首先01背包题目的雏形是

有N件物品和一个容量为V的背包。第i件物品的费用是c[i]，价值是w[i]。求解将哪些物品装入背包可使价值总和最大。

从这个题目中可以看出，01背包的特点就是：每种物品仅有一件，可以选择放或不放。

其状态转移方程是：

f[i][v]=max{f[i-1][v],f[i-1][v-c[i]]+w[i]}

对于这方方程其实并不难理解，方程之中，现在需要放置的是第i件物品，这件物品的体积是c[i],价值是w[i],因此f[i-1][v]代表的就是不将这件物品放入背包，而f[i-1][v-c[i]]+w[i]则是代表将第i件放入背包之后的总价值，比较两者的价值，得出最大的价值存入现在的背包之中。

### 01背包的状态转换方程 f[i,j] = Max{ f[i-1,j-Wi]+Pi( j >= Wi ),  f[i-1,j] }

f[i,j]表示在前i件物品中选择若干件放在承重为 j 的背包中，可以取得的最大价值。

Pi表示第i件物品的价值。

决策：为了背包中物品总价值最大化，第 i件物品应该放入背包中吗 ？

**题目描述：**

有编号分别为a,b,c,d,e的五件物品，它们的重量分别是2,2,6,5,4，它们的价值分别是6,3,5,4,6，现在给你个承重为10的背包，如何让背包里装入的物品具有最大的价值总和？

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| name | weight | value | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| a | 2 | 6 | 0 | 6 | 6 | 9 | 9 | 12 | 12 | 15 | 15 | 15 |
| b | 2 | 3 | 0 | 3 | 3 | 6 | 6 | 9 | 9 | 9 | 10 | 11 |
| c | 6 | 5 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 10 | 11 |
| d | 5 | 4 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 10 | 10 |
| e | 4 | 6 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |

只要你能通过找规律手工填写出上面这张表就算理解了01背包的动态规划算法。

首先要明确这张表是至底向上，从左到右生成的。

# [经典背包问题 01背包+完全背包+多重背包](http://blog.csdn.net/lyhvoyage/article/details/8545852)

01 背包

有n 种不同的物品，每个物品有两个属性，size 体积，value 价值，现在给一个容量为 w 的背包，问最多可带走多少价值的物品。

1. **int** f[w+1];   //f[x] 表示背包容量为x 时的最大价值
2. **for** (**int** i=0; i<n; i++)
3. **for** (**int** j=w; j>=size[i]; j--)
4. f[j] = max(f[j], f[j-size[i]]+value[i]);

完全背包

如果物品不计件数，就是每个物品不只一件的话，稍微改下即可

1. **for** (**int** i=0; i<n; i++)
2. **for** (**int** j=size[i]; j<=w; j++)
3. f[j] = max(f[j], f[j-size[i]]+value[i]);

        f[w] 即为所求    
        初始化分两种情况：  
        1、如果背包要求正好装满则初始化 f[0] = 0, f[1~w] = -INF;

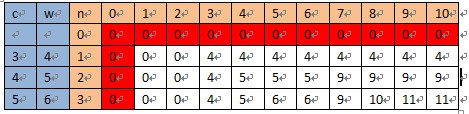
        2、如果不需要正好装满 f[0~v] = 0;

        举例：

01背包

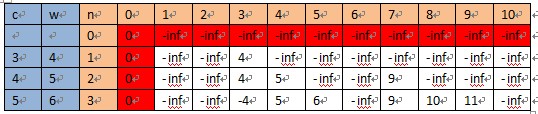
V=10，N=3，c[]={3,4,5}, w={4,5,6}

（1）背包不一定装满

      计算顺序是：从右往左，自上而下：因为每个物品只能放一次，前面的体积小的会影响体积大的

（2）背包刚好装满

      计算顺序是：从右往左，自上而下。注意初始值，其中-inf表示负无穷



完全背包：

V=10，N=3，c[]={3,4,5}, w={4,5,6}

（1）背包不一定装满

计算顺序是：从左往右，自上而下：  每个物品可以放多次，前面的会影响后面的



（2）背包刚好装满

计算顺序是：从左往右，自上而下。注意初始值，其中-inf表示负无穷

[](http://dongxicheng.org/wp-content/uploads/2011/08/complete-knapsack-full.jpg)

多重背包：    
         多重背包问题要求很简单，就是每件物品给出确定的件数，求可得到的最大价值    
         多重背包转换成 01 背包问题就是多了个初始化，把它的件数C 用二进制分解成若干个件数的集合，这里面数字可以组合成任意小于等于C的件数，而且不会重复，之所以叫二进制分解，是因为这样分解可以用数字的二进制形式来解释    
       比如：7的二进制 7 = 111 它可以分解成 001 010 100 这三个数可以组合成任意小于等于7 的数，而且每种组合都会得到不同的数    
       15 = 1111 可分解成 0001  0010  0100  1000 四个数字    
        如果13 = 1101 则分解为 0001 0010 0100 0110 前三个数字可以组合成  7以内任意一个数，即1、2、4可以组合为1——7内所有的数，加上 0110 = 6 可以组合成任意一个大于6 小于等于13的数，比如12，可以让前面贡献6且后面也贡献6就行了。虽然有重复但总是能把 13 以内所有的数都考虑到了，基于这种思想去把多件物品转换为，多种一件物品，就可用01 背包求解了。    
       看代码：  

1. **int** n;  //输入有多少种物品
2. **int** c;  //每种物品有多少件
3. **int** v;  //每种物品的价值
4. **int** s;  //每种物品的尺寸
5. **int** count = 0; //分解后可得到多少种物品
6. **int** value[MAX]; //用来保存分解后的物品价值
7. **int** size[MAX];  //用来保存分解后物品体积
9. scanf("%d", &n);    //先输入有多少种物品，接下来对每种物品进行分解
11. **while** (n--)     //接下来输入n中这个物品
12. {
13. scanf("%d%d%d", &c, &s, &v);  //输入每种物品的数目和价值
14. **for** (**int** k=1; k<=c; k<<=1)   //<<右移 相当于乘二
15. {
16. value[count] = k\*v;
17. size[count++] = k\*s;
18. c -= k;
19. }
20. **if** (c > 0)
21. {
22. value[count] = c\*v;
23. size[count++] = c\*s;
24. }
25. }

定理：一个正整数n可以被分解成1,2,4,…,2^(k-1),n-2^k+1（k是满足n-2^k+1>0的最大整数）的形式，且1～n之内的所有整数均可以唯一表示成1,2,4,…,2^(k-1),n-2^k+1中某几个数的和的形式。

证明如下：

（1） 数列1,2,4,…,2^(k-1),n-2^k+1中所有元素的和为n，所以若干元素的和的范围为：[1, n]；

（2）如果正整数t<= 2^k – 1,则t一定能用1,2,4,…,2^(k-1)中某几个数的和表示，这个很容易证明：我们把t的二进制表示写出来，很明显，t可以表示成n=a0\*2^0+a1\*2^1+…+ak\*2^（k-1），其中ak=0或者1，表示t的第ak位二进制数为0或者1.

（3）如果t>=2^k,设s=n-2^k+1，则t-s<=2^k-1，因而t-s可以表示成1,2,4,…,2^(k-1)中某几个数的和的形式，进而t可以表示成1,2,4,…,2^(k-1)，s中某几个数的和（加数中一定含有s）的形式。

（证毕！）

        现在用count 代替 n 就和01 背包问题完全一样了

杭电2191题解：此为多重背包用01和完全背包：

1. #include<stdio.h>
2. #include<string.h>
3. **int** dp[102];
4. **int** p[102],h[102],c[102];
5. **int** n,m;
6. **void** comback(**int** v,**int** w)//经费，重量。完全背包；
7. {
8. **for**(**int** i=v; i<=n; i++)
9. **if**(dp[i]<dp[i-v]+w)
10. dp[i]=dp[i-v]+w;
11. }
12. **void** oneback(**int** v,**int** w)//经费，重量；01背包；
13. {
14. **for**(**int** i=n; i>=v; i--)
15. **if**(dp[i]<dp[i-v]+w)
16. dp[i]=dp[i-v]+w;
17. }
18. **int** main()
19. {
20. **int** ncase,i,j,k;
21. scanf("%d",&ncase);
22. **while**(ncase--)
23. {
24. memset(dp,0,**sizeof**(dp));
25. scanf("%d%d",&n,&m);//经费，种类；
26. **for**(i=1; i<=m; i++)
27. {
28. scanf("%d%d%d",&p[i],&h[i],&c[i]);//价值，重量，数量;
29. **if**(p[i]\*c[i]>=n) comback(p[i],h[i]);
30. **else**
31. {
32. **for**(j=1; j<c[i]; j<<1)
33. {
34. oneback(j\*p[i],j\*h[i]);
35. c[i]=c[i]-j;
36. }
37. oneback(p[i]\*c[i],h[i]\*c[i]);
38. }
39. }
40. printf("%d\n",dp[n]);
41. }
42. **return** 0;
43. }

只是用01背包，用二进制优化：

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** nCase,Limit,nKind,i,j,k,  v[111],w[111],c[111],dp[111];
6. //v[]存价值，w[]存尺寸，c[]存件数
7. //在本题中，价值是米的重量，尺寸是米的价格
8. **int** count,Value[1111],size[1111];
9. //count存储分解完后的物品总数
10. //Value存储分解完后每件物品的价值
11. //size存储分解完后每件物品的尺寸
12. cin>>nCase;
13. **while**(nCase--)
14. {
15. count=0;
16. cin>>Limit>>nKind;
17. **for**(i=0; i<nKind; i++)
18. {
19. cin>>w[i]>>v[i]>>c[i];
20. //对该种类的c[i]件物品进行二进制分解
21. **for**(j=1; j<=c[i]; j<<=1)
22. {
23. //<<左移1位，相当于乘2
24. Value[count]=j\*v[i];
25. size[count++]=j\*w[i];
26. c[i]-=j;
27. }
28. **if**(c[i]>0)
29. {
30. Value[count]=c[i]\*v[i];
31. size[count++]=c[i]\*w[i];
32. }
33. }
34. //经过上面对每一种物品的分解，
35. //现在Value[]存的就是分解后的物品价值
36. //size[]存的就是分解后的物品尺寸
37. //count就相当于原来的n
38. //下面就直接用01背包算法来解
39. memset(dp,0,**sizeof**(dp));
40. **for**(i=0; i<count; i++)
41. **for**(j=Limit; j>=size[i]; j--)
42. **if**(dp[j]<dp[j-size[i]]+Value[i])
43. dp[j]=dp[j-size[i]]+Value[i];
45. cout<<dp[Limit]<<endl;
46. }
47. **return** 0;
48. }

未优化的：

1. #include<iostream>
2. #include<cstdio>
3. #include<cstring>
4. **using** **namespace** std;
6. **int** Value[105];
7. **int** Cost[105];
8. **int** Bag[105];
9. **int** dp[105];
11. **int** main()
12. {
13. **int** C,m,n;
14. scanf("%d",&C);
15. **while**(C--)
16. {
17. scanf("%d%d",&n,&m);
18. **for**(**int** i = 1; i <= m; i++)
19. scanf("%d%d%d",&Cost[i],&Value[i],&Bag[i]);
20. memset(dp,0,**sizeof**(dp));
21. **for**(**int** i=1; i<= m; i++)
22. **for**(**int** j=1; j<=Bag[i]; j++)
23. **for**(**int** k=n; k>=Cost[i]; k--)
24. dp[k]=max(dp[k], dp[k-Cost[i]]+Value[i]);
25. printf("%d\n",dp[n]);
26. }
27. **return** 0;
28. }

# 2546饭卡

**Problem Description**

电子科大本部食堂的饭卡有一种很诡异的设计，即在购买之前判断余额。如果购买一个商品之前，卡上的剩余金额大于或等于5元，就一定可以购买成功（即使购买后卡上余额为负），否则无法购买（即使金额足够）。所以大家都希望尽量使卡上的余额最少。  
某天，食堂中有n种菜出售，每种菜可购买一次。已知每种菜的价格以及卡上的余额，问最少可使卡上的余额为多少。

**Input**

多组数据。对于每组数据：  
第一行为正整数n，表示菜的数量。n<=1000。  
第二行包括n个正整数，表示每种菜的价格。价格不超过50。  
第三行包括一个正整数m，表示卡上的余额。m<=1000。  
n=0表示数据结束。

**Output**

对于每组输入,输出一行,包含一个整数，表示卡上可能的最小余额。

**Sample Input**

1

50

5

10

1 2 3 2 1 1 2 3 2 1

50

0

**Sample Output**

-45

32

代码清单：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

int cmp(int a,int b)

{

return a<b;

}

int dp[1010],val[1010];

int main()

{

int n,m,i,j;

while(scanf("%d",&n),n)

{

for(i=1;i<=n;i++)

scanf("%d",&val[i]);

sort(val+1,val+n+1,cmp);

scanf("%d",&m);

if(m<5)

{

printf("%d\n",m);

continue;

}

memset(dp,0,sizeof(dp));

for(i=1;i<n;i++)

{

for(j=m-5;j>=val[i];j--)

{

dp[j] = max(dp[j],dp[j-val[i]]+val[i]);

}

}

printf("%d\n",m-val[n]-dp[m-5]);

}

return 0;

}

# 1203 I NEED A OFFER!

**Problem Description**

Speakless很早就想出国，现在他已经考完了所有需要的考试，准备了所有要准备的材料，于是，便需要去申请学校了。要申请国外的任何大学，你都要交纳一定的申请费用，这可是很惊人的。Speakless没有多少钱，总共只攒了n万美元。他将在m个学校中选择若干的（当然要在他的经济承受范围内）。每个学校都有不同的申请费用a（万美元），并且Speakless估计了他得到这个学校offer的可能性b。不同学校之间是否得到offer不会互相影响。“I NEED A OFFER”，他大叫一声。帮帮这个可怜的人吧，帮助他计算一下，他可以收到至少一份offer的最大概率。（如果Speakless选择了多个学校，得到任意一个学校的offer都可以）。

**Input**

输入有若干组数据，每组数据的第一行有两个正整数n,m(0<=n<=10000,0<=m<=10000)   
后面的m行，每行都有两个数据ai(整型),bi(实型)分别表示第i个学校的申请费用和可能拿到offer的概率。   
输入的最后有两个0。

**Output**

每组数据都对应一个输出，表示Speakless可能得到至少一份offer的最大概率。用百分数表示，精确到小数点后一位。

**Sample Input**

10 3

4 0.1

4 0.2

5 0.3

0 0

**Sample Output**

44.0%

代码清单：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<stdlib.h>

#include<algorithm>

using namespace std;

int val[10010];

double dp[10010],p[10010];

int main()

{

int n,m,i,j;

while(scanf("%d%d",&n,&m),n||m)

{

for(i=1;i<=m;i++)

scanf("%d%lf",&val[i],&p[i]);

for(i = 0;i<=n;i++)

dp[i] = 1.0;

// memset(dp,0,sizeof(dp));//非char类型的，只能初始化为0或者-1

for(i=1;i<=m;i++)

{

for(j=n;j>=val[i];j--)

{

dp[j] = min(dp[j],dp[j-val[i]]\*(1-p[i]));

}

}

printf("%.1lf%%\n",(1-dp[n])\*100);

}

return 0;

}

# 2602 Bone Collector

**Problem Description**

Many years ago , in Teddy’s hometown there was a man who was called “Bone Collector”. This man like to collect varies of bones , such as dog’s , cow’s , also he went to the grave（坟墓） …  
The bone collector had a big bag with a volume of V ,and along his trip of collecting there are a lot of bones , obviously , different bone has different value and different volume, now given the each bone’s value along his trip , can you calculate out the maximum of the total value the bone collector can get ?



**Input**

The first line contain a integer T , the number of cases.  
Followed by T cases , each case three lines , the first line contain two integer N , V, (N <= 1000 , V <= 1000 )representing（代表） the number of bones and the volume of his bag. And the second line contain N integers representing the value of each bone. The third line contain N integers representing the volume of each bone.

**Output**

One integer per line representing the maximum of the total value (this number will be less than 231).

**Sample Input**

1

5 10

1 2 3 4 5

5 4 3 2 1

**Sample Output**

14

代码清单：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<stdlib.h>

#include<algorithm>

using namespace std;

int val[10010],dp[10010],v[10010];

int main()

{

int t,N,V,i,j;

scanf("%d",&t);

while(t--)

{

scanf("%d%d",&N,&V);

for(i=1;i<=N;i++)

scanf("%d",&val[i]);

for(i=1;i<=N;i++)

scanf("%d",&v[i]);

memset(dp,0,sizeof(dp));//非char类型的，只能初始化为0或者-1

for(i=1;i<=N;i++)

{

for(j=V;j>=v[i];j--)

{

dp[j] = max(dp[j],dp[j-v[i]]+val[i]);

}

}

printf("%d\n",dp[V]);

}

return 0;

}

# 2955 Robberies（抢劫案）

**Problem Description**

The aspiring（有抱负的，有志气的） Roy the Robber has seen a lot of American movies, and knows that the bad guys usually gets caught in the end, often because they become too greedy（贪婪的）. He has decided to work in the lucrative（赚钱的） business of bank robbery only for a short while, before retiring（即将退休） to a comfortable job at a university.

  
For a few months now, Roy has been assessing（评估） the security of various banks and the amount of cash（现金） they hold. He wants to make a calculated risk, and grab（抢夺） as much money as possible.  
His mother, Ola, has decided upon a tolerable probability（概率） of getting caught. She feels that he is safe enough if the banks he robs together give a probability less than this.

**Input**

The first line of input gives T, the number of cases. For each scenario（方案；脚本）, the first line of input gives a floating point number P, the probability Roy needs to be below, and an integer N, the number of banks he has plans for. Then follow N lines, where line j gives an integer Mj and a floating point number Pj .   
Bank j contains Mj millions, and the probability of getting caught from robbing it is Pj .

**Output**

For each test case, output a line with the maximum number of millions he can expect to get while the probability of getting caught is less than the limit set.  
  
Notes and Constraints  
0 < T <= 100  
0.0 <= P <= 1.0  
0 < N <= 100  
0 < Mj <= 100  
0.0 <= Pj <= 1.0  
A bank goes bankrupt（破产） if it is robbed, and you may assume（假定） that all probabilities are independent as the police have very low funds（基金）.

**Sample Input**

3

0.04 3

1 0.02

2 0.03

3 0.05

0.06 3

2 0.03

2 0.03

3 0.05

0.10 3

1 0.03

2 0.02

3 0.05

**Sample Output**

2

4

6

代码清单：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<stdlib.h>

#include<algorithm>

using namespace std;

int money[10010];

double dp[10010],p[10010];

int main()

{

int t,n,i,j;

double P;

scanf("%d",&t);

while(t--)

{

int sum = 0;

scanf("%lf%d",&P,&n);

for(i=1;i<=n;i++)

{

scanf("%d%lf",&money[i],&p[i]);

sum += money[i];

}

memset(dp,0,sizeof(dp));

dp[0]=1;//?

for(i=1;i<=n;i++)

{

for(j=sum;j>=money[i];j--)

{

dp[j] = max(dp[j],dp[j-money[i]]\*(1-p[i]));

}

}

// for(i=sum;i>=0&&dp[i]<(1-P);i--);

// printf("%d\n",i);

/\*

\*背包容量必然是钱数，因为他能抢的银行有限，钱数也是有限的。

\*然后是求最大逃跑概率，而题中每项给出的是被抓概率，所以要先被1减一下。

\*还有最后求得的逃跑概率随着抢银行的数量增加而减少，多抢一个银行，

\*其钱数必将转化为概率的乘积，所以动态方程也要做出改变。

\*最后遍历，剩余的钱数越多，说明所抢的钱数越少，逃跑几率越大。

\*所以从大到小遍历背包容量，一旦大于p，即为最大概率跳出。

\*/

for(i=sum;i>=0;i--)

if(dp[i]>1-P)

{

printf("%d\n",i);

break;

}

}

return 0;

}

# 2159 FATE

**Problem Description**

最近xhd正在玩一款叫做FATE的游戏，为了得到极品装备，xhd在不停的杀怪做任务。久而久之xhd开始对杀怪产生的厌恶感，但又不得不通过杀怪来升完这最后一级。现在的问题是，xhd升掉最后一级还需n的经验值，xhd还留有m的忍耐度，每杀一个怪xhd会得到相应的经验，并减掉相应的忍耐度。当忍耐度降到0或者0以下时，xhd就不会玩这游戏。xhd还说了他最多只杀s只怪。请问他能升掉这最后一级吗？

**Input**

输入数据有多组，对于每组数据第一行输入n，m，k，s(0 < n,m,k,s < 100)四个正整数。分别表示还需的经验值，保留的忍耐度，怪的种数和最多的杀怪数。接下来输入k行数据。每行数据输入两个正整数a，b(0 < a,b < 20)；分别表示杀掉一只这种怪xhd会得到的经验值和会减掉的忍耐度。(每种怪都有无数个)

**Output**

输出升完这级还能保留的最大忍耐度，如果无法升完这级输出-1。

**Sample Input**

10 10 1 10

1 1

10 10 1 9

1 1

9 10 2 10

1 1

2 2

**Sample Output**

0

-1

1

二维背包。。。

代码清单：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<stdlib.h>

#include<algorithm>

using namespace std;

int ex[110],re[110],dp[110][110];

int main()

{

int m,n,k,s,i,j,t;

while(scanf("%d%d%d%d",&n,&m,&k,&s)!=EOF)

{

int sum = 0;

for(i=1;i<=k;i++)

{

scanf("%d%d",&ex[i],&re[i]);

}

memset(dp,0,sizeof(dp));

for(i=1;i<=k;i++)//从第1个怪到第k个怪

{

for(j=1;j<=s;j++)//杀怪数量从1到s

{

for(t=re[i];t<=m;t++)//耐久度为0时经验必为0

{

dp[j][t] = max(dp[j][t],dp[j-1][t-re[i]]+ex[i]);

}

}

}

for(i=0;i<=m;i++)

{

if(n<=dp[s][i])

{

break;

}

}

if(i==m+1)printf("-1\n");

else

{

printf("%d\n",m-i);

}

}

return 0;

}

# 2191悼念512汶川大地震遇难同胞——珍惜现在，感恩生活

**Problem Description**

急！灾区的食物依然短缺！  
为了挽救灾区同胞的生命，心系灾区同胞的你准备自己采购一些粮食支援灾区，现在假设你一共有资金n元，而市场有m种大米，每种大米都是袋装产品，其价格不等，并且只能整袋购买。  
请问：你用有限的资金最多能采购多少公斤粮食呢？



**Input**

输入数据首先包含一个正整数C，表示有C组测试用例，每组测试用例的第一行是两个整数n和m(1<=n<=100, 1<=m<=100),分别表示经费的金额和大米的种类，然后是m行数据，每行包含3个数p，h和c(1<=p<=20,1<=h<=200,1<=c<=20)，分别表示每袋的价格、每袋的重量以及对应种类大米的袋数。

**Output**

对于每组测试数据，请输出能够购买大米的最多重量，你可以假设经费买不光所有的大米，并且经费你可以不用完。每个实例的输出占一行。

**Sample Input**

1

8 2

2 100 4

4 100 2

**Sample Output**

400

代码清单：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<stdlib.h>

#include<algorithm>

using namespace std;

int price[110],weight[110],num[110],dp[1010];

int main()

{

int c,m,n,i,j,k;

scanf("%d",&c);

while(c--)

{

scanf("%d%d",&n,&m);

for(i=1;i<=m;i++)

{

scanf("%d%d%d",&price[i],&weight[i],&num[i]);

}

memset(dp,0,sizeof(dp));

for(i=1;i<=m;i++)

{

for(j=1;j<=num[i];j++)

{

for(k=n;k>=price[i];k--)

{

dp[k] = max(dp[k],dp[k-price[i]]+weight[i]);

}

}

}

printf("%d\n",dp[n]);

}

return 0;

}

# 1114 Piggy-Bank

**Problem Description**

Before ACM can do anything, a budget（预算） must be prepared and the necessary financial （金融的，财政的）support obtained. The main income for this action comes from Irreversibly Bound Money (IBM). The idea behind is simple. Whenever some ACM member has any small money, he takes all the coins and throws them into a piggy-bank. You know that this process is irreversible, the coins cannot be removed without breaking the pig. After a sufficiently（充分地） long time, there should be enough cash in the piggy-bank to pay everything that needs to be paid.   
  
But there is a big problem with piggy-banks. It is not possible to determine（vt.确定,vi.下决心） how much money is inside. So we might break the pig into pieces only to find out that there is not enough money. Clearly, we want to avoid this unpleasant situation. The only possibility is to weigh the piggy-bank and try to guess how many coins are inside. Assume that we are able to determine the weight of the pig exactly（确切地） and that we know the weights of all coins of a given currency（货币）. Then there is some minimum amount of money in the piggy-bank that we can guarantee（保证）. Your task is to find out this worst case and determine the minimum amount of cash inside the piggy-bank. We need your help. No more prematurely（过早） broken pigs!

**Input**

The input consists of T test cases. The number of them (T) is given on the first line of the input file. Each test case begins with a line containing two integers E and F. They indicate（指示） the weight of an empty pig and of the pig filled with coins. Both weights are given in grams（克）. No pig will weigh more than 10 kg, that means 1 <= E <= F <= 10000. On the second line of each test case, there is an integer number N (1 <= N <= 500) that gives the number of various coins used in the given currency. Following this are exactly N lines, each specifying（指定）one coin type. These lines contain two integers each, P and W (1 <= P <= 50000, 1 <= W <=10000). P is the value of the coin in monetary units（货币单位）, W is it's weight in grams.

**Output**

Print exactly one line of output for each test case. The line must contain the sentence "The minimum amount of money in the piggy-bank is X." where X is the minimum amount of money that can be achieved using coins with the given total weight. If the weight cannot be reached exactly, print a line "This is impossible.".

**Sample Input**

3

10 110

2

1 1

30 50

10 110

2

1 1

50 30

1 6

2

10 3

20 4

**Sample Output**

The minimum amount of money in the piggy-bank is 60.

The minimum amount of money in the piggy-bank is 100.

This is impossible.

代码清单：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<stdlib.h>

#include<algorithm>

using namespace std;

int value[510],weight[510],dp[1000010];

int main()

{

int t,e,f;

int n,i,j,w;

scanf("%d",&t);

while(t--)

{

scanf("%d%d",&e,&f);

w = f-e;

scanf("%d",&n);

for(i=1;i<=n;i++)

{

scanf("%d%d",&value[i],&weight[i]);

}

for(i = 1; i <= w; i++)

dp[i]=10000000;//因为要求小的，所以dp数组必须存大数

dp[0]=0;

for(i=1;i<=n;i++)

{

for(j=weight[i];j<=w;j++)

{

dp[j] = min(dp[j],dp[j-weight[i]]+value[i]);

}

}

if(dp[w]==10000000)

printf("This is impossible.\n");

else

printf("The minimum amount of money in the piggy-bank is %d.\n",dp[w]);

}

return 0;

}

# 1059 Dividing

**Problem Description**

Marsha and Bill own a collection of marbles. They want to split the collection among themselves so that both receive an equal share of the marbles. This would be easy if all the marbles had the same value, because then they could just split the collection in half. But unfortunately, some of the marbles are larger, or more beautiful than others. So, Marsha and Bill start by assigning a value, a natural number between one and six, to each marble. Now they want to divide the marbles so that each of them gets the same total value.   
Unfortunately, they realize that it might be impossible to divide the marbles in this way (even if the total value of all marbles is even). For example, if there are one marble of value 1, one of value 3 and two of value 4, then they cannot be split into sets of equal value. So, they ask you to write a program that checks whether there is a fair partition of the marbles.

**Input**

Each line in the input describes one collection of marbles to be divided. The lines consist of six non-negative integers n1, n2, ..., n6, where ni is the number of marbles of value i. So, the example from above would be described by the input-line ``1 0 1 2 0 0''. The maximum total number of marbles will be 20000.   
The last line of the input file will be ``0 0 0 0 0 0''; do not process this line.

**Output**

For each colletcion, output ``Collection #k:'', where k is the number of the test case, and then either ``Can be divided.'' or ``Can't be divided.''.   
Output a blank line after each test case.

**Sample Input**

1 0 1 2 0 0

1 0 0 0 1 1

0 0 0 0 0 0

**Sample Output**

Collection #1:

Can't be divided.

Collection #2:

Can be divided.

**Source**

代码清单：

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <algorithm>

using namespace std;

int a[6],w[100005],dp[100005];

int main()

{

int i,j,sum,cur,temp;

cur=1; while(scanf("%d%d%d%d%d%d",&a[0],&a[1],&a[2],&a[3],&a[4],&a[5]),a[0]||a[1]||a[2]||a[3]||a[4]||a[5])

{

memset(dp,0,sizeof(dp));

sum=temp=0;

for(i=0;i<6;i++)

{

sum+=(a[i]\*(i+1));

for(j=1;j<=a[i];j<<=1)

{

w[temp++]=j\*(i+1);

a[i]-=j;

}

if(a[i]>0)

w[temp++]=a[i]\*(i+1); //二进制分解

}

printf("Collection #%d:\n",cur++);

if(sum%2!=0) //sum为奇数时直接输出

{

printf("Can't be divided.\n\n");

continue;

}

sum/=2;

for(i=0;i<temp;i++) //之后就可以转换问为01背包求解

for(j=sum;j>=w[i];j--)

dp[j]=max(dp[j],dp[j-w[i]]+w[i]);

if(dp[sum]==sum)

printf("Can be divided.\n\n");

else

printf("Can't be divided.\n\n");

}

return 0;

}

# 1171 Big Event in HDU

**Problem Description**

Nowadays, we all know that Computer College is the biggest department in HDU. But, maybe you don't know that Computer College had ever been split into Computer College and Software College in 2002.  
The splitting is absolutely a big event in HDU! At the same time, it is a trouble thing too. All facilities（工具，设备） must go halves（一半）. First, all facilities are assessed, and two facilities are thought to be same if they have the same value. It is assumed that there is N (0<N<1000) kinds of facilities (different value, different kinds).

**Input**

Input contains multiple test cases. Each test case starts with a number N (0 < N <= 50 -- the total number of different facilities). The next N lines contain an integer V (0<V<=50 --value of facility) and an integer M (0<M<=100 --corresponding（相应的） number of the facilities) each. You can assume that all V are different.  
A test case starting with a negative integer terminates input and this test case is not to be processed.

**Output**

For each case, print one line containing two integers A and B which denote（代表） the value of Computer College and Software College will get respectively（分别）. A and B should be as equal as possible. At the same time, you should guarantee that A is not less than B.

**Sample Input**

2

10 1

20 1

3

10 1

20 2

30 1

-1

**Sample Output**

20 10

40 40

代码清单：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<stdlib.h>

#include<algorithm>

using namespace std;

int value[5010],dp[1000010];

int main()

{

int n,i,j,a,b,k,sum;

while(~scanf("%d",&n),n>0)

{

memset(value,0,sizeof(value));

memset(dp,0,sizeof(dp));

k = 0;

sum = 0;

for(i = 0;i<n;i++)

{

scanf("%d%d",&a,&b);

while(b--)

{

value[k++] = a;//将价值存入数组,构造01背包

sum+=a;

}

}

for(i = 0;i<k;i++)

{

for(j = sum/2;j>=value[i];j--)//01背包

{

dp[j] = max(dp[j],dp[j-value[i]]+value[i]);

}

}

printf("%d %d\n",sum-dp[sum/2],dp[sum/2]);

}

return 0;

}

# 1160 FatMouse's Speed

**Problem Description**

FatMouse believes that the fatter a mouse is, the faster it runs. To disprove this, you want to take the data on a collection of mice and put as large a subset（子集） of this data as possible into a sequence so that the weights are increasing, but the speeds are decreasing（降低）.

**Input**

Input contains data for a bunch of mice, one mouse per line, terminated（结束） by end of file（文件）.  
  
The data for a particular mouse will consist of a pair of integers: the first representing（代表） its size in grams and the second representing its speed in centimeters per second. Both integers are between 1 and 10000. The data in each test case will contain information for at most 1000 mice.  
  
Two mice may have the same weight, the same speed, or even the same weight and speed.

**Output**

Your program（程序） should output a sequence of lines of data; the first line should contain a number n; the remaining n lines should each contain a single positive integer (each one representing a mouse). If these n integers are m[1], m[2],..., m[n] then it must be the case that   
  
W[m[1]] < W[m[2]] < ... < W[m[n]]  
  
and   
  
S[m[1]] > S[m[2]] > ... > S[m[n]]  
  
In order for the answer to be correct, n should be as large as possible.  
All inequalities（不等式） are strict（严格的）: weights must be strictly increasing, and speeds must be strictly decreasing. There may be many correct outputs for a given input, your program only needs to find one.

**Sample Input**

6008 1300

6000 2100

500 2000

1000 4000

1100 3000

6000 2000

8000 1400

6000 1200

2000 1900

**Sample Output**

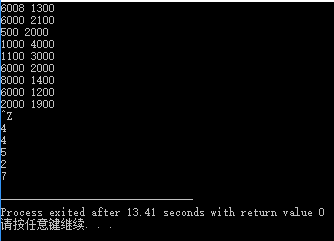
4

4

5

9

7



代码清单：

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct

{

int pos;

int w,s;

}mice;

typedef struct

{

int pre;

int sum;

}dpp;

mice m[1010];

dpp dp[1010];

int cmp(const void \*a, const void \*b)

{

mice \*p = (mice \*)a, \*q = (mice \*)b;

if(p->s != q->s) return p->s - q->s;

return q->w - p->w;

}

int main()

{

int a,b,max1,pr,max2;

int n = 1;

int i,j,f;

while(~scanf("%d%d", &a, &b))

{

m[n].pos = n;

m[n].w = a;

m[n++].s = b;

}

qsort(m, n, sizeof(mice), cmp);

max2 = 0;

for(i = 1; i < n; i++)

{

max1 = 0;

for(j = i-1; j > 0; j--)

{

if(m[j].s < m[i].s && m[j].w > m[i].w && max1 < dp[j].sum)

{

max1 = dp[j].sum;

pr = j;

}

}

if(!max1)

{

dp[i].sum = 1;

dp[i].pre = 0;

}

else

{

dp[i].sum = max1+1;

dp[i].pre = pr;

}

if(max2 < dp[i].sum)

{

max2 = dp[i].sum;

f = i;

}

}

printf("%d\n", max2);

while(dp[f].pre)

{

printf("%d\n", m[f].pos);

f = dp[f].pre;

}

printf("%d\n", m[f].pos);

return 0;

}

# 2844 Coins

**Problem Description**

Whuacmers use coins.They have coins of value A1,A2,A3...An Silverland dollar. One day Hibix opened purse and found there were some coins. He decided to buy a very nice watch in a nearby shop. He wanted to pay the exact price(without change) and he known the price would not more than m.But he didn't know the exact price of the watch.  
  
You are to write a program which reads n,m,A1,A2,A3...An and C1,C2,C3...Cn corresponding to the number of Tony's coins of value A1,A2,A3...An then calculate how many prices(form 1 to m) Tony can pay use these coins.

**Input**

The input contains several test cases. The first line of each test case contains two integers n(1 ≤ n ≤ 100),m(m ≤ 100000).The second line contains 2n integers, denoting A1,A2,A3...An,C1,C2,C3...Cn (1 ≤ Ai ≤ 100000,1 ≤ Ci ≤ 1000). The last test case is followed by two zeros.

**Output**

For each test case output the answer on a single line.

**Sample Input**

3 10

1 2 4 2 1 1

2 5

1 4 2 1

0 0

**Sample Output**

8

4

代码清单：

#include <stdio.h>

#include <algorithm>

#include <string.h>

using namespace std;

const int MAX=100000;

int dp[MAX];

int c[MAX],w[MAX];

int v;

void ZeroOnePack(int cost,int wei)//01

{

int i;

for(i = v;i>=cost;i--)

{

dp[i] = max(dp[i],dp[i-cost]+wei);

}

}

void CompletePack(int cost,int wei)//完全

{

int i;

for(i = cost;i<=v;i++)

{

dp[i] = max(dp[i],dp[i-cost]+wei);

}

}

void MultiplePack(int cost,int wei,int cnt)//多重

{

if(v<=cnt\*cost)//如果总容量比这个物品的容量要小，那么这个物品可以直到取完，相当于完全背包

{

CompletePack(cost,wei);

return ;

}

else//否则就将多重背包转化为01背包

{

int k = 1;

while(k<=cnt)

{

ZeroOnePack(k\*cost,k\*wei);

cnt = cnt-k;

k = 2\*k;

}

ZeroOnePack(cnt\*cost,cnt\*wei);

}

}

int main()

{

int n;

while(~scanf("%d%d",&n,&v),n+v)

{

int i;

for(i = 0;i<n;i++)

scanf("%d",&c[i]);

for(i = 0;i<n;i++)

scanf("%d",&w[i]);

memset(dp,0,sizeof(dp));

for(i = 0;i<n;i++)

{

MultiplePack(c[i],c[i],w[i]);

}

int sum = 0;

for(i = 1;i<=v;i++)

{

if(dp[i]==i)

{

sum++;

}

}

printf("%d\n",sum);

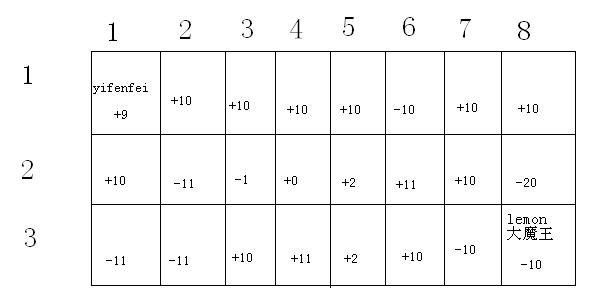
}

return 0;

}

# 2771 命运

**Problem Description**

穿过幽谷意味着离大魔王lemon已经无限接近了！  
可谁能想到，yifenfei在斩杀了一些虾兵蟹将后，却再次面临命运大迷宫的考验，这是魔王lemon设下的又一个机关。要知道，不论何人，若在迷宫中被困1小时以上，则必死无疑！  
可怜的yifenfei为了去救MM，义无返顾地跳进了迷宫。让我们一起帮帮执着的他吧！  
命运大迷宫可以看成是一个两维的方格阵列，如下图所示：  
   
yifenfei一开始在左上角，目的当然是到达右下角的大魔王所在地。迷宫的每一个格子都受到幸运女神眷恋或者痛苦魔王的诅咒，所以每个格子都对应一个值，走到那里便自动得到了对应的值。  
现在规定yifenfei只能向右或者向下走，向下一次只能走一格。但是如果向右走，则每次可以走一格或者走到该行的列数是当前所在列数倍数的格子，即：如果当前格子是（x,y），下一步可以是（x+1,y），(x,y+1)或者(x,y\*k) 其中k>1。   
为了能够最大把握的消灭魔王lemon，yifenfei希望能够在这个命运大迷宫中得到最大的幸运值。  


**Input**

输入数据首先是一个整数C，表示测试数据的组数。  
每组测试数据的第一行是两个整数n,m，分别表示行数和列数(1<=n<=20,10<=m<=1000)；  
接着是n行数据，每行包含m个整数，表示n行m列的格子对应的幸运值K ( |k|<100 )。

**Output**

请对应每组测试数据输出一个整数，表示yifenfei可以得到的最大幸运值。

**Sample Input**

1

3 8

9 10 10 10 10 -10 10 10

10 -11 -1 0 2 11 10 -20

-11 -11 10 11 2 10 -10 -10

**Sample Output**

52

代码清单：

#include <stdio.h>

#include <string.h>

#include <algorithm>

using namespace std;

int map[25][1005],dp[25][1005];

int main()

{

int t,n,m,i,j,k;

scanf("%d",&t);

while(t--)

{

scanf("%d%d",&n,&m);

for(i = 1; i<=n; i++)

for(j = 1; j<=m; j++)

scanf("%d",&map[i][j]);

for(i = 0; i<=n; i++)

dp[i][0] = -99999999;

for(i = 0; i<=m; i++)

dp[0][i] = -99999999;

dp[1][0] = dp[0][1] = 0;

for(i = 1; i<=n; i++)

{

for(j = 1; j<=m; j++)

{

dp[i][j] = max(dp[i-1][j],dp[i][j-1]);

for(k = 2; k<=m; k++)

{

if(j/k == (double)j/k)//找倍数关系

dp[i][j] = max(dp[i][j],dp[i][j/k]);

}

dp[i][j]+=map[i][j];

}

}

printf("%d\n",dp[n][m]);

}

return 0;

}

# 1011 Starship Troopers

树形dp背包

**Problem Description**

You, the leader of Starship Troopers, are sent to destroy a base of the bugs. The base is built underground. It is actually a huge cavern, which consists of many rooms connected with tunnels. Each room is occupied by some bugs, and their brains hide in some of the rooms. Scientists have just developed a new weapon and want to experiment it on some brains. Your task is to destroy the whole base, and capture as many brains as possible.  
  
To kill all the bugs is always easier than to capture their brains. A map is drawn for you, with all the rooms marked by the amount of bugs inside, and the possibility of containing a brain. The cavern's structure is like a tree in such a way that there is one unique path leading to each room from the entrance. To finish the battle as soon as possible, you do not want to wait for the troopers to clear a room before advancing to the next one, instead you have to leave some troopers at each room passed to fight all the bugs inside. The troopers never re-enter a room where they have visited before.  
  
A starship trooper can fight against 20 bugs. Since you do not have enough troopers, you can only take some of the rooms and let the nerve gas do the rest of the job. At the mean time, you should maximize the possibility of capturing a brain. To simplify the problem, just maximize the sum of all the possibilities of containing brains for the taken rooms. Making such a plan is a difficult job. You need the help of a computer.

**Input**

The input contains several test cases. The first line of each test case contains two integers N (0 < N <= 100) and M (0 <= M <= 100), which are the number of rooms in the cavern and the number of starship troopers you have, respectively. The following N lines give the description of the rooms. Each line contains two non-negative integers -- the amount of bugs inside and the possibility of containing a brain, respectively. The next N - 1 lines give the description of tunnels. Each tunnel is described by two integers, which are the indices of the two rooms it connects. Rooms are numbered from 1 and room 1 is the entrance to the cavern.  
  
The last test case is followed by two -1's.

**Output**

For each test case, print on a single line the maximum sum of all the possibilities of containing brains for the taken rooms.

**Sample Input**

5 10

50 10

40 10

40 20

65 30

70 30

1 2

1 3

2 4

2 5

1 1

20 7

-1 -1

**Sample Output**

50

7

 题意：是有n个洞组成一棵树，你有m个士兵，你从1号房间开始攻打，每个洞有a个"bugs"和b的价值。你的一个士兵可以打20个"bugs"，为了拿到这个洞的价值b你必须留下k个士兵消灭这个洞的所有"bugs"（k\*20>="bugs"的数量，且留下的士兵不可以再去攻打其他的洞，且必须攻打了前面的洞才可以攻打后面的洞）。问你花费这m个士兵可以得到的最大价值是多少。

代码清单：

#include <iostream>

#include <vector>

#include <cstring>

#include <cstdio>

#include <algorithm>

using namespace std;

const int Ni = 110;

int n,m;

int cost[Ni],weg[Ni];

int dp[Ni][Ni];

bool vis[Ni];

vector<int> dv[Ni];

void dfs(int p)

{

int i,j,k;

int temp=(cost[p]+19)/20;

for(i=temp;i<=m;i++) dp[p][i]=weg[p];

vis[p]=1;

for(i=0;i<dv[p].size();i++)

{

int t=dv[p][i];

if(vis[t]) continue;

dfs(t);

for(j=m;j>=temp;j--)

{

for(k=1; k<=j-temp; k++)//留下temp攻打p

dp[p][j]=max(dp[p][j],dp[p][j-k]+dp[t][k]);

}

}

}

int main()

{

int i;

while(scanf("%d%d",&n,&m),n!=-1||m!=-1)

{

for(i=0; i<=n; i++) dv[i].clear();

memset(dp,0,sizeof(dp));

memset(vis,0,sizeof(vis));

for(i=1; i<=n; i++)

scanf("%d%d",cost+i,weg+i);

for(i=1; i<n; i++)

{

int u,v;

scanf("%d%d",&u,&v);

dv[u].push\_back(v);

dv[v].push\_back(u);

}

if(m==0) {printf("0\n");continue;}

dfs(1);

printf("%d\n",dp[1][m]);

}

return 0;

}

# 1248 寒冰王座

**Problem Description**

不死族的巫妖王发工资拉,死亡骑士拿到一张N元的钞票(记住,只有一张钞票),为了防止自己在战斗中频繁的死掉,他决定给自己买一些道具,于是他来到了地精商店前.  
死亡骑士:"我要买道具!"  
地精商人:"我们这里有三种道具,血瓶150块一个,魔法药200块一个,无敌药水350块一个."  
死亡骑士:"好的,给我一个血瓶."  
说完他掏出那张N元的大钞递给地精商人.  
地精商人:"我忘了提醒你了,我们这里没有找客人钱的习惯的,多的钱我们都当小费收了的,嘿嘿."  
死亡骑士:"......"  
死亡骑士想,与其把钱当小费送个他还不如自己多买一点道具,反正以后都要买的,早点买了放在家里也好,但是要尽量少让他赚小费.  
现在死亡骑士希望你能帮他计算一下,最少他要给地精商人多少小费.

**Input**

输入数据的第一行是一个整数T(1<=T<=100),代表测试数据的数量.然后是T行测试数据,每个测试数据只包含一个正整数N(1<=N<=10000),N代表死亡骑士手中钞票的面值.  
注意:地精商店只有题中描述的三种道具.

**Output**

对于每组测试数据,请你输出死亡骑士最少要浪费多少钱给地精商人作为小费.

**Sample Input**

2

900

250

**Sample Output**

0

50

//完全背包

#include<stdio.h>

#include<string.h>

int dp[10001];

int tool[4]={0,150,200,350};

int main()

{

int t,n,i,j;

scanf("%d",&t);

while(t--)

{

scanf("%d",&n);

memset(dp,0,sizeof dp);

for(i=1;i<=3;i++)/\*和0 1 背包问题一样 这里其实指的是 种类数 即该种类存不存在 要不要会出现哪种结果 然后比较两者的大小\*/

for(j=tool[i];j<=n;j++)

dp[j]=dp[j]>dp[j-tool[i]]+tool[i]?dp[j]:dp[j-tool[i]]+tool[i];

printf("%d\n",n-dp[n]);

}

return 0;

}

# 1284钱币兑换问题

**Problem Description**

在一个国家仅有1分，2分，3分硬币，将钱N兑换成硬币有很多种兑法。请你编程序计算出共有多少种兑法。

**Input**

每行只有一个正整数N，N小于32768。

**Output**

对应每个输入，输出兑换方法数。

**Sample Input**

2934

12553

**Sample Output**

718831

13137761

//完全背包

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

int dp[32768];

int main()

{

int n,i,j;

int val[3]={1,2,3};

while(scanf("%d",&n)!=EOF)

{

memset(dp,0,sizeof(dp));

dp[0]=1;

for(i=0;i<3;i++)

for(j=val[i];j<=n;j++)

dp[j]+=dp[j-val[i]];

printf("%d\n",dp[n]);

}

}

# 1712 ACboy needs your help

**Problem Description**

ACboy has N courses this term, and he plans to spend at most M days on study.Of course,the profit he will gain from different course depending on the days he spend on it.How to arrange the M days for the N courses to maximize the profit?

**Input**

The input consists of multiple data sets. A data set starts with a line containing two positive integers N and M, N is the number of courses, M is the days ACboy has.  
Next follow a matrix A[i][j], (1<=i<=N<=100,1<=j<=M<=100).A[i][j] indicates if ACboy spend j days on ith course he will get profit of value A[i][j].  
N = 0 and M = 0 ends the input.

**Output**

For each data set, your program should output a line which contains the number of the max profit ACboy will gain.

**Sample Input**

2 2

1 2

1 3

2 2

2 1

2 1

2 3

3 2 1

3 2 1

0 0

**Sample Output**

3

4

6

//背包分组

#include <stdio.h>

#include <iostream>

#include <algorithm>

#include <math.h>

#include <string.h>

using namespace std;

int a[105][105],dp[105];

int n,m;

int main()

{

while(~scanf("%d%d",&n,&m))

{

if(n==0&&m==0)

break;

for(int i=1;i<=n;i++)

{

for(int j=1;j<=m;j++)

scanf("%d",&a[i][j]);

}

memset(dp,0,sizeof(dp));

for(int i=1;i<=n;i++) //n门课

{

for(int j=m;j>=0;j--) //花费时间

{

for(int k=1;k<=j;k++) //获得的价值

{

dp[j]=max(dp[j],dp[j-k]+a[i][k]);

}

}

}

printf("%d\n",dp[m]);

}

return 0;

}

# 2079选课时间(多重背包)

**Problem Description**

又到了选课的时间了，xhd看着选课表发呆，为了想让下一学期好过点，他想知道学n个学分共有多少组合。你来帮帮他吧。（xhd认为一样学分的课没区别）

**Input**

输入数据的第一行是一个数据T，表示有T组数据。  
每组数据的第一行是两个整数n(1 <= n <= 40)，k(1 <= k <= 8)。  
接着有k行，每行有两个整数a(1 <= a <= 8),b(1 <= b <= 10)，表示学分为a的课有b门。

**Output**

对于每组输入数据，输出一个整数，表示学n个学分的组合数。

**Sample Input**

2

2 2

1 2

2 1

40 8

1 1

2 2

3 2

4 2

5 8

6 9

7 6

8 8

**Sample Output**

2

445

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <algorithm>

#include <cmath>

using namespace std;

const int N = 55;

int sum;

int dp[N];

struct node

{

int a, b;

}cla[N];

int main()

{

// freopen("in.txt", "r", stdin);

int t, n, k;

scanf("%d", &t);

while(t --)

{

memset(dp, 0, sizeof(dp));

scanf("%d%d", &n, &k);

for(int i = 1; i <= k; i ++)

scanf("%d%d", &cla[i].a, &cla[i].b);

dp[0] = 1;

for(int i = 1; i <= k; i ++)

for(int j = n; j >= cla[i].a; j --)//容量

for(int l = 1; l <= cla[i].b; l ++)//数量

{

if(j - cla[i].a \* l >= 0)

dp[j] += dp[j - cla[i].a \* l];

else

break;

}

printf("%d\n", dp[n]);

}

return 0;

}