

Problem 1 – Events

A company wants to build a calendar system to **keep track of events**. Each event has date, time, title and location. The system should be able to add events, delete events and list upcoming events. Your task is to model the calendar system and design a data structures to hold the events. You need to write a program that executes a sequence of commands. Each command consists of a single text line and produces zero, one or more text lines. The commands are described below:

- **AddEvent date | title | location** – adds an event by given data and time, title and optional location. If the location is not specified, the command takes the form "**AddEvent date | title**". If the event already exists, it is added again (duplicates are accepted). The result of the command execution is "**Event added**".
- **DeleteEvents title** – deletes all events matching given title in case insensitive manner. The result of the command execution is "**X events deleted**" where **X** is the number of deleted events or "**No events found**" (if no events match the given title).
- **ListEvents date | count** – lists the events starting from the given date and time. The number of the listed events should be the given **count** or less (if not enough events are available). If no events are available, the command should output "**No events found**". Each event should be printed on a separate line in the format "**date | title | location**". If the location is missing, the events should be printed in the format "**date | title**". The listed events should be ordered chronologically by date and time and then by title and location (in ascending order).
- **End** – indicates the end of the input sequence of commands. "**End**" stops the commands processing without any output. It is always the last command in the input sequence.

Input

The input data comes from the console. It consists of a sequence of commands, each staying at a separate single line. The input ends by the "**End**" command.

The **input data will always be valid** and in the described format. There is no need to check it explicitly.

Output

The output should be on the console. It should consist of the outputs from each of the commands from the input sequence.

Constraints

- The **date** will always be given in the format "**yyyy-MM-ddTHH:mm:ss**" and should be printed in the same format. The dates are in the range [01.01.2000T00:00:00...01.01.2020T00:00:00]. Thus "**2012-03-09T14:01:54**" is valid date, but "**2011-1-1T1:1:1**" and "**2011-3-22 6:43:28**" are not.
- The **title** and **location** will be non-empty English text (less than 1000 characters) and cannot contain "|" and "\n", as well as leading or trailing whitespace.
- The **count** will always be integer number in the range [1...100].
- There is a single space on the left and on the right of each "|" in the input.
- There is a single space between each command and its arguments.
- The input cannot exceed **2.5 MB**.
- Allowed working time for your program: 1.00 second. Allowed memory: 64 MB.

Hints

- For improved performance, you should use **StringBuilder** to store your output and print it at the end of the program execution.
- Note that in the standard sorting order the missing value (**null**) is before any other value.

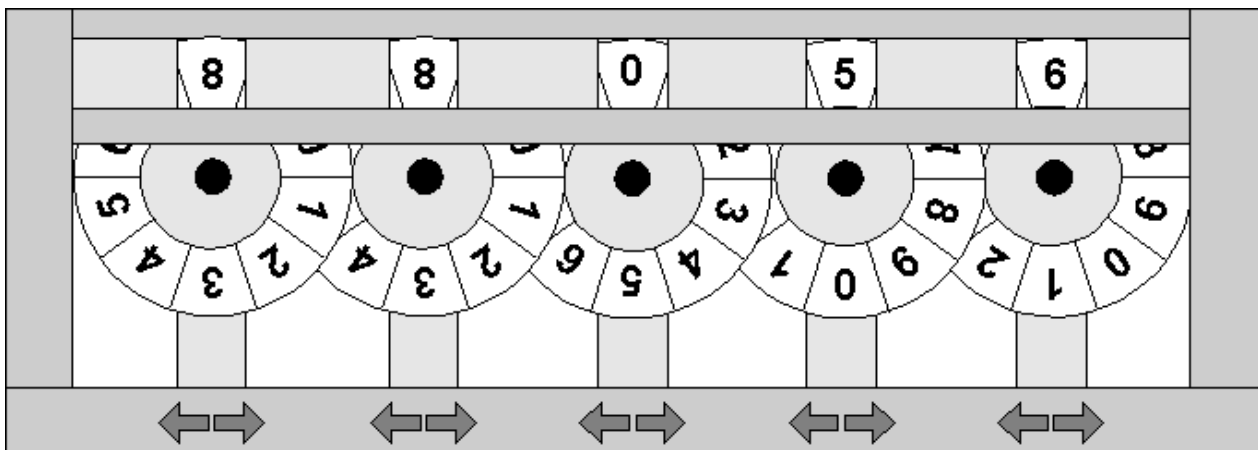
Examples

Input example	Output example
<pre>AddEvent 2012-01-21T20:00:00 party Viki home AddEvent 2012-03-26T09:00:00 C# exam AddEvent 2012-03-26T09:00:00 C# exam AddEvent 2012-03-26T08:00:00 C# exam AddEvent 2012-03-07T22:30:00 party Vitosha ListEvents 2012-03-07T08:00:00 3 DeleteEvents c# exam DeleteEvents My granny's bushes ListEvents 2013-11-27T08:30:25 25 AddEvent 2012-03-07T22:30:00 party Club XXX ListEvents 2012-01-07T20:00:00 10 AddEvent 2012-03-07T22:30:00 Party Club XXX ListEvents 2012-03-07T22:30:00 3 End</pre>	<pre>Event added Event added Event added Event added Event added 2012-03-07T22:30:00 party Vitosha 2012-03-26T08:00:00 C# exam 2012-03-26T09:00:00 C# exam 3 events deleted No events found No events found Event added 2012-01-21T20:00:00 party Viki home 2012-03-07T22:30:00 party Club XXX 2012-03-07T22:30:00 party Vitosha Event added 2012-03-07T22:30:00 party Club XXX 2012-03-07T22:30:00 party Vitosha 2012-03-07T22:30:00 Party Club XXX</pre>

Problem 2 – Risk Wins, Risk Loses

One of the most favorite Bulgarian TV presenters is Kutsi Vuptsarov (he is also well known with his song "Hey, grandma"). He loves all possible games which involve any kind of wheels. One day he thought that his once famous game "Risk Wins, Risk Loses" must be reborn with some new wheel games. Since you are a big fan of his jokes (and his song) you should help him by writing a program which finds the best solution of his new 5-wheel based game.

Digits in the range from 0 to 9 are painted consecutively clockwise on the periphery of each wheel. The topmost digits of the wheels form a five-digit integer. For example, in the following figure the wheels form the integer 88056.



Each one of the five wheels has two buttons associated with it. When you press the button marked with a left arrow you rotate the wheel one digit clockwise and when you press the one marked with the right arrow you rotate it one digit in the opposite direction.

The game starts with an initial configuration of the wheels, given by the topmost digits of the wheels. You will be given **N** forbidden configurations and a target configuration, also given by the topmost digits.

Your task is to write a program that calculates the minimum number of button presses required to transform the initial configuration to the target configuration by never passing through a forbidden one.

Input

The input data should be read from the console. The first input line contains the initial configurations of the wheels. The second line contains the target configuration of the wheels. On the third line there will be the number **N**. Each of the following **N** lines contains a forbidden configuration.

The input data will always be valid and in the described format. There is no need to check it explicitly.

Output

The output data should be printed on the console. On the only output line print the minimum number of button presses required. If the target configuration is not reachable then print -1.

Constraints

- The number of the wheels will always be exactly 5.
- **N** will be between 0 and 150 000, inclusive.
- All configurations will be given by the topmost digits of the wheels.
- Allowed working time for your program: 0.15 seconds. Allowed memory: 16 MB.

Examples

Input example	Output example	Input example	Output example
88056 86508 5 88057 88047 85508 87508 86408	14	00000 65536 10 00001 00010 00100 01000 10000 90000 09000 00900 00090 00009	-1
You need at least 14 button presses in order to change 88056 to 86508 without passing through forbidden combination.		You could never change the initial combination because of the forbidden combinations.	

Problem 3 – Validate HTML

An open HTML tag (e.g. <tag>) marks the beginning of an element with some name (e.g. "tag").

A close HTML tag marks the end of a HTML element. In close tags the element name is prepended with a forward slash (e.g. </tag>).

You are given a sequence of opening and closing HTML tags (not necessarily with real HTML tags names) and your task is to write a program that validates this sequence. All of the tags can be nested (e.g. in the sequence "<a>" the tag b is nested of tag a).

A valid sequence of HTML tags is any sequence of opening and closing HTML tags for which:

- all opened tags are closed
- there aren't any closed tags before they are opened
- all of the tags are closed only after all their nested tags are closed

Input

The input data should be read from the console.

On the first line you should read the number **N**.

On each of the next **N** lines you will be given a HTML code which you should validate.

The input data will always be valid and in the described format. There is no need to check it explicitly.

Output

The output data should be printed on the console.

For each of the **N** HTML codes from the input you should output a single line containing "VALID" or "INVALID", depending on the validity of the HTML code.

Constraints

- **N** will be between 2 and 50, inclusive.
- The sum of all opening and closing tags will be between 4 and 10 000, inclusive.
- All of the tag names will contain only small Latin letters ('a'-'z') and their length will be between 2 and 10 characters, inclusive.
- The html codes will contain only the symbols '<', '>', '/' and the small Latin letters from 'a' to 'z'.
- Allowed working time for your program: 0.25 seconds. Allowed memory: 16 MB.

Examples

Input example	Output example
<pre>6 <html><head></head><body><i></i></body></html> <i></i><test></test><e><taggg></taggg></e> <e><abv></e></abv> <x><y><z><a></e></x></y></z> <c><a></c> <i></i><i></i><i></i><i></i><i></i></pre>	<pre>VALID VALID INVALID INVALID INVALID VALID</pre>

Problem 4 – Reach N

You start with the integer **1** and you apply a sequence of operations until you reach the integer **N**. Each operation can be one of the following:

- Increment the current number by 1
- If the current number is greater than 1, decrement it by 1
- Raise the current number to any positive integral power

Return the minimum possible number of operations required to reach **N**.

Input

The input data should be read from the console.

You will be given 4 lines with numbers. Find the minimum possible number of operations for each of the 4 numbers.

The input data will always be valid and in the described format. There is no need to check it explicitly.

Output

The output data should be printed on the console.

For each of the 4 numbers given in the input write a single line, containing the minimum possible number of operations required to reach the number, starting from 1.

Constraints

- For each of the 4 test cases **N** will be between 1 and 10^{18} , inclusive.
- Allowed working time for your program: 0.15 seconds. Allowed memory: 16 MB.

Example

Input example	Output example	Explanation
8 1 80 123456789	2 0 4 2566	<p>First number: minimum 2 steps:</p> <ol style="list-style-type: none"> 1. Increment by 1: $1 + 1 = 2$ 2. Raise to the power of 3: $2^3 = 8$ <p>Second number: 0 steps The number doesn't need any operations.</p> <p>Third number: minimum 4 steps:</p> <ol style="list-style-type: none"> 1. Increment by 1: $1 + 1 = 2$ 2. Increment by 1: $2 + 1 = 3$ 3. Raise to the power of 4: $3^4 = 81$ 4. Decrement by 1: $81 - 1 = 80$ <p>Fourth number: minimum 2566 steps</p>

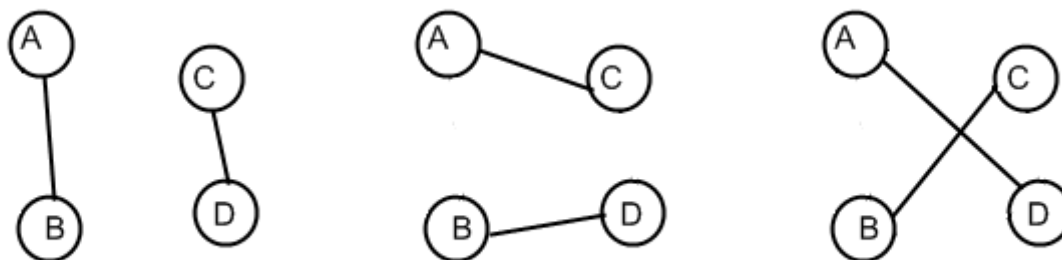
Problem 5 – Connect All Cities

There is a country with **N** cities, some of which are connected with bidirectional roads. Your task is to reconfigure the roads so that it is possible to get from each city to every other city. You must do this

using the minimum possible number of transformations, where each transformation consists of the following steps:

1. Choose four different cities A, B, C and D, where roads (A, B) and (C, D) exist, but (A, C), (A, D), (B, C) and (B, D) do not exist.
2. Destroy roads (A, B) and (C, D).
3. Build two new roads - either (A, C) and (B, D), or (A, D) and (B, C).

The following images show an example of this transformation. From the first situation you can get the second one or the third one:



Input

The input data should be read from the console.

On the first line you should read **T** – the number of the tests you should solve.

Each of the **T** tests will be in the following format:

On the first line of each test case you will be given the number **N** – the number of the cities in the country.

On the each of the next **N** lines you will be given **N** symbols altogether constructing a two dimensional array **G** where the **j**-th character of the **i**-th line is '1' if there is a road between cities **i** and **j**, and '0' otherwise.

The input data will always be valid and in the described format. There is no need to check it explicitly.

Output

The output data should be printed on the console.

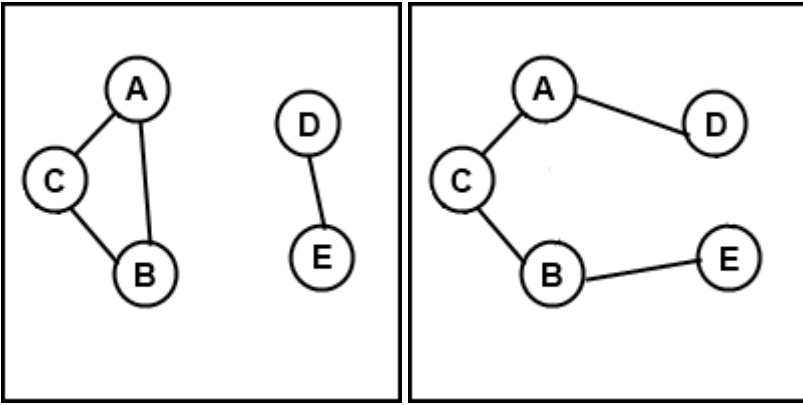
For each of the test cases print one line containing the minimal number of transformations required to accomplish your task, or print **-1** if it is impossible.

Constraints

- **T** will be between 3 and 20, inclusive.
- **N** will be between 2 and 50, inclusive.
- For each **i** and **j**, **G[i][j]** will be equal to **G[j][i]** (if city A is connected to city B, then city B is also connected to city A).
- For each **i**, **G[i][i]** will be equal to '0' (no city is connected to itself).

- Allowed working time for your program: 0.10 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example	Explanation
3 2 01 10 5 01100 10100 11000 00001 00010 6 010000 101000 010100 001000 000001 000010	0 1 -1	<p>First test: The cities are already connected.</p> <p>Second test: You need one transformation shown on the pictures below:</p>  <p>Third test: Here it is impossible to connect all the cities using the given transformation, no matter how many times you do it.</p>