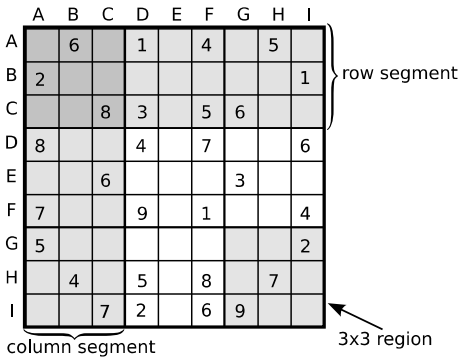
Problem A: Sudoku

Oh no! Bill just realized that the sudoku puzzle he had spent the last ten minutes trying to solve essentially was last week's puzzle, only rotated counterclockwise. How cheap! Couldn't the magazine afford to make a new one every week? Of course, he had no way of knowing about this before he started to solve it, as the holes to fill with digits were other than last week. Nevertheless, realizing that this week's puzzle was a simple derivative of last week's certainly took the fun out of solving the rest of it.



The sudoku board consists of 9×9 cells. These can be grouped into 3×3 *regions* of 3×3 cells each. Some of the cells are filled with a digit 1 through 9 while the rest of them are left empty. The aim of the game is to fill each empty cell with a digit $1 \ldots 9$ so that every row, every column and every region contains each of the numbers $1 \ldots 9$ exactly once. A proper sudoku puzzle always has exactly one solution.

Help Bill avoid unpleasant surprises by creating a program that checks whether an unsolved sudoku puzzle is in fact derived from an earlier puzzle by simple operations.

The allowed operations are:

1. Rotating the entire puzzle clockwise or counterclockwise.
2. Swapping two columns within a 3×9 column segment.
3. Swapping two rows within a 9×3 row segment.
4. Swapping entire row or column segments.
5. Applying a permutation f of the digits $1 \ldots 9$ to every cell (i.e. replace x by $f(x)$ in every cell).

An operation is considered being performed on the sudoku solution (rather than on the unsolved puzzle) and always guarantees that if the board before the transformation was a solution to a sudoku puzzle, it still is afterwards.

Input

The input starts with the number of test cases $0 \leq N \leq 50$ on a single line.

Then for every test case follow nine lines describing last week's puzzle solution, from top to bottom. Each line corresponds to a row in the puzzle and consists of nine digits ($1 \ldots 9$), describing the contents of the cell from left to right.

Last week's solution is followed by nine lines describing this week's unsolved puzzle. Here, also, every line corresponds to a puzzle row and every digit ($0 \ldots 9$) describes the contents of a cell. 0 indicates that the cell is empty. The rows are presented ordered from top to bottom, and within each row, the cells are ordered from left to right.

After every test case except the last one follows a blank line. Every unsolved puzzle is guaranteed to be uniquely solvable and last week's solution is always a proper sudoku solution.

Output

For every test case, output `Yes` if the sudoku puzzle can be derived from the given solved puzzle using the allowed operations, or `No` if this is not possible.

Sample input	**Sample output**
2 963174258 178325649 254689731 821437596 496852317 735961824 589713462 317246985 642598173 060104050 200000001 008305600 800407006 006000300 700901004 500000002 040508070 007206900 534678912 672195348 198342567 859761423 426853791 713924856 961537284 287419635 345286179 010900605 025060070 870000902 702050043 000204000 490010508 107000056 040080210 208001090	Yes No

Problem B
He is Offside!

Source file name: `web.c, web.cpp` *or* `web.java`

Hemisphere Network is the largest television network in Tumbolia, a small country located east of South America (or south of East America). The most popular sport in Tumbolia, unsurprisingly, is soccer; many games are broadcast every week in Tumbolia.
Hemisphere Network receives many requests to replay dubious plays; usually, these happen when a player is deemed to be *offside* by the referee. An attacking player is *offside* if he is nearer to his opponents' goal line than the second last opponent. A player is *not* offside if

- he is level with the second last opponent or
- he is level with the last two opponents.

Through the use of computer graphics technology, Hemisphere Network can take an image of the field and determine the distances of the players to the defending team's goal line, but they still need a program that, given these distances, decides whether a player is offside.

Input

The input file contains several test cases. The first line of each test case contains two integers A and D separated by a single space indicating, respectively, the number of attacking and defending players involved in the play ($2 \leq A, D \leq 11$). The next line contains A integers B_i separated by single spaces, indicating the distances of the attacking players to the goal line ($1 \leq B_i \leq 10^4$). The next line contains D integers C_j separated by single spaces, indicating the distances of the defending players to the goal line ($1 \leq C_j \leq 10^4$). The end of input is indicated by $A = D = 0$.

The input must be read from standard input.

Output

For each test case in the input print a line containing a single character: "`Y`" (uppercase) if there is an attacking player offside, and "`N`" (uppercase) otherwise.

The output must be written to standard output.

Sample Input	**Sample output**
23 500 700 700 500 500 2 2 200 400 200 1000 3 4 530 510 490 480 470 50 310 0 0	N Y N

C - Do It Wrong, Get It Right

Source file name: `doit.c`, `doit.cpp` *or* `doit.java`

In elementary school, students learn to subtract fractions by first getting a common denominator and then subtracting the numerators. However, sometimes a student will work the problem incorrectly and still arrive at the correct answer. For example, for the problem

$$\frac{5}{4} - \frac{9}{12}$$

one can subtract the numbers in the numerator and then subtract the numbers in the denominator, simplify and get the answer. i.e.

$$\frac{5}{4} - \frac{9}{12} = \frac{-4}{-8} = \frac{4}{8} = \frac{1}{2}.$$

For a given fraction b/n, your task is to find all of the values a and m, where $a \geq 0$ and $m > 0$, for which

$$\frac{a}{m} - \frac{b}{n} = \frac{a-b}{m-n}.$$

Input

There will be several test cases in the input. Each test case will consist of a single line with two integers, b and n ($1 \leq b, n \leq 10^6$) separated by a single space. The input will end with a line with two 0s.

The input must be read from standard input.

Output

For each case, output all of the requested fractions on a single line, sorted from smallest to largest. For equivalent fractions, print the one with the smaller numerator first. Output each fraction in the form a/m with no spaces immediately before or after the /. Output a single space between fractions. Output no extra spaces, and do not separate answers with blank lines.

The output must be written to standard output.

Sample input	**Sample output**
9 12 12 14 4 12 0 0	0/24 5/20 8/16 8/8 5/4 0/28 9/21 9/7 0/24 3/18 3/6

E - Triangle Count

Source file name: `triangle.c`, `triangle.cpp` *or* `triangle.java`

Given an equilateral triangle with side length N divided into a triangular grid of triangles with side length 1, count the total number of triangles present in the grid.

Input

The input will contain several test cases; each test case will be in a line containing an integer N ($1 \leq N \leq 500$).

The input must be read from standard input.

Output

Print a line with the answer, follow the format below.

The output must be written to standard output.

Sample input	**Sample output**
2	5
3	13
4	27
5	48

Problem G: Prime Path

The ministers of the cabinet were quite upset by the message from the Chief of Security stating that they would all have to change the four-digit room numbers on their offices.
— It is a matter of security to change such things every now and then, to keep the enemy in the dark.
— But look, I have chosen my number 1033 for good reasons. I am the Prime minister, you know!
— I know, so therefore your new number 8179 is also a prime. You will just have to paste four new digits over the four old ones on your office door.
— No, it's not that simple. Suppose that I change the first digit to an 8, then the number will read 8033 which is not a prime!
— I see, being the prime minister you cannot stand having a non-prime number on your door even for a few seconds.
— Correct! So I must invent a scheme for going from 1033 to 8179 by a path of prime numbers where only one digit is changed from one prime to the next prime.

Now, the minister of finance, who had been eavesdropping, intervened.
— No unnecessary expenditure, please! I happen to know that the price of a digit is one pound.
— Hmm, in that case I need a computer program to minimize the cost. You don't know some very cheap software gurus, do you?
— In fact, I do. You see, there is this programming contest going on...
Help the prime minister to find the cheapest prime path between any two given four-digit primes! The first digit must be nonzero, of course. Here is a solution in the case above.

```
1033
1733
3733
3739
3779
8779
8179
```

The cost of this solution is 6 pounds. Note that the digit 1 which got pasted over in step 2 can not be reused in the last step – a new 1 must be purchased.

Input

One line with a positive number: the number of test cases (at most 100). Then for each test case, one line with two numbers separated by a blank. Both numbers are four-digit primes (without leading zeros).

Output

One line for each case, either with a number stating the minimal cost or containing the word `Impossible`.

Sample input	**Sample output**
3 1033 8179 1373 8017 1033 1033	6 7 0