Bucknell University
Spring Programming Contest
Expert Division, 2014

Pull off your eyepatch and take a closer look at Atlantis!

Name this problem Eye.

Your pirate fleet is preparing an exploration mission to Atlantis, discovered just off the coast of your favorite South Seas island. Atlantis is apparently not a continent but is composed of millions of small, unexplored islands. One of the goals of the mission is to build a complete map of the tiny islands. Your fleet is intending to map the islands by tracking the coast lines. The idea is to leave a row boat at one point on the coast of an island and let it's crew discover the map of the entire coast, then come back as the boat crew returns to the starting place and take it to the next island. Your crew is currently afraid of sharks in the water and tigers in the middle of the islands and has decided to map each island while **carrying** the row boat along the edge of the island (on the land, not in the water).

You are working with the pirate team that is programming the navigation software for these missions, and some decisions were already made: the surface will be discretised into squared, equal-sized areas; for simplification purposes, each area will be considered as being entirely covered by land or water. Also, the row boat will always start its job on a square of the coast, with the sea at its right; therefore, the coast will be tracked in a counter clockwise direction (see Figure 1).

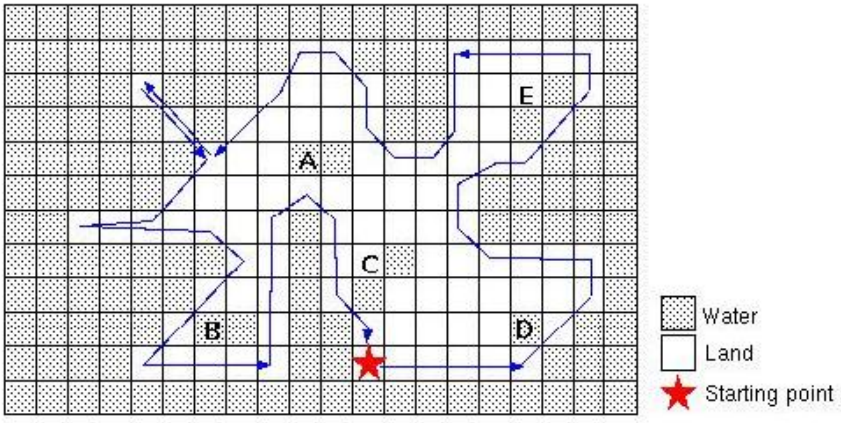


Figure 1

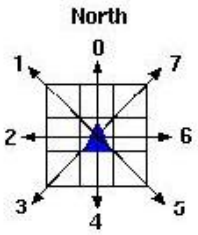


Figure 2

The row boats will have a limited perception range (its foggy in Atlantis) that will be able to determine the kind of surface of the 8 areas neighboring the one where the row boat and crew are located. Its movement capabilities will also be limited: the row boat will only have the possibility of (i) rotating to one of 8 fixed directions (as integers from 0 to 7, 0 being North (see Figure 2) and (ii) moving to the neighbor position in front of it. Each time the row boat moves to a new position, the pirates carrying it can see the new surroundings. There are no obstacles to worry about: the islands have smooth, sand coasts. There are lakes (e.g., A, B, C, D and E in Figure 1), but the crew must not waste time with them: the goal is to track the seacoast only.

Given the current position and direction of the row boat, and a view of the immediate surrounding world, decide the direction of the next move. Note that you are not being asked to program the whole coast tracking software; you are just programming a part of it. The direction must be computed in such a way that an iterative call of the program, with views from an environment as the one described, would allow the row boat to track the coast.

Your program must be able to process several scenarios. Each scenario comprises the row boat's current position and direction, and a view. Figure 3 illustrates three hypothetical scenarios and the intended result: the direction of the next move.

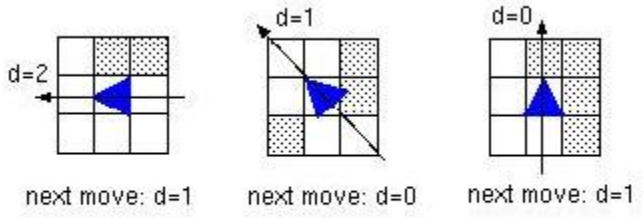


Figure 3

Input

The input represents several scenarios. Input for each scenario consists of 9 lines as follows: First line: x y d, where x and y are the (always positive) coordinates of the row boat's current position, and d is the row boat's current direction, an integer such that 0<=d<=7 (see Figure 2); The next 8 lines have the format: x_i y_i s_i, where s_i is a number representing the surface type of the neighboring position (x_i, y_i); land surface is represented by 1, and water surface by 0. You can assume that the origin of coordinate system is in the left bottom corner of the map. Successive values in a line are separated by one or more blanks. The integer -1 follows the data of the last scenario.

Output

For each given scenario, your program has to output the direction of the row boat's next movement. Output for successive scenarios should be written in successive lines.

Sample Input

```
22 25 2
22 26 0
21 26 1
21 25 1
21 24 1
22 24 1
23 24 1
23 25 1
23 26 0
21 26 1
21 27 1
20 27 1
20 26 1
20 25 0
21 25 1
22 25 1
22 26 0
22 27 0
21 27 0
21 28 0
20 28 1
20 27 1
20 26 1
21 26 1
22 26 0
22 27 0
22 28 0
-1
```

Sample Output

```
1
0
1
```

Some fancy dancing in your hula skirt may increase your stash of gold!

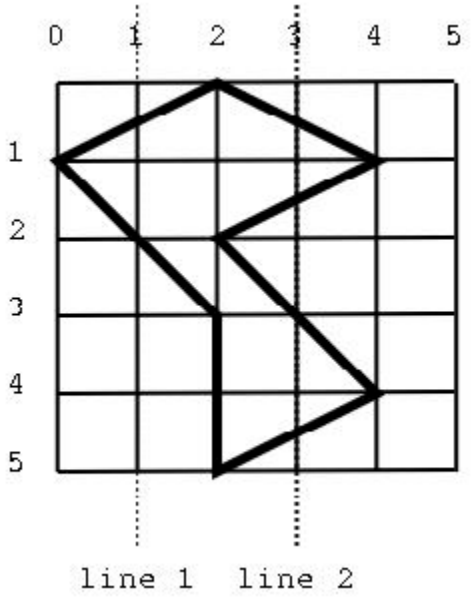
Name this problem Hula.

In the Age of Discovery many sailboats that carried gold and other treasures never made it to their destinations and sank in the sea. For a modern day pirate, it's not necessary to actually attack and rob ships to get rich, it's sufficient to just take a map of sea that depicts approximate locations of shipwrecks, traverse those regions and look for gold using electronic detectors.

As modern pirates, you'd like to help to collect that sunken gold. You happen to own a small rowboat (it just got back from mapping Atlantis) you want to send it to look for gold. You have a map of the shape of your local region (with many sunken treasures in it), and you want to decide whether it is useful for your ship to pass through this region along a chosen line. A pass is considered useful is the ship passes over a treasure region that exceeds some threshold length.

A treasure area is essentially a polygon where vertices are approximated by integer values. Ships will always follow vertical lines in their pass, and never pass over the vertices of the polygon: in those points there are usually dangerous vortexes or shallows.

Your task is to decide if the pass along a given line is useful. You will be given a sequence of coordinates defining the treasure area, a value for the threshold and the x-coordinate of the line to be followed by the ship.



As an example, the above figure illustrates the location of a treasure area in the sea, where "line1" and "line2" are possible flight paths. If the threshold is set to 2 length units, line 2 will be an interesting water path but line 1 will be not.

Given the shape of the treasure area, the threshold and the x-coordinate of the line to be followed by the ship, decide whether the line is useful. Assume that the given x-coordinate does not intersect the polygon at any vertex.

Input

The first line of the input contains the number T of test cases, followed by T input blocks.

The first line of each input block consists of one integer N, the number of vertices of the polygon representing the treasure area. Each of the following N input lines contains the X-Y integer coordinates of one vertex. The following line has an integer indicating the threshold for the decision. The last line of each input block has an integer for the x-coordinate to be evaluated. The set of vertices starts at an arbitrary point in the polygon and there are at most 10000 vertices in a polygon.

Output

For each input block output a single line, containing "YES" if the chosen line is useful and "NO" otherwise.

Sample Input

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

```
1
```

Sample Output

```
YES
NO
```

Lei some flowery stock bets!

Name this problem Lei.

If you ever see a televised report on stock market activity, you'll hear the anchorperson say something like ``Gainers outnumbered losers14 to 9," which means that for every 14 stocks that increased in value that day, approximately 9 other stocks declined in value. Often, as you hear that, you'll see on the screen something like this: Gainers 1498, Losers 902

As a pirate with a head for numbers, you'll notice that the anchorperson could have said ``Gainers outnumbered losers 5 to 3", which is a more accurate approximation to what really happened. After all, the exact ratio of winners to losers is (to the nearest millionth) 1.660754, and he reported a ratio of 14 to 9, which is 1.555555, for an error of 0.105199; he could have said ``5 to 3", and introduced an error of only 1.666667-1.660754=0.005913. The estimate ``5 to 3" is not as accurate as ``1498 to 902" of course; evidently, another goal is to use small integers to express the ratio. So, why did the anchorperson say ``14 to 9?" Because his algorithm is to lop off the last two digits of each number and use those as the approximate ratio.

Your first urge, as a pirate, is of course to raid the TV studio and plunder it for gold but you quickly realize that you are smarter than the anchorman and could simply give better approximations.

What the anchorman needs is a list of rational approximations of increasing accuracy, so that he can pick one to read on the air. Specifically, he needs a sequence a_1, a_2, ..., a_n where a_1 is a rational number with denominator 1 that most exactly matches the true ratio of winners to losers (rounding up in case of ties), a_{i+1} is the rational number with least denominator that provides a more accurate approximation than a_i, and a_n is the exact ratio, expressed with the least possible denominator. Given this sequence, the anchorperson can decide which ratio gives the best tradeoff between accuracy and simplicity.

For example, if 5 stocks rose in price and 4 fell, the best approximation with denominator 1 is 1/1; that is, for every stock that fell, about one rose. This answer differs from the exact answer by 0.25 (1.0 vs 1.25). The best approximations with two in the denominator are 2/2 and 3/2, but neither is an improvement on the ratio 1/1, so neither would be considered. The best approximation with three in the denominator 4/3, is more accurate than any seen so far, so it is one that should be reported. Finally, of course, 5/4 is exactly the ratio, and so it is the last number reported in the sequence.

Can you automate this process and help the anchorpeople (and make them pay you voluntarily without all the bother of plundering their studio)?

Input

The input contains several pairs of positive integers. Each pair is on a line by itself, beginning in the first column and with a space between the two numbers. The first number of a pair is the number of gaining stocks for the day, and the second number is the number of losing stocks for

the day. The total number of stocks never exceeds 5000. The input is terminated by a line with two zeros.

Output

For each input pair, the standard output should contain a series of approximations to the ratio of gainers to losers. The first approximation has '1' as denominator, and the last is exactly the ratio of gainers to losers, expressed as a fraction with least possible denominator. The approximations in between are increasingly accurate and have increasing denominators, as described above.

The approximations for a pair are printed one to a line, beginning in column one, with the numerator and denominator of an approximation separated by a slash (``/''). A blank line separates one sequence of approximations from another.

Sample Input

```
5 4
1498 902
0 0
```

Sample Output

```
1/1
4/3
5/4

2/1
3/2
5/3
48/29
53/32
58/35
63/38
68/41
73/44
78/47
83/50
88/53
93/56
377/227
470/283
563/339
656/395
749/451
```

Fend the colors off with your sword!

Name this problem Sword.

Pirate captains are very fond of colors.. black.. white.. red.. all shades.. Your captain, the Dread Pirate Guattery is considering having you repaint all his ships but he just can't decide on a color scheme. His decorator tells him that he's simply picked too many colors. He wants you to write software to reduce his current set of chosen colors to a smaller set.

A color reduction is a mapping from a set of discrete colors to a smaller one. The solution to this problem requires that you perform just such a mapping in a standard twenty-four bit RGB color space. The input consists of a target set of sixteen RGB color values, and a collection of arbitrary RGB colors to be mapped to their closest color in the target set. For our purposes, an RGB color is defined as an ordered triple (R,G,B) where each value of the triple is an integer from 0 to 255. The distance between two colors is defined as the Euclidean distance between two three-dimensional points. That is, given two colors (R_1, G_1, B_1) and (R_2, G_2, B_2), their distance D is given by the equation

D = square root($(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2$)

Solve the decorating problems or prepare to walk the plank!

Input

The input is a list of RGB colors, one color per line, specified as three integers from 0 to 255 delimited by a single space. The first sixteen colors form the target set of colors to which the remaining colors will be mapped. The input is terminated by a line containing three -1 values.

Output

For each color to be mapped, output the color and its nearest color from the target set.

Sample Input

```
0 0 0
255 255 255
0 0 1
1 1 1
128 0 0
0 128 0
128 128 0
0 0 128
126 168 9
35 86 34
133 41 193
```

```
128 0 128
0 128 128
128 128 128
255 0 0
0 1 0
0 0 0
255 255 255
253 254 255
77 79 134
81 218 0
-1 -1 -1
```

Sample Output

```
(0,0,0) maps to (0,0,0)
(255,255,255) maps to (255,255,255)
(253,254,255) maps to (255,255,255)
(77,79,134) maps to (128,128,128)
(81,218,0) maps to (126,168,9)
```

Hold your horses and keep your hat on already!

Name this problem Hat.

In your last plundering venture, you brought back several links of chain, some of which may be connected. They are made from the purest gold and coveted by pirates everywhere.

You want the pieces joined into a single end-to-end strand of chain to display draped off your finest pirate hat. You take the links to your local pirate jeweler who tells you that the cost of joining them depends on the number of chain links that must be opened. Being a thrifty pirate, you don't want to spend a penny more than you have to. In order to minimize the cost, you carefully calculate the minimum number of links that have to be opened to rejoin all the links into a single sequence.

Input

The input consists of descriptions of sets of chain links, one set per line. Each set is a list of integers delimited by one or more spaces. Every description starts with an integer n, which is the number of chain links in the set, where $1 \leq n \leq 15$. We will label the links 1, 2, ..., n. All links are initially closed. The integers following n describe which links are connected to each other. Every connection is specified by a pair of integers i, j where $1 \leq i, j \leq n$ and $i \neq j$, indicating that chain links i and j are connected, i.e., one passes through the other. There is no limit to the number of links that can be connected. If no links are connected, there will be no numbers after n. The input is terminated by a 0.

Output

For each set of chain links in the input, output a single line which reads

Minimum links to open is M

where M is the minimal number of links that have to be opened and closed such that all links can be joined into one single chain.

Sample Input

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

Sample Output

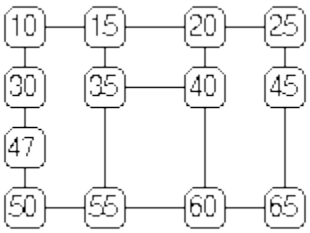
```
Minimum links to open is 1
Minimum links to open is 2
Minimum links to open is 1
Minimum links to open is 1
Minimum links to open is 1
```

Hook into the local pirate network!

Name this problem Hook.

Your pirate fleet has its own computer network. Raids on this network are almost always fatal… Pirates also get impatient quickly and can't stand to have network messages (packets) looping around forever inside their network. To avoid this, each message includes a Time To Live (TTL) field. This field contains the number of nodes (ships, computers, etc.) that can retransmit the message, forwarding it along toward its destination, before the message is unceremoniously dropped (off the plank). Each time a ship receives a message it decrements the TTL field by 1. If the destination of the message is the current ship, then the TTL field's value is ignored. However, if the message must be forwarded, and the decremented TTL field contains zero, then the message is not forwarded.

In this problem you are given the description of a number of networks, and for each network you are asked to determine the number of nodes that are not reachable given an initial node and TTL field value. Consider the following pirate network where the ships have been numbered:



If a message with a TTL field of 2 was sent from node 35 it could reach nodes 15, 10, 55, 50, 40, 20 and 60. It could not reach nodes 30, 47, 25, 45 or 65, since the TTL field would have been set to zero on arrival of the message at nodes 10, 20, 50 and 60. If we increase the TTL field's initial value to 3, starting from node 35 a message could reach all except node 45.

Input

There will be multiple network configurations provided in the input. Each network description starts with an integer NC specifying the number of connections between network nodes. An NC value of zero marks the end of the input data. Following NC there will be NC pairs of positive integers. These pairs identify the nodes that are connected by a communication line. There will be no more than one (direct) communication line between any pair of nodes, and no network will contain more than 30 nodes. Following each network configuration there will be multiple queries as to how many nodes are not reachable given an initial node and TTL field setting. These queries are given as a pair of integers, the first identifying the starting node and the second giving the initial TTL field setting. The NC pairs and the queries may all be on one line or may

be on several lines. Each set of queries is terminated by a pair of zeroes. The input is terminated by a single 0.

Output

For each query display a single line showing the number of nodes not reachable, the starting node number, and the initial TTL field setting in the format seen below.

Sample Input

```
16
10 15 15 20 20 25 10 30 30 47 47 50 25 45 45 65
15 35 35 55 20 40 50 55 35 40 55 60 40 60 60 65
35 2 35 3 0 0
16
8 15 15 20 20 25 8 30 30 47 47 50 25 45 45 65
15 3 3 55 20 40 50 55 3 40 55 60 40 60 60 65
3 2 3 3 0 0
0
```

Sample Output

```
5 nodes not reachable from node 35 with TTL = 2.
1 nodes not reachable from node 35 with TTL = 3.
5 nodes not reachable from node 3 with TTL = 2.
1 nodes not reachable from node 3 with TTL = 3.
```

You may need a compass to keep your pirates in order!

Name this problem Compass.

The pirate mess hall is in disarray. It consists of long tables and benches. The pirates all sit down as fast as possible for meals (they are afraid of not getting a seat). Unfortunately, there is a protocol that says the pirates on each bench must be ordered by rank before they can eat. Pirates generally refuse to scoot over to get into the right order so they are reduced to swapping seats. Your Captain, the Dread Pirate Guattery has realized that these benches are just like arrays and is relying on your Computer Science background to help keep his unruly pirates in order.

Sorting an array can be done by swapping certain pairs of adjacent entries in the array. This is the fundamental technique used in the well-known bubble sort. If we list the identities of the pairs to be swapped, in the sequence they are to be swapped, we obtain what might be called a swap map. For example, suppose we wish to sort bench B whose pirates are ranked 3, 2, and 1 in that order. If the seats (array indices) for this bench are 1, 2, and 3, sorting the bench can be accomplished by swapping B_2 and B_3, then swapping B_1 and B_2, and finally swapping B_2 and B_3. If a pair is identified in a swap map by indicating the seat of the first element of the pair to be swapped, then this sorting process would be characterized with the swap map 2 1 2.

It is instructive to note that there may be many ways in which swapping of adjacent seats can be used to sort a bench. The previous bench, containing pirates with ranks 3 2 1, could also be sorted by swapping B_1 and B_2, then swapping B_2 and B_3, and finally swapping B_1 and B_2 again. The swap map that describes this sorting sequence is 1 2 1.

For a given bench, how many different swap maps exist? A little thought will show that there are an infinite number of swap maps, since sequential swapping of an arbitrary pair of elements will not change the order of the elements. Thus the swap map 1 1 1 2 1 will also leave our pirates in ascending order. But how many swap maps of minimum size will place a given bench in order? That is the question you are to answer in this problem.

Input

The input data will contain an arbitrary number of test cases, followed by a single 0. Each test case will be on a single line and will have an integer N that gives the size of an array, and will be followed by the N integer values in the array. In no test case will N be larger than 5. (If you have more than 5 pirates, the bench breaks).

Output

For each test case, print the number of swap maps in the format seen in the sample output.

Sample Input

```
2 9 7
2 12 50
3 3 2 1
3 9 1 5
0
```

Sample Output

```
There are 1 swap maps.
There are 0 swap maps.
There are 2 swap maps.
There are 1 swap maps.
```