Problem A. Periodic Points

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

Computing the number of fixed points and, more generally, the number of periodic orbits within a dynamical system is a question attracting interest from different fields of research. However, dynamics may turn out to be very complicated to describe, even in seemingly simple models. In this task you will be asked to compute the number of periodic points of period n of a piecewise linear map f mapping the real interval $[0; m]$ into itself. That is to say, given a map $f : [0; m] \to [0; m]$, you have to calculate the number of solutions to the equation $f^n(x) = x$ for $x \in [0; m]$, where f^n is the result of iterating function f for n times, i.e.

$$f^n = \overbrace{f \circ \ldots \circ f \circ f}^{nf's},$$

where $\circ$ stands for composition of maps: $(g \circ h)(x) = g(h(x))$.

Fortunately, the maps you will have to work with satisfy some particular properties:

- m will be a positive integer and the image of every integer in $[0; m]$ under f is again an integer in $[0; m]$, that is, for every $k \in \{0, 1, \ldots, m\}$ we have that $f(k) \in \{0, 1, \ldots, m\}$.
- for every $k \in \{0, 1, \ldots, m-1\}$, the map f is linear in the interval $[k; k+1]$. This means that for every $x \in [k; k+1]$, its image satisfies $f(x) = (k+1-x)f(k) + (x-k)f(k+1)$, which is equivalent to its graph on $[k; k+1]$ being a straight line segment.

Since there might be many periodic points you will have to output the result modulo an integer.

Input

The input consists of several test cases, separated by single blank lines. Each test case begins with a line containing the integer m ($1 \le m \le 80$). The following line describes the map f, it contains $m+1$ integers $f(0), f(1), \ldots, f(m)$, each of them between 0 and m, inclusive. The test case ends with a line containing two integers separated by a blank space, n ($1 \le n \le 5\,000$) and the modulus used to compute the result, mod ($2 \le mod \le 10\,000$).

The input will finish with a line containing 0.

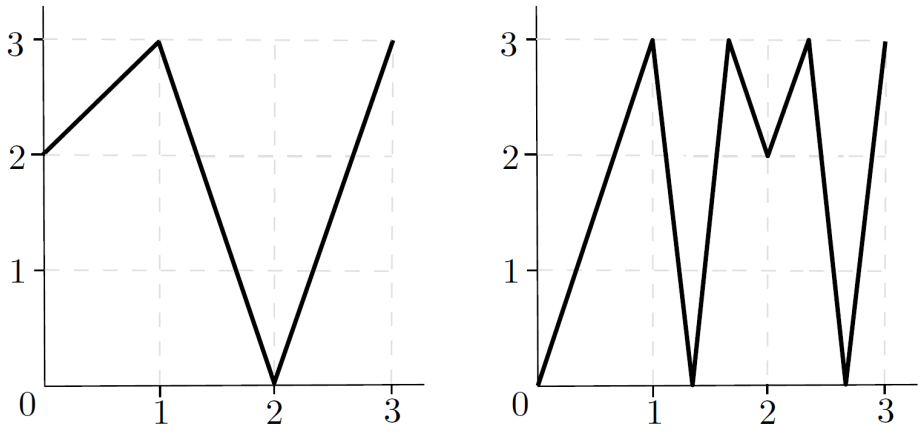
Output

For each case, your program should output the number of solutions to the equation $f^n(x) = x$ in the interval $[0; m]$ modulo mod. If there are infinitely many solutions, print `Infinity` instead.

Example

standard input	standard output
2 2 0 2 2 10 3 0 1 3 2 1 137 3 2 3 0 3 20 10000 0	4 Infinity 9074

The following picture shows the graphs of the third map in the sample input, f_3 (left), and of its square, f_3^2 (right).



Problem B. Palindromic DNA

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

A DNA sequence is composed of a series of four possible nucleobases, namely Adenine, Guanine, Thymine and Cytosine; we will refer to each of these bases by their initial. For our purposes, nucleobases have an associated cyclic "order": `A` is followed by `G`, which in turn is followed by `T`, which is followed by `C`, which is followed by `A` again. State-of-the-art research in genomics has revealed the startling fact that many diseases are caused by certain subsequences of bases not forming a palindromic sequence! Your mission as a leading researcher at ICPC laboratories is to take a DNA string S and a series of subsets $P_1, \ldots, P_t$ of indices to characters (nucleobases) in S, and transform S so that each of the restrictions of the resulting string to $P_1, \ldots, P_t$ are palindromic. (The restriction of S to a subset $P = \{i_1, i_2, \ldots, i_k\}$ of indices, where $0 \le i_1 < i_2 < \ldots < i_k < |S|$, is the string $S_{i_1} S_{i_2} \ldots S_{i_k}$). It is possible to inspect any base of S at will, but only three transformations can be applied to a base:

1. Leave it unaltered.
2. Increase it by 1 in the cyclic order of nucleobases (e.g. turn `C` into `A`).
3. Decrease it by 1 (e.g. turn `T` into `G`).

Moreover, owing to limitations of current technology, it is impossible to modify two bases in consecutive positions of the sequence. Is our goal achievable?

By way of example, consider DNA sequence `AGTAT`. Number positions starting from 0, and suppose we have the three subsets $P_1 = \{1, 4\}$, $P_2 = \{0, 1\}$ and $P_3 = \{0, 2, 4\}$. One solution is to increase the first character and decrease the last, yielding $S_0 =$ `GGTAG`. The restrictions of S_0 to P_1, P_2 and P_3 are `GG`, `GG` and `GTG`, respectively; all of them are palindromic. One case where no solution is possible is when the string is `CATGC`, and we require the subsequences determined by positions $\{0, 3\}$ and $\{3, 4\}$ be palindromic. Here, characters 3, 0 and 4 would all need to become a `T`. But this entails modifying consecutive characters 3 and 4, which is not allowed.

Input

The first line of each test case has two integers N and T ($1 \le N \le 10\,000, 1 \le T \le 6\,000$), the sequence length and number of subsets to consider. The next line contains the DNA sequence of length N, all of whose characters are in `ACGT`. The subsets are described by the following T lines. Each line starts by "L:", where L ($0 \le L \le N$) is the number of positions in the subset, and is followed by T distinct integers between 0 and $N - 1$ in increasing order. Subsets may overlap partially or totally.

A blank line separates different test cases. The input file is terminated by a line containing `0 0`.

Output

In a single line per test case, print `YES` if the task is solvable and `NO` otherwise.

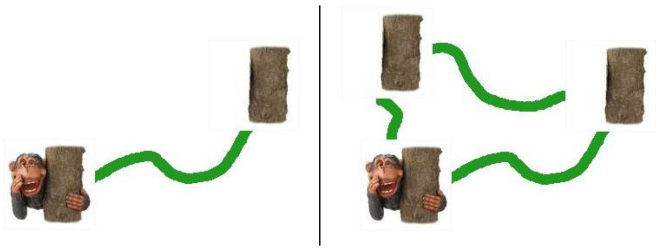
Example

standard input	standard output
5 3 AGTAT 2: 1 4 2: 0 1 3: 0 2 4 5 3 CATGC 0: 2: 0 3 2: 3 4 0 0	YES NO

Problem C. Jumping Monkey

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

You are a hunter chasing a monkey in the forest, trying to shoot it down with your all-powerful automatic machine gun. The monkey is hiding somewhere behind the branches of one of the trees, out of your sight. You can aim at one of the trees and shoot; your bullets are capable of going through the branches and killing the monkey instantly if it happens to be in that tree. If it isn't, the monkey takes advantage of the time it takes you to reload and takes a leap into a neighbouring tree without you noticing. It never stays in the same place after a shot. You would like to find out whether there is an strategy that allows you to capture the monkey for sure, irrespective of its initial location and subsequent jumps. If so, you need to determine the shortest sequence of shots guaranteeing this.



As an example, consider the situation in which there are only two neighboring trees in the forest (left hand side of the figure above). It is then possible to make sure you capture the monkey by shooting twice at the same tree. Your first shot succeeds if the monkey happened to be there in the first place. Otherwise, the monkey was behind the other tree and it will necessarily have moved when you shoot for the second time. However, depending on the shape of the forest it may not possible for you to ensure victory. One example of this is if there are three trees, all connected to one another (right hand side of the figure above). No matter where you aim at, there are always two possible locations for the monkey at any given moment. (Note that here we are concerned with the worst-case scenario where the monkey may consistently guess your next target tree).

Input

The input consists of several test cases, separated by single blank lines. Each test case begins with a line containing two integers n and m ($1 \leq n \leq 21$); n is the number of trees in the forest, and m is the number of adjacency relations between trees. Each of the following m lines contains two distinct integers between 0 and $n-1$ (inclusive), the identifiers of the trees in an adjacent pair. The order of both trees within a pair carries no meaning, and no pair appears more than once. You may further assume that no tree is adjacent to itself, and there is always a path between any two trees in the forest.

The test cases will finish with a line containing only two zeros (also preceded with a blank line).

Output

Print a line for each test case. The line should contain the single word `Impossible` if the task is impossible. Otherwise, it must contain the shortest sequence of shots with the required property, in the format $L : V_1 V_2 \ldots V_L$, where L is the length of the sequence, and $V_1, V_2, \ldots, V_L$ are space-separated integers containing the identifiers of the trees to shoot at in the right order. If several shortest sequences exist, print the lexicographically smallest one. (A sequence is smaller than another in lexicographic order if the first element on which they differ is smaller in the first one).

Example

standard input	standard output
2 1 0 1 3 3 0 1 1 2 2 0 4 3 0 1 2 3 1 3 0 0	2: 0 0 Impossible 4: 1 3 3 1

Problem D. 3-sided Dice

Input file: standard input
Output file: standard output

Just like every fall, the organizers of the Southwestern Europe Dice Simulation Contest are busy again this year. In this edition you have to simulate a 3-sided die that outputs each of three possible outcomes (which will be denoted by 1, 2 and 3) with a given probability, using three dice in a given set. The simulation is performed this way: you choose one of the given dice at random, roll it, and report its outcome. You are free to choose the probabilities of rolling each of the given dice, as long as each probability is strictly greater than zero. Before distributing the materials to the contestants, the organizers have to verify that it is actually possible to solve this task.

For example, in the first test case of the sample input you have to simulate a die that yields outcomes 1, 2 and 3 with probabilities $\frac{3}{10}$, $\frac{4}{10}$ and $\frac{3}{10}$. We give you three dice, and in this case the i-th of them always yields outcome i, for each $i = 1, 2, 3$. Then it is possible to simulate the given die in the following fashion: roll the first die with probability $\frac{3}{10}$, the second one with probability $\frac{4}{10}$ and the last one with probability $\frac{3}{10}$.

Input

The input consists of several test cases, separated by single blank lines. Each test case consists of four lines: the first three of them describe the three dice you are given and the last one describes the die you have to simulate. Each of the four lines contains 3 space-separated integers between 0 and 10 000 inclusive. These numbers will add up to 10 000, and represent 10 000 times the probability that rolling the die described in that line yields outcome 1, 2 and 3, respectively.

The test cases will finish with a line containing only the number zero repeated three times (also preceded with a blank line).

Output

For each case, your program should output a line with the word `YES` if it is feasible to produce the desired die from the given ones, and `NO` otherwise.

Example

standard input	standard output
0 0 10000 0 10000 0 10000 0 0 3000 4000 3000 0 0 10000 0 10000 0 3000 4000 3000 10000 0 0 0 0 0	YES NO

Problem E. Assembly Line

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

The last worker in a production line at the factory of Automated Composed Machinery is worried. She knows that her job hangs in the balance unless her productivity increases. Her work consists of assembling a set of pieces in a given sequence, but the time spent on assembling pieces a and b and then c may not be the same as that on assembling pieces b and c, and then assembling a with the resulting component. Only two consecutive pieces may be assembled at a time, and once they are assembled they behave as another piece in terms of the time needed for further assembly.

In order to aid her, you need to find the optimal way to assemble all components. The input to your program will be a set of symbols representing (types of) pieces, and a so-called assembly table representing the time it takes to assemble them, as well as the type of the resulting component. For instance, we may have two symbols $\{a; b\}$, and the following table:

	a	b
a	3–b	5–b
b	6–a	2–b

This means, for example, that two pieces of type a and a may assembled in 3 minutes, and the result is a component of type b, in that the time required to assemble it again with another piece of, say, type a is 6 minutes, and so on. Note that the table is not symmetric, i.e. assembling b and a may be more time-consuming than a and b.

For a sequence of components labelled aba, the two possible solutions are:

- $(ab)a = ba = a$ with time $time(ab) + time(ba) = 5 + 6 = 11$.
- $a(ba) = aa = b$ with time $time(ba) + time(aa) = 6 + 3 = 9$.

So the result for this case would be a piece of type b in 9 minutes (denoted 9–b).

Input

The input consists of several test cases. Each test case begins with a line containing a natural number k ($1 \le k \le 26$), followed by a line with k symbols (characters in `[a-z]`) separated by spaces. The following k lines contain the assembly table: the i-th line has k pairs of the form time-result, where time is an integer between 0 and 1 000 000 inclusive, and result a symbol belonging to the preceding set. The j-th pair in the i-th line represents the time to compose pieces of types represented by the i-th and j-th symbols, along with the type of the resulting piece. After the table, a line with an integer n indicates the number of lines that follow, each line being a string of at most 200 symbols. Each of these lines is a sequence of components that need to be assembled together in the right order.

The input will finish with a line containing 0, which should not be processed.

Output

For each test case, print n lines, each with an integer time and a symbol result in the format time-result. Each line represents the minimum time and the type of the resulting piece for the corresponding case in the input. In case of a tie among several possible results with the same minimum time, choose from among those the piece whose type letter appears first in the line that contained the k symbols at the beginning of the test case. (For example, if that line was a c b and both c and b can be obtained with minimum cost 5, print 5–c).

There must be an empty line between the output of different test cases.

Example

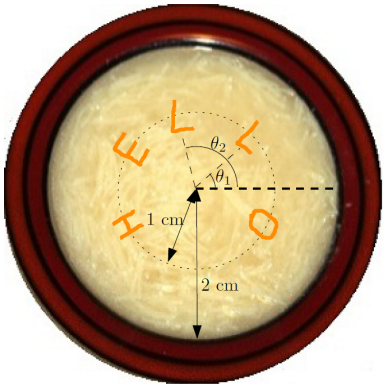
standard input	standard output
2 a b 3-b 5-b 6-a 2-b 2 aba bba 2 m e 5-e 4-m 3-e 4-m 1 eme 0	9-b 8-a 7-m

Problem F. Alphabet Soup

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

Peter is having lunch at home. Unfortunately for him, today's meal is soup. As Peter's mother is aware that he doesn't like it very much, she has cooked a special soup using pasta pieces shaped like letters from the alphabet, numbers and other characters. She has a special knife with which she can prepare an unlimited supply of pasta pieces that may come in S different forms. The soup always has P pasta pieces in it, and is so thick that the pieces never move.

Despite her efforts, Peter is still not happy with today's menu and asks how many days in his life he will have to eat soup. His mother promises him that she will prepare a different soup every day, and that on no day will the dish contain the same shapes in all positions as any soup dish previously served. However, the number P of pasta pieces, as well as the positions in which pieces float, will remain the same every day. Peter is not easily fooled (or so he thinks), and he cleverly realizes that this can still make him eat soup for ages. In an attempt to reduce the number of configurations, he tells his mother he will not accept any dish which can be obtained by rotating one of the configurations previously seen.



Consider the dish as a circle of radius 2 centered at the origin. All the symbols will be floating in the soup at a given angle (in millidegrees) at distance 1 from the origin. Two plates are considered equal if you can perform a rotation of one of the dishes about its center so that the positions of the symbols, as well as the symbols themselves, are the same in both.

Your program will be given the number of possible symbols Peter's mother has available, and the angles determining the location of each of the pasta pieces (measured clockwise in millidegrees). Write a program that returns the number of possible plates Peter's mother can prepare. As this number can be very large, output the number modulo 100 000 007, which is prime.

Input

The first line of input in each test case contains two numbers: S ($2 \le S \le 1\,000$), the number of symbols Peter's mother can use, and P ($P > 0$), the number of pasta pieces floating in the soup. Each of the next P lines contain the angle A ($0 \le A < 360\,000$) of one of the P pieces (measured clockwise in millidegrees). All angles will be different.

Different tests cases are separated by a blank line. After the last test case there is a line with $S = P = -1$.

Output

For each test case output a single integer in a line by itself, the number of different plates Peter's mother can cook modulo 100 000 007.

Example

standard input	standard output
2 4 0 90000 180000 270000 100 5 0 45000 90000 180000 270000 -1 -1	6 99999307

Problem G. Game, Set and Match

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

In this problem you need to assist in computing the probability of winning at tennis. Here is a brief explanation of how the scoring system works. In a tennis *match*, players play a certain number of consecutive *sets*. Each set is in turn made up of a series of *games* (and may include a *tie-break* if needed). Finally each game is made of *points*.

Points. Every point is started by one of the players serving (i.e. hitting the ball into the service box in the opposite court) and the other receiving serve. The server then attempts to return the ball into the server's court and players alternate hitting the ball across the net. When one of the players fails to make a legal return (e.g. if the ball is knocked out of the court), he or she loses the point. The specifics of how points are won are not important to us.

Games. The scoring system within a game is peculiar to say the least. As the player wins points in a game, his score goes from the initial value of 0 (read "love") to 15, 30, or 40 (yes, just when you think you're starting to spot a pattern in this mess it breaks down). There is no a-priori limit to the length of a game (meaning the number of points played), but a player's score is always indicated by one of these numbers according to the following rules. When a player has three points (score 40) and wins the following point as well, he wins the game unless the scoreline was 40–40 (read "deuce") to start with. A player needs to win two consecutive points from deuce to win the game. Winning one gives him advantage; if followed by a second winning point the game is won by him, but if followed by a losing point the score reverts to deuce. Example: at 40–30, if the first player wins the next point he wins the game. However, if the second player wins the next three points the game is his.

Sets. A player wins a set if he wins at least four games (in the current set) and he is two games ahead of his opponent but, as you may be starting to suspect, there is yet another exception. In case the scoreline for the number of games won reaches six-all (6–6), a tie-break is played instead to decide the set. Example: at 5–4, if the first player wins the next game he takes the set 6–4. But if he loses, the set is still undecided and can eventually go to either 7–5, 5–7 or a tie-break.

Tie-break. A tie-break (and the set to which it belongs) is won when a player wins at least seven points by a margin of two points or more.

Match. The winner of a match is the first player to win 2 sets (the wins do not need to be consecutive). Hence a match may go to 2 or 3 sets depending on how the game develops.

Rafa has been carefully studying his past performances against his next opponent and he knows he wins each point with probability precisely p, irrespective of whether he is serving or receiving and regardless of all other points played. Can you help him assess his chances of winning the match?

Input

Each test case is described by a single floating point number p, $0 \leq p \leq 1$ in its own line. A value of -1 for p marks the end of the input.

Output

For each test case, print a single line with the probabilities of Rafa winning a given game, set and match, respectively. These three numbers must be separated by a space character. Your answers should be accurate to within an absolute error of 10^{-6}.

Example

standard input	standard output
0.5 0.3 0.7 -1	0.50000000000 0.50000000000 0.50000000000 0.09921103448 0.00016770463 0.00000008437 0.90078896552 0.99983229537 0.99999991563

Problem H. Non-negative Partial Sums

Input file: standard input
Output file: standard output

You are given a sequence of n numbers $a_0, \ldots, a_{n-1}$. A cyclic shift by k positions ($0 \le k \le n-1$) results in the following sequence: $a_k, a_{k+1}, \ldots, a_{n-1}, a_0, a_1, \ldots, a_{k-1}$. How many of the n cyclic shifts satisfy the condition that the sum of the first i numbers is greater than or equal to zero for all i such that $1 \le i \le n$?

Input

Each test case consists of two lines. The first contains the number n ($1 \le n \le 10^6$), the number of integers in the sequence. The second contains n integers $a_0, \ldots, a_{n-1}$ ($-1000 \le a_i \le 1000$) representing the sequence of numbers. The input will finish with a line containing 0.

Output

For each test case, print one line with the number of cyclic shifts of the given sequence which satisfy the condition stated above.

Example

standard input	standard output
3 2 2 1 3 -1 1 1 1 -1 0	3 2 0

Problem I. Beehives

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

Bees are one of the most industrious insects. Since they collect nectar and pollen from flowers, they have to rely on the trees in the forest. For simplicity they numbered the n trees from 0 to $n-1$. Instead of roaming around all over the forest, they use a particular list of paths. A path is based on two trees, and they can move either way i.e. from one tree to another in straight line. They dont use paths that are not in their list.

As technology has been improved a lot, they also changed their working strategy. Instead of hovering over all the trees in the forest, they are targeting particular trees, mainly trees with lots of flowers. So, they planned that they will build some new hives in some targeted trees. After that they will only collect their foods from these trees. They will also remove some paths from their list so that they dont have to go to a tree with no hive in it.

Now, they want to build the hives such that if one of the paths in their new list go down (some birds or animals disturbs them in that path) its still possible to go from any hive to another using the existing paths. They dont want to choose less than two trees and as hive-building requires a lot of work, they need to keep the number of hives as low as possible. Now you are given the trees with the paths they use, your task is to propose a new bee hive colony for them.

Input

Input starts with an integer T ($T \leq 50$), denoting the number of test cases.

Each case starts with a blank line. Next line contains two integers n ($2 \leq n \leq 500$) and m ($0 \leq m \leq 20000$), where n denotes the number of trees and m denotes the number of paths. Each of the next m lines contains two integers u v ($0 \leq u, v < n, u \neq v$) meaning that there is a path between tree u and v. Assume that there can be at most one path between tree u to v, and needless to say that a path will not be given more than once in the input.

Dataset is huge. Use faster I/O methods.

Output

For each case, print the case number and the number of beehives in the proposed colony or `impossible` if its impossible to find such a colony.

Example

standard input	standard output
3 3 3 0 1 1 2 2 0 2 1 0 1 5 6 0 1 1 2 1 3 2 3 0 4 3 4	Case 1: 3 Case 2: impossible Case 3: 3

Problem J. RNA Secondary Structure

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

RNA, which stands for Ribonucleic Acid, is one of the major macromolecules that are essential for all known forms of life. It is made up of a long chain of components called nucleotides. Each component is made up of one of 4 bases and are represented using `A`, `C`, `G` or `U`. The primary structure of RNA is a sequence of these characters. The secondary structure of RNA refers to the base pairing interactions between different components. More specifically, base `A` can pair up with base `U` and base `C` can pair up with base `G`. The stability of the RNA secondary structure depends on the total number of base pairs that can be formed. The final structure is the one that contains the maximum number of base pairs.

Lets represent the primary structure as a string consisting of characters from the set $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\}$. The rules of secondary structure formation are as follows:

1. Any base `A` can form a pair with any base `U`.
2. Any base `C` can form a pair with any base `G`.
3. Each base can be a part of at most one pair.
4. Let's assume $w < x$, $y < z$ and $w < y$. If a base at index w forms a pair with a base at index x and a base at index y forms a pair with a base at index z, then one of the following two conditions must be true:
   - $y > x$;
   - $z < x$.
5. There can be at most K pairs between `C` and `G`.

You will be given the primary structure of the RNA of a certain species and your job is to figure out the total number of base pairings in the final secondary structure based on the constraints mentioned above.

You will be given the primary structure in a compressed format that uses run-length encoding. In this type of data compression, consecutive characters having the same value is replaced with a single character followed by its frequency. For example, `AAAACCGAAUUG` will be represented using `A4C2G1A2U2G1`. That means the primary structure will be given in the format $< c_1 f_1 c_2 f_2 c_3 f_3 \ldots c_n f_n >$, where c_i is from the set $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\}$ and f_i is a positive integer.

The species that we are dealing with have the following properties:

1. $f_1 + f_2 + f_3 + \ldots + f_n \leq 10050$;
2. $f_1 \leq 5000$;
3. $f_n \leq 5000$;
4. $f_2 + \ldots + f_{n-1} \leq 50$.

Input

The first line of input is an integer T ($T \leq 200$) that indicates the number of test cases. Each case contains two lines. The first line is the primary structure given in run-length encoded format. The second line gives you the value of K ($0 \leq K \leq 20$), that gives an upper limit on the number of `C`–`G` base pairs that can be in the final secondary structure.

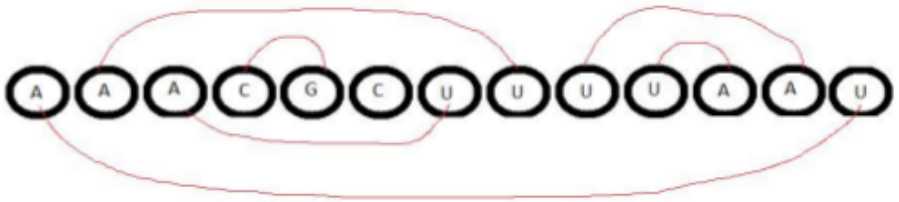
Output

For each case, output the case number followed by the maximum number of base pairs that can be formed. Look at the samples for exact format.

Example

standard input	standard output
3 A3C1G1C1U4A2U1 1 A3C1G1C1U4A2U1 0 A100U200 2	Case 1: 6 Case 2: 5 Case 3: 100

One possible final secondary structure for case 1 is depicted below that shows the 6 base pairings.



Problem K. The Moon of Valencia

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

It is well known that the Moon of Valencia is magical. Everyone talks about a mystery that happens at night. People remember what time they entered the first bar, what time arrived to the hotel and how happy they arrived, but nobody remembers the bars and pubs they visited.

The Valencia hotels have hired you for developing an application that helps customers to remember. The application will inform customers with one of all the possible sequences of bars and pubs. Customers have to provide the following information: departure time and place, arrival time and place, and degree of satisfaction on arrival.

The application uses a map with the location of each bar or pub. Each bar or pub produces a different degree of satisfaction when visited. But people gets angry when walks from one place to another, thats the reason why walking between different places reduces the degree of satisfaction. The reduction considered depends on the amount of minutes people needs to get one place from another one. If the amount of minutes is not an integer the remaining seconds should be considered a portion of a minute, i.e., 30 seconds imply 0.5 minutes. People walk at a speed of 4 km/h, can stay in a bar or pub the time they like, but for getting the satisfaction must remain at least 15 minutes. People can decide not to visit a bar or pub, i.e., they can use a path from the origin to the target and to enter in a subset of all the bars or pubs reachable with the path. Entering to the departure place is optional, like entering others places. The goal grade of satisfaction is computed up to the door of the target place, without entering it. So the grade of satisfaction of the target place (bar, pub or hotel) should not be computed.

Input

Input consists of several test cases. Each case begins with the map description, which is followed by the list of arrivals your application have to check. The description of a map begins with the word `MAP` in capital letters followed by two integer numbers, P and M, where P is the number of places and M is the number of paths connecting two places ($1 \leq P \leq 64$). Places and paths are described one per line. Each place is described with two coordinates (real numbers represent kilometers), its grade of satisfaction (a real number), the ID and the name. The paths connecting two places are identified by the their identifiers. Each pair of places can only be connected by one path.

It is guaranteed that no crossing paths exist.

The list of arrivals to be processed begins after a line with the word `ARRIVALS` in uppercase. Each arrival is described in a single line, including departure time, departure place, arrival time, arrival place and the grade of satisfaction on arrival, a real number.

Output

The output for each case must begin with the word `MAP` in capital letters followed by a number indicating the number of test case. The first test case is `MAP 1`, second is `MAP 2`, and so on. For each arrival proposed in the input it must appear a line in the output specifying a valid path found or the string `Impossible!` indicating that it is not possible to find a path from origin to target with the required grade of satisfaction. A path found will be valid if the absolute difference between the required grade of satisfaction and the obtained one is less than 0.1, and contains no loops, i.e. a place cant appear in the path found more than once.

Each path found must begin with the string `PATH FOUND:` in capital letters, followed by the obtained grade of satisfaction with three decimal digits and then the sequence of places from origin up to target. The ID of the unvisited places must appear preceded by the `!` sign. Notice as the ID of the target place never is preceded by this sign.

Hint: set up your solution for running as fast as possible by using this example, it should be enough for

all test cases.

Example

Example test is available at `http://rain.ifmo.ru/~buzdalov/moon-input.txt`.

Example answer is available at `http://rain.ifmo.ru/~buzdalov/moon-output.txt`.

The figure below shows the map corresponding to the first map in the example input case.