信息学奥林匹克竞赛

Olympiad in Informatics

动态规划之背包问题

陈 真

太原市第五中学校

背包的基础类型

01背包

01背包模型：给定N种物品，其中第i种物品的体积为wi，价值为vi，每种物品只有**一件**。有一体积为M的背包，请选择一些物品放入背包，使得物品总体积不超过M的前提下，物品的价值总和最大。

阶段划分

可按已处理物品的件数将选择过程划分为N个阶段，设阶段变量i代表前i件物品。

设计状态

可用二维状态变量(i,j)代表前i件物品，背包体积为j，设最优值函数f代表放入背包物品的最大价值和。

一是设f[i][j]代表前i件物品选择若干件放入体积为j的背包得到的最大价值和（**体积j可以不装满**）；

二是设f[i][j]代表前i件物品选择若干**恰好**放入体积为j的背包得到的最大价值和（**体积j必须装满**）。

确定决策

对于状态f[i][j]，问题进入了第i阶段，决策对象为第i种物品，可以做出的决策有：选、不选。

转移方程

$$f[i][j]=\max\begin{cases} f[i-1][j] & \text{不选第i种物品} \\ f[i-1][j-w[i]]+v[i] & j>=w[i]\ \text{选1件第i种物品} \end{cases}$$

确定边界

若状态**f[i][j]**含义为第一种，边界**f[i][j]=0**

若状态**f[i][j]**含义为第二种，边界为**f[0][0]=0**，其余**f[i][j]**的值设为负无穷，

确定目标

若状态**f[i][j]**含义为第一种，目标解为**f[n][m]**

若状态f[i][j]含义为第二种，由于不确定最后放入的物品所占体积，须枚举j，目标解为max(f[n][j])

背包的基础类型

01背包

01背包模型：给定N种物品，其中第i种物品的体积为wi，价值为vi，每种物品只有**一件**。有一体积为M的背包，请选择一些物品放入背包，使得物品总体积不超过M的前提下，物品的价值总和最大。

【题目01】采药（题目来源：NOIP2005普及组 ）

【问题描述】辰辰是个天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到一个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

如果你是辰辰，你能完成这个任务吗?

【输入格式】第一行有 2 个整数 M（1≤M≤1000）和 N（1≤N≤100），用一个空格隔开，M 代表总共能够用来采药的时间，N 代表山洞里的草药的数目。

接下来的 N 行每行包括两个在 1 到 100 之间（包括 1 和 100）的整数，分别表示采摘某株草药的时间wi和这株草药的价值vi。

【输出格式】输出在规定的时间内可以采到的草药的最大总价值。

【输入输出样例】

样例输入	样例输出
10 4 2 1 3 3 4 5 7 9	12

背包的基础类型

01背包

01背包模型：给定N种物品，其中第i种物品的体积为wi，价值为vi，每种物品只有**一件**。有一体积为M的背包，请选择一些物品放入背包，使得物品总体积不超过M的前提下，物品的价值总和最大。

【题目01】采药（题目来源：NOIP2005普及组）

```cpp
#include<bits/stdc++.h>
using namespace std;
int f[105][1005];
int w[105],v[105];
int main(){
    int m,n;
    //m为背包容量，n为物品种数
    cin>>m>>n;
    for(int i=1;i<=n;i++){
        cin>>w[i]>>v[i];
    }
    //由于f数组为全局变量，因此边界f[i][j]=0省略了
    for(int i=1;i<=n;i++){
        for(int j=0;j<=m;j++){
            f[i][j]=f[i-1][j];
            if(j>=w[i])
            f[i][j]=max(f[i][j],f[i-1][j-w[i]]+v[i]);
        }
    }
    cout<<f[n][m]<<endl;
    return 0;
}
```

背包的基础类型

01背包优化1：滚动数组

观察状态转移方程f[i][j]=max(f[i-1][j],f[i-1][j-w[i]]+v[i])，每一阶段的状态的最优解只和上一阶段有关，而与更早之前的阶段无关。

	0	1	2	3	4	5	6	7	8	9	10
0	0	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞
1	0	−∞	1	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞
2	0	−∞	1	3	−∞	4	−∞	−∞	−∞	−∞	−∞
3	0	−∞	1	3	5	4	6	8	−∞	9	−∞
4	0	−∞	1	3	5	4	6	9	−∞	10	12

```
for(int i=1;i<=n;i++){
        for(int j=0;j<=m;j++){
            f[i&1][j]=f[(i-1)&1][j];
            if(j>=w[i])
            f[i&1][j]=max(f[i&1][j],f[(i-1)&1][j-w[i]]+v[i]);
        }
    }
```

滚动数组思路：只需要开辟数组f[2][m]，让第i阶段的状态存储在f[i&1][j]中即可，当i为奇数时，i&1=1；当i为偶数时，i&1=0。状态f[i&1][j]的值相当于在f[0][j]和f[1][j]中滚动存储，顾名滚动数组优化。空间复杂度优化。

背包的基础类型

01背包优化2：降维优化

前面表中，**第3阶段到第4阶段的状态转**移情况，发现第4阶段的每一个状态首先都会由一个垂直箭头（不选第4种物品）转移，再看能否由斜线箭头（选第4种物品）转移。

更一般地，若不选第i种物品，第i阶段所有状态的值会与第i-1阶段所有状态的值一样，等价于将f[i-1][j]复制了一份给f[i][j]，因此可进一步地将滚动数组f[2][m]中的第一维也省略，令**f[j]代表前i种物品选择若干恰好装入体积为j的背包的最大价值和（体积j必须装满）。**

正序枚举j

	0	1	2	3	4	5	6	7	8	9	10
1	0	-∞	1	-∞	2	-∞	3	-∞	4	-∞	5

倒序枚举j

	0	1	2	3	4	5	6	7	8	9	10
1	0	-∞	1	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞

```
1  f[0]=0;
2  for(int i=1;i<=n;i++){
3      for(int j=m;j>=w[i];j--){//倒序枚举j
4          f[j]=max(f[j],f[j-w[i]]+v[i]);
5          }
6      }
7      int ans=-1e9;
8      for(int j=0;j<=m;j++)ans=max(ans,f[j]);
```

	0	1	2	3	4	5	6	7	8	9	10
0	0	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
1	0	-∞	1	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
2	0	-∞	1	3	-∞	4	-∞	-∞	-∞	-∞	-∞
3	0	-∞	1	3	5	4	6	8	-∞	9	-∞
4	0	-∞	1	3	5	4	6	9	-∞	10	12

背包的基础类型

常见01背包错误写法：

请你分析错因

```
1   //01背包 二维
2   #include <bits/stdc++.h>
3   using namespace std;
4   int f[1001][1001];
5   int main() {
6       int m, n, c, w;
7       cin >> m >> n;
8       for (int i = 1; i <= n; i++) {
9           cin >> w >> c;
10          for (int j = 0; j <= m; j++)
11              if (j >= w)
12                  f[i][j] = max(f[i][j - w] + c, f[i - 1][j]);
13      }
14      cout << f[n][m];
15      return 0;
16  }
```

错

错因分析：没有处理j<w时的状态，导致求出的一部分结果丢失。

背包的基础类型

【题目02】集合 Subset Sums

【问题描述】对于从1到N (1 <= N <= 39) 的连续整数集合，能划分成两个子集合，且保证每个集合的数字和是相等的。

举个例子，如果N=3，对于{1，2，3}能划分成两个子集合，他们每个的所有数字和是相等的：{3} 和 {1,2} 这是唯一一种分法（交换集合位置被认为是同一种划分方案，因此不会增加划分方案总数）

如果N=7，有四种方法能划分集合{1，2，3，4，5，6，7}，每一种分法的子集合各数字和是相等的:{1,6,7} 和 {2,3,4,5} {注 1+6+7=2+3+4+5} {2,5,7} 和 {1,3,4,6} {3,4,7} 和 {1,2,5,6} {1,2,4,7} 和 {3,5,6}

给出N，你的程序应该输出划分方案总数，如果不存在这样的划分方案，则输出0。程序不能预存结果直接输出（不能打表）。

【输入格式】输入文件只有一行，且只有一个整数N

【输出格式】输出划分方案总数，如果不存在则输出0。

【样例输入】7

【样例输出】4

背包的基础类型

【题目02】集合 Subset Sums

分析：

N个数的前N项和NUM（**背包容量**），N个数（**物品数量**）

对于每个数，数本身的大小(**物品的体积**)和 方法数（**物品的价值**）

注意：本题求解的是方案总数。另外当n为偶数时，不存在方案。

```
if (j >= i)
    f[i][j] = f[i - 1][j] + f[i - 1][j - i];
else
    f[i][j] = f[i - 1][j];
```

最终方案： f[n][num / 2]

背包的基础类型

【题目03】竞赛真理

【问题描述】Antares在经历了无数次学科竞赛的失败以后，得到了一个真理：做一题就要对一题！但是要完全正确地做对一题是要花很多时间（包括调试时间），而竞赛的时间有限。所以开始做题之前最好先认真审题，估计一下每一题如果要完全正确地做出来所需要的时间，然后选择一些有把握的题目先做。当然，如果做完了预先选择的题目之后还有时间，但是这些时间又不足以完全解决一道题目，应该把其他的题目用贪心之类的算法随便做做，争取“骗”一点分数。 问题求解： 根据每一题解题时间的估计值，确定一种做题方案（即哪些题目认真做，哪些题目“骗”分，哪些不做），使能在限定的时间内获得最高的得分。

根据每一题解题时间的估计值，确定一种做题方案（即哪些题目认真做，哪些题目“骗”分，哪些不做），使能在限定的时间内获得最高的得分，从文件读入数据。

背包的基础类型

【题目03】竞赛真理

【输入格式】第一行有两个正整数N和T，表示题目的总数以及竞赛的时限（单位秒）。

以下的N行，每行4个正整数W1i 、T1i 、W2i 、T2i ，分别表示第i题：完全正确做出来的得分，完全正确做出来所花费的时间（单位秒），“骗”来的分数，“骗”分所花费的时间（单位秒）。

其中，3≤N≤30，2≤T≤1080000，1≤ W1i 、W2i ≤30000，1≤T1i 、T2i≤T。

【输出格式】一个整数，直接把所能得到的最高分值。

【样例输入1】

```
4 10800
18 3600 3 1800
22 4000 12 3000
28 6000 0 3000
32 8000 24 6000
```

【样例输入2】

```
3 7200
50 5400 10 900 50 7200
10 900 50 5400 10 900
```

【样例输出2】

```
70
```

背包的基础类型

【题目03】竞赛真理

N题，每道题给出正解得分、时间，“骗”分得分、时间，求解在给定时间T内如何做题使得得分最高，输出这个分数

重要条件：对于每道题，要么做正解，要么骗分，要么不做

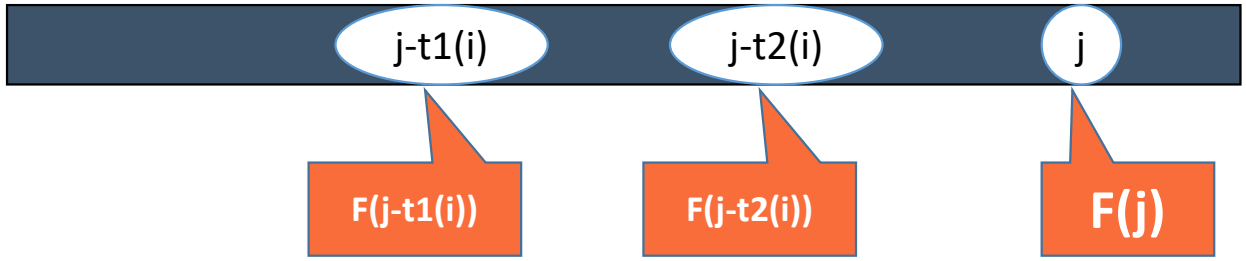
01背包

容积是？体积是？价值是？

状态：f(j)表示时间为j时的最大得分

决策转移：t1(i)表示第i题正解用时，v1(i)正解得分

t2(i)表示第i题骗分用时，v2(i)骗分得分



背包的基础类型

完全背包

全背包的模型为：给定N种物品，其中第i个物品的体积为wi，价值为vi，每种物品有**无数件**。有一体积为M的背包，请选择一些物品放入背包，使得物品总体积不超过M的前提下，物品的价值总和最大。

阶段划分 设阶段变量i代表前i种物品。

设计状态 设f[i][j]代表前i种物品选择若干恰好放入体积为j的背包的最大价值和（体积j必须装满）。

确定决策 对于状态f[i][j]，问题进入第i阶段，决策对象为第i种物品，做出的决策即第i种物品选几件，可以为0件、1件、2件、...、j/w[i]件。设决策变量为k，那么决策集合为{0<=k<=j/w[i]|k}。

转移方程 $f[i][j]=max(f[i-1][j-k\cdot w[i]]+k\cdot v[i])\ \ (0<=k<=j/w[i]|k)$。

确定边界 **f[i][j]=-∞,f[0][0]=0，含义与01背包问题一致。**

确定目标 max(f[n][j]) (0<=j<=m|j)

```
1  f[0][0]=0;
2  for(int i=1;i<=n;i++){//枚举阶段
3      for(int j=0;j<=m;j++){//枚举状态
4          for(int k=0;k<=j/w[i];k++)//枚举决策
5              f[i][j]=max(f[i][j],f[i-1][j-k*w[i]]+k*v[i]);
6      }
7  }
```

背包的基础类型

完全背包优化：降维优化

	0	1	2	3	4	5	6	7	8	9	10
0	0	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞
1	0	−∞	1	−∞	2	−∞	3	−∞	4	−∞	5

对于状态f[1][j]，决策对象为第1种物品，体积为2，价值为1。其中：

对于状态f[1][0]，决策集合为{选0件}；

对于状态f[1][2]，决策集合为{选0件,选1件}；

对于状态f[1][4]，决策集合为{选0件，选1件，选2件}；

对于状态f[1][6]，决策集合为{选0件，选1件，选2件，选3件}；

对于状态f[1][8]，决策集合为{选0件，选1件，选2件，选3件，选4件}；

对于状态f[1][10]，决策集合为{选0件，选1件，选2件，选3件，选4件，选5件}；

```
1  f[0]=0;
2  for(int i=1;i<=n;i++){
3      for(int j=w[i];j<=m;j++){//正序枚举j
4          f[j]=max(f[j],f[j-w[i]]+v[i]);
5      }
6  }
```

若第i种物品选0件，第i阶段所有状态的值会与第i-1阶段所有状态的值一样，等价于将f[i-1][j]复制了一份给f[i][j]。因此，可省略i这一维度，优化该算法的空间复杂度为O(m)，因为当第i种物品选1件时，状态f[i][j]的值是由本阶段状态f[i][j-w[i]]的值转移，所以体积j**正序枚举**即可，

背包的基础类型

【题目04】Elimination（来源：CF417A ）

【问题描述】在2214年的的 "Russian代码杯" 的决赛选手将是在淘汰赛中的获胜者。两种形式的选拔赛 淘汰赛有两种形式，我们把他叫正常赛与特殊赛，正常赛中的题目为c道，获胜的n个人可以参加决赛。特殊赛题目为d道，获胜的1个人可参加决赛。此外，有k位大佬拥有报送名额，不用参加淘汰赛。（黑幕）组委会需要你组织好各轮淘汰赛，比赛用到的题目要最少，但是保证在所有淘汰赛结束后，有n:m位选手进入决赛。

【输入格式】

第1行输入c和d（1<=c,d<=100）

第2行输入n和m（1<=n,m<=100）

第3行输入k（1<=k<=100） （c,d,n,m,k为正整数）

【输出格式】输出最少需要多少道题目

【样例输入1】

1 10

7 2

1

【样例输出1】2

背包的基础类型

【题目04】Elimination（来源：CF417A）

只有两种物品的完全背包

背包容量为N*m个选手晋级，价值：题目量最少

物品1：N个人晋级，需要C道题目

物品2：1个人晋级，需要d道题目

```
f[i] = min(f[i - n] + c, f[i - 1] + d)
```

注意：题目**不是要求必须晋级 n·m 人，而是要求至少晋级 n·m人**

答案：寻找f[n*m-k]到f[n*m]的最小值

背包的基础类型

【题目05】Cut Ribbon（题目来源：CF189A ）

【问题描述】给一长度为n的缎带，要求将其剪成若干长度为a,b,c的缎带，且缎带数量尽可能多。

【输入格式】

输入仅一行，四个正整数n,a,b,c(n,a,b,c≤4000)。

【输出格式】

输出仅一行，即缎带数量的最大值。

【样例输入1】5 5 3 2

【样例输出1】2

说明/提示

In the first example Polycarpus can cut the ribbon in such way: the first piece has length 2, the second piece has length 3.

In the second example Polycarpus can cut the ribbon in such way: the first piece has length 5, the second piece has length 2.

背包的基础类型

【题目05】Cut Ribbon（题目来源：CF189A ）

【分析】 有三种物品的完全背包问题

背包的容量为n，每件物品的体积为 a，b，c，价值为1。求最多放置的物品数量。

背包一定要装满！！！

【实现】将计算最大数量的f数组全部取−1或0XC0，跑完全背包的方程时，**记住判断放入这个物品前时的f数组是不是-1**，如果不是，再执行状态转移方程。另外，一个长度为0的绸带，无论a、b、c是多少，都只能切成0段，赋初始值。

背包的基础类型

【题目06】买干草（来源：USACO08NOV)

【问题描述】约翰的干草库存已经告罄，他打算为奶牛们采购H(1≤H≤50000)镑干草.
他知道N(1≤N≤100)个干草公司，现在用1到N给它们编号.第i公司卖的干草包重量 为Pi(1≤Pi≤5,000)磅，需要的开销为Ci(1≤Ci≤5,000)美元.每个干草公司的货源都十分充足， 可以卖出无限多的干草包.
帮助约翰找到最小的开销来满足需要，即采购到至少H镑干草.

【输入格式】

* Line 1: Two space-separated integers: N and H
* Lines 2..N+1: Line i+1 contains two space-separated integers: P_i and C_i

【输出格式】

* Line 1: A single integer representing the minimum cost FJ needs to pay to obtain at least H pounds of hay.

【样例输入1】 2 15 3 2 5 3

【样例输出2】9

【说明/提示】

FJ can buy three packages from the second supplier for a total cost of 9.

完全背包?
背包容量?
答案如何统计?

背包的基础类型

【题目07】 货币系统（NOIP2018）

【问题描述】网友的国度中共有 **n种**不同面额的货币，第 i种货币的面额**为 a[i]**，你可以假设每一种货币都有无穷多张。为了方便，我们把**货币种数为 n、面额数组为a[1...n]** 的货币系统记作**(n,a)** 。

在一个完善的货币系统中，每一个非负整数的金额x都应该可以被表示出，即对每一个非负整数x，都存在n个非负整数**t[i]** 满足 **a[i]*t[i]**的和为 x。然而， 在网友的国度中，货币系统可能是不完善的，即可能存在金额x不能被该货币系统表示出。例如在货币系统**n=3 , a[2,5,9]**中，金额 1,3就无法被表示出来。

两个**货币系统 (n,a)和 (n,b)**是等价的，当且仅当对于任意非负整数x，它要么均可以被两个货币系统表出，要么不能被其中任何一个表出。

现在网友们打算简化一下货币系统。他们希望找到一个货币系统 (m,b)，满足 (m,b)与原来的货币系统(n,a)等价，且 m尽可能的小。他们希望你来协助完成这个艰巨的任务：**找到最小的m** 。

背包的基础类型

【题目07】 货币系统（NOIP2018)

【输入格式】输入文件的第一行包含一个整数 T，表示数据的组数。

接下来按照如下格式分别给出 T组数据。 每组数据的第一行包含一个正整数n 。接下来一行包含n 个由空格隔开的正整数a[i] 。

【输出格式】输出文件共有 T行，对于每组数据，输出一行一个正整数，表示所有与(n,a)等价的货币系统 (m,b)中，最小的 m。

样例输入

```
2
4
3 19 10 6
5
11 29 13 19 17
```

样例输出

```
2
5
```

样例解释

在第一组数据中，货币系统（2，[3,10]）和给出的货币系统(n,a) 等价，并可以验证不存在 m<2的等价的货币系统，因此答案为2 。

在第二组数据中，可以验证不存在 m<n 的等价的货币系统，因此答案为5 。

背包的基础类型

【题目07】 货币系统（NOIP2018)

【分析】dp[i]表示i面值最多能被几张钱表示

若其不能被表示dp[i]=-inf 能表示

且只有它自己则dp[i]=1 初始化dp[0]=0

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

```
dp[0]=0;
for(int i=1;i<=n;i++)
    for(int j=money[i];j<=maxx;j++)
        dp[j]=max(dp[j],dp[j-money[i]]+1);
for(int i=1;i<=n;i++)
    if(dp[money[i]]==1) ans++;
```

背包的基础类型

【题目08】 飞扬的小鸟（NOIP2016）

【问题描述】Flappy Bird 是一款风靡一时的休闲手机游戏。玩家需要不断控制点击手机屏幕的频率来调节小鸟的飞行高度，让小鸟顺利通过画面右方的管道缝隙。如果小鸟一不小心撞到了水管或者掉在地上的话，便宣告失败。

为了简化问题，我们对游戏规则进行了简化和改编:

游戏界面是一个长为 n，高为 m 的二维平面，其中有 k 个管道（忽略管道的宽度）。

小鸟始终在游戏界面内移动。小鸟从游戏界面最左边任意整数高度位置出发，到达游戏界面最右边时，游戏完成。

小鸟每个单位时间沿横坐标方向右移的距离为 1，竖直移动的距离由玩家控制。如果点击屏幕，小鸟就会上升一定高度 x，每个单位时间可以点击多次，效果叠加；如果不点击屏幕，小鸟就会下降一定高度 y。小鸟位于横坐标方向不同位置时，上升的高度 x 和下降的高度 y 可能互不相同。

小鸟高度等于 0 或者小鸟碰到管道时，游戏失败。小鸟高度为 m时，无法再上升。

现在,请你判断是否可以完成游戏。如果可以，输出最少点击屏幕数；否则，输出小鸟最多可以通过多少个管道缝隙。

背包的基础类型

【题目08】 飞扬的小鸟（NOIP2016）

【输入格式】第 1行有 3个整数 n,m,k，分别表示游戏界面的长度，高度和水管的数量，每两个整数之间用一个空格隔开；

接下来的 n 行，每行 2 个用一个空格隔开的整数 x 和 y，依次表示在横坐标位置 0~n−1上玩家点击屏幕后，小鸟在下一位置上升的高度 x，以及在这个位置上玩家不点击屏幕时，小鸟在下一位置下降的高度 y。

接下来 k行，每行 3 个整数 p,l,h，每两个整数之间用一个空格隔开。每行表示一个管道，其中 p 表示管道的横坐标，l表示此管道缝隙的下边沿高度，h表示管道缝隙上边沿的高度（输入数据保证 p各不相同，但不保证按照大小顺序给出）。

【输出格式】共两行。

第一行，包含一个整数，如果可以成功完成游戏，则输出 111，否则输出 000。

第二行，包含一个整数，如果第一行为 111，则输出成功完成游戏需要最少点击屏幕数，否则，输出小鸟最多可以通过多少个管道缝隙。

背包的基础类型

【题目08】 飞扬的小鸟（NOIP2016）

用f[i][j]表示**横坐标为i时高度为j的最少点击次数。**

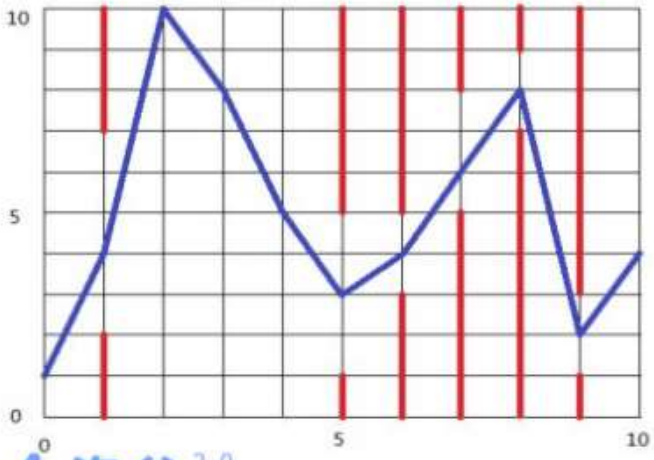
用正无穷来表示不可能达到这个状态。

于是我们可以分析出状态转移的方式：

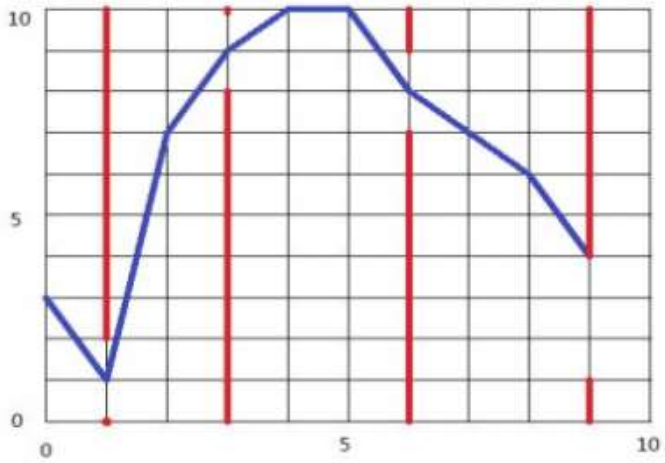
上升——**完全背包**转移方式

下降——**01背包**转移方式

超过m变为m——特判



输入输出样例1说明



输入输出样例2说明

如下图所示，蓝色直线表示小鸟的飞行轨迹，红色直线表示管道。

背包的基础类型

多重背包

多重背包的模型为：给定N种物品，其中第i种物品的体积为wi，价值为vi，每种物品有**Ci件**。有一体积为M的背包，请选择一些物品放入背包，使得物品总体积不超过M的前提下，物品的价值总和最大

阶段划分 **多重背包的仍然按已处理物品种数划分阶段，设阶段变量i代表前i种物品。**

设计状态 设f[i][j]代表从前i种物品选择若干恰好放入体积为j的背包的最大价值和（体积j必须装满）。

确定决策 对于状态f[i][j]，问题进入第i阶段，决策对象为第i种物品，做出的决策即第i种物品选几件，可以为0件、1件、2件、...、min(c[i],j/w[i])件。设决策变量为k，那么决策集合为{0<=k<=min(c[i],j/w[i])|k}。

转移方程 $f[i][j]=max(f[i-1][j-k\cdot w[i]]+k\cdot v[i])$

确定边界 **f[i][j]=-∞,f[0][0]=0**

确定目标 **max(f[n][j])**

```
1  f[0][0]=0;
2  for(int i=1;i<=n;i++){//枚举阶段
3      for(int j=0;j<=m;j++){//枚举状态
4          for(int k=0;k<=min(c[i],j/w[i]);k++)//枚举决策
5              f[i][j]=max(f[i][j],f[i-1][j-k*w[i]]+k*v[i]);
6          }
7  }
```

背包的基础类型

多重背包优化：二进制拆分

任意一种物品最多取num[i]件，除了从取1件、2件、3件这样循环一遍，有没有更高效的做法。

有，二进制拆分

【二进制拆分】：把第i种物品换成若干件物品，使得原问题中第i种物品可取的每种策略，取0..n[i]件——均能等价于取若干件代换以后的物品。另外，取超过n[i]件的策略必不能出现。

联想：任何一个十进制数都可以用一个二进制数表示。

比如，$7=4+2+1=(111)_2$

$14=8+4+2=(1110)_2$

类似地，取num[i]件物品的各种情况也可以用$\log_2$num[i]件物品表示出来。

背包的基础类型

多重背包优化：二进制拆分

【二进制拆分】：如何表示?

将第i种物品分成若干件物品，其中每件物品有一个系数，这件物品的费用和价值均是原来的费用和价值乘以这个系数。使这些系数分别为1,2,4,...,2^(k-1),n[i]-2^k+1，且**k是满足n[i]-2^k+1>0的最大整数。**

例如，如果n[i]为13，就将这种物品分成系数分别为1,2,4,6的四件物品。

分成的这几件物品的系数和为n[i]，表明不可能取多于n[i]件的第i种物品。另外这种方法也能保证对于0..n[i]间的每一个整数，均可以用若干个系数的和表示

背包的基础类型

多重背包优化：二进制拆分

【二进制拆分】：从2^0，2^1，2^2，…，2^{k-1}这k个2的整数幂种选出若干个相加，可以表示出0~2^k-1之间的任何整数。

进一步，我们求出满足$2^0+2^1+...+2^p<=t$的最大整数p，设$R=t-2^0-2^1-...-2^p$，那么：

① 根据p的最大性，有$2^0+2^1+...+2^{p+1}>t$，可推出$2^{p+1}>R$，因此2^0，2^1，…，2^p选出若干个相加可以表示出0~R之间的任何整数。

② 从$2^0+2^1+...+2^p$以及R之间选出若干个相加，可以表示出表示出R~$R+2^{p+1}-1$之间的任何整数；而根据R的定义，$R+2^{p+1}-1=Ci$，因此从2^0，2^1，…，2^p，R中选出若干个相加可以表示出1~C之间的任何整数。

如，t为9时

1+2+4（也就是$2^0+2^1+2^2$）<=7

1+2+4+8>7

因而，p为2，R为9-7=2

用1，2，4可以表示出1-7之间所有数字

在加上2，可以表示出8、9

同时，能够表示的数字最大不会超过9.

背包的基础类型

多重背包优化：二进制拆分

```
int f[40005],w[105],v[105],c[105];
int w1[200005],v1[200005],n1;
void pre(int vv,int ww,int cc){
    int num=1;
    while(cc-num>0){
        n1++;
        w1[n1]=ww*num;
        v1[n1]=vv*num;
        cc-=num;
        num*=2;
    }
    n1++;
    w1[n1]=ww*cc;
    v1[n1]=vv*cc;
}
```

```
int main(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>v[i]>>w[i]>>c[i];
        pre(v[i],w[i],c[i]);//将物品进行二进制拆分
    }
    //f[i][j]代表前i件物品选择若干放入体积为j的背包的最大价值和（体积j必须装满）。
    memset(f,0xcf,sizeof(f));
    f[0]=0;
    for(int i=1;i<=n1;i++){//用二进制拆分后的物品进行01背包
        for(int j=m;j>=w1[i];j--){//倒序枚举j
            f[j]=max(f[j],f[j-w1[i]]+v1[i]);
        }
    }
    int ans=0;
    for(int j=0;j<=m;j++){
        ans=max(ans,f[j]);
    }
    cout<<ans<<endl;
    return 0;
}
```

背包的衍生类型

混合背包

混合背包的模型为：给定N种物品，其中第i种物品的体积为wi，价值为vi，某种物品只有一件，某种物品有无数件，某种物品有c[i]件。有一体积为M的背包，请选择一些物品放入背包，使得物品总体积不超过M的前提下，物品的价值总和最大。

混合背包的阶段划分、状态设计与01背包一样，区别在于对于状态f[i][j]，需要讨论第i种物品是**01、完全**还是**多重**的，即**针对物品的属性不同做不同的决策**即可，这里不展开讨论。

背包的衍生类型

混合背包

```
for(int i=1;i<=n;i++)
{
    if(m[i]==-1)//完全背包
        for(int j=w[i];j<=V;j++)
            f[j]=max(f[j],f[j-w[i]]+v[i]);
    else//01与多重背包
    {
        int x=m[i];
        for(int k=1;k<=x;k<<=1)
        {
            for(int j=V;j>=w[i]*k;j--)
                f[j]=max(f[j],f[j-w[i]*k]+v[i]*k);
            x-=k;
        }
        if(x)
            for(int j=V;j>=w[i]*x;j--)
                f[j]=max(f[j],f[j-w[i]*x]+v[i]*x);
    }
}
```

背包的衍生类型

分组背包

分组背包的模型为：给定**N组**物品，其中第i组有c[i]件物品，第i组的第j件物品的体积为wij，价值为vij。有一体积为M的背包，请选择一些物品放入背包，使得每组**至多选择一件**且物品总体积不超过M的前提下，物品的价值总和最大。

阶段划分

可按已处理的组数将问题求解过程划分为N个阶段，设阶段变量i代表前i组。

设计状态

设f[i][j]代表从前i组物品选择若干恰好放入体积为j的背包的最大价值和。

确定决策

对于状态f[i][j]，问题进入第i阶段，决策对象为第i组物品，由于每组物品至多选1件，因此对第i组物品做出的决策为选0件、选1件，若是选1件，则需要枚举第i组的c[i]件物品，考虑选哪一件即可。

转移方程

$$f[i][j]=\max\begin{cases} f[i-1][j] & \text{第 i 组物品选 0 件} \\ f[i-1][j-w[i][k]]+v[i][k] & (j>=w[i][k]\ \text{第 i 组物品选 1 件，假设为第 k 件}) \end{cases}$$

确定边界

f[i][j]=-∞,f[0][0]=0

确定目标

max(f[n][j]) {0<=j<=m|j}

背包的衍生类型

分组背包

```
int ww,vv,c;
//m为背包容量，n为物品种数
cin>>m>>n;
for(int i=1;i<=n;i++){
    cin>>ww>>vv>>c;
    w[c].push_back(ww);
    v[c].push_back(vv);
    group[c]=1;
}
int n1=0;//n1代表组数
for(int i=1;i<=n;i++)if(group[c])n1++;
memset(f,0xcf,sizeof(f));
f[0]=0;
for(int i=1;i<=n1;i++){
    for(int j=m;j>=0;j--){
        for(int k=0;k<w[i].size();k++)
        if(j>=w[i][k])
        f[j]=max(f[j],f[j-w[i][k]]+v[i][k]);
    }
}
int ans=-1e9;
for(int j=0;j<=m;j++)ans=max(ans,f[j]);
cout<<ans<<endl;
```

背包的衍生类型

【题目09】最佳课题选择

【问题描述】Matrix67 要在下个月交给老师 n篇论文，论文的内容可以从 m个课题中选择。由于课题数有限，Matrix67 不得不重复选择一些课题。完成不同课题的论文所花的时间不同。具体地说，对于某个课题 i，若 Matrix67 计划一共写 x篇论文，则完成该课题的论文总共需要花费 Ai×xBi个单位时间。给定与每一个课题相对应的 Ai和 Bi的值，请帮助 Matrix67 计算出如何选择论文的课题使得他可以花费最少的时间完成这 n篇论文。

【输入格式】第一行两个整数 n和 m，分别代表需要完成的论文数和可供选择的课题数

接下来 m 行每行两个整数。其中，第 i行的两个数分别代表与第 i个课题相对应的时间系数 Ai和指数 Bi。

【输出格式】输出完成 n篇论文所需要耗费的最少时间。

【样例输入】

```
10 3
2 1
1 2
2 1
```

【样例输出】

```
19
```

背包的衍生类型

【题目09】最佳课题选择

c[i][j]表示a[i]这个课题被选择j次时对答案的贡献.

然后分组背包 dp[V]表示在V体积时的贡献.

转移的时候只考虑V-j时的答案贡献.

$$c[i][j] = a[i] \cdot j^{b[i]}$$

```
dp[0]=0;
for(int i=1;i<=n;i++)
    for(int v=m;v>=0;v--)
        for(int j=1;j<=v;j++)
        {
            dp[v]=min(dp[v],dp[v-j]+c[i][j]);
        }
```

背包的衍生类型

多维费用背包

多维费用背包的模型为：给定N种物品，其中第i种物品价值为vi，但是选择它会产生多个费用，分别记为w_1i,w_2i,...，物品的类型可能是01、完全、多重的。背包的体积也受多个费用限制，分别记为M_1,M_2,...，请选择一些物品放入背包，使得物品各项体积不超过M_1,M_2,...的前提下，物品的价值总和最大。

针对多维费用背包问题，阶段划分与普通背包问题一样，**只是在设计状态时增加费用维度**即可。设f[i][j_1][j_2][...]代表从前i种物品中选择若干恰好放入费用1体积为j_1，费用2体积为j_1，...的背包的最大价值和（体积j_1,j_2,...必须装满）即可。至于状态转移，取决于物品属性是01、完全还是多重背包。

背包的衍生类型

【题目10】看电影（题目来源：HDU3496)

【问题描述】新学期到了，朵朵明天要去上学，她决定今晚要玩得开心。朵朵很喜欢看动画片，因此她希望叔叔可以帮她买一些电影。但是爷爷只给了朵朵L分钟时间看动画片，之后她必须睡觉。朵朵列出了N部电影，编号1到N，这些电影都是她非常喜欢的，每一部电影有一个价值Vi（Vi>0），Vi值越高，代表朵朵越喜欢这部电影，每部电影播放时间为Ti分钟，并且如果朵朵选择观看某部电影，一定会看完。但是有个问题是商店很奇怪，只卖M部电影，因此叔叔必须帮朵朵从N部电影中挑出M部，使朵朵最喜欢，且播放时间不能超多L分钟。

【输入格式】第1行有1个整数t，表示测试数据组数，对于每组测试数据：

第一行有3个整数，分别是N（N<=100），M（M<=N），L（L<=1000）。

第2行到N+1行，每行有2个整数ti和vi，代表第i部电影的播放时间和价值。

【输出格式】对于每组测试数据，输出一个整数表示所选电影的最大价值（答案小于2^{31}）。

如果无法满足朵朵的需求，输出0。

样例输入	样例输出
1 3 2 10 11 100 1 2 9 1	3

背包的衍生类型

【分析】每部电影选择产生的费用有两个：一是花费了播放时间，二是占用了一部电影的名额；背包的体积也受两个费用的限制：一是总的播放时间L，二是总的影片数量M。因此，这是一个二维费用的背包问题。

//f[j][k]代表从前n部电影中选择j部且播放时间恰好为k的最大价值和（时间k必须用完）

```
while(q--){
    cin>>n>>m>>l;
    for(int i=1;i<=n;i++)cin>>t[i]>>v[i];
    memset(f,0xcf,sizeof(f));
    f[0][0]=0;
    for(int i=1;i<=n;i++){//枚举阶段
        for(int j=m;j>=1;j--){//倒序枚举背包体积j
            for(int k=l;k>=0;k--){//倒序枚举背包体积k
                if(k>=t[i])
                f[j][k]=max(f[j][k],f[j-1][k-t[i]]+v[i]);
            }
        }
    }
    int ans=0;
    for(int k=0;k<=l;k++){
        ans=max(ans,f[m][k]);
    }
    cout<<ans<<endl;
}
```

背包的衍生类型

有依赖性的背包

【题目11】金明的预算方案

n元钱，m件物品，附件买之前要先买主件。一个主件最多被2个附件依赖。

每个物品有价格和重要度，在钱数不超过n的情况下最大化价格*重要度。

```
1000 5//n和m
800 2 0
400 5 1
300 5 1
400 3 0
500 2 0
```

```
800 2 0
    依赖1 400 5 1
    依赖1 300 5 1
400 3 0
500 2 0
```

背包的衍生类型

有依赖性的背包

【题目11】金明的预算方案

对于主件和其附件组成的一个整体，决策有哪几种?

都不选，选主件，主件+附件1，主件+附件2，主件+附件1+附件2

考虑到所有这些策略都是互斥的（也就是说，你只能选择一种策略）。

v[i][0]表示主件重要度，v[i][1]附件1，v[i][2]附件2

W[i][0]表示主件价格， W[i][1]附件1， W[i][2]附件2

状态设计:

f(j)表示j元钱得到的最大值

f(j)=max{f(j),f(j-w0)+v0*w0, f(j-w0-w1)+v0*w0+v1*w1,

f(j-w0-w2)+v0*w0+v2*w2,f(j-w0-w1-w2)+w0v0+w1v1+w2v2}

背包的衍生类型

【题目12】Video Game Troubles（题目来源：USACO）

【问题描述】农夫约翰的奶牛们打游戏上瘾了！他发现奶牛们玩游戏之后比原先产更多的奶。约翰想要在给定的预算内购入一些游戏主机和一些游戏，使他的奶牛们生产最多的奶牛以养育最多的小牛。约翰考察了 N 种游戏主机，第 i 种主机的价格是 Pi，该主机有 Gi个游戏。很明显，奶牛必须先买进一种游戏主机，才能买进在这种主机上运行的游戏。在每种主机中，游戏 j 的价格为GPj，每头奶牛在玩了该游戏后的牛奶产量为PVj。农夫约翰的预算为 V。请帮助他确定应该买什么游戏主机和游戏，使得他能够获得的牛奶产出值的和最大。

【输入格式】第1行包含两个整数N和V，代表有N种主机，约翰的预算为V。

从第2~N+1行，第i+1行首先包含了一个整数为主机价格Pi，接下来有一个整数Gi代表该主机上的游戏数量，接下来包含Gi对数字，每一对数字代表第i台主机的第j款游戏的价格GPj和奶牛玩游戏之后的产奶量PVj。

【输出格式】输出一个整数，为约翰在预算内能获得的牛奶产出值的最大和。

【输入输出样例】

样例输入	样例输出
3 800 300 2 30 50 25 80 600 1 50 130 400 3 40 70 30 40 35 60	210

背包的衍生类型

【题目12】Video Game Troubles（题目来源：USACO）

思路：将每台主机及其游戏看做一个组，按**已处理的主机数进行阶段划分**，设阶段变量i代表已处理前i台主机，**设f[i][j]代表从前i台主机中恰好花费j元购买游戏得到的最大产奶量（j元必须用完）**。当问题进入第i阶段时，分两步进行。

【第一步】：先忽略第i台主机的价格，只对第i台主机的Gi款游戏做01背包。此时得到的f[i][j]为从前i台主机中花费j元购买游戏得到的**伪收益**（若是购买第i台主机的游戏，还没有算买第i台主机花的钱）。

【第二步】：再考虑第i台主机的价格，计算第i阶段的**真实收益f1[i][j]**。对于第i阶段，要么不买第i台主机及其游戏，此时f1[i][j]=f1[i-1][j]；要么购买第i台主机的若干款游戏，而购买第i台主机的若干款游戏的最优收益记录在f[i][j]里，但由于f[i][j]的计算忽略主机价格，因此真实收益要把主机价格减去，此时f1[i][j]=f[i][j-P[i]]。这就好比你花100元只买第i台主机的游戏的最大收益为500，假设第i台主机价格为100，那么你花200元时的真实收益等价于花100元只买游戏时得到的伪收益。

综上，维护两个状态，f[i][j]代表从前i台主机中恰好花费j元购买游戏得到的最大产奶量（j元必须用完，不考虑第i台主机价格的**伪收益**），f1[i][j]代表代表从前i台主机中恰好花费j元购买游戏得到的最大产奶量（j元必须用完，考虑第i台主机价格的**真实收益**）。

在第i阶段开始前，复制f[i][j]的值为f1[i-1][j]的值，即f[i][j]初值为第i-1阶段花费j元的最大产奶量（真实收益）；然后只对第i台主机的Gi款游戏做01背包计算第i阶段花费j元的最大产奶量f[i][j]（伪收益）；最后计算第i阶段花费j元的最大产奶量f1[i][j]（真实收益）

注意在对第i台主机的Gi款游戏做01背包时，是按照已处理的游戏数量划分阶段，但状态f[i][j]中省略了在对第i台主机的Gi款游戏做01背包时的阶段维度，因此体积j要倒序枚举。

背包的常见应用技巧

应用技巧1：背包问题演化之背包容量动态调整。

【题目13】 纪念品（NOIP2019）

【问题描述】小伟突然获得一种超能力，他知道未来 T 天 N 种纪念品每天的价格。某个纪念品的价格是指购买一个该纪念品所需的金币数量，以及卖出一个该纪念品换回的金币数量。

每天，小伟可以进行以下两种交易**无限次**：

1.任选一个纪念品，若手上有足够金币，以当日价格购买该纪念品；

2.卖出持有的任意一个纪念品，以当日价格换回金币。

每天卖出纪念品换回的金币可以**立即**用于购买纪念品，当日购买的纪念品也可以**当日卖出**换回金币。当然，一直持有纪念品也是可以的。

T天之后，小伟的超能力消失。因此他一定会在第 T天卖出**所有**纪念品换回金币。

小伟现在有 M 枚金币，他想要在超能力消失后拥有尽可能多的金币。

背包的常见应用技巧

应用技巧1：背包问题演化之背包容量动态调整。

【题目13】 纪念品（NOIP2019）

【输入格式】第一行包含三个正整数 T,N,MT，相邻两数之间以一个空格分开，分别代表未来天数 T，纪念品数量 N，小伟现在拥有的金币数量 M。

接下来 T 行，每行包含 N 个正整数，相邻两数之间以一个空格分隔。第 i行的 N个正整数分别为 Pi,1，Pi,2……,Pi,N，其中 Pi,j表示第 i天第 j种纪念品的价格。

【输出格式】

输出仅一行，包含一个正整数，表示小伟在超能力消失后最多能拥有的金币数量。

【输入样例】

```
6 1 100
50
20
25
 20
25
 50
```

【输出样例】

```
305
```

【输入输出样例 1 说明】最佳策略是：
第二天花光所有 100 枚金币买入 5 个纪念品 1；
第三天卖出 5 个纪念品 1，获得金币 125 枚；
第四天买入 6 个纪念品 1，剩余 5 枚金币；
第六天必须卖出所有纪念品换回 300 枚金币，第四天剩余 5 枚金币,共 305 枚金币。
超能力消失后，小伟最多拥有 305 枚金币。

背包的常见应用技巧

应用技巧1：背包问题演化之背包容量动态调整。

【题目13】 纪念品（NOIP2019）

【分析】把当日**手里的钱**当做背包的**容量**，把商品**当日价格**当成它的**消耗**，**价值为**把商品**两日价格差。**

假设dp[i]为用 i元钱去购买商品所能盈利的最大值（**不含成本**），p[i][j]为第i天商品j的价格

```
dp[k] = max(dp[k], dp[k-p[i][j]]+p[i+1][j]-p[i][j]);
```

注意：背包的容量在每天结束之后需要更新，每一天结束后把总钱数加上今天赚的钱

背包的常见应用技巧

应用技巧2：背包问题演化之物品价值动态变化。

【题目14】烹调方案（题目来源：洛谷P1417）

【问题描述】一共有n件食材，每件食材有三个属性，ai，bi和ci，你希望能在T时间内做出最美味的食物，但是这些食物美味程度的计算方式比较奇葩，烹饪第i件食材要花去ci的时间，如果在t时刻完成第i样食材的烹饪则得到ai-t*bi的美味指数。

请你设计一个烹饪方案使得所得到的食物美味指数最大。

【输入格式】第一行是两个正整数T和n，表示总的时间和食材个数。

下面一行n个整数，代表每件食材的属性ai；

下面一行n个整数，代表每件食材的属性bi；

下面一行n个整数，代表每件食材的属性ci。

【输出格式】输出食物的最大美味指数。

【输入输出样例】

样例输入	样例输出
100000 4 652 743 587 4926 6 5 3 4 4 5 3 6	6731

背包的常见应用技巧

应用技巧2：背包问题演化之物品价值动态变化。

【题目14】烹调方案（题目来源：洛谷P1417)

【分析】本问题中，烹饪一件食材得到的美味指数等于ai-t*bi，烹饪一件食材得到的美味度受ai、bi和烹饪完成时间三个因素的影响，对于所有食材思考应该如何排序。

若两件食材的属性分别是a1,b1,c1和a2,b2,c2，当前时间为t，假设先烹饪第一件食材再烹饪第二件食材比先烹饪第二件食材再烹饪第一件食材更优，因为需要得到最大的食物美味指数，则有a1-(t+c1)*b1+a2-(t+c1+c2)*b2>a2-(t+c2)*b2+a1-(t+c1+c2)*b1，化简该式得到：c1*b2<c2*b1，即如果两件食材满足这个条件，那么就应该先烹饪第一件食材再烹饪第二件食材。

我们首先按照条件c1*b2<c2*b1对所有食材进行排序后，该问题转化为01背包模板。按已烹饪食材数量划分阶段，设阶段变量i代表已烹饪前i件食材，设状态f[i][j]代表从前i件食材中选择若干恰好用时间j完成烹饪得到的最大美味指数（时间j必须用完），则f[i][j]=max(f[i-1][j],f[i-1][j-c[i]]+ai-j*bi)。

背包的常见应用技巧

应用技巧2：背包问题演化之物品价值动态变化。

【题目14】烹调方案（题目来源：洛谷P1417)

//f[j]代表用时间j恰好烹饪前i件食材得到的食物最大美味指数（时间j必须用完）

```
struct node{
    int a,b,c;
}food[55];
bool cmp(node x,node y){
    return x.c*y.b<y.c*x.b;
}
long long max(long long a,long long b){
    if(a>b)return a;else return b;
}
long long f[100005];
```

```
int main(){
    int T,n;
    cin>>T>>n;
    for(int i=1;i<=n;i++)cin>>food[i].a;
    for(int i=1;i<=n;i++)cin>>food[i].b;
    for(int i=1;i<=n;i++)cin>>food[i].c;
    sort(food+1,food+1+n,cmp);
    memset(f,0xcf,sizeof(f));
    f[0]=0;
    for(int i=1;i<=n;i++){
        for(int j=T;j>=food[i].c;j--){
            f[j]=max(f[j],f[j-food[i].c]+(long long)food[i].a-(long long)j*food[i].b);
        }
    }
    long long ans=0;
    for(int j=0;j<=T;j++){
        ans=max(ans,f[j]);
    }
    cout<<ans<<endl;
    return 0;
}
```

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

涉及的背包问题：**要求在背包容量（费用）的限制下求可以取到的最大价值**

如果要求的是“总价值最小”“总件数最小”，只需将状态转移方程中的max改成min即可。

“输出方案”

“求方案总数”

输出字典序最小的最优方案

“最优方案的总数”

“求次优解、第K优解”

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

输出方案

背包问题是要求一个最优值，如果要求输出这个最优值的方案

方法：记录下每个状态的最优值是由状态转移方程的哪一项推出来的，换句话说，记录下它是由哪一个策略推出来的。便可根据这条策略找到上一个状态，从上一个状态接着向前推即可。

01背包：转移方程为f[v]=max{f[v],f[v-w[i]]+c[i]}。

借助数组path[i][v]，设path[i][v]=0表示推出f[v]的值时是采用了方程的前一项（也即f[v]没变），path[i][v]表示采用了方程的后一项。注意这两项分别表示了两种策略：未选第i个物品及选了第i个物品。

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

【题目15】CD （UVA 624)

【问题描述】你即将开车出远门，当然希望在车上能聆听一些美好的音乐。你的车上只有播放录音带的设备，但是你最喜欢的音乐却都存放在CD上。所以你需要把CD上的音乐转录到录音带上。现在你必须解决的问题是：你的空白录音带长共N分钟，你如何选择CD上的歌使得尽可能的利用录音带的空间。以下是一些此问题的假设：

(1)CD上的歌最多不会超过20首。(2)没有任何一首歌的长度超过N分钟。

(3)要录在录音带上的歌不能重复。(4)每首歌的长度以一整数表达。

(5)N也是一个整数。

你必须找出该放哪些CD上的歌到录音带上（按CD上的顺序），使得录音带空白的空间最小。

【输入格式】多组数据，每行一个N，歌曲的数量，每首歌的时间长度

【输出格式】输出歌在CD上的顺序，以及他们的时间总和

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

【题目15】CD （UVA 624)

【样例输入】

5 3 1 3 4

10 4 9 8 4 2

20 4 10 5 7 4

90 8 10 23 1 2 3 4 5 7

45 8 4 10 44 43 12 9 8 2

【样例输出】

1 4 sum:5

8 2 sum:10

10 5 4 sum:19

10 23 1 2 3 4 5 7 sum:55 4 10

12 9 8 2 sum:45

【思考1】分析该题属于背包类型中的哪一类？ 物品？ 价值？ 容量?

【思考2】怎么记录是哪个物品对答案做了贡献?

【思考3】如何输出对答案做贡献的物品?

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

输出字典序最小方案

“字典序最小”是1..N号物品的选择方案排列出来以后字典序最小。

【分析】一般而言，求一个字典序最小的最优方案，只需要在转移时注意策略。首先，子问题的定义，如果存在一个选了物品1的最优方案，那么答案一定包含物品1，原问题转化为一个背包容量为v-c[1]，物品为2..N的子问题。反之，如果答案不包含物品1，则转化成背包容量仍为V，物品为2..N的子问题。不管答案怎样，子问题的物品以i..N而非前所述的1..i的形式来定义的，所以状态的定义和转移方程都要改一下。另一种方法是先把物品逆序排列一下，按物品已被逆序排列来叙述。

在这种情况下，可以按照前面经典的状态转移方程来求值，只是输出方案的时候要注意：从N到1输入时，如果**f[i][v]==f[i-1][v]及f[i][v]==f[i-1][v-c[i]]+w[i]**同时成立，应该按照后者（即选择了物品i）来输出方案。

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

【题目16】机器分配

【问题描述】总公司拥有高效设备M台，准备分给下属的N个分公司。各分公司若获得这些设备，可以为国家提供一定的盈利。问：如何分配这M台设备才能使国家得到的盈利最大？求出最大盈利值。其中M≤15，N≤10。分配原则：每个公司有权获得任意数目的设备，但总台数不超过设备数M。

【输入格式】第一行有两个数，第一个数是分公司数N，第二个数是设备台数M.接下来是一个N*M的矩阵，表明了第 I个公司分配 J台机器的盈利。

【输出格式】第1行为最大盈利值；第2到第n为第i分公司分x台

P.S.要求答案的字典序最小

【思考1】分析该题属于背包类型中的哪一类？ 物品？ 价值？ 容量?

【思考2】怎么记录是哪个物品对答案做了贡献?

【思考3】如何输出最小字典序方案?

输出字典序最小方案

【样例输入】

```
3 3
30 40 50
20 30 50
20 25 30
```

【样例输出】

```
70
1 1
2 1
3 1
```

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

求方案总数

对于一个给定了背包容量、物品费用、物品间相互关系（分组、依赖等）的背包问题，除了再给定每个物品的价值后求可得到的最大价值外，还可以得到装满背包或将背包装至某一指定容量的**方案总数**。

对于这类改变问法的问题，一般只需将状态转移方程中的max改成sum即可。

例如若每件物品均是完全背包中的物品

转移方程为：

f[i][v]=sum{f[i-1][v],f[i][v-c[i]]}

初始条件f[0][0]=1。

事实上，这样做可行的原因在于状态转移方程已经考察了所有可能的背包组成方案。

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

求方案总数

最优方案是指**物品总价值最大的方案。**结合**求最大总价值**和**方案总数**两个问题的思路

最优方案的总数：f[i][v]意义同前述，g[i][v]表示这个子问题的最优方案的总数，则在求f[i][v]的同时求g[i][v]的伪代码如下：

```
for i=1..N
    for v=0..V
        f[i][v]=max{f[i-1][v],f[i-1][v-c[i]]+w[i]}
        g[i][v]=0
       if(f[i][v]==f[i-1][v])
              inc(g[i][v],g[i-1][v]
      if(f[i][v]==f[i-1][v-c[i]]+w[i])
              inc(g[i][v],g[i-1][v-c[i]])
```

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

求方案总数

【题目17】钱币兑换问题 （HDU1284）

【问题描述】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。

【输入格式】每行只有一个正整数N，N小于32768。

【输出格式】对应每个输入，输出兑换方法数。

【输入样例】2934

12553

【输出样例】718831

13137761

分析：设dp[j]表示面值为j的总方案数

如果dp[j - a[i]] != 0则 dp[j] = dp[j] + dp[j - a[i]]

1 <= i <= n,a[i] <= j <= m。

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

求方案总数

【题目18】货币系统

【问题描述】给你一个n种面值的货币系统，求组成面值为m的货币有多少种方案。样例：设n=3，m=10，要求输入和输出的格式如下：

【样例输入1】money.in

```
3  10        //3种面值组成面值为10的方案
1         //面值1
2         //面值2
5         //面值5
```

【样例输出1】money.out

```
10          //有10种方案
```

【样例输入2】

```
5  20
3  2   8   7   1
```

【样例输出2】100

【样例输入3】

```
4  50
3  13  7  9
```

【样例输出3】17

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

求方案总数

【题目19】摆花（NOIP2012）

【问题描述】小明的花店新开张，为了吸引顾客，他想在花店的门口摆上一排花，共m盆。通过调查顾客的喜好，小明列出了顾客最喜欢的n种花，从1到n标号。

为了在门口展出更多种花，规定第i种花不能超过 ai盆，摆花时同一种花放在一起，且不同种类的花需按标号的从小到大的顺序依次摆列。

试编程计算，一共有多少种不同的摆花方案。

【输入格式】第一行包含两个正整数n和m，中间用一个空格隔开。

第二行有n个整数，每两个整数之间用一个空格隔开，依次表示a1,a2,a3,……an。

【输出格式】一个整数，表示有多少种方案。注意：因为方案数可能很多，请输出方案数对1000007取模的结果。

Input1

```
2 4
3 2
```

Output1

```
2
```

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

求方案总数

【题目19】摆花（NOIP2012）

【分析】

设dp[i][j]指摆到第i种花后总共摆了j盆的方案数

转移方程：dp[i][j]=dp[i-1][j-k]+dp[i][j]

第i种花由i-1种决定，k表示这种花摆多少要枚举（枚举范围0~a[i]），j−k表示没有用这k盆花装饰时的方案数。

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

求次优解、第K优解

对于求**次优解、第K优解**类的问题，如果相应的最优解问题能写出状态转移方程、用动态规划解决，那么求次优解往往可以相同的复杂度解决，第K优解则比求最优解的复杂度上多一个系数K。

基本思想：将每个状态都表示成有序队列，将状态转移方程中的max/min转化成有序队列的合并。以01背包为例说明。

01背包求最优解的状态转移方程：

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

如果要求第K优解，那么状态f[i][v]就应该是一个大小为K的数组f[i][v][1..K]。其中f[i][v][k]表示前i个物品、背包大小为v时，第k优解的值。 “f[i][v]是一个大小为K的数组” 可以简单地理解为在原来的方程中加了一维。显然f[i][v][1..K]这K个数是由大到小排列的，所以我们把它认为是一个有序队列。

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

【题目20】背包的第k优解

DD 和好朋友们要去爬山啦！他们一共有 K 个人，每个人都会背一个包。这些包的容量是相同的，都是 V。可以装进背包里的一共有 N 种物品，每种物品都有给定的体积和价值。

在 DD 看来，合理的背包安排方案是这样的：

1.每个人背包里装的物品的总体积恰等于包的容量。

2.每个包里的每种物品最多只有一件，但两个不同的包中可以存在相同的物品。

3.任意两个人，他们包里的物品清单不能完全相同。

在满足以上要求的前提下，所有包里的所有物品的总价值最大是多少呢?

【输入格式】第一行有三个整数：K、V、N。

第二行开始的 N 行，每行有两个整数，分别代表这件物品的体积和价值。

【输出格式】只需输出一个整数，即在满足以上要求的前提下所有物品的总价值的最大值。

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

【题目20】背包的第k优解

【输入样例】

2 10 5
3 12
7 20
2 4
5 6
1 1

【输出样例】

57

【样例说明】一种可以得到最优解的方案是：第一个人背体积为 7、2、1 的三种物品，价值为 25。第二个人背体积为 3、7 的两种物品，价值为 32。总价值 57。

【数据范围】保证总人数 K<=50。 每个背包的容量 V<=5000。 物品种类数 N<=200。其它正整数都不超过 5000。 输入数据保证存在满足要求的方案。

背包的常见应用技巧

应用技巧3： 背包问题演化之背包最优解延伸

【题目20】背包的第k优解

【分析】：f[j][u]表示空间为j时的第u优解;

初始化： f[j][u]=负无穷，f[0][1]=0

u的循环操作:

1.已知**f[j]=max(f[j-w[i]]+v[i],f[j]),**即f[j]有**两种**取值方案;

用两个数组**st1[u]**和**st2[u]**分别储存**f[j-w[i]][u]+v[i]**和**f[j][u]**;

2.因为**f[j][u]一定优于f[j][u+1],**所以st1[u]一定优于st1[u+1],st2[u]同理,所以只需要比较st1和st2中的元素就可以得到当前可以取到的最优值;

所以**f[i][u]等于st1[tail1]和st2[tail2]中较大的**;

3.因为不能有一样的方案,所以**st1[tail1],st2[tail2]**中较大的数被f[i][u]取值后,对应的**tail++**(tail在u循环开始前定义为0或者1);