E • Route Redundancy

A city is made up exclusively of one-way streets. Each street in the city has a capacity, the maximum number of cars it can carry per hour. Any route (path) also has a capacity, which is the minimum of the capacities of the streets along that route.

The *redundancy ratio* from point ***A*** to point ***B*** is the ratio of the maximum number of cars that can get from ***A*** to ***B*** in an hour using all routes simultaneously, to the maximum number of cars that can get from ***A*** to ***B*** in an hour using just one route. The minimum redundancy ratio is the number of cars that can get from ***A*** to ***B*** in an hour using all possible routes simultaneously, divided by the capacity of the single route with the largest capacity.

Input

The first line of input contains a single integer ***P***, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set consists of several lines and represents a directed graph with positive integer weights.

The first line of each data set contains five space separated integers. The first integer, ***D*** is the data set number. The second integer, ***N*** (2 <= ***N*** <= 1000), is the number of nodes in the graph. The third integer, ***E***, (***E*** >= 1), is the number of edges in the graph. The fourth integer, ***A***, (0 <= ***A*** < ***N***), is the index of point **A**. The fifth integer, ***B***, (0 <= ***B*** < ***N*****,** **A** != **B**), is the index of point **B**.

The remaining ***E*** lines describe each edge. Each line contains three space separated integers. The first integer, ***U*** (0 <= ***U*** < N), is the index of node **U**. The second integer, ***V*** (0 <= ***V*** < ***N*****,** V **!=** U), is the index of node **V**. The third integer, ***W*** (1 <= ***W*** < 1000), is the capacity (weight) of the path from **U** to **V**.

Output

For each data set there is one line of output. It contains the data set number (***N***) followed by a single space, followed by a floating-point value which is the minimum *redundancy ratio* to 3 digits after the decimal point.

Sample Input	Sample Output
1 1 7 11 0 6 0 1 3 0 3 3 1 2 4 2 0 3 2 3 1 2 4 2 3 4 2 3 5 6 4 1 1 4 6 1 5 6 9	1 1.667

Problem F
Finding Seats Again

Source file name: `seats.c`, `seats.cpp` *or* `seats.java`

A set of n^2 computer scientists went to the movies. Fortunately, the theater they chose has a square layout: n rows, each one with n seats. However, these scientists are not all from the same research area and they want to seat together. Indeed, there are K independent research groups of scientists among them (no scientist belongs to two of them) with a distiguished leader for each group. Then the leader bought the tickets for his whole group, and he did it in such a way that all his group could seat occupying a rectangular set of seats (and everyone in this set of seats belongs to the same group). Every group was placed satisfying this bizarre condition, although the scientists did not care where the actual assigned areas were.

The usher was informed of the situation and he decided to annotate in a theater map a satisfactory seats deploying. He thought that if he wrote the position of each group's leader in the map indicating besides the corresponding group size, he could tell where to accomodate every scientist. But he discovered that it is not so easy!

The usher asks for your help. You must tell him a way to place the K rectangular areas with the given sizes, and with the corresponding leader for each group seated where it was originally assigned.

Input

Input consists of several test cases, each one defined by a set of lines:

- the first line in the case contains two numbers n and K separated by blanks, with n representing the size of the theater ($0 < n < 20$) and K the number of groups ($K \leq 26$);
- the next n lines describe the usher's map. A one-digit decimal number in the map indicates the seat of a leader and the size of his group. A point indicates that no leader will sit there.

The end of the input is indicated by the line

```
0 0
```

The input must be read from the file seats.in.

Output

For each test case, display an answer consisting in n lines each one of them with n characters representing a seat occupation for the theater. Each group is assigned to an uppercase letter and all of its members are identified with that letter. No two groups are assigned to the same letter.

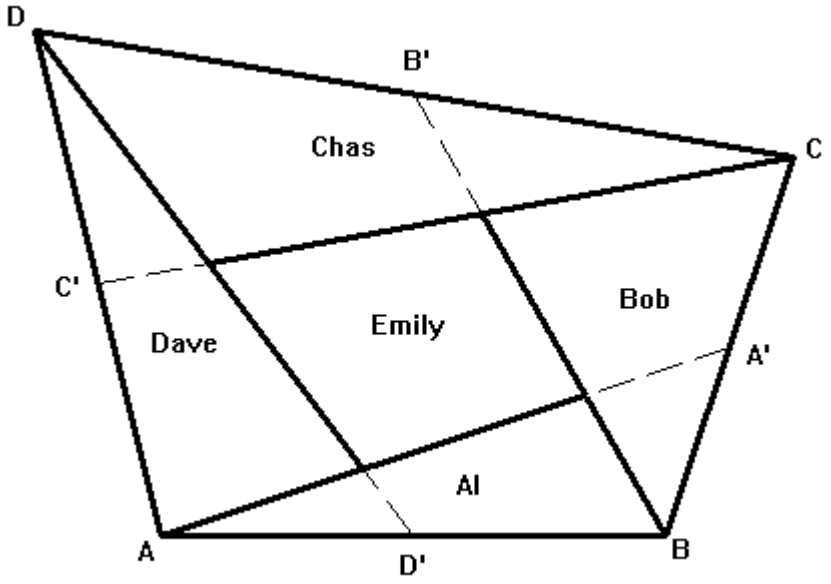
The output must be written to standard output.

Sample input	**Output for the sample input**
3 3 3.4 ... .2. 7 18 ...4.2. ...45.. 222..3. ...2..3 .24...2 ...2.3. 22..3.. 0 0	ABB ABB ACC AAAABCC DDDDBEF GHIIBEF GHJKBEF LLJKBMM NOJPQQQ NOJPRRR

G • Rancher's Gift

Rancher Joel has a tract of land in the shape of a convex quadrilateral that he wants to divide among his sons Al, Bob, Chas and Dave, who wish to continue ranching on their portions, and his daughter Emily, who wishes to grow vegetables on her portion.

The center of the tract is most suitable for vegetable farming so Joel decides to divide the land by drawing lines from each corner (`A`, `B`, `C`, `D` in counter clockwise order) to the center of an opposing side (respectively `A'`, `B'`, `C'` and `D'`). Each son would receive one of the triangular sections and Emily would receive the central quadrilateral section. As shown in the figure, Al's tract is to be bounded by the line from `A` to `B`, the line from `A` to the midpoint of `BC` and the line from `B` to the midpoint of `CD`; Bob's tract is to be bounded by the line from `B` to `C`, the line from `B` to the midpoint of `CD` and the line from `C` to the midpoint of `DA`, and so on.



Your job is to write a program that will help Rancher Joel determine the area of each child's tract and the length of the fence he will have to put around Emily's parcel to keep her brothers' cows out of her crops.

For this problem, `A` will always be at `(0, 0)` and `B` will always be at `(x, 0)`. Coordinates will be in *rods* (a *rod* is 16.5 *feet*). The returned areas should be in *acres* to 3 decimal places (an *acre* is 160 square *rods*) and the length of the fence should be in *feet*, rounded up to the next *foot*.

Input

The first line of input contains a single integer P, ($1 \le P \le 1000$), which is the number of data sets that follow. Each data set is a single line that contains of a decimal integer followed by five (5) space separated floating-point values. The first (integer) value is the data set number, N. The floating-point values are `B.x`, `C.x`, `C.y`, `D.x` and `D.y` in that order (where `V.x` indicates the `x` coordinate of `V` and `V.y` indicates the `y` coordinate of `V`). Recall that the `y` coordinate of `B` is always zero (`0`). The supplied coordinates will always specify a valid convex quadrilateral.

Output

For each data set there is a single line of output. It contains the data set number, N, followed by a single space followed by five (5) space separated floating point values to ***three*** (3) decimal place accuracy, followed by a single space and a decimal integer. The floating-point values are the areas in *acres* of the properties of Al, Bob, Chas, Dave, and Emily respectively. The final integer is the length of fence in *feet* required to fence in Emily's property (rounded up to the next foot).

Sample Input
3 1 200 250 150 -50 200 2 200 200 100 0 100 3 201.5 157.3 115.71 -44.2 115.71

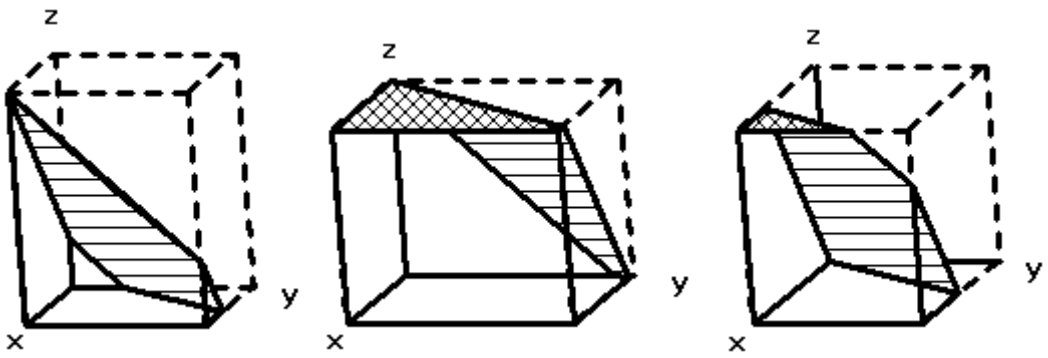
Sample Output
1 35.000 54.136 75.469 54.167 54.666 6382 2 25.000 25.000 25.000 25.000 25.000 4589 3 29.144 29.144 29.144 29.144 29.144 4937

I • The Golden Ceiling

The main office of the *Bank of Zork* was built in the *Aragain Village* (later known as *Flatheadia*) in the year 722 of the Great Underground Empire (*GUE*). In 788 *GUE*, the chairman, J. Pierpont Flathead, decided (shortly before his unexplained disappearance in 789), that it was time to completely redesign the already ornate bank atrium with a ceiling covered in brilliant gold leaf.

This new ceiling was not your ordinary ceiling. Although the atrium is essentially a big box, the ceiling would be slanted (supposedly, making the atrium look bigger). At the time, the exact dimensions of the rectangular atrium and the slope and location of the slanted ceiling had not been finalized. Flathead wanted to know how much gold leaf he would have to order from the *Frobozz Magic Gold Leaf Company* to cover the ceiling for different atrium dimensions and ceiling slants. He also wanted to allow the slanted part to possibly hit the floor of the box and/or the top of the box.

Consider the following rough sketches of some possible atrium configurations:



Note: The dashed outline represents the original box, the horizontally ruled surface is the slanted part of the ceiling and the cross hatched surface is the part of the top of the box not cut off by the plane. The walls and floor of the atrium are transparent. The total area to be covered (the *ceiling*) is the slanted part plus any part of the top of the original box that is not cut off by the plane.

Your job is to write a program that Flathead could have used to calculate the amount gold leaf required to cover the *ceiling* for a particular configuration.

As a sad epilogue, the main branch was brought to ruins when the Curse of Megaboz befell it in 883*GUE*. Between the barbarian invasions of the 880's and the countless looters that had tread the underground ruins in the years that followed, the entire bank with all its valuables, as well as its very expensive gold leaf ceiling, had been removed or vandalized. More information can be found on-line at: http://www.thezorklibrary.com/history/bank_of_zork.html.

(Continued on next page)

Input

The first line of input contains a single integer ***P***, ($1 \le P \le 1000$), which is the number of data sets that follow. Each data set is a single line that contains the data set number, ***N***, followed by a space, followed by seven space separated double precision floating point values, **L**, **W**, **H**, **A**, **B**, **C** and **D**. The values **L**, **W** and **H** specify the *length, width* and *height* of the atrium in *Flathead Units* (*FU*'s), respectively, and are always positive values. The values **A**, **B**, **C** and **D** specify the coefficients of the plane equation for the slanted part of the ceiling:

$$\mathbf{A}x + \mathbf{B}y + \mathbf{C}z = \mathbf{D}$$

where: $0 <= x <= \mathbf{L}$, $0 <= y <= \mathbf{W}$, $0 <= z <= \mathbf{H}$.

One corner of the original box is always at the origin (0, 0, 0) and the other at (**L**, **W**, **H**). The plane will never be vertical (C will be >= 1.0) and the plane will always pass through the interior of the box (there will be points (x,y,z) in the box and strictly above the plane (Ax + BY +Cz > D).and others strictly below the plane (Ax + By + Cz < D).

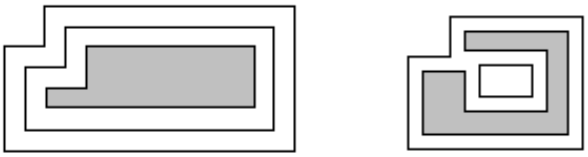
Output

For each data set there is one line of output. It contains the data set number (***N***) followed by a single space, followed by an integer value that is the number of square *FU*'s required to cover the *ceiling* in gold leaf (rounded *up* to the next square *FU*).

Sample Input	Sample Output
3 1 10 12 15 -1.3 1 1.1 3.5 2 10 12 10 -1.3 1 1.1 11 3 13 9 10 -1.3 1 1.1 0	1 166 2 164 3 144

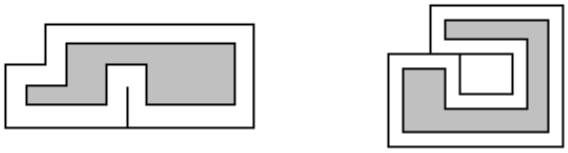
Problem A: Bordering on Madness

Bob Roberts owns a design business which creates custom artwork for various corporations. One technique that his company likes to use is to take a simple rectilinear figure (a figure where all sides meet at 90 or 270 degrees and which contains no holes) and draw one or more rectilinear borders around them. Each of these borders is drawn so that it is a set distance d away from the previously drawn border (or the original figure if it is the first border) and then the new area outlined by each border is painted a unique color. Some examples are shown below (without the coloring of the borders).



The example on the left shows a simple rectilinear figure (grey) with two borders drawn around it. The one on the right is a more complicated figure; note that the border may become disconnected.

These pieces of art can get quite large, so Bob would like a program which can draw prototypes of the finished pieces in order to judge how aesthetically pleasing they are (and how much money they will cost to build). To simplify things, Bob never starts with a figure that results in a border where 2 horizontal (or vertical) sections intersect, even at a point. This disallows such cases as those shown below:



Input

Input will consist of multiple test cases. The first line of the input file will contain a single integer indicating the number of test cases. Each test case will consist of two or more lines. The first will contain three positive integers n, m and d indicating the number of sides of the rectlinear figure, the number of borders to draw, and the distance between each border, where $n \leq 100$ and $m \leq 20$. The remaining lines will contain the n vertices of the figure, each represented by two positive integers indicating the x and y coordinates. The vertices will be listed in clockwise order starting with the vertex with the largest y value and (among those vertices) the smallest x value.

Output

For each test case, output three lines: the first will list the case number (as shown in the examples), the second will contain m integers indicating the length of each border, and the third will contain m integers indicating the additional area contributed to the artwork by each border. Both of these sets of numbers should be listed in order, starting from the border nearest the original figure. Lines two and three should be indented two spaces and labeled as shown in the examples. Separate test cases with a blank line.

Sample Input

```
2
6 2 10
20 30 100 30 100 0 0 0 0 10 20 10
10 1 7
20 50 70 50 70 0 0 0 0 30
20 30 20 10 60 10 60 40 20 40
```

Sample Output

```
Case 1:
  Perimeters: 340 420
  Areas: 3000 3800

Case 2:
  Perimeters: 380
  Areas: 2660
```

Problem B: Jack of All Trades

Jack Barter is a wheeler-dealer of the highest sort. He'll trade anything for anything, as long as he gets a good deal. Recently, he wanted to trade some red agate marbles for some goldfish. Jack's friend Amanda was willing to trade him 1 goldfish for 2 red agate marbles. But Jack did some more digging and found another friend Chuck who was willing to trade him 5 plastic shovels for 3 marbles while Amanda was willing to trade 1 goldfish for 3 plastic shovels. Jack realized that he could get a better deal going through Chuck (1.8 marbles per goldfish) than by trading his marbles directly to Amanda (2 marbles per goldfish).

Jack revels in transactions like these, but he limits the number of other people involved in a chain of transactions to 9 (otherwise things can get a bit out of hand). Normally Jack would use a little program he wrote to do all the necessary calculations to find the optimal deal, but he recently traded away his computer for a fine set of ivory-handled toothpicks. So Jack needs your help.

Input

Input will consist of multiple test cases. The first line of the file will contain an integer n indicating the number of test cases in the file. Each test case will start with a line containing two strings and a positive integer $m \leq 50$. The first string denotes the items that Jack wants, and the second string identifies the items Jack is willing to trade. After this will be m lines of the form

a_1 $name_1$ a_2 $name_2$

indicating that some friend of Jack's is willing to trade an amount a_1 of item $name_1$ for an amount a_2 of item $name_2$. (Note this does not imply the friend is also willing to trade a_2 of item $name_2$ for a_1 of item $name_1$.) The values of a_1 and a_2 will be positive and ≤ 20. No person will ever need more than $2^{31} - 1$ items to complete a successful trade.

Output

For each test case, output the phrase `Case i:` (where `i` is the case number starting at 1) followed by the best possible ratio that Jack can obtain. Output the ratio using 5 significant digits, rounded. Follow this by a single space and then the number of ways that Jack could obtain this ratio.

Sample Input

```
2
goldfish marbles 3
1 goldfish 2 marbles
5 shovels 3 marbles
1 goldfish 3 shovels
this that 4
7 this 2 that
14 this 4 that
7 this 2 theother
1 theother 1 that
```

Sample Output

```
Case 1: 1.8000 1
Case 2: 0.28571 3
```

Problem F: Tanks a Lot

Imagine you have a car with a very large gas tank - large enough to hold whatever amount you need. You are traveling on a circular route on which there are a number of gas stations. The total gas in all the stations is exactly the amount it takes to travel around the circuit once. When you arrive at a gas station, you add all of that station's gas to your tank. Starting with an empty tank, it turns out there is at least one station to start, and a direction (clockwise or counter-clockwise) where you can make it around the circuit. (On the way home, you might ponder why this is the case - but trust us, it is.)

Given the distance around the circuit, the locations of the gas stations, and the number of miles your car could go using just the gas at each station, find all the stations and directions you can start at and make it around the circuit.

Input

There will be a sequence of test cases. Each test case begins with a line containing two positive integers c and s, representing the total circumference, in miles, of the circle and the total number of gas stations. Following this are s pairs of integers t and m. In each pair, t is an integer between 0 and $c-1$ measuring the clockwise location (from some arbitrary fixed point on the circle) around the circumference of one of the gas stations and m is the number of miles that can be driven using all of the gas at the station. All of the locations are distinct and the maximum value of c is 100,000. The last test case is followed by a pair of 0's.

Output

For each test case, print the test case number (in the format shown in the example below) followed by a list of pairs of values in the form `i d`, where `i` is the gas station location and `d` is either `C, CC`, or `CCC`, indicating that, when starting with an empty tank, it is possible to drive from location `i` around in a clockwise (`C`) direction, counterclockwise (`CC`) direction, or either direction (`CCC`), returning to location `i`. List the stations in order of increasing location.

Sample Input

```
10 4
2 3 4 3 6 1 9 3
5 5
0 1 4 1 2 1 3 1 1 1
0 0
```

Sample Output

```
Case 1: 2 C 4 CC 9 C
Case 2: 0 CCC 1 CCC 2 CCC 3 CCC 4 CCC
```

Problem G: The Worm Turns

Winston the Worm just woke up in a fresh rectangular patch of earth. The rectangular patch is divided into cells, and each cell contains either food or a rock. Winston wanders aimlessly for a while until he gets hungry; then he immediately eats the food in his cell, chooses one of the four directions (north, south, east, or west) and crawls in a straight line for as long as he can see food in the cell in front of him. If he sees a rock directly ahead of him, or sees a cell where he has already eaten the food, or sees an edge of the rectangular patch, he turns left or right and once again travels as far as he can in a straight line, eating food. He never revisits a cell. After some time he reaches a point where he can go no further so Winston stops, burps and takes a nap.

For instance, suppose Winston wakes up in the following patch of earth (X's represent stones, all other cells contain food):

	0	1	2	3	4
0					X
1					
2					
3		X	X		
4					

If Winston starts eating in row 0, column 3, he might pursue the following path (numbers represent order of visitation):

	0	1	2	3	4
0	4	3	2	1	X
1	5	18	17	16	15
2	6	19	20	21	14
3	7	X	X	22	13
4	8	9	10	11	12

In this case, he chose his path very wisely: every piece of food got eaten. Your task is to help Winston determine where he should begin eating so that his path will visit as many food cells as possible.

Input

Input will consist of multiple test cases. Each test case begins with two positive integers, m and n, defining the number of rows and columns of the patch of earth. Rows and columns are numbered starting at 0, as in the figures above. Following these is a non-negative integer r indicating the number of rocks, followed by a list of $2r$ integers denoting the row and column number of each rock. The last test case is followed by a pair of zeros. This should not be processed. The value $m \times n$ will not exceed 625.

Output

For each test case, print the test case number (beginning with 1), followed by four values:

```
amount row column direction
```

where `amount` is the maximum number of pieces of food that Winston is able to eat, `(row, column)` is the starting location of a path that enables Winston to consume this much food, and `direction` is

one of `E, N, S, W`, indicating the initial direction in which Winston starts to move along this path. If there is more than one starting location, choose the one that is lexicographically least in terms of row and column numbers. If there are optimal paths with the same starting location and different starting directions, choose the first valid one in the list `E, N, S, W`. Assume there is always at least one piece of food adjacent to Winston's initial position.

Sample Input

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

Sample Output

```
Case 1: 22 0 3 W
```

Problem H: You'll be Working on the Railroad

Congratulations! Your county has just won a state grant to install a rail system between the two largest towns in the county — Acmar and Ibmar. This rail system will be installed in sections, each section connecting two different towns in the county, with the first section starting at Acmar and the last ending at Ibmar. The provisions of the grant specify that the state will pay for the two largest sections of the rail system, and the county will pay for the rest (if the rail system consists of only two sections, the state will pay for just the larger section; if the rail system consists of only one section, the state will pay nothing). The state is no fool and will only consider simple paths; that is, paths where you visit a town no more than once. It is your job, as a recently elected county manager, to determine how to build the rail system so that the county pays as little as possible. You have at your disposal estimates for the cost of connecting various pairs of cities in the county, but you're short one very important requirement — the brains to solve this problem. Fortunately, the lackeys in the computing services division will come up with something.

Input

Input will contain multiple test cases. Each case will start with a line containing a single positive integer $n \leq 50$, indicating the number of railway section estimates. (There may not be estimates for tracks between all pairs of towns.) Following this will be n lines each containing one estimate. Each estimate will consist of three integers s e c, where s and e are the starting and ending towns and c is the cost estimate between them. (Acmar will always be town 0 and Ibmar will always be town 1. The remaining towns will be numbered using consecutive numbers.) The costs will be symmetric, i.e., the cost to build a railway section from town s to town e is the same as the cost to go from town e to town s, and costs will always be positive and no greater than 1000. It will always be possible to somehow travel from Acmar to Ibmar by rail using these sections. A value of $n = 0$ will signal the end of input.

Output

For each test case, output a single line of the form

```
c1 c2 ... cm cost
```

where each `ci` is a city on the cheapest path and `cost` is the cost to the county (note `c1` will always be `0` and `cm` will always be `1` and `ci` and `ci+1` are connected on the path). In case of a tie, print the path with the shortest number of sections; if there is still a tie, pick the path that comes first lexicographically.

Sample Input

```
7
0 2 10
0 3 6
2 4 5
3 4 3
3 5 4
4 1 7
5 1 8
0
```

Sample Output

```
0 3 4 1 3
```

Problem A. Alien Communication Masterclass

Input file:	`acm.in`
Output file:	`acm.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Andrea is a famous science fiction writer, who runs masterclasses for her beloved readers. The most popular one is the Alien Communication Masterclass (ACM), where she teaches how to behave if you encounter alien life forms or at least alien artifacts.

One of the lectures concerns retrieving useful information based on aliens' writings. Andrea teaches that based on alien mathematical formulas, one could derive the base of the numeral system used by the aliens, which in turn might give some knowledge about aliens' organisms. (For example, we use numeral system with base 10, due to the fact that we have ten fingers on our upper extremities).

Suppose for simplicity that aliens use the same digits as we do, and they understand and denote addition, subtraction, multiplication, parentheses and equality the same way as we do.

For her lecture, Andrea wants an example of a mathematical equality that holds in numeral systems with bases $a_1, a_2, \cdots, a_n$, but doesn't hold in numeral systems with bases $b_1, b_2, \cdots, b_m$. Provide her with one such formula.

Input

The first line of the input file contains two integer numbers, n and m ($1 \le n, m \le 8$). The second line contains n numbers, $a_1, a_2, \cdots, a_n$. The third line contains m numbers, $b_1, b_2, \cdots, b_m$.

All a_i and b_i are distinct and lie between 2 and 10, inclusive.

Output

Output any syntactically correct mathematical equality that holds in numeral systems with bases $a_1, a_2, \cdots, a_n$, but doesn't hold in numeral systems with bases $b_1, b_2, \cdots, b_m$. The equality can contain only digits 0 through 9, addition ('`+`'), subtraction and unary negation ('`-`'), multiplication ('`*`'), parentheses ('`(`' and '`)`') and equality sign ('`=`'). There must be exactly one equality sign in the output.

Any whitespace characters in the output file will be ignored. The number of non-whitespace characters in the output file must not exceed 10 000.

Examples

acm.in	acm.out
1 2 2 3 9	(10 - 1) * (10 - 1) + 1 = 10
2 2 9 10 2 3	2 + 2 = 4

Problem B. Bug2

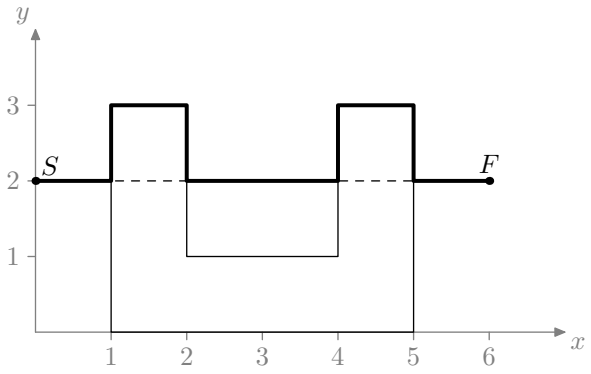
Input file:	`bug2.in`
Output file:	`bug2.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

There is a variety of navigation problems around us. This is a problem about an algorithm called *Bug2*.

Bug algorithms solve the following navigation problem. There is a two-dimensional map containing obstacles of an arbitrary shape, and start and finish points are given. There is also an "agent", who initially stands at the start point S, and its task is to reach the finish point F. It knows the coordinates of the finish point, and at any moment of time it can determine its own coordinates. The agent has $O(1)$ memory, so it cannot store the map of obstacles. The only way it can get information about the outer world is to detect whether it touches an obstacle. The Agent is able to move around the obstacle following its boundary. The problem is to reach the finish point when it is possible, and correctly report the fact of unreachability otherwise.

The *Bug2* algorithm, which belongs to the family of Bug algorithms, works as follows:

1. Move towards F until one of the following happens:
   - The finish point is reached. Then the algorithm stops.
   - An obstacle is encountered. Then go to step 2.
2. Define the current point as H. Move around the boundary of the obstacle in the clockwise direction until one of the following happens:
   - The finish point is reached. Then the algorithm stops.
   - The point H is reached. Then the finish point is unreachable, and the algorithm stops.
   - A point L is reached, which lies on the line SF, $|LF| < |HF|$ and it is possible to move towards F without hitting an obstacle immediately. In this case, go to step 1.



One may prove the correctness of an algorithm, that is, that it reaches the finish point in finite time (that is, the length of the resulting path is finite) if it is possible, and reports that the finish point is unreachable in finite time otherwise.

Given a set of polygonal obstacles, a start and a finish point, determine the length of the path that an agent directed by *Bug2* algorithm will traverse.

Input

The first line of the input file contains five integer numbers n, x_S, y_S, x_F, y_F — the number of obstacles and the coordinates of start and finish points.

The rest of the input file describes obstacles. Each description starts with a line containing an integer m ($m \geq 3$) — the number of vertices in the obstacle. The following m lines contain pairs of integer numbers x_i, y_i — the coordinates of vertices of the obstacle, given in the clockwise direction. The obstacle does not have self-intersections or self-tangencies.

The total number of vertices in all the obstacles does not exceed 300 000. No coordinate exceeds 10^6 by an absolute value.

No vertex of an obstacle lies on a line SF. Both start and finish point will lie strictly outside any obstacle. No two obstacles share common points. If there are two points A and B where obstacle boundaries intersect with the line SF, either $|AF| = |BF|$ or $\big||AF| - |BF|\big| > 10^{-6}$ will be true.

Output

Output the length of a path traversed by the agent directed by the described algorithm. The absolute or relative error of 10^{-6} is acceptable.

Example

bug2.in	bug2.out
1 0 2 6 2 8 1 0 1 3 2 3 2 1 4 1 4 3 5 3 5 0	10.0

Problem C. Commuting Functions

Input file:	commuting.in
Output file:	commuting.out
Time limit:	3 seconds
Memory limit:	256 megabytes

Two functions f and g ($f, g : X \to X$) are *commuting* if and only if $f(g(x)) = g(f(x))$ for each $x \in X$. For example, functions $f(x) = x + 1$ and $g(x) = x - 2$ are commuting, whereas functions $f(x) = x + 1$ and $g(x) = 2x$ are not commuting.

Each function h ($h : N_n \to N_n$, where $N_n = 1, 2, \ldots, n$ and n is positive integer) can be represented as a *value list* — a list in which the i-th element is equal to $h(i)$. For example, a function $h(x) = \lceil x/2 \rceil$ from N_5 to N_5 has the value list $[1, 1, 2, 2, 3]$.

The value lists are ordered lexicographically: list $[a_1 \ldots a_n]$ is smaller than list $[b_1 \ldots b_n]$ if and only if there exists such an index k that $a_k < b_k$, and $a_l = b_l$ for any index $l < k$.

The function f ($f : X \to X$) is *bijective* if for every y in X, there is exactly one x in X such that $f(x) = y$.

Given a bijective function f ($f : N_n \to N_n$, n is positive integer), find the function g that is commuting with f and has the lexicographically smallest possible value list.

Input

The first line contains single integer number n — the number of the elements in the value list of bijective function f ($1 \le n \le 200\,000$).

The second line of the input file contains the value list of the function f.

Output

The single line of the output file must contain n integer numbers — the value list of function g that commutes with the function f and has the lexicographically smallest value list.

Examples

commuting.in	commuting.out
10 1 2 3 4 5 6 7 8 9 10	1 1 1 1 1 1 1 1 1 1
10 2 3 4 5 6 7 8 1 9 10	1 2 3 4 5 6 7 8 9 9

Problem F. Frames

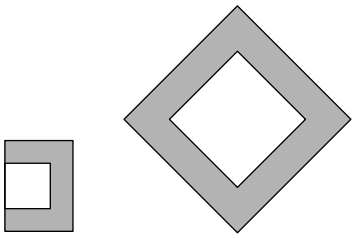
Input file:	`frames.in`
Output file:	`frames.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Vasya and Petya are playing an interesting game. Rules are pretty easy: there are two frames, the players have to make a translation of the second frame in such a way that the area of the intersection of the frames would be as large as possible. Both players think for a minute, and write down the translation vector of the second frame. The player whose vector gives a larger intersection area wins.

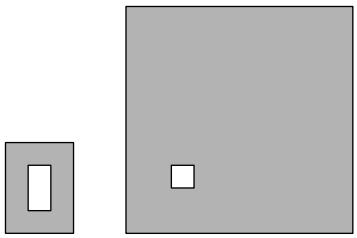
The game has many subtle cases, so Vasya wants to cheat and write a program that finds the best translation vector.

For this game the *frame* is a difference of two rectangles: the inner one and the outer one. The inner rectangle lies strictly inside the outer one. Sides of both rectangles are parallel to the coordinate axes.

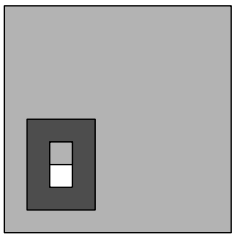
To make the definition more clear let us consider some examples.



Incorrect frames



Correct Frames



Frames intersection

The area of the frame is $(W \cdot H - w \cdot h)$, where W, H — dimensions of the outer rectangle and w, h — dimensions of the inner one $(0 < w < W; 0 < h < H)$.

Write a program that finds a translation of one frame relative to another that results in maximum frames intersection area.

Input

Each frame is described by four points — two opposite corners of the outer rectangle, followed by two opposite corners of the inner rectangle. Points are described by their coordinates — pairs of integer numbers x and y. Coordinates do not exceed 10^8 by absolute value.

The first line of the input file contains the description of the first frame.

The second line of the input file contains the description of the second frame.

Output

The first line of the output file must contain an integer number A — the maximal possible intersection area of the given two frames achievable by a translation.

The second line of the output file must contain a pair of integer numbers x and y — coordinates of the translation vector of the second frame that provides the intersection area A. Coordinates must not exceed 10^{18} by the absolute value.

Example

frames.in	frames.out
2 2 5 6 3 3 4 5 0 0 10 10 2 2 3 3	10 1 1

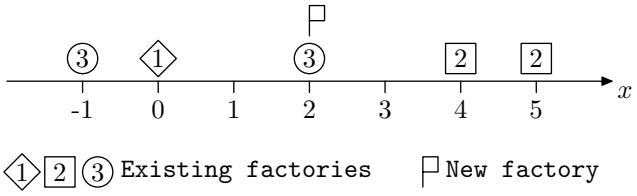
Problem G. Gadgets Factory

Input file:	gadgets.in
Output file:	gadgets.out
Time limit:	3 seconds
Memory limit:	256 megabytes

Mr. Smith is a very rich gadgets fan. As soon as he realized that he cannot buy all the gadgets he wants just because they are not yet produced, he decided to build his own Gadgets Factory.

The Gadgets Factory will be built at a place called 'Silicon Road". This road concentrates production of highly technological parts required to produce gadgets. The Silicon Road is straight and the factories are placed very close to it, so the road can be considered as an axis, and factories can be considered as points on it.

There are n parts needed to produce gadgets, and m factories that produce these parts. Mr. Smith wants to minimize the sum of squared distances to the nearest factories that produce each of required parts. Help him to find all the points in which that sum is minimal.



Input

The first line of the input file contains integer numbers n and m ($1 \le n \le 10\,000$; $n \le m \le 100\,000$).

Next m lines contain pairs of integer numbers x_i and p_i, x_i is the coordinate of i-th factory, and p_i is the identifier of the part that it produces ($|x_i| \le 100\,000$; $x_i \le x_{i+1}$; $1 \le p_i \le n$).

For each required part there is at least one factory that produces it.

Output

The first line of the output file should contain an integer number k — the number of points where the Gadgets Factory can be built.

Next k lines should contain these points in ascending order. The values should be accurate within an absolute error of 10^{-6}.

Examples

gadgets.in	gadgets.out
3 5 -1 3 0 1 2 3 4 2 5 2	1 2.0
2 5 1 1 2 2 3 1 4 2 5 1	4 1.5 2.5 3.5 4.5

Problem H. Horrible Truth

Input file:	horrible.in
Output file:	horrible.out
Time limit:	3 seconds
Memory limit:	256 megabytes

In a Famous TV Show "Find Out" there are n characters and only one Horrible Truth. To make the series breathtaking all way long, the screenplay writer decided that every episode should show exactly one important event.

There are three types of the *important events* in this series:

- character A finds out the Truth;
- character A finds out that the other character B knows the Truth;
- character A finds out that the other character B doesn't know the Truth.

Initially, nobody knows the Truth. All events must be correct, and every fact found out must be true. If some character finds out some fact, she cannot find it out once again.

Moreover, to give the audience some sense of action, the writer does not want an episode to show the important event of the same type as in the previous episode.

Your task is to determine the maximal possible number of episodes in the series and to create an example of a screenplay plan.

Input

The only line of the input contains a single integer n — the number of characters in the TV show ($1 \le n \le 100$).

Output

In the first line of the output file output a single integer k — the maximal possible number of episodes in the series. Then write k lines, each containing a description of an episode. For the episode in which character A (characters are numbered 1 through n) finds out the Truth, write the line "A 0". For an episode in which character A finds out that character B knows the Truth, write the line "A B". Similarly, for an episode in which character A finds out that character B doesn't know the Truth, write the line "A -B".

If there are several plans providing the maximal possible number of episodes, output any one of them.

Example

horrible.in	horrible.out
3	13 2 -1 1 0 2 1 1 -2 3 1 3 -2 2 0 1 2 1 -3 3 2 2 -3 3 0 1 3

Problem J. Journey

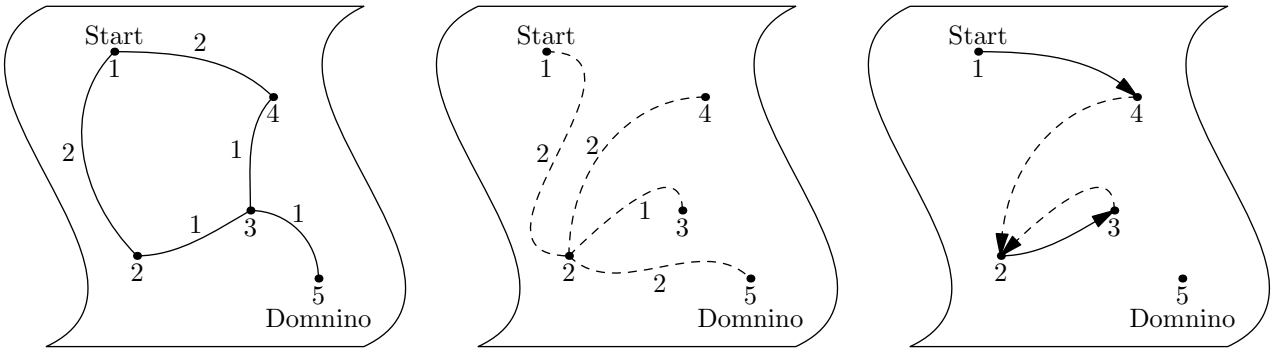
Input file:	`journey.in`
Output file:	`journey.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

The army of Rzeczpospolita is moving from the city Kostroma to the village Domnino. Two hetmans, Stefan and Konstantin, lead the army.

Stefan procured the roadmap of Kostroma province, and every night he routes the army from one village to the other along some road. Konstantin bought the map of secret trails between villages in advance, and every day he leads the march along the one of such trails. Each hetman asks their guide Ivan Susanin for a route before each march.

The length of each road is indicated on Stefan's map. So Stefan knows the minimal distance from each village to the Domnino village according to his map. Similarly Konstantin knows the minimal distance from each village to Domnino village along trails on his map.

Ivan Susanin does not want to be disclosed as a secret agent, so each time he chooses a road (for Stefan) or a trail (for Konstantin) so that the minimal distance to the Domnino village according to the map owned by the asking hetman is strictly decreasing.



Stefan's map | Konstantin' map | Susanin's route

Help Ivan to find the longest possible route to the Domnino village.

Input

The first line of the input file contains three integer numbers n, s and t — number of villages in Kostroma province, and numbers of start and Domnino village ($2 \le n \le 1000$; $1 \le s, t \le n$). Villages are numbered from 1 to n. Start and Domnino villages are distinct.

Two blocks follow, the first one describing Stefan's map, and the second one describing Konstantin's map.

The first line of each block contains an integer number m — the number of roads/trails between villages ($n - 1 \le m \le 100\,000$). Each of the following m lines contains three integer numbers a, b, and l — describing the road/trail between villages a and b of length l ($1 \le a, b \le n$; $1 \le l \le 10^6$).

Rzeczpospolita army can move in any direction along a road or a trail. It's guaranteed that one can travel from any village to any other using each of the maps. The army starts its movement in the evening from the village number s and moves one road each night and one trail each day.

Output

Output the total length of the longest route that Ivan Susanin can arrange for Rzeczpospolita army before reaching the Domnino village (along the roads and trails). If Ivan Susanin can route the army forever without reaching the Domnino village, output the number "`-1`".

Examples

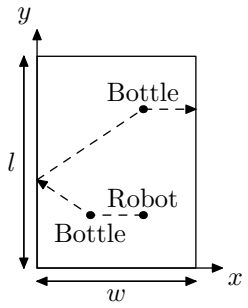
journey.in	journey.out
5 1 5 5 1 2 2 1 4 2 2 3 1 3 4 1 5 3 1 4 1 2 2 2 4 2 2 3 1 2 5 2	-1
3 1 3 4 1 2 10 2 3 10 1 3 20 2 3 30 4 2 1 10 1 3 10 1 1 10 2 3 10	20

Problem K. Kitchen Robot

Input file:	`kitchen.in`
Output file:	`kitchen.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Robots are becoming more and more popular. They are used nowadays not only in manufacturing plants, but also at home. One programmer with his friends decided to create their own home robot. As you may know most programmers like to drink beer when they gather together for a party. After the party there are a lot of empty bottles left on the table. So, it was decided to program robot to collect empty bottles from the table.

The table is a rectangle with the length l and width w. Robot starts at the point (x_r, y_r) and n bottles are located at points (x_i, y_i) for $i = 1, 2, \ldots, n$. To collect a bottle robot must move to the point where the bottle is located, take it, and then bring to some point on the border of the table to dispose it. Robot can hold only one bottle at the moment and for simplicity of the control program it is allowed to release bottle only at the border of the table.



You can assume that sizes of robot and bottles are negligibly small (robot and bottles are points), so the robot holding a bottle is allowed to move through the point where another bottle is located.

One of the subroutines of the robot control program is the route planning. You are to write the program to determine the minimal length of robot route needed to collect all the bottles from the table.

Input

The first line of the input file contains two integer numbers w and l — the width and the length of the table ($2 \le w, l \le 1000$).

The second line of the input contains an integer number n — the number of bottles on the table ($1 \le n \le 18$). Each of the following n lines contains two integer numbers x_i and y_i — coordinates of the i-th bottle ($0 < x_i < w$; $0 < y_i < l$). No two bottles are located at the same point.

The last line of the input file contains two integer numbers x_r and y_r — coordinates of the robot's initial position ($0 < x_r < w$; $0 < y_r < l$). Robot is not located at the same point with a bottle.

Output

Output the length of the shortest route of the robot. Your answer should be accurate within an absolute error of 10^{-6}.

Example

kitchen.in	kitchen.out
3 4 2 1 1 2 3 2 1	5.60555127546399

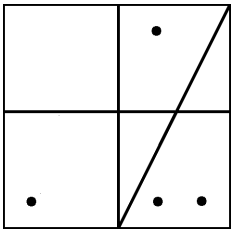
Problem A
Cookie

Source file name: `cookie.c`, `cookie.cpp` *or* `cookie.java`

The *Association of Cookie Makers* (ACM) has developed its newest and most decadent cookie in history, the ''Square Chocolate Chip Cookie''. This new cookie is a biscuit delight with lots and lots of chocolate chips for all the cookie lovers out there. However, in the development of this revolutionary type of cookie, the chocolate chips sprinklers have had some problems; though all the chips fall on the cookie, they are not evenly spread.

With a tradition of more than 26 years of cookie making, ACM sprinklers cannot be replaced. That is why the sales department has decided to cut the cookie in several pieces to maximize revenue. Each cut is made by a laser that moves across the cookie following a straight line path. Any chip chocolate on the laser path is destroyed during the cut process. Each piece of a square cookie is then valued according to its *chocolate chip density*, defined as the number of chips that it has, divided by its area.

The following figure shows an original cookie with four chips and three cuts. It is apparent that the piece at the lower right corner has a maximal chocolate chip density.



As part of the product testing team at ACM, you have to write a program that detects the *maximal chocolate chip density* among the resulting pieces, given an original square chocolate cookie, the positions on it that received a chocolate chip, and the cuts that should be made.

Input

The input consists of several test cases. The first line in a test case contains three integers L, C, and K, separated by blanks, where L is the length of the cookie side, C is the number of chips, and K is the number of cuts ($2 \leq L \leq 500$, $0 \leq C \leq 5000$, $0 \leq K \leq 50$). Each one of the following C lines contains two blank-separated integers $\hat{x}$ and $\hat{y}$, representing the (x, y) coordinates of a chocolate chip ($0 < \hat{x} < L$, $0 < \hat{y} < L$) in a Cartesian coordinate system where the cookie is represented by the square with vertices $(0, 0)$, $(L, 0)$, (L, L), and $(0, L)$. Each one of the following K lines contains four blank-separated integers x_1, y_1, x_2, and y_2, representing two distinct points on a straight line that defines a cut on the same coordinate system ($0 \leq x_1, y_1, x_2, y_2 \leq L$). You may suppose that there are not two chocolate chips placed at the same location. The last test case is followed by a line containing three zeros.

The input must be read from the file cookie.in.

Output

For each case, output a single line with a number indicating the maximal chocolate chip density among all the resulting pieces after the cuts. You may suppose that no answer will be greater than 10^2. The answer should be formatted and approximated to three decimal places. The floating point delimiter must be '.' (i.e., the dot). The rounding applies towards the *nearest neighbor* unless both neighbors are equidistant, in which case the result is rounded up (e.g., 78.3712 is rounded to 78.371; 78.5766 is rounded to 78.577; 78.3745 is rounded to 78.375, etc.).

The output must be written to standard output.

Sample input	**Output for the sample input**

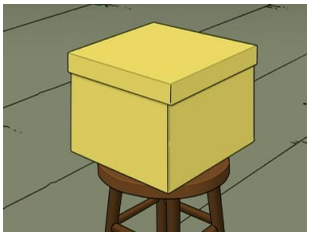
`6 4 3`
`1 1`
`4 1`
`5 1`
`4 5`
`3 0 3 6`
`0 3 6 3`
`3 0 6 6`
`0 0 0`
	`0.296`

Problem F
The Farnsworth Parabox

Source file name: `parabox.c`, `parabox.cpp` *or* `parabox.java`

Professor Farnsworth, a renowned scientist that lives in year 3000 working at *Planet Express Inc.*, performed a failed experiment that nearly killed him. As a sub-product, some strange *boxes* were created. Farnsworth gave one of the boxes to Leela, who accidentally discovered that it leads to a *parallel universe*. After that, the *Planet Express* crew traveled to the new discovered parallel universe using the box, meeting their corresponding parallel copies, including a parallel Professor Farnsworth who also created some boxes.

Simultaneously, some parallel copies of the Professor created similar boxes in some existing parallel universes. As a result, some universes, including the original one, were endowed with a (possibly empty) collection of boxes leading to other parallel universes. However, the boxes have a bug: besides allowing travels among different parallel universes, they allow for time travels. So, a particular box leads to a distinct parallel universe possibly allowing a voyager to gain or lose a certain number of time units.



One of the boxes invented by Farnsworth Professor, from *Futurama*.

More precisely, given two distinct universes A and B, and a non-negative integer number t, a (A, B)-box with *time displacement* t is an object designed to travel between the two universes that can be used *directly* (traveling from A to B) or *reversely* (traveling from B to A). A such box exists in both universes, allowing travels among both universes. A voyager that uses the (A, B)-box directly can travel from universe A to universe B landing t time units in the future. On the other hand, a voyager that uses the (A, B)-box reversely can travel from universe B to universe A landing t time units in the past. Box building requires so much energy that there may be built at most one box to travel between a given pair of different universes.

A *circuit* is defined as a non-empty sequence of parallel universes $\langle s_1, s_2, \ldots, s_m \rangle$ such that:

- The first and the last universe in the sequence are the same (i.e., $s_1{=}s_m$).
- For every k ($1{\leq}k{<}m$) there is a (s_k, s_{k+1})-box or a (s_{k+1}, s_k)-box to travel (directly or reversely) from universe s_k to universe s_{k+1}.

The possible existence of circuits is very interesting. Using the corresponding boxes of a circuit, a voyager may experiment real time travels. Professor Farnsworth wants to know if there is a circuit that starts in the original universe and allows travels to the past, constituting a phenomenon

known as the *Farnsworth Parabox*. For example, imagine that there are three universes, A, B and C, and that there exist the following boxes: a (A, B)-box with time displacement 3, a (A, C)-box with time displacement 2, and a (B, C)-box with time displacement 4. Clearly, the sequence $\langle A, B, C, A\rangle$ is a circuit, that allows to travel five time units in the future, starting and ending at universe A.

The original Farnsworth Professor, who lives in the original universe, wants to know if the *Farnsworth Parabox* is true or not. Can you help him?

Input

There are several cases to solve. Each case begins with a line with two integer numbers N and B, indicating the number of parallel universes (including the original) and the number of existing boxes, respectively ($2 \leq N \leq 10^2$, $1 \leq B \leq N \cdot (N-1)/2$). The distinct universes are identified uniquely with the numbers $1, 2, \ldots, N$, where the original universe is the number 1. Each one of the next B lines contains three integer numbers i, j and t, describing a (i, j)-box to travel between the universe i and the universe j with time displacement t ($1 \leq i \leq N$, $1 \leq j \leq N$, $i \neq j$, $0 \leq t \leq 10^2$). The input ends with a line with two 0 values.

The input must be read from the file parabox.in.

Output

For each test case output one line with the letter '`Y`' if the *Farnsworth Parabox* is true; or with the letter '`N`', otherwise.

The output must be written to standard output.

Sample input	**Output for the sample input**
2 1 2 1 1 3 3 1 2 3 1 3 2 2 3 4 4 4 1 2 2 3 2 2 3 4 2 1 4 2 0 0	N Y N

Problem G
Square Garden

Source file name: `garden.c, garden.cpp` *or* `garden.java`

The farmer Rick has a square garden of side L meters long, divided in a grid with L^2 square modules, each one of side 1 meter long. Rick wants to cultivate N modules of the garden and he knows that the production will be better if the cultivated area receives more water. He uses drip irrigation technology, which is done by means of hoses installed along the perimeter of the cultivated area.

It should be clear that there are different ways to choose the N modules that should be cultivated. The following figure shows two ways to cultivate N=8 modules in a square garden with side L=3. The left diagram shows a cultivated surface with a perimeter of length 16; the right one depicts another possibility, with perimeter 12.

Rick wants to maximize the perimeter of the selected area in order to optimize the production. Then you have been hired to help him determining the largest perimeter that an area of N modules of the garden may attain.

Input

There are several cases to consider. Each case is described with a line with two integer numbers L and N separated by one blank, indicating the length of the garden's side and the number of modules that must be cultivated, respectively ($1 \leq L \leq 10^6$, $0 \leq N \leq L^2$). Input ends with a line with two 0 values.

The input must be read from the file garden.in.

Output

For each case, output a line with the maximum perimeter that can be achieved.

The output must be written to standard output.

Sample input	**Output for the sample input**
1 0 1 1 2 3 3 8 0 0	0 4 8 16

Problem H
Teamface

Source file name: `teamface.c, teamface.cpp` *or* `teamface.java`

Teamface is the new social network from NewPie High School, whose purpose is to increase collaboration among team members from all their sport teams. This new application is one of the new actions taken to conquer all national championships and ratify NewPie's place as the top school in the country.

In Teamface, members of teams can befriend each other under the following rules:

1. There are N students in NewPie teams ($N>0$), identified with numbers 1, 2, ..., N.
2. A student can belong to only one of the T NewPie teams ($T>0$).
3. All teams have the same number of members, M ($2 \leq M \leq N$).
4. Every team has exactly one *captain* (one of its members).
5. All members of a given team are Teamface friends among them. They can share strategies, post-match comments and watch replays online.
6. A student must have less than $\lfloor M/2 \rfloor$ friends not in his/her team (this is meant to prevent distractions).

Following the NewPie honor code, all players are registered in Teamface and have added their friends according to these rules.

This year's sponsors have requested the list of players for each team in order to send the training uniforms for the next training session, to be held on Monday. They need a list that contains, for each team, the number of training uniforms of each size. Information about team names and team captains was received correctly, but due to an administrative error at NewPie, the players' information was apparently incomplete. Indeed, NewPie sent, for each player, his/her identifier, size, and a list with the Teamface friends' identifiers.

Today is Saturday afternoon and nobody at NewPie can answer any question. But your boss, one of the NewPie sponsors, realized that it is possible to build the required list without any further information. He has chosen you to do this as soon as possible.

Input

There are several cases to solve. Each case begins with a line with two integer numbers T and N, indicating the number of teams and the number of students, respectively ($1 \leq T \leq 100$, $1 \leq N \leq 5000$). Then, T lines follow, each one containing a non-empty string w and an integer number i ($1 \leq i \leq N$), where w represents the name of a NewPie Team and i represents the identifier of its captain. You may assume that w does not contain any blank and that its length does not exceed 30 characters. This is followed by N lines, one per NewPie student. Each one of these lines contains the student's identifier j ($1 \leq j \leq N$), his/her uniform size as an integer s_j ($1 \leq s_j \leq 50$), an integer number f_j indicating how many friends j has ($0 \leq f_j < N$), and the list

containing the identifiers of his/her Teamface friends (that list does not have repeated elements and does not include the identifier j). Every line in the input has no leading blanks and its corresponding data is separated by one blank. The input ends with a line with two 0 values.

The input must be read from the file teamface.in.

Output

The output for each case starts with "`Case` i`:`" on a single line, where i is the case number starting at 1. For each team in the test case (in the same order that they are given in the input), your program should output the name of the team on a single line. Then it should output a line with two integers t and n, where n is the number of uniforms needed for a given size t ($1 \leq t \leq 50$, $1 \leq n$), for all distinct sizes needed in the team. Sizes should be printed in ascending order.

The output must be written to standard output.

Sample input	**Output for the sample input**
3 12 LightningSalsa 1 PinkChiguiro 6 MightyPiranha 12 1 16 4 2 3 4 5 2 14 4 1 3 4 6 3 14 4 1 2 4 9 4 14 3 1 2 3 5 16 4 1 6 7 8 6 14 4 2 5 7 8 7 14 3 5 6 8 8 18 4 5 6 7 10 9 16 4 3 10 11 12 10 16 4 8 9 11 12 11 14 3 9 10 12 12 14 3 9 10 11 2 6 TamalSuperDragon 1 ArequipeNinjas 4 1 12 2 2 3 2 14 2 1 3 3 18 2 1 2 4 20 2 5 6 5 22 2 4 6 6 24 2 4 5 0 0	Case 1: LightningSalsa 14 3 16 1 PinkChiguiro 14 2 16 1 18 1 MightyPiranha 14 2 16 2 Case 2: TamalSuperDragon 12 1 14 1 18 1 ArequipeNinjas 20 1 22 1 24 1

Problem J
Lazy Professor

Source file name: `lazyprof.c, lazyprof.cpp` *or* `lazyprof.java`

Professor Yzal does not like to grade the exams of his students, so that he assigns this task to his assistants. Final students' grades are defined as weighted sums of the grades obtained in the exams. But Professor Yzal does not define *a priori* the exam's relative weights. Instead of that, only when the assistants end their job (and all the students' grades in each exam are known) Professor Yzal decides the exam's weights. He does it trying to be pleasant with the whole class, so that the assigned weights should maximize the average final grade that the class obtains, and expecting that eventual complaints about his grading criteria will be as few as possible. And finally, maybe rewarding harder exams, Yzal defines that the weight of every particular exam should lie within a specific range of values.

This term is about to finish and you were hired to help Yzal with his lazy grading job. Your task is to write a program that, given the students' grades and the *reasonable range* of weights for each exam, determines the maximum possible average according to the following rules:

- Each exam must have a weight, defined as an integer number in the closed interval $[0, 100]$, describing the assigned percentage.
- The weight of an exam i must be in an integer closed interval $[min_i, max_i]$ that describes the corresponding reasonable range previously defined by Yzal.
- The sum of all exam weights must be 100.
- The final grade of each student is a weighted sum calculated as the sum of his/her own grades previously multiplied by the corresponding exam weight divided by 100.
- The exam weights are so chosen that the average of the students' final grades is maximized.

Input

There are several cases to consider. Each test case is described as follows:

- A line with two integer numbers S and N, separated by a blank, ($1 \leq S \leq 100$, $1 \leq N \leq 20$), where S corresponds to the number of students and N corresponds to the number of exams in the course.
- Each one of the following S lines describes the grades of each student, containing N integer numbers $c_{j1}, c_{j2}, \ldots, c_{jN}$, separated by a blank, ($0 \leq c_{ji} \leq 100$ for each $1 \leq i \leq N$), where c_{ji} indicates the grade obtained by the student j in the exam i ($1 \leq j \leq S$, $1 \leq i \leq N$).
- Each one of the following N lines describes the reasonable range of weights of each exam, containing two integer numbers min_i, max_i, separated by a blank ($0 \leq min_i \leq max_i \leq 100$), where min_i and max_i represent the minimum and maximum weight that can be assigned to the exam i, respectively ($1 \leq i \leq N$).

You can suppose that

$$\sum_{i=1}^{N} min_i \leq 100, \quad \text{and} \quad \sum_{i=1}^{N} max_i \geq 100.$$

The last test case is followed by a line containing two zeros.

The input must be read from the file lazyprof.in.

Output

For each given case, output a single line with a number indicating the maximum possible average of the students' final grades if the weights are defined according to the rules. The answer should be formatted and approximated to two decimal places. The floating point delimiter must be '.' (i.e., the dot). The rounding applies towards the *nearest neighbor* unless both neighbors are equidistant, in which case the result is rounded up (e.g., 78.312 is rounded to 78.31; 78.566 is rounded to 78.57; 78.345 is rounded to 78.35, etc.).

The output must be written to standard output.

Sample input	**Output for the sample input**
1 1 0 0 100 2 2 50 90 70 50 0 100 0 100 2 2 50 90 70 50 30 70 30 70 2 2 50 90 70 50 50 50 50 50 2 2 73 52 92 81 20 50 60 80 0 0	0.00 70.00 67.00 65.00 72.90

Problem D

Diccionário Portuñol

Problem code name: `diccionario`

Portuñol is a special language that was naturally developed in Latin America. Since almost half of Latin America speaks Portuguese (Português) and almost half speaks Spanish (Español), the mixing of both languages is natural.

Each word in Portuñol is made by taking a non-empty prefix of a Portuguese word and a non-empty suffix of a Spanish word, and concatenating them together. A *prefix* of a word is any word that can be obtained by erasing zero or more characters from its right end. A *suffix* of a word is any word that can be obtained by erasing zero or more characters from its left end. The name of the language itself comes from taking a prefix of the word "Português" (Portu) and a suffix of the word "Español" (ñol), and concatenating them.

Of course, not every possible way of combining two words will result in something meaningful, or even pronounceable, but that is not important. We want you to write a program to count the number of different Portuñol words.

You will be given two non-empty sets of words to test your program. The first set will represent Portuguese words and the second set will represent Spanish words. You need to calculate the number of different Portuñol words that can be made using the prefix and suffix rule described above. Note that the same word may be constructed in several ways, but it still needs to be counted as one. Also note that the input sets are just to test your program, so they do not need to be made out of actual Portuguese or Spanish words.

Input

Each test case is described using several lines. The first line contains two integers P and S representing respectively the number of Portuguese words and the number of Spanish words ($1 \leq P, S \leq 1000$). Each of the next P lines contains a Portuguese word, and after that each of the next S lines contains a Spanish word. Each word is a non-empty string of at most 1000 characters; each character is one of the 26 standard lowercase letters (from '`a`' to '`z`'). You may assume that within each test case no two Portuguese words are the same, and that the sum of the lengths of all the Portuguese words is at most 10^5. The same holds for the Spanish words.

The last test case is followed by a line containing two zeros.

Output

For each test case output a line with an integer representing the number of different words that can be constructed by concatenating a non-empty prefix of a word in the first set (Portuguese words) and a non-empty suffix of a word in the second set (Spanish words).

Sample input	**Output for the sample input**
3 3 mais grande mundo mas grande mundo 1 5 a aaaaa aaaaaa aaaaaaa a aaaaaaaaa 1 1 abc abc 0 0	182 9 8

Problem E
Electrical Pollution

Problem code name: `electrical`

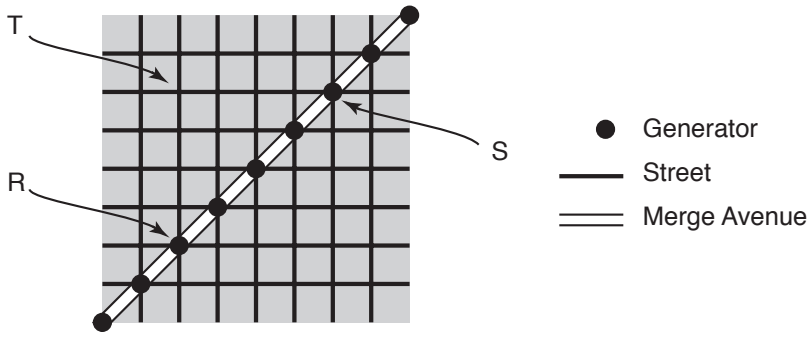
Sortonia is the capital of the North Nlogonia province. The city is laid out with almost all of its streets in a square grid, aligned to either the North-South or the West-East direction. The only exception is Merge Avenue, which runs Southwest-Northeast, splitting city blocks along their diagonals.

Sortonia is also one of the greenest cities in Nlogonia. The local university developed technology to harness the magnetic field of Earth for energy generation. As a consequence, all intersections of Merge Avenue have power generators installed, supplying all the homes and businesses of the city.

This technology was praised by environmentalists at the time for eliminating Sortonia's carbon footprint, but soon after its introduction, thousands of bees and birds were found dead in the city. Puzzled, the Queen of Nlogonia ordered the queendom's biophysicists to investigate the phenomenon.

After many months of study, they discovered that the generators used by Sortonians created anomalies in the local magnetic field. The birds and bees that use the Earth's magnetic field to guide their flight were confused by these anomalies, started flying in circles and eventually died of exhaustion.

According to the biophysicists' theoretical models, each generator creates an anomaly that is represented as an integer value. Each anomaly propagates indefinitely in all four compass directions. Points that are not directly north, south, west or east of the generator are unaffected by it. On the other hand, if a point is aligned with two generators then the anomaly at that point is the sum of the two anomalies produced by those generators. As an example, consider the picture below that represents a certain portion of Sortonia. The anomaly at point R is just the one produced by the generator at that point, while the anomaly at point T is the sum of the anomalies produced by the generator at point R and the generator at point S.



The biophysicists would like to measure the anomalies for some city intersections, but these measurements require expensive equipment and technical expertise. So they plan to measure only a subset of the city's intersections and extrapolate other data from them. Predicting an anomaly from a set of measurements might require combining several of them in complicated ways. Thus, the Queen ordered you to write a program that predicts the anomalies at certain intersections, given the measurements previously made.

Input

Each test case is described using several lines. The first line contains two integers M and Q representing respectively the number of measurements and the number of queries ($1 \leq M, Q \leq 10^4$). Each of the next M lines describes a measurement using three integers X, Y and A, indicating that the measured anomaly at point (X, Y) is A ($-10^7 \leq X, Y \leq 10^7$ and $-10^4 \leq A \leq 10^4$). After that, each of the

next Q lines describes a query using two integers X' and Y', indicating that the anomaly at point (X', Y') must be predicted ($-10^7 \leq X', Y' \leq 10^7$). All positions are measured in city blocks; the first coordinate increases from West to East, while the second coordinate increases from South to North. Point $(0, 0)$ is located on Merge Avenue. You may assume that within each test case each point is not measured more than once. Likewise, each point is not queried more than once. You may also assume that all the measurements are consistent.

The last test case is followed by a line containing two zeros.

Output

For each test case output $Q + 1$ lines. In the i-th line write the answer to the i-th query. If the information given by the measurements is enough to predict the anomaly at the queried point, then write an integer representing the predicted anomaly at the queried point. Otherwise write the character '`*`' (asterisk). Print a line containing a single character '`-`' (hyphen) after each test case.

Sample input	**Output for the sample input**
3 3 30 -10 3 30 20 15 40 20 2 -10 40 40 -10 -10 -10 6 8 0 1 11 0 3 8 1 0 11 3 0 8 4 4 0 3 5 6 1 5 0 3 3 0 4 3 0 2 2 4 4 4 5 5 0 0	-10 -10 * - 9 8 8 * * * 0 * -

Problem F

File Retrieval

Problem code name: `file`

The operating system of your computer indexes the files on your hard disk based on their contents, and provides textual search over them. The content of each file is a non-empty string of lowercase letters. To do a search, you specify a key, which is also a non-empty string of lowercase letters. The result is a list of all the files that contain the key as a substring. A string s is a substring of a string t if t contains all characters of s as a contiguous sequence. For instance, "foofoo", "cafoo", "foota" and "foo" all contain "foo" as a substring, while "foa", "fofo", "fioo" and "oofo" do not.

You know the content of each file on your hard disk, and wonder whether each subset of the files is searchable. A subset of the files is searchable if there exists at least one key that produces exactly the list of those files as a result. Given the contents of the files on your hard disk, you are asked to compute the number of non-empty searchable subsets.

Input

Each test case is described using several lines. The first line contains an integer F representing the number of files on your hard disk ($1 \leq F \leq 60$). Each of the next F lines indicates the content of one of the files. The content of a file is a non-empty string of at most 10^4 characters; each character is one of the 26 standard lowercase letters (from '`a`' to '`z`').

The last test case is followed by a line containing one zero.

Output

For each test case output a line with an integer representing the number of non-empty searchable subsets.

Sample input	**Output for the sample input**
6 form formal malformed for man remake 3 cool cool old 0	11 3

Problem G

Garden Fence

Problem code name: `garden`

Gary is a careful gardener that has a rectangular field full of trees. There are two kinds of trees in his land: pines and larches. To improve their vitality, he decided to start using a specific fertilizer for each kind of tree, instead of the generic fertilizer he was using so far.

Since Gary has many trees, fertilizers cannot be placed individually on each tree. For this reason he decided to build a fence to separate the field in two, and use the pine fertilizer on one side and the larch fertilizer on the other side. The new fence will be built over a straight line connecting two distinct points located on the boundary of the land.

Sadly, each fertilizer is great for the kind of tree it is intended, but deadly for the other. After building the fence and deciding which fertilizer will be used on each side, larches in pines' side and pines in larches' side will be cut down, to prevent a slow death that will ruin the landscape. Furthermore, before building the fence it is necessary to cut down trees of any kind lying directly over the line where the fence will be located.

Of course, Gary loves his trees. Depending on their kind, age and other factors, each tree has a certain value. The gardener wants to build the fence and select where to use each fertilizer in such a way that his loss is minimized, where the loss is the sum of the values of the trees that will be cut down.

You were hired to build the fence. Before starting your work, tell Gary how much he will lose when choosing optimally the location of the fence and the fertilizer for each side.

Input

Each test case is described using several lines. The first line contains two integers P and L, representing respectively the number of pines and the number of larches ($1 \leq P, L \leq 1000$). Each of the next P lines describes a pine. After that, each of the next L lines describes a larch. Trees are modeled as points in the XY plane. Each tree is described using three integers X, Y and V, where X and Y are the coordinates of the tree ($-10^5 \leq X, Y \leq 10^5$), and V is its value ($1 \leq V \leq 1000$). You may assume that within each test case no two trees have the same location.

The last test case is followed by a line containing two zeros.

Output

For each test case output a line with an integer representing the minimum possible loss for the gardener.

Sample input	**Output for the sample input**
2 3 2 2 10 4 4 10 2 4 10 4 2 10 3 3 10 2 3 2 2 20 4 4 20 2 4 10 4 2 10 3 3 10 1 1 -10000 -10000 1000 10000 10000 1000 2 2 0 0 4 0 2 2 0 1 3 0 4 1 4 1 0 1 1000 0 -1 1000 1 0 1000 -1 0 1000 0 0 1 0 0	10 20 0 2 1

Problem H

Hedge Mazes

Problem code name: `hedge`

The Queen of Nlogonia is a fan of mazes, and therefore the queendom's architects built several mazes around the Queen's palace. Every maze built for the Queen is made of rooms connected by corridors. Each corridor connects a different pair of distinct rooms and can be transversed in both directions.

The Queen loves to stroll through a maze's rooms and corridors in the late afternoon. Her servants choose a different challenge for every day, that consists of finding a simple path from a start room to an end room in a maze. A simple path is a sequence of *distinct* rooms such that each pair of consecutive rooms in the sequence is connected by a corridor. In this case the first room of the sequence must be the start room, and the last room of the sequence must be the end room. The Queen thinks that a challenge is good when, among the routes from the start room to the end room, *exactly* one of them is a simple path. Can you help the Queen's servants to choose a challenge that pleases the Queen?

For doing so, write a program that given the description of a maze and a list of queries defining the start and end rooms, determines for each query whether that choice of rooms is a good challenge or not.

Input

Each test case is described using several lines. The first line contains three integers R, C and Q representing respectively the number of rooms in a maze ($2 \leq R \leq 10^4$), the number of corridors ($1 \leq C \leq 10^5$), and the number of queries ($1 \leq Q \leq 1000$). Rooms are identified by different integers from 1 to R. Each of the next C lines describes a corridor using two distinct integers A and B, indicating that there is a corridor connecting rooms A and B ($1 \leq A < B \leq R$). After that, each of the next Q lines describes a query using two distinct integers S and T indicating respectively the start and end rooms of a challenge ($1 \leq S < T \leq R$). You may assume that within each test case there is at most one corridor connecting each pair of rooms, and no two queries are the same.

The last test case is followed by a line containing three zeros.

Output

For each test case output $Q + 1$ lines. In the i-th line write the answer to the i-th query. If the rooms make a good challenge, then write the character '`Y`' (uppercase). Otherwise write the character '`N`' (uppercase). Print a line containing a single character '`-`' (hyphen) after each test case.

Sample input	**Output for the sample input**
6 5 3 1 2 2 3 2 4 2 5 4 5 1 3 1 5 2 6 4 2 3 1 2 2 3 1 4 1 3 1 2 0 0 0	Y N N - N Y Y -

Problem B: Crosswords Insider

You have an insider at the New York Times who sends you a list of the answers for the crossword puzzle, but his list does not say which answer goes with which clue. His list occasionally contains errors or omissions, but is usually correct. Given the list of answers and the shape of the crossword puzzle, figure out a valid assignment.

Input

The input will consist of one or more problem sets.

Each problem set begins with a line containing two integers `M` and `N`. Zero values for these will indicate the end of the problem sets. `M` denotes the number of words to be placed into the puzzle and will be in the range $1 \ldots 150$. `N` denotes the number of rows in the puzzle and will be in the range $1 \ldots 16$.

This is followed by `M` lines, each containing a single word, left-justified on the line. A word will contain only alphabetic characters and words will not be duplicated within any problem set. Words will be $2 \ldots 16$ characters in length.

This is followed by `N` lines denoting a puzzle template as a series of '.' and '#' characters, left-justified and followed immediately by the end of line. Each line will contain the same number of these characters. That number may range from $1 \ldots 16$. A '.' indicates a position in the puzzle where a character may be written. A '#' indicates a position at which a character may not appear.

Output

Your program should determine if all the given words can be arranged into the puzzle template in such a way as to leave no '.' positions unfilled and without filling any '#' positions. Vertical and horizontal words may intersect at a common letter, but two horizontal or two vertical words may neither intersect nor be adjacent to one another without at least one intervening '#'.

Your program should print the string "Problem " followed by the problem set number. If no solution is possible, it should then print, on the remainder of that output line, the string ": No layout is possible.".

If a solution is possible, then the program should, beginning on the next line after the problem set number, print the `N` lines of the crossword puzzle as presented in the input but with the appropriate characters substituted for the '.' positions.

If multiple solutions are possible, you may print any solution.

Example

Input

```
4 4
tow
cat
row
care
```

```
...
.#.
...
.##
0 0
```

Output:

```
Problem 1
cat
a#o
row
e##
Problem 2: no layout is possible.
```

Problem C: Doors and Penguins

The organizers of the Annual Computing Meeting have invited a number of vendors to set up booths in a large exhibition hall during the meeting to showcase their latest products. As the vendors set up their booths at their assigned locations, they discovered that the organizers did not take into account an important fact — each vendor supports either the Doors operating system or the Penguin operating system, but not both. A vendor supporting one operating system does not want a booth next to one supporting another operating system.

Unfortunately the booths have already been assigned and even set up. There is no time to reassign the booths or have them moved. To make matter worse, these vendors in fact do not even want to be in the same room with vendors supporting a different operating system.

Luckily, the organizers found some portable partition screens to build a wall that can separate the two groups of vendors. They have enough material to build a wall of any length. The screens can only be used to build a straight wall. The organizers need your help to determine if it is possible to separate the two groups of vendors by a single straight wall built from the portable screens. The wall built must not touch any vendor booth (but it may be arbitrarily close to touching a booth). This will hopefully prevent one of the vendors from knocking the wall over accidentally.

Input

The input consists of a number of cases. Each case starts with 2 integers on a line separated by a single space: D and P, the number of vendors supporting the Doors and Penguins operating system, respectively ($1 \leq D, P \leq 500$). The next D lines specify the locations of the vendors supporting Doors. This is followed by P lines specifying the locations of the vendors supporting Penguins. The location of each vendor is specified by four positive integers: x_1, y_1, x_2, y_2. (x_1, y_1) specifies the coordinates of the southwest corner of the booth while (x_2, y_2) specifies the coordinates of the northeast corner. The coordinates satisfy $x_1 < x_2$ and $y_1 < y_2$. All booths are rectangular and have sides parallel to one of the compass directions. The coordinates of the southwest corner of the exhibition hall is $(0, 0)$ and the coordinates of the northeast corner is $(15000, 15000)$. You may assume that all vendor booths are completely inside the exhibition hall and do not touch the walls of the hall. The booths do not overlap or touch each other.

The end of input is indicated by $D = P = 0$.

Output

For each case, print the case number (starting from 1), followed by a colon and a space. Next, print the sentence:

```
It is possible to separate the two groups of vendors.
```

if it is possible to do so. Otherwise, print the sentence:

```
It is not possible to separate the two groups of vendors.
```

Print a blank line between consecutive cases.

Example

Input

```
3 3
10 40 20 50
50 80 60 90
30 60 40 70
30 30 40 40
50 50 60 60
10 10 20 20
2 1
10 10 20 20
40 10 50 20
25 12 35 40
0 0
```

Output:

```
Case 1: It is possible to separate the two groups of vendors.

Case 2: It is not possible to separate the two groups of vendors.
```

Problem G: Zoned Out

"You want to build an office building in our town? Wonderful! We're so glad to have you!" The mayor was beaming as he led me back into City Hall. "We just need to make sure your building permit application meets all the town zoning regulations. We can check that immediately."

He led me through a door labelled "Zoning". I was staggered by the sight of a huge room with row after row of desks. At each desk sat a clerk with a logbook, an inbox, and a generaous supply of pencils and erasers.

The mayor took my application form, then flourished a sheet of paper containing several lines of numbered boxes. He placed the paper in the inbox of the nearest desk and said, "We just run this past our zoning clerks, and if it gets back to this desk with all the boxes checked, then your application is approved." The clerk at the desk grabbed the form from the inbox, marked checks in a few boxes, wrote a few lines into a logbook on his desk, then ran to a nearby copier, made copies of the form and distributed to several other desks. I watched as the clerks at those other desks each added more checks but also erased some, wrote in their logbooks, made copies, and distributed the altered forms. Soon copies of the form seemed to be flying all around the room.

I turned to the mayor and said, "But none of them actually *looked* at my application!".

"Oh no", said the Mayor, "each of our clerks has carefully studied a few parts of our zoning regulations. Some are convinced that *any* application will automatically satisfy some of the zoning rules. They also believe that *no* application will ever satisfy some of the other rules. So each clerk simply checks off some boxes and erases the check marks from some others."

"We're very proud", he added, "of the decisive nature of our zoning process. We recruit our staff from the nearby law school and choose only the most officious and argumentative junior students."

"How," I asked, "do they decide where the marked-up copies should go?"

"I'm not really sure," said the mayor. "I think they mainly give them to their fellow clerks that they most dislike, trying to increase the workload of their enemies." Just then a sheet of paper was placed on the closest desk. "Let's see how you're coming along", he said, picking up the paper. "No, still four boxes left empty. We'll just have to wait and see if you can do better."

I sighed and said, "It seems to me that this could go on forever."

"No", said the mayor. "Each clerk keeps a logbook recording all the versions of your application that they have sent on. When they get a form, they start by adding all marks they have ever seen on prior versions of the form, then do their own marks and erasures. If, after all that, it matches one of the previous versions, they won't send it on a second time."

I resigned myself to a long wait.

Input

The input consists of one or more problem sets.

The first line in each set contains the number of boxes on the form and the number of clerks in the office. Both are positive integers. The end of input is signalled by a line with zeros for both of these numbers.

The remainder of the input consists of three lines of data for each clerk, starting with clerk #0. (Boxes and clerks are both identified by number, starting at 0.) Each line will contain 0 or more integers, separated by whitespace, as follows:

- One line contains the numbers of the boxes that always marked by that clerk.
- The next line contains the numbers of the boxes that are always erased by that clerk.
- The third line contains the numbers of all those clerks to whom this one sends copies of the altered form.

Output

For each problem set, print a single line containing the numbers of the boxes marked on the last copy of the form that leaves the desk of clerk #0. The numbers should be printed in ascending order, separated by a single blank space.

Example

Input

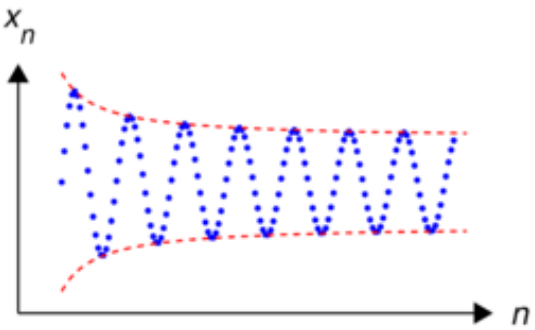
```
6 5
1

1 2
2 3
4
4
5
1
3
2

1
4
2
1 0
0 0
```

Output:

```
1 3 4 5
```

Problem B

Mega Inversions



The n^2 upper bound for any sorting algorithm is easy to obtain: just take two elements that are misplaced with respect to each other and swap them. Conrad conceived an algorithm that proceeds by taking not two, but *three* misplaced elements. That is, take three elements $a_i > a_j > a_k$ with $i < j < k$ and place them in order a_k, a_j, a_i. Now if for the original algorithm the steps are bounded by the maximum number of inversions $\frac{n(n-1)}{2}$, Conrad is at his wits' end as to the upper bound for such triples in a given sequence. He asks you to write a program that counts the number of such triples.

Input specifications

The first line of the input is the length of the sequence, $1 \leq n \leq 10^5$.
The next line contains the integer sequence $a_1, a_2, \ldots, a_n$.
You can assume that all $a_i \in [1, n]$.

Output specifications

Output the number of inverted triples.

Sample input 1	Sample output 1
3 1 2 3	0

Sample input 2	Sample output 2
4 3 3 2 1	2

Problem F

Knigs of the Forest

All moose are knigs of the forest, but your latest moose-friend, Karl-Älgtav, is more interesting than most. In part because of his fondness of fermented blueberries, and in part because of the tribe he lives in. Each year his tribe holds a tournament to determine that year's alpha-moose. The winner gets to mate with all the moose-chicks, and then permanently leaves the tribe. The pool of contenders stays constant over the years, apart from the old alpha-moose being replaced by a newcomer in each tournament.

Karl-Älgtav has recently begun to wonder when it will be his turn to win all the chicks, and has asked you to help him determine this. He has supplied a list of the strength of each of the other male moose in his tribe that will compete during the next $n-1$ years, along with their time of entry into the tournament. Assuming that the winner each year is the moose with greatest strength, determine when Karl-Älgtav becomes the alpha-moose.

Input specifications

The first line of input contains two space separated integers k ($1 \leq k \leq 10^5$) and n ($1 \leq n \leq 10^5$), denoting the size of the tournament pool and the number of years for which you have been supplied sufficient information.
Next is a single line describing Karl-Älgtav, containing the two integers y ($2011 \leq y \leq 2011 + n - 1$) and p ($0 \leq p \leq 2^{31} - 1$). These are his year of entry into the tournament and his strength, respectively.
Then follow $n + k - 2$ lines describing each of the other moose, in the same format as for Karl-Älgtav.
Note that exactly k of the moose will have 2011 *as their year of entry, and that the remaining $n - 1$ moose will have unique years of entry.*
You may assume that the strength of each moose is unique.

Output specifications

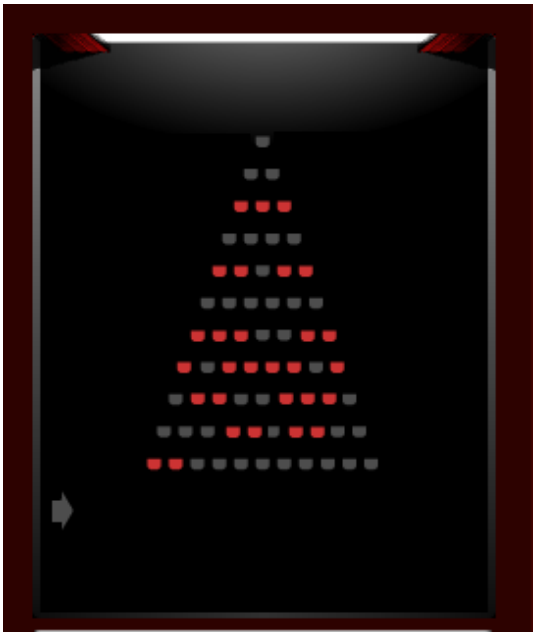
The year Karl-Älgtav wins the tournament, or `unknown` if the given data is insufficient for determining this.

Sample input 1	**Sample output 1**
2 4 2013 2 2011 1 2011 3 2014 4 2012 6	2013

Sample input 2	**Sample output 2**
2 4 2011 1 2013 2 2012 4 2011 5 2014 3	unknown

Problem H

Private Space



People are going to the movies in groups (or alone), but normally only care to socialize within that group. Being Scandinavian, each group of people would like to sit at least one space apart from any other group of people to ensure their privacy, unless of course they sit at the end of a row. The number of seats per row in the cinema starts at X and decreases with one seat per row (down to a number of 1 seat per row). The number of groups of varying sizes is given as a vector $(N_1, \ldots, N_n)$, where N_1 is the number of people going alone, N_2 is the number of people going as a pair etc. Calculate the seat-width, X, of the widest row, which will create a solution that seats all (groups of) visitors using as few rows of seats as possible. The cinema also has a limited capacity, so the widest row may not exceed 12 seats.

Input specifications

The first line of input contains a single integer n ($1 \leq n \leq 12$), giving the size of the largest group in the test case.
Then follows a line with n integers, the i-th integer (1-indexed) denoting the number of groups of i persons who need to be seated.

Output specifications

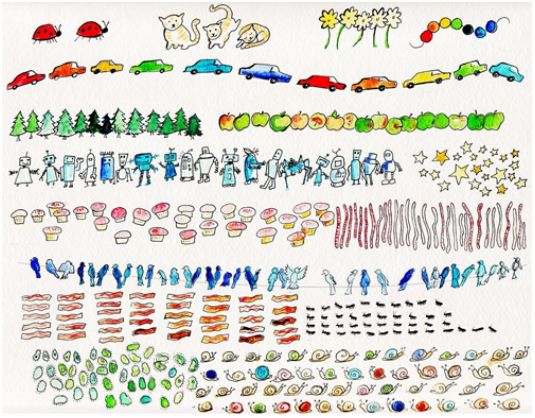
A single number; the size of the smallest widest row that will accommodate all the guests. If this number is greater than 12, output `impossible` instead.

Sample input 1	Sample output 1
3 0 1 1	3

Sample input 2	Sample output 2
3 2 1 1	4

Problem I

Prime Time



Odd and Even have had their share of fun times playing the good old prime game:
They start with an arbitrary natural number, and take turns either adding 1 or dividing by a prime (assuming the result is still a natural number), and the one to reach 1 is the winner. However, now that they have a new friend, Ingmariay, they have decided to expand the rules of the game to allow for three-player action:
Instead of determining a winner for each round of play, they instead score points; the lowest number each of them has claimed during the round is the amount of points they get. (If any of them did not have the opportunity to claim any numbers, the starting number will be their score for that round.) At the end of the day, the player with the fewest points wins. And to avoid bad blood between themselves, they have all agreed that each of them only will focus on minimizing their own scores, and that whenever a player can choose different numbers that will result in the same score, that player will choose the lowest of those numbers. They have also agreed on a fixed order of play: Odd → Even → Ingmariay → ..., but they alternate who gets to start.

You recently missed one of their exciting evenings of play, because you had to make problems for the NCPC event. Fortunately for you, they had recorded the numbers and starting players for each round, and told you that since they always play optimally, you could use this to simulate the event for yourself. Oh joy!

As an example round, assume that Even is chosen as the starting player, and with the starting number 15. Then Even claims 16, Ingmariay 8, Odd 4, Even 2 and Ingmariay 1. Odd gets 4 points, Even 2 and Ingmariay 1.

Input specifications

The first line of input contains a single integer n ($1 \leq n \leq 1000$), the number of rounds they played that evening.
Then follow n lines each beginning with the first character of the name of the starting player (either 'O', 'E' or 'I'), followed by a space and then the starting number for that round, in the range $[1, 10000]$
Note: If the starting number is 1*, all players receive* $'$ *point for that round.*

Output specifications

Output a single line with the score at the end of the day for each of the three contestants, in the order "Odd", "Even", "Ingmariay".

Sample input 1	Sample output 1
1 O 4	2 1 4

Sample input 2	Sample output 2
3 O 13 I 14 E 15	6 29 16

Problem J

Enemy Division

Picture by The U.S. Army.

Captain Keram has to make a difficult decision. It is year 2147 and there is a big war in the world. His soldiers have been together since the war started, two years ago, and some of them have become enemies. Luckily, each soldier has at most 3 enemies.

They need to attack another country soon, and Keram is worried that soldiers who are enemies might not cooperate well during the battle. He has decided to divide them into groups such that every soldier has at most one enemy in his group. He also wants to make it simple, so he wants to use as few groups as possible. Can you divide the soldiers into groups for him?

Input specifications

On the first line there are two integers n and m, $2 \leq n \leq 100\,000, 0 \leq m \leq 3n/2$, where n is the number of soldiers and m is the number of enemy pairs. Then follow m lines, each containing two space separated integers a_i, b_i, denoting that soldiers a_i and b_i are enemies, where $1 \leq a_i < b_i \leq n$. You can assume that all soldiers have at most 3 enemies.

Output specifications

The first line of output contains the minimal number of groups of soldiers k. Each of the next k lines contains a space separated list of a soldiers in a unique group.

Sample input 1	Sample output 1
4 4 1 2 2 3 3 4 1 4	2 1 3 2 4

Problem A

Arranging Heaps

A mining company extracts terbium, a rare metal used for constructing lightweight magnets, from river sand. They mine the Long River at N mining points, each of them identified by its distance from the river source. At each mining point, a relatively small but highly valued heap of mineral ore is extracted from the river.

To collect the mineral ore, the company regroups the N produced heaps into a smaller number of K heaps, each located at one of the initial mining points. The newly formed heaps are then collected by trucks.

To regroup the N heaps, they use a barge, which in practice can carry any amount of mineral ore because it is very large. The barge starts at the river source and can only travel downriver, so the heap produced at a mining point X can be taken to a mining point Y only if $Y > X$. Each heap is moved completely to another mining point, or not moved at all. The cost of moving a heap of weight W from a mining point X to a mining point Y is $W \times (Y - X)$. The total cost of the regrouping is the sum of the costs for each heap movement. Notice that a heap which is not moved has no influence on the total cost.

Given the values for N and K, the N mining points, and the weight of the heap each mining point produced, write a program that calculates the minimum total cost to regroup the N initial heaps into K heaps.

Input

Each test case is described using several lines. The first line contains two integers N and K denoting respectively the number of initial heaps and the desired number of heaps after regrouping ($1 \leq K < N \leq 1000$). Each of the next N lines describes one of the initial heaps with two integers X and W indicating that the mining point X produced a heap of weight W ($1 \leq X, W \leq 10^6$). Within each test case the heaps are given in strictly ascending order considering their mining points.

Output

For each test case output a line with an integer representing the minimum total cost to regroup the N initial heaps into K heaps.

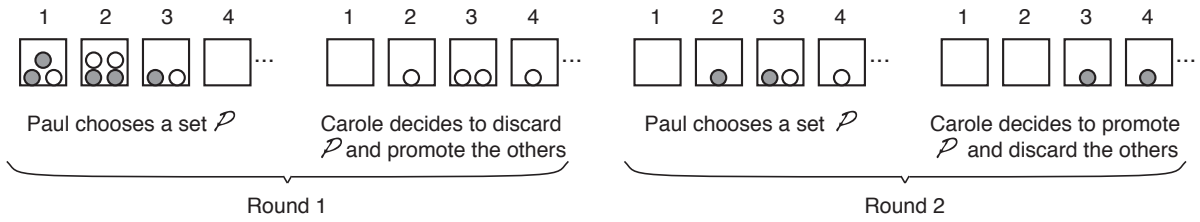
Sample input	**Output for the sample input**
3 1 20 1 30 1 40 1 3 1 11 3 12 2 13 1 6 2 10 15 12 17 16 18 18 13 30 10 32 1 6 3 10 15 12 17 16 18 18 13 30 10 32 1	30 8 278 86

Problem B

Boxes and Stones

Paul and Carole like to play a game with S stones and B boxes numbered from 1 to B. Before beginning the game they arbitrarily distribute the S stones among the boxes from 1 to $B-1$, leaving box B empty. The game then proceeds by rounds. At each round, first Paul chooses a subset $\mathcal{P}$ of the stones that are in the boxes; he may choose as many stones as he wants from as many boxes as he wants, or he may choose no stones at all, in which case $\mathcal{P}$ is empty. Then, Carole decides what to do next: she can either *promote* the subset $\mathcal{P}$ and *discard* the remaining stones (that is, those stones not chosen by Paul in the first step); or she may *discard* the subset $\mathcal{P}$ and *promote* the remaining stones.

To *promote* a given subset means to take each stone in this subset and move it to the box with the next number in sequence, so that if there was a stone in this subset inside box b, it is moved to box $b+1$. To *discard* a given subset means to remove every stone in this subset from its corresponding box, so that those stones are not used in the game for the remaining rounds. The figure below shows an example of the first two rounds of a game.



Paul and Carole play until at least one stone reaches box number B, in which case Paul wins the game, or until there are no more stones left in the boxes, in which case Carole wins the game. Paul is a very rational player, but Carole is a worthy rival because she is not only extremely good at this game, but also quite lucky. We would like to know who is the best player, but before that we must first understand how the outcome of a game depends on the initial distribution of the stones. In particular, we would like to know in how many ways the S stones can initially be distributed among the first $B-1$ boxes so that Carole can be certain that she can win the game if she plays optimally, even if Paul never makes a mistake.

Input

Each test case is described using one line. The line contains two integers S ($1 \leq S \leq 200$) and B ($2 \leq B \leq 100$), representing respectively the number of stones and the number of boxes in the game.

Output

For each test case output a line with an integer representing the number of ways in which the S stones may be distributed among the first $B-1$ boxes so that Carole is certain that she can win the game. Because this number can be very large, you are required to output the remainder of dividing it by 10^9+7.

Sample input	**Output for the sample input**
2 3 8 4 42 42	2 0 498467348

Problem C

Cellphone Typing

A research team is developing a new technology to save time when typing text messages in mobile devices. They are working on a new model that has a complete keyboard, so users can type any single letter by pressing the corresponding key. In this way, a user needs P keystrokes to type a word of length P.

However, this is not fast enough. The team is going to put together a dictionary of the common words that a user may type. The goal is to reduce the average number of keystrokes needed to type words that are in the dictionary. During the typing of a word, whenever the following letter is uniquely determined, the cellphone system will input it automatically, without the need for a keystroke. To be more precise, the behavior of the cellphone system will be determined by the following rules:

1. The system never guesses the first letter of a word, so the first letter always has to be input manually by pressing the corresponding key.

2. If a non-empty succession of letters $c_1c_2\ldots c_n$ has been input, and there is a letter c such that every word in the dictionary which starts with $c_1c_2\ldots c_n$ also starts with $c_1c_2\ldots c_nc$, then the system inputs c automatically, without the need of a keystroke. Otherwise, the system waits for the user.

For instance, if the dictionary is composed of the words "`hello`", "`hell`", "`heaven`" and "`goodbye`", and the user presses "`h`", the system will input "`e`" automatically, because every word which starts with "`h`" also starts with "`he`". However, since there are words that start with "`hel`" and with "`hea`", the system now needs to wait for the user. If the user then presses "`l`", obtaining the partial word "`hel`", the system will input a second "`l`" automatically. When it has "`hell`" as input, the system cannot guess, because it is possible that the word is over, or it is also possible that the user may want to press "`o`" to get "`hello`". In this fashion, to type the word "`hello`" the user needs three keystrokes, "`hell`" requires two, and "`heaven`" also requires two, because when the current input is "`hea`" the system can automatically input the remainder of the word by repeatedly applying the second rule. Similarly, the word "`goodbye`" needs just one keystroke, because after pressing the initial "`g`" the system will automatically fill in the entire word. In this example, the average number of keystrokes needed to type a word in the dictionary is then $(3 + 2 + 2 + 1)/4 = 2.00$.

Your task is, given a dictionary, to calculate the average number of keystrokes needed to type a word in the dictionary with the new cellphone system.

Input

Each test case is described using several lines. The first line contains an integer N representing the number of words in the dictionary ($1 \leq N \leq 10^5$). Each of the next N lines contains a non-empty string of at most 80 lowercase letters from the English alphabet, representing a word in the dictionary. Within each test case all words are different, and the sum of the lengths of all words is at most 10^6.

Output

For each test case output a line with a rational number representing the average number of keystrokes needed to type a word in the dictionary. The result must be output as a rational number with exactly two digits after the decimal point, rounded if necessary.

Sample input	**Output for the sample input**
4 hello hell heaven goodbye 3 hi he h 7 structure structures ride riders stress solstice ridiculous	2.00 1.67 2.71

Problem E
Environment Protection

Arsenic & Cyanide Mining (ACM) is a corporation that has recently decided to start developing its mines in the lands near your hometown. As a member of the citizen's regulatory committee for ACM's operations, your task is to control how much the corporation can mine from those lands, so that you get to keep the jobs and other benefits without sacrificing the environment and the health of the local residents.

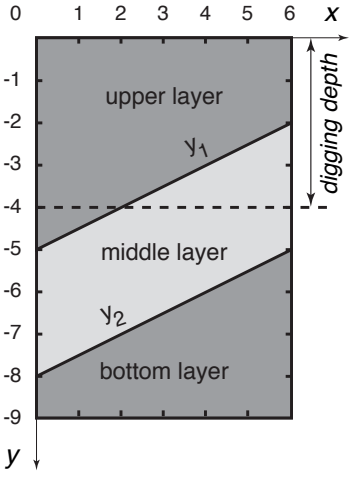
The ACM has plans to mine several rectangular patches of land. A patch of land has width W, can be dug up to a maximum depth D, and has a flat surface which we consider to be at depth 0. The minerals in a patch are organized in three layers, which may vary in their depth along the width of the patch, but always have the same profile along its whole length. This is why the ACM is only interested in the profile along the width of each patch, and has performed exploratory work in order to precisely determine its shape. As a result, they discovered that the two interfaces between the three layers of minerals can be represented by two functions $y_1(x)$ and $y_2(x)$, where the first describes the boundary between the top layer and the middle layer, and the second describes the boundary between the middle layer and the bottom layer. These functions are always such that

$$-D < y_2(x) < y_1(x) < 0 \qquad \text{for} \qquad 0 \le x \le W \ ,$$

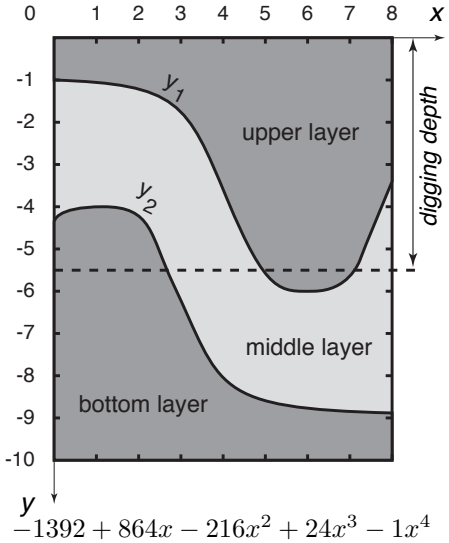
so that the layers' boundaries never touch each other. Besides, each function has the form $y_i(x) = p_i(x)/q_i(x)$, where

$$p_i(x) = \sum_{k=0}^{K} P_{i,k}\, x^k \qquad \text{and} \qquad q_i(x) = \sum_{k=0}^{K} Q_{i,k}\, x^k \ ,$$

for $i = 1, 2$ and a certain integer K. The figure below shows the profiles of two patches of land in the way the ACM represents them. The patch on the left has width $W = 6$ and depth $D = 9$, while the patch on the right has $W = 8$ and $D = 10$. The boundaries of the layers of each patch are described by the functions defined below them.



$$y_1(x) = \frac{-10 + 1x}{2 + 0x}$$
$$y_2(x) = \frac{-16 + 1x}{2 + 0x}$$



$$y_1(x) = \frac{-1392 + 864x - 216x^2 + 24x^3 - 1x^4}{1312 - 864x + 216x^2 - 24x^3 + 1x^4}$$
$$y_2(x) = \frac{-73 + 36x - 54x^2 + 36x^3 - 9x^4}{17 - 4x + 6x^2 - 4x^3 + 1x^4}$$

The ACM will dig everything in a patch of land up to a certain digging depth d, and then sell all the minerals thus obtained to make a profit. However, the minerals in the top and the bottom layers

are essentially worthless, so the profit of the whole operation comes exclusively from those minerals in the middle layer. In fact, the profit is proportional to the area A of the middle layer in the profile that is at depth at most d. Given the description of a patch of land and an integer A, you would like to know the digging depth d you should allow the ACM to dig the patch so that they get an area of minerals from the middle layer in the profile of exactly A. In the figure above you can see the answer for the two test cases in the sample input. For the patch on the left, in order to get an area $A = 4$ the digging depth must be $d = 4.00000$, while for the patch on the right an area $A = 14$ requires a digging depth $d = 5.51389$.

Input

Each test case is described using five lines. The first line contains four integers W, D, A and K, where W is the width of the patch of land the ACM wants to mine ($1 \leq W \leq 8$), D is its depth ($1 \leq D \leq 10$), A is the area of the middle layer in the profile that the ACM must get ($1 \leq A \leq W \times D$), and K allows the definition of the interfaces $y_1(x)$ and $y_2(x)$ as explained above ($0 \leq K \leq 8$). Each of the other lines contains $K + 1$ integers between -10^8 and 10^8, inclusive. The second line contains the coefficients of $p_1(x)$ from $P_{1,0}$ to $P_{1,K}$. The third line contains the coefficients of $q_1(x)$ from $Q_{1,0}$ to $Q_{1,K}$. The fourth line contains the coefficients of $p_2(x)$ from $P_{2,0}$ to $P_{2,K}$. The fifth line contains the coefficients of $q_2(x)$ from $Q_{2,0}$ to $Q_{2,K}$. Within each test case A is strictly less than the total area of the middle layer in the profile, and there exists a single value d such that a digging depth d yields an area of minerals from the middle layer in the profile of exactly A. Besides, $q_1(x) \neq 0$, $q_2(x) \neq 0$ and $-D < y_2(x) < y_1(x) < 0$, for $0 \leq x \leq W$.

Output

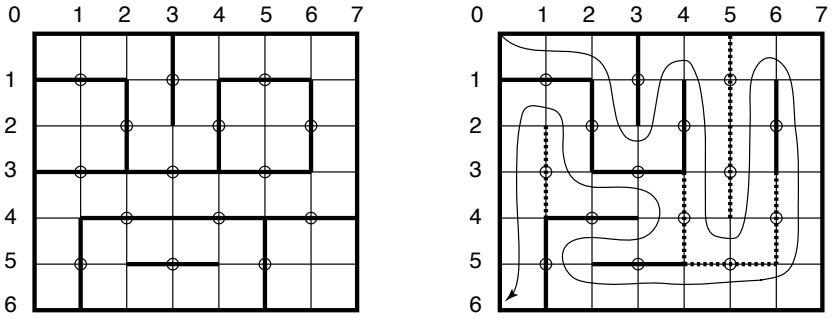
For each test case output a line with a rational number representing the depth d that the ACM should be allowed to dig the patch of land so that they get an area of minerals from the middle layer in the profile of exactly A. The result must be output as a rational number with exactly five digits after the decimal point, rounded if necessary.

Sample input	**Output for the sample input**
6 9 4 1 -10 1 2 0 -16 1 2 0 8 10 14 4 -1392 864 -216 24 -1 1312 -864 216 -24 1 -73 36 -54 36 -9 17 -4 6 -4 1	4.00000 5.51389

Problem F

Fix the Pond

A shrimp farm uses a rectangular pond built as a grid with $2N$ rows and $2N+1$ columns of square cells, for a given integer N. Each cell side is one meter long. The pond has exactly $(2N-1) \times N$ barriers of length two meters, used to temporarily isolate smaller sections inside the pond for breeding different kinds of shrimp. The barriers have their middle points fixed precisely at the integer coordinates (a, b), for all $0 < a < 2N$ and $0 < b < 2N+1$, where both a and b are odd, or both are even. Each barrier can be rotated around its middle point to change the pond configuration; however, by being rotated, a barrier switches between only two possible positions, always parallel to the pond sides, vertical or horizontal. The left part of the figure below shows a pond configuration, with $N = 3$.



At the end of every season the pond is closed for maintenance and cleaning. It must then be reconfigured so that a special machine can sweep the pond floor. The machine starts its work at the top left cell, and needs to pass through every cell exactly once, finishing in the bottom left cell. The right part of the figure shows one such reconfiguration, where six barriers were switched. For this example, though, four barrier switches would have been enough.

You must write a program that given a pond configuration, determines the minimum number of barrier switches needed to reconfigure the pond as specified above. There is always at least one possible way to reconfigure the pond as specified.

Input

Each test case is described using several lines. The first line contains an integer N indicating that the pond has $2N$ rows and $2N+1$ columns ($1 \leq N \leq 300$). Each of the next $2N-1$ lines contains a string of N characters describing the orientation of the barriers. In the i-th line, the j-th character indicates the orientation of the barrier whose middle point is at coordinates $(i, 2j-1)$ if i is odd, or $(i, 2j)$ if i is even, for $i = 1, 2, \ldots, 2N-1$ and $j = 1, 2, \ldots, N$. The character is the uppercase letter "`V`" if the orientation is vertical, or the uppercase letter "`H`" if it is horizontal.

Output

For each test case output a line with an integer representing the minimum number of barrier switches needed to reconfigure the pond as specified.

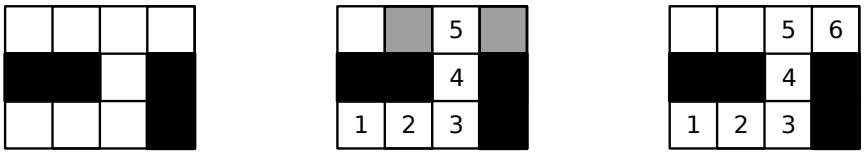
Sample input	**Output for the sample input**
3 HVH VVV HHH HHH VHV 1 H 1 V	4 0 1

Problem G

Game of Tiles

The Game of Tiles is a game for two players played over a rectangular board in the form of a table of R rows and C columns of square cells called tiles. At the beginning of the game, some of the tiles may be painted black and the rest remain white. Then, Player 1 and Player 2 alternate turns making a move and the first one that cannot make a valid move loses the game. The first move of the game is done by Player 1 and consists of choosing a white tile and writing the number 1 on it. After that, each subsequent move i consists of writing number i on an unused white tile that is adjacent horizontally or vertically (but not diagonally) to the tile numbered $i - 1$. Note that Player 1 always writes odd numbers and Player 2 always writes even numbers.

The following figure shows three examples of possible configurations of a board with $R = 3$ and $C = 4$ during a game. On the left it shows the initial configuration. On the center it shows an intermediate state, where cells in gray mark the possible moves for Player 2. And on the right it shows the configuration when the game is won by Player 2, who chose the appropriate move.



Your task is to write a program that given the initial configuration of the board, determines which player will win, if both of them play optimally.

Input

Each test case is described using several lines. The first line contains two integers R and C representing respectively the number of rows and columns of the board ($1 \leq R, C \leq 50$). The i-th of the next R lines contains a string B_i of C characters that describes the i-th row of the initial board. The j-th character of B_i is either "`.`" (dot) or the uppercase letter "`X`", representing that the tile at row i and column j is respectively white or black. Within each test case at least one of the tiles is white.

Output

For each test case output a line with an integer representing the number of the player (1 or 2) who will win the game if both of them play optimally.

Sample input	**Output for the sample input**
3 4 .... XX.X ...X 3 4 .... .X.X ...X 3 4 .... .X.X .... 1 1 . 1 11 ....X......	2 1 1 1 2

Problem J

Joining Couples

Air traffic regulations in Nlogonia require that each city must register exactly one outbound flight to another city. Passengers can use this flight only in the direction registered, that is, there may be a flight registered from city X to city Y and no flight registered from city Y to city X. Thus, the number of registered flights is equal to the number of cities. This rule, as one can imagine, makes air travel somewhat complicated, but tradition and a strong ruling by the Queen makes any changes difficult. Besides, some companies even make a profit from the problems caused by the rule.

The Association for Couple Matching (ACM) is setting up a new service to help customers find their long lasting soulmates: the Internet Connecting Program for Couples (ICPC). The service consists of computing the minimum total number of flights a couple needs to take to meet one another (perhaps in a city where neither of them lives in). Assuming the couple's starting cities are A and B, the agency will try to find a city C such that C is reachable by air travel from both A and B, and the sum of the number of flights needed to go from A to C and the number of flights needed to go from B to C is minimized. Note that C may be equal to A or B or both.

You will be given the list of all available flights, and a list of queries consisting of pairs of cities where the members of a couple live. For each query, you must compute the minimum total number of flights that are needed for them to meet.

Input

Each test case is described using several lines. The first line contains an integer N representing the number of cities ($2 \leq N \leq 10^5$). Cities are identified by different integers from 1 to N. The second line contains N integers F_i, where F_i indicates that the registered outbound flight from city i is to city F_i ($1 \leq F_i \leq N$, $F_i \neq i$ for $i = 1, 2, \ldots, N$). The third line contains an integer Q representing the number of queries ($1 \leq Q \leq 10^5$). Each of the next Q lines describes a query with two integers A and B indicating the couple's starting cities ($1 \leq A, B \leq N$). Within each test case, if it is possible to travel by air from city X to city Y, the maximum number of flights needed to do so is 10^4.

Output

For each test case output Q lines. In the i-th line write an integer with the answer to the i-th query. If the corresponding couple can meet by air travel, write the minimum total number of flights that the couple must take to meet one another; if it is impossible for the couple to meet by air travel, write the number -1.

Sample input	**Output for the sample input**
3 2 1 2 3 1 2 1 3 1 1 7 2 1 4 5 3 5 6 5 1 3 4 7 7 4 6 2 2 1	1 2 0 -1 3 3 -1 1

Problem A

Phone List

Given a list of phone numbers, determine if it is consistent in the sense that no number is the prefix of another. Let's say the phone catalogue listed these numbers:

- Emergency 911
- Alice 97 625 999
- Bob 91 12 54 26

In this case, it's not possible to call Bob, because the central would direct your call to the emergency line as soon as you had dialled the first three digits of Bob's phone number. So this list would not be consistent.

Input specifications

The first line of input gives a single integer, $1 \leq t \leq 40$, the number of test cases. Each test case starts with n, the number of phone numbers, on a separate line, $1 \leq n \leq 10000$. Then follows n lines with one unique phone number on each line. A phone number is a sequence of at most ten digits.

Output specifications

For each test case, output "YES" if the list is consistent, or "NO" otherwise.

Sample input

```
2
3
911
97625999
91125426
5
113
12340
123440
12345
98346
```

Output for sample input

```
NO
YES
```

Problem B

Cuckoo Hashing

One of the most fundamental data structure problems is the dictionary problem: given a set D of words you want to be able to quickly determine if any given query string q is present in the dictionary D or not. Hashing is a well-known solution for the problem. The idea is to create a function $h : \Sigma^* \rightarrow [0..n-1]$ from all strings to the integer range $0, 1, .., n-1$, i.e. you describe a fast deterministic program which takes a string as input and outputs an integer between 0 and $n-1$. Next you allocate an empty hash table T of size n and for each word w in D, you set $T[h(w)] = w$. Thus, given a query string q, you only need to calculate $h(q)$ and see if $T[h(q)]$ equals q, to determine if q is in the dictionary. Seems simple enough, but aren't we forgetting something? Of course, what if two words in D map to the same location in the table? This phenomenon, called collision, happens fairly often (remember the Birthday paradox: in a class of 24 pupils there is more than 50% chance that two of them share birthday). On average you will only be able to put roughly $\sqrt{n}$-sized dictionaries into the table without getting collisions, quite poor space usage!

A stronger variant is Cuckoo Hashing[1]. The idea is to use two hash functions h_1 and h_2. Thus each string maps to two positions in the table. A query string q is now handled as follows: you compute both $h_1(q)$ and $h_2(q)$, and if $T[h_1(q)] = q$, *or* $T[h_2(q)] = q$, you conclude that q is in D. The name "Cuckoo Hashing" stems from the process of creating the table. Initially you have an empty table. You iterate over the words d in D, and insert them one by one. If $T[h_1(d)]$ is free, you set $T[h_1(d)] = d$. Otherwise if $T[h_2(d)]$ is free, you set $T[h_2(d)] = d$. If both are occupied however, just like the cuckoo with other birds' eggs, you evict the word r in $T[h_1(d)]$ and set $T[h_1(d)] = d$. Next you put r back into the table in its alternative place (and if that entry was already occupied you evict that word and move it to its alternative place, and so on). Of course, we may end up in an infinite loop here, in which case we need to rebuild the table with other choices of hash functions. The good news is that this will not happen with great probability even if D contains up to $n/2$ words!

Input specifications

On the first line of input is a single positive integer $1 \leq t \leq 50$ specifying the number of test cases to follow. Each test case begins with two positive integers $1 \leq m \leq n \leq 10000$ on a line of itself, m telling the number of words in the dictionary and n the size of the hash table in the test case. Next follow m lines of which the i:th describes the i:th word d_i in the dictionary D by two non-negative integers $h_1(d_i)$ and $h_2(d_i)$ less than n giving

[1]Cuckoo Hashing was suggested by the danes R. Pagh and F. F. Rödler in 2001

the two hash function values of the word d_i. The two values may be identical.

Output specifications

For each test case there should be exactly one line of output either containing the string "`successful hashing`" if it is possible to insert all words in the given order into the table, or the string "`rehash necessary`" if it is impossible.

Sample input

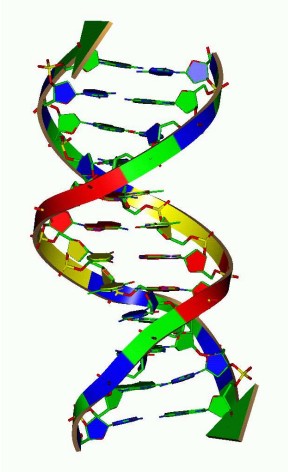
```
2
3 3
0 1
1 2
2 0
5 6
2 3
3 1
1 2
5 1
2 5
```

Output for sample input

```
successful hashing
rehash necessary
```

Problem D

Copying DNA



Evolution is a seemingly random process which works in a way which resembles certain approaches we use to get approximate solutions to hard combinatorial problems. You are now to do something completely different.

Given a DNA string S from the alphabet {`A`,`C`,`G`,`T`}, find the minimal number of copy operations needed to create another string T. You may reverse the strings you copy, and copy both from S and the pieces of your partial T. You may put these pieces together at any time. You may only copy contiguous parts of your partial T, and all copied strings must be used in your final T. Example: From S = "`ACTG`" create T = "`GTACTATTATA`"

1. Get `GT.........` by copying and reversing "`TG`" from S.
2. Get `GTAC.......` by copying "`AC`" from S.
3. Get `GTAC...TA..` by copying "`TA`" from the partial T.
4. Get `GTAC...TAAT` by copying and reversing "`TA`" from the partial T.
5. Get `GTACAATTAAT` by copying "`AAT`" from the partial T.

Input specifications

The first line of input gives a single integer, $1 \leq t \leq 100$, the number of test cases. Then follow, for each test case, a line with the string S of length $1 \leq m \leq 18$, and a line with the string T of length $1 \leq n \leq 18$.

Output specifications

Output for each test case the number of copy operations needed to create T from S, or "`impossible`" if it cannot be done.

Sample input

```
5
ACGT
GTAC
A
C
ACGT
TGCA
ACGT
TCGATCGA
A
AAAAAAAAAAAAAAAAA
```

Output for sample input

```
2
impossible
1
4
6
```

Problem E

Circle of Debt

The three friends Alice, Bob, and Cynthia always seem to get in situations where there are debts to be cleared among themselves. Of course, this is the "price" of hanging out a lot: it only takes a few resturant visits, movies, and drink rounds to get an unsettled balance. So when they meet as usual every Friday afternoon they begin their evening by clearing last week's debts. To satisfy their mathematically inclined minds they prefer clearing their debts using as little money transaction as possible, i.e. by exchanging as few bank notes and coins as necessary. To their surprise, this can sometimes by harder than it sounds. Suppose that Alice owes Bob 10 crowns and this is the three friends' only uncleared debt, and Alice has a 50 crown note but nothing smaller, Bob has three 10 crown coins and ten 1 crown coins, and Cynthia has three 20 crown notes. The best way to clear the debt is for Alice to give her 50 crown note to Cynthia, Cynthia to give two 20 crown notes to Alice and one to Bob, and Bob to give one 10 crown coin to Cynthia, involving a total of only five notes/coins changing owners. Compare this to the straightforward solution of Alice giving her 50 crown note to Bob and getting Bob's three 10 crown notes and all his 1 crown coins for a total of fourteen notes/coins being exchanged!

Input specifications

On the first line of input is a single positive integer, $1 \leq t \leq 50$, specifying the number of test cases to follow. Each test case begins with three integers $ab, bc, ca \leq 1000$ on a line of itself. ab is the amount Alice owes Bob (negative if it is Bob who owes Alice money), bc the amount Bob owes Cynthia (negative if it is Cynthia who is in debt to Bob), and ca the amount Cynthia owes Alice (negative if it is Alice who owes Cynthia).

Next follow three lines each with six non-negative integers $a_{100}, a_{50}, a_{20}, a_{10}, a_5, a_1$, $b_{100}, \ldots, b_1$, and $c_{100}, \ldots c_1$, respectively, where a_{100} is the number of 100 crown notes Alice got, a_{50} is the number of her 50 crown notes, and so on. Likewise, $b_{100}, \ldots, b_1$ is the amount of notes/coins of different value Bob got, and $c_{100}, \ldots, c_1$ describes Cynthia's money. Each of them has at most 30 coins (i.e. $a_{10}+a_5+a_1, b_{10}+b_5+b_1$, and $c_{10}+c_5+c_1$ are all less than or equal to 30) and the total amount of all their money together (Alice's plus Bob's plus Cynthia's) is always less than 1000 crowns.

Output specifications

For each test case there should be one line of output containing the minimum number of bank notes and coins needed to settle the balance. If it is not possible at all, output the string "`impossible`".

Sample input

```
3
10 0 0
0 1 0 0 0 0
0 0 0 3 0 10
0 0 3 0 0 0
-10 -10 -10
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
-10 10 10
3 0 0 0 2 0
0 2 0 0 0 1
0 0 1 1 0 3
```

Output for sample input

```
5
0
impossible
```

Problem F

Full Tank?

After going through the receipts from your car trip through Europe this summer, you realised that the gas prices varied between the cities you visited. Maybe you could have saved some money if you were a bit more clever about where you filled your fuel?

To help other tourists (and save money yourself next time), you want to write a program for finding the cheapest way to travel between cities, filling your tank on the way. We assume that all cars use one unit of fuel per unit of distance, and start with an empty gas tank.

Input specifications

The first line of input gives $1 \leq n \leq 1000$ and $0 \leq m \leq 10000$, the number of cities and roads. Then follows a line with n integers $1 \leq p_i \leq 100$, where p_i is the fuel price in the ith city. Then follow m lines with three integers $0 \leq u, v < n$ and $1 \leq d \leq 100$, telling that there is a road between u and v with length d. Then comes a line with the number $1 \leq q \leq 100$, giving the number of queries, and q lines with three integers $1 \leq c \leq 100$, s and e, where c is the fuel capacity of the vehicle, s is the starting city, and e is the goal.

Output specifications

For each query, output the price of the cheapest trip from s to e using a car with the given capacity, or "`impossible`" if there is no way of getting from s to e with the given car.

Sample input

```
5 5
10 10 20 12 13
0 1 9
0 2 8
1 2 1
1 3 11
2 3 7
2
10 0 3
20 1 4
```

Output for sample input

```
170
impossible
```

Problem G

Nested Dolls

Dilworth is the world's most prominent collector of Russian nested dolls: he literally has thousands of them! You know, the wooden hollow dolls of different sizes of which the smallest doll is contained in the second smallest, and this doll is in turn contained in the next one and so forth. One day he wonders if there is another way of nesting them so he will end up with fewer nested dolls? After all, that would make his collection even more magnificent! He unpacks each nested doll and measures the width and height of each contained doll. A doll with width w_1 and height h_1 will fit in another doll of width w_2 and height h_2 if and only if $w_1 < w_2$ and $h_1 < h_2$. Can you help him calculate the smallest number of nested dolls possible to assemble from his massive list of measurements?

Input specifications

On the first line of input is a single positive integer $1 \leq t \leq 20$ specifying the number of test cases to follow. Each test case begins with a positive integer $1 \leq m \leq 20000$ on a line of itself telling the number of dolls in the test case. Next follow $2m$ positive integers $w_1, h_1, w_2, h_2, \ldots, w_m, h_m$, where w_i is the width and h_i is the height of doll number i. $1 \leq w_i, h_i \leq 10000$ for all i.

Output specifications

For each test case there should be one line of output containing the minimum number of nested dolls possible.

Sample input

```
4
3
20 30 40 50 30 40
4
20 30 10 10 30 20 40 50
3
10 30 20 20 30 10
4
10 10 20 30 40 50 39 51
```

Output for sample input

```
1
2
3
2
```

Problem I

Moogle

You got the original idea of making map software, called Moogle Maps, for the new cool Maple mPhone. It will even be capable of indicating the location of a house address like "Main Street 13". However, since the mPhone has limited storage capacity, you need to reduce the data amount. You don't want to store the exact location of every single house number. Instead only a subset of the house numbers will be stored exactly, and the others will be linearly interpolated. So you want to select house numbers that will minimise the average interpolation error, given how many house locations you have capacity to store. We view the street as a straight line, and you will always store the first and the last house location.

Given that you've stored the locations x_i and x_j for the houses with numbers i and j respectively, but no other house in between, the interpolated value for a house with number k with $i < k < j$ is $x_i + (x_j - x_i) \cdot \frac{k-i}{j-i}$.

Input specifications

The first line of input gives a single integer, $1 \leq t \leq 50$, the number of test cases.

For each test case, there are two lines. The first contains $2 \leq h \leq 200$ and $2 \leq c \leq h$, where h is the number of houses in the street and c is the number of house locations that can be stored. The second contains h integers in increasing order giving the location of the h houses. Each location is in the interval $[0, 1000000]$.

Output specifications

For each test case, output the average interpolation error over all the h houses for the optimal selection of c house locations to store. The output should be given with four decimal places, but we will accept inaccuracies of up to ± 0.001.

Sample input

```
2
4 3
0 9 20 40
10 4
0 10 19 30 40 90 140 190 202 210
```

Output for sample input

```
0.2500
0.3000
```

Problem A
Cookie Crumbs

Source file name: `cookie.c`, `cookie.cpp` *or* `cookie.java`

Cookie Monster likes chocolate chip cookies. He especially likes ones with lots of chocolate chips. Sometimes he likes to experiment with new, unusual cookies. The other day, he tried baking cookies in the shape of rectangles, and using rectangular chocolate chips. Unfortunately, since this was his first time attempting to make rectangular cookies, a few things went wrong. He used way too many chocolate chips. He also turned the oven on too high, and all the chocolate melted and leaked out, leaving only the cookie with holes where the chocolate chips used to be. When the chocolate chips melted, the cookie became disconnected into many cookie crumbs. Cookie Monster needs your help to count the crumbs to make sure he has not lost any in the oven.

Input

The first line of input contains the number of cases. Each case is described by several lines: the first line contanin four integers x_1, y_1, x_2, y_2, each between -1000000000 and 1000000000, giving the x- and y-coordinates of two opposite corners of the cookie. The sides of the cookie are parallel to the x and y coordinate axes. The second line contains an integer $0 \leq n \leq 100$, the number of chocolate chips in the cookie. The following n lines each describe one of the chocolate chips using four integers x_1, y_1, x_2, y_2, each between -1000000000 and 1000000000, the x- and y-coordinates of two opposite corners of the chocolate chip. The sides of each chocolate chip are parallel to the x and y coordinate axes. Chocolate chips can overlap, and they can be partially or completely outside the cookie. The cookie and each chocolate chip will have a non-zero area. A chocolate chip is considered to include the points on its perimeter; therefore, crumbs that would meet only at their corners are considered distinct crumbs.

The input must be read from standard input.

Output

For each test case output a single integer, the number of crumbs (disconnected components) that the cookie splits into after the chocolate chips have melted away.

The output must be written to standard output.

Sample input	**Output for the sample input**
1 0 0 100 100 2 0 0 50 50 50 50 100 100	2

Problem C
Buying Gas

Source file name: `buying.c, buying.cpp` *or* `buying.java`

Gasoline is expensive these days. Many people go out of their way to find the cheapest gas. You could write a computer program to help with this. But on the other hand, searching around too much can waste a lot of time. It could even happen that the driver searches so long that he runs out of gas, which is a major nuisance.

Deciding where to buy gas is even more important on a long trip. Such a trip already takes a long time, so you don't want to waste any more time than you have to. And the consequences of running out are all the more bothersome when far away from home.

Your task here is to write a program that finds the optimal places where to buy gas on a long trip. Of course, the answer must ensure that the car never runs out of gas. Furthermore, it must minimize the number of times that the car stops to fill up.

Input

The input contains several test cases. The first line of input contains the number of cases. Each case is described by several lines: the first line contains three integers $0 < n \leq 100$, $0 \leq m \leq 100000$, and $0 \leq d \leq 100000$, the capacity of the gas tank in litres, the number of gas stations along the route, and the total length of the trip in kilometres. The following m lines each contain two integers, the distance in kilometres from the starting point of the trip to the gas station, and the price of gas at the gas station in tenths of a cent per litre. The car begins the trip with a full tank of gas, and uses 0.1 litres of gas for each kilometre travelled.

The input must be read from standard input.

Output

Find the optimal set of gas stations at which to stop to get gas. For each test case output a single integer, the number of gas stations in this set. If it is not possible to make the trip without running out of gas, output the integer -1.

The output must be written to standard output.

Sample input	**Output for the sample input**
1 50 2 1000 500 1293 750 1337	1

Problem E
Paper Route

Source file name: `paper.c`, `paper.cpp` *or* `paper.java`

As a poor, tuition-ridden student, you've decided to take up a part time job as a paperboy/papergirl. You've just been handed your paper route: a set of addresses (conveniently labelled 1 to N).

Every morning, you start at the newspaper office (which happens to be address number 0). You have to plan a route to deliver a newspaper to every address - and you also want to get to class right after you're done. Conveniently, there are only N roads in your area connecting the addresses, and each of them takes a known time to traverse. Also, you've precalculated the time it takes to get to campus from each address, including the newspaper office (through some combination of biking, busing, or hitching a ride).

How soon can you be done delivering papers and be in your seat at school?

Input

The input contains several test cases. The first line of input contains the number of cases. Each case is described by several lines: First, there will be a single integer N (the number of addresses, $1 \leq N \leq 100000$). Next, there will be $N+1$ lines, each with an integer c_i (starting with $i = 0$, $0 \leq c_i \leq 1000000000$), the time it takes to get from location i to campus. Finally, the input will contain N lines, each with three integers a, b, c ($0 \leq a, b \leq N$, $a \neq b$, $0 \leq c \leq 1000$). Each of these lines describes a road between locations a and b taking c minutes to traverse.

It is guaranteed that you will be able to reach all the addresses. (Remember that location 0 is the newspaper office.)

The input must be read from standard input.

Output

For each test case output a single integer, the minimum time it will take to deliver all the papers and get to class.

The output must be written to standard output.

Sample input	**Output for the sample input**
2 1 3 4 0 1 1 0 2 2	7

Problem F
Party Location

Source file name: `party.c`, `party.cpp` *or* `party.java`

After the programming contest, all of the contestants would like to throw a party. After the party, however, it will be late, and the contestants will be too tired to walk a long way home. In particular, each contestant refuses to come to the party if it is more than 2.5 km from his or her house.

The solution is to hold the party as close to as many of the contestants' houses as possible. This is where you come in: your job is to determine the optimal location for the party, so that as many contestants as possible will be willing to attend it.

We consider the city to be a flat square, 50 km on each side. A contestant can walk directly from the party in a straight line to his or her house (there are no obstacles).

Input

The input contains several test cases separated by a blank line. The first line of input contains the number of cases. Each case consists of a number of lines, each containing two floating point numbers indicating the (x, y) coordinates of the house of one of the contestants. Each coordinate is between 0.0 and 50.0 (km). Each house is at a distinct location. There are at most 200 contestants.

The last line in the input is a blank line.

The input must be read from standard input.

Output

For each test case output consists of a single integer: the maximum number of contestants that can attend the party.

The output must be written to standard output.

Sample input	**Output for the sample input**
2 4.0 4.0 4.0 5.0 5.0 6.0 1.0 20.0 1.0 21.0 1.0 22.0 1.0 25.0 1.0 26.0 4.0 5.0 4.0 4.0 5.0 6.0 1.0 20.0 1.0 25.0 1.0 22.0 1.0 26.0 1.0 21.0	4 4

H - You'll be Working on the Railroad

Congratulations! Your county has just won a state grant to install a rail system between the two largest towns in the county — Acmar and Ibmar. This rail system will be installed in sections, each section connecting two different towns in the county, with the first section starting at Acmar and the last ending at Ibmar. The provisions of the grant specify that the state will pay for the two largest sections of the rail system, and the county will pay for the rest (if the rail system consists of only two sections, the state will pay for just the larger section; if the rail system consists of only one section, the state will pay nothing). The state is no fool and will only consider simple paths; that is, paths where you visit a town no more than once. It is your job, as a recently elected county manager, to determine how to build the rail system so that the county pays as little as possible. You have at your disposal estimates for the cost of connecting various pairs of cities in the county, but you're short one very important requirement — the brains to solve this problem. Fortunately, the lackeys in the computing services division will come up with something.

Input

Input will contain multiple test cases. Each case will start with a line containing a single positive integer $n \leq 50$, indicating the number of railway section estimates. (There may not be estimates for tracks between all pairs of towns.) Following this will be n lines each containing one estimate. Each estimate will consist of three integers s e c, where s and e are the starting and ending towns and c is the cost estimate between them. (Acmar will always be town 0 and Ibmar will always be town 1. The remaining towns will be numbered using consecutive numbers.) The costs will be symmetric, i.e., the cost to build a railway section from town s to town e is the same as the cost to go from town e to town s, and costs will always be positive and no greater than 1000. It will always be possible to somehow travel from Acmar to Ibmar by rail using these sections. A value of n = 0 will signal the end of input.

Output

For each test case, output a single line of the form

```
c1 c2 ... cm cost
```

where each ci is a city on the cheapest path and cost is the cost to the county (note c1 will always be 0 and cm will always be 1 and ci and ci+1 are connected on the path). In case of a tie, print the path with the shortest number of sections; if there is still a tie, pick the path that comes first lexicographically.

Sample Input

```
7
0 2 10
0 3 6
2 4 5
3 4 3
3 5 4
4 1 7
5 1 8
0
```

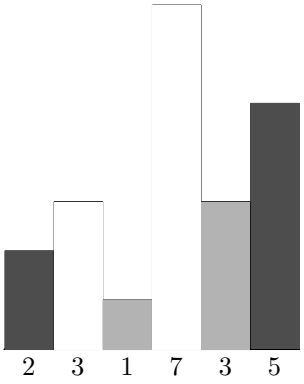
Sample Output

```
0 3 4 1 3
```

Problem B. Rectangle

Input file: `standard input`
Output file: `standard output`

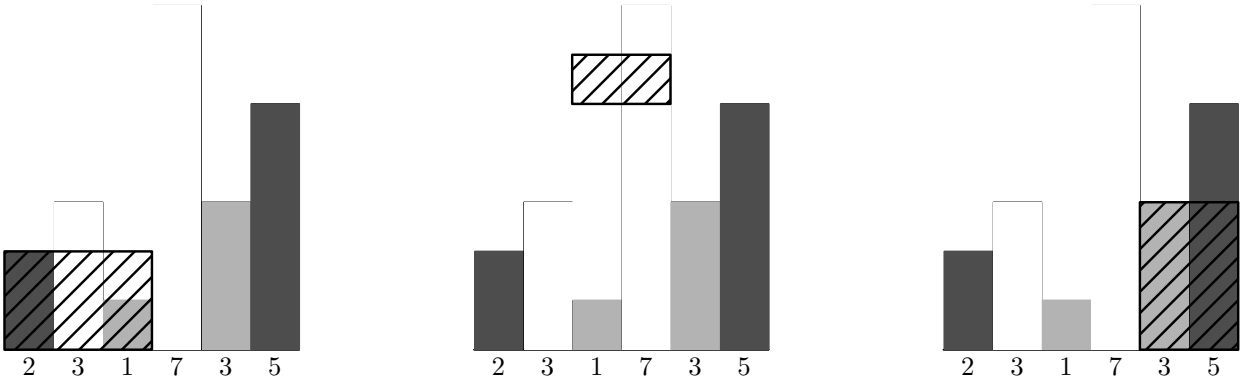
A *histogram* is a polygon composed of a sequence of rectangular bars aligned at a common base line. All bars have equal widths but may have different integer heights. Additionally, each bar is fully colored with a single color. For example, the figure below shows the histogram that consists of bars with the heights 2, 3, 1, 7, 1, 3, 5, measured in units where 1 is the width of each bar. In this case, every bar is colored with one out of three colors:



There are a lot of rectangles that can be placed inside this histogram (infinitely many, in fact). We will say that a rectangle is *nice* if and only if:

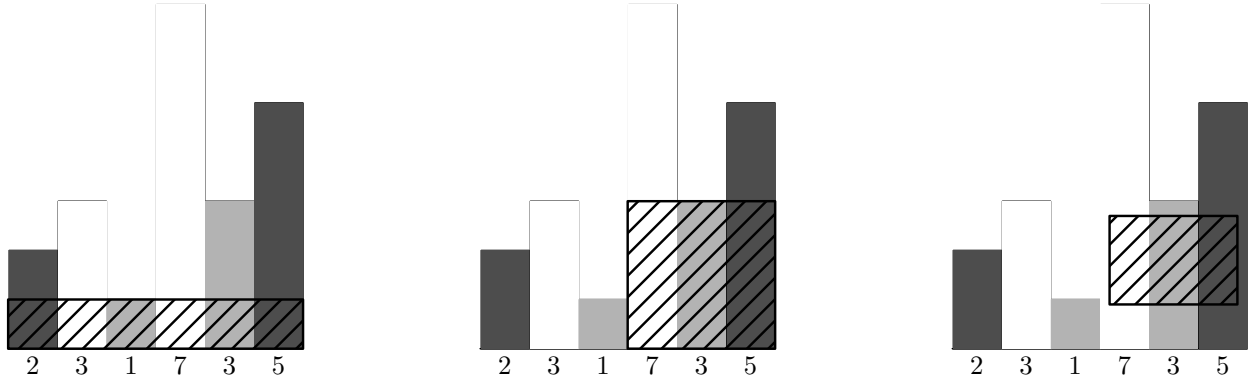
- It lies completely inside the histogram.
- It covers one or several regions of positive area for every color in the histogram (barely touching an area is not enough; the total area covered for each color must be strictly positive).

Here are some examples of rectangles that are not nice:



The rectangles in the two leftmost figures are not nice because they don't lie completely inside the histogram. The rectangle in the rightmost figure is not nice because the total white area covered is not strictly positive, and a nice rectangle must cover a positive area of every color.

On the other hand, here are some examples of nice rectangles:



The area of the nice rectangle in the leftmost figure is $6 \times 1 = 6$ square units, the area of the nice rectangle in the middle figure is $3 \times 3 = 9$ square units and the area of the nice rectangle in the rightmost figure is $1.8 \times 2.6 = 4.68$ square units.

Clearly, the area of the rectangle in the middle figure is largest. In fact, it turns out it doesn't exist a nice rectangle for this histogram with a larger area!

In this problem, you are given a histogram and your task is to compute the maximum area of all the possible nice rectangles. By the way, it is not hard to prove that this area will always be an integer.

Input

The input contains several test cases (at most 50).

Each test case is described by several lines. The first line contains two integer $\mathcal{N}$ and $\mathcal{C}$, the number of bars and the total number of colors in the histogram, respectively ($1 \leq \mathcal{N} \leq 10^5$ and $1 \leq \mathcal{C} \leq min(30, \mathcal{N})$).

The following line contains $\mathcal{N}$ integers h_i, the height of the ith bar from left to right ($1 \leq h_i \leq 10^9$). The following line contains $\mathcal{N}$ integers c_i, the color of the ith bar from left to right. Each color is represented as an integer between 0 and $\mathcal{C} - 1$, inclusive. You can assume that for each histogram there will be at least one bar of every color.

The last line of the input contains two zeros and should not be processed.

Output

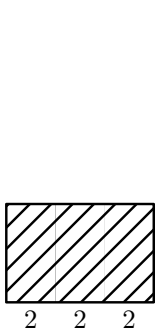
For each test case, output a single integer on a single line — the area of the largest nice rectangle for this histogram.

Sample input and output

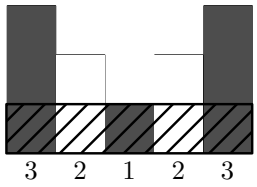
standard input	standard output
6 3 2 3 1 7 3 5 2 0 1 0 1 2 3 1 2 2 2 0 0 0 5 2 3 2 1 2 3 1 0 1 0 1 6 4 1 2 3 4 5 6 0 1 2 3 1 0 7 2 1 2 3 4 3 2 1 0 1 0 1 0 1 0 10 2 2 1 2 1 1 2 1 2 2 1 1 0 0 0 1 0 0 1 1 0 3 2 1000000000 999999997 999999999 0 1 1 0 0	9 6 5 12 10 10 2999999991

Explanation of the sample cases

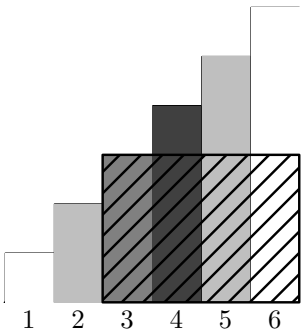
Below are some of the sample test cases with their respective largest nice rectangles:



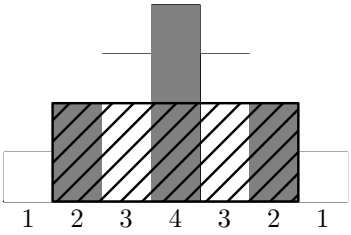
Example 2



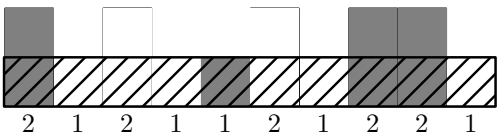
Example 3



Example 4



Example 5



Example 6

Problem F. Ancestors

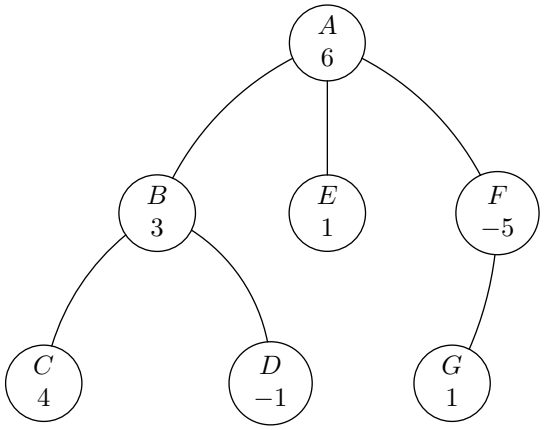
Input file: `standard input`
Output file: `standard output`

You will be given a connected undirected tree. Each node will have an associated integer that we will call its *value*. We also define the *value of a subset of nodes* as the sum of values of the nodes in the subset.

Consider the subsets of nodes of this tree whose size is between 1 and $\mathcal{K}$, inclusive, and satisfy the property that within each subset no node is an ancestor[1] of another. We will call these subsets the *$\mathcal{K}$-anti-ancestral subsets* of the tree.

Your task is, given the tree, to find the $\mathcal{K}$-anti-ancestral subset of maximum value.

As an example, consider the tree below. Each node is named with a capital letter and the value of each node is the integer below the letter:



Suppose $\mathcal{K} = 3$. Then the subset of nodes $\{A\}$, $\{B, E\}$, $\{C, E\}$, $\{D, G\}$ and $\{B, E, F\}$ are some of the $\mathcal{K}$-anti-ancestral subsets of the tree, because they all have between 1 and $\mathcal{K} = 3$ nodes and within each subset there doesn't exist a pair of nodes such that one is an ancestor of the other.

On the other hand, $\{A, B\}$ is not a $\mathcal{K}$-anti-ancestral subset because A is an ancestor of B, $\{A, G\}$ isn't one because A is an ancestor of G and $\{C, B, D\}$ isn't one either because B is an ancestor of both C and D. The subset $\{C, D, E, F\}$, even though it doesn't contain a node that is an ancestor of another one, is not a $\mathcal{K}$-anti-ancestral subset because it has $4 > \mathcal{K} = 3$ nodes. Similarly, the empty set $\emptyset$ is not one either because it contains 0 elements.

The value of the $\mathcal{K}$-anti-ancestral subset $\{A\}$ is 6; the value of $\{B, E\}$ is $3 + 1 = 4$; the value of $\{C, E\}$ is $4 + 1 = 5$; the value of $\{D, G\}$ is $-1 + 1 = 0$ and the value of $\{B, E, F\}$ is $3 + 1 + (-5) = -2$. Since the tree is small, it's easy to convince yourself by inspection that the maximum possible value of any $\mathcal{K}$-anti-ancestral subset is 6. However, notice that there might be several ways to achieve the maximum value: $\{A\}$ and $\{C, E, G\}$ both have value 6. You are only asked to find the value of the maximum $\mathcal{K}$-anti-ancestral subset so it doesn't really matter how many of them exist.

Input

The input file contains several test cases (at most 50).

Each test case is described on three lines:

- The first line contains two integers N and $\mathcal{K}$, the number of nodes in the tree and the parameter $\mathcal{K}$ as explained above. Assume $2 \leq N \leq 10^5$ and $1 \leq \mathcal{K} \leq 100$ (notice that these constraints guarantee that there will always exist at least one $\mathcal{K}$-anti-ancestral subset, e.g. by taking a single node). The

[1] Formally, we say that node u is an ancestor of node v if u lies on the path from v to the root of the tree.

nodes will be numbered 0 through $N-1$. The root of the tree will always be node 0. You can assume that the tree will always be connected, i.e., there will always exist a path connecting any pair of nodes.

- The second line contains N integers separated by spaces. These integers represent the values of nodes 0 through $N-1$ (the first integer is the value of node 0, the second integer is the value of node 1 and so on — the last integer is the value of node $N-1$). The value of any node will always be between -100 and 100, inclusive.
- The third line contains the description of the tree. It contains $N-1$ integers separated by spaces that represent the parent nodes of nodes 1 through $N-1$ (the first integer is the parent of node 1, the second integer is the parent of node 2 and so on — the last integer is the parent of node $N-1$). Notice that since 0 is always the root of the tree it doesn't have a parent and that's why this line contains only $N-1$ numbers instead of N.

The last line contains two zeros and should not be processed.

See the sample input for clarification about this format. The first test case is the tree given in the figure above.

Output

For each test case, output one integer on a single line — the maximum possible value of any $\mathcal{K}$-anti-ancestral subset.

Sample input and output

standard input	standard output
7 3 6 3 4 -1 1 -5 1 0 1 1 0 0 5 2 1 -1 1 0 3 3 1 2 3 0 0 8 8 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 4 4 -27 -45 -67 -98 2 0 2 0 0	6 1 5 8 -27

Problem I. Stones

Input file: `standard input`
Output file: `standard output`

Alicia and Roberto like to play games. Today, they are playing a game where there's a pile of stones on the table. The two players alternate turns removing some stones from the pile. On the first turn, a player can remove any number of stones but he or she must leave at least one stone on the table. On following turns, a player can remove at most twice the number of stones that the other player removed on the previous turn. Each player must always remove at least one stone.

The player that takes the last stone wins.

For example, let's imagine there's a pile of 8 stones on the table, and let's imagine Alicia starts.

- On the first turn, Alicia must remove a number of stones between 1 and 7 (according to the rules above, on the first turn she can remove any number of stones as long as there remains at least 1). Let's say Alicia takes 2 stones, leaving 6 on the table.
- On the second turn, Roberto must remove at least 1 stone and at most 4 stones (because 4 is twice the number of stones that Alicia removed in the previous turn). Let's say he takes 3, leaving 3 stones on the table.
- On the third turn, Alicia must remove at least 1 and at most 6 stones (6 is twice the number of stones Roberto took in the previous turn). Since only 3 stones remain on the table, she simply takes the 3 stones and wins the game!

However, Roberto could have played better. If he had taken only 1 stone on the second turn, he would have left 5 on the table with Alicia having to take between 1 and 2 stones. If Alicia took 2, Roberto could win immediately by taking the 3 stones that would remain. So Alicia's only choice is to take 1 stone, leaving 4 stones and Roberto having to take between 1 and 2 stones. Roberto would then take 1 stone, leaving 3 stones and Alicia having to take between 1 and 2 stones. Roberto could then win after Alicia's move, regardless of whether she takes 1 or 2 stones. In fact, it turns out that Roberto always has a winning strategy if there are 8 stones and Alicia plays first, even if Alicia plays in the best possible way!

Your task is to determine who wins the game if both players play optimally, assuming Alicia always plays first.

Input

The input contains several test cases.

Each test case contains a single integer n ($2 \leq n \leq 1000$), the number of stones that are initially on the table.

The last line of the input contains a single 0 and should not be processed.

Output

For each test case, output `Alicia` or `Roberto` on a single line, depending on who wins if both players play optimally and Alicia plays on the first turn.

Sample input and output

standard input	standard output
8 2 4 986 987 0	Roberto Roberto Alicia Alicia Roberto

Problem A. Asteroids

Input file: `asteroids.in`
Output file: `asteroids.out`

Association of Collision Management (ACM) is planning to perform the controlled collision of two asteroids. The asteroids will be slowly brought together and collided at negligible speed. ACM expects asteroids to get attached to each other and form a stable object.

Each asteroid has the form of a convex polyhedron. To increase the chances of success of the experiment ACM wants to bring asteroids together in such manner that their centers of mass are as close as possible. To achieve this, ACM operators can rotate the asteroids and move them independently before bringing them together.

Help ACM to find out what minimal distance between centers of mass can be achieved.

For the purpose of calculating center of mass both asteroids are considered to have constant density.

Input

Input file contains two descriptions of convex polyhedra.

The first line of each description contains integer number n — the number of vertices of the polyhedron ($4 \le n \le 60$). The following n lines contain three integer numbers x_i, y_i, z_i each — the coordinates of the polyhedron vertices ($-10^4 \le x_i, y_i, z_i \le 10^4$). It is guaranteed that the given points are vertices of a convex polyhedron, in particular no point belongs to the convex hull of other points. Each polyhedron is non-degenerate.

The two given polyhedra have no common points.

Output

Output one floating point number — the minimal distance between centers of mass of the asteroids that can be achieved. Your answer must be accurate up to 10^{-5}.

Sample input and output

asteroids.in	asteroids.out
8 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1 5 0 0 5 1 0 6 -1 0 6 0 1 6 0 -1 6	0.75

Problem C. Central Element

Input file: `standard input`
Output file: `standard output`

This is an interactive problem.

There is a permutation P of numbers 1 through n, not known to you, $P = \langle P_1, P_2, ..., P_n \rangle$. You can ask the following type of questions: Given three distinct positions i, j and k, which of P_i, P_j and P_k is central? Element is central if it is neither minimal nor maximal.

For example, if the permutation is $\langle 2, 1, 4, 3 \rangle$, and you ask about positions 1, 2, and 3, you receive 2, because 2 is the central element of the set $\{P_1, P_2, P_3\} = \{2, 1, 4\}$. Note that you don't get the information at which position among 1, 2, and 3 it is located.

Your task is to find the permutation P. Actually, for each permutation P there is a set $S(P)$ of permutations that cannot be distinguished from P using the allowed questions. You must find any permutation from this set.

Interaction protocol

First, your program must read from the standard input one line with the integer n, the size of the permutation.

The program must write to the standard output one line with three positions that you ask a question about and wait for a line in the standard input with a response, then write next question and read next response, and so on until you know the permutation P up to $S(P)$.

Once you know the answer, output one line with the word "`OK`" and the permutation P.

Input

The first line of the standard input contains n, the size of the permutation ($3 \le n \le 200$).

Each of the next lines of the standard input contains response to your question — the number that is central among the numbers at the asked positions.

Output

When you're asking questions, each line of the standard output should contain three different integers from the range of 1 to n, space-separated. You can ask at most 2 000 questions.

When you're stating the answer, the line of the standard output should contain the word "`OK`", and the numbers $P_1, P_2, \ldots, P_n$, all space-separated. After printing this line your program must exit.

You must flush standard output after printing each line.

Sample input and output

standard input	standard output
4 2 3 2 3	1 2 3 2 3 4 1 2 4 1 3 4 OK 2 1 4 3

Problem D. Database

Input file: `database.in`
Output file: `database.out`

Peter studies the theory of relational databases. Table in the relational database consists of values that are arranged in rows and columns.

There are different *normal forms* that database may adhere to. Normal forms are designed to minimize the redundancy of data in the database. For example, a database table for a library might have a row for each book and columns for book name, book author, and author's email.

If the same author wrote several books, then this representation is clearly redundant. To formally define this kind of redundancy Peter has introduced his own normal form. A table is in Peter's Normal Form (PNF) if and only if there is no pair of rows and a pair of columns such that the values in the corresponding columns are the same for both rows.

How to compete in ACM ICPC	Peter	peter@neerc.ifmo.ru
How to win ACM ICPC	Michael	michael@neerc.ifmo.ru
Notes from ACM ICPC champion	Michael	michael@neerc.ifmo.ru

The above table is clearly not in PNF, since values for 2rd and 3rd columns repeat in 2nd and 3rd rows. However, if we introduce unique author identifier and split this table into two tables — one containing book name and author id, and the other containing book id, author name, and author email, then both resulting tables will be in PNF.

How to compete in ACM ICPC	1
How to win ACM ICPC	2
Notes from ACM ICPC champion	2

1	Peter	peter@neerc.ifmo.ru
2	Michael	michael@neerc.ifmo.ru

Given a table your task is to figure out whether it is in PNF or not.

Input

The first line of the input file contains two integer numbers n and m ($1 \le n \le 10\,000$, $1 \le m \le 10$), the number of rows and columns in the table. The following n lines contain table rows. Each row has m column values separated by commas. Column values consist of ASCII characters from space (ASCII code 32) to tilde (ASCII code 126) with the exception of comma (ASCII code 44). Values are not empty and have no leading and trailing spaces. Each row has at most 80 characters (including separating commas).

Output

If the table is in PNF write to the output file a single word "`YES`" (without quotes). If the table is not in PNF, then write three lines. On the first line write a single word "`NO`" (without quotes). On the second line write two integer row numbers r_1 and r_2 ($1 \le r_1, r_2 \le n$, $r_1 \ne r_2$), on the third line write two integer column numbers c_1 and c_2 ($1 \le c_1, c_2 \le m$, $c_1 \ne c_2$), so that values in columns c_1 and c_2 are the same in rows r_1 and r_2.

Sample input and output

database.in	database.out
3 3 How to compete in ACM ICPC,Peter,peter@neerc.ifmo.ru How to win ACM ICPC,Michael,michael@neerc.ifmo.ru Notes from ACM ICPC champion,Michael,michael@neerc.ifmo.ru	NO 2 3 2 3
2 3 1,Peter,peter@neerc.ifmo.ru 2,Michael,michael@neerc.ifmo.ru	YES

Problem E. Exclusive Access 2

Input file: `exclusive.in`
Output file: `exclusive.out`

Having studied mutual exclusion protocols in the previous year's competition you are now facing a more challenging problem. You have a big enterprise system with a number concurrently running processes. The system has several resources — databases, message queues, etc. Each concurrent process works with two resources at a time. For example, one process might copy a job from a particular database into the message queue, the other process might take a job from the message queue, perform the job, and then put the result into some other message queue, etc.

All resources are protected from concurrent access by mutual exclusion protocols also known as *locks*. For example, to access a particular database process acquires the lock for this database, then performs its work, then releases the lock. No two processes can hold the same lock at the same time (that is the property of mutual exclusion). Thus, the process that tries to acquire a lock *waits* if that lock is taken by some other process.

The main loop of the process that works with resources P and Q looks like this:

```
loop forever
  DoSomeNonCriticalWork()
  P.lock()
  Q.lock()
  WorkWithResourcesPandQ()
  Q.unlock()
  P.unlock()
end loop
```

The order in which locks for resources P and Q are taken is important. Consider a case where process c had acquired lock P with `P.lock()` and is waiting for lock Q in `Q.lock()`. It means that lock Q is taken by some other process d. If the process d is working (not waiting), then we say that there is a *wait chain* of length 1. If d had acquired lock Q and is waiting for another lock R, which is acquired by a working process e, then we say that there is a *wait chain* of length 2, etc. If any process in this wait chain waits for lock P that is already taken by process c, then we say that the wait chain has infinite length and the system *deadlocks*.

For this problem, we are interested only in alternating wait chains where processes hold their first locks and wait for the second ones. Formally:

> *Alternating wait chain* of length n ($n \geq 0$) is an alternating sequence of resources R_i ($0 \leq i \leq n+1$) and distinct processes c_i ($0 \leq i \leq n$): $R_0\ c_0\ R_1\ c_1\ \ldots\ R_n\ c_n\ R_{n+1}$, where process c_i acquires locks for resources R_i and R_{i+1} in this order. Alternating wait chain is a deadlock when $R_0 = R_{n+1}$.

You are given a set of resources each process works with. Your task is to decide the order in which each process has to acquire its resource locks, so that the system never deadlocks and the maximum length of any possible alternating wait chain is minimized.

Input

The first line of the input file contains a single integer n ($1 \leq n \leq 100$) — the number of processes.

The following n lines describe resources that each process needs. Each resource is designated with an uppercase English letter from `L` to `Z`, so there are at most 15 resources. Each line describing process contains two different resources separated by a space.

Output

On first line of the output file write a singe integer number m — the minimally possible length of the maximal alternating wait chain.

Then write n lines — one line per process. On each line write two resources in the order they should be taken by the corresponding process to ensure this minimal length of the maximal alternating wait chain. Separate resources on a line by a space. If there are multiple satisfying orderings, then write any of them. The order of the processes in the output should correspond to their order in the input.

Sample input and output

exclusive.in	exclusive.out
2 P Q R S	0 P Q R S
6 P Q Q R R S S T T U U P	0 P Q R Q R S T S T U P U
4 P Q P Q P Q P Q	0 P Q P Q P Q P Q
3 P Q Q R R P	1 P Q Q R P R
6 P Q Q S S R R P P S R Q	2 P Q Q S R S P R P S R Q

Problem F. Funny Language

Input file: `funny.in`
Output file: `funny.out`

There is a well know game with words. Given a word you have to write as many other words as possible using the letters from the given word. If the letter repeats multiple times in the original word, you can use it up to as many times in the new words. The order of letters in the original word does not matter. For example, given the word CONTEST you can write NOTE, NET, ON, TEST, SET, etc.

Now you are in charge of writing a new dictionary. Your task is to sneak n new words into it. You know in advance m words W_i ($1 \le i \le m$) that you will have to play a game with and you need to figure out which new n words to add to the dictionary to maximize the total number of words you can write out of these m words.

More formally, find such a set of nonempty words S where $|S| = n$, $W_i \notin S$ for any i, and $\sum_{1 \le i \le m} |S_i|$ is maximal, where $S_i \subset S$ is the set of words that can be written using letters from W_i.

Input

The first line of the input file contains two integer numbers n ($1 \le n \le 100$) — the number of new words you can add to the dictionary and m ($1 \le m \le 1\,000$) — the number of words you will play the game with. The following m lines contain original words. Each word consists of at most 100 uppercase letters from A to Z.

Output

Write to the output file n lines with a new word on a line.

Sample input and output

funny.in	funny.out
3 5 A ACM ICPC CONTEST NEERC	C CN E

Problem G. Garbling Game

Input file: `garbling.in`
Output file: `garbling.out`

Pavel had invented a new game with a matrix of integer numbers. He takes $r \times c$ matrix with r rows and c columns that is filled with numbers from 1 to rc left to right and top to bottom (1 is written in the upper-left corner, rc is written in the lower-right corner). Then he starts to rearrange the numbers is the matrix by following the rules that are explained below and writes down a sequence of numbers on a separate piece of paper. He calls it *garbling* of the matrix.

The rules of rearrangement are defined by *garbling map* that is $(r-1) \times (c-1)$ matrix of letters L, R, and N. Initial 4×5 matrix and the sample 3×4 garbling map for it are shown below.

(1)	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

L	R	L	R
N	L	L	R
L	N	N	L

Pavel garbles the matrix in a series of turns. On his first turn Pavel takes the number in the first row and the first column (it is put in parenthesis on the above picture for clarity) and writes it down.

Having written down the number he performs one *garbling turn*:

Pavel looks at the letter in the garbling map that corresponds to the position of the number he had just written down (one the first turn it is the letter in the upper-left corner). Depending on the letter in the garbling map the 2×2 block of the matrix whose upper-left corner contains the number he had just written (highlighted in the above picture) is rearranged in the following way:

- R — the block is rotated *clockwise*.
- L — the block is rotated *counterclockwise*.
- N — Pavel does not change the matrix on this turn.

On the second turn Pavel takes the number in the first row and second column, writes it down, and performs the garbling turn, and so on. In $c-1$ turns he finishes the first row and moves to the second row and so on he proceeds left to right and top to bottom. In $(r-1)(c-1)$ turns he had written down $(r-1)(c-1)$ numbers and garbled the whole matrix, so he starts again in the upper-left corner continuing garbling the matrix from left to right and top to bottom.

The matrices below show the effect of the first four turns with the sample garbling map.

2	(7)	3	4	5
1	6	8	9	10
11	12	13	14	15
16	17	18	19	20

2	6	(7)	4	5
1	8	3	9	10
11	12	13	14	15
16	17	18	19	20

2	6	4	(9)	5
1	8	7	3	10
11	12	13	14	15
16	17	18	19	20

2	6	4	3	9
(1)	8	7	10	5
11	12	13	14	15
16	17	18	19	20

The following sequence of numbers is written down in the first 4 turns: `1 7 7 9`. On 5th turn the number from the second row and the first column is written, but the matrix remains unchanged, since the second row and the first column of the garbling map contains N. In six turns Pavel gets `1 7 7 9 1 8`.

Given the garbling map and the number of moves Pavel makes in this game, find out how many times each number gets written down by Pavel. You need to provide the answer modulo 10^5.

Input

The first line of the input file contains three integer numbers — r, c, and n, where r, c ($2 \le r$, $c \le 300$) are the dimensions of the initial matrix, n ($0 \le n < 10^{100}$) is the number of turns Pavel makes.

The following $r-1$ lines contain garbling map with $c-1$ characters R, L, or N on a line.

Output

Write to the output file rc lines with one integer number per line. On i-th line write the number of times number i gets written down by Pavel modulo 10^5 while he makes his n turns.

Sample input and output

garbling.in	garbling.out
4 5 6 LRLR NLLR LNNL	2 0 0 0 0 0 2 1 1 0 0 0 0 0 0 0 0 0 0 0 0
4 5 666666 LRLR NLLR LNNL	37038 37038 0 0 30864 37036 11112 30864 30864 30864 30864 30864 11110 30865 18519 30864 30864 0 18518 18518

Problem I. Inspection

Input file: `inspection.in`
Output file: `inspection.out`

You are in charge of a team that inspects a new ski resort. A ski resort is situated on several mountains and consists of a number of slopes. Slopes are connected with each other, forking and joining. A map of the ski resort is represented as an acyclic directed graph. Nodes of the graph represent different points in ski resort and edges of the graph represent slopes between the points, with the direction of edges going downwards.

Your team has to inspect each slope of the ski resort. Ski lifts on this resort are not open yet, but you have a helicopter. In one flight the helicopter can drop one person into any point of the resort. From the drop off point the person can ski down the slopes, inspecting each slope as they ski. It is fine to inspect the same slope multiple times, but you have to minimize the usage of the helicopter. So, you have to figure out how to inspect all the slopes with the fewest number of helicopter flights.

Input

The first line of the input file contains a single integer number n ($2 \le n \le 100$) — the number of points in the ski resort. The following n lines of the input file describe each point of the ski resort numbered from 1 to n. Each line starts with a single integer number m_i ($0 \le m_i < n$ for i from 1 to n) and is followed by m_i integer numbers a_{ij} separated by spaces. All a_{ij} are distinct for each i and each a_{ij} ($1 \le a_{ij} \le n$, $a_{ij} \ne i$) represents a slope going downwards from point i to point a_{ij}. Each point in the resort has at least one slope connected to it.

Output

On the first line of the output file write a single integer number k — the minimal number of helicopter flights that are needed to inspect all slopes. Then write k lines that describe inspection routes for each helicopter flight. Each route shall start with single integer number from 1 to n — the number of the drop off point for the helicopter flight, followed by the numbers of points that will be visited during inspection in the corresponding order as the slopes are inspected going downwards. Numbers on a line shall be separated by spaces. You can write routes in any order.

Sample input and output

inspection.in	inspection.out
8 1 3 1 7 2 4 5 1 8 1 8 0 2 6 5 0	4 1 3 4 8 1 3 5 8 2 7 6 7 5

Problem J. Java Certification

Input file: `javacert.in`
Output file: `javacert.out`

You have just completed Java Certification exam that contained n questions. You have a score card that explains your performance. The example of the scorecard is given below.

You have correctly answered 78 questions out of 87.

Basic Concepts	100%
Declarations	100%
Expressions	83%
Classes and Interfaces	92%
Multithreading	75%
Collections	93%

From this scorecard you can infer that the questions are broken down into m categories (in the above example $m = 6$). Each category contains n_i questions ($1 \le n_i \le n$), so that $\sum_{1 \le i \le m} n_i = n$. You know that you have correctly answered k questions out of n (in the above example $k = 78$ and $n = 87$), so you can easily find the number of incorrect answers $w = n - k$ (in the above example $w = 9$).

You do remember several questions that you were unsure about and you can guess what category they belong to. To figure out if your answers on those questions were right or wrong, you really want to know how many incorrect answers you have given in each category.

Let w_i ($0 \le w_i \le n_i$) be the number of *incorrect* answers in i-th category, $\sum_{1 \le i \le m} w_i = w$. From the scorecard you know the *percentage of correct answers* in each category. That is, for each i from 1 to m you know the value of $100(n_i - w_i)/n_i$ rounded to the nearest integer. The value with a fractional part of 0.5 is rounded to the nearest even integer.

It may not be possible to uniquely find the valid values for w_i. However, you guess that the questions are broken down into categories in a mostly uniform manner. You have to find the valid values of w_i and n_i, so that to minimize the difference between the maximum value of n_i and the minimum value of n_i. If there are still multiple possible values for w_i and n_i, then find any of them.

Input

The first line of the input file contains three integer numbers — k, n, and m, where k ($0 \le k \le n$) is the number of correctly answered questions, n ($1 \le n \le 100$) is the total number of questions, m ($1 \le m \le 10$) is the number of question categories. The following m lines of the input file contain one integer number from 0 to 100 (inclusive) on a line — percentages of the number of the correct answers in each category. The input file always corresponds to some valid set of w_i and n_i.

Output

Write to the output file m lines with two integer numbers w_i and n_i on a line, separated by a space — the number of incorrect answers and the total number of questions in each category, satisfying constraints as given in the problem statement.

Sample input and output

javacert.in	javacert.out
78 87 6 100 100 83 92 75 93	0 13 0 13 3 18 1 13 4 16 1 14

Problem K. K-equivalence

Input file: `kequiv.in`
Output file: `kequiv.out`

Consider a set K of positive integers.

Let p and q be two non-zero decimal digits. Call them K-equivalent if the following condition applies:

> For every $n \in K$, if you replace one digit p with q or one digit q with p in the decimal notation of n then the resulting number will be an element of K.

For example, when K is the set of integers divisible by 3, the digits 1, 4, and 7 are K-equivalent. Indeed, replacing a 1 with a 4 in the decimal notation of a number never changes its divisibility by 3.

It can be seen that K-equivalence is an equivalence relation (it is reflexive, symmetric and transitive).

You are given a finite set K in form of a union of disjoint finite intervals of positive integers.

Your task is to find the equivalence classes of digits 1 to 9.

Input

The first line contains n, the number of intervals composing the set K ($1 \le n \le 10\,000$).

Each of the next n lines contains two positive integers a_i and b_i that describe the interval $[a_i, b_i]$ (i. e. the set of positive integers between a_i and b_i, inclusive), where $1 \le a_i \le b_i \le 10^{18}$. Also, for $i \in [2..n]$: $a_i \ge b_{i-1} + 2$.

Output

Represent each equivalence class as a concatenation of its elements, in ascending order.

Output all the equivalence classes of digits 1 to 9, one at a line, sorted lexicographically.

Sample input and output

kequiv.in	kequiv.out
1 1 566	1234 5 6 789
1 30 75	12 345 6 7 89

Problem A
Advanced edit distance

Source file name: `aed.c`, `aed.cpp` *or* `aed.java`

The edit distance of two strings S and T is the minimum number of edit operations that need to be done to transform S into T. The valid edit operations are:

- Insert a single character at any position.
- Modify an existing character.
- Remove an existing character.

For example, the edit distance of "pantera" and "aorta" is 5, because the following chain of edits is valid (and there is no shorter chain):

"pantera" $\rightarrow$ "antera" $\rightarrow$ "aotera" $\rightarrow$ "aoera" $\rightarrow$ "aora" $\rightarrow$ "aorta".

We define the advanced edit distance in a similar way, but adding the swap of two adjacent characters as an extra valid operation. With this setting, the advanced edit distance of "pantera" and "aorta" is 4:

"pantera" $\rightarrow$ "antera" $\rightarrow$ "antra" $\rightarrow$ "aotra" $\rightarrow$ "aorta".

You need to write a program that calculates the advanced edit distance of two given words.

Input

The input contains several test cases. Each test case is described in a single line that contains two non-empty words, each of them of at most 1000 lowercase letters, separated by a single space. The last line of the input contains two asterisks separated by a single space and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with an integer representing the advanced edit distance of the two input words.

The output must be written to standard output.

Sample input	Output for sample input

`pantera aorta`
`zero zero`
`* *`
	`4`
	`0`

Problem B
Back to the polygon

Source file name: `backtopol.c, backtopol.cpp` *or* `backtopol.java`

A *simple* polygon is a polygon that does not overlap with itself. A *diagonal* of a simple polygon is a segment within the polygon that connects two non-consecutive vertices. A *triangulation* of a simple polygon of N edges is the drawing of exactly $N-3$ diagonals that do not touch each other anywhere, with the possible exception of their endpoints. A triangulation divides the polygon in exactly $N-2$ triangles that do not overlap and only touch each other along their edges.

In this problem, you are given the triangulation of a simple polygon, which means, the set of triangles in which a polygon was divided. From them, you need to reconstruct the original polygon.

Input

The input contains several test cases, each one described in several lines. The first line of each test case contains an integer N ($3 \leq N \leq 500$), the number of edges of the original polygon. Each of the next $N-2$ lines describes one triangle in the triangulation of the polygon. Each triangle is given by six integers X_1, Y_1, X_2, Y_2, X_3 and Y_3 separated by single spaces, where X_i and Y_i are the coordinates in the XY plane of the i-th vertex of the triangle ($-1000 \leq X_i, Y_i \leq 1000$). The triangles and their vertices are not given in any specific order. The last line of the input contains a single -1 and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with $2N$ integers separated by single spaces. These integers must represent the coordinates in the XY plane of the vertices of the original polygon, in clockwise order. To make the output unique, the first vertex to be listed is the one with the smallest X coordinate, and if there are many of those, the one with the smallest Y coordinate among them.

The output must be written to standard output.

Sample input	**Output for sample input**
5 0 0 10 9 10 0 10 9 0 9 0 0 10 9 0 9 5 13 3 0 1 1 1 1 0 10 -1 -2 2 -2 2 -1 -1 -2 0 -1 -2 3 -2 3 2 3 1 2 0 -1 2 1 1 2 -1 -2 2 -1 0 -1 2 1 0 -1 2 0 2 3 2 2 1 2 0 -1 -2 3 1 2 -1	0 0 0 9 5 13 10 9 10 0 0 1 1 1 1 0 -2 3 2 3 2 2 1 2 2 1 2 0 0 -1 2 -1 2 -2 -1 -2

Problem C
Charly & Nito

Source file name: `charly.c`, `charly.cpp` *or* `charly.java`

Charly and Nito are friends and they like to be together at a nice bar in Palermo Hollywood. About at 3 a.m. they start to feel sleepy and want to go home. They want to get home quickly so each of them uses a path that minimizes the distance to his home. However, Charly and Nito also like to walk together while they talk about the "good old times", so they want to walk together as much as possible.

Charly and Nito live in a city that can be modeled as a set of streets and junctions. Each street connects a pair of distinct junctions and can be walked in both directions. No two streets connect the same pair of junctions. Charly and Nito do not live together, and they do not live at the bar. There is at least one path from the bar to Charly's home; the same occurs with Nito's home.

Given information about the streets and junctions in the city, the locations of the bar, Charly's home and Nito's home, you must tell Charly and Nito the maximum distance that they can walk together without forcing them to walk more than the minimum distance from the bar to their respective homes. Charly and Nito also want to know how much each of them will walk alone.

Input

The input contains several test cases, each one described in several lines. The first line of each test case contains five integers J, B, C, N and S separated by single spaces. The value J is the number of junctions in the city ($3 \leq J \leq 5000$); each junction is identified by an integer number between 1 and J. The values B, C and N are the identifiers of the junctions where the bar, Charly's home and Nito's home are located, respectively ($1 \leq B, C, N \leq J$); these three junction identifiers are different. The value S is the number of streets in the city ($2 \leq S \leq 150000$). Each of the next S lines contains the description of a street. Each street is described using three integers E_1, E_2 and L separated by single spaces, where E_1 and E_2 identify two distinct junctions that are endpoints of the street ($1 \leq E_1, E_2 \leq J$), and L is the length of the street ($1 \leq L \leq 10^4$). You may assume that each street has a different pair of endpoints, and that there exist paths from junction B to junctions C and N. The last line of the input contains the number -1 five times separated by single spaces and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with three integers T, C and N separated by single spaces, where T is the maximum distance that Charly and Nito can walk together, C is the distance that Charly walks alone, and N is the distance that Nito walks alone.

The output must be written to standard output.

Sample input	Output for sample input
5 3 2 1 6 3 4 10 4 5 10 5 1 3 5 2 4 1 3 23 2 3 24 8 1 7 8 8 1 2 1 2 4 1 2 3 1 4 5 1 3 5 1 5 6 1 6 8 1 6 7 1 -1 -1 -1 -1 -1	20 4 3 4 1 1

Problem D
Doing the word wrap

Source file name: `doingww.c`, `doingww.cpp` *or* `doingww.java`

You are developing a visual component for a web browser. The component is known as *textarea*, and its main functionality is to show a given text using one or more lines. Every textarea has a *linewidth* W, which is the number of characters that can fit in a single line.

The text that needs to be shown is a sequence of words. The textarea must display the text using lines of W characters, without breaking any word, and placing a single space between each pair of consecutive words that are in the same line. Any number of trailing spaces may be left at the end of each line. So the behavior of the textarea is quite simple: it keeps adding words to a line until the next word does not fit; each time this occurs, a new line is started.

With the permanent growing in the amount of information that web pages must show, you have to make a smart textarea that uses as little space as possible, even when dealing with very long texts. Given a text to show and a number of lines L, you must set the linewidth W to the minimum possible value such that the text is shown using at most L lines.

Input

The input contains several test cases, each one described in exactly two lines. The first line of each test case contains two integers L and N separated by a single space, where L is the maximum number of lines the textarea can have ($1 \leq L \leq 10^8$), and N is the number of words the text to show is made of ($1 \leq N \leq 10^5$). The second line contains the text to show, formed by N non-empty words of at most 25 lowercase letters each, separated by an arbitrary number of spaces. The last line of the input contains the number -1 twice separated by a single space and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with an integer W representing the minimum linewidth such that the textarea has at most L lines.

The output must be written to standard output.

Sample input	Output for sample input
1 2 hello word 2 2 racing club -1 -1	10 6

Problem E
Edit distance

Source file name: `edit.c`, `edit.cpp` *or* `edit.java`

The edit distance of two strings S and T is the minimum number of edit operations that need to be done to transform S into T. The valid edit operations are:

- Insert a single character at any position.
- Modify an existing character.
- Remove an existing character.

For example, the edit distance of "pantera" and "aorta" is 5, because the following chain of edits is valid (and there is no shorter chain):

"pantera" $\rightarrow$ "antera" $\rightarrow$ "aotera" $\rightarrow$ "aoera" $\rightarrow$ "aora" $\rightarrow$ "aorta".

In this problem, given a value K and a word S, we need to construct a word T such that the edit distance of S and T is at most K. There are of course several possibilities for that, so we will ask that you choose the word T that comes first alphabetically.

A word always comes alphabetically after any proper prefix. Among two words that are not prefixes of each other, the one that comes first alphabetically is the one that has, in the first position at wich they differ from left to right, a letter closest to the beginning of the alphabet. Notice that the empty word (that has zero characters) is a valid word and is alphabetically before any other word.

Input

The input contains several test cases. Each test case is described in a single line that contains an integer K ($0 \leq K \leq 1000$) and a non-empty word S of at most 1000 lowercase letters, separated by a single space. The last line of the input contains the number -1 and an asterisk separated by a single space and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with a word T of lowercase letters such that the edit distance of S and T is at most K. If there are several words in that situation, output the one that comes first alphabetically.

The output must be written to standard output.

Sample input	Output for sample input
`4 abcadef`	`aaaaaadef`
`1000 zero`	
`0 zero`	`zero`
`-1 *`	

Problem G
G key

Source file name: `gkey.c`, `gkey.cpp` *or* `gkey.java`

Leandro and Fede are traveling by train and to spend some time they decided to start playing the guitar. They want to play together some songs, but Fede's memory is not working well because he caught a little cold. To work it out, Leandro wants to show to Fede some music scores of several punk rock songs. A punk rock song is a sequence of notes, and there are twelve possible notes, divided in two groups. The first group has seven natural notes called A, B, C, D, E, F and G, while the second group has five alterations called A#, C#, D#, F# and G#.

The way to draw a music score is as follows: you start with an empty one, and then you draw note by note from left to right in the same order they appear in the song. Each line is divided in four groups, each group having room for four notes.

As punk rock is quite simple the drawing is even easier. Natural notes have always the same shape called *eighth note*, and they differ only in the height at which they are drawn. Alterations are drawn like natural notes but with a preceeding "#" sign. To make it simpler, Leandro will avoid the drawing of the G key.

Can you help Leandro writing a program for drawing punk rock songs given the sequence of notes?

Input

The input contains several test cases. Each test case is described in a single line that contains the number of notes N ($1 \leq N \leq 100$), followed by the sequence of N notes. Each note is one of A, A#, B, C, C#, D, D#, E, F, F#, G and G#. Values in each line are separated by single spaces. The last line of the input contains a single -1 and should not be processed as a test case.

The input must be read from standard input.

Output

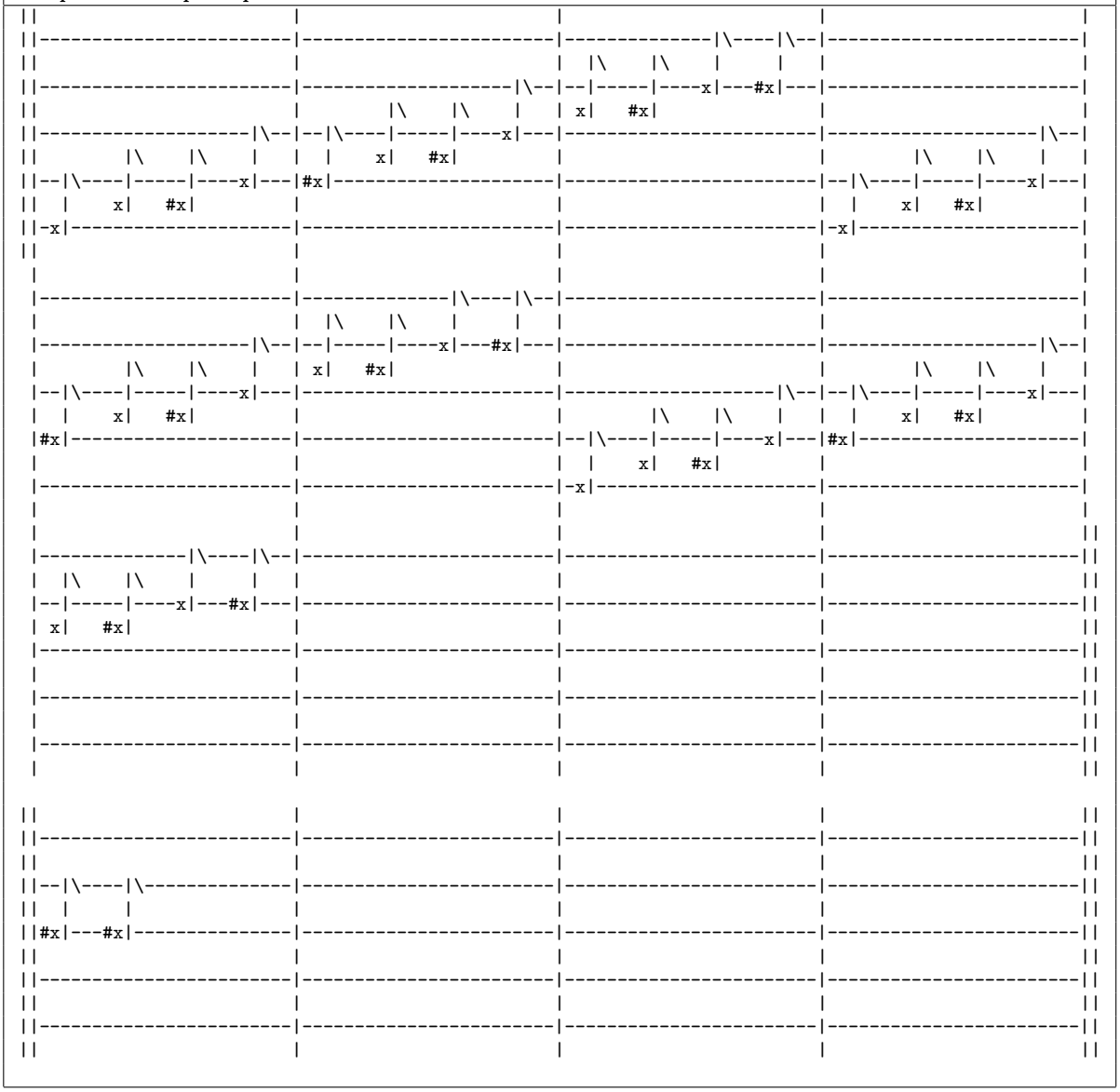
For each test case output the music score of the input song and print a blank line after each test case (even after the last one). You have to follow the sample input and output for drawing the music scores. Every score line has the same background formed by the characters "`|`" (pipe) and "`-`" (hyphen). They differ in the borders (first and last borders are doubled), and of course in the notes they have inside. Each eighth note is drawn consecutively as in the sample, and the different heights are those shown. Alterations are preceded by a character "`#`" (sharp sign). There must be no trailing spaces at the end of printed lines, neither empty score lines (without notes inside).

The output must be written to standard output.

Sample input

```
36 E F F# G G# A A# B C C# D D# E F F# G G# A A# B C C# D D# E F F# G G# A A# B C C# D D#
2 B# B#
-1
```

Output for sample input



Problem A
The Starflyer Agents

Source file name: `agents.c`, `agents.cpp` *or* `agents.java`

Famed investigator Paula Myo, working on behalf of the 2011 established Commonwealth government, is determined to stop the Starflyer from spying. The Starflyer is a "human-friendly" and powerful alien sentinel intelligence that was found by a space exploration frigate in the Dyson Alpha solar system in year 2285. It is not clear what the Starflyer's real intentions are towards the Commonwealth ... so, it is always better to be safe than sorry!!!

The Starflyer has the ability to control technological equipment; it typically infiltrates droids and uses them as agents. As a matter of fact, droids are carefully identified and tracked in the Commonwealth. Every droid has a history of software updates and each software update is tagged with a hash. A *hash* is a term built recursively from variable, constant, and function symbols as follows:

- any variable and any constant is a hash;
- if each $h_1, \ldots, h_k$ is a hash and f is a function symbol, then $f(h_1, \ldots, h_k)$ is a hash.

As a security measure, a well-kept secret from the general population, the Commonwealth enforces the following policy on droid software updates: for each droid, the tags of any software updates must be compatible. Two hashes h_1 and h_2 are *compatible* if there is a mapping θ from variables to hashes such that $h_1\theta = h_2\theta$, where $h_1\theta$ (resp., $h_2\theta$) denotes the simultaneous replacement of any occurrence of each variable x in h_1 (resp., h_2) with the hash $\theta(x)$. A sequence of hashes $h_1, \ldots, h_n$ is *compatible* if there is θ such that $h_1\theta, \ldots, h_n\theta$ are all equal.

For example, assume that X, Y, Z are variables, c, d are constants, and f, g are function symbols, and consider the hashes h_1, h_2, and h_3 as follows:

$$h_1 : f(X, g(c)) \qquad h_2 : f(f(Y), Z) \qquad h_3 : f(c, g(Y, d))$$

Observe that h_1 and h_2 are compatible because the mapping $\theta = \{X \mapsto f(Y), Z \mapsto g(c)\}$ satisfies $h_1\theta = h_2\theta$. However, any other pair from h_1, h_2, and h_3 is not compatible. Therefore, any sequence of hashes containing h_1, h_2, and h_3 is not compatible because there is no mapping θ such that $h_1\theta = h_2\theta = h_3\theta$.

Detective Myo has just been briefed on the aforementioned security policy. She strongly believes that the Starflyer infiltrates the droids via software updates without having any knowledge of the security policy. If her intuition is right, then this is the chance to detect and stop some Starflyer agents. You have been assigned to Myo's team: your task is to write an algorithm for determining if a sequence of hashes is compatible or not.

Can you help Detective Myo to uncover the Starflyer agents?

Input

The input consists of several test cases. The first line of each test case contains a string *name* and a natural number n separated by a blank ($2 \leq n \leq 20$, $1 \leq |name| \leq 16$). Then n lines follow, each containing a hash h_i ($1 \leq i \leq n$, $1 \leq |h_i| \leq 512$). You can suppose that:

- The *name* is an alphanumeric text (without blanks) that has a length less than or equal to 16 characters.
- Each one of the n hashes was built according to the above definition and has a length less than or equal to 512 characters.
- The variables, constants, and function symbols are formed exclusively from alphabetic characters. The first character of a variable symbol is an uppercase letter and the first character of a constant or function symbol is a lowercase letter.

The last test case is followed by a line with the text "`END 0`".

The input must be read from the file agents.in.

Output

For each test case, a line must be printed. If the sequence of hashes $h_1, \ldots, h_n$ is compatible, then print the line

```
analysis inconclusive on XXX
```

or if the sequence of hashes $h_1, \ldots, h_n$ is not compatible, then print the line

```
XXX is a Starflyer agent
```

where `XXX` corresponds to *name* in the test case.

The output must be written to standard output.

Sample Input	**Sample output**
r2d2 3 f(X,g(c)) f(f(Y),Z) f(c,g(Y,d)) c3po 2 f(X,g(c)) f(f(Y),Z) PC2 2 f(f(Y),Z) f(c,g(Y,d)) END 0	r2d2 is a Starflyer agent analysis inconclusive on c3po PC2 is a Starflyer agent

Problem B
Sewing Buttons with Grandma

Source file name: `buttons.c, buttons.cpp` *or* `buttons.java`

After so many years of studying math in the Academy of Colombian Mathematics (ACM) in the tropic, Eloi has finally decided to visit his grandmother for the winter in the north hemisphere. "Buttons and patches and the cold wind blowing, the days pass quickly when I am sewing" she says – Eloi now remembers how grandma quilts have love in every stitch. As a matter of fact, he has decided to learn the useful handicraft art of sewing buttons with grandma.

Eloi has realized that there is an interesting mathematical puzzle related to the task of sewing buttons to the front of a shirt. Given a collection of n_1 buttons of color c_1, n_2 buttons of color c_2, ..., n_k buttons of color c_k, and a shirt with m front buttonholes, in how many ways the buttonholes can be assigned m buttons from the $n_1+\cdots+n_k$ buttons?

Input

The input consists of several test cases. The first line of each test case contains two integers m and k separated by a blank ($1 \leq m \leq 50$, $1 \leq k \leq 50$). Then k lines follow each containing an integer n_i with $1 \leq n_i \leq 50$. The last test case is followed by a line containing two zeros.

The input must be read from the file buttons.in.

Output

The output consists of one line per test case containing the number of ways the m buttonholes can be assigned m buttons from the $n_1+\cdots+n_k$ buttons, assuming that the colors $c_1, \ldots, c_k$ are pairwise distinct.

The output must be written to standard output.

Sample Input	**Sample output**
1 3 3 1 1 3 2 4 2 3 2 1 1 0 0	3 7 0

Problem C
Document Compression

Source file name: `compression.c, compression.cpp` *or* `compression.java`

Alice needs to send text documents to Bob using the Internet. They want to use the communication channel as efficiently as possible, thus they decided to use a document codification scheme based on a set of *basis documents*. The scheme works as follows:

- Each document to be transmitted is represented using a set of terms. This ignores the frequency of each term and its position in the original document, i.e., a document is represented by the list of different terms that it contains.
- There is a set of basis documents that has been previously agreed upon by Alice and Bob. The basis documents are numbered and they are also represented using sets of terms. Both Alice and Bob have a copy of the set, so it does not need to be transmitted.
- Before transmitting a particular document, Alice finds the basis documents that need to be combined to obtain the original content of the document. The documents are combined by uniting their corresponding sets of terms.
- Alice transmits to Bob the list of indexes of the basis documents that represent the original document.
- Bob rebuilds the original document by combining the corresponding basis documents specified by Alice.

For example, suppose that the basis document set contains the following documents, each one specified by the list of different terms that it contains:

Document 1: {2, 5, 7, 10}
Document 2: {8, 9, 10}
Document 3: {1, 2, 3, 4}

Now, suppose that Alice wants to transmit the document {1, 2, 3, 4, 5, 7, 10}. This document can be obtained by combining the basis documents 1 and 3, so Alice transmits this list of two indices and Bob uses it to rebuild the original document.

Your work is, given a set of basis documents and a document to be transmitted, to find the minimum number of basis documents needed to represent it, if it can be represented; otherwise, you have to indicate that it is impossible to attain a representation.

Input

Each test case is described as follows:

- A line with two integer numbers M and N ($1 \leq M \leq 100$, $1 \leq N \leq 100$), separated by blanks. M corresponds to the number of basis documents and N corresponds to the number of documents to codify.
- Each one of the following $M+N$ lines contains the numbers $k, t_1, t_2, \ldots, t_k$, separated by blanks ($1 \leq k \leq 16$, $1 \leq t_i \leq 16$ for each $1 \leq i \leq k$). The first value indicates the number of different terms in the document and the following numbers correspond to the different terms in the document. The first M lines describe the basis documents and the next N lines describe the documents to codify.

The last test case is followed by a line containing two zeros.

The input must be read from the file compression.in.

Output

For each test case you must print a line containing the integers $r_1, r_2, \ldots, r_N$ (with a blank between each two numbers), where r_i is the minimum number of basis documents required to represent the i-th document of the input ($1 \leq i \leq N$). If it is impossible to represent the i-th document, then r_i must be the number 0.

The output must be written to standard output.

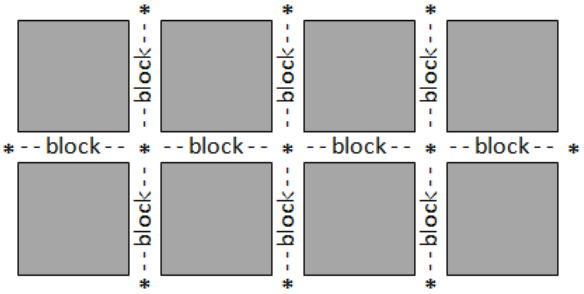
Sample Input	**Sample output**
3 1 4 2 5 7 10 3 8 9 10 4 1 2 3 4 7 1 2 3 4 5 7 10 2 3 2 2 1 2 4 3 2 3 4 3 3 1 2 4 3 1 4 2 0 0	2 1 0 2

Problem E
Edgetown's Traffic Jams

Source file name: `edgetown.c`, `edgetown.cpp` *or* `edgetown.java`

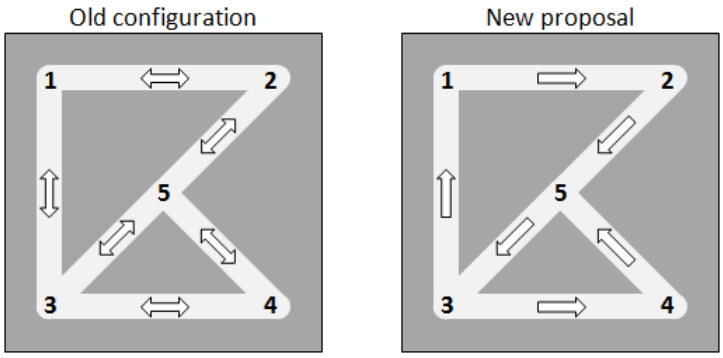
Edgetown is proudly a full-communicated city. That means that Edgetowners have decided that it must be possible to travel from any point in the city to any other point, using a car. Every point in Edgetown is located on a block, and every block connects two intersections (so that there are no intersections between a block nor two blocks connecting the same intersections). The city authorities have traditionally warranted this feature ensuring that the city plan is connected (i.e., there are not isolated intersections) and that some blocks are two-way.

Given two intersections X and Y in Edgetown, the *distance* from X to Y is measured as the minimum number of blocks that should be traveled going from X to Y. The following diagram shows a possible configuration with eight blocks and eleven intersections (marked with asterisks).



Lately there have been traffic jams at several points, almost at all times. Experts recommend a simple solution: just change some two-way blocks to be one-way blocks. However, it is clear that this should be done carefully, because accessibility among city points may be lost. On the other hand, even if accessibility is guaranteed, it is possible that distances between specific intersections may be significantly augmented.

After a lot of discussions, the Mayor's advisers have recommended to accept any proposal that increases the distance between any two intersections by a factor A augmented by a constant B, with respect to the old configuration (i.e., if the actual distance from one intersection to another is x, then the new distance should be at most $A \cdot x + B$).



You are hired to develop a program to decide if a given proposal to orient city blocks satisfies the requirements.

Input

There are several cases to analyze. Each case is described with a set of lines:

- The first line contains a non-negative integer n ($3 \leq n \leq 100$) that represents the number of intersections in Edgetown. Suppose that the intersections are identified by natural numbers in the set $\{1, \ldots, n\}$.
- The line $i+1$ ($1 \leq i \leq n$) begins with the number i and follows with a list of intersection numbers different from i (without repetitions). That means that the intersection i is connected by a block to each of the intersections numbered by elements in the list.
- The next n lines describe, with the same already specified format, the new proposal. In the description of the blocks in the proposal should be understood that the blocks are oriented going out from the first element in the line to each of the adjacent elements (the same fact applies to the old configuration).
- The case description ends with a line with two integer values A and B ($0 \leq A \leq 10$, $0 \leq B \leq 10$).

The last test case is followed by a line containing a single 0.

The input must be read from the file edgetown.in.

Output

For each case print one line with the word "`Yes`" if the proposal satisfies the given requirements, or the word "`No`" otherwise. Answers should be left aligned.

The output must be written to standard output.

Sample Input	**Sample output**
5 1 2 3 2 1 5 3 4 5 1 4 3 5 5 2 3 4 1 2 2 5 3 1 4 4 5 5 3 1 2 5 1 2 3 2 1 5 3 4 5 1 4 3 5 5 2 3 4 1 2 2 5 3 1 4 4 5 5 3 2 0 3 1 2 2 1 3 3 1 2 1 2 2 3 3 1 0 2 0	Yes No Yes

Problem F
Flight Control

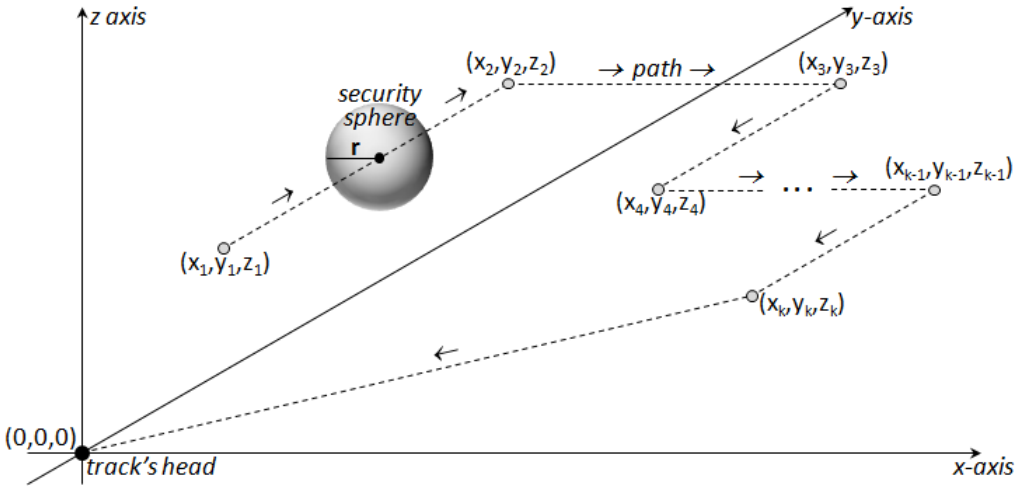
Source file name: `flight.c`, `flight.cpp` *or* `flight.java`

Air traffic controllers are the people who expedite and maintain a safe and orderly flow of air traffic in the global air traffic control system. The position of the air traffic controller is one that requires highly specialized skills. Because controllers have an incredibly large responsibility while on duty, this profession is, according to Wikipedia, "regarded around the world as one of the most difficult jobs today, and can be notoriously stressful depending on many variables (equipment, configurations, weather, traffic volume, human factors, etc.)".

An air traffic controller has access to the following information for each aircraft on the screen:

- *size*: a positive integer number r indicating the radius (measured in meters) of a *security sphere* whose center always is the current position of the aircraft;
- *speed*: a positive integer number s indicating the constant speed (measured in meters per second) of the aircraft along its route;
- *route*: a sequence of points $(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_k, y_k, z_k)$ with integer coordinates (measured in meters) in the three-dimensional Cartesian plane, indicating the path followed by the aircraft before it returns to the head of the *track* which is located at the point $(0, 0, 0)$.

Each aircraft begins its journey at the position (x_1, y_1, z_1) and then, it directly flies in a straight line at constant speed from (x_1, y_1, z_1) to (x_2, y_2, z_2), ..., from $(x_{k-1}, y_{k-1}, z_{k-1})$ to (x_k, y_k, z_k), and finally from (x_k, y_k, z_k) to $(0, 0, 0)$, where each position is relative to the head of the track. After the aircraft arrives at the head of the track, it disappears from the controller's screen.



On a day-to-day basis, air traffic controllers deal with conflict detection and resolution. Therefore, for an air traffic controller is very important to have an alarm system to indicate the specific points where it must take corrective actions to prevent accidents.

It is noteworthy that, geometrically speaking, a *conflict warning* occurs when the security spheres of two aircrafts touch. Formally, a *conflict warning* begins when two aircrafts approach at a distance less than or equal to the sum of the radius of their security spheres, is maintained while this condition is satisfied, and ends when their security spheres stop touching (i.e., when the distance between both is greater than the sum of the radius of their security spheres). Distances are measured with an acceptable error threshold $\varepsilon > 0$.

Despite years of effort and the billions of dollars that have been spent on computer software designed to assist air traffic control, success has been largely limited to improving the tools at the disposal of the controllers. However, today you have the chance to improve the impact of computer software in the air traffic control world.

You have been hired by the *International Center of Planning Control* (ICPC) to determine the quality of the traffic routes defined by the *Aircraft Controller Management* (ACM), through the measurement of the number of dangerous situations that should fix the air traffic controller. Your task is to write a program that, given the information of two aircrafts, determines the number of different conflict warnings that would arise if both aircrafts follow the scheduled route starting at the same time and finishing at the track's head.

Input

The first line of the input contains the number of test cases. Each test case specifies the information of the two studied aircrafts, where each aircraft is described as follows:

- The first line contains three integer numbers r, s and k separated by blanks ($1 \leq r \leq 100$, $1 \leq s \leq 1000$, $1 \leq k \leq 100$), where r is the radius of the security sphere (in meters), s is the speed (in meters per second), and k is the number of points that define the route of the aircraft.
- Each one of the next k lines contains three integer numbers x_i, y_i, and z_i separated by blanks ($-10^4 \leq x_i \leq 10^4$, $-10^4 \leq y_i \leq 10^4$, $0 \leq z_i \leq 10^4$), describing the coordinates (in meters) of the i-th point on the route of the aircraft ($1 \leq i \leq k$).

For each route, you may assume that either $x_i \neq x_{i+1}$, $y_i \neq y_{i+1}$ or $z_i \neq z_{i+1}$ for all $1 \leq i < k$, and that either $x_k \neq 0$, $y_k \neq 0$ or $z_k \neq 0$. The acceptable error threshold is $\varepsilon = 10^{-10}$ meters.

The input must be read from the file flight.in.

Output

For each test, print one line informing the number of different conflict warnings.

The output must be written to standard output.

Sample Input	**Sample output**
7 20 300 2 10000 1000 5000 1000 100 500 20 100 3 10000 1000 2000 1000 100 500 100 0 0 20 300 2 0 10000 5000 0 -10000 5000 20 300 3 10000 0 5010 -10000 0 5010 -8000 1000 3010 20 300 2 0 10000 5000 0 -10000 5000 20 300 2 10000 0 5010 -10000 0 5010 25 200 2 3000 6000 3000 4000 5000 3000 25 200 2 3000 6000 3005 4000 5000 3005 20 300 2 5000 4000 3000 4000 5000 3000 20 300 2 3000 6000 3005 4000 5000 3005 10 100 1 -1000 0 0 10 100 2 1000 21 0 -1000 21 0 10 100 1 -1000 0 0 10 100 2 1000 20 0 -1000 20 0	0 1 2 1 1 0 1

Problem G
Gas Stations

Source file name: `gas.c`, `gas.cpp` *or* `gas.java`

G gas stations are authorized to operate over a road of length L. Each gas station is able to sell fuel over a specific *area of influence*, defined as the closed interval $[x-r, x+r]$, where x is the station's *location* on the road ($0 \leq x \leq L$) and r is its *radius of coverage* ($0 < r \leq L$). The points covered by a gas station are those within its radius of coverage.

It is clear that the areas of influence may interfere, causing disputes among the corresponding gas stations. It seems to be better to close some stations, trying to minimize such interferences without reducing the service availability along the road.

The owners have agreed to close some gas stations in order to avoid as many disputes as possible. You have been hired to write a program to determine the maximum number of gas stations that may be closed, so that every point on the road is in the area of influence of some remaining station. By the way, if some point on the road is not covered by any gas station, you must acknowledge the situation and inform about it.

Input

The input consists of several test cases. The first line of each test case contains two integer numbers L and G (separated by a blank), representing the length of the road and the number of gas stations, respectively ($1 \leq L \leq 10^8$, $1 \leq G \leq 10^4$). Each one of the next G lines contains two integer numbers x_i and r_i (separated by a blank) where x_i is the location and r_i is the radius of coverage of the i-th gas station ($0 \leq x_i \leq L$, $0 < r_i \leq L$). The last test case is followed by a line containing two zeros.

The input must be read from the file gas.in*.*

Output

For each test case, print a line with the maximum number of gas stations that can be eliminated, so that every point on the road belongs to the area of influence of some not closed station. If some point on the road is not covered by any of the initial G gas stations, print -1 as the answer for such a case.

The output must be written to standard output.

Sample Input	**Sample output**
40 3 5 5 20 10 40 10 40 5 5 5 11 8 20 10 30 3 40 10 40 5 0 10 10 10 20 10 30 10 40 10 40 3 10 10 18 10 25 10 40 3 10 10 18 10 25 15 0 0	0 2 3 -1 1