2018 Denison Spring Programming Contest
Granville, Ohio
24 February, 2018

Rules:

1. There are **six** problems to be completed in **four hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. No whitespace should appear in the output except between printed fields.
4. All whitespace, either in input or output, will consist of exactly one blank character.
5. The allowed programming languages are C, C++, Python 2, Python 3, and Java.
6. All programs will be re-compiled prior to testing with the judges' data.
7. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contest officials (e.g., that might generate a security violation).
8. The input to all problems will consist of multiple test cases.
9. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
10. All communication with the judges will be handled by the PC^2 environment.
11. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: Calendar Counting

The Arcturan Banking Corporation (ABC) has run into a bit of a calendaring problem. Its banks on different planets across the galaxy use the local calendar system and it's becoming really hard to keep track of loan durations. You have been employed by ABC to automate the process of counting the number of days between two dates in any calendar system.

Fortunately, all calendar systems coincidentally have the same form. Each year is split into months of varying lengths. The length of each month is the same each year—the civilized galaxy doesn't bother with leap days—and the length of the year is simply the sum of the days in all the months in that year. For example, a planet may have months called Winter, Spring, Summer, Autumn, each of which has 25 days, so that a year is 100 days long. Your task is to count the number of local days between a pair of given dates.

Input

The first line of each input will consist of an integer m indicating the number of months in the local calendar system ($1 \leq m \leq 20$). (A value of $m = 0$ will indicate the end of the input.) The next m lines will each consist of (in chronological order) the name of the month in the year followed by its length. The names will be strings with no whitespaces and the length can be between 1 and 100 days (inclusive). The next two lines will be two dates in this calendar system, given in the order: year, month, and day of the month. The first date will always precede the second. You may assume that the date is always in a valid format—the month will be valid and the day of the month will be between 1 and the length of that month. The year will be an integer between -10000 and 10000 (inclusive).

Output

For each input, you should output the case number and the number of days from the first to the second date in the format shown in the Sample Output.

Sample Input

```
4
Winter 25
Spring 25
Summer 25
Autumn 25
3000 Summer 25
3000 Autumn 1
12
Naj 31
Bef 28
Ram 31
R'pa 30
Yam 31
Nuj 30
Luj 31
Gua 31
Pes 30
T'co 31
```

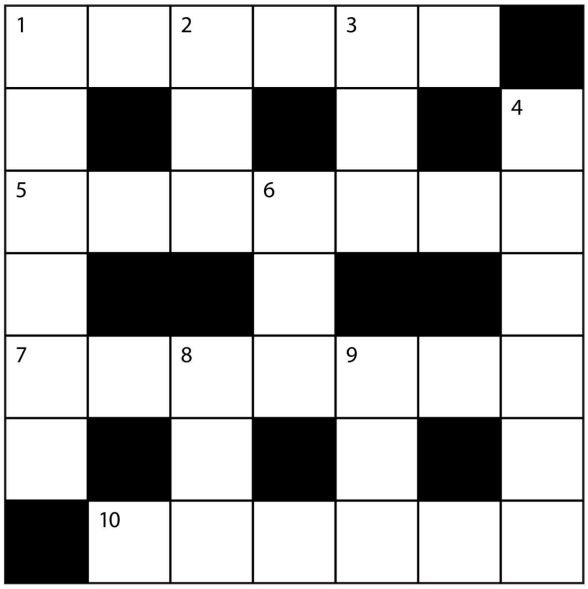
```
Von 30
Ced 31
2017 Naj 1
2018 Ced 31
5
One 1
Two 2
Three 3
Four 4
Five 5
1000 Three 2
2000 Two 1
0
```

Sample Output

```
Case 1: 1
Case 2: 729
Case 3: 14997
```

Problem B: Crossword

To solve a crossword puzzle, you have to fill in all the words going across and down. An example crossword puzzle grid is shown below:



Clue numbers are inserted into white squares in the grid that have a clue (a two or more letter sequence of white squares going across or down) going left to right in each row, starting from the top row. For this problem, you have to determine the position of a particular clue number in the crossword grid. For example, in the grid above, clue 5 is at row 3, column 1 (3, 1) and clue 4 is at row 2, column 7 (2, 7).

Input

Each input will consist of multiple lines. The first line will contain r c n, where r is the number of rows in the grid ($1 \leq r \leq 100$), c is the number of columns in the grid ($1 \leq c \leq 100$), and n is the clue number ($1 \leq n \leq 100$). The following h lines of the clue will give the h lines of the grid, with dash/minus ($-$) representing blank/white spaces and uppercase X representing black spaces as shown in the Sample Input. The input will end with the line 0 0 0.

Output

For each input, you should output the case number and the position of this clue in the format shown in the Sample Output. Note that row and column numbering start at 1. If the clue does not exist, you should output DNE.

Sample Input

```
7 7 5
------X
-X-X-X-
```

```
-------
-XX-XX-
-------
-X-X-X-
X------
7 7 4
------X
-X-X-X-
-------
-XX-XX-
-------
-X-X-X-
X------
7 7 11
------X
-X-X-X-
-------
-XX-XX-
-------
-X-X-X-
X------
0 0 0
```

Sample Output

```
Case 1: 3 1
Case 2: 2 7
Case 3: DNE
```

Problem C: Getting Out of Hand

In trading card games (such as *Hearthstone*), each player has a hand of cards with varying mana costs and a mana total that they are allowed to use each turn. The player can play any combination of cards in her hand so long as the sum of the mana costs is at most the mana total for the turn. For this problem, you will count the total number of possible plays that the player has on a given turn.

For example, suppose that the player has the cards Defile (mana cost 2), two copies of Hellfire (mana cost 3), and Doomguard (mana cost 5) on a 6 mana turn. Then, the player has six options: (1) play no cards, (2) play Defile alone, (3) play one Hellfire, (4) play Defile and one Hellfire, (5) play two Hellfires, or (6) play the Doomguard.

Input

Each input will be over multiple lines. The first line of the input will have n m, where n is the number of unique cards in the hand and m is the mana total for the turn ($1 \leq n \leq 100$, $1 \leq m \leq 100$). (A line with 0 0 will indicate the end of input.) The next n lines will have the information for the n different types of cards. Each line will contain two entries c r, where c is its mana cost of that card and r is the number of copies of that card that are in the hand ($0 \leq c \leq 10$, $1 \leq r \leq 10$).

Output

For each input, you should output the case number and the number of different plays possible for that hand as shown in the Sample Output. Since the number of combinations can get very large, you should give the answer modulo 1 million (1000000).

Sample Input

```
3 6
2 1
3 2
5 1
1 10
1 10
4 8
2 1
2 1
2 1
2 1
0 0
```

Sample Output

```
Case 1: 6
Case 2: 11
Case 3: 16
```

Problem D: Hashtag Segmentation

Twitter hashtags aren't allowed to have spaces in them which can cause ambiguity. For example, a hashtag such as `#denisonproud` can be segmented to read as "denison proud" or "den i son proud." Your task is to write a program that takes as input a hashtag and a dictionary of words and outputs the number of ways the hashtag can be split into words in the dictionary.

For example, for the hashtag `#statefarmisthere` and a dictionary of words that includes state, farm, is, there, far, mist, and here, there are two possible segmentations: "state farm is there" and "state far mist here."

Input

Each input will be over multiple lines. The first line of the input will have the number of words w ($1 \leq w \leq 100$) in the dictionary. A value of zero will indicate the end of input. The next w lines will consist of words in the dictionary, one per line. These words may be in any order and there will be no repeated words. Finally, the last line of each input will have the hashtag (of length at most 100 characters). The hashtag may have repeated instances of words in the dictionary.

Output

For each input, you should output the case number and the number of ways the hashtag can be read in the format shown in the Sample Output. The answer in each case will not exceed two billion.

Sample Input

```
5
den
i
son
proud
denison
denisonproud
7
state
farm
is
there
far
mist
here
statefarmisthere
3
a
i
ai
aiaiai
0
```

Sample Output

```
Case 1: 2
Case 2: 2
Case 3: 8
```

Problem E: Kevin Bacon

You've likely heard of the six degrees of Kevin Bacon. The conjecture is that almost all actors can be connected back to Kevin Bacon using six or fewer films. Two actors are connected if they have acted in the same film.

We will call a path from some actor to Kevin Bacon a minimal path if it uses the minimal number of films possible. Of course, there may be more than one minimal path. Some actors are connected more strongly; we will say actors A and B are strongly connected if for any minimal path connecting A and B and any film F on that path there is another minimal path connecting A and B that does not contain film F.

For this problem, you will determine how many actors are strongly connected to Kevin Bacon. For example, Kevin Bacon and Mark Wahlberg co-starred in *Patriots Day* and Mark Wahlberg co-starred in *Deepwater Horizon* with Kurt Russell, giving a minimal path of length two between Kevin Bacon and Kurt Russell. Kurt Russell is also strongly connected to Kevin Bacon since he was in *Forest Gump* (as the voice of Elvis) with Tom Hanks and Tom Hanks was in *Apollo 13* with Kevin Bacon, making a second minimal length path between the actors. Since removing any single movie does not change the distance between Kevin Bacon and Kurt Russell, we can see that they are strongly connected.

Your task is to count the number of actors that are strongly connected to Kevin Bacon.

Input

Each input will be over multiple lines. The first line of the input will have the total number of films N ($1 \leq N \leq 100$). (An input of $N = 0$ will indicate the end of input.) The next N lines will give the cast of the movie in the following format: M F $ACTOR_1$ $ACTOR_2$ ... $ACTOR_M$, where M ($1 \leq 10 \leq M$) is the number of actors in the film, F is the name of the film, and $ACTOR_1$ $ACTOR_2$ ... $ACTOR_M$ is the cast of actors in that film. Films and actor names are upper case strings separated by single spaces. The spaces normally in names will be an underscore character. `KEVIN_BACON` will always be an actor in at least one film. There will be at most 100 distinct actors in any test case.

Output

Output will consist of the case number followed by the number of actors other than Kevin Bacon strongly connected to him as in the Sample Output.

Sample Input

```
4
5 APOLLO_13 TOM_HANKS KEVIN_BACON BILL_PAXTON GARY_SINISE ED_HARRIS
5 PATRIOTS_DAY MARK_WAHLBERG JK_SIMMONS KEVIN_BACON JOHN_GOODMAN MICHELLE_MONAGHAN
2 DEEPWATER_HORIZON MARK_WAHLBERG KURT_RUSSELL
3 FOREST_GUMP TOM_HANKS ROBIN_WRIGHT KURT_RUSSELL
0
```

Sample Output

```
Case 1: 1
```

Problem F: Parking

As you arrive at work each day, you try to park in your favorite spot. The parking lot is a linear array of parking locations, each with its own number. Sometimes, when you arrive at your spot, it is already taken by another car. Thus you attempt to park in the next open spot available. You will always look forward (increasing parking spot numbers) from your favorite spot; you will never drive your car backwards in the lot even if you saw an empty spot previously. If you are unable to find an empty spot after a little bit of searching, you give up and leave the lot to park elsewhere. The goal of this problem is to model several people attempting to park in this same way to determine how many of them can fit in the lot.

Input

Each input consists of two lines of data. The first line contains two integers m and n ($1 \leq m, n \leq 1000$). The parameter m indicates how many spots are in the lot, numbered 0 to $m-1$ in sequence. The parameter n indicates how many employees will arrive attempting to park in the lot. The lot will be initially empty at the start of the day.

The second line consists of n pairs of integers, separated by spaces. Each pair is given as s_i and k_i. The i^{th} person arriving to the lot will first attempt to park in spot s_i, ignoring any open spots before then. If spot s_i is full, then person i will look at the next k_i consecutive spots to see if one of them is open, taking the first open spot they encounter. If none of these next k_i spots are available, person i leaves the lot, even if there are other open spots before or after.

Note that $0 \leq s_i \leq m-1$, $0 \leq k_i$ for all i and that $s_i + k_i \leq m-1$ for all i. You should process the employees *in the order* they appear, as the order will affect who can and cannot park.

For example, if a person arrives with $s_i = 5$ and $k_i = 3$, then they will first attempt to park in Spot 5. If that is full, they look to the next 3 consecutive spots: first Spot 6, then Spot 7, then Spot 8. They don't look at any spots after number 8. If all those are full, they leave the lot.

The input ends with a single line where $m = 0$ and $n = 0$.

Output

For each problem you are to output the number of spots that remain open in this lot after all n employees have attempted to park. The answers should appear one per line, following the formatting of the example output below.

Sample Input

```
6 4
4 1 1 3 4 1 4 1
3 2
2 0 2 0
0 0
```

Sample Output

```
Case 1: 3
Case 2: 2
```