17th Annual Denison Spring Programming Contest
25 February 2006

Rules:

1. 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.
2. All programs will be re-compiled prior to testing with the judges' data.
3. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed.
4. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
5. All communication with the judges will be handled by the PC^2 environment.
6. The allowed programming languages are C, C++ and Java.
7. Judges' decisions are to be considered final.
8. There are **six** questions to be completed in **four hours**.
9. **Important:** No extraneous whitespace should appear in your output. Specifically, there should be no blank lines, unless specifically called for. A line should *never* end with whitespace. Do not use tabs in your output, only spaces. Data fields in an output line should be separated by a single whitespace, unless specified otherwise. Any deviations to these guidelines will result in a "Format Error" from the judges.

Problem A: GPA

All students at Henry's Eclectic Yodelling University (good 'ol HEY U) take four classes each semester. In each class they earn grades ranging from the high of an A to a low of F. With plusses and minuses, it is possible to earn between 4.0 to 0.0 grade points in each class (see below for the table). A student's semester GPA is computed by adding the grade points from the four classes and dividing by 4.

Given a student's semester GPA, you are to determine the number of possible ways in which the student could have earned that GPA. You must account for all different assignment of grades among the four courses that yield that semester's GPA.

All GPA inputs are given by the following notation: $x.y$ where x is the integer part of their semester GPA and y is the fractional part of their GPA in twelfths. For example: 3.8 means 3 and $\frac{8}{12}$.

Assume grade points are assigned as follows (note the F+):

	A 4	A- $3\frac{2}{3}$
B+ $3\frac{1}{3}$	B 3	B- $2\frac{2}{3}$
C+ $2\frac{1}{3}$	C 2	C- $1\frac{2}{3}$
D+ $1\frac{1}{3}$	D 1	D- $\frac{2}{3}$
F+ $\frac{1}{3}$	F 0	

Input

The first line of input contains a positive integer n indicating the number of students to consider. This is followed by n lines each containing a semester GPA.

Output

For each GPA, output a line with of the form:

`Student` s `could earn this GPA in` w `ways.`

where the students are numbered starting at 1 and w indicates the number of different ways the student could have earned that GPA.

Sample Input

```
2
4.0
3.11
```

Sample Output

```
Student 1 could earn this GPA in 1 ways.
Student 2 could earn this GPA in 4 ways.
```

Problem B: Zeros and Ones

Consider strings of characters made up by concatenating any number of the strings 0, 01 or 11. For example 00011111 is one such string, as is 001011, but 1011 is not. [It is interesting that the number of strings of length n that can be constructed in this manner is $\frac{1}{3}(-1)^n + \frac{2}{3}2^n$. These are called the Jacobstahl numbers. We're sure you'll find this interesting to look at once you get back home.] Your job is simply to determine if a given string can be constructed in this manner.

Input

There will be multiple problem sets. Input for each problem will be on one line. Each line (except the last) will be of the form

n s

where n is a non-negative integer no larger than 100 and s is a string of 0's and 1's of length n. A value of $n = 0$ indicates end of input.

Output

Each problem should generate one line of code, either,

`String` m `can be generated.`

or

`String` m `can not be generated.`

accordingly, where m is the number of the problem (stating at 1).

Sample Input

```
8 00011111
6 001011
4 1011
0
```

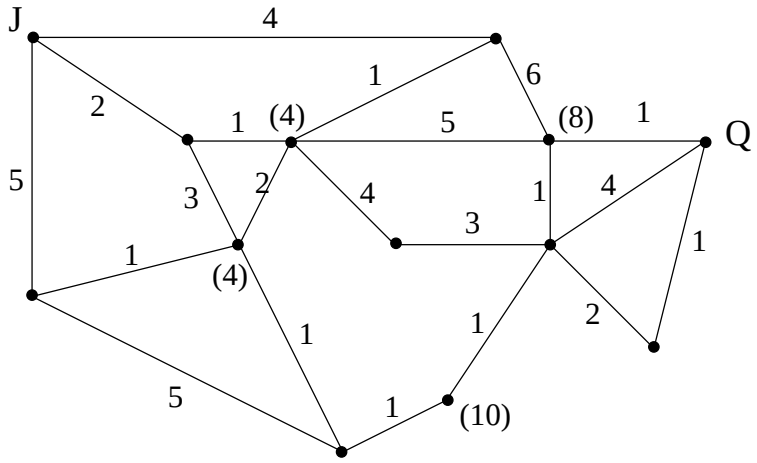
Sample Output

```
String 1 can be generated.
String 2 can be generated.
String 3 can not be generated.
```

Problem C: Spy vs. Spy

The name's Blond, Jane Blond. Her arch-nemisis Sylvester has tied-up Q in the underground sewers of London. Jane has a map of the sewers (and has computed her travel time along each stretch of sewer pipe) but Sylvester has placed time bombs at intersections of tunnels. Jane has stolen information as to where the bombs are and what time each is set to explode. She also knows the location of Q. Once a bomb explodes, that intersection is destroyed and is henceforth impassable. (It goes without saying Jane does NOT want to be at an intersection when the bomb explodes. But because of her finely honed athletic abilities, she will escape unscathed if she misses the explosion by even a second.) Jane wants to get to Q in the least amount of time possible. You are to find that minimal time.

For example, in the following situation, Jane (starting at the intersection marked `J`) can get to Q (marked with `Q`) in a minimum time of 13. The integers on each sewer line indicate travel time and the integers at some nodes (surrounded by parentheses) indicate the time a bomb goes off at that node. Jane starts running at time zero and all times are relative to time 0.



Input

There will be multiple input sets. Each input set will start with a line containing an integer n. (A value of $n = 0$ indicates end of input.) There are $n + 1$ nodes in the input set numbered $0, 1, \cdots, n$. Jane is at node 0 and Q is tied-up at node n. The next line has two integers p and b ($0 \leq b \leq 10$), indicating the number of sewer pipes and bombs, respectively. The next lines will contain p triples n_1 n_2 t indicating there is a pipe from node n_1 to node n_2 that takes time t (< 50) to travel. This information might span many lines, but no triple will be broken across a line. Following these will be b lines each of the form n t indicating there is a bomb at node n set to go off at time t. There will be at most one bomb at any intersection. Assume $n \leq 40$ and the sewer system described is connected.

Output

For each input set, output a line of the form

`Jane can reach Q in time` t.

where t is the minimal time required, if Jane can reach Q at all. If Jane cannot reach Q, output

`Jane cannot reach Q.`

Sample Input

```
12
20 4
0 1 4 0 2 2 3 0 5 2 4 1 2 5 3 5 4 2 1 4 1 1 6 6 4 6 5 4 7 4 7 8 3 9 5 1
3 9 5 6 8 1 9 10 1 10 8 1 8 11 2 11 12 1 8 12 4 6 12 1
5 4
4 4
10 10
6 8
0
```

Sample Output

Jane can reach Q in time 13.

Problem D: Encryption

Here's a scheme for encrypting a message. Let's suppose the message consists of the characters $c_1c_2c_3\cdots$, where each c_i is a lower case letter. Now suppose our key is the letter `'b'`. Assuming the letters are given the values $a = 1, b = 2, \cdots, z = 26$, then to encrypt the first letter c_1, simply add `b` (that is, 2) to the character c_1 (with "wrap around") to get the first encrypted letter e_1. For the remaining letters we perform the same operation but replace the key letter (`b`, in our example) with c_{i-1}. Thus, we get e_2 by adding c_1 to c_2, and e_3 by adding c_2 to c_3, and so on (all with "wrap around", of course).

For example, if our key were `b` and our message were `andawaywego`, then the resulting encrypted message would be `corexxzvblv`. You job is to recover the original message when given the encrypted message and the key.

Input

The first line of input will be an integer n indicating the number of test sets to follow. Each test set will consist of a single lower case letter, indicating the key, followed by a space, followed by a string of between 5 and 40 lower case letters, which is the encrypted message.

Output

Each test case should produce one line of output which is simply the original message. As is usual for encryption, print out the message in groups of 5 (except possibly for the last group) separated by a single space.

Sample Input

```
2
b corexxzvblv
n dhgvysnzvwupbeuok
```

Sample Output

```
andaw ayweg o
progr ammin gisli fe
```

Problem E: Standing In Line

When the students at Henry's Eclectic Yodelling University (good 'ol HEY U) register for their four courses (see the GPA problem) they always seem to spend time standing in a long line in the gym. The students at HEY U have the tradition of making these lines very straight and evenly spaced. So straight, in fact, that it is impossible to look forward or backward in the line except by looking over the heads of those in the line. As a consequence, a short person can't see who else is in line except the persons immediately in front or behind her. She also cannot be seen by others. The question here is to see if one student in line can see another.

Input

There are many test sets for this problem. The first line of a test set contains an integer n (< 100) indicating the number of students to consider. ($n = 0$ indicates the end of input.) This is followed by a line with n positive integers indicating the height, in inches, of each student in line. (Assume these heights are ≥ 55 and ≤ 84.) The following line starts with a positive integer s followed by s pairs of integers indicating a pair of students in line. (Student 1 is the first one in line, student 2 is the second one, etc.) So, the pair 5 36 indicates student 5 and student 36.

Output

For each test case, generate one line of output. This line should start with `Case` k`:` followed by a single space, where k is the number of the test case, where numbering starts at 1. There then follows a series of s answers, either `YES` or `NO`, indicating whether the first student in the pair can or cannot see the other. For our purposes, the eyes of all students are exactly 5 inches from the top of their head. Student A can see student B if the student A can see at least one inch of the top of the student B.

Sample Input

```
4
65 61 62 64
2 1 4 4 1
8
60 60 60 60 60 60 60 61
3 1 4 1 3 8 3
5
55 55 55 80 55
3 1 4 2 4 4 1
0
```

Sample Output

```
Case 1: YES NO
Case 2: NO NO NO
Case 3: YES YES YES
```

Problem F: Eye in the Sky

You're job at *Big Brother Is Watching* (BBIW) is to look at satellite images (maps) and try to find certain suspicious objects. The satellite maps are rectangles of pixels in grey scale, with each pixel a value from 0 through 7. Likewise, each object in a small rectangle of pixels in the same grey scale. The orientation of the objects is the same as on the maps (no need to rotate them) but the picture from the satellite may have been taken at a different time of day or weather conditions and so may be lighter or darker than the prototype image. The relative contrast between pixels will remain the same, however. (So, two pixels whose values in the prototype are 3 and 6 might appear in the amp with values 1 and 4, respectively.) You are to find how many times an object might appear in a map.

For example, below is a map (on the right) and the prototype of two objects on the left. The first object has 3 matches on the map and so might appear 3 times while the second one might appear twice. (Object one has two matches along the left edge of the map — they overlap — and one match 4 columns in from the left. Object two has a match near the center-bottom of the map – top row is `234` — and an identical map near the upper right corner.)

```
32          000000000000
33          100000000123
32          110000000230
            100210234223
123         112220341111
230         102210334111
223         002000000121
```

Input

There will be multiple input sets. Each input set will consist of a map and at least one object. The first line of input will have two integers r and c, indicating the number of rows and columns of the map, respectively. (A value of $r = 0$ will indicate end of input.) There follows r lines, each containing c digits in the range 0 through 7, with no spaces between them. This is the map. You may assume that neither r nor c is greater than 40.

Following this is a line containing n ($1 \leq n \leq 5$) indicating the number of objects to search for. The n lines that follow each contain information about the object. Each line will start with a pair of integers, a b, indicating the number of rows and columns for the object (both values greater than 1 but no more than 6). Following this pair will be the ab pixels giving first row 1, then row 2, etc. There will be a single space following each of a and b but no spaces between the pixel information.

Output

Output for each input set should start with the line `Map` m `information:` where m is the map number, starting at one. Following this is information about each object for that map. Each object should generate the line `Object` p `may appear` q `times.` where p is the number of the object, and q is the appropriate number. Start numbering objects at one for each new map. Following these lines should be the line `***`.

Sample Input

```
7 12
000000000000
100000000123
110000000230
100210234223
112220341111
102210334111
002000000121
2
3 2 323332
3 3 123230223
4 4
0100
1000
1100
0000
2
2 2 1111
2 2 0101
0 0
```

Sample Output

```
Map 1 information:
Object 1 may appear 3 times.
Object 2 may appear 2 times.
***
Map 2 information:
Object 1 may appear 3 times.
Object 2 may appear 0 times.
***
```