Palindromic DNA

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 $0 \leq i_1 < i_2 < \ldots < i_k < |S|$, 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 $S' =$ `GGTAG`. The restrictions of S' 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 ($1 \leq N \leq 10\,000, 1 \leq T \leq 6\,000$), 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 ($0 \leq L \leq N$) 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.

Sample Input

```
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
```

Sample Output

```
YES
NO
```

Assembly line

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 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 ($1 \leq k \leq 26$), 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.

Sample Input

```
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
```

Sample Output

```
9-b
8-a

7-m
```

Jumping monkey

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.

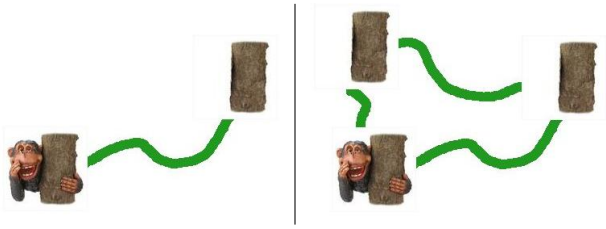


Figure 2

As an example, consider the situation in which there are only two neighboring trees in the forest (left hand side of Figure 2). 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 Figure 2). 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).

Sample Input

```
2 1
0 1

3 3
0 1
1 2
2 0

4 3
0 1
2 3
1 3

0 0
```

Sample Output

```
2: 0 0
Impossible
4: 1 3 3 1
```

F
Haunted Graveyard

Tonight is Halloween and Scared John and his friends have decided to do something fun to celebrate the occasion: crossing the graveyard. Although Scared John does not find this fun at all, he finally agreed to join them in their adventure. Once at the entrance, the friends have begun to cross the graveyard one by one, and now it is the time for Scared John. He still remembers the tales his grandmother told him when he was a child. She told him that, on Halloween night, "haunted holes" appear in the graveyard. These are not usual holes, but they transport people who fall inside to some point in the graveyard, possibly far away. But the scariest feature of these holes is that they allow one to travel in time as well as in space; i.e., if you fall inside a "haunted hole", you appear somewhere in the graveyard a certain time before (or after) you entered the hole, in a parallel universe otherwise identical to ours.

The graveyard is organized as a grid of $W \times H$ cells, with the entrance in the cell at position $(0, 0)$ and the exit at $(W - 1, H - 1)$. Despite the darkness, Scared John can always recognize the exit, and he will leave as soon as he reaches it, determined never to set foot anywhere in the graveyard again. On his way to the exit, he can walk from one cell to an adjacent one, and he can only head to the North, East, South or West. In each cell there can be either one gravestone, one "haunted hole", or grass:

- If the cell contains a gravestone, you cannot walk over it, because gravestones are too high to climb.
- If the cell contains a "haunted hole" and you walk over it, you will appear somewhere in the graveyard at a possibly different moment in time. The time difference depends on the particular "haunted hole" you fell into, and can be positive, negative or zero.
- Otherwise, the cell has only grass, and you can walk freely over it.

He is terrified, so he wants to cross the graveyard as quickly as possible. And that is the reason why he has phoned you, a renowned programmer. He wants you to write a program that, given the description of the graveyard, computes the minimum time needed to go from the entrance to the exit. Scared John accepts using "haunted holes" if they permit him to cross the graveyard quicker, but he is frightened to death of the possibility of getting lost and being able to travel back in time indefinitely using the holes, so your program must report these situations.

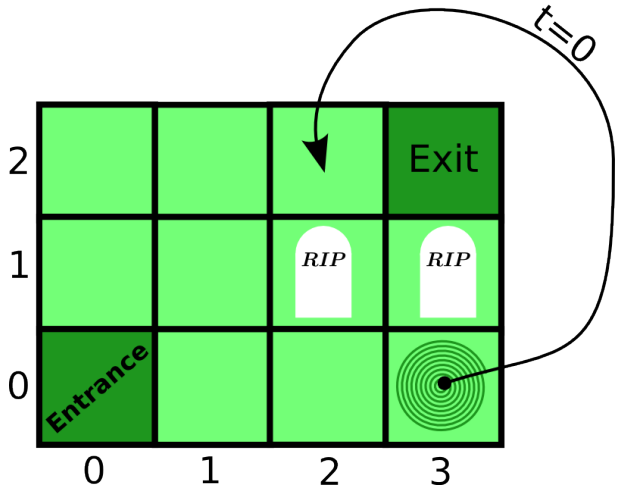


Figure 3: Sample graveyard

Figure **??** illustrates a possible graveyard (the second test case from the sample input). In this case there are two gravestones in cells $(2,1)$ and $(3,1)$, and a "haunted hole" from cell $(3,0)$ to cell $(2,2)$ with a difference in time of 0 seconds. The minimum time to cross the graveyard is 4 seconds, corresponding to the path:

$$(0,0) \rightarrow^{East}_{1\ sec} (1,0) \rightarrow^{East}_{1\ sec} (2,0) \rightarrow^{East}_{1\ sec} (3,0) \rightarrow^{hole}_{0\ sec} (2,2) \rightarrow^{East}_{1\ sec} (3,2)$$

If you do not use the "haunted hole", you need at least 5 seconds.

Input

The input consists of several test cases. Each test case begins with a line containing two integers W and H ($1 \leq W, H \leq 30$). These integers represent the width W and height H of the graveyard. The next line contains an integer G ($G \geq 0$), the number of gravestones in the graveyard, and is followed by G lines containing the positions of the gravestones. Each position is given by two integers X and Y ($0 \leq X < W$ and $0 \leq Y < H$).

The next line contains an integer E ($E \geq 0$), the number of "haunted holes", and is followed by E lines. Each of these contains five integers $X1, Y1, X2, Y2, T$. $(X1, Y1)$ is the position of the "haunted hole" ($0 \leq X1 < W$ and $0 \leq Y1 < H$). $(X2, Y2)$ is the destination of the "haunted hole" ($0 \leq X2 < W$ and $0 \leq Y2 < H$). Note that the origin and the destination of a "haunted hole" can be identical. T ($-10\,000 \leq T \leq 10\,000$) is the difference in seconds between the moment somebody enters the "haunted hole" and the moment he appears in the destination position; a positive number indicates that he reaches the destination after entering the hole. You can safely assume that there are no two "haunted holes" with the same origin, and the destination cell of a "haunted hole" does not contain a gravestone. Furthermore, there are neither gravestones nor "haunted holes" at positions (0,0) and (W-1,H-1).

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

Output

For each test case, if it is possible for Scared John to travel back in time indefinitely, output `Never`. Otherwise, print the minimum time in seconds that it takes him to cross the graveyard from the entrance to the exit if it is reachable, and `Impossible` if not.

Sample Input

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

Sample Output

```
Impossible
4
Never
```

Problem A
Almost Shortest Path

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

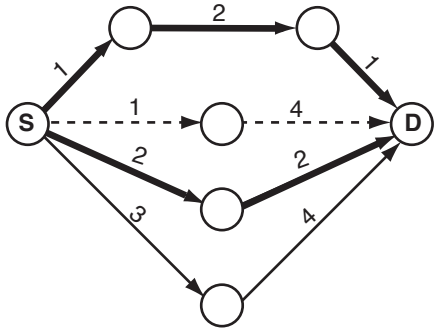
Finding the shortest path that goes from a starting point to a destination point given a set of points and route lengths connecting them is an already well known problem, and it's even part of our daily lives, as shortest path programs are widely available nowadays.

Most people usually like very much these applications as they make their lives easier. Well, maybe not that much easier.

Now that almost everyone can have access to GPS navigation devices able to calculate shortest paths, most routes that form the shortest path are getting slower because of heavy traffic. As most people try to follow the same path, it's not worth it anymore to follow these directions.

With this in his mind, your boss asks you to develop a new application that only he will have access to, thus saving him time whenever he has a meeting or any urgent event. He asks you that the program must answer not the shortest path, but the almost shortest path. He defines the almost shortest path as the shortest path that goes from a starting point to a destination point such that no route between two consecutive points belongs to any shortest path from the starting point to the destination.

For example, suppose the figure below represents the map given, with circles representing location points, and lines representing direct, one-way routes with lengths indicated. The starting point is marked as `S` and the destination point is marked as `D`. The bold lines belong to a shortest path (in this case there are two shortest paths, each with total length 4). Thus, the almost shortest path would be the one indicated by dashed lines (total length 5), as no route between two consecutive points belongs to any shortest path. Notice that there could exist more than one possible answer, for instance if the route with length 3 had length 1. There could exist no possible answer as well.



Input

The input contains several test cases. The first line of a test case contains two integers N ($2 \leq N \leq 500$) and M ($1 \leq M \leq 10^4$), separated by a single space, indicating respectively the number of points in the map and the number of existing one-way routes connecting two points directly. Each point is identified by an integer between 0 and $N-1$. The second line contains two integers S and D, separated by a single space, indicating respectively the starting and the destination points ($S \neq D$; $0 \leq S, D < N$). Each one of the following M lines contains three integers U, V and P ($U \neq V$; $0 \leq U, V < N$; $1 \leq P \leq 10^3$), separated by single spaces, indicating the existence of a one-way route from U to V with distance P. There is at most one

route from a given point U to a given point V, but notice that the existence of a route from U to V does not imply there is a route from V to U, and, if such road exists, it can have a different length. The end of input is indicated by a line containing only two zeros separated by a single space.

The input must be read from standard input.

Output

For each test case in the input, your program must print a single line, containing `-1` if it is not possible to match the requirements, or an integer representing the length of the almost shortest path found.

The output must be written to standard output.

Sample input	**Output for the sample input**
7 9 0 6 0 1 1 0 2 1 0 3 2 0 4 3 1 5 2 2 6 4 3 6 2 4 6 4 5 6 1 4 6 0 2 0 1 1 1 2 1 1 3 1 3 2 1 2 0 3 3 0 2 6 8 0 1 0 1 1 0 2 2 0 3 3 2 5 3 3 4 2 4 1 1 5 1 1 3 0 1 0 0	5 -1 6

Problem E
Electric Bill

File code name: `electric`

It's year 2100. Electricity has become very expensive. Recently, your electricity company raised the power rates once more. The table below shows the new rates (consumption is always a positive integer):

Range (Crazy-Watt-hour)	**Price (Americus)**
$1 \sim 100$	2
$101 \sim 10000$	3
$10001 \sim 1000000$	5
> 1000000	7

This means that, when calculating the amount to pay, the first 100 CWh have a price of 2 Americus each; the next 9900 CWh (between 101 and 10000) have a price of 3 Americus each and so on.

For instance, if you consume 10123 CWh you will have to pay $2\times100+3\times9900+5\times123 = 30515$ Americus.

The evil mathematicians from the company have found a way to gain even more money. Instead of telling you how much energy you have consumed and how much you have to pay, they will show you two numbers related to yourself and to a random neighbor:

A: the total amount to pay if your consumptions were billed together; and

B: the absolute value of the difference between the amounts of your bills.

If you can't figure out how much you have to pay, you must pay another 100 Americus for such a "service". You are very economical, and therefore you are sure you cannot possibly consume more than any of your neighbors. So, being smart, you know you can compute how much you have to pay. For example, suppose the company informed you the following two numbers: $A = 1100$ and $B = 300$. Then you and your neighbor's consumptions had to be 150 CWh and 250 CWh respectively. The total consumption is 400 CWh and then A is $2\times100+3\times300 = 1100$. You have to pay $2\times100+3\times50 = 350$ Americus, while your neighbor must pay $2\times100+3\times150 = 650$ Americus, so B is $|350-650| = 300$.

Not willing to pay the additional fee, you decided to write a computer program to find out how much you have to pay.

Input

The input contains several test cases. Each test case is composed of a single line, containing two integers A and B, separated by a single space, representing the numbers shown to you

($1 \leq A, B \leq 10^9$). You may assume there is always a unique solution, that is, there exists exactly one pair of consumptions that produces those numbers.

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

Output

For each test case in the input, your program must print a single line containing one integer, representing the amount you have to pay.

Sample input	**Output for the sample input**
1100 300 35515 27615 0 0	350 2900

Problem F
Ordering Tasks

Input: standard input
Output: standard output
Time Limit: 1 second
Memory Limit: 32 MB

John has n tasks to do. Unfortunately, the tasks are not independent and the execution of one task is only possible if other tasks have already been executed.

Input

The input will consist of several instances of the problem. Each instance begins with a line containing two integers, **1 <= n <= 100** and **m**. **n** is the number of tasks (numbered from **1** to **n**) and **m** is the number of direct precedence relations between tasks. After this, there will be **m** lines with two integers **i** and **j**, representing the fact that task **i** must be executed before task **j**. An instance with **n = m = 0** will finish the input.

Output

For each instance, print a line with **n** integers representing the tasks in a possible order of execution.

Sample Input

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

Sample Output

```
1 4 2 5 3
```

(The Joint Effort Contest, Problem setter: Rodrigo Malta Schmidt)