2000 ACM Local Programming Contest

FEI SUT, Bratislava, Slovakia

October 19, 2000

2000 ACM Local Programming Contest FEI SUT, Bratislava, Slovakia

Problem A
Unidirectional TSP

Source: undirect.(c|cc|pas)
Input: standard input, Output: standard output

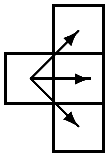
Background

Problems that require minimum paths through some domain appear in many different areas of computer science. For example, one of the constraints in VLSI routing problems is minimizing wire length. The Traveling Salesperson Problem (TSP) — finding whether all the cities in a salesperson's route can be visited exactly once with a specified limit on travel time — is one of the canonical examples of an NP-complete problem; solutions appear to require an inordinate amount of time to generate, but are simple to check.

This problem deals with finding a minimal path through a grid of points while traveling only from left to right.

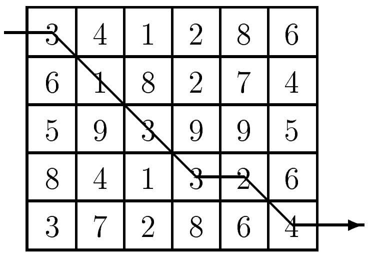
Problem

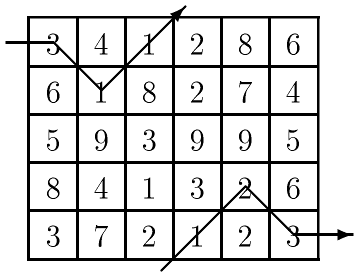
Given an $m \times n$ matrix of integers, you are to write a program that computes a path of minimal weight. A path starts anywhere in column 1 (the first column) and consists of a sequence of steps terminating in column n (the last column). A step consists of traveling from column i to column $i+1$ in an adjacent (horizontal or diagonal) row. The first and last rows (rows 1 and m) of a matrix are considered adjacent, i.e., the matrix "wraps" so that it represents a horizontal cylinder. Legal steps are illustrated below.



The *weight* of a path is the sum of the integers in each of the n cells of the matrix that are visited.

For example, two slightly different 5×6 matrices are shown below (the only difference is the numbers in the bottom row).





The minimal path is illustrated for each matrix. Note that the path for the matrix on the right takes advantage of the adjacency property of the first and last rows.

Input

The input consists of a sequence of matrix specifications. Each matrix specification consists of the row and column dimensions in that order on a line followed by $m \cdot n$ integers where m is the row dimension and n is the column dimension. The integers appear in the input in row major order, i.e., the first n integers constitute the first row of the matrix, the second n integers constitute the second row and so on. The integers on a line will be separated from other integers by one or more spaces. Note: integers are not restricted to being positive. There will be one or more matrix specifications in an input file. Input is terminated by end-of-file.

For each specification the number of rows will be between 1 and 9 inclusive; the number of columns will be between 1 and 100 inclusive. No path's weight will exceed integer values representable using 30 bits.

Output

Two lines should be output for each matrix specification in the input file, the first line represents a minimal-weight path, and the second line is the cost of a minimal path. The path consists of a sequence of n integers (separated by one or more spaces) representing the rows that constitute the minimal path. If there is more than one path of minimal weight the path that is lexicographically smallest should be output.

Sample Input

```
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6
3 7 2 8 6 4
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6
3 7 2 1 2 3
2 2
9 10 9 10
```

Sample Output

```
1 2 3 4 4 5
16
1 2 1 5 4 5
11
1 1
19
```

Problem B
Reflections

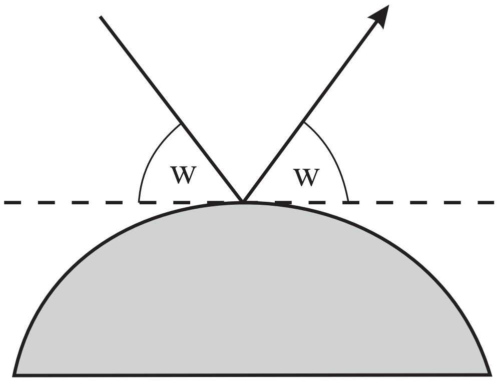
Source: reflect.(c|cc|pas)
Input: standard input, Output: standard output

Problem

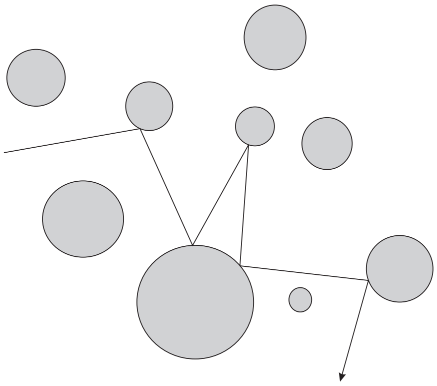
Rendering realistic images of imaginary environments or objects is an interesting topic in computer graphics. One of the most popular methods for this purpose is *ray-tracing*.

To render images using ray-tracing, one computes (traces) the path that rays of light entering a scene will take. We ask you to write a program that computes such paths in a restricted environment.

For simplicity, we will consider only two-dimensional scenes. All objects in the scene are totally reflective (mirror) spheres. When a ray of light hits such a sphere, it is reflected such that the angle of the incoming ray and the leaving ray against the tangent are the same:



The following figure shows a typical path that a ray of light may take in such a scene:



Your task is to write a program, that given a scene description and a ray entering the scene, determines which spheres are hit by the ray.

Input

The input consists of a series of scene descriptions. Each description starts with a line containing the number n ($n \leq 25$) of spheres in the scene. The following n lines contain three integers x_i, y_i, r_i each, where (x_i, y_i) is the center, and $r_i > 0$ is the radius of the i-th sphere. Following this is a line containing four integers x, y, d_x, d_y, which describe the ray. The ray originates from the point (x, y) and initially points in the direction (d_x, d_y). At least one of d_x and d_y will be non-zero.

The spheres will be disjoint and non-touching. The ray will not start within a sphere, and never touch a sphere tangentially.

A test case starting with $n = 0$ terminates the input. This case should not be processed.

Output

For each scene first output the number of the scene. Then print the numbers of the spheres that the ray hits in its first ten deflections (the numbering of spheres is according to their order in the input).

If the ray hits at most ten spheres (and then heads towards infinity), print `inf` after the last sphere it hits. If the ray hits more than 10 spheres, print three points (...) after the tenth sphere.

Output a blank line after each test case.

Sample Input

```
3
3 3 2
7 7 1
8 1 1
3 8 1 -4
2
0 0 1
5 0 2
2 0 1 0
0
```

Sample Output

```
Scene 1
1 2 1 3 inf

Scene 2
2 1 2 1 2 1 2 1 2 1 ...
```

Problem C
Sorting Slides

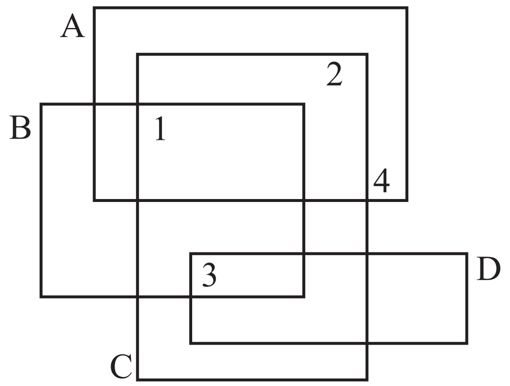
Source: slides.(c|cc|pas)
Input: standard input, Output: standard output

Problem

Professor Clumsey is going to give an important talk this afternoon. Unfortunately, he is not a very tidy person and has put all his transparencies on one big heap. Before giving the talk, he has to sort the slides. Being a kind of minimalist, he wants to do this with the minimum amount of work possible.

The situation is like this. The slides all have numbers written on them according to their order in the talk. Since the slides lie on each other and are transparent, one cannot see on which slide each number is written.

Well, one cannot *see* on which slide a number is written, but one may *deduce* which numbers are written on which slides. If we label the slides which characters A, B, C, ... as in the figure above, it is obvious that D has number 3, B has number 1, C number 2 and A number 4.



Your task, should you choose to accept it, is to write a program that automates this process.

Input

The input consists of several heap descriptions. Each heap descriptions starts with a line containing a single integer n, the number of slides in the heap. The following n lines contain four integers x_{min}, x_{max}, y_{min} and y_{max}, each, the bounding coordinates of the slides. The slides will be labeled as A,B,C,... in the order of the input.

This is followed by n lines containing two integers each, the $x-$ and $y-$coordinates of the n numbers printed on the slides. The first coordinate pair will be for number 1, the next pair for 2, etc. No number will lie on a slide boundary.

The input is terminated by a heap description starting with $n = 0$, which should not be processed.

Output

For each heap description in the input first output its number. Then print a series of all the slides whose numbers can be uniquely determined from the input. Order the pairs by their letter identifier.

If no matchings can be determined from the input, just print the word none on a line by itself. Output a blank line after each test case.

Sample Input

```
4
6 22 10 20
4 18 6 16
8 20 2 18
10 24 4 8
9 15
19 17
11 7
21 11
2
0 2 0 2
0 2 0 2
1 1
1 1
0
```

Sample Output

```
Heap 1
(A,4) (B,1) (C,2) (D,3)

Heap 2
none
```

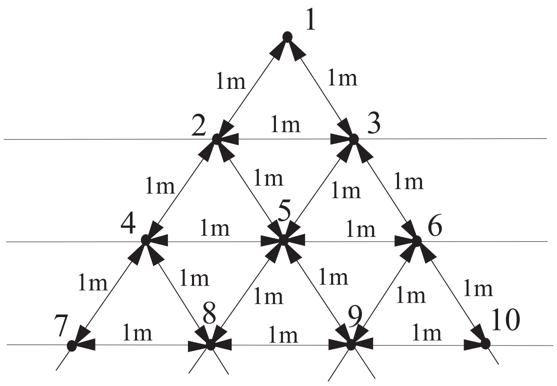
2000 ACM Local Programming Contest FEI SUT, Bratislava, Slovakia

Problem D
Defensive Triples

Source: triples.(c|cc|pas)
Input: standard input, Output: standard output

Problem

Long time ago, when battles took place in the kingdom, one clever soldier has patented "the Offensive Position". At first, soldiers had to number themselves starting from 1. Then the first soldier went into the battle. The other soldiers stood up behind him in the lines. In each line there was one more soldier than in the previous line. Soldiers in a line were ordered by numbers from the left to right and neighbouring soldiers were one meter far from each other. Between each soldier (except of those on the left and on the right borders) and two soldiers from the previous line was exactly one meter space. The soldier on the left border of a line was distanced one meter from the soldier on the left border of previous line. The same for the soldier on the right border. Soldier number 1 (the king) stood up where he liked.



Many brave young men lived in the kingdom in that time. They seemed even mightier, when formed this way. Just in case, this system had also the defensive strategy using the *Defensive Triples*. *Defensive Triple* is formed by three soldiers, who have the same distance from each other. Despite the effort of the king and his strategists, the meaning of this defensive strategy was never fully understood by simple people. But people have always tried to find out which of the soldiers form the *Defensive Triple*.

Input

In each line of the standard input are three numbers A, B, C ($0 < A, B, C < 1000000$) separated by single space, representing the numbers of three soldiers. The input ends with three zeroes.

Output

For each input line output `YES`, if given soldiers form the *Defensive Triple*, output `NO` otherwise.

Sample Input

```
2 8 6
4 1 9
0 0 0
```

Sample Output

```
YES
NO
```

Problem E
Circuits

Source: circuits.(c|cc|pas)
Input: standard input, Output: standard output

Problem

Lately, Philomena has become keen on electricity. Or, maybe on electrical engineering, but it does not matter. So she started peeking all around the house and found some lamps, some switches, some other parts she did not know and a whole pile of wires. She connected all the lamps, switches and other parts by the wires, added batteries and finally watched what she constructed actually. She kept on turning switches, but she could not make all the lamps shine at the same time. What should she do?

N lamps numbered $1, ..., N$ and M switches numbered $1, ..., M$ are given. We know of each lamp if it is shining or not. The i-th switch influences a couple of lamps. If a lamp influenced by the switch have shone before turning the switch, after turning the switch it will be off, and conversely, if it has been off, it will be turned on. Your task is to write a program that will find out if it is possible to turn on all the lamps at the same time. If it is possible, you should also output the number of switches (in increasing order) one should turn.

Input

The first line of input contains integer number K, the number of data blocks. Then K blocks follow: The first line of each block contains two integer numbers N and M separated by one space, $1 \leq N, M \leq 30$. The second line contains N numbers (separated by space) determining which lamps are shining. If the i-th number is zero, the i-th lamp is off, if the i-th number is one, the i-th lamp is on. The following M lines describe the switches. In the $(j+2)$-th line there are the numbers of the lamps influenced by the j-th switch. A zero follows the last lamp number in each line. The neighbouring numbers are separated by one space. The data blocks are separated by one empty line.

Output

The output contains one line for each input data block. Output the sentence `Impossible` in the i-th line if you can not turn on all lamps at the same time according to the data from the i-th block. If this is not the case, the i-th line must contain the sequence of switch numbers in increasing order that should be turned. The sequence should be terminated by zero. All the numbers in the line should be separated by one space. If there are more than one such sequences, your program should print the lexicographically smallest one. A sequence $s_1 < s_2 < ... < s_k$ is lexicographically smaller than $s'_1 < s'_2 < ... < s'_{k'}$ if either such i exists that $s_1 = s'_1, s_2 = s'_2, ...s_i = s'_i$ and $s_{i+1} < s'_{i+1}$, or $k \leq k'$ and $s_i = s'_i$ for all $i = 1...k$.

Sample input

```
2
```

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

3 3
0 1 1
1 2 0
2 3 0
1 3 0
```

Sample output

```
1 3 4 0
Impossible
```

2000 ACM Local Programming Contest FEI SUT, Bratislava, Slovakia

Problem F
The king's weaver

Source: cloth.(c|cc|pas)
Input: standard input, Output: standard output

Problem

In a kingdom, cloth is needed for various purposes, but the most often for the king's suit, for the earls' suits and the servants' suits. This is why they employ a weaver who weaves and weaves all day long. Every evening he takes the long bar of cloth he weaved that day and he divides it into parts for the earls, the servants and, of course, for the king, too. Some cloth parts have to be thrown away. Even the king's weaver can not work without mistake, so there are some defects in the cloth.

Your task is to advice the weaver how he should cut the cloth in the evening to earn as much as possible. He gets 5 golden coins for 1 meter of king's suit cloth, 3 golden coins for 1 meter of earls' suit cloth, and 2 golden coins for 1 meter of cloth for servants. Cloth for the king's suit must be without defects and must be at least 50 meters long (the kings have long coats). Cloth for earls must be at least 20 meters long and it may have no more than 1 defect per each whole 10 meters of the length (i.e. even for 19 meters a maximum of 1 defect). Cloth for servants should not be shorter than 10 meters and it should have no more than 3 defects per 10 meters of length; i.e. for 19 meters length there are 3 defects allowed, but in 20 meters there can be 6 defects. If the cloth is not suitable even for the servants (it does not fulfill any of the criteria), it has to be thrown away. No more than one defect can be in the same meter of cloth and the cloth can only be cut in whole meters (it is not possible to take 2.5 meters of the cloth).

Input

The first line of input contains one integer number k. Then k data blocks follow. The first line of each block contains two integer numbers l and n, separated by one space. l denotes the length of the cloth the weaver weaved that day ($l < 10000$) and n is the number of defects in the cloth ($n \leq l$). The next n lines contain an increasing sequence of n numbers (one number per line) denoting the defective meters of the cloth. (Number 12 means there is a defect in the 12th meter of the cloth.)

Output

Print one line for each input data block, containing the maximum price the weaver can get for the day's work (it will not be more than 30000 golden coins).

Sample input

```
1
25 5
1
2
```

```
3
17
22
```

Sample output

```
66
```

Hint

In the sample above the first 3 meters of cloth were cut and thrown away. The rest is a piece of cloth of 22 meters in length suitable for earls (at least 20 meters and no more than 1 defect per 10 meters of the length).

Problem G
All Squares

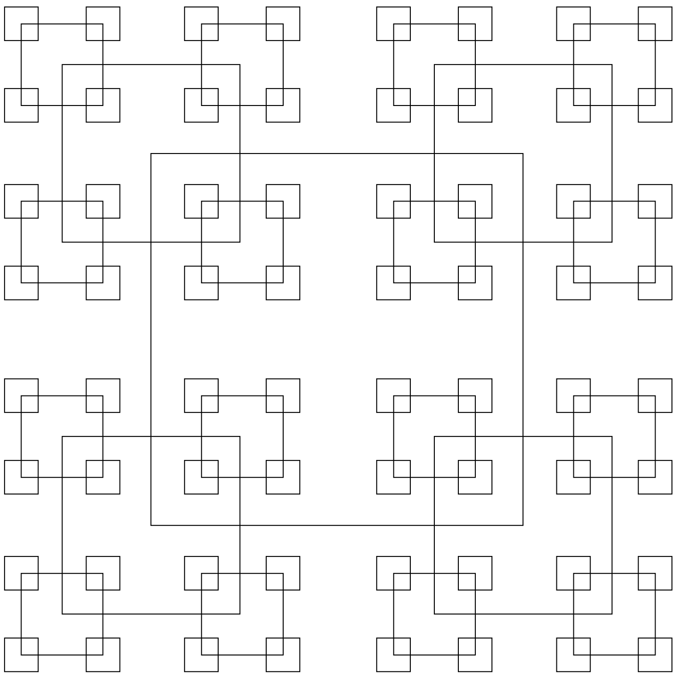
Source: squares.(c|cc|pas)
Input: standard input, Output: standard output

Problem

Geometrically, any square has a unique, well-defined centre point. On a grid this is only true if the sides of the square are an odd number of points long. Since any odd number can be written in the form 2k+1, we can characterise any such square by specifying k, that is we can say that a square whose sides are of length 2k+1 has size k. Now define a pattern of squares as follows.

1. The largest square is of size k (that is sides are of length 2k+1) and is centered in a grid of size 1024 (that is the grid sides are of length 2049).
2. The smallest permissible square is of size 1 and the largest is of size 512, thus $1 \leq k \leq 512$.
3. All squares of size $k > 1$ have a square of size k div 2 centered on each of their 4 corners. (Div implies integer division, thus 9 div 2 = 4).
4. The top left corner of the screen has coordinates (0,0), the bottom right has coordinates (2048, 2048).

Hence, given a value of k, we can draw a unique pattern of squares according to the above rules. Furthermore any point on the screen will be surrounded by zero or more squares. (If the point is on the border of a square, it is considered to be surrounded by that square). Thus if the size of the largest square is given as 15, then the following pattern would be produced.



Write a program that will read in a value of k and the coordinates of a point, and will determine how many squares surround the point.

Input will consist of a series of lines. Each line will consist of a value of k and the coordinates of a point. The file will be terminated by a line consisting of three zeroes (0 0 0).

Output will consist of a series of lines, one for each line of the input. Each line will consist of the number of squares containing the specified point, right justified in a field of width 3.

Sample input

```
15 1024 1024
15 1009 1024
15 1008 1024
15 1019 1019
15 1020 1020
15 1021 1021
500 113 941
0 0 0
```

Sample output

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