Problem 1: Minesweeper

The Problem

Have you ever played Minesweeper? It's a cute little game which comes within a certain Operating System which name we can't really remember. Well, the goal of the game is to find where are all the mines within a MxN field. To help you, the game shows a number in a square which tells you how many mines there are adjacent to that square. For instance, suppose the following 4x4 field with 2 mines (which are represented by an * character):

```
*...
....
.*..
....
```

If we would represent the same field placing the hint numbers described above, we would end up with:

```
*100
2210
1*10
1110
```

As you may have already noticed, each square may have at most 8 adjacent squares.

The Input

The input will consist of an arbitrary number of fields. The first line of each field contains two integers n and m ($0 < n,m <= 100$) which stands for the number of lines and columns of the field respectively. The next n lines contains exactly m characters and represent the field. Each safe square is represented by an "." character (without the quotes) and each mine square is represented by an "*" character (also without the quotes). The first field line where n = m = 0 represents the end of input and should not be processed.

The Output

For each field, you must print the following message in a line alone:

```
Field #x:
```

Where x stands for the number of the field (starting from 1). The next n lines should contain the field with the "." characters replaced by the number of adjacent mines to that square. There must be an empty line between field outputs.

Sample Input

```
4 4
*...
....
.*..
....
3 5
**...
.....
.*...
0 0
```

Sample Output

```
Field #1:
*100
2210
1*10
1110

Field #2:
**100
33200
1*100
```

Problem 2:Generate Numbers

John von Neumann suggested in 1946 a method to create a sequence of pseudo-random numbers. His idea is known as the "middle-square"-method and works as follows: We choose an initial value a_0, which has a decimal representation of length at most n. We then multiply the valuea_0 by itself, add leading zeros until we get a decimal representation of length $2 \times n$ and take the middle n digits to form a_i. This process is repeated for each a_i with $i>0$. In this problem we use $n = 4$.

Example 1: a_0=5555, a_0^2=30858025, a_1=8580,...

Example 2: a_0=1111, a_0^2=01234321, a_1=2343,...

Unfortunately, this random number generator is not very good. When started with an initial value it does not produce all other numbers with the same number of digits.

Your task is to check for a given initial value a_0 how many different numbers are produced.

The Input

The input contains several test cases. Each test case consists of one line containing a_0 ($0 < a_0 < 10000$). Numbers are possibly padded with leading zeros such that each number consists of exactly 4 digits. The input is terminated with a line containing the value 0.

The Output

For each test case, print a line containing the number of different values a_i produced by this random number generator when started with the given value a_0. Note that a_0 should also be counted.

Sample Input

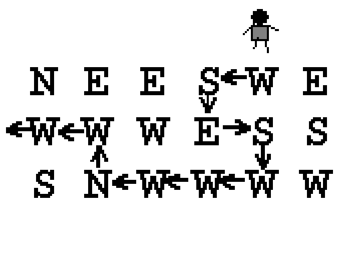
```
5555
0815
6239
0
```

Sample Output

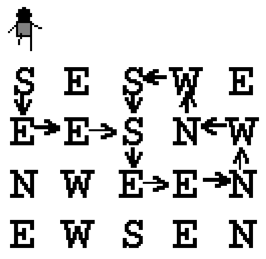
```
32
17
111
```

Note that the third test case has the maximum number of different values among all possible inputs.

Problem 3:Robot Motion Control



Grid 1



Grid 2

A robot has been programmed to follow the instructions in its path. Instructions for the next direction the robot is to move are laid down in a grid. The possible instructions are

`N` north (up the page)
`S` south (down the page)
`E` east (to the right on the page)
`W` west (to the left on the page)

For example, suppose the robot starts on the north (top) side of Grid 1 and starts south (down). The path the robot follows is shown. The robot goes through 10 instructions in the grid before leaving the grid.

Compare what happens in Grid 2: the robot goes through 3 instructions only once, and then starts a loop through 8 instructions, and never exits.

You are to write a program that determines how long it takes a robot to get out of the grid or how the robot loops around.

There will be one or more grids for robots to navigate. The data for each is in the following form. On the first line are three integers separated by blanks: the number of rows in the grid, the number of columns in the grid, and the number of the column in which the robot enters from the north. The possible entry columns are numbered starting with one at the left. Then come the rows of the direction instructions. Each grid will have at least one and at most 10 rows and columns of instructions. The lines of instructions contain only the characters `N`, `S`, `E`, or `W` with no blanks. The end of input is indicated by a row containing `0 0 0`.

For each grid in the input there is one line of output. Either the robot follows a certain number of instructions and exits the grid on any one the four sides or else the robot follows the instructions on a certain number of locations once, and then the instructions on some number of locations repeatedly. The sample input below corresponds to the two grids above and illustrates the two forms of output. The word "step" is always immediately followed by "(s)" whether or not the number before it is 1.

Sample Input

```
3 6 5
NEESWE
WWWESS
SNWWWW
4 5 1
SESWE
EESNW
NWEEN
EWSEN
0 0 0
```

Sample Output

```
10 step(s) to exit
3 step(s) before a loop of 8 step(s)
```

Problem 4:Bicoloring

In 1976 the ``Four Color Map Theorem" was proven with the assistance of a computer. This theorem states that every map can be colored using only four colors, in such a way that no region is colored using the same color as a neighbor region.

Here you are asked to solve a simpler similar problem. You have to decide whether a given arbitrary connected graph can be bicolored. That is, if one can assign colors (from a palette of two) to the nodes in such a way that no two adjacent nodes have the same color. To simplify the problem you can assume:

- No node will have an edge to itself.
- The graph is nondirected. That is, if a node *a* is said to be connected to a node *b*, then you must assume that *b* is connected to *a*.
- The graph will be strongly connected. That is, there will be at least one path from any node to any other node.

Input

The input consists of several test cases. Each test case starts with a line containing the number *n* ($1 < n < 200$) of different nodes. The second line contains the number of edges *l*. After this, *l* lines will follow, each containing two numbers that specify an edge between the two nodes that they represent. A node in the graph will be labeled using a number *a* ($0 \leq a < n$).

An input with $n = 0$ will mark the end of the input and is not to be processed.

Output

You have to decide whether the input graph can be bicolored or not, and print it as shown below.

Sample Input

```
3
3
0 1
1 2
2 0
9
8
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0
```

Sample Output

```
NOT BICOLORABLE.
BICOLORABLE.
```

Problem 5:Monitoring the Amazon(Graph Traversal)

Background

A network of autonomous, battery-powered, data acquisition stations has been installed to monitor the climate in the region of Amazon. An order-dispatch station can initiate transmission of instructions to the control stations so that they change their current parameters. To avoid overloading the battery, each station (including the order-dispatch station) can only transmit to two other stations. The destinataries of a station are the two closest stations. In case of draw, the first criterion is to chose the westernmost (leftmost on the map), and the second criterion is to chose the southernmost (lowest on the map).

The Problem

You are commissioned by Amazon State Government to write a program that decides if, given the localization of each station, messages can reach all stations.

Input

The input consists of an integer *N*, followed by *N* pairs of integers *Xi*, *Yi*, indicating the localization coordinates of each station. The first pair of coordinates determines the position of the order-dispatch station, while the remaining *N-1* pairs are the coordinates of the other stations. The following constraints are imposed: $-20 \leq Xi, Yi \leq 20$, and $1 \leq N \leq 1000$. The input is terminated with $N = 0$.

Output

For each given expression, the output will echo a line with the indicating if all stations can be reached or not (see sample output for the exact format).

Sample Input

```
4
1 0 0 1 -1 0 0 -1
8
1 0 1 1 0 1 -1 1 -1 0 -1 -1 0 -1 1 -1
6
0 3 0 4 1 3 -1 3 -1 -4 -2 -5
0
```

Sample Output

```
All stations are reachable.
All stations are reachable.
There are stations that are unreachable.
```

Problem 6:Another Rock-Paper-Scissors Problem

Sonny uses a very peculiar pattern when it comes to playing rock-paper-scissors. He likes to vary his moves so that his opponent can't beat him with his own strategy.
Sonny will play rock (R) on his first game, followed by paper (P) and scissors (S) for his second and third games, respectively. But what if someone else is using the same strategy? To thwart those opponents, he'll then play paper to beat rock, scissors to beat paper, and rock to beat scissors, in that order, for his 4th , through 6th games. After that, he'll play scissors, rock, and paper for games 7–9 to beat anyone copying his last set of moves. Now we're back to the original order—rock, paper, scissors—but instead of being predictable and using the same moves, do you know what would be better? You guessed it! Sonny then plays the sequence of moves that would beat anyone trying to copy his whole strategy from his first move, and on it goes...
To recap, in symbolic form, Sonny's rock-paper-scissors moves look like this:

R P S PSR SRP PSRSRPRPS SRPRPSPSR PSRSRPRPSSRPRPSPSRRPSPSRSRP ...

The spaces are present only to help show Sonny's playing pattern and do not alter what move he'll play on a certain game. Naturally, your job is to beat Sonny at his own game! If you know the number of the game that you'll be playing against Sonny, can you figure out what move you would need to play in order to beat him?

Input

Each line of the input contains a single integer N, 1 < N < 10^12, the number of the game you'll be playing against Sonny. An integer N = 1 indicates it would be Sonny's first game, N = 7 indicates it would be the 7th game, and so forth. The input terminates with a line with N = 0.

For example:

```
1
7
33
0
```

Warning: N may be large enough to overflow a 32-bit integer, so be sure to use a larger data type (i.e. long in Java or long in C/C++) in your program.

Output

For each test case, output a single line which contains the letter corresponding to the move you would need to play to beat Sonny on that game. For example, the correct output for the sample input above would be:

```
P
R
S
```

Problem 7: City Driving

You recently started frequenting Mumbai in your free time and realized that driving in the city is a huge pain. There are only N locations in the city that interest you, though, and you have decided to try to improve your driving experience. Since you lack a GPS and cannot remember too many different routes, you wrote down the directions and how long it takes to get between N different pairs of locations (the same in both directions), ensuring that using only these paths you can get from any location to any other one.
Now you are planning your trip for the weekend and you need to figure out the fastest way to get between Q pairs of locations in the city using only the routes you have written down.

Input

The input consists of multiple test cases. The first line contains a single integer N, 3<N<100,000, the number of locations of interest and the number of routes you wrote down. The next N lines each contain three integers u, v, and w (1 < w < 1,000), indicating that you have directions from location u to location v and vice-versa (0-indexed) which take w time.
The following line contains a single integer Q, 1 < Q < 10,000, the number of pairs of locations you need to compute the travel time for. The next Q lines each contains two integers u and v, indicating that you should find the minimum time to get from location u to location v. The input terminates with a line with N = 0.
For example:

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

Output

For each test case, print out Q lines, one for each pair of locations u and v you are finding the fastest routes
For each line should simply contain the minimum time it takes to travel from location u to location v. For example, the correct output for the sample input above would be:

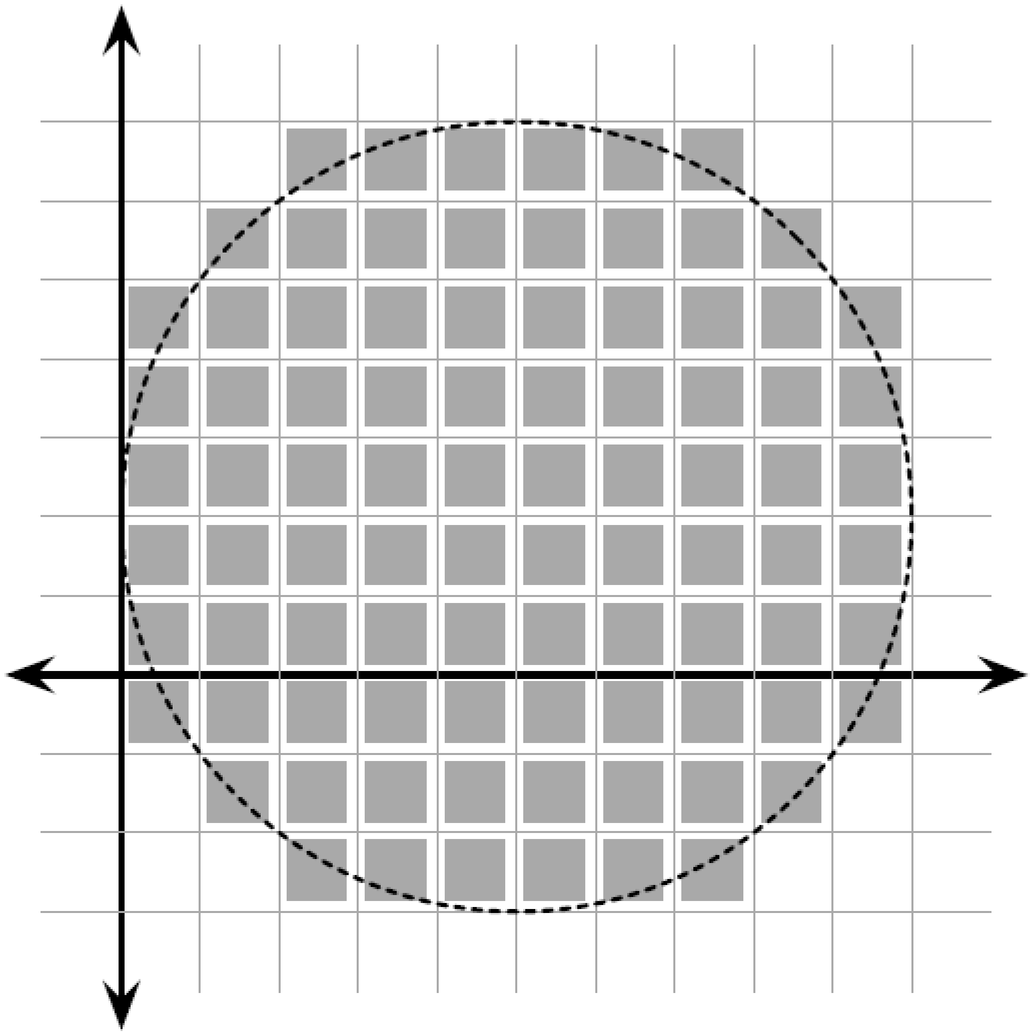
```
11
9
5
```

Problem 8: Counting Pixels

Description

Did you know that if you draw a circle that fills the screen on your 1080p high definition display, almost a million pixels are lit? That's a lot of pixels! But do you know exactly how many pixels are lit? Let's find out!

Assume that our display is set on a Cartesian grid where every pixel is a perfect unit square. For example, one pixel occupies the area of a square with corners (0, 0) and (1, 1). A circle can be drawn by specifying its center in grid coordinates and its radius. On our display, a pixel is lit if any part of is covered by the circle being drawn; pixels whose edge or corner are just touched by the circle, however, are not lit.



Your job is to compute the exact number of pixels that are lit when a circle with a given position and radius is drawn.

Input

The input consists of several test cases, each on a separate line. Each test case consists of three integers, x, y, and r ($1 < x; y; r < 10^6$), specifying respectively the center (x,y) and radius of the circle drawn. Input is followed by a single line with x = y = r = 0, which should not be processed.
For example:

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

Output

For each test case, output on a single line the number of pixels that are lit when the specified circle is drawn.
Assume that the entire circle will fit within the area of the display.
For example:

```
4
88
```

Problem 9: Product-sum numbers

A natural number, N, that can be written as the sum and product of a given set of at least two natural numbers, $\{a_1, a_2, \ldots, a_k\}$ is called a product-sum number: $N = a_1 + a_2 + \ldots + a_k = a_1 \times a_2 \times \ldots \times a_k$.

For example, $6 = 1 + 2 + 3 = 1 \times 2 \times 3$.

For a given set of size, k, we shall call the smallest N with this property a minimal product-sum number. The minimal product-sum numbers for sets of size, k = 2, 3, 4, 5, and 6 are as follows.

k=2: $4 = 2 \times 2 = 2 + 2$
k=3: $6 = 1 \times 2 \times 3 = 1 + 2 + 3$
k=4: $8 = 1 \times 1 \times 2 \times 4 = 1 + 1 + 2 + 4$
k=5: $8 = 1 \times 1 \times 2 \times 2 \times 2 = 1 + 1 + 2 + 2 + 2$
k=6: $12 = 1 \times 1 \times 1 \times 1 \times 2 \times 6 = 1 + 1 + 1 + 1 + 2 + 6$

Hence for 2≤k≤6, the sum of all the minimal product-sum numbers is 4+6+8+12 = 30; note that 8 is only counted once in the sum.

In fact, as the complete set of minimal product-sum numbers for 2≤k≤12 is {4, 6, 8, 12, 15, 16}, the sum is 61.

Problem: What is the sum of all the minimal product-sum numbers for 2≤*k* ≤12000?

Input: Nothing

Output: A single number giving the answer of the Problem

Problem 10: Edit distance

Given a string, an edit script is a set of instructions to turn it into another string. There are four kinds of instructions in an edit script:

• Add (‘a’): Output one character. This instruction does not consume any characters
from the source string.
• Delete (‘d’): Delete one character. That is, consume one character from the source
string and output nothing.
• Modify (‘m’): Modify one character. That is, consume one character from the source
string and output a different character.
• Copy (‘c’): Copy one character. That is, consume one character from the source string
and output the same character.

A shortest edit script is an edit script that minimizes the total number of adds, deletes,
and modifies. Given two strings, generate a shortest edit script that changes the first into the second.

Input

The input consists of two strings on separate lines. The strings contain only alphanumeric characters. Each string has length between 1 and 17000, inclusive.

Output

The output is a shortest edit script. Each line is one instruction, given by the one-letter
code of the instruction (a, d, m, or c), followed by a space, followed by the character written
(or deleted if the instruction is a deletion).
In case of a tie, you may generate any shortest edit script.

Example

Input

```
abcde
xabzdey
```

Output

```
a    x
c    a
c    b
m    z
c    d
c    e
a    y
```