Tenth Quinnipiac University Programming Competition for High School Students

Quinnipiac University

April 25, 2015

Instructions

1. Program submissions are done using the Programming Contest Control System (PC^2). See the separate sheet for instructions on submitting your programs.
2. You may assume the input is formatted as shown in the problem, and your program must work on such input. You may assume there are no tabs in the input or trailing spaces.
3. Your output must match the specified output *exactly*. This includes not having any trailing spaces.
4. If your program crashes, produces incorrect output, or times out, no information will be given as to the input that caused the problem.

1 Prime Factorization

Every number can be broken down into the product of its prime factors. This process is known as prime factorization. For example, $12 = 2 \times 2 \times 3$.
Write a program that takes a list of numbers and prints out each number followed by its prime factorization (written as a product). The prime factors should be listed in increasing order. If the number is a prime or is 1 (unique) then only that number would be listed.

Input

The first line contains the number N of cases with $0 < N \leq 1000$. Each of the remaining N lines contains a whole number X with $0 < X < 10000$.

Output

The output consists of one line per number X starting with `X=` and followed by the prime factors separated by x. If the number is already prime or is unique it should just output `X=X`.

Input

```
4
12
13
1
252
```

Output

```

12=2x2x3
13=13
1=1
252=2x2x3x3x7
```

2 ROT-13

A simple form of encryption is to take every letter and substitute it with the letter 13 letters after it in the alphabet (wrapping around). That is, $A \rightarrow N, B \rightarrow O, C \rightarrow P, \ldots, M \rightarrow Z, N \rightarrow A, O \rightarrow B, \ldots, Z \rightarrow M.$ It is commonly used in online forums as the equivalent to writing an answer upside down since it is easy to decrypt in the Latin alphabet, just apply another ROT-13.
For example, `MESSAGE` would get encrypted to `ZRFFNTR`, and `ZRFFNTR` becomes `MESSAGE`.

Input

The first line of input consists of a single value N, with $0 < N \leq 1000$, describing the number of lines to encrypt. Each line is a sequence of at most 100 characters.

Output

The output consists of one line for each input line with every alphabetic character replaced with its ROT-13 equivalent. The alphabetic characters will only be upper case letters (A-Z).

Input

```
5
ONE MESSAGE
SECRET PASSWORD
"GO BOBCATS," THEY CHEERED.
IF I HAVE IT, I DON’T SHARE IT. IF I SHARE IT, I DON’T HAVE IT. WHAT IS IT?
N FRPERG.
```

Output

```
BAR ZRFFNTR
FRPERG CNFFJBEQ
"TB OBOPNGF," GURL PURRERQ.
VS V UNIR VG, V QBA’G FUNER VG. VS V FUNER VG, V QBA’G UNIR VG. JUNG VF VG?
A SECRET.
```

3 Printing Big Numbers

Suppose you are given a number like 123 and want to print it in text but in large characters. One way is to convert each digit to a "graphical" representation. So, 123 becomes

```
  1    22   33333
  1   2  2      3
  1     2     33
  1    2        3
  1   22222 33333
```

Each digit has its own representation as illustrated below with the spaces for the representation converted to periods '.' so that they are easier to see. When printing out a number, each "digit" will be separated by a single space.

```
..1..  .22..  33333  4..4.  55555  6....  77777  88888  .9999  00000
..1..  2..2.  ....3  4..4.  5....  6....  ...7.  8...8  9...9  0...0
..1..  ..2..  ..33.  44444  5555.  66666  ..7..  .888.  .9999  0...0
..1..  .2...  ....3  ...4.  ....5  6...6  ..7..  8...8  ....9  0...0
..1..  22222  33333  ...4.  5555.  66666  ..7..  88888  ....9  00000
```

Input

The first line consists of the amount N of numbers to process, $0 < N \leq 100$. This is then followed by N lines, each line consisting of a number containing up to 20 digits. The number will only contain digits in the range 0-9 and will not have any leading 0s (except for the number 0). There will also be no other symbols such as a $-$ sign.

Output

For each number, the graphical representation of that number should be printed with each digit separated by a single space and each number separated by dashes (-) that are the same length as the representation of the number above it. Note the last digit of each number should not include a space after it but all the spaces for that digit should be printed (that is, each digit should be a 5×5 grid).

Input

```
3
3748596
3
102
```

Output

```
33333 77777 4  4  88888 55555  9999 6
    3    7  4  4  8   8 5      9   9 6
  33    7   44444  888  5555   9999 66666
    3  7      4  8   8     5      9 6   6
33333  7      4  88888 5555       9 66666
-----------------------------------------
33333
    3
  33
    3
33333
-----
  1   00000  22
  1   0   0 2  2
  1   0   0   2
  1   0   0  2
  1   00000 22222
```

4 Fraction Calculator

Fractions appear everywhere. In programming, one can convert a fraction like $\frac{1}{2}$ into a floating point number like 0.5 to do calculations more easily. However, floating point values are susceptible to precision errors so it is often better to work directly with fractions. For this problem, you are tasked with doing some basic operations on a pair of fractions. For example, $\frac{1}{2} + \frac{1}{3} = \frac{1}{6}$.

Input

The first line of the input is the number of test cases, N, with $0 < N \leq 100$. Each of the remaining N lines contains 5 values of the form $n_1\ d_1\ op\ n_2\ d_2$ with $-100 \leq n_1, n_2, d_1, d_2 \leq 100$ and with op either `+`, `-`, `*`, `/`.

Output

For each test case, you should output the resulting fraction *as simplified as possible* in the form:

```
numerator denominator
```

If the resulting number is a whole number, it should just be the whole number. You do not need to output mixed fractions, e.g. $\frac{11}{6}$ should be `11 6`. If the number is negative, the negative sign should appear in the numerator, e.g. $-\frac{3}{4}$ should be `-3 4`. If the denominator is ever zero, output `ERROR`.

Input

```
14
1 3 + 1 2
1 2 - 1 3
1 3 + 1 6
3 8 * 10 9
3 6 / 15 9
6 7 * 14 3
3 20 - 7 10
5 1 * 0 5
6 10 / 0 3
5 3 + 1 6
5 0 - 5 0
5 0 * 0 5
-1 2 + 1 -3
-2 3 - 1 -6
```

Output

```

5 6
1 6
1 2
5 12
3 10
4
-11 20
0
ERROR
11 6
ERROR
ERROR
-5 6
-1 2
```

5 Intersecting Lines

Determining if two line segments intersect is a fundamental process in computer graphics, useful for clipping, ray tracing, shadow effects, reflections, and collision detection. For this problem, you will write a program to determine if pairs of line segments intersect or do not. Every line will be represented by integer coordinates.

Input

The first line of the input is the number of pairs of line segments, N, with $0 < N \leq 100$. Each of the remaining N lines represents two line segments of the form $A_x\ A_y\ B_x\ B_y\ C_x\ C_y\ D_x\ D_y$ with all values in the range $(-1000, 1000)$. The first line segment is denoted by its endpoints (A_x, A_y) and (B_x, B_y) and the second line segment by its endpoints (C_x, C_y) and (D_x, D_y).

Output

For each pair of line segments, the output is a single line containing either `INTERSECT`, `PARALLEL`, or `DO NOT INTERSECT` depending on whether the two line segments intersect, are parallel, or do not intersect. The endpoints are considered part of the line segment. So, if an endpoint of one line segment is on the other line segment, we consider that an intersection. If the two line segments are parallel even if they overlap or intersect, we consider this to be a parallel case.

Input

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

Output

```
DO NOT INTERSECT
PARALLEL
INTERSECT
PARALLEL
PARALLEL
INTERSECT
```

6 Word Search

For this problem, you will implement a simple Word Search where the input consists of a grid of letters and then a list of words. The grid should be searched for each word to see if the word is present in the grid. Matches should be found by searching in the grid across a row (either left to right or right to left), up or down a column, or diagonally (in any of the four directions).

Input

The first line of the input indicates the number of rows in the grid, $0 < R \leq 100$. The second line of the input indicates the number of columns in the grid, $0 < C \leq 100$. This is then followed by R lines of C (upper-case) letters each. This is followed by a single line containing the number of words to search, $0 < N \leq 100$. This is followed by N lines, each containing a single (upper-case) word.

Output

For each of the N words, if the word can be found in the grid either horizontally, vertically, or diagonally in any direction, then the location of the first letter in the word on the grid is output as `R C`. If no match is found, `NOT FOUND` is to be output. In the event that a word can be matched in multiple locations the output should be the one with the starting location in the smallest row (and then smallest column if the rows are tied).

Input

```
6
5
WATER
APEOS
EPATC
PLQRA
LESPY
ALGNZ
7
WATER
APPLE
TAPE
COT
TEA
ALL
EAT
```

Output

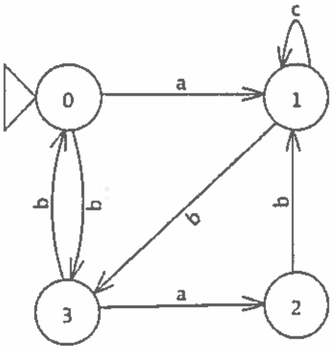
```
1 1
1 2
3 4
3 5
1 3
3 3
NOT FOUND
```

7 State Machines

A state machine is a mathematical model of computation used to represent simple computer programs. A state machine consists of a set of states, a set of transitions between states which are triggered by specific events (symbols), and a start state.[1] The machine is run on a given input consisting of a sequence of symbols (characters) by starting initially at the start state. Each symbol in the input is processed one at a time. If there exists a transition from the current state s to another state t using the current symbol c, then t becomes the new state. Note that the new state might still be the same state (states can refer back to themselves). If there is no transition from s with that symbol, the program is said to REJECT the given input. The final state of the input is the last state that the machine is in once the last symbol has been processed.

In our case, we shall label all states in numerical order $0, 1, 2, \ldots$, shall let the start state be state 0, and let the set of symbols be the lower-case letters $a, b, c, \ldots, z$. The following figure illustrates graphically one state machine consisting of 4 states. For instance, when in state 1 on a `b` the machine would advance to state 3, on a `c` it would remain in state 1, and on any other symbol it would reject the input. Running the program on the input `acba` would progress as follows. The machine would start in state 0. It would then process the first character in the input, `a`, and transition to state 1. It would then process the second character, `c`, and remain in state 1. On the next character, `b`, the machine would transition to state 3. And finally, on the final character `a`, the machine would end up in the final state of 2.

Similarly, running the program on the input `acbca`, would go from state 0 to state 1, to state 1 again, to state 3, and then reject because there is no transition from state 3 for the symbol `c`.



Your task is to write a program to simulate a state machine on a given input and identify the final state or report `REJECT` if no such transition is possible. The machines provided will all be deterministic, from any state s on any given symbol there will be at most *one* transition given. For example, in the above state machine, we could not add a transition from state 1 to state 2 on a `c` because there is already a transition from state 1 to state 1 on a `c`. But, we could add a transition from state 1 to state 3 on an `a`.

Input

The first line of the input will be a number N with $0 < N \leq 100$ indicating the number of test cases. Each test case consists of:

1. A number K (on its own line) with $0 < K \leq 10$ indicating the number of states in the current machine,

[1] A final state is often required as well but we shall skip that for this problem.

2. a number T (on its own line) with $0 < T \leq 100$ indicating the number of transitions,
3. T lines each containing two numbers S_1 and S_2 with $0 \leq S_1, S_2 < K$ and a character C with C in the range a to z. This indicates that there is a transition from state S_1 to state S_2 on symbol C,
4. a number I (on its own line) with $0 < I \leq 100$ indicating the number of inputs to try,
5. and I lines each containing an input string of symbols (in range a to z) of length at most 100.

Output

The output consists of one line for each input string indicating the final state of running the input on the given machine and `REJECT` otherwise.

Input

```
2
4
7
0 1 a
1 1 c
1 3 b
0 3 b
3 0 b
3 2 a
2 1 b
4
acba
acbca
accccc
acccbbbabcba
3
9
0 1 a
1 0 b
1 2 a
2 1 b
2 0 a
0 2 b
0 0 c
1 1 d
2 2 e
8
aab
abc
ababc
abaabac
aae
adaeac
bebdbc
aaabbbaa
```

Output

```
2
REJECT
1
2
1
0
0
REJECT
2
0
0
2
```

8 Instant-Runoff Voting System

Many countries with more than two political parties have adopted a process for electing candidates based on ranking them to create an instant-runoff vote. Every voter ranks the candidates in order, but they are not necessarily required to rank every candidate. The election process works as follows.

- The first place votes are counted for each candidate.
- If there is a single candidate that wins more than half of the votes, that candidate is declared the winner.
- If not, the candidate with the fewest number of votes is eliminated and the ballots are adjusted to remove that candidate from the ballots.
- Any empty ballots (no more remaining candidates ranked) are discarded and the process repeats.

If during the election process, two candidates are left both with exactly half of the votes, the result is declared a "TIE." If during the election process, two (or more) candidates have exactly the same number of fewest votes, then an "ERROR" is reported unless that number is 0 in which case the candidates can be removed.

Input

The first line of the input contains the number N of cases with $0 < N \leq 1000$. Each of the test cases consists of the following information:

- A line containing the number K of candidates with $0 < K \leq 10$.
- K lines, each line containing the name of a candidate using at most 20 (upper-case) characters. (Each candidate will have a unique name).
- A line containing the number V of votes with $0 < V \leq 1000$.
- V lines, each line containing an ordering of candidates with each candidate separated by a single space. The ordering will be valid so only valid candidate names will be used and no candidate name will be repeated. Not all candidates might be listed but there will always be at least one candidate listed.

Output

The output consists of N lines, one per case. Each line should be the result of that particular election: either the winning candidate, a "TIE" if two candidates both receive exactly half the votes, an "ERROR" if during the election it was not possible to identify a candidate to discard (two or more candidates with the fewest (non-zero) number of votes).

For example, if one election had four candidates: WASHINGTON, ADAMS, JEFFERSON, and FRANKLIN. And there were 6 ballots cast:

```
WASHINGTON JEFFERSON ADAMS, WASHINGTON ADAMS, WASHINGTON JEFFERSON, ADAMS, ADAMS JEFFERSON, JEFFERSON
```

After the first round, there is no winner but FRANKLIN receives no votes and is immediately eliminated, but this does not change the ballots. In the second round, JEFFERSON received only 1 first place vote and so is eliminated leaving the ballots as:

```
WASHINGTON ADAMS, WASHINGTON ADAMS, WASHINGTON, ADAMS, ADAMS
```

After this round, WASHINGTON has over half of the remaining 5 ballot votes with 3 and is declared the winner.

Input

```
3
4
WASHINGTON
ADAMS
JEFFERSON
MADISON
10
WASHINGTON ADAMS JEFFERSON MADISON
WASHINGTON JEFFERSON ADAMS MADISON
ADAMS WASHINGTON JEFFERSON MADISON
JEFFERSON WASHINGTON ADAMS MADISON
WASHINGTON ADAMS JEFFERSON MADISON
ADAMS WASHINGTON JEFFERSON MADISON
JEFFERSON WASHINGTON ADAMS MADISON
MADISON WASHINGTON JEFFERSON
ADAMS
ADAMS
3
WASHINGTON
ADAMS
JEFFERSON
3
WASHINGTON JEFFERSON
ADAMS
JEFFERSON WASHINGTON
3
WASHINGTON
ADAMS
JEFFERSON
11
WASHINGTON JEFFERSON
WASHINGTON JEFFERSON ADAMS
WASHINGTON ADAMS JEFFERSON
WASHINGTON JEFFERSON
ADAMS JEFFERSON
ADAMS JEFFERSON
ADAMS
ADAMS JEFFERSON
JEFFERSON ADAMS WASHINGTON
JEFFERSON WASHINGTON ADAMS
JEFFERSON
```

Output

```
WASHINGTON
ERROR
TIE
```