The 31st Annual ACM International Collegiate Programming Contest

Sponsored by IBM

Arab and North African Ninth Regional Contest

Al-Akhawayn University
Ifrane, Morocco
December, 2006

The problem set is made of 16 numbered pages

[A] Being Smarty!

Program:	`smarty.(c|cpp|java)`
Input:	`smarty.in`
Balloon Color:	`Red (Rouge)`

Description

A web *template engine* is a software that is designed to process web templates and content information to produce web documents. A template is an html page, but without the content. In a way, a template system facilitates the separation between *the information* in a web page, and *the presentation* of it.
A template system normally comes with a (restricted and specialized) programming language to allow the variation of the presentation depending on certain properties of the information. For example, when presenting a bank statement, the bank may decide to display in red any transaction with an amount above $1,000 in order to grab the attention of the user.
Another technique, frequently used in printing tables, is to alternate the background color of rows to make it easier for the reader to visually follow a row. For example, the background color in Table (a) alternates after each row, while in Table (b) the color alternates every three rows.
A properly designed template language would have a construct to allow the designer to alternate the properties of table rows. In this problem we shall concentrate on one such construct that takes three arguments: N, P_1, and P_2. The template engine would then apply P_1 on the first N rows, P_2 on the second N rows, and then back to P_1 on the third N rows, and so on.
Write a program that takes the current row number (starting at one,) the number N, and properties P_1, and P_2 and determines which of P_1 or P_2 should be applied to the current row.

Properties of Elements

Sym.	Name	At.#
Ac	actinium	89
Ag	silver	47
Al	aluminum	13
Am	americium	95
Ar	argon	18
As	arsenic	33
At	astatine	85
Au	gold	79
B	boron	5
Ba	barium	56
Be	beryllium	4
Bi	bismuth	83

Table (a)

Properties of Elements

Sym.	Name	At.#
Ac	actinium	89
Ag	silver	47
Al	aluminum	13
Am	americium	95
Ar	argon	18
As	arsenic	33
At	astatine	85
Au	gold	79
B	boron	5
Ba	barium	56
Be	beryllium	4
Bi	bismuth	83

Table (b)

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a separate line. Each line specifies four values: R, N, P_1, and P_2, all separated by one or more spaces.

R is the current row number (first row is numbered 1) while N is as described above. Note that $0 < R, N < 1,000,000,000$.

P_1 and P_2 are properties. A property is a string made of upper- or lower-case letters, digits, and/or spaces. A property may be surrounded by double quotes, (but the double quotes are not part of the property.) If a property contains spaces, the surrounding double quotes are mandatory. No property will be longer than 512 characters (including the double quotes, if present.)

The last line of the input file is made of a single zero.

Output Format

For each test case, output the result on a single line using the following format:

```
k.␣result
```

Where `k` is the test case number (starting at 1,) and `result` is P_1 or P_2. Note that the double quotes are never printed. In addition, all letters are printed in lower case.

Sample Input/Output

smarty.in

```
1 1 red blue
2 1 red blue
3 1 red blue
6 3 "big font" "small font"
0
```

OUTPUT

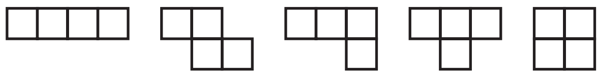
```
1. red
2. blue
3. red
4. small font
```

[B] The Game of Tetris

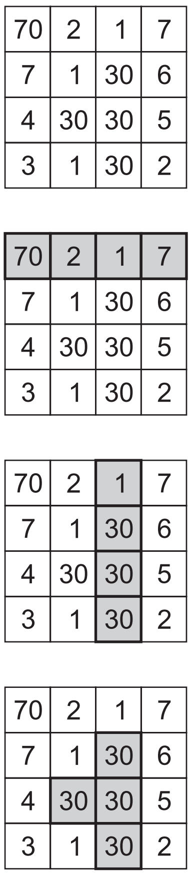
Program:	`tetris.(c|cpp|java)`
Input:	`tetris.in`
Balloon Color:	`Orange (Orange)`

Description

Tetris is a famous video game that has 5 pieces similar to these:



In this problem, you're given an `NxN` grid of integers. We want to place a single piece of Tetris on the grid such that the sum of the numbers below the piece are the maximum. Notice that all but the last Tetris piece can be rotated by 90 degrees. Some pieces even have four different orientations. Any orientation is acceptable as long as the piece completely fits inside the grid. For example, the left-most piece can be placed on the first row of the grid, with a sum of 80. It can also be placed, for example, on the third column, yielding a sum of 91. As a matter of fact, in a `4x4` grid, we can have 77 different ways to place the Tetris pieces. In the sample grid shown on the right, the largest sum that can be achieved is 120.
Write a program that determines the largest such sum for a given grid.



70	2	1	7
7	1	30	6
4	30	30	5
3	1	30	2

70	2	1	7
7	1	30	6
4	30	30	5
3	1	30	2

70	2	1	7
7	1	30	6
4	30	30	5
3	1	30	2

70	2	1	7
7	1	30	6
4	30	30	5
3	1	30	2

Input Format

Your program will be tested on one or more test cases. The first line of a test case has a single integer `N` denoting the grid size where $4 \le N \le 100$. The grid will be specified using `N` lines starting on the second line in a row major format. Each line will have `N` integers separated by one or more spaces. The absolute value of each integer in the grid will not exceed 1,000,000.
The end of the input cases is specified by a zero on a separate line.

Output Format

For each test case, output the result on a single line using the following format:

```
k.␣result
```

Where `k` is the test case number (starting at 1,) and `result` is the largest sum that can be obtained.

Sample Input/Output

tetris.in

```
4
70  2  1  7
 7  1 30  6
 4 30 30  5
 3  1 30  2
0
```

OUTPUT

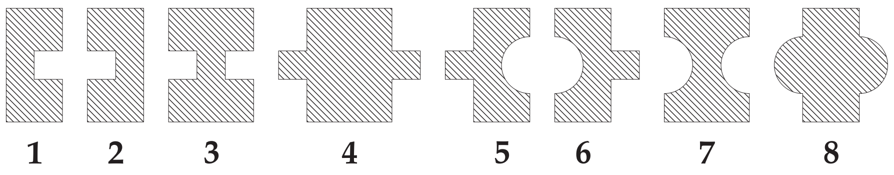
```
1. 120
```

[C] Wooden Blocks

Program:	`blocks.(c|cpp|java)`
Input:	`blocks.in`
Balloon Color:	`Yellow (Jaune)`

Description

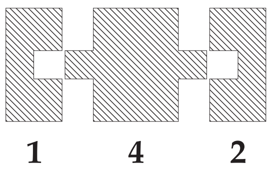
BLOCKS is a game where you're given wooden pieces that come in eight shapes as shown below:

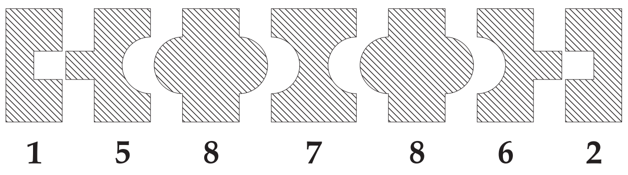


The objective of the game is to assemble the widest rectangle that can be made of a subset of the given pieces with the following conditions:

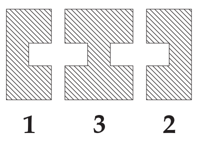
1. The rectangle must have smooth edges. In other words, the left-most piece must be (piece #1) and the right-most piece must be (piece #2.)
2. Adjacent pieces must interlock properly. For example, either piece #4 or piece #5 must appear to the right of piece #1. Similarly, piece #4 can appear to the right of piece #1 or piece #3.
3. No piece interlocks to the left of piece #1. No piece interlocks to the right of piece #2.
4. For each piece #1, the rectangle must have a matching piece #2. Similarly, for each piece #5, there must be a matching piece #6.

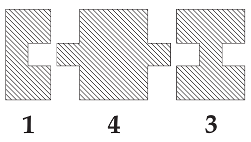
For example, the following two examples are valid arrangements:

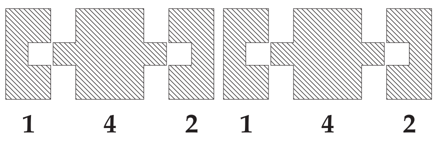




Whereas the following three are not:







A computer company is interested in building a BLOCKS video game and has hired you to write a program that determines if a given pieces' arrangement is valid according to the rules above, or not.

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a separate input line. Each piece is specified using the digit associated with it as in the previous figure. An arrangement is specified by listing its digits with no spaces between the digits. Each arrangement will have at least one piece, but no more than 10,000 pieces.

The last line in the input file will have a single 0. That line is not part of the test cases.

Output Format

For each test case, output the result on a single line using the following format:

```
k.␣result
```

Where `k` is the test case number (starting at 1,) and `result` is `"VALID"` if the arrangement is valid, or `"NOT"` if it's not.

Sample Input/Output

blocks.in

```
142
1587862
132
143
0
```

OUTPUT

```
1. VALID
2. VALID
3. NOT
4. NOT
```

[D] Walk Like an Egyptian

Program:	`walk.(c|cpp|java)`
Input:	`walk.in`
Balloon Color:	`Green (Vert)`

Description

Walk Like an Egyptian is an old multi-player board game played by children of the Sahara nomad tribes. Back in the old days, children would collect stones, and number each one of them. A game with N players requires N^2 stones. Each player chooses N stones. The stones are then laid out on an $N \times N$ grid in a peculiar order as in Figure (a) (for $N = 4$.) The player whose stone is placed in the top-right corner loses the round. Another round is then played but with $N-1$ players. In total, $N-1$ rounds are played to determine the winner.

16	15	14	13
5	6	7	12
4	3	8	11
1	2	9	10

Figure (a)

There is a story why the stones are arranged in this order. It is said that back in the days of the Pharaohs, when entering a dark room in a Pyramid, workers would use the following "algorithm" to be able to walk in the room without losing anybody: (see Figure (b)).

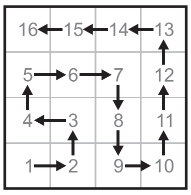


Figure (b)

1. The first worker stands in the lower-left corner of the room.
2. The next three workers stand around the first forming a quarter of a circle by going in an anti-clockwise direction.
3. The next five workers stand around the last three, again forming a quarter of a circle but this time going in a clockwise direction.
4. The workers keep repeating the last two steps until the room is filled with workers. Each time they hit the left or bottom walls, they start a larger quarter circle and alternate their direction between clockwise and anti-clockwise.

Write a program that determines the stone placed on the top-right corner.

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a separate input line. Each test case will specify the number of players N where $0 < N < 1,000$.

The end of the test cases is indicated by a line made of a single zero.

Output Format

For each test case, output the result on a single line using the following format:

```
N␣=>␣result
```

Where `N` is the number of players for the this test case, and `result` is the number on the stone placed at the top-right corner of the grid.

Sample Input/Output

walk.in

```
4
2
0
```

OUTPUT

```
4 => 13
2 => 3
```

[E] The Great Sahara

Program:	`sahara.(c|cpp|java)`
Input:	`sahara.in`
Balloon Color:	`Purple (Violet)`

Description

Sahara is a two player board game played on a hexagon-shaped grid made out of 54 triangles as the one shown in Figure (a). Each player has 6 (tetraeder) pyramids, initially placed as seen Figure (b). Player one has the dark pyramids, player two has the lighter ones. The players take turns in moving one of their own pyramids. A pyramid is moved by tipping the pyramid on its side into an adjacent space. For example, a pyramid at location 11 can be moved to location 3, 10, or 12 (assuming the destination location is free.)
The objective of the game is to trap a pyramid of the opponent. A pyramid is trapped if it can't be moved. For example, a pyramid at location 11 is trapped if locations 3, 10, and 12 are all occupied (regardless of which player's pyramids occupy these locations.) Similarly, a pyramid at location 28 is trapped if both locations 17 and 29 are occupied. For example, in Figure (c) on the next page, player one can win the game by moving his pyramid from location 30 to location 29 and thus trapping the opponent's pyramid at location 28.
Write a program that determines if the first player can trap an opponent's pyramid in a single move.

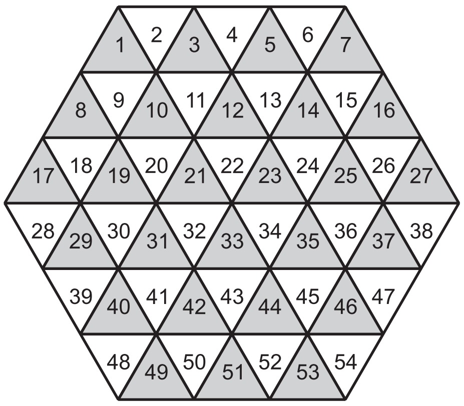


Figure (a)

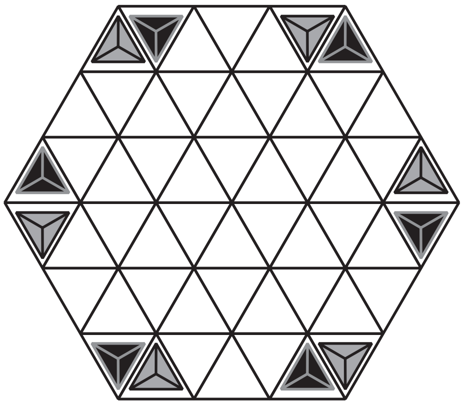


Figure (b)

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a single line. Each test case is made of 12 numbers in the range `[1,54]`. The first six numbers specify the locations of the first player's pyramids. The last six are for the second player. The locations are numbered in the same way as in Figure (a). Numbers are separated using one or more spaces. All test cases specify a valid game position where no pyramid is already trapped.
The last line of the input file will have a single zero.

Output Format

For each test case, output the result on a single line using the following format:

```
k.␣result
```

Where `k` is the test case number (starting at 1,) and `result` is `"TRAPPED"` if the first player can trap a pyramid of the opponent by moving one of his pyramids. Otherwise, `result` is `"FREE"`.

Sample Input/Output

sahara.in
```
30 2 7 44 22 45   28 41 46 17 27 1
30 2 7 44 22 45   28 41 46 18 27 1
0
```

OUTPUT
```
1. TRAPPED
2. FREE
```

The first test case corresponds to figure (c) while the second to figure (d).

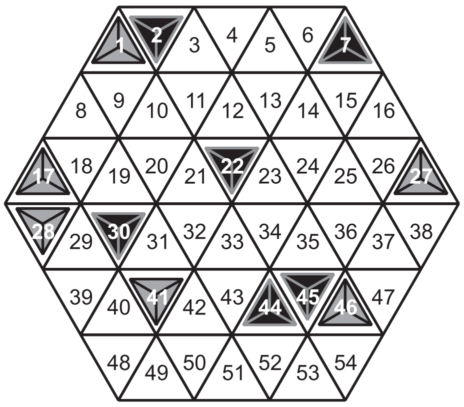


Figure (c)

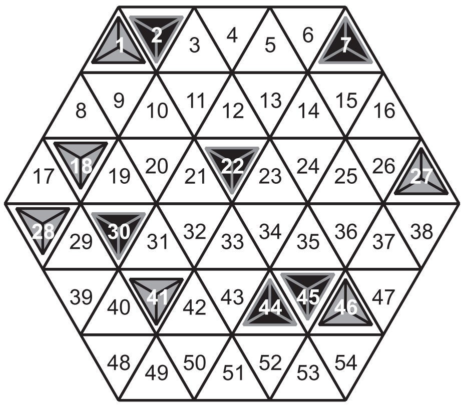


Figure (d)

[F] Johnny Hates Number Theory

Program:	`prime.(c|cpp|java)`
Input:	`prime.in`
Balloon Color:	`White (Blanc)`

Description

Johnny hates Number Theory! Actually, back in 2002, we came to know that Johnny couldn't count and in 2005 we knew that Johnny couldn't yet add. (But we did know in 2003 that Johnny was street smart enough to solve difficult graph problems!) Why Johnny decided to study Number Theory is incomprehensible to us.

Anyhow, back to Johnny. Johnny just failed his comprehensive exam and that was all because of Euler's Totient function (φ). Johnny is so angry that he decides to create his own Totient function. Here's how he described it to his advisor:

In number theory, the prime factors of a positive integer are the prime numbers that divide into that integer exactly, without leaving a remainder. Johnny defines function $F(n)$, for $n \geq 2$, to be the non-decreasing list of prime numbers whose product is n. For example, $F(8) = \ll 2, 2, 2 \gg$, $F(60) = \ll 2, 2, 3, 5 \gg$, and $F(71) = \ll 71 \gg$ (71 is a prime.) Let $O(n)$ be the length of the list $F(n)$ (i.e. its ordinal.) For example, $O(8) = 3$, $O(60) = 4$, and $O(71) = 1$. Johnny also defines function $p(n)$ over positive integers as follows:

$$p(n) = \begin{cases} 0 & \text{if } n = 1. \\ -1 & \text{if } n \text{ is a prime number.} \\ O(n) & \text{otherwise.} \end{cases}$$

The following table illustrates $p(n)$ for the first twenty positive integers:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$p(n)$	0	-1	-1	2	-1	2	-1	3	2	2	-1	3	-1	2	2	4	-1	3	-1	3

Given two positive integers a and b where $a \leq b$, Johnny defines his very own Totient function $\varphi(a, b)$ as follows:

$$\varphi(a, b) = \left(\sum_{k=a}^{b} p(k) \right) - (b - a + 1)$$

For example, $\varphi(1, 4) = -4$, $\varphi(16, 16) = 3$, and $\varphi(8, 12) = 4$.

For his dissertation, Johnny needs a program that determines the maximal φ within a given range $[L, U]$. In other words, given two positive integers L, U such that $L \leq U$, the program must find the maximum $\varphi(a, b)$ where $L \leq a \leq b \leq U$. For example, the maximal φ within the range $[1, 20]$ is 7 (which is $\varphi(8, 16)$.)

Write the program Johnny needs!

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a single line. Each test case is specified using two positive integers L and U separated by one or more spaces, and satisfying the following property: $1 \leq L \leq U < 1,000,000$

The end of the test cases is indicated by a line made of two -1's. That last line is is not part of the test cases.

Output Format

For each test case, output the result on a single line using the following format:

```
k.␣result
```

Where `k` is the test case number (starting at 1,) and `result` is the maximal φ that can be found within the range $[L, U]$.

Sample Input/Output

prime.in

```
1 5
1 20
10 20
900000 901000
-1 -1
```

OUTPUT

```
1. 1
2. 7
3. 5
4. 2551
```

[G] Minimum Spanning Tree

Program:	`mst.(c|cpp|java)`
Input:	`mst.in`
Balloon Color:	`Dark Blue (Blue)`

Description

Given graph `G` which is a connected, weighted, undirected graph, a *spanning tree* `T` is a subgraph of `G` which is: (1) a tree that (2) connects all the vertices of `G` together. The *weight* of a spanning tree is the sum of the weights of the edges in that tree. A *minimum spanning tree* is a spanning tree: (3) whose weight is less than or equal to the weight of every other spanning tree.

Write a program that determines if a given tree `T` is a Minimum Spanning Tree for a given graph `G`.

Input Format

Your program will be tested on one or more test cases. For each test case you'll be given a graph `G` and one or more trees to test. The first line of a test case will have a single positive integer n denoting the number of vertices in `G` (where $1 < n \leq 1000$). The vertices are numbered starting from 1. The next $(n-1)$ lines specify the upper triangle of the graph's adjacency matrix as seen here:

$$\begin{array}{lllll} W_{1,2} & W_{1,3} & \dots & W_{1,n-1} & W_{1,n} \\ W_{2,3} & W_{2,4} & \dots & W_{2,n} & \\ \vdots & & & & \\ W_{n-1,n} & & & & \end{array}$$

where $W_{i,j}$ is the weight of the edge between vertices i and j. $W_{i,j} = 0$ iff there is no edge between i and j. Note that $0 \leq W_{i,j} \leq 1000$

Following the graph specification, a test case will specify a single positive number Q on a separate line where $0 < Q \leq 1000$. Q denotes the number of trees to test on the given graph.

Each tree either consists of a single vertex, given by its number, or is specified as:

$$(\ R\ \ T_1\ \ T_2\ \dots\ T_c\ )$$

where R is the number of the vertex at the root and $T_1,\dots,T_c$ (where $0 < c \leq 1000$) are the sub-trees of R specified recursively.

The last line of the input file will have a single zero.

Output Format

For each query, write the result on a separate line using the following format:

```
a.b.␣result
```

where `a` is the test case number (starting at 1,) and `b` is the query number *within* this test case (again starting at 1.) `result` is either `"YES"` or `"NO"` indicating if the tree is a minimum spanning tree or not.

Sample Input/Output

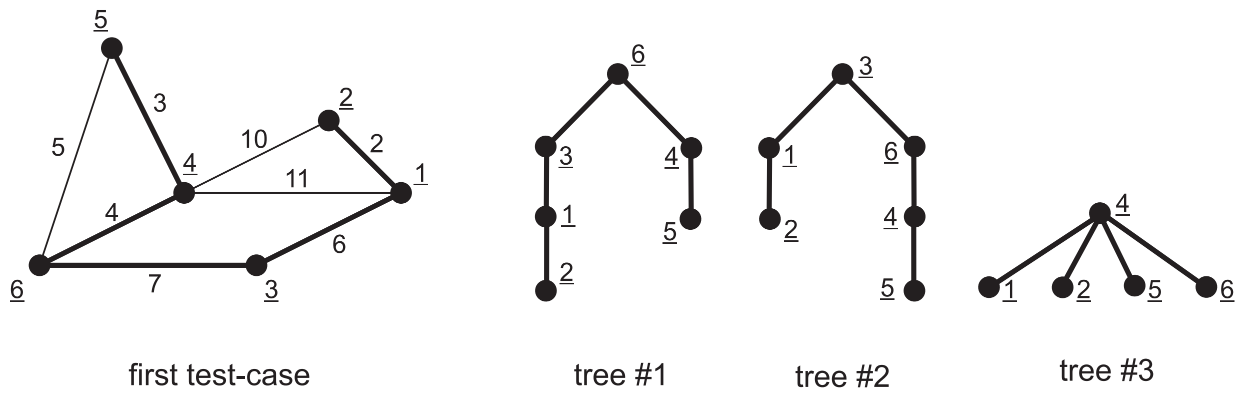
mst.in

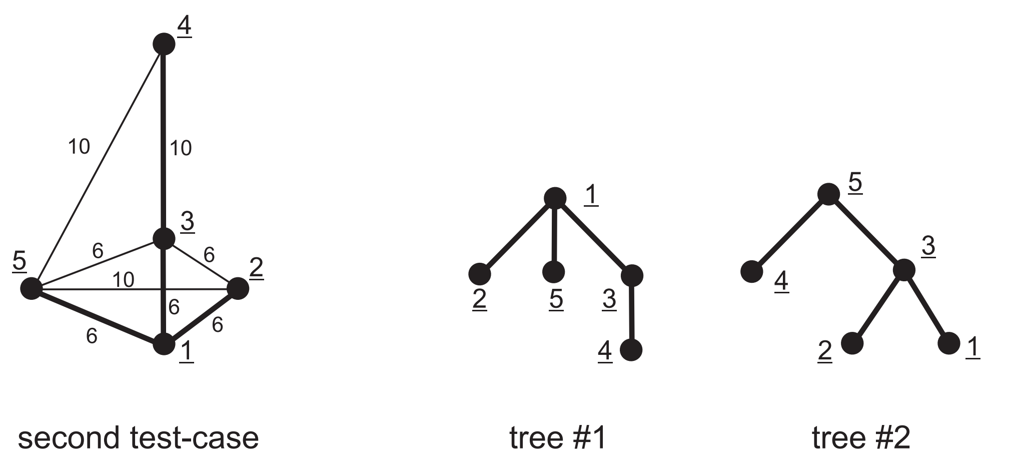
```
6
2  6 11 0 0
0 10  0 0
0  0  7
3  4
5
3
(6 (3 (1 2)) (4 5))
(3 (1 2) (6 (4 5)))
(4 1 2 5 6)
5
 6 6  0 6
 6 0 10
10 6
10
2
(1 2 5 (3 4))
(5 4 (3 2 1))
0
```

OUTPUT

```
1.1 YES
1.2 YES
1.3 NO
2.1 YES
2.2 YES
```

The following figures illustrate the sample I/O. The top half is for the first test case, while the second test case is on the bottom. In the graph, vertex numbers are underlined and the edges of a minimum spanning tree are drawn in thicker lines.



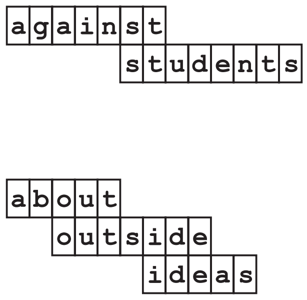


[H] A-to-Z

Program:	`atoz.(c|cpp|java)`
Input:	`atoz.in`
Balloon Color:	`Light Blue (Blue Ciel)`

Description

A-TO-Z is a game usually played by kids in elementary schools to help them improve their spelling skills and to enrich their vocabulary. The game comes with a set of words, each word written on a piece of plastic. Children challenge each other by picking two letters (let's call them C_1 and C_2) and then trying to *connect* these letters by finding a sequence of one or more words (we'll refer to them as W_1, W_2, $\ldots$, W_n) where the first word W_1 starts with C_1 and the last word W_n ends with C_2. Each two consecutive words in the sequence (W_i, W_{i+1}) must *overlap* with at least two letters. Word X *overlaps* by k letters with word Y if the last k letters of X are the same as the first k letters of Y. Take for example the figure on the right, where `'a'` was connected to `'s'` using the two-word sequence `"against"` and `"students"`.



To determine the winner of the game, each sequence is assigned a penalty which is equal to the number of letters in the sequence (but overlapping letters are counted only once.) The player with *the least penalty* wins. Going back to the figure, the first sequence `"against students"` has a penalty of 13, while the second sequence `"about outside ideas"` has a penalty of 11. You can think of the penalty as the width taken when the sequence is laid out as in the figure. The winning sequence is the one with the smallest width.

Write a program that takes a dictionary of words and determines the winning sequence connecting two given letters.

Input Format

Your program will be tested on one or more test cases. Each test case specifies a dictionary of words, and a list of one or more character pairs (called queries,) to connect using the dictionary.

The first line of a test case is a positive number w which denotes the number of words in the dictionary. The words are listed starting at the second line: One word per line. Words are made of lowercase letters only. No word is longer than 64 characters. A dictionary has at most 50,000 words.

Following the dictionary, q queries are specified using $q+1$ lines. The first line specifies the number of queries, q. Each query is specified on a separate line. Each query specifies two lowercase letters C_1 and C_2.

The end of the test cases is identified with an input line that contains a single integer $w = 0$ (which is not part of the test cases.)

Output Format

For each query, write the result on a separate line. If there is no sequence connecting the two letters, your program should print:

```
a.b␣0
```

where `a` is the test case number (starting at 1,) and `b` is the query number within this test case (again starting at 1.)

If, however, there is a sequence, your program should print the following:

```
a.b␣p␣word-1␣word-2␣...␣word-n
```

Where `a`, `b` are as described previously, `p` is the penalty of the sequence. `word-1`, `word-2`, ... `word-n` is the sequence, where `word-1` starts with C_1, `word-n` ends with C_2, and each two consecutive words overlap with at least two characters. Remember, we're interested in the sequence with the minimum possible penalty in the given dictionary.

(If there are more than one solution for a query, print any of them.)

Sample Input/Output

atoz.in

```
9
ones
against
students
about
outside
other
ideas
added
education
3
a s
o s
o t
3
aaabb
aabbbb
bbbbz
2
a z
z a
0
```

OUTPUT

```
1.1 11 about outside ideas
1.2 4 ones
1.3 0
2.1 7 aabbbb bbbbz
2.2 0
```

[I] Casting Out Nines

Program:	`hindu.(c|cpp|java)`
Input:	`hindu.in`
Balloon Color:	`Pink (Rose)`

Description

You must have heard of the great mathematician *Leonardo da Pisa Fibonacci (1170-1240).* Among the many algorithms that he described in his *Liber Abaci* book, (which was first published in 1202,) Fibonacci described the *Casting Out Nines* procedure for checking out addition, subtraction, and multiplication. According to historians, the procedure was transmitted to Europe by the Arabs, but was probably developed somewhere on the Indian subcontinent and is therefore sometimes also called *"the Hindu check."*

"Casting out nines" is an elementary check of a multiplication which makes use of the congruence $10^x \equiv 1 \pmod 9$. Remember that when writing $x \equiv y$ (mod z) we're actually saying that (x mod z) is equal to (y mod z).

Let a, b be natural numbers whose product is c. Let the sums of the digits of these numbers be $\bar{a}$, $\bar{b}$, and $\bar{c}$. Then $a \equiv \bar{a} \pmod 9$, $b \equiv \bar{b} \pmod 9$, and $c \equiv \bar{c} \pmod 9$. Furthermore $a \times b \equiv \bar{a} \times \bar{b} \pmod 9$, and so $\bar{a} \times \bar{b} \equiv \bar{c} \pmod 9$. So if c and $\bar{a} \times \bar{b}$ are incongruent (mod 9), the multiplication has been done incorrectly.

For example, $12345 \times 67890 = 838102050$. The sum-of-digits of 12345 and 67890 are 15 and 30, respectively, and the product of these is 450. Similarly, the sum-of-digits of 838102050 is 27. And since $450 \equiv 27 \equiv 0 \pmod 9$, so the Hindu Check shows agreement.

As another example, say someone incorrectly calculates $13579 \times 24680 = 334129720$. Since $\bar{a} \times \bar{b} = 25 \times 20 = 500 \equiv 5 \pmod 9$ whereas $\bar{c} = 31 \equiv 4 \pmod 9$. So the multiplication is definitely incorrect.

The Hindu check can also be used to check on additions, since $a + b \equiv \bar{a} + \bar{b} \pmod 9$.

Write a program that determines if a given addition or multiplication passes the Hindu Check or not.

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a separate input line. Each line is of the form:

```
a+b=c.
```

Or,

```
a*b=c.
```

Notice the `'.'` at the end of the line. `'a'`, `'b'`, and `'c'` are natural numbers. There are no spaces between the numbers and the symbols (`'+'`, `'*'`, `'='`, and `'.'`,) but trailing white-space may appear after the `'.'`.

The last line of the input file, which is not part of the test cases, will have a single `'.'`.

Output Format

For each test case, output the result on a single line using the following format:

```
k.␣result
```

Where `k` is the test case number (starting at 1,) and `result` is `"PASS"` if the addition or multiplication operation of this test case passes the Hindu Check. Otherwise, `result` is `"NOT!"`.

Sample Input/Output

hindu.in

```
12345*67890=838102050.
13579*24680=334129720.
23+11=34.
.
```

OUTPUT

```
1. PASS
2. NOT!
3. PASS
```