Michigan Invitational Programming Competition 2011

October 9, 2011

- There are 9 problems. Do as many as you can in 5 hours.
- You *can* bring paper notes, code printouts, and books.
- You can consult these API documentation web sites:
  `http://download.oracle.com/javase/6/docs/api/`
  `http://cplusplus.com/doc`
- You *cannot* make use of any code that's in electronic form – from the internet, from the computer, or anywhere else – whether you wrote it yourself or not. You cannot use websites other than those mentioned on this page.
- The running time limit is 60 seconds, and the memory limit is 250MB. (Please note that these limits are for the server machine. The running time on your laptop might be faster or slower.)
- Input is from standard input, output is to standard output.
- Input files might consist of multiple test cases, read the input description for detials.
- The URL for the scoreboard and submitting solutions is
  `http://dolphin.eecs.umich.edu/contest/current`

The Problems	
	Problem Name
A	Crossings
B	Music Mess
C	Pivot
D	Evil
E	Spies
F	Static Analysis
G	Brave King
H	Two Sets
I	Yawner

University of Michigan

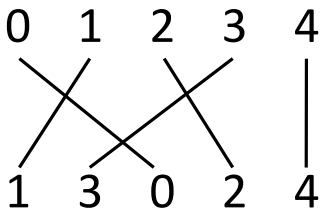
Jump Trading

University of Michigan Jump Trading

A — Crossings

Given a permutation P of $\{0, 1, ..., n-1\}$, we define the crossing number of it as follows. Write the sequence $0, 1, 2, \ldots, n-1$ from left to right above the sequence $P(0), P(1), \ldots, P(n-1)$. Draw a straignt line from 0 in the top line to 0 in the bottom line, from 1 to 1, and so on. The crossing number of P is the number of pairs of lines that cross.

For example, if $n = 5$ and $P = [1, 3, 0, 2, 4]$, then the crossing number of P is 3, as shown in the figure below.



In this problem a permutation will be specified by a tuple (n, a, b), where n is a prime and a and b are integers ($1 \leq a \leq n-1$ and $0 \leq b \leq n-1$). We call this permutation Perm(n, a, b), and the ith element of it is $a * i + b \bmod n$ (with i in the range $[0, n-1]$). So the example above is specified by Perm$(5, 2, 1)$.

Input

There are several test cases in the input file. Each test case is specified by three space-separated numbers n, a, and b on a line. The prime n will be at most 1,000,000. The input is terminated with a line containing three zeros.

Output

For each case in the input print out the case number followed by the crossing number of the permutation. Follow the format in the example output.

Example

Input	Output
5 2 1 19 12 7 0 0 0	Case 1: 3 Case 2: 77

University of Michigan Jump Trading

B — Music Mess

Francis really likes his music. He especially likes that song 'Zombie Nation'. Or was that the album? Or maybe the album was 'Kernkraft 400'. Would anybody really name an album that though? Francis doesn't know and he can't be expected to remember such matters. He's a humble guy and won't be offended if you correct him.

Not all is lost though. Francis does remember the names of the artist, album, and track for some of the songs in his music collection. The only problem is he can't seem to remember which names correspond to what. Luckily for you he does remember that there are no duplicates in the set of all artist, album, and track names.

Additionally Francis' music collection is hierarchical. At the top level he has one or more artists. Each artist then may have one or more albums. Each album then may have one or more tracks. Therefore each track appears on exactly one album and each album was authored by exactly one artist. Help him out by figuring out which names could be artists, which could be albums, and which could be songs.

Input

There are several test cases in the input file. Each test case starts with a single line containing N ($1 \leq N \leq 10,000$), the number of tracks in Francis' collection. The following N lines contain 3 names composed only of characters (a-z, A-Z, 0-9) indicating the artist, album, and track of a single song provided in an unknown order. Each string will contain between 1 and 20 characters. The input is terminated with a line containing 0. The input for each test case will always correspond to at least one legal music library obeying the rules above.

Output

For each case of the input print out the case number followed by three numbers separated by a space. The first indicating how many names could correspond to artists, the second to albums, and the third to tracks. Follow the format shown below.

Example

Input	Output
2 ZombieNation Kernkraft400 Leichenschmaus Zombielicious ZombieNation Supercake53 2 Doolittle Silver Pixies Pixies Doolittle Tame 0	Case 1: 1 4 4 Case 2: 2 2 2

University of Michigan Jump Trading

C — Pivot

The pink surfer is no ordinary surfer. He spends his life on his infinitly long surf board trying not to get himself or his board wet. To do this he balances his board on one buoy from a set of buoys, standing directly over it. He got pretty bored with this pretty quick so he's taken to allowing his board to rotate counter-clockwise around himself. This continues until his board comes into contact with another buoy causing his rotation to stop.

Our friend could just keep rotating around the same buoy but he finds that too boring. Instead when his board hits another buoy he walks from his current position to the buoy that was just hit. This is possible because the two buoys can balance his weight while he walks between them. When he gets above the new buoy he positions himself over it and frees the board from the old buoy and begins rotating counter-clockwise again. The surfer isn't too interesting though, he's been repeating this process his whole life and will do so for as long as he lives.

The other surfers have been teasing the pink surfer about how boring his life must be. You empathize with him though. Who really wants to get wet while surfing anyway? To defend him you wish to compute how many times the pink surfer will walk between buoys before he repeats the same walk twice, let's call this the path length. Specifically you wish to compute all possible distinct path lengths assuming any initial configuration of the pink surfer and his board.

For the purposes of this problem you should model all buoys as points in the 2D plane and the surf board as a line of infinite length. Additionally to eliminate ambiguity no three buoys will lie on the same line in the input.

Input

There are several test cases in the input file. The first line of each test case gives N ($2 \leq N \leq 2,000$). The next N lines give two integers x and y ($0 \leq x, y \leq 10,000,000$) separated by a space giving the positions of the buoys. The last line will contain a single zero.

Output

For each case of the input print out the case number followed by all of the possible path lengths sorted in ascending order with a space between each. Follow the format shown in the example output.

Example

Input	Output
2 0 0 1 0 6 0 0 4 0 0 4 1 1 2 1 1 2 0	Case 1: 2 Case 2: 3 9 18

For the first case note that walking between the same buoys in different directions does not count as the same walk.

University of Michigan Jump Trading

D — Evil

Richard is evil. He wants to give another geometry problem to the participants of this contest and I'm afraid I have no choice but to comply. When asked why exactly he only could respond

> Richard Peng: for lulz

So here's what he wants you to do

> Richard Peng: find a circle that divides red into half
> Richard Peng: without taking any of the blue :D

Fortunately our hero, Mark, has managed to change that circle to an axis parallel rectangle.

Given a set of points in the plane each colored red or blue, find the area of the smallest rectangle that contains exactly half of the red points and none of the blue. The rectangle's sides should be parallel to the x and y axis. There will always be a positive even number of red points. No two points will be at the same position. For the purposes of this problem you can assume that a rectangle contains all points on its border and interior.

Input

There are several test cases in each input file. The first line of each test case contains N ($2 \leq N \leq 20$), the number of points. The following N lines contain x_i, y_i, and c_i ($-1000 \leq x_i, y_i, \leq 1000$, $0 \leq c_i \leq 1$) giving the x and y coordinates of the ith point. The ith point is red if $c_i = 0$ and blue if $c_i = 1$. The last line of input contains a zero.

Output

For each test case output the case number followed by the area of the smallest rectangle that satisfies the conditions above. If it is impossible output -1 instead. Follow the format in the sample output.

Example

Input	Output
7 -10 0 0 -1 0 0 1 0 0 10 0 0 -1 -1 0 1 1 0 0 0 1 7 -4 0 0 -2 0 0 2 0 0 4 0 0 -3 0 1 0 0 1 3 0 1 0	Case 1: 9 Case 2: -1

University of Michigan Jump Trading

E — Spies

In the aftermath of Canada's annexation of Pittsburgh tensions have been pretty high between Canada and the US. You have personally been hired by the US government to track down a number of Canadian spies. Through CIA intelligence the location of a number of spies is partially known. Specifically the CIA knows there are N ($1 \leq N \leq 100000$) spies and have narrowed down each spy to one or two locations.

"Oh My Golly!" exclaims the president. "There could be as many as 2^N possible arrangments for the spies"!

"Not quite, Mr. President. You see the Canadian government would not waste resources by sending two spies to the same location."

Now the task comes to. You are to compute how many ways the spies could be arranged. As this number could be quite large compute it modulo $1,000,000,007$.

Input

Input will consist of multiple test cases. The first line of each case will contain N and M separated by a space ($1 \leq N, M \leq 100,000$) giving the number of spies and the number of cities respectively. After that will be N lines each containing two integers u and v ($0 \leq u, v < M$) indicating the two possible locations of a spy. Interpret lines where $u = v$ as indicating the spy is known to be at location u. The input is terminated with a line containing two zeroes.

Output

For each test case output the case number followed by the number of ways the spies could be arranged modulo $1,000,000,007$. Two arrangements are considered different if any spy has a different location. If there is no way to arrange the spies in different cities output 0. Follow the format in the sample output.

Example

Input	Output
4 5 0 1 1 2 3 4 4 3 3 2 0 0 1 1 0 1 0 0	Case 1: 6 Case 2: 0

The arrangements of the first case are $\{0, 1, 3, 4\}$, $\{0, 1, 4, 3\}$, $\{0, 2, 3, 4\}$, $\{0, 2, 4, 3\}$, $\{1, 2, 3, 4\}$, and $\{1, 2, 4, 3\}$.

F — Static Analysis

Kim, a researcher at the Public Institute of Extraneous Information, has lost her mind. While she searches for it you have been charged with the task of completing her most recent efforts to calculate extra information.

Specifically Kim was investigating several properties of a piece of software developed at the institute called the Data Blob Super-Relator (DBSR). The DBSR program can be thought of as being composed of N functions each of which can call some other functions in the program. The people at the institute aren't your typical programmers. They don't believe in the use of writing loops in their code. Instead they just rely on recursion to do all of their work.

As a demonstrating example let's consider this function for computing the sum of an array. Typically one might implement this algorithm as:

```
Sum(A, length)
  result := 0
  for i = 1 to length
    result := result + A[i]
  return result
```

However not at PIXI! Instead they would make the algorithm recursive similar to:

```
Sum(A, length)
  if length = 0
    return 0
  else
    return A[0] + Sum(A[2:length], length-1)
```

Before her incident, Kim had computed some additional extraneous information about DBSR. She knew for each method which methods it could call (this information is called the call graph). In our example we would only have one entry indicating Sum can call Sum in the call graph. With this and knowledge of the programming style of the DBSR engineers we can compute some interesting properties about the functions that compose it.

But first let us consider the call stack of the DBSR program. Each time a function is called it is pushed onto the top of the call stack. It in turn may call additional functions which will each be pushed even higher on the call stack. When the function completes it will be removed from the call stack. The number of functions on the call stack at any given point is the call stack size.

We wish to partition all of the functions into three categories based on their relation to the call stack; Calculators, Delegators, and Workers. These names correspond roughly to how the methods act. Calculators make quick calculations, Delegators assign work but don't do much themselves, and Workers make up the core of the program's processing routines. To make this more concrete Calculators are those methods that we can put a finite bound on how much larger the call stack size might get before this method terminates. Delegators are those functions that are not Calculators and that we can put a bound on the maximum call stack size of the call stack when the function is called. Workers make up the rest of the functions. These are functions that can appear arbitrarily deep in the call stack and where the call stack could still grow arbitrarily large.

What you are asked to do now is compute how many functions fall into each of these three categories. While you protest that this will slow down your progress on computing the digits of 13 your boss is quite insistent! Given a description of the functions in DBSR and the call graph, compute the number of Calculators, Delegators, and Workers.

Input

Input consists of several test cases. The first line of each case contains N ($1 \leq N \leq 100,000$) and M ($1 \leq M \leq 100,000$) each separated by a space. The next M lines contain two integers u_i ($0 \leq u_i < N$) and v_i ($0 \leq v_i < N$) separated by a space indicating that function u_i can call function v_i in the DBSR program.

Output

For each test case output the case number and the number of Calculators, Delegators, and Workers each separated by a space. Follow the format in the sample output.

Example

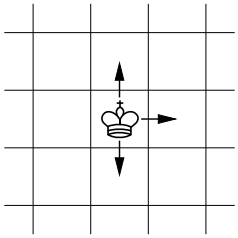
Input	Output
4 4 0 1 1 2 2 1 2 3 0 0	Case 1: 1 1 2

Functions 1 and 2 are Workers, 0 is a Delegator, and 3 is a Calculator.

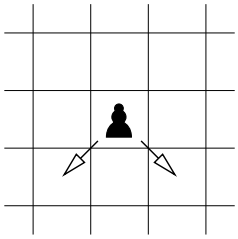
University of Michigan Jump Trading

G — Brave King

The white king is located in the lower left corner of the chessboard. He wants to get to the rightmost column of the board. Unfortunately, a number of black pawns are located on the board and constrain his movements. The king can only move up, down, and right as in the picture:



Each black pawn can check the king if the king is located at one of the two squares below the pawn to the left or to the right as in the picture:



On his journey to the right column the king can capture black pawns, but he wants to capture the minimum number of them. The king can never move to a position that would put him in check by a black pawn. Black pawns do not move during the king's journey.

Input

Input consists of several test cases. The first line of each test case contains two positive integers R and C ($2 \le R, C \le 100$) describing the height and width of the chessboard, respectively. The next R lines describe one row of the chessboard each, from the top to the bottom. Each of these lines is a binary string of length C. Consecutive symbols of the string describe consecutive squares of the chessboard, from left to right. The symbol `1` means that there is a black pawn in the corresponding square. The symbol `0` meas that the corresponding square is empty (or taken by the king). The last line of input will contain two zeroes.

The lower left corner of the chessboard does not contain a black pawn, and the white king placed there is not checked.

Output

For each case of the input, print the case number followed by the minimum number of black pawns the king has to capture to reach the rightmost column, or one word "`Impossible`" if the king cannot reach the rightmost column at all. Follow the format shown in the example.

University of Michigan Jump Trading

Example

Input	Output
4 4 0100 0001 0010 0100 4 3 000 010 001 000 0 0	Case 1: 1 Case 2: Impossible

University of Michigan Jump Trading

H — Two Sets

All Joey really wants in life is to go play at the beach. Unfortunately his math teacher could care less and takes great pleasure in assigning tedius assignments. Fortunately, you love tedius assignments so while Joey's out to catch some waves you get the pleasure of working through his latest task.

Joey's class has just started to go over basic set operations on finite sets. So far they have learned three operations. The complement of set A with respect to set D is all elements in D but not in A. The union of set A and set B is all elements in A or in B (or both). The intersection of set A and set B is all elements in both A and B.

Joey's assignment gives two sets A and B that each contain integers between 1 and N inclusive. Then a formula is given in the following grammar

```
factor := A|B|(term)|factor*factor|-(term)
term := factor|term+factor
formula := factor
```

The factors A and B represent the input sets of Joey's assignment. $factor * factor$ is interpretted to be the intersection of the sets represented by the factors on either side of the * symbol. $term + factor$ then is the union of the sets represented by the left hand term and the right hand factor. -(term) finally is the complement of the term enclosed in parenthesis to the right of the - symbol with respect to set of all integers between 1 and N inclusive.

Finally the assignment asks you to compute the resulting set after evaluating the formula.

Input

Input consists of several test cases. The first line of input contains two integers N ($1 \leq N \leq 200,000$) and M ($1 \leq M \leq 5,000$) where N is defined above and M is the number of characters in the assignment's formula. The next two lines contain descriptions of the set A and set B respectively. Each description contains the number of elements in the set followed by the contents of the set each separated by a space. The next line is the assignment's formula. The last line of input contains two zeroes.

Output

For each test case output the case number and description of the resulting set in the same format as the input. Output the elements of the resulting set in ascending order. Follow the format in the sample output.

Example

Input	Output
10 18 3 1 5 10 4 2 7 5 6 (A*B+-(A)*-(-(B))) 0 0	Case 1: 4 2 5 6 7

University of Michigan Jump Trading

I — Yawner

The yawner has two non-empty sets of positive integers A and B. He spends his boring and pointless life by walking on the number line. You know that initially he is located at the orgin, i.e., the point corresponding to 0. Then in an infinite loop he repeats the following procedure:

- Pick an arbitrary number $x \in A$.
- Make x steps to the right (i.e., move from the current point p to $p + x$).
- *Yawn.*
- Pick an arbitrary number $y \in B$.
- Make y steps to the left (i.e., move from the current point p to $p - y$).
- *Yawn.*

You are desperate to find anything interesting about the life of the yawner. In particular, you started wondering if there is an integer point that the yawner will never yawn at. For instance if $A = \{10, 20\}$ and $B = \{1, 2, 3, 4, 5\}$, the yawner will never yawn at -100. If $A = B = \{3, 6, 9\}$, the yawner will never yawn at 2. If $A = B = \{1, 2\}$, the yawner may yawn at every integer point.

Input

Input consists of serveral test cases. The first line of each test cases contains two positive integers N and M ($1 \leq N, M \leq 100,000$) describing the sizes of the sets A and B respectively. The next N lines contain the elements of A, one per line. The next M lines contain the elements of B, one per line. The last line of input contains two zeroes. The elements of A and B are all positive integers smaller than $1,000,000,000$.

Output

For each case of the input print out the case number and YES if there is an integer point where the yawner can never yawn. Otherwise output NO. Follow the format shown in the sample output.

Example

Input	Output
2 5 10 20 1 2 3 4 5 3 3 3 6 9 3 6 9 2 2 1 2 1 2 0 0	Case 1: YES Case 2: YES Case 3: NO