ACM Scholastic Programming Contest, November 10, 1989

European Regional Finals

Problem A

Multiplying by Rotation

Warning: Not all numbers in this problem are decimal numbers!

Multiplication of natural numbers in general is a cumbersome operation. In some cases however the product can be obtained by moving the last digit to the front.

Example: 179487 * 4 = 717948

Of course this property depends on the numbersystem you use, in the above example we used the decimal representation. In base 9 we have a shorter example:

17 * 4 = 71 (base 9)

as (9 + 7) * 4 = 7 * 9 +1

Input

The input for your program is a textfile. Each line consists of three numbers separated by a space: the base of the number system, the least significant digit of the first factor, and the second factor. This second factor is one digit only hence less than the base. The input file ends with the standard end-of-file marker.

Output

Your program determines for each input line the number of digits of the smallest first factor with the rotamultproperty. The output-file is also a textfile. Each line contains the answer for the corresponding input line.

Sample Input

```
10 7 4
9 7 4
17 14 12
```

Sample Output

```
6
4
2
```

ACM Scholastic Programming Contest, November 10, 1989

European Regional Finals

Problem B

Nesting a Bunch of Brackets

In this problem we consider expressions containing brackets that are properly nested. These expressions are obtained by juxtaposition of properly netsted expressions in a pair of matching brackets, the left one an opening and the right one a closing bracket.

`( a + $ ( b = ) ( a ) )` is properly nested

`( a + $ ) b = ) ( a ( )` is not.

In this problem we have several pairs of brackets, so we have to impose a second condition on the expression: the matching brackets should be of the same kind. Consequently `(())` is OK, but `([))` is not. The pairs of brackets are:

```
(     )
[     ]
{     }
<     >
(*    *)
```

The two characters '`(*`' should be interpreted as one symbol, not as an opening bracket '`(`' followed immediately by an asterisk, and similarly for '`*)`'. The combination '`(*)`' should be interpreted as '`(*`' followed by '`)`'.

Write a program that checks wheter expressions are properly nested. If the expression is not properly nested your program should determine the position of the offending bracket, that is the length of the shortest prefix of the expression that can not be extended to a properly nested expression. Don't forget '`(*`' counts as one, as does '`*)`'. The characters that are not brackets also count as one.

Input

The input is a text-file. Each line contains an expression to be checked followed by and end-of-line marker. No line contains more than 3000 characters. The input ends with a standard end-of-file marker.

Output

The output is a textfile. Each line contains the result of the check of the corresponding inputline, that is `YES` (in upper case), if the expression is OK, and (if it is not OK) `NO` followed by a space and the position of the error.

Sample Input

```
(*a++(*)
(*a{+}*)
```

Sample Output

```
NO 6
YES
```

ACM Scholastic Programming Contest, November 10, 1989

European Regional Finals

Problem C

Filling the Gaps

At the largest conference on coding and cryptography the following theorem needed a proof or a counterexample: Suppose you are given a set of words of equal length; each word consisting of `0`'s, `1`'s and/or `*`'s. Furthermore suppose the pattern of `*`'s is different for all words in the set. By this we mean: if you replace all `0`'s and `1`'s by say `$` you obtain different words.

The claim is: if you replace the `*`'s by `0`'s and `1`'s in all possible ways, then you obtain a set that is at least as big as the set you started with.

Example:

`{ 10*, *0*, *00 }` produces `{ 100, 101, 000, 001 }`

`{ 100, 101, 10* }` produces `{ 100, 101 }`

Notice that the set in the latter example does not satisfy the condidtion mentioned above, so it does not provide a counterexample.

You program has to check for a number of cases:

1. Whether the pattern of `*`'s is different for all words in the set and:
2. Compute the number of words obtained by replacing the `*`'s by `0`'s and `1`'s.

The words will not be longer than 15 symbols.

Input

The input is a text-file that presents a sequence of sets. Each set is described as follows. The first line gives two integers: the length of the words and the number of the words. Then follow the words, each on a separate line. The end of the sequence of sets is indicated by a set with wordlength 0 and number of words equal to 0.

Output

The output is a textfile that contains one line for each set. if the pattern of `*`'s is different for all the words in this set this line should contain `YES` (in uppercase), followed by a space and the number of obtained words, otherwise it should contain `NO` (uppercase) only.

Sample Input

```
3 3
10*
*0*
*00
4 3
1100
1101
110*
0 0
```

Sample Output

```
YES 4
NO
YES 0
```

ACM Scholastic Programming Contest, November 10, 1989

European Regional Finals

Problem D

Simply proportion

Wou will probably have made a document on a word processor. When the job was done and you wanted to print that document you sometimes have the option to print the document with proportional spacing. This means that gaps between letters and words are filled with very small spaces (as small as your printer can handle) in such a way that you do not see a ragged appearance of a line but instead the line is left and right justified on your printout. This could also be done by filling each line with normal spaces, but this usually produces big gaps between words. In the following we call these small spaces **dots**.

In the exercise you are asked to produce an algorithm that fills lines with dots in such a manner that the line has a certain length (measured in dots). We will provide you with lines which contain spaces and certain letters. Each character has a width that can be measured in dots. In order to keep this exercise simple we will only use a subset of letters. These letters are the following:

Character	Width
A	18
B	17
I	10
M	20
S	16
Y	13
'space'	Variable

The minimum number of dots between letters in a word is 3. The number of dots that separates letters in a word is given by the greatest **possible** number that is equal to or smaller than 1/3 of the minimum number of dots that separate words (rounded down to the nearest integer). The minimum number of dots for a single space is 10(there is no upper limit for the number of dots in a space). Note: the begin and end of a line may not contain (empty) dots.

It can (and according to Murphy's law it will) happen that you are left with a number of dots that can't be equally divided in the gaps between letters or words. These leftovers must be equally divided over the spaces between the words beginning from the end of the line.

Input

The input for your program is a text-file. Each input consists of two lines. On the first line there is an integer (say *N*) that tells you the desired length of the line (measured in dots) that will be your output. This integer has a maximum value of 5000. The second line consists of the input that you

have to reformat to the desired length (each word is separated from the other by one space). Each line has at least two words on it and the maximum number of characters on this line is 80. There are no spaces after the last word. It is given that it is possible to fill the given line with dots so that the resulting line has a length of *N* dots. The last two lines of the input are given by:

```
0
SYMIBA
```

Your program must not perform any action on this last input, it is simply there to mark the end of the input.

Output

Your program must convert the given line to the desired length (*N* dots). The ouput is a text-file. Each line corresponds to the second line of each input pair with the number of dots between the letters and words. The spaces between letters and words are denoted in a special manner. If there are 3 dots between the letters `A` and `B` this is denoted by `A/(3)B` in your output. Your output may not contain any spaces, all these spaces must be converted to dots.

Sample Input

```
250
AIM SSY ABABA
200
SSSS AAAA
130
AA B AA
0
SYMBIA
```

Sample Output

```
A/(4)I/(4)M/(18>S/(4)S/(4)Y/(19)A/(4)B/(4)A/(4)B/(4)A
S/(7)S/(7)S/(7)S/(22)A/(7)A/(7)A/(7)A
A/(5)A/(16)B/(16)A/(5)A
```