XXV Maratón Nacional de Programación
ACIS REDIS 2011
ACM ICPC

Problemas

(Este conjunto contiene 10 problemas; páginas numeradas de 1 a 22)

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

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

Sample output

```
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
	```

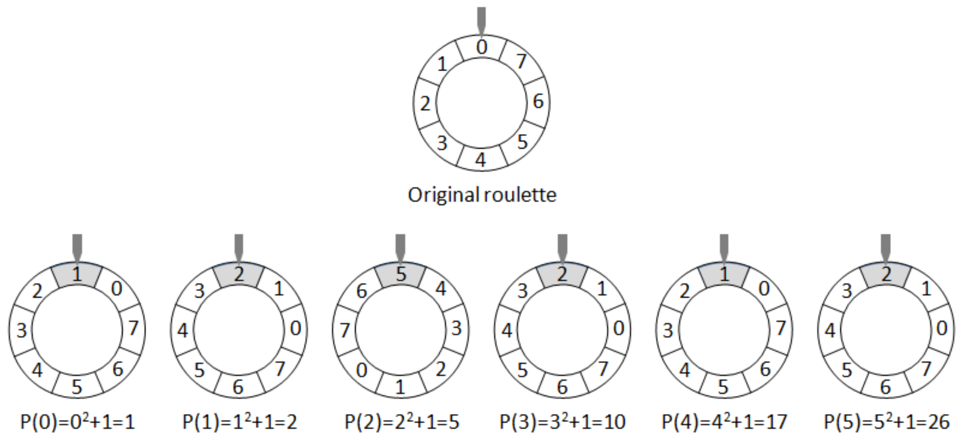
Problem D
Digital Roulette

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

John is developing a videogame that allows players to bet in a wall roulette. Players may bet for integer numbers from 0 to N, for some $N \geq 0$ that represents the maximum number in the roulette.

Of course, the roulette behaves digitally. As a matter of fact, John designed its way to choose a value in the interval $0..N$ (the result of spinning the roulette) with a digital trigger that moves the roulette with a force that depends on an integer value x randomly chosen in the interval $0..M$, where $M \geq 0$ (M is the maximal appliable force). The roulette turns around a distance equivalent to $P(x)$, where P is a polynomial with integer coefficients. One distance unit represents a displacement of one roulette number, counting clockwise.

It is clear that some result values may be produced by different chosen force values. Also, depending on the mechanism parameters, some numbers in the roulette may be not attainable regardless of the force value. For example, if N=7, M=5 and $P(x)$=x^2+1, the mechanism can generate only three different results:



John wants to know how many different result values may be attained by his mechanism. Can you help him?

Input

There are several cases to analyze. Each case is described by three lines:

- The first line contains two non-negative integer numbers N and M, separated by a blank ($1 \leq N \leq 10^7$, $0 \leq M \leq 10^5$).
- The second line contains an integer k, the grad of the polynomial P ($0 \leq k \leq 10$).

- The third line contains $k{+}1$ integers $a_0, a_1, \ldots, a_k$ separated by blanks, indicating the integer coefficients that define the polynomial P, i.e., $P(x) = a_k x^k + \cdots + a_1 x + a_0$. You can assume that $0 \leq a_i \leq N$ for each $0 \leq i \leq k$. If $k>0$ then you may assume that $a_k \neq 0$.

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

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

Output

For each case, print one line indicating how many different numbers are attainable by John's mechanism.

The output must be written to standard output.

Sample Input

```
7 5
2
1 0 1
99 10
0
5
99 10
1
5 25
99 10
1
3 29
99 10
2
3 29 31
0 0
```

Sample output

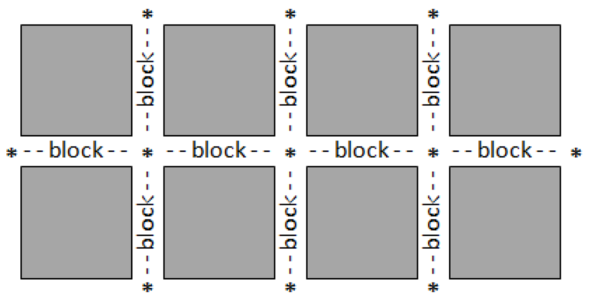
```
3
1
4
11
10
```

Problem E
Edgetown's Traffic Jams

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

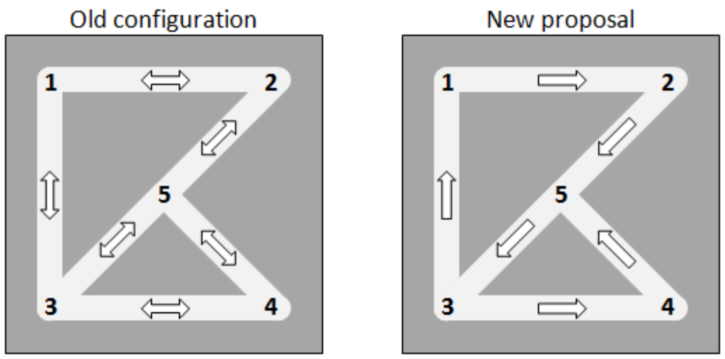
Edgetown is proudly a full-communicated city. That means that Edgetowners have decided that it must be possible to travel from any point in the city to any other point, using a car. Every point in Edgetown is located on a block, and every block connects two intersections (so that there are no intersections 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

```
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
```

Sample output

```
Yes
No
Yes
```

Problem F
Flight Control

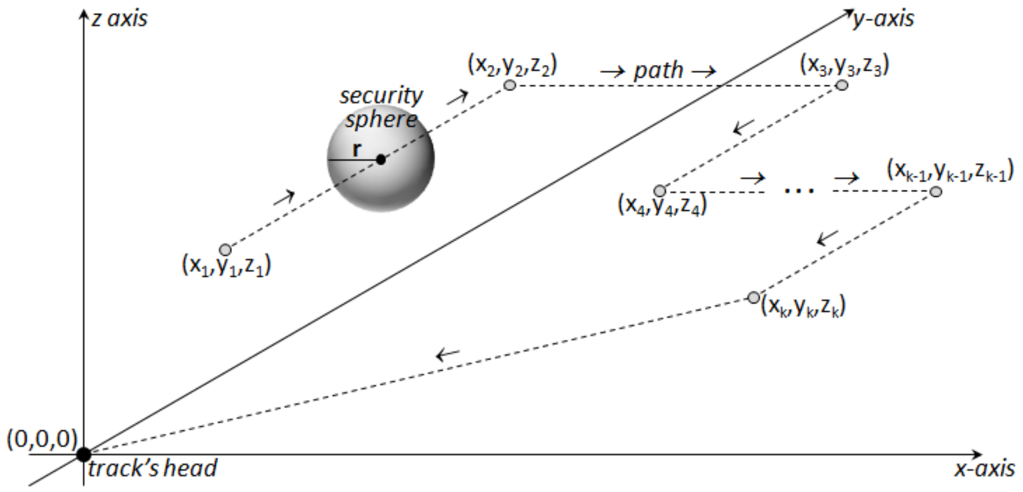
Source file name: `flight.c`, `flight.cpp` *or* `flight.java`

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

```
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
```

Sample output

```
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

```
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
```

Sample output

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

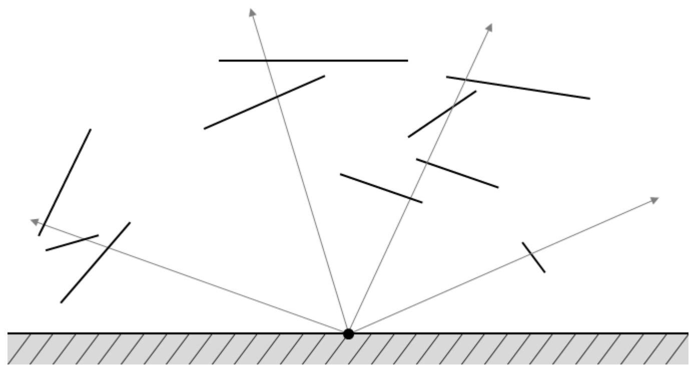
Problem H
Handgun Shooting Sport

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

Crystal billboards present snipers with the opportunity of shooting sports in the abandoned neighborhoods of Crystal City. One popular shooting sport for a sniper is to destroy all crystal billboards in front of a building. The rules of this game enforce the sniper to fix his/her gun to the ground of the building's roof, so it can rotate freely in any direction. However, once the gun is fixed, it cannot be placed in any other location. The goal of the game is to destroy all crystal billboards firing the minimum number of shots.

Every time the sniper fires, any crystal billboard in the way of the bullet is destroyed, even if the bullet only touches one extreme of the billboard. A bullet never changes direction or speed once it is fired, even if it destroys any crystal billboards.

The following figure depicts a sniper in a fixed location facing a layout of ten crystal billboards. The sniper has destroyed all ten crystal billboards and has achieved the goal of the game because he/she destroyed all crystal billboards firing the minimum number of shots possible.



Given the initial location of a sniper and a layout of crystal billboards, your task is to determine the minimum number of shots that would destroy all crystal billboards.

Input

The first line of the input contains a natural number B defining the number of crystal billboards in the shooting layout ($1 \le B \le 10^3$). Each one of the following B lines contains four integer numbers x_1, y_1, x_2, y_2 separated by blanks, defining the coordinates of the line segment with extremes (x_1, y_1) and (x_2, y_2) ($-10^3 \le x_1 \le 10^3$, $0 < y_1 \le 10^3$, $-10^3 \le x_2 \le 10^3$, $0 < y_2 \le 10^3$, $y_1 \cdot x_2 \neq x_1 \cdot y_2$). You can assume that the shooting layout is modeled as the region on the Cartesian plane above the x-axis, that the sniper is located in the origin $(0, 0)$, that each crystal billboard is represented with a line segment whose extremes are not collinear with the origin, and that no pair of crystal billboards intersects. The last test case is followed by a line containing a zero.

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

Output

For each test case, a line must be printed with the minimum number of shots that fired from the sniper location would destroy all crystal billboards in the shooting layout.

The output must be written to standard output.

Sample Input

```
10
-309 98 -258 204
-303 83 -251 98
-218 111 -287 31
-145 204 -23 257
-129 272 59 272
-8 159 74 130
150 146 68 174
59 196 128 242
98 256 241 235
197 61 173 92
3
-100 10 -100 50
-50 100 50 100
100 10 100 50
5
-100 100 100 100
-80 120 80 120
-60 140 60 140
-40 160 40 160
-20 180 20 180
2
-50 50 0 50
-10 70 50 70
2
-50 50 0 50
0 70 50 70
2
-50 50 0 50
10 70 50 70
0
```

Sample output

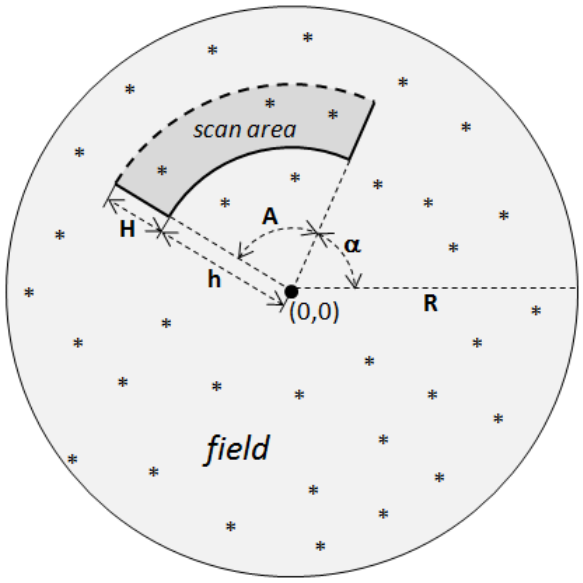
```
4
3
1
1
1
2
```

Problem I
Inspecting Radars

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

Radars Inc. is a worldwide renowned radar maker, whose excellent reputation lies on strict quality assurance procedures and a large variety of radar models that fit all budgets. The company hired you to develop a detailed *inspection* that consists of a sequence of E *experiments* on a specific *surveillance model*.

There is a *field* represented with a polar coordinate plane that contains N objects placed at positions with integer polar coordinates. The inspected model is located at the origin $(0, 0)$ of the field and can detect objects at a distance less than its *detection range* R through a *scan area* defined by four adjustment parameters α, A, h, and H, whose meaning is illustrated with the following figure:



Formally, the *scan area* of the model is the region described by the set of polar points

$$\{(r, \theta) |\, h \le r < h + H,\ \alpha \le \theta \le \alpha + A\}$$

α, A, h and H are four integer values where:

- α specifies the *start angle* of the radar's scan area ($0 \le \alpha < 360$);
- A specifies the *opening angle* of the radar's scan area ($0 \le A < 360$);
- h gives the *internal radius* of the radar's scan area ($0 \le h < R$); and
- H gives the *height* of the radar's scan area ($1 \le H \le R$).

An object placed at (r, θ) will be displayed by the model if $h \le r < h+H$ and $\alpha \le \theta \le \alpha+A$, where the last inequality should be understood modulo 360° (i.e., adding and comparing angles in a circle).

Given N objects placed on the field, you must develop an inspection of the surveillance model through the implementation of E experiments with specific parameterizations. For each experiment you have to find the maximal number of objects on the field that the radar should display if the parameters α ($0 \le \alpha < 360$) and h ($0 \le h < R$) are free to set (as integer numbers), and the parameters H ($1 \le H \le R$) and A ($0 \le A < 360$) are given.

Input

The input consists of several test cases. Each test case is described as follows:

- A line with two integer numbers N and R separated by blanks, representing (respectively) the number of objects located on the field and the detection range of the model ($1 \le N \le 10^4$, $2 \le R \le 10^2$).
- Each one of the following N lines contains two integer numbers r_i and θ_i separated by blanks, specifying the integer polar coordinates (r_i, θ_i) of the i-th object ($1 \le r_i < R$, $0 \le \theta_i < 360$, $1 \le i \le N$).
- The next line has an integer number E indicating the number of experiments of the inspection ($1 \le E \le 10^2$).
- Each one of the following E lines contains two integer numbers H_j and A_j separated by blanks, representing (respectively) the fixed height and the fixed opening angle that parameterize the j-th experiment ($1 \le H_j \le R$, $0 \le A_j < 360$, $1 \le j \le E$).

For each test case you can suppose that there are not two different objects placed at the same integer polar coordinate. The last test case is followed by a line containing two zeros.

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

Output

For each test case of the input, print E lines where the j-th line contains the maximal number of objects on the field that the radar should display according to the parameterization given for the j-th experiment ($1 \le j \le E$).

The output must be written to standard output.

Sample Input	**Sample output**

```
6 100
15 7
15 60
40 15
50 15
45 30
45 90
2
2 1
100 359
9 100
15 7
15 60
40 15
50 15
45 30
45 90
40 45
50 45
78 100
6
100 359
11 30
10 30
11 29
5 30
11 10
0 0
```
	```
	1
	6
	9
	5
	3
	3
	2
	2
	```

Problem J
Philip J. Fry Problem

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

It's been a few months since Bender solved his famous bending problem. A special request has arrived to *Planet Express Inc.*, the interplanetary courier company where Bender works. Professor Farnsworth (founder, CEO, and chairman of the board) immediately prepared the mission and summoned his crew: Philip J. Fry, Turanga Leela, and Bender. To succeed in the mission, the crew has to make n trips τ_1, τ_2, $\ldots$, τ_n, in the exact order established by Professor Farnsworth. He has also provided the crew with a notebook that specifies how many minutes each trip in the spaceship will take.

Nibbler, from *Futurama*.
©*The Curiosity Company* and 20^{th} *Century Fox.*

Nibbler, Leela's stupid pet, is also a member of the crew and a critical element to succeed in the mission. Nibbler's feces, which it expels every now and then, consist of spheres of dark matter that can be used to increase the speed of the spaceship. In fact, a sphere used during a trip halves its duration. Bender's role in the mission is to pick up the heavy spheres and put them in the reactor of the spaceship. However, he will never put two spheres during the same trip: the accumulation of dark matter is extremely dangerous and may destroy the spaceship.

In each trip of the mission, Nibbler expels a certain number of dark matter spheres that can be used to decrease the duration of any of the upcoming trips. In other words, a sphere of dark matter produced during the trip τ_i can be used to duplicate the speed of the spaceship in one of the trips τ_j, with $i<j\leq n$.

Fry is responsible for planning how to use the spheres to reduce the total travel time. Your task is to help Fry in determining what is the minimum duration of the mission if he uses Nibbler's spheres cleverly.

Input

The input consists of several test cases. The first line of each case contains an integer n indicating the number of trips ($1\leq n\leq 100$). Then, n lines follow describing each of the trips τ_1, τ_2, $\ldots$, τ_n: each line contains two integers t_i and b_i separated by blanks, where t_i ($2\leq t_i\leq 1000$) indicates the duration in minutes specified by Professor Farnsworth for the trip τ_i (t_i is always even) and b_i ($0 \leq b_i \leq n$) indicates the number of dark matter spheres that Nibbler will expel during the trip τ_i. The last test case is followed by a line containing a single 0.

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

Output

For each test case, print a line with an integer number indicating the minimum time required to complete the mission.

The output must be written to standard output.

Sample Input

```
2
24 1
10 0
2
10 1
24 0
3
10 0
24 0
38 0
3
10 1
24 0
14 0
3
10 1
24 0
38 0
3
10 1
24 1
38 0
3
10 3
24 0
38 1
0
```

Sample output

```
29
22
72
36
53
41
41
```