Problem 1 – Tasks

You love to solve algorithmic tasks. We know. We also know that you prefer to solve the easiest problems first, and if two problems are equally hard then you prefer to solve the one that is lexicographically smaller (comparing ASCII codes char by char). We give you tasks to solve sometimes and hopefully you solve them. If you don’t have enough time to solve the tasks you keep them in a list and eventually solve them later (starting from easiest one), right? If you want to solve tasks but your list of tasks to solve is empty, then you just rest.

Your task now is to write a program that executes 2 commands:

- Adding a task to your list of tasks for solving:
  The format of the command is "**`New {Complexity} {Task_name}`**", where the three parts of the command are separated by a single space. In this command {Complexity} is the complexity of the task, given as an integer number between 1 and 1 999 999 999, inclusive. Task with complexity 3 is harder than task with complexity 2. {Task_name} is the name of the task that you should solve. The task name will contain small ('a' - 'z') and capital ('A' - 'Z') Latin letters, spaces, dashes ('-') and digits ('0' - '9'). The length of the task name will be between 1 and 6 letters, inclusive. The task name will contain no trailing spaces.
- Solving one task from you list of waiting tasks:
  The format of the command is "**`Solve`**". When executing this command you take easiest of the tasks in your list and solve it. If there is more than one easiest task (with equally smallest complexity) then you solve the task with lexicographically smallest name. If no tasks are available in your list then you are taking a rest.

Input

The input data should be read from the console.

On the first input line you will be given the number of the command lines – **N**.

On each of the next **N** lines there will be a single command (`New` or `Solve` in the described format).

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 "`Solve`" commands output a single line containing the name of the task you are solving, or output “`Rest`” if there is no task that you can solve.

Constraints

- **N** will be between 10 and 50 000, inclusive.
- At least one of the commands will be "`Solve`".
- Allowed working time for your program: 0.25 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
10 New 5 T- K New 8 T- N New 8 T- 11 New 8 T- 1 New 8 T -- Solve Solve Solve Solve New 1999999999 Task	T- K T -- T- 1 T- 11

Input example	Output example
10 New 4 t 1 New 5 T2 New 4 t-1 Solve Solve Solve Solve New 1 t 1 New 1 t 1 Solve	t 1 t-1 T2 Rest t 1

Problem 2 – Terran

Some years after the Brood war, the Terran arsenal was enhanced with an extremely lethal air-to-ground attack unit - the banshee. Besides the powerful Backlash rockets, which can decimate ground forces in a few volleys, banshees have the ability of "cloaking". When a banshee enters "cloaked" mode, it becomes invisible to the naked eye and is undetectable by most radars. Which, as most other terran ideas, has both it's obvious advantages and not so-obvious disadvantages.

It turned out it was too complicated to mount detection hardware on banshees, so basically two banshees cannot see each other when they are cloaked. Also, the cloaking system severely shortens the range of radio contact between two cloaked banshees. That turned out to be a problem, as banshees are most effective when they attack in so called "attack groups".

An "**attack group**" of banshees is a **group of all banshees**, in which **every two banshees can in some way communicate** with each other. **Communicating doesn't need to be direct** - every **banshee can receive a message and send it to any other banshee** in its radio range. **That banshee can again send the message to** all **other banshees in its range** and so on. That is, if banshee A is not in range with banshee C, but banshee B is in range with both banshee A and banshee C, then banshee A and banshee C can communicate.

Furthermore, Terran command centers can receive position information from each and every banshee. Banshee positions are **2D points in the Cartesian coordinate system**. The distance between two banshees is the standard **Euclidean (a.k.a. Cartesian) distance** between their locations**.** Terran commanders **need software**, which **by given banshee locations finds the count of the largest possible attack groups.**

Of course, we can't let the enemy know where our banshees are. That's why banshee locations are encoded in a special way:

- Each banshee coordinate (X and Y) is turned into a sum of numbers (for example the location (6, 10) is represented by the sum 2 + 1 + 0 + 6 for X and the sum 3 + 9 + 2 + (-4) for Y)
- Each number of the sum is turned into a codeword of 4 English letters (there are predefined code words for all of the possible numbers in the sum)
- The code words for the X coordinate are concatenated (made into one string). The same thing goes for the code words for the Y coordinate

Write a program, which by given maximum banshee radio range, code words and their corresponding numbers, and banshee locations encoded with those code words, finds the count of the largest possible attack groups.

Input

The input data should be read from the console.

On the first line there is one double precision floating point (**double**) number - the maximum banshee radio range

On the next line there will be one integer C - the number of code words

On each of the next C lines there will be one string - the code word, and one integer - the meaning of the code word (integers can be from -1000 to 1000 inclusive)

On the next line there will be one integer B - the number of banshees in the coordinate system

On each of the next B lines there will be two strings - the encoded X coordinate and the encoded Y coordinate, separated by a single space.

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 one integer - the count (number) of the largest attack groups of banshees.

Constraints

- 0 < **C** < 500, 0 < **B** < 500
- Minimal coordinates (-20 000, -20 000). Maximal coordinates (20 000, 20 000).
- Encoded coordinate string length < 300 characters
- Allowed working time for your program: 0.10 seconds. Allowed memory: 16 MB.

Examples

Input example	Output example	Explanation
1 3 aaaa 1 bbbb 2 cccc -1 7 aaaacccc ccccaaaa aaaacccc aaaa aaaa aaaa bbbb bbbbaaaacccc bbbbaaaa bbbb bbbbaaaa aaaa bbbbbbbbbbbbbbbbbbbb bbbbbbbbbbbbbbbbbbb	3	The groups of banshee coordinates are: 1. (0, 0), (0, 1), (1, 1); 2. (2, 2), (3, 2), (3, 1); 3. (10, 10); 1. - The distance between (0, 0) and (1, 1) is ~1.42, but the distance from (1, 0) to (1, 1) and (0, 0) is 1, so this is one group

1 3 aaaa 1 bbbb 2 cccc -1 2 aaaacccc ccccaaaa aaaa aaaa	2	Here there are two banshees at coordinates (0, 0) and (1, 1). The distance between them is ~1.42, which is more than 1, so they cannoc communicate and form 2 separate groups

Problem 3 – Play with Krisko

Krisko is playing a new game he received as a reward for his MCTS certification. The game takes place on an acyclic, undirected graph, with sandwiches located on some of the nodes. The number of the nodes in the graph is **N** and the nodes are numbered from 0 to N-1, inclusive.

On each turn, Krisko picks a node with at least 2 sandwiches on it. He then picks up 2 sandwiches from that node, eats one of them, and places the other sandwich on an adjacent node. If at any time there is a sandwich on the target node (numbered **X**), then the game is over and Krisko wins. If he cannot put sandwich on that node through any sequence of moves, he loses. **Krisko is smart, and so if there is a winning sequence of moves he will find it.**

You enjoy frustrating Krisko and want to make him lose the game.

Your task is to write a program that finds the maximum number of sandwiches that can be placed on the board without Krisko being able to win. If more than 2 000 000 000 sandwiches can be placed on the board without Krisko winning, print -1 instead.

Input

The input data should be read from the console.

On the first line you should read **T** – the number of the test cases your program should solve.

Each of the **T** test cases will be in the following format:

On the first line of each test case you will be given the numbers **N** and **X**, separated by a single space.

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 nodes **i** and **j** are connected in the graph, and '**0**' otherwise. See the examples

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** tests cases given in the input write a single line, containing the maximum number of sandwiches that can be placed on the board without Krisko being able to win (or -1 if the number is more than 2 000 000 000).

Constraints

- **T** will be between 4 and 10, inclusive.
- **N** will be between 1 and 50, inclusive.
- **X** will be between 0 and N-1, inclusive.
- For each **i** and **j**, **G[i][j]** will be equal to **G[j][i]**.
- For each **i**, **G[i][i]** will be equal to '0'.
- The graph represented by **G** will have no cycles.
- Allowed working time for your program: 0.10 seconds.
- Allowed memory: 16 MB.

Example

Input example	Output example	Explanation
4 3 1 010 101 010 3 2 010 101 010 4 0 0111 1000 1000 1000 4 1 0111 1000 1000 1000	2 3 3 4	**First test case:** The graph is a straight line: 0-1-2 With node 1 as the target, we can only put one sandwich each on nodes 0 and 2. If you place a second piece on either, Krisko can eat one and move the other to node 1. **Second test case:** The same graph as previous test case, but now node 2 is the target. The optimal strategy places 3 sandwiches on node 0.

Problem 4 – Free Content

A Web site hosts a **catalog of free content**: **books, movies, music and software**. All items in the catalog have title, author, size and URL. The Web site allows **adding, updating and searching** the content.

You are assigned to create 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:

- **Add book: title; author; size; url** – adds a book. If the book already exists, it is added again (duplicates are accepted). The result of the command execution is “**Book added**”.
- **Add movie: title; author; size; url** – adds a movie. If the movie already exists, it is added again (duplicates are accepted). The result of the command execution is “**Movie added**”.
- **Add song: title; author; size; url** – adds a song. If the song already exists, it is added again (duplicates are accepted). The result of the command execution is “**Song added**”.

- **Add application: title; author; size; url** – adds a software application. If the application already exists, it is added again (duplicates are accepted). The result of the command is “**Application added**”.
- **Update: oldUrl; newUrl** – updates the URL of all items matching the specified old URL. The result of the command execution is “**X items updated**” where **X** is the number of matched items.
- **Find: title; count** – lists the items (books / movies / music / software) matching the given title. The number of the listed items should be the given **count** or less (if not enough items are available). If no items are available, the command should output "**No items found**". Each item should be printed on a separate line in format "**Book: title; author; size; url**" / "**Movie: title; author; size; url**" / "**Song: title; author; size; url**" / "**Application: title; author; size; url**". The listed items should by in ascending order by their text representation in the specified format.
- **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 **title** and **author** will be non-empty English text (less than 1000 characters) and cannot contain "**;**" and "**\n**", as well as leading or trailing whitespace.
- The **size** will always be integer number in the range [1…10000000000] and will consist of digits only (without any separators between the digits).
- The **url** will always be valid URL address (according to the standard RFC 1738, less than 2048 characters) and will not contain the "**;**".
- The **count** will always be integer number in the range [1…100].
- There is a single space after each "**:**" and "**;**" separator in the input and no space before it.
- All the text in the catalog (titles, authors and URLs) is case-sensitive, e.g. "*New Era*" is different from "*new era*".
- The input cannot exceed **5 MB**.
- Allowed working time for your program: 2.70 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
Find: One; 3 Add application: Firefox v.11.0; Mozilla; 16148072; http://www.mozilla.org Add book: Intro C#; S.Nakov; 12763892; http://www.introprogramming.info Add song: One; Metallica; 8771120; http://goo.gl/dIkth7gs Add movie: The Secret; Drew Heriot, Sean Byrne & others (2006); 832763834; http://t.co/dNV4d Find: One; 1 Add movie: One; James Wong (2001); 969763002; http://www.imdb.com/title/tt0267804/ Find: One; 10 Update: http://www.introprogramming.info; http://introprograming.info/en/ Find: Intro C#; 1 Update: http://nakov.com; sftp://www.nakov.com End	No items found Application added Book added Song added Movie added Song: One; Metallica; 8771120; http://goo.gl/dIkth7gs Movie added Movie: One; James Wong (2001); 969763002; http://www.imdb.com/title/tt0267804/ Song: One; Metallica; 8771120; http://goo.gl/dIkth7gs 1 items updated Book: Intro C#; S.Nakov; 12763892; http://introprograming.info/en/ 0 items updated

Problem 5 – Nakov Number

The Nakov number is a way of describing the "collaborative distance" between a scientist and Svetlin Nakov by authorship of publications.

Svetlin Nakov is the only person who has a Nakov number equal to zero. To be assigned a finite Nakov number, a scientist must publish a paper in co-authorship with a scientist with a finite Nakov number. The Nakov number of a scientist is the lowest Nakov number of his coauthors + 1. The order of publications and numbers assignment doesn't matter, i.e., after each publication the list of assigned numbers is updated accordingly.

You will be given a list of publications (with **N** elements), each element of which describes the list of authors of a single publication and is formatted as "$AUTHOR_1$ $AUTHOR_2$ ... $AUTHOR_N$" (enclosed by quotes and separated by space) in the input data.

Return the list of Nakov numbers which will be assigned to the authors of the given publications.

Input

The input data should be read from the console.

On the only input line there will be the list of publications. The publications will be separated by a comma and a space (", ") and will be enclosed by quotes. See the examples bellow.

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.

The output lines must consist of a list of all authors with their Nakov numbers.

Each line of the output should be formatted as "AUTHOR NUMBER" (without the quotes, separated by space) if AUTHOR can be assigned a finite Nakov number, and "AUTHOR -1" (without the quotes and separated by space) otherwise.

All authors mentioned in the publications list must be present in the output.

The authors in the output must be ordered lexicographically.

Constraints

- **N** will be between 1 and 60, inclusive.
- Each publication will contain between 1 and 60 characters, inclusive.
- An author is a string of between 1 and 60 uppercase Latin letters ('A'-'Z'), inclusive.
- Each publication will be a list of authors, separated by single spaces.
- The authors in each publication will be distinct.
- There will be at most 100 distinct authors in all publications.
- Svetlin Nakov will be given as "NAKOV", and at least one publication will list him as one of the authors.
- Allowed working time for your program: 0.15 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
"KOLEV GEORGIEV", "NAKOV KOLEV"	GEORGIEV 2 KOLEV 1 NAKOV 0
"NAKOV B", "A B C", "B A E", "D F"	A 2 B 1 C 2 D -1 E 2 F -1 NAKOV 0