P21. Dead Fraction

(Time Limit: 3 seconds)

Mike is frantically scrambling to finish his thesis at the last minute. He needs to assemble all his research notes into vaguely coherent form in the next 3 days. Unfortunately, he notices that he had been extremely sloppy in his calculations. Whenever he needed to perform arithmetic, he just plugged it into a calculator and scribbled down as much of the answer as he felt was relevant. Whenever a repeating fraction was displayed, Mike simply recorded the first few digits followed by "…". For instance, instead of "1/3" he might have written down "0.3333…". Unfortunately, his results require exact fractions! He doesn't have time to redo every calculation, so he needs you to write a program (and FAST!) to automatically deduce the original fractions.

To make this tenable, he assumes that the original fraction is always the simplest one that produces the given sequence of digits; by simplest, he means the the one with smallest denominator. Also, he assumes that he did not neglect to write down important digits; no digit from the repeating portion of the decimal expansion was left unrecorded (even if this repeating portion was all zeroes).

Input

There are several test cases. For each test case there is one line of input of the form "0.*dddd* . . ." where *dddd* is a string of 1 to 9 digits, not all zero. A line containing 0 follows the last case.

Output

For each case, output the original fraction.

Note that an exact decimal fraction has two repeating expansions (e.g. 1/5 = 0.2000 . . . = 0.19999 . . .).

Sample Input

0.2...

0.20...

0.474612399...

0

Sample Input

2/9

1/5

1186531/2500000

P22. Maximum Sum

(Time Limit: 3 seconds)

A problem that is simple to solve in one dimension is often much more difficult to solve in more than one dimension. Consider satisfying a boolean expression in conjunctive normal form in which each conjunct consists of exactly 3 disjuncts. This problem (3-SAT) is NP-complete. The problem 2-SAT is solved quite efficiently, however. In contrast, some problems belong to the same complexity class regardless of the dimensionality of the problem.

Given a 2-dimensional array of positive and negative integers, find the sub-rectangle with the largest sum. The sum of a rectangle is the sum of all the elements in that rectangle. In this problem the subrectangle with the largest sum is referred to as the maximal sub-rectangle.

A sub-rectangle is any contiguous sub-array of size 1×1 or greater located within the whole array. As an example, the maximal sub-rectangle of the array:

$$\begin{array}{rrrr} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{array}$$

is in the lower-left-hand corner:

$$\begin{array}{rr} 9 & 2 \\ -4 & 1 \\ -1 & 8 \end{array}$$

and has the sum of 15.

Input

The input consists of an $N \times N$ array of integers.

The input begins with a single positive integer N on a line by itself indicating the size of the square two dimensional array. This is followed by N^2 integers separated by white-space (newlines and spaces). These N^2 integers make up the array in row-major order (i.e., all numbers on the first row, left-to-right, then all numbers on the second row, left-to-right, etc.). N may be as large as 100. The numbers in the array will be in the range $[-127, 127]$.

Output

The output is the sum of the maximal sub-rectangle.

Sample Input

```
4
0 -2 -7 0 9 2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Sample Output