Problem 1

Who's Got Game?

Massively multiplayer online role-playing games (MMORPGs) have their roots in MUDs (multi-user dungeons) which gained initial popularity in the 1970's. Here's an early excerpt:

> You are standing on the edge of a cliff surrounded by forest to the north and a river to the south. A chill wind blows up the unclimbable and unscaled heights. At the base of the cliff you can just make out the shapes of jagged rocks.
>
> `> go west`
>
> As you approach the edge of the cliff the rock starts to crumble. Hurriedly, you retreat as you feel the ground begin to give way under your feet!
>
> `> leap`
>
> You are splattered over a very large area, or at least most of you is. The rest of your remains are, even now, being eaten by the seagulls (especially your eyes). If you had looked before you leaped you might have decided not to jump!
>
> Would you like to play again?

In order to make things interesting for the player, the best RPGs try to allow for non-linear game-play, so a user's decisions during the game allow for a different experience than others who play the game. However, certain elements of the game must be performed in sequence. For instance, a key must first be discovered before a hidden chest can be opened and a new (more powerful) item collected. The key doesn't appear until you talk to the mage in the water town. After all, you wouldn't want the most powerful weapons in the game to be available right from the start! (Where would the challenge be?) Keeping up with these sequences is no easy task, especially when teams of designers are dreaming up new scenes and items all the time... who can keep up?

In order to help out aspiring dungeon masters, you want to determine if a given set of items or tasks will allow for strictly linear game play, many possible game play sequences, or if a particular set of relationships will result in an unfeasible game.

Input

There will be multiple test cases in the input.

Each test case will begin with a line with two integers n ($1 \leq n \leq 2,000$) and m ($1 \leq m \leq 50,000$), where n is the number of items or tasks, and m is the number of relationships between items.

On each of the next m lines will be two integers, d and u ($1 \leq u,\ d \leq n,\ d \neq u$) which indicate that collecting item or performing action d allows access to item or action u.

The input will end with two 0's on their own line.

Output

For each test case, print a single line of output containing 'Infeasible game' if the proposed gameplay sequences are impossible, 'Linear gameplay' if exactly one sequence is possible, or 'Nonlinear gameplay possible' if multiple arrangements are possible (no matter how many arrangements there are).

Sample Input

```
5 4
1 5
5 2
3 2
4 3
5 4
3 1
4 2
1 5
5 4
2 2
1 2
2 1
0 0
```

Sample Output

```
Nonlinear gameplay possible
Linear gameplay
Infeasible game
```

Problem 2

Sort of Garbage

Here's a message from a physicist friend of mine:

> How's things? Good. I have attached a data file which is plain ascii text to this email. You see, I have a bunch of data that was generated in our lab across the way. Millions of dollars worth of machinery, but the data files are full of garbage! You would think they were scanned from an old dot matrix printout or something. Anyway, the data files seem to change every time I get a new one, can you help me write something to sort it?
>
> Here's what I have observed. The data files are never more than 10,000 lines of data. There is a preamble of text to each data file, but I never know how many lines it is. After the preamble, there are column headings, followed by a bunch of floating point measurements corresponding to the column headings. There are often extra columns, which I don't need in the output.
>
> The output needs to be sorted by the column Q2, then by F2. I need columns Q2, F2, stat, and sys in the output, while all other columns should be ignored.
>
> Also, I never know in what order the columns will be generated. The columns we do keep should appear in the output in the same left-to-right order as they appear in the input. The output file should contain the column labels as well.

Write a program to sort the physicist's data. The program should read from the standard input, discard the preamble (of unknown length), discard unnecessary columns (of unknown count), and write to the standard output. The output should be tab delimited (left aligned), should include the column headers, and be sorted first by column Q2, and in the case of a tie, by column F2.

Input

```
SLAC F2 proton (reanalyzed)
L.W. Whitlow et al., PL B282, 475 (1992)
*
.021  0  :  normalization error, #_of_correlated_sys.err
      x         Q2        F2        i        stat      sys      dummy     dummy
   0.08100   1.02000   0.32385  16.00000   0.00939   0.00453   1.00000   1.00000
   0.08200   1.02000   0.33099  17.00000   0.00530   0.00496   1.00000   1.00000
   0.09700   0.87000   0.31517  31.00000   0.00630   0.00410   1.00000   1.00000
   0.09800   1.44000   0.34013  32.00000   0.00986   0.00340   1.00000   1.00000
```

Output

```
Q2 F2 stat sys
0.87000 0.31517 0.00630 0.00410
1.02000 0.32385 0.00939 0.00453
1.02000 0.33099 0.00530 0.00496
1.44000 0.34013 0.00986 0.00340
```

Problem 3

Suuudoooohhhkuuuuu

Sudoku is the number-placing game that is taking the world by storm! Have you played it yet? Each Sudoku game consists of a 9 x 9 matrix (made of nine 3 x 3 sub-matrices) such as the one below:

•	7	•	5	2	•	8	•	•
•	8	•	•	1	6	•	4	•
2	•	1	•	•	•	•	•	7
6	•	•	8	•	•	7	9	•
•	4	9	•	6	•	2	3	•
•	1	8	•	•	9	•	•	5
1	•	•	•	•	•	6	•	9
•	9	•	6	8	•	•	7	•
•	•	6	•	5	7	•	2	•

Given an initial set of entries, the player enters digits from 1 to 9 in the blank spaces, for as long as:

1. Every row has one of each digit
2. Every column has one of each digit
3. Every 3 x 3 sub-matrix has one of each digit.

For example, in the above board, the 8 in column one can only be placed in the bottom row (row 9). It can't be placed rows 1-6 of column one, because each of these is contained in a submatrix that already has an 8. Row 7 of column one is occupied. It can't be placed in row 8, because that row already has an 8 in column 5. Placing the 8 in any row in column one other than the last would be a wrong move. Each Sudoku game has a unique solution that can be reached logically without guessing.

Your job is to write a program that reads Sudoku boards and determines whether or not any wrong moves have been made.

Input

Input consists of a nonnegative 32-bit integer n on a line by itself, denoting the number of problem instances, followed by n Sudoku boards. Each board consists of a blank line, followed by 9 lines, with each line containing 9 symbols. Each symbol is either a digit from 1 to 9, or a period.

Output

For each problem instance, your program prints one of:

- "ok so far" : if the board has no mistakes, but is not complete.
- "there is an error somewhere" : if the board has at least one mistake.
- "you got it!" : if the board has no mistakes, and is complete.

Sample Input

```
4

.7.52.8..
.8..16.4.
2.1.....7
6..8..79.
.49.6.23.
.18..9..5
1.....6.9
.9.68..7.
..6.57.2.

.7.52.8..
.8..16.4.
2.1.....7
6..8..79.
.49.6.23.
.18..9..5
1.....6.9
.9.68..7.
8.6.57.2.

.7.52.8..
.8..16.4.
2.1.....7
6..8..79.
.49.6.23.
.18..9..5
1.....6.9
89.68..7.
..6.57.2.

974523816
583716942
261498357
625834791
749165238
318279465
157342689
492681573
836957124
```

Sample Output

```
ok so far
ok so far
there is an error somewhere
you got it!
```

Problem 4

Tinfoil Hat

A monoalphabetic substitution cipher is an encryption scheme where plaintext letters are replaced with ciphertext letters according to a fixed substitution table.

For example, consider following substitution table:
$a \rightarrow E$
$b \rightarrow F$
$c \rightarrow G$
$d \rightarrow H$

In this case, plaintext letter "a" is replaced with ciphertext letter "E", "b" with "F", "c" with "G" and "d" with "H". As a result, the plaintext phrase "abcd" would be encrypted as "EFGH" and "abba" with "EFFE". In order to decrypt a given ciphertext phrase (i.e., to find its corresponding plaintext phrase), the process has to be reversed: ciphertext letter "E" is replaced with "a", "F" with "b", "G" with "c" and "H" with "d". Write a program that repeatedly reads in a substitution table and then encrypts or decrypts several plaintext or ciphertext phrases accordingly.

Input

Input comes in the form of possibly several blocks of encryption/decryption requests. Each block starts with an integer, specifying the number of characters in the substitution table. An integer of 0 indicates the end of the overall process. The line right after the integer (in case it was not 0) contains the individual plaintext characters (with whitespace between each character). This line is followed by a new line listing the corresponding ciphertext characters, where each ciphertext character is written directly below its plaintext character. The line after the specification of the substitution table contains an integer which is greater or equal to 0. This integer indicates the number of encryption/decryption requests that follow. Each encryption/decryption request is a new line starting with either "PT:" or "CT:" followed by a whitespace and a phrase. "PT:" indicates that the phrase is plaintext and therefore needs to be encrypted, while "CT:" marks the phrase as ciphertext, which needs to be decrypted.

Output

For each plaintext/ciphertext phrase the corresponding ciphertext/plaintext phrase should be printed. If the original phrase was a plaintext, the result should be the ciphertext, preceded by "CT:" and whitespace. For a given ciphertext, the plaintext should be printed, this time

preceded by "PT:" and whitespace. Each phrase is printed on its own line, with no space between the lines.

Sample Input

```
3
a b c
B C D
2
PT: abc
CT: BCDDCB
26
a b c d e f g h i j k l m n o p q r s t u v w x y z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
5
PT: this
PT: is
PT: a
PT: caesar
PT: cipher
5
d e h l o r w
H J L O Q T V
4
PT: hello
PT: world
CT: LJOOQ
CT: VQTOH
0
```

Sample Output

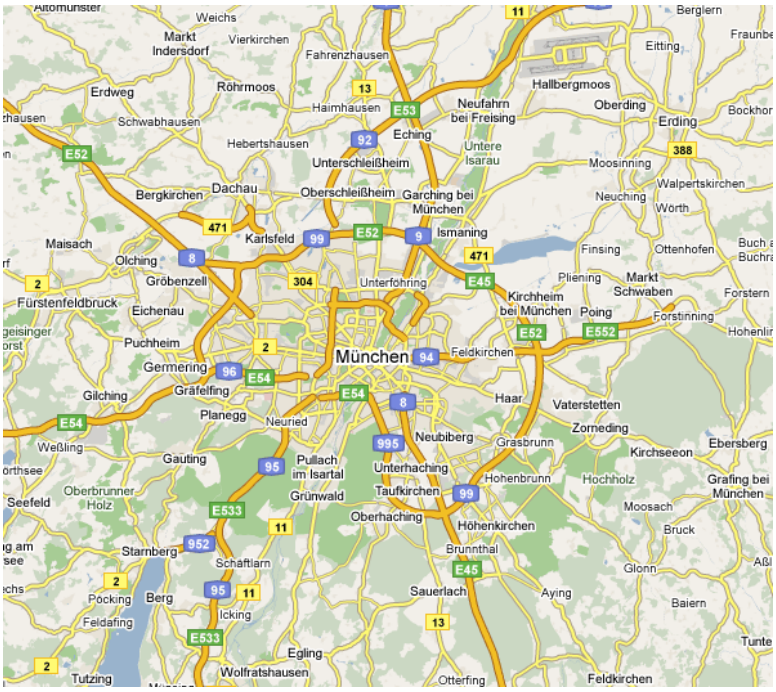
```
CT: BCD
PT: abccba
CT: WKLV
CT: LV
CT: D
CT: FDHVDU
CT: FLSKHU
CT: LJOOQ
CT: VQTOH
PT: hello
PT: world
```

Problem 5

Fun, Fun, Fun "Auf der Autobahn"

Imagine spending a semester at the Technical University Munich, located (of course) in Munich, Germany. In order to get around this city, you could rely on the metro system, which provides great connections within the city and also far into the surrounding countryside. But then, you didn't come to Germany to only study Ohm and Siemens. No! Germany is also the country of Benz and Porsche. Furthermore, there is Germany's largest public entertainment park—the Autobahn! Combine the one with the other, and what do you get??? Fun, fun, fun, auf der Autobahn! Yes, can you imagine?!

And this is where this problem begins. It's about route finding while *increasing the fun factor*. Look at the map of Munich below, specifically at the street system. The thick lines are the Autobahn, the thinner streets are back roads. In order to get from any point A to any other point B, you can typically take many different routes, some consisting of only back roads, some including Autobahn segments. Of course, for fun purposes, you would always try to choose a route that would put you on the Autobahn, at least partially. On the other hand, you're a rational person, and you hate wasting time. That's why in the end you will choose the fastest route after all, and if it means that it is a back-roads-only route, well, there goes the fun... The question is now: Given any two points A and B, will you get to have fun when driving from A to B?



To solve this problem, you are given a map (in form of a graph), which contains all possible locations and intersections (these are the nodes of the graph), and the street segments (these are the edges). Each edge has a distance value associated with it, which specifies the length of this street segment in kilometers. Your average speed that you can drive on a street segment depends on whether it is a back road or the Autobahn. For this problem, assume that you can drive 80 km/h (about 50 mph) on a back road, and 160 km/h (100 mph) on the Autobahn. (Yes, this seems rather slow, as the real fun begins somewhere beyond 200 km/h. But then, it's just an average. . .)

Your task is to write a program that reads in the graph information and determines for several node pairs the total distance (in km) of the route, which takes the shortest time, as well as the number of Autobahn kilometers this route includes. In cases where there are several equally fast routes, your program should select the one(s) with the most Autobahn kilometers.

Input

The input starts with an integer specifying the number of nodes in the graph. In the next line, the node "names" (simple letter combinations) are listed. The next line contains the number of street segments that will be specified in the following lines. Each street segment consists of two nodes that this street connects, the length in kilometers of this segment (given as an integer), and whether it is a back road ("b") or Autobahn ("a"). Assume that all streets are two-way (i.e., bidirectional). Furthermore, for simplicity assume that only one edge can exist between any two nodes (i.e., there are no back roads running parallel to an Autobahn).

After the graph description come the requests. At first, an integer specifies the number of route calculation requests. Each request then consists of two nodes.

Output

For each route request, your program should output the starting point and the destination point as well as the total travel distance in km and the total distance traveled on the Autobahn (also in km).

Sample Input

```
6
A B C D E F
7
A B 10 a
B C 10 a
D A 1 b
E B 5 b
F C 1 b
D E 10 b
E F 10 b
3
A B
D E
F D
```

Sample Output

```
A B 10 10
D E 10 0
F D 22 20
```