XXXII Maratón Nacional de Programación
Colombia - ACIS / REDIS 2018
ACM ICPC

Problems

(This set contains 11 problems; problem pages numbered from 1 to 20)

General Information. Unless otherwise stated, the conditions stated below hold for all the problems. However, some problems might have specific requirements and it is important to read the problem statements carefully.

Program name. Each source file (your solution!) must be called

`<codename>.c`, `<codename>.cpp`, `<codename>.java`, or`<codename>.py`

as instructed below each problem title.

Input.

1. The input must be read from the standard input.
2. In most problems, the input contains several test cases. Each test case is described using a number of lines that depends on the problem.
3. In most cases, when a line of input contains several values, they are separated by single spaces. No other spaces appear in the input. There are no empty lines.
4. Every line, including the last one, has the usual end-of-line mark.
5. If no end condition is given, then the end of input is indicated by the end of the input stream. There is no extra data after the test cases in the input.

Output.

1. The output must be written to the standard output.
2. The result of each test case must appear in the output using a number of lines, which depends on the problem.
3. When a line of results contains several values, they must be separated by single spaces. No other spaces should appear in the output. There should be no empty lines.
4. Every line, including the last one, must have the usual end-of-line mark.
5. After the output of all test cases, no extra data must be written to the output.
6. To output real numbers, if no particular instructions are given, round them to the closest rational with the required number of digits after the decimal point. Ties are resolved rounding to the nearest upper value.

A: All-star Three-point Contest

Source file name: `all.c`, `all.cpp`, `all.java`, *or* `all.py`
Author: Rafael Niquefa

The three-point shooting contest is an important and exciting event in any all-star weekend of any basketball league. In this contest, each participant takes 25 shots, from 5 different spots in the court. Each shot scored gives one point to the player, except for the last shot in each spot which gives two points if scored.

In this problem, you must write a program that takes as input the information of a three point shooting contest and sorts the players by the points scored in descending order. In the event of a tie, the players must be lexicographically sorted.

Input

The input consist of several test cases. Each test case consist of a line with an integer P representing the number of players ($0 < P < 101$). Each of the next P lines is formatted as follows:

```
Player name;# # # # #;# # # # #;# # # # #;# # # # #;# # # # #
```

The `# # # # #` represent the five shots at each spot; each `#` is either 1 if the shot was scored or 0 if it was missed. The player name is a non-empty sequence of at most 20 symbols, only including letters from the English alphabet and spaces. You can assume that no leading or trailing spaces occur in a name, and no two players have the same name.

The input must be read from standard input.

Output

For each test case output $P + 1$ lines. The first of those lines is the case number in the format ''`Case` N:'' where N is the case number starting at 1. Each of the next P lines lines contain the name of a player (as it appear in the input) and the total number of points scored in the contest. The list of names must be sorted in descending order relative to the points scored. If there is a tie, then sort the names in case-insensitive lexicographical ascending order. See the sample output for details on the format.

The output must be written to standard output.

Sample Input

```
3
Michael Jordan;0 1 1 0 1;0 1 1 0 1;0 1 1 0 1;0 0 0 0 1;0 0 0 0 1
Scotty Pippen;0 1 1 0 1;0 1 1 0 1;0 1 1 0 1;0 0 0 0 1;0 0 0 1 1
Charles Barkley;0 1 1 0 1;0 1 1 0 1;0 1 1 0 1;0 0 0 0 1;0 0 1 1 1
3
Michael;0 1 1 0 1;0 1 1 0 1;0 1 1 0 1;0 0 0 0 1;1 1 1 1 1
Scotty;0 1 1 0 1;0 1 1 0 1;0 1 1 0 1;0 0 0 0 1;1 1 1 1 1
Charles;0 1 1 0 1;0 1 1 0 1;0 1 1 0 1;0 0 0 0 1;1 1 1 1 1
3
Charles Barkley;0 1 1 0 1;0 1 1 0 1;0 1 1 0 1;0 0 0 0 1;1 1 1 1 1
Charle s Barkley;0 1 1 0 1;0 1 1 0 1;0 1 1 0 1;0 0 0 0 1;1 1 1 1 1
Charl es Barkley;0 1 1 0 1;0 1 1 0 1;0 1 1 0 1;0 0 0 0 1;1 1 1 1 1
```

Sample Output

```
Case 1:
Charles Barkley 18
Scotty Pippen 17
Michael Jordan 16
Case 2:
Charles 20
Michael 20
Scotty 20
Case 3:
Charl es Barkley 20
Charle s Barkley 20
Charles Barkley 20
```

B: Forming Better Groups

Source file name: `better.c`, `better.cpp`, `better.java`, *or* `better.py`
Author: Edwin Niño

Prof. Smith is tired of the same old end of semester story: lazy students passing his course thanks to group assignments. Unfortunately, group assignments are an important part of the university policies and Prof. Smith cannot get ride of them. However, he has figured out a strategy that can limit the inclusion of lazy students as members of groups with hardworking students. He wants the lazy students to either work or fail his course.

The crux of the idea is to define a *threshold D* so that the following condition limiting how students participate in groups of *three* members is satisfied:

> if G_1, G_2, and G_3 are the grades obtained in the previous assignment by three students willing to form a group, then
>
> $$\max(G_1, G_2, G_3) - \min(G_1, G_2, G_3) \leq D.$$

Prof. Smith wants students to have many options to form groups and choose their teammates. Therefore, he would like to know in advance the number of ways his students could form groups of three members given threshold D and their grades in the previous assignment, while satisfying the condition above.

Please write a program to help Prof. Smith.

Input

The input has several test cases. The first line of a test case consists of two blank-separated integers N ($3 \leq N \leq 21$) and D ($0 \leq D \leq 500$), representing the number of students and the threshold, respectively. You can assume that N is a multiple of 3. The second line consists of a blank-separated list of grades $G_1, G_2, \ldots, G_N$ ($0 \leq G_i \leq 500$, for $1 \leq i \leq N$) corresponding to the grades of the N students in the previous assignment. The input ends with a line containig two blank-separated zeroes.

The input must be read from standard input.

Output

For each test case, output one line containing one integer: the number of ways the N students can form groups of three members given the threshold D and the grades $G_1, G_2, \ldots, G_N$ in their previous assignment when the above-mentioned condition is enforced.

The output must be written to standard output.

Sample Input	Sample Output
6 3 1 6 3 2 4 5 6 8 1 3 3 3 6 5 9 2 1 2 3 4 5 6 7 8 10 0 0	2 10 0

C: Carrol's Scrabble

Source file name: `carrol.c`, `carrol.cpp`, `carrol.java`, *or* `carrol.py`
Author: Mario Sánchez

In the Christmas Eve of 1877, Lewis Carroll invented a game called *Word Links* (it was later renamed to *Doublets* when Vanity Fair published it in 1879). In Word Links, a player is given a challenge consisting of two words with the goal of finding a sequence of valid English words that starts from the first word and ends with the second word. Two words can be linked in such a sequence if and only if they have the same number of letters and they either differ by a single letter or are anagrams of each other.

For example, the following is a valid sequence from `iron` to `lead` with length 7:

`iron`	
`icon`	(replace `c` for `r` in `iron` to get `icon`)
`coin`	(reorganize the letters in `icon` to obtain `coin`)
`corn`	(replace `r` for `i` in `coin` to get `corn`)
`cord`	(replace `d` for `n` in `corn` to get `cord`)
`lord`	(replace `l` for `c` in `cord` to get `lord`)
`load`	(replace `a` for `r` in `lord` to get `load`)
`lead`	(replace `e` for `o` in `load` to get `lead`).

In Carroll's version of the game, the solution of a challenge is the shortest sequence between the two given words. It is possible to have multiple solutions, namely, different sequences of the same length can have the same initial and final words.

With the idea of Scrabble in mind, the World Links game can become Carroll's Scrabble and be more interesting. A solution to a Carroll's Scrabble challenge is a shortest sequence that maximizes the sum of the values of the words in it (when ignoring the two words given in the challenge). The value of each word is obtained by summing the value of its letters by the following rules (capitalization is ignored):

- Letters with 1 point: e, a, i, o, n, r, t, l, s, u.
- Letters with 2 points: d, g.
- Letters with 3 points: b, c, m, p.
- Letters with 4 points: f, h, v, w, y.
- Letters with 5 points: k.
- Letters with 8 points: j, x.
- Letters with 10 points: q, z.

For example, the total value of the previous sequence from `iron` to `lead` is 35: 6 points for `icon`, 6 points for `coin`, 6 points for `corn`, 7 points for `cord`, 5 points for `lord`, and 5 points for `load`. Note that the values of `iron` and `lead` have been omitted.

Given a dictionary of valid words that can be used in Carroll's Scrabble, you are asked to compute the value of optimal solutions to several challenges of the game.

Input

The input provides a dictionary and challenges to solve. The input begins with a line containing an integer N ($0 < N < 10\,000$) representing the number of words in the dictionary; you can assume that no word is repeated in the dictionary. Each of the next N lines provides a word available from the dictionary: each line has a single non-empty word with at most 20 lower case letters. The next line contains an integer Q ($0 < Q < 200$) representing the number of challenges to solve. Each of the next Q lines follows the pattern

$$word_1 \ \texttt{TO} \ word_2$$

where $word_1$ and $word_2$ are words in the dictionary.

The input must be read from standard input.

Output

Output a single line for each challenge. If the challenge has a solution, use the format

$$word_1 \ \texttt{TO} \ word_2 \ \ NS \ \ Val$$

where $word_1$ and $word_2$ are the words in the given challenge, NS is the minimum number of steps required to go from $word_1$ to $word_2$, and Val is the maximum value of the words in such an optimal sequence. Remember that the values of $word_1$ and $word_2$ must not be included in the total value Val of the sequence. If the challenge does not have a solution, use the format

$$word_1 \ \texttt{TO} \ word_2 \ \texttt{IMPOSSIBLE}$$

The output must be written to standard output.

Sample Input	**Sample Output**
14 hot iron icon coin corn cord lord load lead lion cold gold worm warm 5 iron TO lead lead TO gold iron TO icon warm TO hot warm TO cold	iron TO lead 7 35 lead TO gold 5 24 iron TO icon 1 0 warm TO hot IMPOSSIBLE warm TO cold IMPOSSIBLE

D: Dominoes Magic Squares

Source file name: `dominoes.c`, `dominoes.cpp`, `dominoes.java`, *or* `dominoes.py`
Author: Rodrigo Cardoso

A *domino set* is a collection of tiles of the form

$$[a \mid b]$$

with integer labels a and b satisfying $0 \leq a, b \leq 6$. Both $[a \mid b]$ and $[b \mid a]$ are descriptions of the same domino tile. A complete domino set has exactly 28 tiles and the sum of all its labels is 168.

A *magic square* is a square of integer numbers whose rows, columns, and diagonals have the same sum. Since domino tiles can be seen as planar objects of 2 unit squares, they can be used to build magic squares. For instance, the set of domino tiles

$$[1 \mid 4], [5 \mid 2], [4 \mid 4], [2 \mid 3], [5 \mid 4], [5 \mid 3], [1 \mid 3], [3 \mid 3]$$

can be arranged into a magic square of side 4 units with rows, columns, and diagonals adding up to 13:

4	4	2	3
3	3	2	5
1	3	5	4
5	3	4	1

However, it is impossible to build a 4×4 magic square with the following set of titles adding up to 15 in rows, columns, and diagonals:

$$[6 \mid 5], [2 \mid 4], [2 \mid 2], [5 \mid 5], [5 \mid 4], [5 \mid 1], [2 \mid 3], [3 \mid 6].$$

Assume you are given 8 domino tiles: can you arrange them into a 4×4 magic square?

Input

The input consists of several test cases. A test case comprises 8 consecutive lines of input, each one containing two blank-separated integers a and b, $0 \leq a, b \leq 6$, representing the tile $[a \mid b]$. You can assume that a test case does not contain repeated dominoes.

The input must be read from standard input.

Output

For each test case, output one line with the unique character '`Y`' if a magic square can be built with the given domino tiles and '`N`' otherwise.

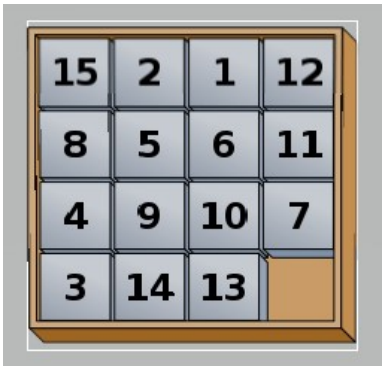
The output must be written to standard output.

Sample Input	Sample Output
1 4 5 2 4 4 2 3 5 4 5 3 1 3 3 3 6 5 2 4 2 2 5 4 5 5 5 1 2 3 3 6	Y N

E: Extended Puzzle

Source file name: `extended.c`, `extended.cpp`, `extended.java`, *or* `extended.py`
Author: Rodrigo Cardoso

The $m \times n$-*puzzle*, as explained in Wikipedia and Wolfram MathWorld, is a sliding puzzle with a frame of m rows and n columns, and a total of mn squared tiles. Each tile is assigned a unique integer number from 1 to mn and it is randomly placed in the frame. The tile assigned mn is removed from the frame, so that the neighboring tiles (horizontally and vertically) may slide into the empty square generating another puzzle configuration. Such a transformation is called a *move*. The figure below depicts a 4×4-puzzle where the tiles assigned 7 and 13 are the only possible moves:



A solution to an $m \times n$-puzzle is a sequence of moves that ends in a configuration with the tiles arranged in ascending order (i.e., $1, 2, ..., mn$) when the rows are considered as a large array; the missing mn tile is assumed to be the last one. In this way, solving an $m \times n$-puzzle can be considered as a sorting process starting from the initial configuration, where each configuration can be represented as a permutation of $1, 2, \ldots, mn$. Recall that a permutation may be classified either as *even* or *odd* depending on the *parity* of the number of inversions it contains: an *inversion* is a pair of elements in the permutation that are out of its usual order. In the picture above, for example, the tiles 8 and 10 are in the usual order, but 15 and 9 are out of the usual order.

A parity argument may be used to show that only half of the starting positions in the $m \times n$-puzzle are possible to solve (no matter how many moves are made). This is done by considering a function of the tile configuration invariant under any valid move, so that it partitions the space of all configurations into either *reachable* and *unreachable*. One such a function is the parity of the initial permutation plus the parity of the Manhattan distance (or ‘‘taxicab’’ distance) from the mn tile to its rightful position (i.e, from the empty square to the last column of the last row in the frame). Recall that the Manhattan distance is the minimum number of horizontal and vertical unit segments required to go from the starting square to the final one. In the above figure, the Manhattan distance from the 4 tile to the last column of the last row in the frame is 4, while it is 0 for the mn (i.e., the empty square) to the to the last column of the last row. This function is invariant because each move changes both the parity of the permutation and the parity of the taxicab distance. Note that, in particular, if the empty square is in the lower right corner then the puzzle is solvable if and only if the permutation of the remaining pieces is even. Although it is not so easy to see, the converse of the former claim is also true: a given configuration can be solved if and only if it represents a permutation whose parity plus that of the taxicab distance from the empty square to the lower right corner is even.

A very famous instance of the game is the 4×4-puzzle proposed by Sam Loyd in the 19th century in which each tile is placed correctly, except for the 14 and 15 ones that are swapped. Besides falsely claiming the game invention, he offered a $1,000 prize for anyone who could provide a solution for that particular instance of

the game: of course, nobody won the prize because there is not such a solution since the initial configuration represents an odd permutation.

Your task is to establish if some given configurations of $m \times n$-puzzles are solvable or not.

Input

The input consists of several test cases. Each test case begins with one line containing two blank-separated positive integers m and n, $1 < m \cdot n \leq 100\,000$, representing the number of rows and columns of an $m \times n$-puzzle, respectively. Then m lines follow, each containing n numbers. The mn numbers listed in the m rows are a permutation of the integer numbers $1, 2, ..., mn$ and represent an initial configuration of an $m \times n$-puzzle.

The input must be read from standard input.

Output

For each test case, output one line with '`Y`' if the given configuration has a solution and '`N`' otherwise.

The output must be written to standard output.

Sample Input	**Sample Output**

`4 4`
`1 2 3 4`
`5 6 7 8`
`9 10 11 12`
`13 15 14 16`
`2 2`
`4 3`
`2 1`
`2 3`
`4 1 3`
`6 2 5`
`2 3`
`4 1 3`
`6 5 2`
	`N`
	`Y`
	`Y`
	`N`

F: A Fibonacci Family Formula

Source file name: `family.c`, `family.cpp`, `family.java`, *or* `family.py`
Author: Rafael García

Everybody knows about Leonardo Fibonacci, the inventor of the famous sequence where the first two terms are 1 and from then on every term is calculated as the sum of the previous two terms.

There is a tradition in the Fibonacci family since the 12th century: when a member is about to become 21 years of age, the family formulates a new sequence in which an extra summation term is added to the last version of the sequence. Then, during the birthday party, everybody has fun asking the new adult random terms in the newly designed sequence.

Of course, the sequence that started this tradition is $1, 1, 1, 1, 1, 1, \ldots$. The second sequence, and most famous one, is $1, 1, 2, 3, 5, 8, \ldots$. The third formulated sequence is $1, 1, 2, 4, 7, 13, \ldots$. And the fourth one is $1, 1, 2, 4, 8, 15, \ldots$.

Your friend Leonardo is 20 and will be the kth family member to celebrate a birthday under this tradition. He is very stressed because he has not found enough time to study the sequence that will be designed for his birthday. However, he has found a paper in his father's office with the following equation:

$$f_n^{(k)} = \begin{cases} 0 & \text{, if } n < 0 \\ 1 & \text{, if } n = 0 \\ f_{n-1}^{(k)} + f_{n-2}^{(k)} + \cdots + f_{n-k}^{(k)} & \text{, if } n \geq 1. \end{cases}$$

Leonardo is certain that this equation describes the nth term in the kth sequence of the tradition, but he is not sure how to quickly calculate such terms. Your task is to help Leonardo by writing a program he can use during his birthday party: when asked for a term, he wishes to answer swiftly.

Input

The input consists of several test cases. A case consists of a line containing two blank-separated integers k and n with $1 \leq k \leq 100$ and $0 \leq n \leq 10^{15}$. The input ends with two blank-separated zeroes.

The input must be read from standard input.

Output

For each test case, output one line with the nth term in the kth sequence, namely, with the value of $f_n^{(k)}$. Since the terms can become very big, your program should calculate the results modulo 1 000 000 009.

The output must be written to standard output.

Sample Input	**Sample Output**
5 5 3 4 2 3 7 0 0 0	16 7 3 1

G: Tron Garbage Collector

Source file name: `garbage.c`, `garbage.cpp`, `garbage.java`, *or* `garbage.py`
Author: Rafael García

According to Wikipedia, a *garbage collector* in computer science is

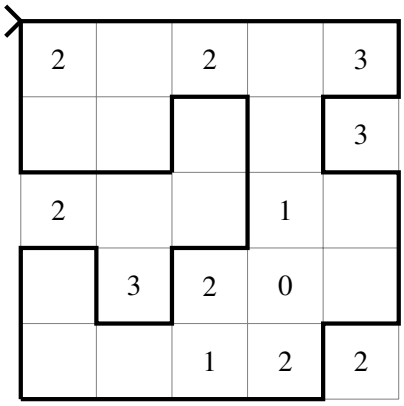
> ... a form of automatic memory management. It attempts to reclaim garbage or memory occupied by objects that are no longer in use by the program.

How does a memory garbage collector work? Think of the main memory as the Squareland: a large grid with dimensions R and C where the lines represent streets and the boxes are labeled with the number of data to be collected. A car is driven through the streets to remove the data adjacent to each street segment it visits. More precisely, a number in a box identifies the number of times such a box must be visited by the car in order to collect the garbage (i.e., the number of times an adjacent street segment needs to be visited by the car). If such a number is 0, then the box can not be visited. The labels 1, 2, 3, and 4 indicate that a box must be visited 1, 2, 3, or 4 times, respectively. Otherwise, the box can be visited any number of times. However, there are additional restrictions: the car route must start and end at the same street intersection and it cannot visit a corner (except for the first one) more than once.

For example, consider the following 5×5 grid representing Squareland:

2		2		3
				3
2			1	
	3	2	0	
		1	2	2

The following is a possible route for the garbage collecting car:



In order to design the garbage collection task, you are required to write a program to determine if there exists a route that can collect all the garbage satisfying the constraints above-mentioned.

Input

The input consists of several test cases. The first line in a test case contains two blank-separated integers $0 < R, C < 6$. Each of the next R lines contains C symbols denoting the number of data values that must be eliminated from the grid:

- If the number of data values that must be eliminated is d, then the number d ($0 \leq d \leq 4$) appears.
- If the collector can visit the streets adjacent to a box but it is not required to remove anything, then the dot (`.`) appears.

The input ends with two blank-separated zeroes.

The input must be read from standard input.

Output

For each test case, output `YES` if it is possible to find a route under the given constraints and `NO` otherwise.

The output must be written to standard output.

Sample Input	**Sample Output**
5 5 2.2.3 ....3 2..1. .320. ..122 5 5 2.2.3 .4..3 2..1. .320. ..122 0 0	YES NO

H: Ghost Hunting

Source file name: `hunting.c`, `hunting.cpp`, `hunting.java`, *or* `hunting.py`
Author: Germán Sotelo

A ghost Pokemon gang has been playing pranks on citizens and damaging public property in Lavender City. Any attempt to apprehend the gang has been futile, since it is impossible to see a ghost with the naked eye.

The major of Lavender City requested help from Silph Corp. After several months of research, the corporation developed a system that allows people to see the ghost Pokemon. The system is based on beacons: when a ghost Pokemon is surrounded by them, it is visible to anyone. However, the beacons use a lot of energy and thus require to be installed in light poles already available in the city. A Pokemon is *surrounded by beacons* if, in order to leave town, it needs to cross through a virtual fence: the idea is that any pair of beacons induce a virtual fence defined by the segment of a straight line starting at one of them and ending at the other one.

The past week, Silph Corp sent a truck full of beacons to create a visibility area in Lavender City. However, the ghost gang attacked the truck and damaged most of the beacons, except for three that are still fully operational. Since the production of beacons takes time, and the situation is getting dangerous by the day, the major wants to find the largest possible area that can be set up to see the ghosts once the three beacons are installed.

Given the location of the light poles, your help has been requested to find such an area.

Input

The input consists of several test cases. Each test case begins with a line containing an integer N ($3 \leq N \leq 2 \times 10^3$) representing the number of light poles in Lavender City. Each of the following N lines contains two blank-separated integers x and y ($-10^6 < x, y < 10^6$) indicating the location (x, y) of a light pole.

The input must be read from standard input.

Output

For each test case, output a single line with the largest possible area, truncated to one decimal, where the ghosts can be seen.

The output must be written to standard output.

Sample Input	**Sample Output**
4 -1 -1 -1 1 1 -1 1 1 3 -1 -1 -1 3 3 3 4 1 1 2 1 4 1 4 6	2.0 8.0 7.5

I: Impossible Communication

Source file name: `impossible.c`, `impossible.cpp`, `impossible.java`, *or* `impossible.py`
Author: Rafael García

At your home university there are N research groups and M procedures to exchange information between them. Some procedures enable information to be sent from a specific group to another group. Other procedures are designed to share information between two or more groups (in either direction for each pair of them).

To be more precise, the research system of your university uses the following notation to identify the information sharing procedures:

- A *simple procedure* that enables information to be sent from the group I to the group J is specified as

$$1 \;\; I \;\; J$$

- A *complex procedure* that enables bi-directional information sharing between the k groups $I_1, I_2, \ldots, I_k$ is described as

$$k \;\; I_1 \;\; I_2 \;\; \cdots \;\; I_k$$

The president of your home university wants to make sure that the procedures to exchange information between research groups are *sufficiently complete*: he wants to make sure that any research group is able to send information to any other group (regardless of whether it is done directly or using intermediaries).

Please, help your president!

Input

The input consists of several test cases. The first line of a case contains two integers N and M indicating, respectively, the number of research groups ($2 \leq N \leq 50\,000$) and the number of information sharing procedures ($1 \leq M \leq 1\,000$). Each of the next M lines identifies one simple or one complex procedure, following the above-described notation. Research groups are represented with the integers $1, 2, \ldots, N$. You can assume that each complex procedure has at most 1 000 research groups. Single blanks are used to separate any pair of consecutive numbers.

The input must be read from standard input.

Output

For each test case output a single line with `YES` if the given set of procedures is sufficiently complete and `NO` otherwise.

The output must be written to standard output.

Sample Input	Sample Output
3 3 1 1 2 1 2 3 1 1 3 4 3 1 1 2 1 2 3 3 1 3 4	NO YES

J: Jawbreaking Candy

Source file name: `jawbreaking.c`, `jawbreaking.cpp`, `jawbreaking.java`, *or* `jawbreaking.py`
Author: Juan Camilo Corena

The *Advanced Cutting Machines* (*ACM*) has developed a new product for cutting rectangular candies into shorter pieces. The width of candies has been optimized already, so this machine's purpose is about optimizing the length of cuts. ACM is very excited about the new machine because it will solve the eternal discussion of how long candies should be for a given audience.

The in-house Mathematics Department of ACM determined how the machine cuts the pieces of candy. The lengths of candy that the machine can cut are those shorter than original one L and can be represented as a fraction of the form $\frac{a \cdot L}{b}$, where integers a and b have to satisfy $0 < a$ and $0 < b \leq n$. Here n represents a cutting resolution for the different models of machine that will be produced; more expensive models will have higher cutting resolution. For example, assume Alice wishes to buy a piece of candy of length at least 320 units. If this piece were to be cut from a longer piece of 500 units of length and the cutting machine can only cut fractions of the candy with a denominator at most $b = 3$, then the following lengths of candy could be cut:

$$\frac{500}{3}, \frac{500}{2}, \frac{500}{1}, \frac{1000}{3}, \frac{1000}{2}, \frac{1000}{1}, \frac{1500}{3}, \frac{1500}{2}, \frac{1500}{1}, \ldots$$

The in-house Mathematics Department predicts that, from these options, Alice would prefer the one with length $\frac{1000}{3}$ because it is the smallest one that is at least 320, as she wished, and will have less calories.

Given L, n, and a desired length of candy to cut d, compute the shortest candy length that the machine can cut that is at least d units of length presented in the form of a reduced fraction (i.e., the numerator and denominator cannot share positive factors other than 1).

Input

The input consists of several test cases, one per line. Each line contains integers $0 < L < 10^6$, $0 < n \leq 5\,000$, and $0 < d \leq L$ separated by a single space. The input ends with a line containing three blank-separated zeroes.

The input must be read from standard input.

Output

For each test case, output the shortest length that can be cut by the machine and that is at least d units of length in the form of a reduced fraction.

The output must be written to standard output.

Sample Input	**Sample Output**

`500 3 320`
`100 5 23`
`0 0 0`
	`1000/3`
	`25/1`

K: `kewl` Texting

Source file name: `kewl.c`, `kewl.cpp`, `kewl.java`, *or* `kewl.py`
Author: Ana Echavarría

Alicia and Roberto are good friends and often text each other. They would like to speed up their texting by designing a mechanism that suggests what word they may want to text next. That way, they can select the suggested word instead of having to type it directly, saving some time.

Being computer science students who just learned about language models, they came up with a solution. For each occurring word w in their past text messages, they have selected a word w' as the suggestion to be shown after typing w. The criteria for selecting w' are the following:

- The word w' is the word that comes after w the most times in the past text messages.
- If two or more words come after w the same number of times in the past texts, then w' is the word among them that appeared the most times overall in the past texts.
- If again, two or more words appeared the same number of times in the past texts, then w' is the first one of them in lexicographical order.
- They also want to suggest words that start or end a text message. Therefore, the start and end of texts (called here {*start*} and {*end*}, respectively) are also considered words. When they sort words lexicographically, {*start*} would come before {*end*} and they would both come before any other word in the texts.

For example, consider the following text messages (one per line):

1. `what a nice day`
2. `this is the nice restaurant he talked about`
3. `we want nice weather and hope it is a nice day`

There are three words that follow the word `nice`, namely, `day`, `weather`, and `restaurant`. However, `day` comes after `nice` 2 times, while `restaurant` and `weather` do so only once. Based on their model, the word suggested after typing `nice` is the word `day`. Similarly, the word `is` is followed once by both `a` and `the`, but `a` is more frequent in the texts overall (it appears 2 times while `the` appears only once); therefore, the suggestion for `is` is the word `a`. The suggestion for `day` is {*end*} since, in most cases, it is used to end a text. On the other hand, the words `what`, `this`, and `we` all come at the start of sentences and they all appear only once in the texts: hence, `this` (the first one in lexicographical order of the three) is the suggestion for {*start*}, i.e., for the beginning of a text message.

Alicia and Roberto would like to know what the sentence generated by always picking the word suggested by their language model would be. Note that such a text message either ends after a finite number of suggestions or never ends. For the example above, the sentence resulting by always picking the model's suggestion is

```
this is a nice day
```

This is because the suggested starting word is `this`, the word suggested after `this` is `is` and so on, until the word `day` is reached (whose suggestion is {*end*}, that is, the end of the text).

Your task is, for a given history of text messages between Alicia and Roberto, to compute the sentence generated by always picking the word suggested by the language model or identify if such a process never terminates.

Input

The input consist of several test cases. Each test case begins with a line containing a single integer n defining the number of text messages. Each of the next n lines represents a text message containing at least one word. Every word in the input consists only of lowercase English letters. There is no limit on the number of text messages. However, the entire set of text messages has at most 10^5 words in total and each word will have at most 20 characters.

The input must be read from standard input.

Output

For each test case, output a single line containing the sentence that would result by always selecting the word suggested by the language model or `INFINITE` if this method would result in an infinite text message. Use a single blank to separate each pair of consecutive words in the output.

The output must be written to standard output.

Sample Input	**Sample Output**
3 what a nice day this is the nice restaurant he talked about we want nice weather and hope it is a nice day 3 a rabbit is white the house is big and tall we have a big house with a door 1 a rose is a rose	this is a nice day INFINITE a rose