正难则反－浅谈逆向思维在解题中的应用

绍兴市第一中学 唐文斌

【摘要】逆向思维是一种思考问题的方式，它有悖于通常人们的习惯，而正是这一特点，使得许多靠正常思维不能或是难于解决的问题迎刃而解。本文通过几个例子，总结了逆向思维在信息学解题中的应用。

【关键字】

逆向思维 容斥原理 参数搜索

二分 动态规划 记忆化

【正文】

引言

我们先看一个简单的问题：

平面上有四个点，构成一个边长为 1 的正方形。现在进行一种操作，每次可以选择两个点 A 和 B，把 A 关于 B 对称到 C，然后把 A 去掉。

求证：不可能经过有限次操作得到一个边长大于 1 的正方形

操作后的结果是相当复杂的，如果我们从正面着手，很难证明命题。不妨从反面来看问题：观察可以发现，每一步操作都是可逆的。即，我们如果可以把正方形变大，也可以把正方形变小。

证明：

不妨设四个顶点都是整点。

假设我们可以通过有限次操作得到一个边长大于 1 的正方形，那么我们把所有操作反过来对原正方形进行操作，我们可以得到一个边长小于 1 的正方形。

因为四个顶点都是整点，操作之后，点的坐标依然是整数。所以我们得到一个边长小于 1 且四个点都是整点的正方形。这显然不可能。

所以假设不成立。命题得证。

上面的例子说明了逆向思维在数学问题中的应用。山重水复疑无路，应用逆向思维，换个角度看问题，便柳暗花明又一村了。

例一、Dinner Is Ready[1]

题目大意：

妈妈烧了 M 根骨头分给 n 个孩子们，第 i 个孩子有两个参数 Min_i 和 Max_i，分别表示这个孩子至少要得到 Min_i 根骨头，至多得到 Max_i 根骨头。

输入：

第一行包含两个数 $n(0<n\leqslant 8)$, $M(0<M)$，表示孩子数量和骨头的数量。

接下来 n 行分别输入 Min_i 和 Max_i $(0\leqslant Min_i\leqslant Max_i\leqslant M)$

输出：

输出一个整数，表示妈妈有多少种分配方案(骨头不能浪费，必须都分给孩子们）

初步分析：

该题模型很简单，即求如下方程组的整数解的个数：

$$\begin{cases}\sum_{i=1}^{n} X_i = M \\ Min_1 \le X_1 \le Max_1 \\ Min_2 \le X_2 \le Max_2 \\ Min_3 \le X_3 \le Max_3 \\ \cdots\cdots \\ Min_n \le X_n \le Max_n\end{cases}$$

我们知道，方程组简单形式 $\begin{cases}\sum_{i=1}^{n} X_i = M \\ X_i \ge 0\end{cases}$ 的整数解个数为 C_{M+n-1}^{n-1}[2]

设 $Y_i = X_i + Min_i$ ，则原方程组转化为 $\begin{cases}\sum_{i=1}^{n} Y_i = M - \sum_{i=1}^{n} Min_i \\ 0 \le Y_1 \le Max_1 - Min_1 \\ 0 \le Y_2 \le Max_2 - Min_2 \\ 0 \le Y_3 \le Max_3 - Min_3 \\ \cdots\cdots \\ 0 \le Y_n \le Max_n - Min_n\end{cases}$

对于下界限制，我可以通过换元得到简单形式，但是因为有上界的限制，我们似乎还无法直接计算出答案。

应用逆向思维：

[1] Zhejiang University Online Judge 1442(acm.zju.edu.cn)

[2] 这是一个很经典的组合问题，可由隔板原理得出答案。

设 S 为全集，表示满足 $X_i \geqslant Min_i$ 的整数解集。

设 S_i 为 S 中满足约束条件 $X_i \leqslant Max_i$ 的整数解的集合，$\overline{S_i}$ 为 S_i 在 S 中的补集，即满足 $X_i > Max_i$。

$|S_i|$ 无法计算，但是，$|\overline{S_i}|$ 可解!!! 我们希望把 $|S_i|$ 的计算转化到可解的 $|\overline{S_i}|$。

于是：

$$Answer = |S_1 \cap S_2 \cap S_3 ... \cap S_n| =$$
$$|S| - (|\overline{S_1}| + |\overline{S_2}| + ... + |\overline{S_n}|) + (|\overline{S_1}| \cap |\overline{S_2}| + |\overline{S_1}| \cap |\overline{S_3}| + ... + |\overline{S_{n-1}}| \cap |\overline{S_n}|) - .... + (-1)^n * |\overline{S_1} \cap \overline{S_1} \cap ... \cap \overline{S_n}|$$

这是一种容斥原理的形式。至此，问题已经解决。我们应用逆向思维，在原集合的模 $|S_i|$ 不可解的情况下，通过可解的 $|\overline{S_i}|$ 得到答案。并给出了一个基于容斥原理的算法。时间复杂度为 $O(2^n * (n+M))$。

例二、Greedy Path[1]

题目大意：

有 n 个城市，被 m 条路连接着。最近成立了一些旅行社，在这些城市之间给旅行者们提供服务。旅行者从城市 i 到城市 j 需要付给旅行社的费用是 $C_{i,j}$，需要的时间为 $T_{i,j}$ 。很多旅行者希望加入旅行社，但是旅行社只有一辆车。于是旅行社的老板决定组织一次旅行大赚一笔。公司里的专家需要提供一条使得贪心函数 $F(G)$最大的回路 G。$F(G)$ 等于总花费除以总时间。但是没有人找到这样的回路，于是公司的领导请你帮忙。

输入：

第一行包含两个数 $n(3 \leqslant n \leqslant 50), m$ 分别表示点数和边数。

接下来 m 行每行包含一条路的描述。输入四个数$A,B,C_{A,B},T_{A,B}$（$0 \leqslant C_{A,B} \leqslant 100$，$0 \leqslant T_{A,B} \leqslant 100$）

输出：

如果不存在这样的路，输出 0。

否则输出回路中包含的城市个数，然后依次输出通过的城市的顺序。如果有很多条这样的路，输出任意一条。

初步分析：

题目要求是求一条回路，但不是边权和最大或者最小，所以我们不能直接使用经典算法。

设 $G = (V, E)$，S 为 G 中所有回路 $C = (V', E')$组成的集合。

我们的目标是找到集合 S 中的一条回路使得 $F(C)$取到最大值：

$$F(C) = \frac{\sum_{e \in E'} C_e}{\sum_{e \in E'} T_e}$$

应用逆向思维：

[1] Saratov State University Online Judge 236(acm.sgu.ru)

如果我们知道 $C^*=(V^*,E^*)\in S$ 是一条最优回路，那么

$$F(C^*)=\frac{\sum_{e\in E^*}C_e}{\sum_{e\in E^*}T_e}\Rightarrow\sum_{e\in E^*}C_e-F(C^*)\times\sum_{e\in E^*}T_e=0$$

于是我们定义函数 $o(t):o(t)=\max_{C=(V',E')\in S}\sum_{e\in E'}C_e-t\sum_{e\in E'}T_e$

我们做一个猜想：如果有 $o(t^*)=0$，那么存在 $C^*=(V^*,E^*)\in S$ 满足

$$t^*=\frac{\sum_{e\in E^*}C_e}{\sum_{e\in E^*}T_e}$$

我们认为 C^*就是一条最优回路。

证明：假设存在另一条回路 $C_1=(V_1,E_1)\in S$ 更优，则：

$$t1=\frac{\sum_{e\in E_1}C_e}{\sum_{e\in E_1}T_e}>t^*\Leftrightarrow 0<\sum_{e\in E1}C_e-t^*(\sum_{e\in E1}T_e)\le o(t^*)$$

但是这与 $o(t^*)=0$ 矛盾。所以 C*是一条最优回路。

如果 t^*是最优答案，则存在：

$$\begin{cases}o(t)=0\Leftrightarrow t=t^*\\ o(t)<0\Leftrightarrow t>t^*\\ o(t)>0\Leftrightarrow t<t^*\end{cases}$$ （性质一）

于是我们就得到了算法：

我们从一个包含 t^*的区间(t_l,t_h)开始。（例如 tl = $\min_{e\in E}\frac{C_e}{T_e}$, th = $\max_{e\in E}\frac{C_e}{T_e}$ ）

每一次，选取 t_l 与 t_h的中点 t，计算 $O(t)$。

O(t)的计算方法：

对于边 $e\in E$，设一个新的参数 $W_e=C_e-t^*T_e$

用 *Floyd* 算法计算有向图的最大权值环。该最大权值即 $O(t)$。

根据性质一，更新 t_l 或 t_h，得到新的上下界，继续二分，直到达到精度要求。

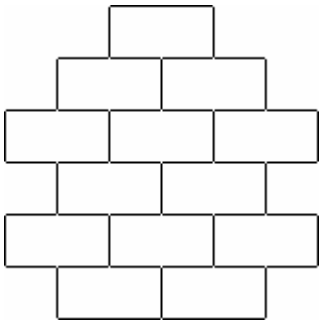
假设二分的次数为 K，则算法的时间复杂度为 $O(K*n^3)$。这虽然不是一个严格的多项式算法，但是对于题目给定的范围，该算法可以很快地求出答案。

例三、Building Towers[1]

题目大意：

有 N 块积木，我们需要用这些积木造塔。每个塔有 *H* 层，最底层包含 *M* 块积木；对于上面的每一层，包含的积木块数必须比下面一层的多 1 或者少 1。

下图是一个合法的塔(*H*=6 , *M*=2 , *N*=13) ：



你的任务是计算一共可以建造出多少种不同的塔（注意积木不一定要全部用完）。两个塔被认为不同，仅当存在一个 *i*（1≤*i*≤*H*），两个塔的第 *i* 层包含不同数目的积木。

我们用 *H* 个正整数来表示一个塔的结构（从底部到顶部）。你还需要输出字典序第 K 小的建塔方案。

输入：

第一行包含三个整数,*N*,*H*,*M* (*N*≤32767, *H*≤60, *M*≤10)。

接下来每行包含一个正整数 *K*（除了最后一行）。最后一行包含一个-1 表示输入的结束。

输出：

第一行包含可以建造的不同的塔的总数。

接下来每一行输出对应的字典序第 *K* 小的塔的结构。

初步分析：

首先，看到题目，思维的惯性让我们下意识的想到了经典、传统的动态规划算法。

用 *F*[*i*,*j*,*k*]表示当前层有 *i* 块砖，还剩下 *j* 层，可用的砖块数量为 *k* 的方案总数。可以写出动态规划方程为：

F[*i*,*j*,*k*] = *F*[*i*+1,*j*+1,*k*-*i*] + *F*[i-1,*j*+1,*k*-*i*]

边界条件: *F*[*M*,*H*,*N*]=1 , 其他为 0

很容易看出，这个动态规划方程一共涉及了 *N***M***K* 个状态，最大约为 60*70*2400 = 10*M*。如果每个状态用一个 *double* 来保存信息，则至少需要 80*M* 的内存（注意，因为我们需要输出字典序第 *K* 小的塔，所以我们无法使用滚动储存的优化）。不论是时间复杂度还是空间复杂度，都让人难以接受。至此，动态规划算法失败了。

应用逆向思维：

自底向上的动态规划在时空上都难以让人承受。我们不妨在处理时“反个方向”，看看自顶向下的搜索算法。

搜索算法向来是“低效”、“指数”的代名词。然而，如果加入记忆化因子，则正好是一个“逆向动态规划”的过程。

[1] Ural Online Judge 1148 (acm.timus.ru)

将动态规划的顺序作一个反转，其好处在于：

首先，自顶向下的看问题，有“一览众山小”的开阔视野，可以顺利地为如何搜，先搜什么后搜什么，以及如何剪枝等作合理的布局。在本题中，这一点体现为搜索过程不会搜索到很多荒唐的状态，例如砖块数目明显不够，奇偶性差异等等。而在一些最优化的问题中，搜索记忆化的方法可以有效地配合最优性剪枝，避免许多明显没有价值的状态，而这正是正向的动态规划所不能企及的。

第二，在类似于本题的计数问题中，可以采用部分记忆化的方法，即只记录那些比较容易被多次搜索到的状态。这样一来，减少了存储的状态量，加快了查询的速度。

实现的时候，我们用三元组（N，H，M）来表示一个状态，即当前层有 M 块砖，还剩下 H 层，可用的砖块数量为 N。$F(N,H,M)$表示该状态下的方案数。用 *Hash* 表存储，并加入以下一些优化：

1） 可以事先计算出当前状态最多和最少还需要的砖块数目。如果 N 小于最小的需求量，那么显然答案为 0；如果 N 大于最大需求量，那么可以把 N 设置为该最大需求量，以避免等同状态的重复存储。

2） 如果 N 不小于最大需求量且 $H \leqslant M$，则答案为 2^H。

3） 为了减少存储的状态量，我们设一个存储上界，即仅当 $F(N,H,M)$不大于该上界时才将其存入 *Hash* 表。为什么这样做是有效的呢？观察可以发现，搜索的状态树是以指数的形式往下展开的。也就是说，在树越深的地方，状态量会越多，进入查询也越频繁，而接近根的地方，状态的分布比较稀疏，当 $F(N,H,M)$较大时，访问它们的频率就会相应的降低。所以我们可以重复搜索这些状态，从而减少存储状态量，加快 *Hash* 表检索速度。

我们不妨来看一下“逆向动态规划”的表现：

N	H	M	正向动态规划所需要处理的状态数量	逆向动态规划中 *Hash* 表中所存储的状态量	正逆向规划中状态数目的比值
1000	40	10	1200000	3055	392.80
1500	50	10	2625000	5371	488.74
1200	60	10	2880000	49875	57.74
1500	60	10	3600000	38018	94.69

即便是在最坏情况下，逆向动态规划也只需要处理大约 1/50 的状态。

至此，我们已经完美地解决了这个问题。

【总结】

例一，是一种补集转化的思想，我们对原集合无从下手，转而去求原集合的补集，从反面看问题，得到答案。

例二，我们没有去看问题的反面，而是去看一个事实的反面。我们现在不知道答案，如果知道答案又将如何呢？

例三，在经典的动态规划无法解决问题的情况下，我们将规划顺序颠倒，得到了逆向动态规划的方法，从而避免许多无效状态，在时间和空间上都取得了质的飞跃。

上述三个例子，都是逆向思维的一些简单应用。“正难则反”的逆向思维，帮助我们打破思维定势，以退为进，避其锋芒，攻其软肋。让我们在山重水复疑无路时柳暗花明又一村；在踏破铁鞋无觅处时得来全不费功夫；让我们在为伊消得人憔悴后蓦然发现那人却在灯火阑珊处。所谓反弹琵琶成新曲，愿逆向思维助你一臂之力，更上一层楼！

【感谢】

特别感谢安徽芜湖一中周源同学的帮助，感谢所有帮助过我的人。
Acm.sgu.ru
Acm.timus.ru
Acm.zju.edu.cn

【参考文献】

刘汝佳，黄亮《算法艺术与信息学竞赛》 清华大学出版社
Gunnar W. Klau 等《The Fractional Prize-Collecting Steiner Tree Problem On Trees》
任初农《说说逆向思维》

【附件】

例三《Building Tower》一题的源程序 Tower.cpp

【附录】

[例一原题]

Dinner Is Ready

Dinner is ready! WishingBone's family rush to the table. WishingBone's mother has prepared many delicious bones. WishingBone has several brothers, namely WishingTwoBones, WishingThreeBones. WishingBone wants at least one bone, WishingTwoBones two and WishingThreeBones three. :-P But as they are not too greedy (or they want to keep shape), they decide that they want at most twice of their minimal requirement. Now let's suppose mother has prepared 7 bones, one way to distribute them is 2-2-3, the other two ways are 1-3-3 and 1-2-4. Of course mother doesn't want to waste any bones, so if she had prepared 13 bones, she would end up without any feasible distribution.

As curious as WishingBone, mother wants to know how many different ways she can distribute those bones.

Input

The first line of input is a positive integer N <= 100, which is the number of test cases.

The first line of each case contains two integers n and m (0 < n <= 8, 0 < m), where n is the number of children and m is the number of bones mother has prepared.

The next n lines have two integers mini and maxi (0 <= mini <= maxi <= m), which is the minimal and maximal requirement of the ith child.

Output

N integers which are the numbers of ways to distribute the m bones, one per line.

Sample Input

```
2
3 7
1 2
2 4
3 6
3 13
1 2
2 4
3 6
```

Sample Output

```
3
0
```

[例二原题]

Greedy Path

There are N towns and M routes between them. Recently, some new travel agency was founded for servicing tourists in these cities. We know cost which tourist has to pay, when traveling from town i to town j which equals to Cij and time needed for this travel - Tij. There are many tourists who want to use this agency because of very low rates for travels, but agency still has only one bus. Head of agency decided to organize one big travel route to gain maximal possible amount of money. Scientists of the company offer to find such a cyclic path G, when greedy function f(G) will be maximum. Greedy function for some path is calculated as total cost of the path (sum of Cij for all (i,j) - routes used in path) divided by total time of path (similar to Cij). But nobody can find this path, and Head of the company asks you to help him in solving this problem.

Input

There are two integers N and M on the first line of input file (3<=N<=50). Next M lines contain routes information, one route per line. Every route description has format A, B, Cab, Tab, where A is starting town for route, B - ending town for route, Cab - cost of route and Tab - time of route (1<=Cab<=100; 1<=Tab<=100; A<>B). Note that order of towns in route is significant - route (i,j) is not equal to route (j,i). There is at most one route (in one direction) between any two towns.

Output

You must output requested path G in the following format. On the first line of output file you must output K - number of towns in the path (2<=K<=50), on the second line - numbers of these towns in order of passing them. If there are many such ways - output any one of them, if there are no such ways - output "0" (without quotes).

Sample test(s)

Input

4 5
1 2 5 1
2 3 3 5
3 4 1 1
4 1 5 2
2 4 1 10

Output

4
1 2 3 4

[例三原题]

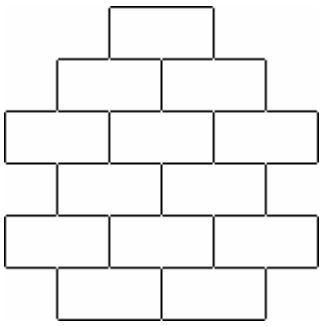
Building Towers

Time Limit: 2 second

Memory Limit: 1000 KB

There are N cubes in a toy box which has 1-unit height, the width is double the height. The teacher organizes a tower-building game. The tower is built by the cubes. The height of the tower is H (h levels). The bottom of the tower contains M cubes; and for all above level, each must contains a number of cubes which is exactly 1 less than or greater than the number of cubes of the level right below it.

Below is an example of a tower of H=6, M=2, N=13.



Your task is to determine how many different towers can be there. Two towers are considered different if there is at least one number i (1< i <=H) so that the i'th level of one tower contains a different number of cubes to the i'th level of the other tower.

Each tower is represented by an array of H positive integers which determines the structure of the tower from the bottom level up to the top level. Those arrays are sorted ascendingly.

Input

1st line contains three positive number N, H and M (N <= 32767, H<=60, M<=10).

Each following line (except the last)contains an integer k. The last line contains number -1.

Output

1st line is the total of different towers that can be founded.

Each following line contains an array that represents the structure of the tower corresponding to the number K from input.

Sample Input

22 5 4
1
10
-1

Sample Output

10
4 3 2 1 2
4 5 4 5 4

Hint

Numbers which are on the same line should be separated by at least one space.

You don't have to use all N cubes.