Problem A
Goldbach's Conjecture
Input: goldbach.txt

Goldbach's Conjecture: For any even number n greater than or equal to 4, there exists at least one pair of prime numbers p_1 and p_2 such that $n = p_1 + p_2$.

This conjecture has not been proved nor refused yet. No one is sure whether this conjecture actually holds. However, one can find such a pair of prime numbers, if any, for a given even number. The problem here is to write a program that reports the number of all the pairs of prime numbers satisfying the condition in the conjecture for a given even number.

A sequence of even numbers is given as input. Corresponding to each number, the program should output the number of pairs mentioned above. Notice that we are interested in the number of essentially different pairs and therefore you should not count (p_1, p_2) and (p_2, p_1) separately as two different pairs.

Input

An integer is given in each input line. You may assume that each integer is even, and is greater than or equal to 4 and less than 2^{15}. The end of the input is indicated by a number 0.

Output

Each output line should contain an integer number. No other characters should appear in the output.

Sample Input

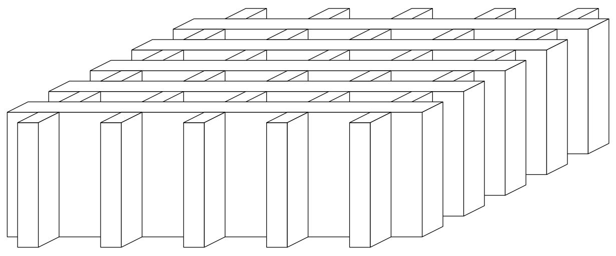
```
6
10
12
0
```

Output for the Sample Input

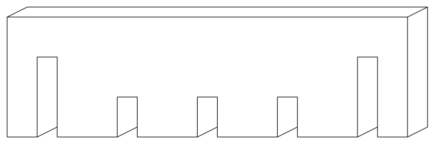
```
1
2
1
```

Problem B
Lattice Practices
Input: lattice.txt

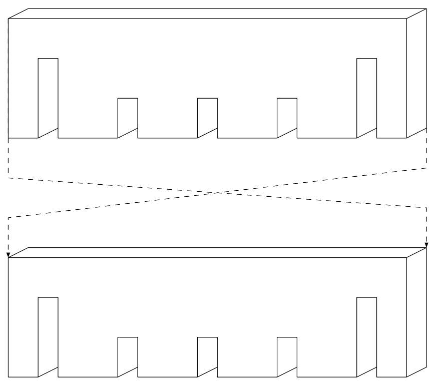
Once upon a time, there was a king who loved beautiful costumes very much. The king had a special cocoon bed to make excellent cloth of silk. The cocoon bed had 16 small square rooms, forming a 4×4 lattice, for 16 silkworms. The cocoon bed can be depicted as follows:



The cocoon bed can be divided into 10 rectangular boards, each of which has 5 slits:



Note that, except for the slit depth, there is no difference between the left side and the right side of the board (or, between the front and the back); thus, we cannot distinguish a symmetric board from its rotated image as is shown in the following:



Slits have two kinds of depth, either shallow or deep. The cocoon bed should be constructed by fitting five of the boards vertically and the others horizontally, matching a shallow slit with a deep slit.

Your job is to write a program that calculates the number of possible configurations to make the lattice. You may assume that there is no pair of identical boards. Notice that we are interested in the number of essentially different configurations and therefore you should not count mirror image configurations and rotated configurations separately as different configurations.

The following is an example of mirror image and rotated configurations, showing vertical and horizontal boards seperately, where shallow and deep slits are denoted by '1' and '0' respectively.

(the original)

1	0	0	0	1
0	1	0	0	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1

0	1	1	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	0
1	1	1	1	0

(mirror image)

0	0	0	0	1
0	0	0	1	1
0	0	1	1	1
0	1	0	0	1
1	0	0	0	1

1	1	1	1	0
1	1	1	0	0
1	1	0	0	0
1	0	1	1	0
0	1	1	1	0

(rotated 1)

1	0	0	0	0
1	1	0	0	0
1	1	1	0	0
1	0	0	1	0
1	0	0	0	1

0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	1	1	0	1
0	1	1	1	0

(rotated 2)

0	1	1	1	1
1	0	1	1	1
1	1	0	1	1
1	1	0	0	1
0	0	0	0	0

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	1	1	0
1	1	1	1	1

Notice that a rotation may exchange positions of a vertical board and a horizontal board.

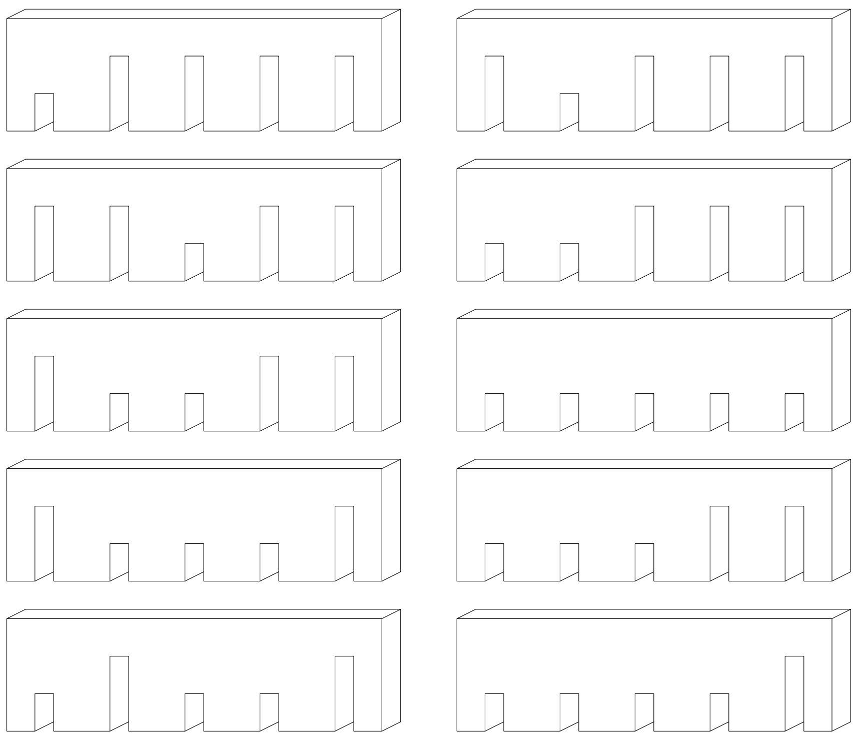
Input

The input consists of multiple data sets, each in a line. A data set gives the patterns of slits of 10 boards used to construct the lattice. The format of a data set is as follows:

xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx

Each x is either '0' or '1'. '0' means a deep slit, and '1' a shallow slit. A block of five slit descriptions corresponds to a board. There are 10 blocks of slit descriptions in a line. Two adjacent blocks are separated by a space.

For example, the first data set in the Sample Input means the set of the following 10 boards:



The end of the input is indicated by a line consisting solely of three characters "END".

Output

For each data set, the number of possible configurations to make the lattice from the given 10 boards should be output, each in a separate line.

Sample Input

```
10000 01000 00100 11000 01100 11111 01110 11100 10110 11110
```

```
10101 01000 00000 11001 01100 11101 01110 11100 10110 11010
END
```

Output for the Sample Input

```
40
6
```

Problem C
Mobile Phone Coverage
Input: mobile.txt

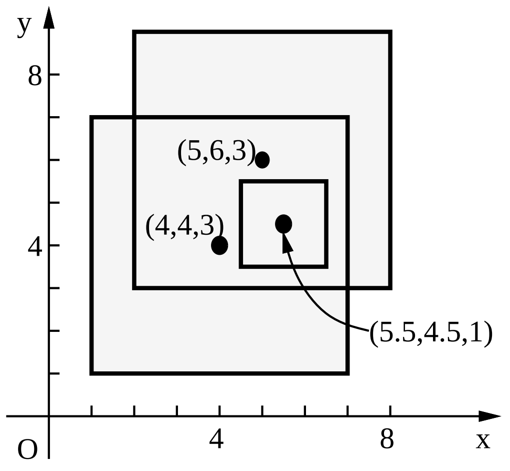
A mobile phone company ACMICPC (Advanced Cellular, Mobile, and Internet-Connected Phone Corporation) is planning to set up a collection of antennas for mobile phones in a city called Maxnorm. The company ACMICPC has several collections for locations of antennas as their candidate plans, and now they want to know which collection is the best choice.

For this purpose, they want to develop a computer program to find the coverage of a collection of antenna locations. Each antenna A_i has power r_i, corresponding to "radius". Usually, the coverage region of the antenna may be modeled as a disk centered at the location of the antenna (x_i, y_i) with radius r_i. However, in this city Maxnorm such a coverage region becomes the square $[x_i - r_i, x_i + r_i] \times [y_i - r_i, y_i + r_i]$. In other words, the distance between two points (x_p, y_p) and (x_q, y_q) is measured by the max norm $\max\{\,|x_p - x_q|,\ |y_p - y_q|\,\}$, or, the L_∞ norm, in this city Maxnorm instead of the ordinary Euclidean norm $\sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$.

As an example, consider the following collection of 3 antennas

```
4.0 4.0 3.0
5.0 6.0 3.0
5.5 4.5 1.0
```

depicted in the following figure



where the i-th row represents x_i, y_i, r_i such that (x_i, y_i) is the position of the i-th antenna and r_i is its power. The area of regions of points covered by at least one antenna is 52.00 in this case.

Write a program that finds the area of coverage by a given collection of antenna locations.

Input

The input contains multiple data sets, each representing a collection of antenna locations. A data set is given in the following format.

n
x_1 y_1 r_1
x_2 y_2 r_2
$\cdots$
x_n y_n r_n

The first integer n is the number of antennas, such that $2 \leq n \leq 100$. The coordinate of the i-th antenna is given by (x_i, y_i), and its power is r_i. x_i, y_i and r_i are fractional numbers between 0 and 200 inclusive.

The end of the input is indicated by a data set with 0 as the value of n.

Output

For each data set, your program should output its sequence number (1 for the first data set, 2 for the second, etc.) and the area of the coverage region. The area should be printed with two digits to the right of the decimal point, after rounding it to two decimal places.

The sequence number and the area should be printed on the same line with no spaces at the beginning and end of the line. The two numbers should be separated by a space.

Sample Input

```
3
4.0 4.0 3.0
5.0 6.0 3.0
5.5 4.5 1.0
2
3.0 3.0 3.0
1.5 1.5 1.0
0
```

Output for the Sample Input

```
1 52.00
2 36.00
```

Problem D
Napoleon's Grumble
Input: palin.txt

Legend has it that, after being defeated in Waterloo, Napoleon Bonaparte, in retrospect of his days of glory, talked to himself "Able was I ere I saw Elba." Although, it is quite doubtful that he should have said this in English, this phrase is widely known as a typical *palindrome.*

A palindrome is a symmetric character sequence that looks the same when read backwards, right to left. In the above Napoleon's grumble, white spaces appear at the same positions when read backwards. This is not a required condition for a palindrome. The following, ignoring spaces and punctuation marks, are known as the first conversation and the first palindromes by human beings.

> "Madam, I'm Adam."
> "Eve."
> *(by Mark Twain)*

Write a program that finds palindromes in input lines.

Input

A multi-line text is given as the input. The input ends at the end of the file.

There are at most 100 lines in the input. Each line has less than 1,024 Roman alphabet characters.

Output

Corresponding to each input line, a line consisting of *all* the character sequences that are palindromes in the input line should be output. However, trivial palindromes consisting of only one or two characters should not be reported.

On finding palindromes, any characters in the input except Roman alphabets, such as punctuation characters, digits, spaces, and tabs, should be ignored. Characters that differ only in their cases should be looked upon as the same character. Whether or not the character sequences represent a proper English word or sentence does not matter.

Palindromes should be reported all in uppercase characters. When two or more palindromes are reported, they should be separated by a space character. You may report palindromes in any order.

If two or more occurrences of the same palindromes are found in the same input line, report only once. When a palindrome overlaps with another, even when one is completely included in the other, both should be reported. However, palindromes appearing in the center of another palindrome, whether or not they also appear elsewhere, should not be reported. For example, for an input line of "AAAAAA", two palindromes "AAAAAA" and "AAAAA" should be output, but not "AAAA" nor "AAA". For "AAABCAAAAAA", the output remains the same.

One line should be output corresponding to one input line. If an input line does not contain any palindromes, an empty line should be output.

Sample Input

```
As the first man said to the
first woman:
"Madam, I'm Adam."
She responded:
"Eve."
```

Output for the Sample Input

```
TOT

MADAMIMADAM MADAM
ERE DED
EVE
```

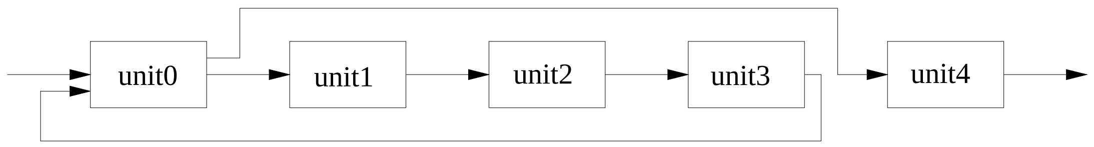
Note that the second line in the output is empty, corresponding to the second input line containing no palindromes. Also note that some of the palindromes in the third input line, namely "ADA", "MIM", "AMIMA", "DAMIMAD", and "ADAMIMADA", are not reported because they appear at the center of reported ones. "MADAM" *is* reported, as it does not appear in the center, but only once, disregarding its second occurrence.

Problem E
Pipeline Scheduling
Input: pipeline.txt

An *arithmetic pipeline* is designed to process more than one task simultaneously in an overlapping manner. It includes function units and data paths among them. Tasks are processed by *pipelining*; at each clock, one or more units are dedicated to a task and the output produced for the task at the clock is cascading to the units that are responsible for the next stage; since each unit may work in parallel with the others at any clock, more than one task may be being processed at a time by a single pipeline.

In this problem, a pipeline may have a feedback structure, that is, data paths among function units may have directed loops as shown in the next figure.

Example of a feedback pipeline



Since an arithmetic pipeline in this problem is designed as special purpose dedicated hardware, we assume that it accepts just a single sort of task. Therefore, the timing information of a pipeline is fully described by a simple table called a *reservation table*, which specifies the function units that are busy at each clock when a task is processed without overlapping execution.

Example of "reservation table"

clock	0	1	2	3	4	5	6
unit0	X	.	.	.	X	X	.
unit1	.	X	.	.	.	.	.
unit2	.	.	X	.	.	.	.
unit3	.	.	.	X	.	.	.
unit4	.	.	.	.	.	.	X

In reservation tables, 'X' means "the function unit is busy at that clock" and '.' means "the function unit is not busy at that clock." In this case, once a task enters the pipeline, it is processed by unit0 at the first clock, by unit1 at the second clock, and so on. It takes seven clock cycles to perform a task.

Notice that no special hardware is provided to avoid simultaneous use of the same function unit.

Therefore, a task must not be started if it would conflict with any tasks being processed. For instance, with the above reservation table, if two tasks, say task 0 and task 1, were started at clock 0 and clock 1, respectively, a conflict would occur on unit0 at clock 5. This means that you should not start two tasks with single cycle interval. This invalid schedule is depicted in the following process table, which is obtained by overlapping two copies of the reservation table with one being shifted to the right by 1 clock.

Example of "conflict"

clock	0	1	2	3	4	5	6	7
unit0	0	1	.	.	0	C	1	.
unit1	.	0	1	.	.	.	.	.
unit2	.	.	0	1	.	.	.	.
unit3	.	.	.	0	1	.	.	.
unit4	.	.	.	.	.	.	0	1

('0's and '1's in this table except those in the first row represent tasks 0 and 1, respectively, and 'C' means the conflict.)

Your job is to write a program that reports the minimum number of clock cycles in which the given pipeline can process 10 tasks.

Input

The input consists of multiple data sets, each representing the reservation table of a pipeline. A data set is given in the following format.

n
$x_{0,0}\ x_{0,1}\ \cdots\ x_{0,n-1}$
$x_{1,0}\ x_{1,1}\ \cdots\ x_{1,n-1}$
$x_{2,0}\ x_{2,1}\ \cdots\ x_{2,n-1}$
$x_{3,0}\ x_{3,1}\ \cdots\ x_{3,n-1}$
$x_{4,0}\ x_{4,1}\ \cdots\ x_{4,n-1}$

The integer $n (< 20)$ in the first line is the width of the reservation table, or the number of clock cycles that is necessary to perform a single task. The second line represents the usage of unit0, the third line unit1, and so on. $x_{i,j}$ is either 'X' or '.'. The former means *reserved* and the latter *free*. There are no spaces in any input line. For simplicity, we only consider those pipelines that consist of 5 function units. The end of the input is indicated by a data set with 0 as the value of n.

Output

For each data set, your program should output a line containing an integer number that is the minimum number of clock cycles in which the given pipeline can process 10 tasks.

Sample Input

```
7
X...XX.
.X.....
..X....
...X...
......X
0
```

Output for the Sample Input

```
34
```

In this sample case, it takes 41 clock cycles to process 10 tasks if each task is started as early as possible under the condition that it never conflicts with any previous tasks being processed.

	00000000001111111111222222222233333333334
clock	01234567890123456789012345678901234567890
unit0	0.1.00112.3.22334.5.44556.7.66778.9.8899.
unit1	.0.1.....2.3.....4.5.....6.7.....8.9.....
unit2	..0.1.....2.3.....4.5.....6.7.....8.9....
unit3	...0.1.....2.3.....4.5.....6.7.....8.9...
unit4	......0.1.....2.3.....4.5.....6.7.....8.9

(The digits in the table except those in the clock row represent the task number.)

However, it takes only 34 clock cycles if each task is started at every third clock.

	0000000000111111111122222222223333
clock	0123456789012345678901234567890123
unit0	0..100211322433544655766877988.99.
unit1	.0..1..2..3..4..5..6..7..8..9.....
unit2	..0..1..2..3..4..5..6..7..8..9....
unit3	...0..1..2..3..4..5..6..7..8..9...
unit4	......0..1..2..3..4..5..6..7..8..9

(The digits in the table except those in the clock row represent the task number.)

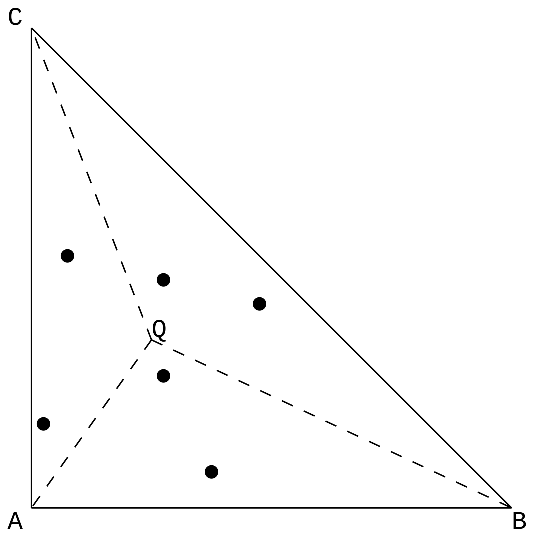
This is the best possible schedule and therefore your program should report 34 in this case.

Problem F
Triangle Partition
Input: triangle.txt

Suppose that a triangle and a number of points inside the triangle are given. Your job is to find a partition of the triangle so that the set of the given points are divided into three subsets of equal size.

Let A, B and C denote the vertices of the triangle. There are n points, P_1, P_2, ..., P_n, given inside △ABC. You are requested to find a point Q such that each of the three triangles △QBC, △QCA and △QAB contains at least $n/3$ points. Points on the boundary line are counted in both triangles. For example, a point on the line QA is counted in △QCA and also in △QAB. If Q coincides a point, the point is counted in all three triangles.

It can be proved that there always exists a point Q satisfying the above condition. The problem will be easily understood from the figure below.



Input

The input consists of multiple data sets, each representing a set of points. A data set is given in the following format.

n
x_1 y_1
x_2 y_2
...

x_n y_n

The first integer n is the number of points, such that $3 \leq n \leq 300$. n is always a multiple of 3. The coordinate of a point P_i is given by (x_i, y_i). x_i and y_i are integers between 0 and 1000.

The coordinates of the triangle ABC are fixed. They are A(0, 0), B(1000, 0) and C(0, 1000).

Each of P_i is located strictly inside the triangle ABC, not on the side BC, CA nor AB. No two points can be connected by a straight line running through one of the vertices of the triangle. Speaking more precisely, if you connect a point P_i with the vertex A by a straight line, another point P_j never appears on the line. The same holds for B and C.

The end of the input is indicated by a 0 as the value of n. The number of data sets does not exceed 10.

Output

For each data set, your program should output the coordinate of the point Q. The format of the output is as follows.

q_x q_y

For each data set, a line of this format should be given. No extra lines are allowed. On the contrary, any number of space characters may be inserted before q_x, between q_x and q_y, or after q_y.

Each of q_x and q_y should be represented by a fractional number (e.g., `3.1416`) but not with an exponential part (e.g., `6.023e+23` is not allowed). Four digits should follow the decimal point.

Note that there is no unique "correct answer" in this problem. In general, there are infinitely many points which satisfy the given condition. Your result may be any one of these "possible answers".

In your program, you should be careful to minimize the effect of numeric errors in the handling of floating-point numbers. However, it seems inevitable that some rounding errors exist in the output. We expect that there is an error of 0.5×10^{-4} in the output, and will judge your result accordingly.

Sample Input

```
3
100 500
200 500
300 500
6
100 300
100 600
200 100
200 700
```

```
500 100
600 300
0
```

Output for the Sample Input

```
166.6667  555.5556
333.3333  333.3333
```

As mentioned above, the results shown here are not the only solutions. Many other coordinates for the point Q are also acceptable. The title of this section should really be “Sample Output for the Sample Input”.

Problem G
BUT We Need a Diagram
Input: but.txt

Consider a data structure called BUT (Binary and/or Unary Tree). A BUT is defined inductively as follows:

- Let l be a letter of the English alphabet, either lowercase or uppercase (in the sequel, we say simply "a letter"). Then, the object that consists only of l, designating l as its label, is a BUT. In this case, it is called a 0-ary BUT.
- Let l be a letter and C a BUT. Then, the object that consists of l and C, designating l as its label and C as its component, is a BUT. In this case, it is called a unary BUT.
- Let l be a letter, L and R BUTs. Then, the object that consists of l, L and R, designating l as its label, L as its left component, and R as its right component, is a BUT. In this case, it is called a binary BUT.

A BUT can be represented by an expression in the following way.

- When a BUT B is 0-ary, its representation is the letter of its label.
- When a BUT B is unary, its representation is the letter of its label followed by the parenthesized representation of its component.
- When a BUT B is binary, its representation is the letter of its label, a left parenthesis, the representation of its left component, a comma, the representation of its right component, and a right parenthesis, arranged in this order.

Here are examples:

```
a
A(b)
a(a,B)
a(B(c(D),E),f(g(H,i)))
```

Such an expression is concise, *but* a diagram is much more appealing to our eyes. We prefer a diagram:

```
D  H i
-  ---
c E g
--- -
 B  f
 ----
  a
```

to the expression `a(B(c(D),E),f(g(H,i)))`.

Your mission is to write a program that converts the expression representing a BUT into its diagram. We want to keep a diagram as compact as possible assuming that we display it on a conventional character terminal with a fixed pitch font such as Courier. Let's define the diagram D for a BUT B inductively along the structure of B as follows:

- When B is 0-ary, D consists only of a letter of its label. The letter is called the root of D, and also called the leaf of D.
- When B is unary, D consists of a letter l of its label, a minus symbol S, and the diagram C for its component, satisfying the following constraints:
  - l is just below S.
  - The root of C is just above S.

  l is called the root of D, and the leaves of C are called the leaves of D.
- When B is binary, D consists of a letter l of its label, a sequence of minus symbols S, the diagram L for its left component, and the diagram R for its right component, satisfying the following constraints:
  - S is contiguous, and is in a line.
  - l is just below the central minus symbol of S, where, if the center of S locates on a minus symbol s, s is the central, and if the center of S locates between adjacent minus symbols, the left one of them is the central.
  - The root of L is just above the leftmost minus symbol of S, and the root of R is just above the rightmost minus symbol of S.
  - In any line of D, L and R do not touch or overlap each other.
  - No minus symbols are just above the leaves of L and R.

  l is called the root of D, and the leaves of L and R are called the leaves of D.

Input

The input to the program is a sequence of expressions representing BUTs. Each expression except the last one is terminated by a semicolon. The last expression is terminated by a period. White spaces (tabs and blanks) should be ignored. An expression may extend over multiple lines. The number of letters, i.e., the number of characters except parentheses, commas, and white spaces, in an expression is at most 80.

You may assume that the input is syntactically correct and need not take care of error cases.

Output

Each expression is to be identified with a number starting with 1 in the order of occurrence in the input. Output should be produced in the order of the input.

For each expression, a line consisting of the identification number of the expression followed by a colon should be produced first, and then, the diagram for the BUT represented by the expression should be produced.

For a diagram, output should consist of the minimum number of lines, which contain only letters or minus symbols together with minimum number of blanks required to obey the rules shown above.

Sample Input

```
a(A,b(B,C));
x( y( y( z(z), v( s, t ) ) ), u ) ;

a( b( c,
      d(
         e(f),
         g
       )
    ),
   h( i(
         j(
            k(k,k),
            l(l)
          ),
         m(m)
       )
    )
 );

a(B(C),d(e(f(g(h(i(j,k),l),m),n),o),p))
.
```

Output for the Sample Input

```
1:
 B C
 ---
A b
---
 a
2:
z s t
- ---
z  v
----
 y
 -
 y u
 ---
  x
3:
   k k l
   --- -
 f  k  l m
 -  ---- -
 e g j   m
 --- -----
c d    i
---    -
 b     h
 -------
    a
4:
j k
---
 i l
 ---
  h m
  ---
   g n
   ---
    f o
    ---
   C e p
   - ---
   B  d
   ----
    a
```

Problem H
Digital Racing Circuit
Input: circuit.txt

You have an ideally small racing car on an x-y plane ($0 \leq x$, $y \leq 255$, where bigger y denotes upper coordinate). The racing circuit course is figured by two solid walls. Each wall is a closed loop of connected line segments. End point coordinates of every line segment are both integers (See Figure 1). Thus, a wall is represented by a list of end point integer coordinates (x_1, y_1), (x_2, y_2), ..., (x_n, y_n). The start line and the goal line are identical.

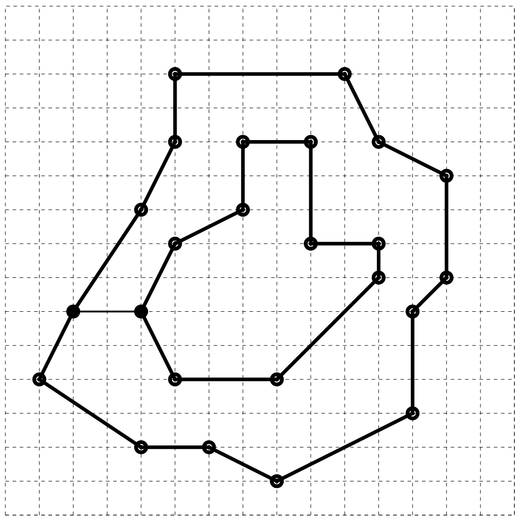


Figure 1. A Simple Course

For the qualification run, you start the car at any integer coordinate position on the start line, say (s_x, s_y).

At any clock t ($\geq$ 0), according to the acceleration parameter at t, ($a_{x,t}$, $a_{y,t}$), the velocity changes instantly to ($v_{x,t-1} + a_{x,t}$, $v_{y,t-1} + a_{y,t}$), if the velocity at $t - 1$ is ($v_{x,t-1}$, $v_{y,t-1}$). The velocity will be kept constant until the next clock. It is assumed that the velocity at clock -1, ($v_{x,-1}$, $v_{y,-1}$) is (0, 0). Each of the acceleration components must be either -1, 0, or 1, because your car does not have so fantastic engine and brake. In other words, any successive pair of velocities should not differ more than 1 for either x-component or y-component. Note that your trajectory will be piecewise linear as the walls are.

Your car should not touch nor run over the circuit wall, or your car will be crashed, even at the start line. The referee watches your car's trajectory very carefully, checking whether or not the trajectory touches the wall or attempts to cross the wall.

The objective of the qualification run is to finish your run within as few clock units as possible, without suffering from any interference by other racing cars. That is, you run alone the circuit around clockwise and reach, namely touch or go across the goal line without having been crashed. Don't be crashed even in the last trajectory segment after you reach the goal line. But we don't care whatever happens after that clock.

Your final lap time is the clock t when you reach the goal line for the first time after you have once left the start line. But it needs a little adjustment at the goal instant. When you cross the goal line, only the necessary fraction of your car's last trajectory segment is counted. For example, if the length of your final trajectory segment is 3 and only its 1/3 fraction is needed to reach the goal line, you have to add only 0.333 instead of 1 clock unit.

Drivers are requested to control their cars as cleverly as possible to run fast but avoiding crash. ALAS! The racing committee decided that it is too DANGEROUS to allow novices to run the circuit. In the last year, it was reported that some novices wrenched their fingers by typing too enthusiastically their programs. So, this year, you are invited as a referee assistant in order to accumulate the experience on this dangerous car race.

A number of experienced drivers are now running the circuit for the qualification for semi-finals. They submit their driving records to the referee. The referee has to check the records one by one whether it is not a fake.

Now, you are requested to make a program to check the validity of driving records for any given course configuration. Is the start point right on the start line without touching the walls? Is every value in the acceleration parameter list either one of -1, 0, and 1? Does the length of acceleration parameter list match the reported lap time? That is, aren't there any excess acceleration parameters after the goal, or are these enough to reach the goal? Doesn't it involve any crash? Does it mean a clockwise running all around? (Note that it is not inhibited to run backward temporarily unless crossing the start line again.) Is the lap time calculation correct? You should allow a rounding error up to 0.01 clock unit in the lap time.

Input

The input consists of a course configuration followed by a number of driving records on that course.

A course configuration is given by two lists representing the inner wall and the outer wall, respectively. Each list shows the end point coordinates of the line segments that comprise the wall. A course configuration looks as follows:

$i_{x,1}$ $i_{y,1}$ ..... $i_{x,N}$ $i_{y,N}$ 99999
$o_{x,1}$ $o_{y,1}$ ..... $o_{x,M}$ $o_{y,M}$ 99999

where data are alternating x-coordinates and y-coordinates that are all non-negative integers (≤ 255) terminated by 99999. The start/goal line is a line segment connecting the coordinates ($i_{x,1}$, $i_{y,1}$) and ($o_{x,1}$, $o_{y,1}$). For simplicity, $i_{y,1}$ is assumed to be equal to $o_{y,1}$; that is, the start/goal line is horizontal on the x-y plane. Note that N and M may vary among course configurations, but do not exceed 100, because too many curves involve much danger even for experienced drivers. You need not check the validity of the course configuration.

A driving record consists of three parts, which is termintated by 99999: two integers s_x, s_y for the start position (s_x, s_y), the lap time with three digits after the decimal point, and the sequential list of acceleration parameters at all clocks until the goal. It is assumed that the length of the acceleration parameter list does not exceed 500. A driving record looks like the following:

> s_x s_y
> *lap-time*
> $a_{x,0}$ $a_{y,0}$ $a_{x,1}$ $a_{y,1}$ ... $a_{x,L}$ $a_{y,L}$ 99999

Input is terminated by a null driving record; that is, it is terminated by a 99999 that immediately follows 99999 of the last driving record.

Output

The test result should be reported by simply printing OK or NG for each driving record, each result in each line. No other letter is allowed in the result.

Sample Input

```
6 28 6 32 25 32 26 27 26 24 6 24 99999
2 28 2 35 30 35 30 20 2 20 99999

3 28
22.667
0 1 1 1 1 0 0 -1 0 -1 1 0 0 0 1 0 -1 0 0 -1 -1 -1 -1 0 -1 0 -1 -1 -1 1 -1 1
-1 1 -1 0 1 0 1 1 1 1 1 0 1 1 99999

5 28
22.667
0 1 -1 1 1 0 1 -1 1 -1 1 0 1 0 1 0 -1 -1 -1 0 -1 -1 -1 0 -1 1 -1 -1 -1 1
-1 0 -1 1 -1 0 1 0 1 0 1 1 1 1 1 1 99999

4 28
6.333
0 1 0 1 1 -1 -1 -1 0 -1 0 -1 0 -1 99999

3 28
20.000
0 -1 1 -1 1 0 1 1 1 1 1 0 -1 0 -1 0 -1 1 -1 1
-1 1 -1 0 -1 -1 -1 -1 -1 -1 -1 0 1 0 1 -1 1 -1 1 -1 99999

99999
```

Output for the Sample Input

```
OK
NG
NG
NG
```