Cutting Sticks

You have to cut a wood stick into pieces. The most affordable company, The Analog Cutting Machinery, Inc. (ACM), charges money according to the length of the stick being cut. Their procedure of work requires that they only make one cut at a time.

It is easy to notice that different selections in the order of cutting can led to different prices. For example, consider a stick of length 10 meters that has to be cut at 2, 4 and 7 meters from one end. There are several choices. One can be cutting first at 2, then at 4, then at 7. This leads to a price of 10 + 8 + 6 = 24 because the first stick was of 10 meters, the resulting of 8 and the last one of 6. Another choice could be cutting at 4, then at 2, then at 7. This would lead to a price of 10 + 4 + 6 = 20, which is a better price.

Your boss trusts your computer abilities to find out the minimum cost for cutting a given stick.

Input

The input will consist of several input cases. The first line of each test case will contain a positive number l that represents the length of the stick to be cut. You can assume $l < 1000$. The next line will contain the number n ($n < 50$) of cuts to be made.

The next line consists of n positive numbers c_i ($0 < c_i < l$) representing the places where the cuts have to be done, given in strictly increasing order.

An input case with $l = 0$ will represent the end of the input.

Output

You have to print the cost of the optimal solution of the cutting problem, that is the minimum cost of cutting the given stick. Format the output as shown below.

Sample Input

```
100
3
25 50 75
10
4
4 5 7 8
0
```

Sample Output

```
The minimum cutting is 200.
The minimum cutting is 22.
```

Miguel Revilla
2000-08-21

Shoemaker's Problem

Shoemaker has N jobs (orders from customers) which he must make. Shoemaker can work on only one job in each day. For each i_{th} job, it is known the integer T_i ($1<=T_i<=1000$), the time in days it takes the shoemaker to finish the job. For each day of delay before starting to work for the i_{th} job, shoemaker must pay a fine of S_i ($1<=S_i<=10000$) cents. Your task is to help the shoemaker, writing a programm to find the sequence of jobs with minimal total fine.

The Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

First line of input contains an integer N ($1<=N<=1000$). The next N lines each contain two numbers: the time and fine of each task in order.

The Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

You programm should print the sequence of jobs with minimal fine. Each job should be represented by its number in input. All integers should be placed on only one output line and separated by one space. If multiple solutions are possible, print the first lexicographically.

Sample Input

```
1

4
3 4
1 1000
2 2
5 5
```

Sample Output

```
2 1 3 4
```

Alex Gevak

September 16, 2000(Revised 4-10-00, Antonio Sanchez)

Stacking Boxes

Background

Some concepts in Mathematics and Computer Science are simple in one or two dimensions but become more complex when extended to arbitrary dimensions. Consider solving differential equations in several dimensions and analyzing the topology of an *n*-dimensional hypercube. The former is much more complicated than its one dimensional relative while the latter bears a remarkable resemblance to its ``lower-class'' cousin.

The Problem

Consider an *n*-dimensional ``box'' given by its dimensions. In two dimensions the box (2,3) might represent a box with length 2 units and width 3 units. In three dimensions the box (4,8,9) can represent a box $4 \times 8 \times 9$ (length, width, and height). In 6 dimensions it is, perhaps, unclear what the box (4,5,6,7,8,9) represents; but we can analyze properties of the box such as the sum of its dimensions.

In this problem you will analyze a property of a group of *n*-dimensional boxes. You are to determine the longest *nesting string* of boxes, that is a sequence of boxes $b_1, b_2, \ldots, b_k$ such that each box b_i nests in box b_{i+1} ($1 \leq i < k$).

A box D = ($d_1, d_2, \ldots, d_n$) nests in a box E = ($e_1, e_2, \ldots, e_n$) if there is some rearrangement of the d_i such that when rearranged each dimension is less than the corresponding dimension in box E. This loosely corresponds to turning box D to see if it will fit in box E. However, since any rearrangement suffices, box D can be contorted, not just turned (see examples below).

For example, the box D = (2,6) nests in the box E = (7,3) since D can be rearranged as (6,2) so that each dimension is less than the corresponding dimension in E. The box D = (9,5,7,3) does NOT nest in the box E = (2,10,6,8) since no rearrangement of D results in a box that satisfies the nesting property, but F = (9,5,7,1) does nest in box E since F can be rearranged as (1,9,5,7) which nests in E.

Formally, we define nesting as follows: box D = ($d_1, d_2, \ldots, d_n$) *nests* in box E = ($e_1, e_2, \ldots, e_n$) if there is a permutation π of $1 \ldots n$ such that ($d_{\pi(1)}, d_{\pi(2)}, \ldots, d_{\pi(n)}$) ``fits'' in ($e_1, e_2, \ldots, e_n$) i.e., if $d_{\pi(i)} < e_i$ for all $1 \leq i \leq n$.

The Input

The input consists of a series of box sequences. Each box sequence begins with a line consisting of the the number of boxes k in the sequence followed by the dimensionality of the boxes, n (on the same line.)

This line is followed by k lines, one line per box with the n measurements of each box on one line separated by one or more spaces. The i^{th} line in the sequence ($1 \leq i \leq k$) gives the measurements for the i^{th} box.

There may be several box sequences in the input file. Your program should process all of them and determine, for each sequence, which of the k boxes determine the longest nesting string and the length of that nesting string (the number of boxes in the string).

In this problem the maximum dimensionality is 10 and the minimum dimensionality is 1. The maximum number of boxes in a sequence is 30.

The Output

For each box sequence in the input file, output the length of the longest nesting string on one line followed on the next line by a list of the boxes that comprise this string in order. The ``smallest'' or ``innermost'' box of the nesting string should be listed first, the next box (if there is one) should be listed second, etc.

The boxes should be numbered according to the order in which they appeared in the input file (first box is box 1, etc.).

If there is more than one longest nesting string then any one of them can be output.

Sample Input

```
5 2
3 7
8 10
5 2
9 11
21 18
8 6
5 2 20 1 30 10
23 15 7 9 11 3
40 50 34 24 14 4
9 10 11 12 13 14
31 4 18 8 27 17
44 32 13 19 41 19
1 2 3 4 5 6
80 37 47 18 21 9
```

Sample Output

```
5
3 1 2 4 5
```

4
7 2 5 6

Problem A

The Poor Giant

Input: Standard Input

Output: Standard Output

Time Limit: 1 second

On a table, there are n apples, the i-th apple has the weight k+i(1<=i<=n). Exactly one of the apples is sweet, lighter apples are all bitter, while heavier apples are all sour. The giant wants to know which one is sweet, the only thing he can do is to eat apples. He hates bitter apples and sour apples, what should he do?
For examples, n=4, k=0, the apples are of weight 1, 2, 3, 4. The gaint can first eat apple #2.
if #2 is sweet, the answer is #2
if #2 is sour, the answer is #1
if #2 is bitter, the answer might be #3 or #4, then he eats #3, he'll know the answer regardless of the taste of #3
The poor gaint should be prepared to eat some bad apples in order to know which one is sweet. Let's compute the total weight of apples he must eat in all cases.
#1 is sweet: 2
#2 is sweet: 2
#3 is sweet: 2 + 3 = 5
#4 is sweet: 2 + 3 = 5
The total weights = 2 + 2 + 5 + 5 = 14.
This is not optimal. If he eats apple #1, then he eats total weight of 1, 3, 3, 3 when apple #1, #2, #3 and #4 are sweet respectively. This yields a solution of 1+3+3+3=13, beating 14. What is the minimal total weight of apples in all cases?

Input

The first line of input contains a single integer t(1<=t<=100), the number of test cases. The following t lines each contains a positive integer n and a non-negative integer k(1<=n+k<=500).

Output

For each test case, output the minimal total weight in all cases as shown in the sample output.

Sample Input **Sample Output**

Problem setter: Rujia Liu, Member of Elite Problemsetters' Panel

Problem G
e-Coins

Input: standard input
Output: standard output
Time Limit: 10 seconds
Memory Limit: 32 MB

At the Department for Bills and Coins, an extension of today's monetary system has newly been proposed, in order to make it fit the new economy better. A number of new so called e-coins will be produced, which, in addition to having a value in the normal sense of today, also have an InfoTechnological value. The goal of this reform is, of course, to make justice to the economy of numerous dotcom companies which, despite the fact that they are low on money surely have a lot of **IT** inside. All money of the old kind will keep its conventional value and get zero InfoTechnological value.

To successfully make value comparisons in the new system, something called the e-modulus is introduced. This is calculated as **SQRT(X*X+Y*Y)**, where **X** and **Y** hold the sums of the conventional and InfoTechnological values respectively. For instance, money with a conventional value of **$3** altogether and an InfoTechnological value of **$4** will get an e-modulus of **$5**. Bear in mind that you have to calculate the sums of the conventional and InfoTechnological values separately before you calculate the e-modulus of the money.

To simplify the move to e-currency, you are assigned to write a program that, given the e-modulus that shall be reached and a list of the different types of e-coins that are available, calculates the smallest amount of e-coins that are needed to exactly match the e-modulus. There is no limit on how many e-coins of each type that may be used to match the given e-modulus.

Input

A line with the number of problems **n (0<n<=100)**, followed by **n** times:

- A line with the integers **m (0<m<=40)** and **S (0<S<=300)**, where **m** indicates the number of different e-coin types that exist in the problem, and **S** states the value of the e-modulus that shall be matched exactly.
- **m** lines, each consisting of one pair of non-negative integers describing the value of an e-coin. The first number in the pair states the conventional value, and the second number holds the InfoTechnological value of the coin.

When more than one number is present on a line, they will be separated by a space. Between each problem, there will be one blank line.

Output

The output consists of **n** lines. Each line contains either a single integer holding the number of coins necessary to reach the specified e-modulus **S** or, if **S** cannot be reached, the string **"not possible"**.

Sample Input:

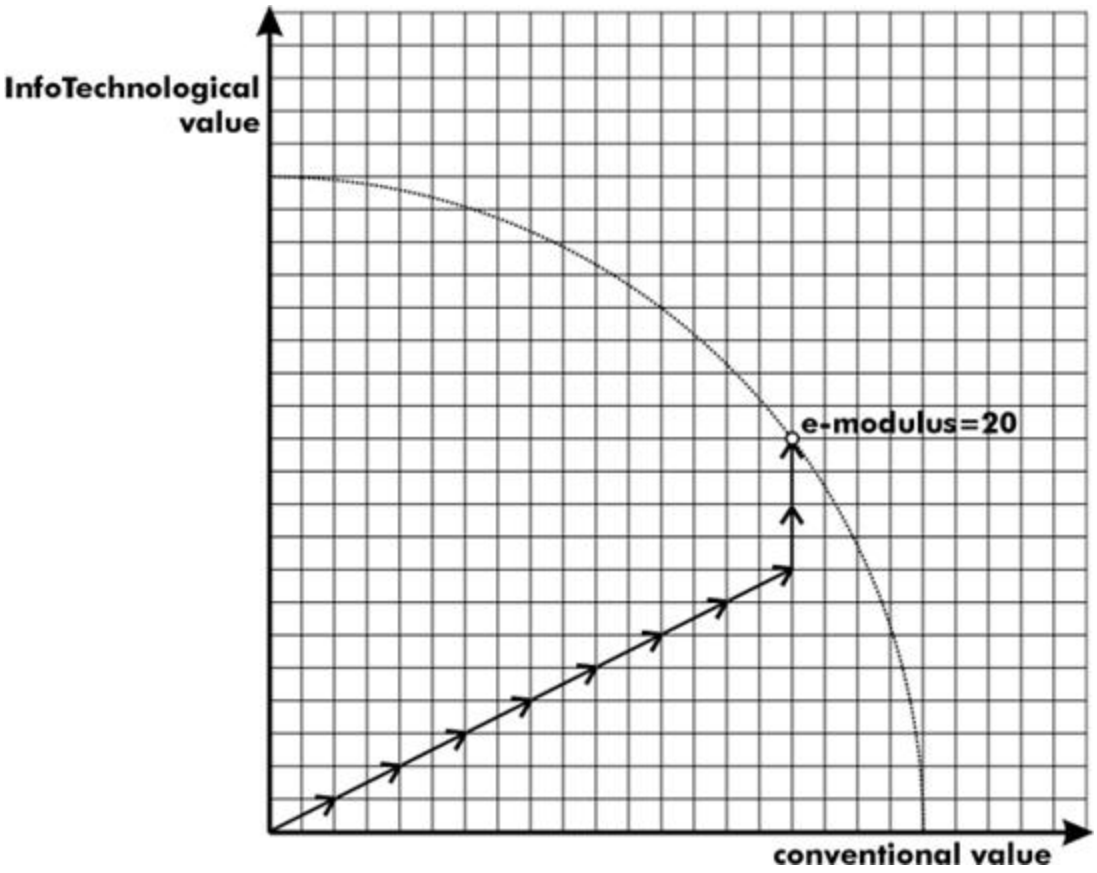
```
3
2 5
0 2
2 0
3 20
0 2
```

```
2 0
2 1
3 5
3 0
0 4
5 5
```

Sample Output:

```
not possible
10
2
```

(Joint Effort Contest, Problem Source: Swedish National Programming Contest, arranged by department of Computer Science at Lund Institute of Technology.)



Problem B: Advanced Causal Measurements (ACM)

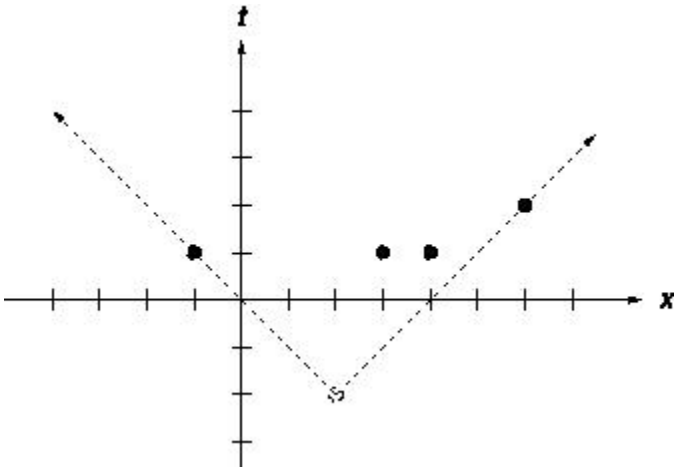
Causality is a very important concept in theoretical physics. The basic elements in a discussion of causality are *events*. An event e is described by its time of occurrence t, and its location, $\boldsymbol{x}$, and we write $e = (t,\boldsymbol{x})$. For our concerns, all events happen in the one dimensional geometric space and thus locations are given by a single real number x as a coordinate on x-axis. Usually, theoretical physicists like to define the speed of light to be 1, so that time and space have the same units (actual physical units frighten and confuse theorists).

One event $e_1 = (t_1,x_1)$ is a *possible cause* for a second event $e_2 = (t_2,x_2)$ if a signal emitted at e_1 could arrive at e_2. Signals can't travel faster than the speed of light, so this condition can be stated as:

$$e_1 \text{ is a } \textit{possible cause} \text{ for } e_2 \textbf{ iff } t_2 >= t_1+|x_2-x_1|$$

Thus an event at (-1,1) could cause events at (0,0), (1,2), and (1,3), for example, but could not have caused events at (1,4) or (-2,1). Note that one event can cause several others.

Recently, scientists have observed several unusual events in the geometrically one dimensional universe, and using current theories, they know how many causes were responsible for these observations, but they know nothing about the time and space coordinates of the causes. You asked to write a program to determine the latest time at which the earliest cause could have occurred (i.e. the time such that at least one cause must have occurred on or before this time). Somewhat surprisingly, all the observed events have both space and time coordinates expressed by integer numbers in the range $-1000000 \le t, x \le 1000000$.



The figure on the right illustrates the first case from input: the earliest single event as a possible cause of all four events.

The first line of input is the number of cases which follow. Each case begins with a line containing the number n of events and the number m of causes, $1 \le n, m \le 100000$. Next follows n lines containing the t and x coordinates for each event.

Output consists of a single line for each case in the format as in the sample output, giving the latest time at which the earliest cause could have occurred, this will be an integer as our time units are not divisible.

Sample Input

```
4
4 1
1 -1
1 3
1 4
2 6
4 2
1 -1
1 3
1 4
2 6
4 3
1 -1
1 3
1 4
2 6
4 4
1 -1
1 3
1 4
2 6
```

Output for Sample Input

```
Case 1: -2
Case 2: 0
Case 3: 0
Case 4: 1
```

Daniel Robbins

Problem D

The Grand Dinner

Input: standard input
Output: standard output
Time Limit: 15 seconds
Memory Limit: 32 MB

Each team participating in this year's **ACM World Finals** contest is expected to join the grand dinner to be arranged after the prize giving ceremony ends. In order to maximize the interaction among the members of different teams, it is expected that no two members of the same team sit at the same table.

Now, given the number of members in each team (including contestants, coaches, reserves, guests etc.) and the seating capacity of each available table, you are to determine whether it is possible for the teams to sit as described in the previous paragraph. If such an arrangement is possible you must also output one possible seating arrangement. If there are multiple possible arrangements, any one is acceptable.

Input

The input file may contain multiple test cases. The first line of each test case contains two integers **M (1 £ M £ 70)** and **N (1 £ N £ 50)** denoting the number of teams and the number of tables respectively. The second line of the test case contains **M** integers where the **i**-th **(1 £ i £ M)** integer $\mathbf{m_i}$ **(1 £** $\mathbf{m_i}$ **£ 100)** indicates the number of members of team **i**. The third line contains **N** integers where the **j**-th **(1 £ j £ N)** integer $\mathbf{n_j}$ **(2 £** $\mathbf{n_j}$ **£ 100)** indicates the seating capacity of table **j**.

A test case containing two zeros for **M** and **N** terminates the input.

Output

For each test case in the input print a line containing either **1** or **0** depending on whether or not there exists a valid seating arrangement of the team members. In case of a successful arrangement print **M** additional lines where the **i**-th **(1 £ i £ M)** of these lines contains a table number (an integer from **1** to **N**) for each of the members of team **i**.

Sample Input

```
4 5
4 5 3 5
3 5 2 6 4
4 5
4 5 3 5
3 5 2 6 3
0 0
```

Sample Output

```
1
1 2 4 5
1 2 3 4 5
```

```
2 4 5
1 2 3 4 5
0
```

(World Finals Warm-up Contest, Problem Setter: Rezaul Alam Chowdhury)

Problem B: Bachet's Game

Bachet's game is probably known to all but probably not by this name. Initially there are **n** stones on the table. There are two players Stan and Ollie, who move alternately. Stan always starts. The legal moves consist in removing at least one but not more than **k** stones from the table. The winner is the one to take the last stone.

Here we consider a variation of this game. The number of stones that can be removed in a single move must be a member of a certain set of **m** numbers. Among the **m** numbers there is always 1 and thus the game never stalls.

Input

The input consists of a number of lines. Each line describes one game by a sequence of positive numbers. The first number is **n** <= 1000000 the number of stones on the table; the second number is **m** <= 10 giving the number of numbers that follow; the last **m** numbers on the line specify how many stones can be removed from the table in a single move.

Input

For each line of input, output one line saying either `Stan wins` or `Ollie wins` assuming that both of them play perfectly.

Sample input

```
20 3 1 3 8
21 3 1 3 8
22 3 1 3 8
23 3 1 3 8
1000000 10 1 23 38 11 7 5 4 8 3 13
999996 10 1 23 38 11 7 5 4 8 3 13
```

Output for sample input

```
Stan wins
Stan wins
Ollie wins
Stan wins
Stan wins
Ollie wins
```

Problem Setter: Piotr Rudnicki