P1 :: Exponent This!

An integer N is *exponent unique* if its prime decomposition contains no prime factor **p** that appears the same number of times as another prime factor **q**.

For example:

36	= 4 * 9	$= 2^2 * 3^2$	not *exponent unique* → **false**
200	= 8 * 25	$= 2^3 * 5^2$	is *exponent unique* → **true**

Given a number N (in the range $1 < N \leq 2147483647$) determine whether N is *exponent unique*. You should observe that if N is a prime number, then it is also *exponent unique*.

Input Format

Your program will read from standard input. You will receive only one line of input which will contain a single positive integer by itself that represents the value N.

Output Format

Your program will write to standard output. Your program should produce just a single line of output.

If the input number N is *exponent unique*, then your program must output the string **true** on a line by itself. If N is not *exponent unique* then your program must output the string **false** on a line by itself.

Sample Input and Corresponding Sample Output

Sample Input	Sample Output
199	true
200	true
201	false
207	true
36	false

P2 :: Sequence This!

You are to write a program that takes as input a string of digits, $\mathbf{S_0}$, and then calculates the next five terms of the sequence ($\mathbf{S_1}$ through $\mathbf{S_5}$) where sequence term $\mathbf{S_{n+1}}$ is derived by “describing” term $\mathbf{S_n}$.

For example, if $\mathbf{S_0}$ is “10444221” then $\mathbf{S_0}$ is described from left to right as

“one 1, one 0, three 4’s, two 2’s, one 1”

This string is converted into digits from left to right resulting in an $\mathbf{S_1}$ value of “1110342211”.

Input Format

Your program will read from standard input. You will receive only one line of input which will contain a non-empty sequence of digits (ranging from ‘0’ ≤ d_i ≤ ‘9’) representing the base term $\mathbf{S_0}$. The length of the input string $\mathbf{S_0}$ will be no greater than 10.

Output Format

Your program will write to standard output. Your program will output five lines where each line contains a string of digits (with no spaces) on the line by itself.

Sample Input and Corresponding Sample Output

Sample Input	**Sample Output**

`26`	`1216`
	`11121116`
	`31123116`
	`132112132116`
	`11131221121113122116`

`9`	`19`
	`1119`
	`3119`
	`132119`
	`1113122119`

`0`	`10`
	`1110`
	`3110`
	`132110`
	`1113122110`

`11111111`	`81`
	`1811`
	`111821`
	`31181211`
	`132118111221`

P3 :: Table This!

You are to write a program that prints the truth table for a Boolean expression. Your program must compute the following Boolean operators over Boolean arguments **p**, **q**, and **r**. In the truth table below, seven Boolean operators are shown.

p	q
0	0
0	1
1	0
1	1

p AND q
0
0
0
1

p IMPL q
1
1
0
1

p NAND q
1
1
1
0

p NOR q
1
0
0
0

p OR q
0
1
1
1

p XNOR q
1
0
0
1

p XOR q
0
1
1
0

Thus, “1 NOR 0” evaluates to the binary digit 0, while “0 NOR 0” evaluates to 1. The Boolean expression “1 NOR 0 XOR 1 AND 1” evaluated from left to right is 1.

(((1 **NOR** 0) **XOR** 1) **AND** 1)
((0 **XOR** 1) **AND** 1)
(1 **AND** 1)
1

A Boolean expression (over variables **p**, **q**, and **r**) is computed from left to right. Thus the truth table with these three variables contains eight entries. For example, the truth table for “**p** AND **q**” over these three variables is:

p	q	r
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

p AND q
0
0
0
0
0
0
1
1

Input Format

Your program will read from standard input. You will receive only one line of input which will contain a Boolean expression E over the variables **p**, **q**, and **r** using the Boolean operators (**AND**, **IMPL**, **NAND**, **NOR**, **OR**, **XNOR**, **XOR**). The variables and operators are separated by a single space. Note that the operators are always represented in CAPITAL LETTERS and the variables are in lower case.

Output Format

Your program will write to standard output. Your program should produce (in total) nine lines of output. The first line of output is the table header “`p q r E`” (note the capitalization). The next eight lines of output contain the truth table where the first three columns contain all eight potential values of **p**, **q**, and **r** (from “`0 0 0`” to “`1 1 1`”) and the final column contains the computed values of expression E for these values of **p**, **q**, and **r**.

Sample Input and Corresponding Sample Output

Sample Input	**Sample Output**
`p AND q`	`p q r E`
	`0 0 0 0`
	`0 0 1 0`
	`0 1 0 0`
	`0 1 1 0`
	`1 0 0 0`
	`1 0 1 0`
	`1 1 0 1`
	`1 1 1 1`
`p XOR q XOR p`	`p q r E`
	`0 0 0 0`
	`0 0 1 0`
	`0 1 0 1`
	`0 1 1 1`
	`1 0 0 0`
	`1 0 1 0`
	`1 1 0 1`
	`1 1 1 1`
`q NOR r IMPL p`	`p q r E`
	`0 0 0 0`
	`0 0 1 1`
	`0 1 0 1`
	`0 1 1 1`
	`1 0 0 1`
	`1 0 1 1`
	`1 1 0 1`
	`1 1 1 1`

P4 :: Climb This!

A WORD LADDER is formed from an *anchor* word that is transformed, one letter at a time, to form a *ladder* of words that ultimately reaches a *target* word. Note that the order of the unaffected letters cannot change. A sample word ladder appears to the right. Your program must determine if a sequence of words forms a WORD LADDER.

`HEAD`	*anchor* word. replace D with L.
`HEAL`	replace H with V.
`VEAL`	replace A with I.
`VEIL`	replace E with A.
`VAIL`	replace V with T.
`TAIL`	*target* word.

Input Format

Your program will read from standard input. The first line will contain a single positive integer *n* by itself that represents the number of lines *n* that will appear in the input. Each of the next *n* lines will contain a single word of uppercase letters by itself. These *n* lines form the proposed set of words that may or may not form a WORD LADDER.

You may assume: (1) all words have between three and ten uppercase letters; (2) all words in the input have the same number of letters as every other word; (3) $2 < n < 20$.

Output Format

Your program must write to standard output. If a WORD LADDER exists, then your program must output only the words "WORD LADDER" on a single line by itself. If no such WORD LADDER exists, then your program must output two lines. The first must contain the words "NO LADDER" on a single line by itself. The second must contain the words "BROKEN BY X" on a line by itself where X is the *first* word that "breaks" the WORD LADDER.

Sample Input and Corresponding Sample Output

Sample Input	**Sample Output**

`3`
`BOY`
`TOY`
`TOE`
	`WORD LADDER`

`6`
`HEAD`
`HEAL`
`VEAL`
`VEIL`
`VAIL`
`TAIL`
	`WORD LADDER`

`4`
`CAME`
`CARE`
`CANT`
`WENT`
	`NO LADDER`
	`BROKEN BY CANT`

P5 :: Histogram This!

Given a set of lines of English text (including spaces and punctuation), you are to report the top ten most frequently used characters (regardless of capitalization) and their count totals. For these calculations, you are to ignore all characters other than the (lowercase and uppercase) letters ‘a’ through ‘z’. If two letters have the exact same count total, then you should order the letters alphabetically in order (i.e., the letter ‘a’ should be output before ‘b’).

Input Format

Your program will read from standard input. You will receive a series of lines containing an arbitrary number of characters including spaces (though each line will have no more than 80 characters). The final line of input will be a single line containing a single “.” (period) on a line by itself. The input will always contain at least ten different letters.

Output Format

Your program will write to standard output. Your program will produce ten lines of output of the form “character count” where the counts are decreasing and ties are ordered alphabetically.

Sample Input and Corresponding Sample Output

Sample Input

```
Given a set of lines of English text (including
spaces and punctuation), you are to report the
top ten most frequent letters
.
```

Sample Output

```
e 14
t 14
n 10
o 8
s 7
i 6
a 5
r 5
u 5
l 4
```

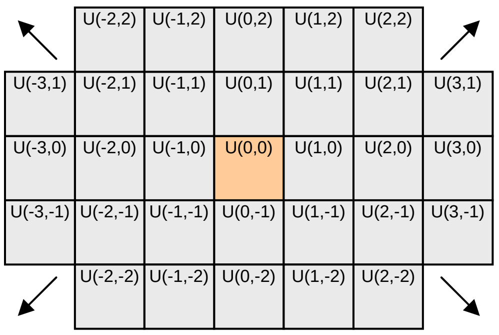
Sample Input

```
abcDEFghijKL
.
```

Sample Output

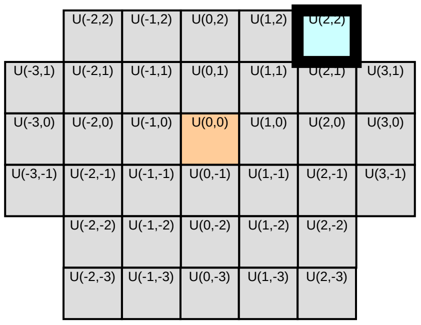
```
a 1
b 1
c 1
d 1
e 1
f 1
g 1
h 1
i 1
j 1
```

P6 :: Box This!

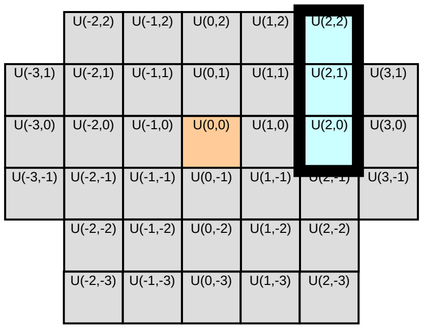

U(-2,2) U(-1,2) U(0,2) U(1,2) U(2,2)
U(-3,1) U(-2,1) U(-1,1) U(0,1) U(1,1) U(2,1) U(3,1)
U(-3,0) U(-2,0) U(-1,0) U(0,0) U(1,0) U(2,0) U(3,0)
U(-3,-1) U(-2,-1) U(-1,-1) U(0,-1) U(1,-1) U(2,-1) U(3,-1)
U(-2,-2) U(-1,-2) U(0,-2) U(1,-2) U(2,-2)

You are the operator of a fleet of unmanned drone spacecraft. You must track the movement of a set of drones moving in an imaginary two-dimensional plane divided into square unit regions. Each region is determined by an (x,y) label. The center square is U(0,0).

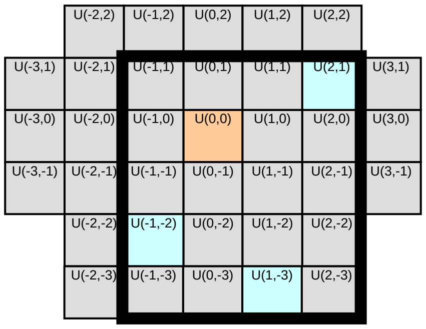
Given a set of drones in the plane, we define the concept of a *BoundingRectangle* to be the smallest rectangle (with purely horizontal and vertical edges) that wholly contains all drones in the fleet. The area of the rectangle is the sum total of the unit regions contained within the BoundingRectangle. Assume there is a drone fleet composed of three drones:


U(-2,2) U(-1,2) U(0,2) U(1,2) U(2,2)
U(-3,1) U(-2,1) U(-1,1) U(0,1) U(1,1) U(2,1) U(3,1)
U(-3,0) U(-2,0) U(-1,0) U(0,0) U(1,0) U(2,0) U(3,0)
U(-3,-1) U(-2,-1) U(-1,-1) U(0,-1) U(1,-1) U(2,-1) U(3,-1)
U(-2,-2) U(-1,-2) U(0,-2) U(1,-2) U(2,-2)
U(-2,-3) U(-1,-3) U(0,-3) U(1,-3) U(2,-3)

If the drones are all located within the region U(2,2), then the area of the BoundingRectangle is 1.


U(-2,2) U(-1,2) U(0,2) U(1,2) U(2,2)
U(-3,1) U(-2,1) U(-1,1) U(0,1) U(1,1) U(2,1) U(3,1)
U(-3,0) U(-2,0) U(-1,0) U(0,0) U(1,0) U(2,0) U(3,0)
U(-3,-1) U(-2,-1) U(-1,-1) U(0,-1) U(1,-1) U(2,-1) U(3,-1)
U(-2,-2) U(-1,-2) U(0,-2) U(1,-2) U(2,-2)
U(-2,-3) U(-1,-3) U(0,-3) U(1,-3) U(2,-3)

If the drones are located within regions U(2,0), U(2,2), and U(2,1) then the area of the BoundingRectangle is 3.


U(-2,2) U(-1,2) U(0,2) U(1,2) U(2,2)
U(-3,1) U(-2,1) U(-1,1) U(0,1) U(1,1) U(2,1) U(3,1)
U(-3,0) U(-2,0) U(-1,0) U(0,0) U(1,0) U(2,0) U(3,0)
U(-3,-1) U(-2,-1) U(-1,-1) U(0,-1) U(1,-1) U(2,-1) U(3,-1)
U(-2,-2) U(-1,-2) U(0,-2) U(1,-2) U(2,-2)
U(-2,-3) U(-1,-3) U(0,-3) U(1,-3) U(2,-3)

If the drones are located within regions U(-1,-2), U(2,1), and U(1,-3) then the area of the BoundingRectangle is 20.

Each drone is assigned an initial U(x,y) region and a velocity determined by two integers (dx, dy). After one second of flight, a drone located originally in the U(x,y) region moves itself to the region U(x+dx, y+dy). You can assume that all drone movements are perfectly precise and that drone's cannot interfere and hit each other, even if two drones find themselves located at the exact same (x,y) location at the exact same time.

Your task is to project the movement of the fleet of drones 180 seconds into the future given their original locations and initial velocity. You must identify the earliest time (in seconds) when the BoundingRectangle of the fleet has the smallest area. Note that it may be possible for two distinct drone fleet configurations to have BoundingRectangles with the same minimal area; if this happens, you must report the earliest one that occurs. You should only compute the BoundingRectangle at second intervals; do not concern yourself with fractions of a second.

Input Format

Your program will read from standard input. The first line of input will contain a positive integer N on a line by itself. Thereafter, there will be N lines each containing four integers "x y dx dy" separated by spaces that represent the location (x,y) of a drone and its initial velocity (dx, dy). It is guaranteed that 2 < N < 10.

The N lines therefore contain the location (at time T=0) of the N drones and their initial velocity.

Output Format

Your program will write to standard output. Your output will be a single line containing the two integers T and M separated by a space. The integer T represents the time 0 ≤ T ≤ 180 and the second integer M represents the area of the smallest BoundingRectangle of the drone fleet, which occurs at time T. In the event there is a tie, then output the smallest T (i.e., the earliest) for which the minimum BoundingRectangle occurs.

Sample Input and Corresponding Sample Output

Sample Input	**Sample Output**

`2`
`-2 2 1 0`
`4 -2 -1 1`
	`3 2`

`3`
`-2 3 1 -1`
`-2 0 1 0`
`1 -3 0 1`
	`3 1`

`4`
`-20 -17 1 2`
`-11 15 1 -1`
`90 1 -5 1`
`60 -20 -2 3`
	`10 459`

`3`
`-2 2 1 0`
`-2 0 1 0`
`-2 -2 1 0`
	`0 5`

P7 :: Compute This!

You have been asked to write a program that computes the value of small polynomial functions. A polynomial is a mathematical expression involving a sum of powers for a variable multiplied by coefficients. A polynomial in one variable with integer coefficients a_i is given by the formula: $a_n x^n + \ldots + a_2 x^2 + a_1 x + a_0$.

This polynomial can be represented by the string "a_nx^n+…+a_2x^2+a_1x+a_0". For example, the polynomial $3x^5 - 5x^2 + 3$ is represented by the string "`3x^5-5x^2+3`" (where the carat character "^" represents the exponent). Your program will read a polynomial representation of *f(x)* and then print the value of *f(x)* for a given integer *x*. If a coefficient is 1, it is omitted, as with the polynomial $x^5 - x^2 + 9$ which is represented by the string "`x^5-x^2+9`". If a power is 1, it is omitted, as with the polynomial x-2 represented by the string "`x-2`".

Input Format

Your program will read two lines from standard input. The first line will contain a string representing a polynomial *f(x)*. The second line will contain an integer *x* such that $-10 \le x \le 10$. You can assume that the input is properly formatted and the polynomial powers from left-to-right are in decreasing order. No power appears multiple times but some powers may be missing. Coefficients cannot be zero and are guaranteed to be between -5 and 5 inclusively. Each power value appearing after the "^" character will be a single digit from 2 to 5. The variable '*x*' will always appear in lower case.

Output Format

Your program must write to standard output a single integer on a line by itself that represents the value *f(x)*.

Sample Input and Corresponding Sample Output

Sample Input	**Sample Output**

`3x^5-5x^2+3`
`2`
	`79`

`x-2`
`-5`
	`-7`

`5x^5-4x^4+3x^3`
`2`
	`120`

`x^3-x^2+x+1`
`9`
	`658`

`7`
`2`
	`7`

`-3x^2+9`
`3`
	`-18`