Problem 1: Battleship

In the board game “Battleship”, each player secretly arranges a set of ship playing pieces horizontally or vertically on a grid. The grid is labeled with letters A, B, C, ..., J for the rows and numbers 1, 2, ..., 10 for the columns. Players take turns calling out a sequence of missile locations fired onto the grid such as G4, representing row G, column 4. When one player calls out a missile location, the opponent calls out “hit” or “miss” according to whether or not the missile fired hits one of the ships on the grid. For example, suppose ships are represented in the following grid by a sequence of asterisks (*).

```
   1  2  3  4  5  6  7  8  9  10
A  *  *  
B  
C  *  
D  *  
E  *  
F  
G  *  *  *  *  *
H  
I  *  *  *  
J  *
```

The missile locations B7, E4 and F9 would all be misses. The missile locations C3, G7 and F10 would all be hits. You will be given a grid with the ship locations and a sequence of missile locations. You must determine the number of hits, number of misses and determine whether or not all ships have been sunk.
**Input**

The first line contains a positive integer \( n \), where \( 3 \leq n \leq 26 \), representing the size of the \( n \times n \) grid. The second line contains an integer \( m \), \( 1 \leq m \leq n^2 \), representing the number of missile locations. The next \( n \) lines represent the grid containing the ship locations. Each line will contain a sequence of \( n \) characters, where 0 (the upper-case letter “oh”) represents a blank and * represents one of the locations of a ship. (Note that the number and sizes of the ships may vary from that of the actual board game.) The next \( m \) lines represent missile locations. Each of the lines will contain a code consisting of an upper-case letter (A, B, C, ...), followed immediately by an integer 1, 2, ..., \( n \). You may assume that the code represents a valid location on the grid and that all missile locations are unique.

**Output**

The first line of output must contain the number of hits, followed by a single space and the word “hits” (or the word “hit” in the case of only one hit). The second line of output must contain the number of misses, followed by a single space and the word “misses” (or the word “miss” in the case of only one miss). The last line of output must contain one of two phrases “all ships sunk” or “not all ships sunk” according to whether or not all ships on the grid have been sunk.

**Sample Input**

5
9
000**
0*000
0*000
00000
A3
E1
E2
B2
B3
C2
D2
C4
A4

**Sample Output**

4 hits
5 misses
not all ships sunk
Problem 2: Power-prime

A power-prime is a number with the following structure: \( pxq \), where \( p \) and \( q \) are prime numbers, and \( x = p^q \). For example, the following are power-prime numbers:

\[
\begin{align*}
283 \\
193612
\end{align*}
\]

The first is a power-prime because \( 2^3 \) is 8. The second is a power-prime because \( 19^2 = 361 \).

The Problem

Given a minimum and maximum digits, output the number of power-primes with no fewer and no more digits than the given minimum and maximum values. For example, here are the power-primes between 3 and 5 digits long: 242, 283, 392, 2325, 3273, 5252, 7492, 21287, 32435, 51253, and 73433. So given 3 and 5 as input, your program would report 11, the number of power-primes whose length is between 3 and 5.

Input

Your program will be given multiple instances of the problem above. Each line of input contains 2 integers between 3 and 15 (or both 0), such that minimum <= maximum. The first is the minimum number of digits, the second the maximum number of digits. The last line contains two zeros, signaling the end of input.

Output

For each instance, one per line, your program will output the number of power-primes with a length in digits between the given minimum and maximum.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 5</td>
<td>11</td>
</tr>
<tr>
<td>4 4</td>
<td>4</td>
</tr>
<tr>
<td>15 15</td>
<td>2346</td>
</tr>
<tr>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>
Problem 3: Top Secret

A technique for encoding and decoding secret messages is called a cipher. The original message is called the plaintext, and the encoded message is called the ciphertext.

For this problem, the plaintext message itself contains the information for its encoding. You first go through the plaintext and calculate the frequency count $n_i$ for each character $C_i$. The ciphertext is generated by replacing each character $C_i$ that has a positive frequency count with the letter in the alphabet that is $n_i$ characters before $C_i$, wrapping around at the beginning of the alphabet, if necessary. Character encodings are done in alphabetical order for the letters in the plaintext. If the encoding of a character conflicts with the mapping that has already been determined, it would be converted to the available character immediately preceding that has not yet been mapped, wrapping around at the beginning of the alphabet, if necessary. As an example, suppose there are 4 occurrences of the letter "z", 4 occurrences of the letter "y", and 1 occurrence of the letter "w". We will first map "w", then "y" and finally "z". Therefore in the plaintext, the "w" will be replaced by "v" which is the 1st character before "w". Each "y" would be replaced by "u", the 4th letter before "y" as shown below:

$$a, b, c, \ldots r, s, t, u, v, w, x, y, z$$

When we map "z", we try to replace it by "v", the 4th letter before "z", but since "v" is already used, we try "u" next. The letter "u" is also used, so "z" will be encoded to "t".

The first data line contains a positive integer indicating the number of data sets to be processed. Each data set will have two messages, one message per line. The first message is a plaintext. Your program should use it to generate the code mapping scheme, and output its corresponding ciphertext. The second message is a ciphertext that your program should convert back to the plaintext using the same mapping scheme. Note that this ciphertext will contain only characters that the plaintext has mapped into. Therefore, in the example above, the ciphertext message to be decoded will contain "t", "u" and "v" only. Only lowercase letters and blank space will be used. Each message can be at most 80 characters long. Note that extra blank spaces between words in the message should be replaced by a single blank space in your output.
**Sample Input**

2
yz yywz zyz
tt uu vv
ababab b
w x

**Sample Output**

ut uuvt tut
zz yy ww
xwxwxw w
b a
Problem 4: The Straw that Broke the Network's Back

Consider a network of damaged and undamaged computers with reflexive connections (i.e., if a is connected to b, then b is connected to a). A computer can communicate with another computer if there is a path between the two computers that does not contain a damaged computer. A network is considered damaged if there exists a pair of undamaged computers that cannot communicate with each other.

Given a description of a network of computers, and a sequence of computers in the order that they are damaged, report the first damaged computer that causes the network to be damaged and "Survived" otherwise.

Input

Your program will be given multiple instances of this problem. The first line of input of each instance contains an integer N between 0 and 100. This is the number of lines in the rest of that instance's input. If N is 0, there are no more instances. Each subsequent line of an instance's input contains an integer M (between 1 and 100) followed by a space-delimited list of computer names. M is the number of computer names in the list. Computer names are case sensitive string of alphabetic characters. The length of each name is between 1 and 15 characters.

The first list of computer names is a list of all the computers in the network. Each computer has a unique name. The next N-2 lists of computers define the connections between the computers. In each of these lines, the first computer in the list is connected to the other computers in the list. The last list of computers is a list of the damaged computers, in the order that they become damaged.

Output

For each instance, print the first damaged computer that causes the network to be damaged. If all of the computers are damaged without damaging the network, report, "Survived".
<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
</table>
| 5
4 a b c d
3 a b c
2 b d
2 c d
3 b c d
5
4 a b c d
3 a b c
2 b d
2 c d
4 b d c a
5
4 a b c d
3 a b c
2 b d
2 c d
4 a b d c
0 | c
Survived
Survived |
Problem 5: Line Clipping

An important technique in computer graphics is called line clipping. A screen contains one or more viewing windows. A line defined in coordinates for the screen is “clipped” to the portion that lies within a viewing window on the screen.

A viewing window is defined by four non-negative integer values called clipping planes: left, right, bottom, and top. All points (x, y) within the viewing window satisfy

\[
0 \leq \text{left} \leq x \leq \text{right},
\]

and

\[
0 \leq \text{bottom} \leq y \leq \text{top}.
\]

Given two integer-valued coordinate points that define a line, and four clipping planes (left, right, bottom, top) that define a viewing window, a line is clipped by redefining its endpoints so that it lies within the window (or the line is eliminated if no part of it lies within the window).

NOTE: All points are plotted in integer coordinates on the screen. Thus when redefining the endpoints of a line, you must round each value to the nearest integer.

The first line of input contains four non-negative integer values, separated by single spaces, representing the left, right, bottom and top clipping planes of a window. The next line of input contains a positive integer n, representing the number of lines to be clipped. Each of the following n input lines contain four non-negative integer values x1 y1 x2 y2, separated by single spaces, representing the two endpoints (x1, y1) and (x2, y2) of a line to be clipped.

For each line processed, output the integer-coordinate endpoints of the corresponding clipped line in the same format and order as the input: x1 y1 x2 y2. If a clipped line contains only a single point at the boundary of the window, output only that point. If a line lies entirely outside of the window, output “no line”.

=>

after clipping
Sample Input
20 400 80 350
5
7 100 200 300
50 380 300 450
440 300 360 390
380 0 410 120
250 303 250 80

Sample Output
20 113 200 300
no line
400 345 396 350
400 80
250 303 250 80
**Problem 6: Scrambling**

In the popular Zynga app “Scramble With Friends”, you are given a jumbled grid of letters and have to find as many words as possible in a given time limit. Words are found by following a path through the grid in any direction: horizontally, vertically or diagonally (with no square in the grid repeated within a word). For example, consider the grid of letters shown below. The two grids shown to the right demonstrate finding the words TRILLED and DATES. Notice that a path can cross over itself, but it cannot repeat a used letter.

```
T S L E
B L I R
D E T M
S A U C
```

```
T S L E
B L I R
D E T M
S A U C
```

```
T S L E
B L I R
D E T M
```

```
T S L E
B L I R
D E T M
S A U C
```

Each word that is found can be described as a starting location (row number, column number), followed by a successive sequence of directional moves (N, NE, E, SE, S, SW, W, NW). In the above examples, the paths can be described as follows:

**TRILLED**

3,3
NE W N SW S W

**DATES**

3,1
SE NE W SW

**Input**

The first line contains a positive integer \( n \), where \( 3 \leq n \leq 10 \), representing the size of the \( n \times n \) grid. The following \( n \) lines of input contain \( n \) letters each, with no spaces, representing the letters in the grid. The next line of input represents an integer \( m \), \( 1 \leq m \leq 10 \), representing the number of words to be found in the grid. The next \( m \) lines of input contain the words to be found, one word per line.
Output
For each word, output the word to be found. On the next line, output “not found” if the word was not found in the grid. Otherwise, output the starting location: row number, followed by a comma, immediately followed by the column number (no space). On the next line, output the sequence of directional moves describing the path, separated by single spaces. If there is more than one occurrence of the same word, output all paths in the format described above, ordering the paths by the starting location and using row major order for all square locations in the path (sorted first by row number, then by column number). Put a blank line between the lines of output for each word.

Sample Input

5
SRENG
OLIOA
VISKE
THAOR
PDTAL
4
LORE
NOSE
ROILS
SAILORS

Sample Output

LORE
2,2
W NE E
5,5
NW E N

NOSE
not found

ROILS
1,2
SW SE N NW
1,2
SW SE N SE

SAILORS
3,3
S NW N W NE W
Problem 7: Truth Is Beautiful

Propositional Calculus is a knowledge representation language in Artificial Intelligence, where each proposition represents a piece of knowledge which can either be T (true) or F (false). In this problem we will use uppercase letters to represent propositional symbols, and connectives are represented as follows:

Negation: - (1)
Conjunction (and): & (2)
Disjunction (or): + (3)
Implication: > (4)
Equivalence: = (5)

where the integer parentheses indicates the precedence order of operation (1 to be the highest precedence, and 5 to be the lowest precedence,) and () may be used to group expressions to change the order of evaluation. For example, P&Q>Q is the same as (P&Q)>Q since & has higher order of precedence than >.

Recall that:
- P is F if P is T, and T if P is F
- P&Q is T only if both P and Q are T, otherwise it is F
- P+Q is F if both P and Q are F, otherwise it is T
- P>Q is F only if P is T and Q is F, otherwise it is T
- P=Q is T if and only if P and Q have the same truth value

Expressions formed by propositional symbols and these connectives are called sentences. Here are two sentence examples

Q & P
Q & (Q > P) > P

Note that any number of blank spaces may be used in a sentence.

Two sentences P and Q are logically equivalent if and only if the equivalence P = Q is true for all truth values. For example, DeMorgan’s Laws tells us that -(P&Q) is logically equivalent to -P+-Q. Equivalent sentences can be used to change a propositional calculus expression into a syntactically different but logically equivalent form.

An implication P>Q is sound if whenever the premise P is true, the conclusion Q is also true. For example, the modus ponens

Q & (Q > P) > P

is sound. Sound implications are used to infer new knowledge.
You are to write a program that will generate the truth table of a sentence. The first data line in the input contains an integer, N, indicating the number of sentences your program should process. Each of the following N lines contains a legal sentence. Each sentence can be at most 80 characters long. Your program should generate the truth table for each sentence. If the sentence is an implication, your program should also determine whether the implication is sound and output “sound” or “not sound”. If the sentence is an equivalence, your program should also determine whether they are indeed logically equivalent, and output “equivalent” or “not equivalent”. If the sentence is neither an implication nor an equivalence, your program should also output “neither”. Note that in the truth table you must list propositional symbols in alphabetical order, and eliminate all blank space in the sentence. Also, use two blank spaces between each item, and use two blank lines to separate two sets of output.

**Sample Input**

```
3
Q & P
Q & (Q > P) > P
(Y & X) = (X + Y)
```

**Sample Output**

```
P Q Q&P
T T T
T F F
F T F
F F F
neither

P Q Q&(Q>P)>P
T T T
T F T
F T T
F F T
sound

X Y (Y&X)=(X+Y)
T T T
T F F
F T F
F F T
not equivalent
```