time limit may be exceeded on problem instances with a large number of vendors such as the second example below.

Source

Greater New York 2006