

Problem 1 - Protoss

For a lot of time now, we've wondered how the highly-advanced alien race - the Protoss - can conduct short-range telecommunication without any radio transmitter/receiver. Recent studies have shown Protoss have evolved an "organic antenna" of sort in their skull. This antenna transmits and receives at very high frequencies and is capable of sending what we would call "text" - giving the Protoss the ability of sending telepathic messages to each other.

The Dark Templar - the Protoss' most valued assassins have been a pain in our backs for a long time. Being invisible to the naked eye, they've been able to cause severe damage to our bases lacking sufficient detection hardware. With these new discoveries we can tap into the Dark Templar's communications and find what they are up to and prepare before they can do any harm.

Dark Templar attack in groups. First they pick a target area. Then each of them tells all the others in which area he is currently in. Then they go from their current areas to the target area. The good thing is, the Protoss are as advanced as they are superstitious. Dark templar believe that they must move only directly east, west, north or south of their current area (so on a map that would be left, right up or down), or they will lose their powers. Basically, if a dark templar wants to go in the area that is upper-left of his current area, he has to first go up and then go left - a total of 2 transitions from one area to another. The "distance" at which a dark templar is from some area we will call the total number of transitions from one area to another he has to make to reach that area (so in the example above the "distance" is 2).

We have a map of areas, which is a matrix of area names. Each area name is one or several English letter substrings. If an area name consists of more than one substring, then there are dashes ('-') between the substrings in its name. For each cell $[i][j]$ in the matrix, it's upper (north) cell is $[i-1][j]$, it's right (east) cell is $[i][j+1]$, it's lower (south) cell is $[i+1][j]$ and it's left (west) cell is $[i][j-1]$ (if those cells are in the matrix).

The Protoss have their own secret strings for the substrings in the areas in our map. Luckily each secret name corresponds to exactly one of the substrings used in our area names. For example, if an area's name is "a-b" and the Protoss secret string for "a" is "p" and the one for "b" is "f", then the Protoss area name for "a-b" will be "p-f".

Write a program, which by given **distance**, Protoss **secret strings and their meanings**, **area map** (matrix), **target area** and **dark templar current areas** finds the **number of dark templar within** the specified **distance from the target area**.

Input

The input data should be read from the console.

On the first line there is one integer number - the **distance** from the target area in which to count dark templar

On the next line there will be one integer **S** - the number of secret strings

On each of the next **S** lines there will be two strings - a protoss secret string and the substring it corresponds to

On the next line there will be the numbers **R** and **C** - the number of rows and the number of columns in the matrix, describing the map

On each of the next **R** lines there will be **C** strings, consisting of English letters and dashes, describing the names of the areas in the map

On the next line there will be one or more "secret strings", connected with dashes - the Protoss name of the **target area**

On the next line there will be one integer **T** - the number of dark templar in the areas of the map

On each of the next **T** lines there will be one or more "secret strings", connected with dashes - the Protoss names of areas in which there are dark templar.

The input will always be valid and in the format described.

Output

Output should be printed on the console. On the only output line print one integer - the number of dark templar within the specified distance from the target area.

Constraints

- $0 < S < 10000$
- $0 < T < 20001$
- $R < 255$
- $C < 255$
- Area name < 300 characters
- Allowed working time for your program: 0.20 seconds. Allowed memory: 16 MB.

Examples

Input example	Output example	Explanation
<pre> 1 5 up upper lo lower le left ri right ce center 3 3 upper-left upper upper-right left center right lower-left lower lower-right ce 3 ce up-le up </pre>	2	<p>The target area is "center"</p> <p>The dark templar are in areas "upper-left", "upper" and "center". Those within a distance of 1 are the templar in "center" and in "upper"</p>
<pre> 1 5 up upper lo lower le left ri right ce center 3 3 upper-left upper upper-right left center right lower-left lower lower-right ce 1 lo-ri </pre>	0	<p>There is only one templar in the area "lower-right" and his distance is more than 1</p>

Problem 2 – Most Common

You are given a list of **N** humans with their characteristics (first name, last name, year of birth, eye color, hair color and height in cm.). Find their most common characteristics.

Input

The input data should be read from the console.

The number **N** will be given on the first line.

On each of the next **N** lines you will be given the 6 human characteristics in the following format: "{first_name} {last_name}, {year_of_birth}, {eye_color}, {hair_color}, {height}". Note that there is a space between the first and the last name and after each comma.

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 1st output line print the most common first name. On the 2nd output line print the most common last name. On the 3rd output line print the most common birth year. On the 4th output line print the most common eye color. On the 5th output line print the most common hair color. On the 6th output line print the most common height.

If there are 2 (or more) most common names or colors, print the lexicographically smallest one.

If there are 2 (or more) most common birth years or height, print the least one.

Constraints

- **N** will be between 8 and 50 000, inclusive.
- The first name, the last name and the colors will contain only small and capital Latin letters with length between 1 and 8 characters, inclusive.
- The birth years will be between 1900 and 2012, inclusive.
- The height will be between 150 and 220, inclusive.
- Allowed working time for your program: 0.40 seconds. Allowed memory: 16 MB.

Examples

Input example	Output example
8 Wayne Rooney, 1985, Blue, Brown, 175 Emma Watson, 1990, Brown, Brown, 170 Justin Bieber, 1994, Hazel, Brown, 170 Megan Fox, 1986, Blue, Black, 163 Selena Gomez, 1992, Brown, Brown, 165 Dimitar Berbatov, 1981, Blue, Black, 188 Robyn Fenty, 1988, Green, Blond, 173 Justin Timblake, 1981, Blue, Brown, 182	Justin Berbatov 1981 Blue Brown 170

Problem 3 – Phonebook

A Web site hosts a **phonebook**. Each phonebook entry has **name** and **list of phone numbers**. The Web site allows **adding phonebook entries**, **changing phone numbers** and **listing the phonebook entries with paging**.

Phone numbers can be given in any standard form, e.g. "(02) 981 22 33", "123", "(+1) 123 456 789", "0883 / 456-789", "0888 88 99 00", "888-88-99-00", "+359 (888) 41-80-12", "00359 (888) 41-80-12" or "+359527734522" and should be stored in the system in their canonical form. The canonical form for given phone number is obtained just like in the traditional GSM devices. First all non-digit characters except "+" are removed. After that, if the phone starts with "00", the "00" is replaced with "+". After that, any leading zeroes are removed. Finally the default country code "+359" is inserted in the beginning if the phone number if no country code is available (if the phone does not start with "+").

You are assigned to create a program that executes a sequence of commands against the phonebook. Each command consists of a single text line and produces zero, one or more text lines. The commands are described below:

- **AddPhone(name, phone1, phone2, ...)** – adds a new entry to the phone book. The entry should specify **name** and **list of phones** (at least 1 and at most 10). The names in the phonebook are unique (duplicates are not accepted) and case-insensitive. Adding phones for some name always performs **merging**: only the non-repeating canonical phones enter in the list of phones. The command outputs "**Phone entry created**" as a result when the name is missing in the phonebook and "**Phone entry merged**" when the name already exists in the phonebook.
- **ChangePhone(oldNumber, newNumber)** – changes the specified old phone number in all phonebook entries with a new one. The command prints "**X numbers changed**" as a result, where **X** is the number of the changed old phone numbers in the system. Changing a phone number works with **merging** and thus any duplicating phone numbers are omitted.
- **List(start, count)** – lists the phonebook entries with paging. The page is specified by **start index** and **count** in the phonebook assuming that the entries are sorted by name (case-insensitive). The start index is zero-based. The **count** specifies the page size (the number of phonebook entries to be retrieved). The listed phonebook entries should be printed in the form "**[name: phone1, phone2, ...]**", each on a separate line. The name should appear in the same casing as when it was first added to the phonebook. The phone numbers should be sorted alphabetically (as text). In case the start index is invalid or the count is invalid, the command prints "**Invalid range**".
- **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 output data should be printed on 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 **name** will be non-empty English text (less than 200 characters) and cannot contain ",", ":", and "\n", as well as leading or trailing whitespace. Names are case-insensitive, e.g. "Peter" is the same as "peter" and "PETER".
- The **phone numbers** will contain only digits, whitespace and the special characters "+", "-", "/", "(" and ")". Phone numbers cannot contain ",", and "\n", leading or trailing whitespace. Phone numbers always have of [3..50] digits. Phone numbers could have at most one leading zero.
- The **start** will be integer number in the range [0...1000000].
- The **count** will always be integer number in the range [1...20].
- There is a single space after each "," in the input and no space before it. There are no spaces around the "(" and ")" in the commands.
- The input cannot exceed **2 MB**.
- Allowed working time for your program: 1.50 seconds.
- Allowed memory: 16 MB.
- **Important: Please use StringBuilder to store your output and print it at the end of the input.**

Examples

Input example	Output example
<pre>AddPhone(Kalina, 0899 777 235, 02 / 981 11 11) AddPhone(kalina, +359899777235) AddPhone(KALINA, (+359) 899777236) AddPhone(Kalina (new), 0899 76 15 33) List(0, 1) List(10, 10) ChangePhone((+359) 899 777236, 0883 22 33 44) AddPhone(Zhoro.Telerik, 0893 656 756) AddPhone(Alex St.Zagora, 042 77 33 55) ChangePhone(0893 656 756, 0898 123 456) List(1, 3) ChangePhone(042 77 33 55, 02 98 11 111) ChangePhone((02) 9811111, 088 322 33 44) List(0, 2) ChangePhone((02) 9811111, 088 322 33 44) End</pre>	<pre>Phone entry created Phone entry merged Phone entry merged Phone entry created [Kalina: +35929811111, +359899777235, +359899777236] Invalid range 1 numbers changed Phone entry created Phone entry created 1 numbers changed [Kalina: +35929811111, +359883223344, +359899777235] [Kalina (new): +359899761533] [Zhoro.Telerik: +359898123456] 1 numbers changed 2 numbers changed [Alex St.Zagora: +359883223344] [Kalina: +359883223344, +359899777236] 0 numbers changed</pre>

Problem 4 – LinkedOut

In the well-known social network LinkedIn (<http://linkedin.com>) there are people connected to each other. All connections are two-way connections. That means that if you are connected to UserX, then UserX is also connected to you.

On LinkedIn, there are degrees of connections (1st-degree, 2nd-degree, 3rd-degree, 4th-degree and etc.).

1st-degree – All people you're directly connected to.

2nd-degree – All people who are connected to your 1st-degree connections and are not connected to you.

3rd-degree – All people connected to your 2nd-degree (and not to your 1st-degree) connections and etc.

Generally, your *ith*-degree connections are connected to your **(i-1)th**-degree connections and are not connected to any other degree connection that is lower than **i-1**.

If two users are not connected through any other connections then their degree is **-1**.

Your best friend wants to create similar social network, called LinkedOut. In his network there are exactly **K** users and **N** connections between these users. Since he doesn't know much about algorithms, he can't solve one of the most important problems related to such social networks. The problem is to find the degree of the connection from one user to another.

Write a program for your best friend's social network (LinkedOut) that answers the question: "*What is the degree of the connection between UserX and UserA, UserB, UserC, ..., etc.?*" Your program will receive information about the connections in the entire network, along with the UserX and the users UserA, UserB, UserC ..., etc.

Input

The input data should be read from the console.

On the first line there will be the name of the user whose connections degrees you have to find (UserX).

On the second line there will be the number **N** of the connections in the LinkedOut social network. On each of the next **N** lines there will be one connection between two users, separated by a single space.

On the next line there will be the number **M** of the contacts degrees you should check. On each of the next **M** lines you will get the name of a user, to which you have to find the connection degree from the user on the first line (UserX).

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.

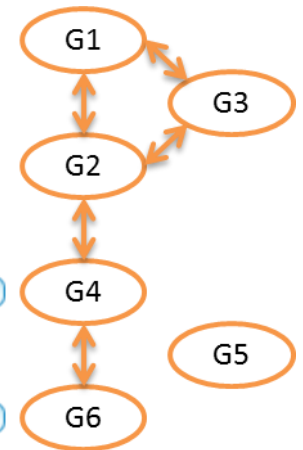
You should print the connection degree of each of the **M** users on a single line in the same order they were given in the input. That is, if the first of the **M** users is Georgy, then on the first output line you have to print the connection degree to Georgy.

Constraints

- **K** will be between 1 and 1000. **N** will be between 0 and 100 000. **M** will be between 1 and 999.
- The usernames in the LinkedOut system will consist of only Latin letters (A-Z, a-z) and digits (0-9) and their length will be between 2 and 20 symbols, inclusive.
- Allowed working time for your program: 0.25 seconds. Allowed memory: 16 MB.

Examples

Input example	Output example
Nikolay 3 Doncho Georgy Doncho Nakov Nikolay Nakov 3 Georgy Vasil Ivan	3 Nikolay -1 ↓ -1 Nakov ↓ 2nd Doncho ↓ 3rd Georgy
Nikolay and Georgy are in 3 rd degree contact (as shown in the picture). The other 2 contacts have nothing in common with Nikolay.	

Input example	Output example
G1 5 G4 G6 G1 G3 G1 G2 G2 G3 G2 G4 5 G2 G3 G4 G5 G6	1 1 2 -1 3 2nd 3rd 

Problem 5 – Buy Graphs

In the Pernik village there is a shop that sells simple planar graphs.

A simple graph is an ordered pair (V, E) where V is a finite non-empty set of objects called vertices, and E is a finite set of edges. Each edge is a two-element subset of V (you can find drawings of several graphs in the test case 4 from the example below).

Note that a simple graph does not contain any loops (edges that connect a vertex to itself) and any duplicate edges. In other words, each edge connects two different vertices, and each pair of vertices is connected by at most one edge.

A graph is called planar if it has a drawing in the plane such that no two edges intersect.

Note that a simple planar graph does not have to be connected.

The cost of any graph with **X** vertices and **Y** edges is $(X^3 + Y^2)$ gold coins.

Ilian has **N** gold coins, and he wants to spend **all of them** on simple planar graphs.

Write a program that gets the number **N** and computes the minimum number of simple planar graphs Ilian has to buy in order to spend **exactly N** gold coins. He is allowed to buy multiple graphs of the same type.

Input

The input data should be read from the console.

On the first input line there will be one number **T** – the number of the test cases your program should solve.

On each of the next **T** lines you will be given a single integer number **N** – the number of the gold coins Ilian has.

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 **T** test cases write a single line containing the minimum number of simple planar graphs Ilian has to buy in order to spend all of his **N** gold coins.

Constraints

- **T** will be an integer number between 2 and 5, inclusive.
- **N** will be an integer number between 1 and 50 000, inclusive.
- Allowed working time for your program: 0.50 seconds.
- Allowed memory: 16 MB.

Example

Input example	Output example	Explanation
4 36 7 72 46	1 7 2 3	<p>Test case 1: For 36 gold coins Ilian can buy a triangle: a simple planar graph with 3 vertices and 3 edges.</p> <p>Test case 2: The only simple planar graph that costs 7 gold coins or less is the graph that consists of a single vertex (and no edges). This graph costs $1^3 + 0^2 = 1$, so Ilian has to buy 7 of them.</p> <p>Test case 3: He can buy 2 triangles for 36 gold coins each. No simple planar graph costs exactly 72 gold coins, hence the optimal answer in this case is 2</p> <p>Test case 4: One optimal solution is to buy graphs worth 1 + 9 + 36 gold coins. All the graphs Ilian can afford are shown in the following picture:</p> <div style="display: flex; flex-wrap: wrap; justify-content: space-around;"> <div style="border: 1px dashed gray; padding: 5px; margin: 5px;"> <p style="text-align: center; color: red; font-weight: bold;">1</p>  <p style="text-align: center;">$(X, Y) = (1, 0)$</p> </div> <div style="border: 1px dashed gray; padding: 5px; margin: 5px;"> <p style="text-align: center; color: red; font-weight: bold;">8</p>  <p style="text-align: center;">$(X, Y) = (2, 0)$</p> </div> <div style="border: 1px dashed gray; padding: 5px; margin: 5px;"> <p style="text-align: center; color: red; font-weight: bold;">9</p>  <p style="text-align: center;">$(X, Y) = (2, 1)$</p> </div> <div style="border: 1px dashed gray; padding: 5px; margin: 5px;"> <p style="text-align: center; color: red; font-weight: bold;">27</p>  <p style="text-align: center;">$(X, Y) = (3, 0)$</p> </div> <div style="border: 1px dashed gray; padding: 5px; margin: 5px;"> <p style="text-align: center; color: red; font-weight: bold;">28</p>  <p style="text-align: center;">$(X, Y) = (3, 1)$</p> </div> <div style="border: 1px dashed gray; padding: 5px; margin: 5px;"> <p style="text-align: center; color: red; font-weight: bold;">31</p>  <p style="text-align: center;">$(X, Y) = (3, 2)$</p> </div> <div style="border: 1px dashed gray; padding: 5px; margin: 5px;"> <p style="text-align: center; color: red; font-weight: bold;">36</p>  <p style="text-align: center;">$(X, Y) = (3, 3)$</p> </div> </div>