Problem A
Cubist Artwork
Input: A.in

International Center for Picassonian Cubism is a Spanish national museum of cubist artworks, dedicated to Pablo Picasso. The center held a competition for an artwork that will be displayed in front of the facade of the museum building. The artwork is a collection of cubes that are piled up on the ground and is intended to amuse visitors, who will be curious how the shape of the collection of cubes changes when it is seen from the front and the sides.

The artwork is a collection of cubes with edges of one foot long and is built on a flat ground that is divided into a grid of one foot by one foot squares. Due to some technical reasons, cubes of the artwork must be either put on the ground, fitting into a unit square in the grid, or put on another cube in the way that the bottom face of the upper cube exactly meets the top face of the lower cube. No other way of putting cubes is possible.

You are a member of the judging committee responsible for selecting one out of a plenty of artwork proposals submitted to the competition. The decision is made primarily based on artistic quality but the cost for installing the artwork is another important factor. Your task is to investigate the installation cost for each proposal. The cost is proportional to the number of cubes, so you have to figure out the minimum number of cubes needed for installation.

Each design proposal of an artwork consists of the front view and the side view (the view seen from the right-hand side), as shown in Figure 1.

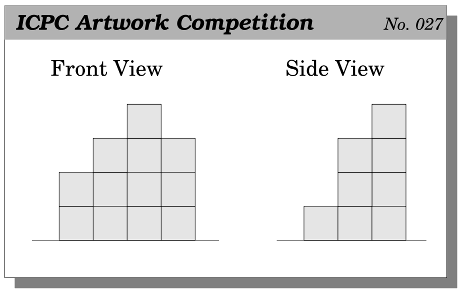
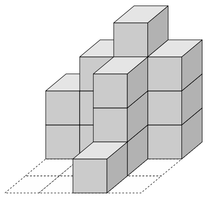
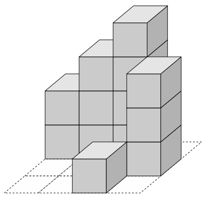


Figure 1: An example of an artwork proposal

The front view (resp., the side view) indicates the maximum heights of piles of cubes for each column line (resp., row line) of the grid.

There are several ways to install this proposal of artwork, such as the following figures.





In these figures, the dotted lines on the ground indicate the grid lines. The left figure makes use of 16 cubes, which is not optimal. That is, the artwork can be installed with a fewer number of cubes. Actually, the right one is optimal and only uses 13 cubes. Note that, a single pile of height three in the right figure plays the roles of two such piles in the left one.

Notice that swapping columns of cubes does not change the side view. Similarly, swapping rows does not change the front view. Thus, such swaps do not change the costs of building the artworks.

For example, consider the artwork proposal given in Figure 2.

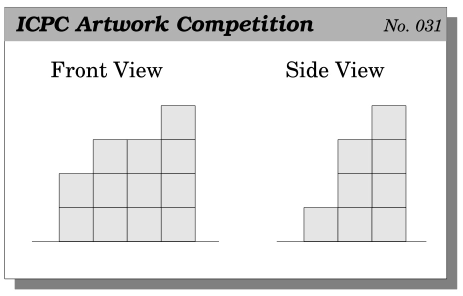
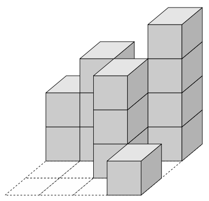


Figure 2: Another example of artwork proposal

An optimal installation of this proposal of artwork can be achieved with 13 cubes, as shown in the following figure, which can be obtained by exchanging the rightmost two columns of the optimal installation of the artwork of Figure 1.



Input

The input is a sequence of datasets. The end of the input is indicated by a line containing two zeros separated by a space. Each dataset is formatted as follows.

$$\begin{array}{llll} w & d & & \\ h_1 & h_2 & \cdots & h_w \\ h'_1 & h'_2 & \cdots & h'_d \end{array}$$

The integers w and d separated by a space are the numbers of columns and rows of the grid, respectively. You may assume $1 \leq w \leq 10$ and $1 \leq d \leq 10$. The integers separated by a space in the second and third lines specify the shape of the artwork. The integers h_i ($1 \leq h_i \leq 20$, $1 \leq i \leq w$) in the second line give the front view, i.e., the maximum heights of cubes per each column line, ordered from left to right (seen from the front). The integers h'_i ($1 \leq h'_i \leq 20$, $1 \leq i \leq d$) in the third line give the side view, i.e., the maximum heights of cubes per each row line, ordered from left to right (seen from the right-hand side).

Output

For each dataset, output a line containing the minimum number of cubes. The output should not contain any other extra characters.

You can assume that, for each dataset, there is at least one way to install the artwork.

Sample Input

```
5 5
1 2 3 4 5
1 2 3 4 5
5 5
2 5 4 1 3
4 1 5 3 2
5 5
1 2 3 4 5
3 3 3 4 5
3 3
7 7 7
7 7 7
3 3
4 4 4
4 3 4
4 3
4 2 2 4
4 2 1
4 4
2 8 8 8
2 3 8 3
10 10
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
10 9
20 1 20 20 20 20 20 18 20 20
20 20 20 20 7 20 20 20 20
0 0
```

Output for the Sample Input

```
15
15
21
21
15
13
32
90
186
```

Problem B
Repeated Substitution with Sed
Input: B.in

Do you know "sed," a tool provided with Unix? Its most popular use is to substitute every occurrence of a string α contained in the input string (actually each input line) with another string β. More precisely, it proceeds as follows.

1. Within the input string, every non-overlapping (but possibly adjacent) occurrences of α are marked. If there is more than one possibility for non-overlapping matching, the leftmost one is chosen.
2. Each of the marked occurrences is substituted with β to obtain the output string; other parts of the input string remain intact.

For example, when α is "`aa`" and β is "`bca`", an input string "`aaxaaa`" will produce "`bcaxbcaa`", but not "`aaxbcaa`" nor "`bcaxabca`". Further application of the same substitution to the string "`bcaxbcaa`" will result in "`bcaxbcbca`", but this is another substitution, which is counted as the second one.

In this problem, a set of substitution pairs (α_i, β_i) $(i = 1, 2, \ldots, n)$, an initial string γ, and a final string δ are given, and you must investigate how to produce δ from γ with a minimum number of substitutions. A single substitution (α_i, β_i) here means simultaneously substituting all the non-overlapping occurrences of α_i, in the sense described above, with β_i.

You may use a specific substitution (α_i, β_i) multiple times, including zero times.

Input

The input consists of multiple datasets, each in the following format.

n
α_1 β_1
α_2 β_2
$\vdots$
α_n β_n
γ
δ

n is a positive integer indicating the number of pairs. α_i and β_i are separated by a single space. You may assume that $1 \leq |\alpha_i| < |\beta_i| \leq 10$ for any i ($|s|$ means the length of the string s),

$\alpha_i \neq \alpha_j$ for any $i \neq j$, $n \leq 10$ and $1 \leq |\gamma| < |\delta| \leq 10$. All the strings consist solely of lowercase letters. The end of the input is indicated by a line containing a single zero.

Output

For each dataset, output the minimum number of substitutions to obtain δ from γ. If δ cannot be produced from γ with the given set of substitutions, output -1.

Sample Input

```
2
a bb
b aa
a
bbbbbbbb
1
a aa
a
aaaaa
3
ab aab
abc aadc
ad dee
abc
deeeeeeeec
10
a abc
b bai
c acf
d bed
e abh
f fag
g abe
h bag
i aaj
j bbb
a
abacfaabe
0
```

Output for the Sample Input

```
3
-1
7
4
```

Problem C
Swimming Jam
Input: C.in

Despite urging requests of the townspeople, the municipal office cannot afford to improve many of the apparently deficient city amenities under this recession. The city swimming pool is one of the typical examples. It has only two swimming lanes. The Municipal Fitness Agency, under this circumstances, settled usage rules so that the limited facilities can be utilized fully.

Two lanes are to be used for one-way swimming of different directions. Swimmers are requested to start swimming in one of the lanes, from its one end to the other, and then change the lane to swim his/her way back. When he or she reaches the original starting end, he/she should return to his/her initial lane and starts swimming again.

Each swimmer has his/her own natural constant pace. Swimmers, however, are not permitted to pass other swimmers except at the ends of the pool; as the lanes are not wide enough, that might cause accidents. If a swimmer is blocked by a slower swimmer, he/she has to follow the slower swimmer at the slower pace until the end of the lane is reached. Note that the blocking swimmer's natural pace may be faster than the blocked swimmer; the blocking swimmer might also be blocked by another swimmer ahead, whose natural pace is slower than the blocked swimmer. Blocking would have taken place whether or not a faster swimmer was between them.

Swimmers can change their order if they reach the end of the lane simultaneously. They change their order so that ones with faster natural pace swim in front. When a group of two or more swimmers formed by a congestion reaches the end of the lane, they are considered to reach there simultaneously, and thus change their order there.

The number of swimmers, their natural paces in times to swim from one end to the other, and the numbers of laps they plan to swim are given. Note that here one "lap" means swimming from one end to the other and then swimming back to the original end. Your task is to calculate the time required for all the swimmers to finish their plans. All the swimmers start from the same end of the pool at the same time, the faster swimmers in front.

In solving this problem, you can ignore the sizes of swimmers' bodies, and also ignore the time required to change the lanes and the order in a group of swimmers at an end of the lanes.

Input

The input is a sequence of datasets. Each dataset is formatted as follows.

n
t_1 c_1
$\dots$
t_n c_n

n is an integer ($1 \leq n \leq 50$) that represents the number of swimmers. t_i and c_i are integers ($1 \leq t_i \leq 300$, $1 \leq c_i \leq 250$) that represent the natural pace in times to swim from one end to the other and the number of planned laps for the i-th swimmer, respectively. t_i and c_i are separated by a space.

The end of the input is indicated by a line containing one zero.

Output

For each dataset, output the time required for all the swimmers to finish their plans in a line. No extra characters should occur in the output.

Sample Input

```
2
10 30
15 20
2
10 240
15 160
3
2 6
7 2
8 2
4
2 4
7 2
8 2
18 1
0
```

Output for the Sample Input

```
600
4800
36
40
```

Problem D
Separate Points
Input: D.in

Numbers of black and white points are placed on a plane. Let's imagine that a straight line of infinite length is drawn on the plane. When the line does not meet any of the points, the line divides these points into two groups. If the division by such a line results in one group consisting only of black points and the other consisting only of white points, we say that the line "separates black and white points".

Let's see examples in Figure 3. In the leftmost example, you can easily find that the black and white points can be perfectly separated by the dashed line according to their colors. In the remaining three examples, there exists no such straight line that gives such a separation.

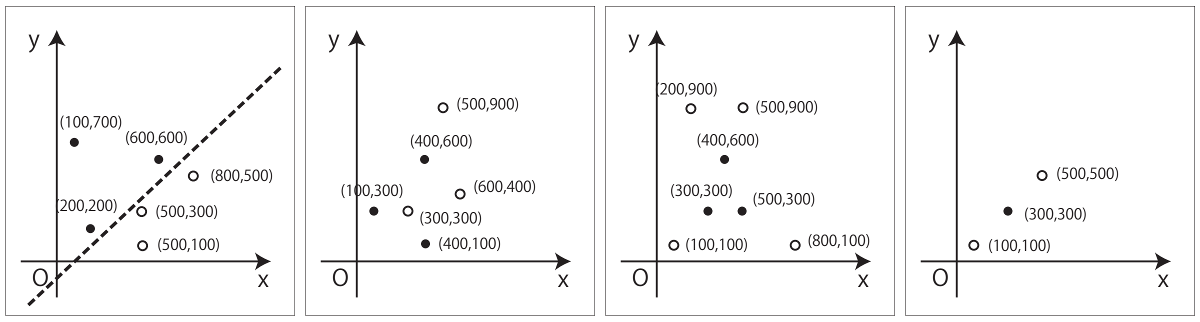


Figure 3: Example planes

In this problem, given a set of points with their colors and positions, you are requested to decide whether there exists a straight line that separates black and white points.

Input

The input is a sequence of datasets, each of which is formatted as follows.

n m
x_1 y_1
$\vdots$
x_n y_n
x_{n+1} y_{n+1}
$\vdots$
x_{n+m} y_{n+m}

The first line contains two positive integers separated by a single space; n is the number of black points, and m is the number of white points. They are less than or equal to 100. Then $n + m$ lines representing the coordinates of points follow. Each line contains two integers x_i and y_i separated by a space, where (x_i, y_i) represents the x-coordinate and the y-coordinate of the i-th point. The color of the i-th point is black for $1 \leq i \leq n$, and is white for $n + 1 \leq i \leq n + m$.

All the points have integral x- and y-coordinate values between 0 and 10000 inclusive. You can also assume that no two points have the same position.

The end of the input is indicated by a line containing two zeros separated by a space.

Output

For each dataset, output "`YES`" if there exists a line satisfying the condition. If not, output "`NO`". In either case, print it in one line for each input dataset.

Sample Input

```
3 3
100 700
200 200
600 600
500 100
500 300
800 500
3 3
100 300
400 600
400 100
600 400
500 900
300 300
3 4
300 300
500 300
400 600
100 100
200 900
500 900
800 100
1 2
300 300
100 100
500 500
1 1
100 100
```

```
200 100
2 2
0 0
500 700
1000 1400
1500 2100
2 2
0 0
1000 1000
1000 0
0 1000
3 3
0 100
4999 102
10000 103
5001 102
10000 102
0 101
3 3
100 100
200 100
100 200
0 0
400 0
0 400
3 3
2813 1640
2583 2892
2967 1916
541 3562
9298 3686
7443 7921
0 0
```

Output for the Sample Input

```
YES
NO
NO
NO
YES
YES
NO
NO
NO
YES
```

Problem E
Origami Through-Hole
Input: E.in

Origami is the traditional Japanese art of paper folding. One day, Professor Egami found the message board decorated with some pieces of origami works pinned on it, and became interested in the pinholes on the origami paper. Your mission is to simulate paper folding and pin punching on the folded sheet, and calculate the number of pinholes on the original sheet when unfolded.

A sequence of folding instructions for a flat and square piece of paper and a single pinhole position are specified. As a folding instruction, two points P and Q are given. The paper should be folded so that P touches Q from above (Figure 4). To make a fold, we first divide the sheet into two segments by creasing the sheet along the *folding line*, i.e., the perpendicular bisector of the line segment PQ, and then turn over the segment containing P onto the other. You can ignore the thickness of the paper.

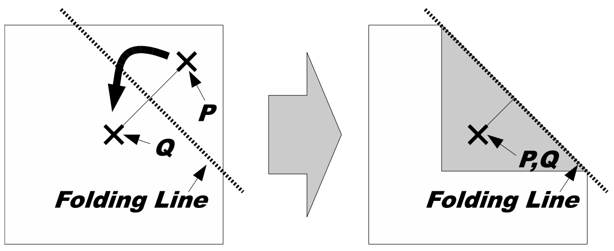


Figure 4: Simple case of paper folding

The original flat square piece of paper is folded into a structure consisting of layered paper segments, which are connected by linear hinges. For each instruction, we fold one or more paper segments along the specified folding line, dividing the original segments into new smaller ones. The folding operation turns over some of the paper segments (not only the new smaller segments but also some other segments that have no intersection with the folding line) to the reflective position against the folding line. That is, for a paper segment that intersects with the folding line, one of the two new segments made by dividing the original is turned over; for a paper segment that does not intersect with the folding line, the whole segment is simply turned over.

The folding operation is carried out repeatedly applying the following rules, until we have no segment to turn over.

- Rule 1: The uppermost segment that contains P must be turned over.
- Rule 2: If a hinge of a segment is moved to the other side of the folding line by the operation, any segment that shares the same hinge must be turned over.

- Rule 3: If two paper segments overlap and the lower segment is turned over, the upper segment must be turned over too.

In the examples shown in Figure 5, (a) and (c) show cases where only Rule 1 is applied. (b) shows a case where Rule 1 and 2 are applied to turn over two paper segments connected by a hinge, and (d) shows a case where Rule 1, 3 and 2 are applied to turn over three paper segments.

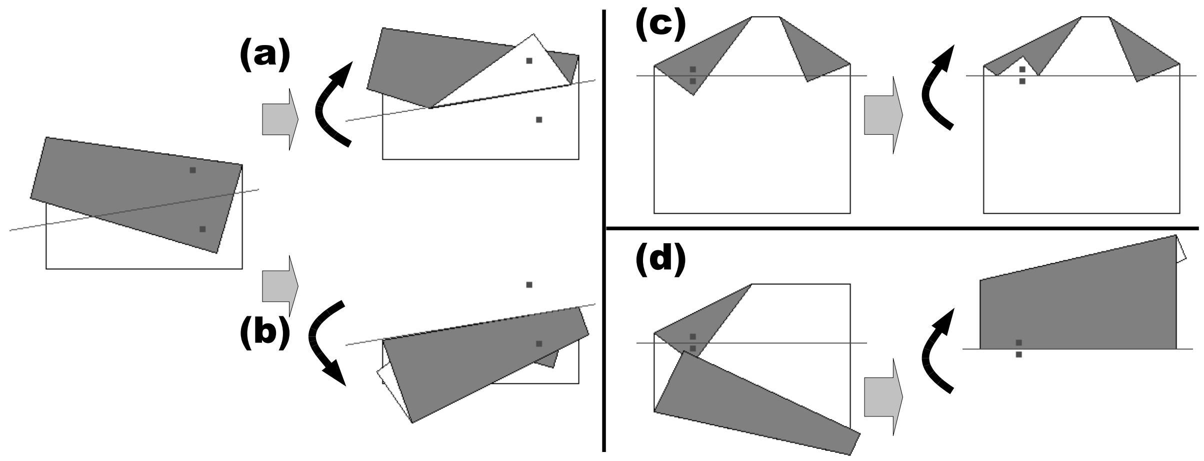


Figure 5: Different cases of folding

After processing all the folding instructions, the pinhole goes through all the layered segments of paper at that position. In the case of Figure 6, there are three pinholes on the unfolded sheet of paper.

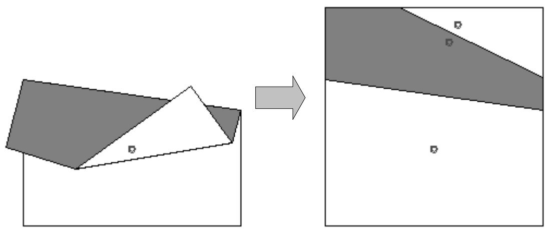


Figure 6: Number of pinholes on the unfolded sheet

Input

The input is a sequence of datasets. The end of the input is indicated by a line containing a zero.

Each dataset is formatted as follows.

$$\begin{array}{llll} k & & & \\ p_x^1 & p_y^1 & q_x^1 & q_y^1 \\ & & \vdots & \\ p_x^k & p_y^k & q_x^k & q_y^k \\ h_x & h_y & & \end{array}$$

For all datasets, the size of the initial sheet is 100 mm square, and, using mm as the coordinate unit, the corners of the sheet are located at the coordinates (0, 0), (100, 0), (100, 100) and (0, 100). The integer k is the number of folding instructions and $1 \leq k \leq 10$. Each of the following k lines represents a single folding instruction and consists of four integers p_x^i, p_y^i, q_x^i and q_y^i, delimited by a space. The positions of point P and Q for the i-th instruction are given by $(p_x^i,\ p_y^i)$ and $(q_x^i,\ q_y^i)$, respectively. You can assume that $P \neq Q$. You must carry out these instructions in the given order. The last line of a dataset contains two integers h_x and h_y delimited by a space, and $(h_x,\ h_y)$ represents the position of the pinhole.

You can assume the following properties:

- The points P and Q of the folding instructions are placed on some paper segments at the folding time, and P is at least 0.01 mm distant from any borders of the paper segments.
- The position of the pinhole also is at least 0.01 mm distant from any borders of the paper segments at the punching time.
- Every folding line, when infinitely extended to both directions, is at least 0.01 mm distant from any corners of the paper segments before the folding along that folding line.
- When two paper segments have any overlap, the overlapping area cannot be placed between any two parallel lines with 0.01 mm distance. When two paper segments do not overlap, any points on one segment are at least 0.01 mm distant from any points on the other segment.

For example, Figure 5 (a), (b), (c) and (d) correspond to the first four datasets of the sample input.

Output

For each dataset, output a single line containing the number of the pinholes on the sheet of paper, when unfolded. No extra characters should appear in the output.

Sample Input

```
2
90 90 80 20
80 20 75 50
50 35
2
```

```
90 90 80 20
75 50 80 20
55 20
3
5 90 15 70
95 90 85 75
20 67 20 73
20 75
3
5 90 15 70
5 10 15 55
20 67 20 73
75 80
8
1 48 1 50
10 73 10 75
31 87 31 89
91 94 91 96
63 97 62 96
63 80 61 82
39 97 41 95
62 89 62 90
41 93
5
2 1 1 1
-95 1 -96 1
-190 1 -191 1
-283 1 -284 1
-373 1 -374 1
-450 1
2
77 17 89 8
103 13 85 10
53 36
0
```

Output for the Sample Input

```
3
4
3
2
32
1
0
```

Problem F
Chemist's Math
Input: F.in

You have probably learnt *chemical equations* (chemical reaction formulae) in your high-school days. The following are some well-known equations.

$$2\mathrm{H}_2 + \mathrm{O}_2 \rightarrow 2\mathrm{H}_2\mathrm{O} \tag{1}$$
$$\mathrm{Ca(OH)}_2 + \mathrm{CO}_2 \rightarrow \mathrm{CaCO}_3 + \mathrm{H}_2\mathrm{O} \tag{2}$$
$$\mathrm{N}_2 + 3\mathrm{H}_2 \rightarrow 2\mathrm{NH}_3 \tag{3}$$

While Equations (1)–(3) all have balanced left-hand sides and right-hand sides, the following ones do not.

$$\mathrm{Al} + \mathrm{O}_2 \rightarrow \mathrm{Al}_2\mathrm{O}_3 \quad (\textit{wrong}) \tag{4}$$
$$\mathrm{C}_3\mathrm{H}_8 + \mathrm{O}_2 \rightarrow \mathrm{CO}_2 + \mathrm{H}_2\mathrm{O} \quad (\textit{wrong}) \tag{5}$$

The equations must follow *the law of conservation of mass*; the quantity of each chemical element (such as H, O, Ca, Al) should not change with chemical reactions. So we should "adjust" the numbers of molecules on the left-hand side and right-hand side:

$$4\mathrm{Al} + 3\mathrm{O}_2 \rightarrow 2\mathrm{Al}_2\mathrm{O}_3 \quad (\textit{correct}) \tag{6}$$
$$\mathrm{C}_3\mathrm{H}_8 + 5\mathrm{O}_2 \rightarrow 3\mathrm{CO}_2 + 4\mathrm{H}_2\mathrm{O} \quad (\textit{correct}) \tag{7}$$

The coefficients of Equation (6) are $(4, 3, 2)$ from left to right, and those of Equation (7) are $(1, 5, 3, 4)$ from left to right. Note that the coefficient 1 may be omitted from chemical equations.

The coefficients of a *correct* equation must satisfy the following conditions.

1. The coefficients are all positive integers.
2. The coefficients are relatively prime, that is, their greatest common divisor (g.c.d.) is 1.
3. The quantities of each chemical element on the left-hand side and the right-hand side are equal.

Conversely, if a chemical equation satisfies the above three conditions, we regard it as a *correct* equation, no matter whether the reaction or its constituent molecules can be chemically realized in the real world, and no matter whether it can be called a reaction (e.g., $\mathrm{H}_2 \rightarrow \mathrm{H}_2$ is considered correct). A chemical equation satisfying Conditions 1 and 3 (but not necessarily Condition 2) is called a *balanced* equation.

Your goal is to read in chemical equations with missing coefficients like Equation (4) and (5), line by line, and output the sequences of coefficients that make the equations *correct*.

Note that the above three conditions do not guarantee that a correct equation is uniquely determined. For example, if we "mix" the reactions generating H_2O and NH_3, we would get

$$xH_2 + yO_2 + zN_2 + uH_2 \rightarrow vH_2O + wNH_3 \tag{8}$$

but $(x, y, z, u, v, w) = (2, 1, 1, 3, 2, 2)$ does not represent a unique correct equation; for instance, $(4, 2, 1, 3, 4, 2)$ and $(4, 2, 3, 9, 4, 6)$ are also "correct" according to the above definition! However, we guarantee that every chemical equation we give you will lead to a *unique* correct equation by adjusting their coefficients. In other words, we guarantee that (i) every chemical equation can be balanced with positive coefficients, and that (ii) *all* balanced equations of the original equation can be obtained by multiplying the coefficients of a unique correct equation by a positive integer.

Input

The input is a sequence of chemical equations (*without* coefficients) of the following syntax in the Backus-Naur Form:

```
 <chemical_equation> ::= <molecule_sequence> "->" <molecule_sequence>
 <molecule_sequence> ::= <molecule> | <molecule> "+" <molecule_sequence>
          <molecule> ::= <group> | <group> <molecule>
             <group> ::= <unit_group> | <unit_group> <number>
        <unit_group> ::= <chemical_element> | "(" <molecule> ")"
  <chemical_element> ::= <uppercase_letter>
                       | <uppercase_letter> <lowercase_letter>
  <uppercase_letter> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I"
                       | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R"
                       | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
  <lowercase_letter> ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i"
                       | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r"
                       | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
            <number> ::= <non_zero_digit>
                       | <non_zero_digit> <digit>
    <non_zero_digit> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
             <digit> ::= "0" | <non_zero_digit>
```

Each chemical equation is followed by a period and a newline. No other characters such as spaces do not appear in the input. For instance, the equation

$$Ca(OH)_2 + CO_2 \rightarrow CaCO_3 + H_2O$$

is represented as

```
Ca(OH)2+CO2->CaCO3+H2O.
```

Each chemical equation is no more than 80 characters long, and as the above syntax implies, the `<number>`'s are less than 100. Parentheses may be used but will not be nested (maybe a good news to some of you!). Each side of a chemical equation consists of no more than 10 top-level molecules. The coefficients that make the equations *correct* will not exceed 40000. The chemical equations in the input have been chosen so that 32-bit integer arithmetic would suffice with appropriate precautions against possible arithmetic overflow. You are free to use 64-bit arithmetic, however.

The end of the input is indicated by a line consisting of a single period.

Note that our definition of `<chemical_element>` above allows chemical elements that do not exist or unknown as of now, and excludes known chemical elements with three-letter names (e.g., ununbium (Uub), with the atomic number 112).

Output

For each chemical equation, output a line containing the sequence of positive integer coefficients that make the chemical equation *correct*. Numbers in a line must be separated by a single space. No extra characters should appear in the output.

Sample Input

```
N2+H2->NH3.
Na+Cl2->NaCl.
Ca(OH)2+CO2->CaCO3+H2O.
CaCl2+AgNO3->Ca(NO3)2+AgCl.
C2H5OH+O2->CO2+H2O.
C4H10+O2->CO2+H2O.
A12B23+C34D45+ABCD->A6D7+B8C9.
A98B+B98C+C98->A98B99C99.
A2+B3+C5+D7+E11+F13->ABCDEF.
.
```

Output for the Sample Input

```
1 3 2
2 1 2
1 1 1 1
1 2 1 2
1 3 2 3
2 13 8 10
2 123 33042 5511 4136
1 1 1 1
15015 10010 6006 4290 2730 2310 30030
```

Problem G
Malfatti Circles
Input: G.in

The configuration of three circles packed inside a triangle such that each circle is tangent to the other two circles and to two of the edges of the triangle has been studied by many mathematicians for more than two centuries. Existence and uniqueness of such circles for an arbitrary triangle are easy to prove. Many methods of numerical calculation or geometric construction of such circles from an arbitrarily given triangle have been discovered. Today, such circles are called the *Malfatti circles*.

Figure 7 illustrates an example. The Malfatti circles of the triangle with the vertices (20, 80), (−40, −20) and (120, −20) are approximately

- the circle with the center (24.281677, 45.219486) and the radius 21.565935,
- the circle with the center (3.110950, 4.409005) and the radius 24.409005, and
- the circle with the center (54.556724, 7.107493) and the radius 27.107493.

Figure 8 illustrates another example. The Malfatti circles of the triangle with the vertices (20, −20), (120, −20) and (−40, 80) are approximately

- the circle with the center (25.629089, −10.057956) and the radius 9.942044,
- the circle with the center (53.225883, −0.849435) and the radius 19.150565, and
- the circle with the center (19.701191, 19.203466) and the radius 19.913790.

Your mission is to write a program to calculate the radii of the Malfatti circles of the given triangles.

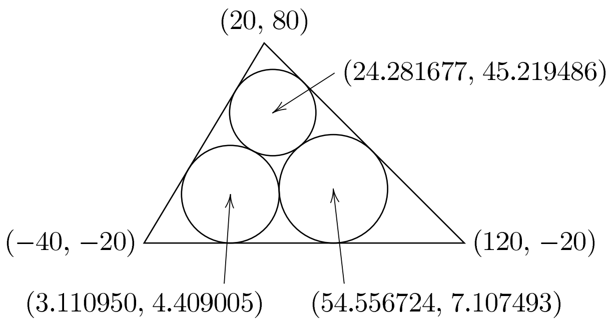


Figure 7: Example of the Malfatti circles #1.

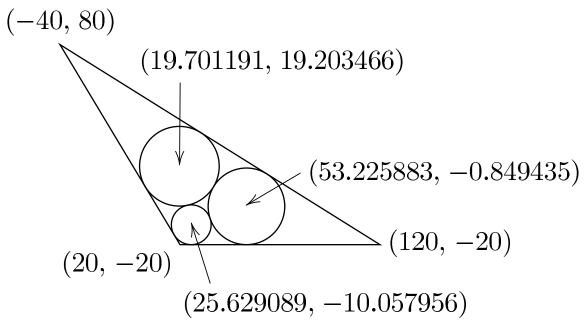


Figure 8: Example of the Malfatti circles #2.

Input

The input is a sequence of datasets. A dataset is a line containing six integers x_1, y_1, x_2, y_2, x_3 and y_3 in this order, separated by a space. The coordinates of the vertices of the given triangle are (x_1, y_1), (x_2, y_2) and (x_3, y_3), respectively. You can assume that the vertices form a triangle counterclockwise. You can also assume that the following two conditions hold.

- All of the coordinate values are greater than -1000 and less than 1000.
- None of the Malfatti circles of the triangle has a radius less than 0.1.

The end of the input is indicated by a line containing six zeros separated by a space.

Output

For each input dataset, three decimal fractions r_1, r_2 and r_3 should be printed in a line in this order separated by a space. The radii of the Malfatti circles nearest to the vertices with the coordinates (x_1, y_1), (x_2, y_2) and (x_3, y_3) should be r_1, r_2 and r_3, respectively.

None of the output values may have an error greater than 0.0001. No extra character should appear in the output.

Sample Input

```
20 80 -40 -20 120 -20
20 -20 120 -20 -40 80
0 0 1 0 0 1
0 0 999 1 -999 1
897 -916 847 -972 890 -925
999 999 -999 -998 -998 -999
-999 -999 999 -999 0 731
-999 -999 999 -464 -464 999
979 -436 -955 -337 157 -439
0 0 0 0 0 0
```

Output for the Sample Input

```
21.565935 24.409005 27.107493
9.942044 19.150565 19.913790
0.148847 0.207107 0.207107
0.125125 0.499750 0.499750
0.373458 0.383897 0.100456
0.706768 0.353509 0.353509
365.638023 365.638023 365.601038
378.524085 378.605339 378.605339
21.895803 22.052921 5.895714
```

Problem H
Twenty Questions
Input: H.in

Consider a closed world and a set of features that are defined for all the objects in the world. Each feature can be answered with "yes" or "no". Using those features, we can identify any object from the rest of the objects in the world. In other words, each object can be represented as a fixed-length sequence of booleans. Any object is different from other objects by at least one feature.

You would like to identify an object from others. For this purpose, you can ask a series of questions to someone who knows what the object is. Every question you can ask is about one of the features. He/she immediately answers each question with "yes" or "no" correctly. You can choose the next question after you get the answer to the previous question.

You kindly pay the answerer 100 yen as a tip for each question. Because you don't have surplus money, it is necessary to minimize the number of questions in the worst case. You don't know what is the correct answer, but fortunately know all the objects in the world. Therefore, you can plan an optimal strategy before you start questioning.

The problem you have to solve is: given a set of boolean-encoded objects, minimize the maximum number of questions by which every object in the set is identifiable.

Input

The input is a sequence of multiple datasets. Each dataset begins with a line which consists of two integers, m and n: the number of features, and the number of objects, respectively. You can assume $0 < m \leq 11$ and $0 < n \leq 128$. It is followed by n lines, each of which corresponds to an object. Each line includes a binary string of length m which represent the value ("yes" or "no") of features. There are no two identical objects.

The end of the input is indicated by a line containing two zeros. There are at most 100 datasets.

Output

For each dataset, minimize the maximum number of questions by which every object is identifiable and output the result.

Sample Input

```
8 1
11010101
11 4
00111001100
01001101011
01010000011
01100110001
11 16
01000101111
01011000000
01011111001
01101101001
01110010111
01110100111
10000001010
10010001000
10010110100
10100010100
10101010110
10110100010
11001010011
11011001001
11111000111
11111011101
11 12
10000000000
01000000000
00100000000
00010000000
00001000000
00000100000
00000010000
00000001000
00000000100
00000000010
00000000001
00000000000
9 32
001000000
000100000
000010000
000001000
000000100
000000010
000000001
```

```
000000000
011000000
010100000
010010000
010001000
010000100
010000010
010000001
010000000
101000000
100100000
100010000
100001000
100000100
100000010
100000001
100000000
111000000
110100000
110010000
110001000
110000100
110000010
110000001
110000000
0 0
```

Output for the Sample Input

```
0
2
4
11
9
```

Problem I
Hobby on Rails
Input: I.in

ICPC (International Connecting Points Company) starts to sell a new railway toy. It consists of a toy tramcar and many rail units on square frames of the same size. There are four types of rail units, namely, straight (S), curve (C), left-switch (L) and right-switch (R) as shown in Figure 9. A switch has three ends, namely, branch/merge-end (B/M-end), straight-end (S-end) and curve-end (C-end).

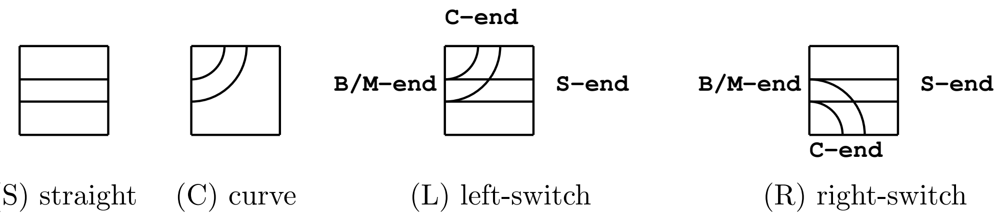


Figure 9: Four rail types

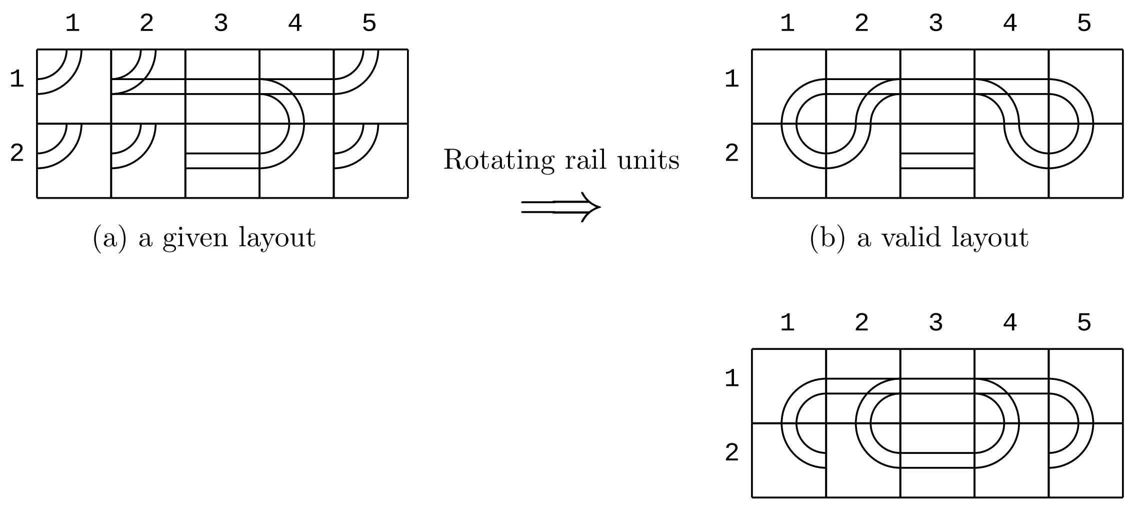
A switch is either in "through" or "branching" state. When the tramcar comes from B/M-end, and if the switch is in the through-state, the tramcar goes through to S-end and the state changes to branching; if the switch is in the branching-state, it branches toward C-end and the state changes to through. When the tramcar comes from S-end or C-end, it goes out from B/M-end regardless of the state. The state does not change in this case.

Kids are given rail units of various types that fill a rectangle area of $w \times h$, as shown in Figure 10(a). Rail units meeting at an edge of adjacent two frames are automatically connected. Each rail unit may be independently rotated around the center of its frame by multiples of 90 degrees in order to change the connection of rail units, but its position cannot be changed.

Kids should make "valid" layouts by rotating each rail unit, such as getting Figure 10(b) from Figure 10(a). A layout is valid when all rails at three ends of every switch are directly or indirectly connected to an end of another switch or itself. A layout in Figure 10(c) is invalid as well as Figure 10(a). Invalid layouts are frowned upon.

When a tramcar runs in a valid layout, it will eventually begin to repeat the same route forever. That is, we will eventually find the tramcar periodically comes to the same running condition, which is a triple of the tramcar position in the rectangle area, its direction, and the set of the states of all the switches.

A periodical route is a sequence of rail units on which the tramcar starts from a rail unit with a running condition and returns to the same rail unit with the same running condition for the



(a) a given layout (b) a valid layout (c) an invalid layout

Figure 10: Rail units in 5×2 matrix form

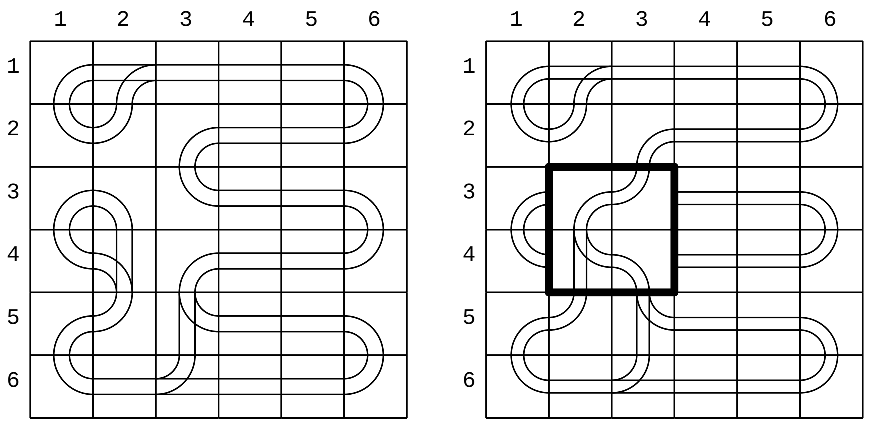
first time. A periodical route through a switch or switches is called the “fun route”, since kids like the rattling sound the tramcar makes when it passes through a switch. The tramcar takes the same unit time to go through a rail unit, not depending on the types of the unit or the tramcar directions. After the tramcar starts on a rail unit on a “fun route”, it will come back to the same unit with the same running condition, sooner or later. The fun time T of a fun route is the number of time units that the tramcar takes for going around the route.

Of course, kids better enjoy layouts with longer fun time. Given a variety of rail units placed on a rectangular area, your job is to rotate the given rail units appropriately and to find the fun route with the longest fun time in the valid layouts.

For example, there is a fun route in Figure 10(b). Its fun time is 24. Let the toy tramcar start from B/M-end at $(1, 2)$ toward $(1, 3)$ and the states of all the switches are the through-states. It goes through $(1, 3)$, $(1, 4)$, $(1, 5)$, $(2, 5)$, $(2, 4)$, $(1, 4)$, $(1, 3)$, $(1, 2)$, $(1, 1)$, $(2, 1)$, $(2, 2)$ and $(1, 2)$. Here, the tramcar goes through $(1, 2)$ with the same position and the same direction, but with the different states of the switches. Then the tramcar goes through $(1, 3)$, $(1, 4)$, $(2, 4)$, $(2, 5)$, $(1, 5)$, $(1, 4)$, $(1, 3)$, $(1, 2)$, $(2, 2)$, $(2, 1)$, $(1, 1)$ and $(1, 2)$. Here, the tramcar goes through $(1, 2)$ again, but with the same switch states as the initial ones. Counting the rail units the tramcar visited, the tramcar should have run 24 units of time after its start, and thus the fun time is 24.

There may be many valid layouts with the given rail units. For example, a valid layout containing a fun route with the fun time 120 is shown in Figure 11(a). Another valid layout containing a fun route with the fun time 148 derived from that in Figure 11(a) is shown in Figure 11(b). The four rail units whose rotations are changed from Figure 11(a) are indicated by the thick lines.

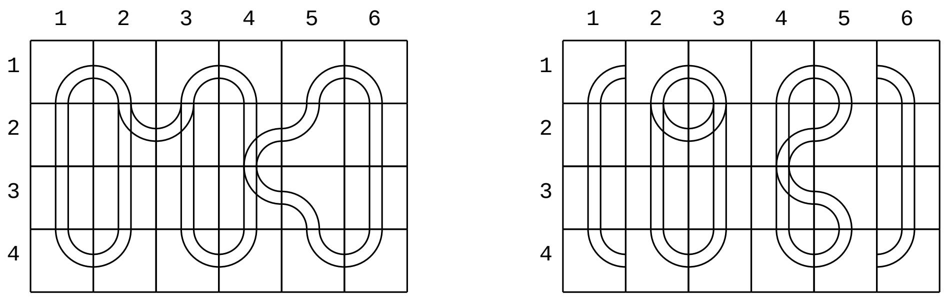
A valid layout depicted in Figure 12(a) contains two fun routes, where one consists of the rail units $(1, 1)$, $(2, 1)$, $(3, 1)$, $(4, 1)$, $(4, 2)$, $(3, 2)$, $(2, 2)$, $(1, 2)$ with $T = 8$, and the other consists of all the remaining rail units with $T = 18$.



(a) a valid layout ($T = 120$)   (b) another valid layout ($T = 148$)

Figure 11: valid layouts

Another valid layout depicted in Figure 12(b) has two fun routes whose fun times are $T = 12$ and $T = 20$. The layout in Figure 12(a) is different from that in Figure 12(b) at the eight rail units rotated by multiples of 90 degrees. There are other valid layouts with some rotations of rail units but there is no fun route with the fun time longer than 20, so that the longest fun time for this example (Figure 12) is 20.



(a) two fun routes ($T = 8$ and $T = 18$)   (b) two fun routes ($T = 12$ and $T = 20$)

Figure 12: Fun routes in valid layouts

Note that there may be simple cyclic routes that do not go through any switches in a valid layout, which are not counted as the fun routes. In Figure 13, there are two fun routes and one simple cyclic route. Their fun times are 12 and 14, respectively. The required time for going around the simple cyclic route is 20 that is greater than fun times of the fun routes. However, the longest fun time is still 14.

Input

The input consists of multiple datasets, followed by a line containing two zeros separated by a space. Each dataset has the following format.

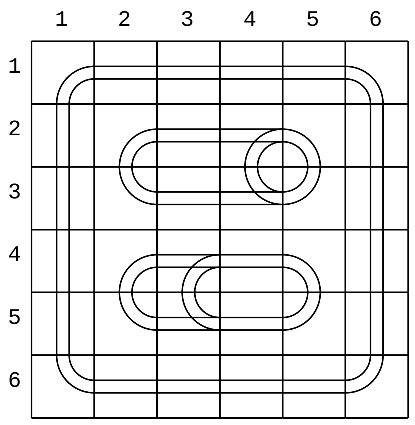


Figure 13: Two fun routes and a simple cyclic route

$$\begin{array}{lll} w & h & \\ a_{11} & \cdots & a_{1w} \\ & \cdots & \\ a_{h1} & \cdots & a_{hw} \end{array}$$

w is the number of the rail units in a row, and h is the number of those in a column. a_{ij} ($1 \le i \le h$, $1 \le j \le w$) is one of uppercase letters '`S`', '`C`', '`L`' and '`R`', which indicate the types of the rail unit at (i, j) position, i.e., straight, curve, left-switch and right-switch, respectively. Items in a line are separated by a space. You can assume that $2 \le w \le 6$, $2 \le h \le 6$ and the sum of the numbers of left-switches and right-switches is greater than or equal to 2 and less than or equal to 6.

Output

For each dataset, an integer should be printed that indicates the longest fun time of all the fun routes in all the valid layouts with the given rail units. When there is no valid layout according to the given rail units, a zero should be printed.

Sample Input

```
5 2
C L S R C
C C S C C
6 4
C C C C C C
S L R R C S
S S S L C S
C C C C C C
6 6
C L S S S C
```

```
C C C S S C
C C C S S C
C L C S S C
C C L S S C
C S L S S C
6 6
C S S S S C
S C S L C S
S C S R C S
S C L S C S
S C R S C S
C S S S S C
4 4
S C C S
S C L S
S L C S
C C C C
6 4
C R S S L C
C R L R L C
C S C C S C
C S S S S C
0 0
```

Output for the Sample Input

```
24
20
148
14
0
178
```

Problem J
Infected Land
Input: J.in

The earth is under an attack of a deadly virus. Luckily, prompt actions of the Ministry of Health against this emergency successfully confined the spread of the infection within a square grid of areas. Recently, public health specialists found an interesting pattern with regard to the transition of infected areas. At each step in time, every area in the grid changes its infection state according to infection states of its directly (horizontally, vertically, and diagonally) adjacent areas.

- An infected area continues to be infected if it has two or three adjacent infected areas.
- An uninfected area becomes infected if it has exactly three adjacent infected areas.
- An area becomes free of the virus, otherwise.

Your mission is to fight against the virus and disinfect all the areas. The Ministry of Health lets an anti-virus vehicle prototype under your command. The functionality of the vehicle is summarized as follows.

- At the beginning of each time step, you move the vehicle to one of the eight adjacent areas. The vehicle is not allowed to move to an infected area (to protect its operators from the virus). It is not allowed to stay in the same area.
- Following vehicle motion, all the areas, except for the area where the vehicle is in, change their infection states according to the transition rules described above.

  Special functionality of the vehicle protects its area from virus infection even if the area is adjacent to exactly three infected areas. Unfortunately, this virus-protection capability of the vehicle does not last. Once the vehicle leaves the area, depending on the infection states of the adjacent areas, the area can be infected.

  The area where the vehicle is in, which is *uninfected*, has the same effect to its adjacent areas as an *infected* area as far as the transition rules are concerned.

The following series of figures illustrate a sample scenario that successfully achieves the goal.

Initially, your vehicle denoted by `@` is found at $(1, 5)$ in a 5×5-grid of areas, and you see some infected areas which are denoted by `#`'s.

	1-begin	1-end	2-begin	2-end	3-begin	3-end

`....@`
`##...`
`#....`
`...#.`
`##.##`
	`.....`
	`##.@.`
	`#....`
	`...#.`
	`##.##`
		`.....`
		`##.@.`
		`###..`
		`#####`
		`..###`
			`.....`
			`##@..`
			`###..`
			`#####`
			`..###`
				`.#...`
				`#.@..`
				`.....`
				`#...#`
				`....#`
					`.#...`
					`#..@.`
					`.....`
					`#...#`
					`....#`
						`.....`
						`...@.`
						`.....`
						`.....`
						`.....`

Figure 14: A Successful Disinfection Sequence

Firstly, at the beginning of time step 1, you move your vehicle diagonally to the southwest, that is, to the area $(2, 4)$. Note that this vehicle motion was possible because this area was not infected at the start of time step 1.

Following this vehicle motion, infection state of each area changes according to the transition rules. The column "1-end" of the figure illustrates the result of such changes at the end of time step 1. Note that the area $(3, 3)$ becomes infected because there were two adjacent infected areas and the vehicle was also in an adjacent area, three areas in total.

In time step 2, you move your vehicle to the west and position it at $(2, 3)$.

Then infection states of other areas change. Note that even if your vehicle had exactly three infected adjacent areas (west, southwest, and south), the area that is being visited by the vehicle is *not* infected. The result of such changes at the end of time step 2 is as depicted in "2-end".

Finally, in time step 3, you move your vehicle to the east. After the change of the infection states, you see that all the areas have become virus free! This completely disinfected situation is the goal. In the scenario we have seen, you have successfully disinfected all the areas in three time steps by commanding the vehicle to move (1) southwest, (2) west, and (3) east.

Your mission is to find the length of the shortest sequence(s) of vehicle motion commands that can successfully disinfect all the areas.

Input

The input is a sequence of datasets. The end of the input is indicated by a line containing a single zero. Each dataset is formatted as follows.

$$\begin{array}{llll} n & & & \\ a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}$$

Here, n is the size of the grid. That means that the grid is comprised of $n \times n$ areas. You may assume $1 \leq n \leq 5$. The rest of the dataset consists of n lines of n letters. Each letter a_{ij} specifies the state of the area at the beginning: '`#`' for infection, '`.`' for free of virus, and '`@`' for the initial location of the vehicle. The only character that can appear in a line is '`#`', '`.`', or '`@`'. Among $n \times n$ areas, there exists exactly one area which has '`@`'.

Output

For each dataset, output the minimum number of time steps that is required to disinfect all the areas. If there exists no motion command sequence that leads to complete disinfection, output -1. The output should not contain any other extra character.

Sample Input

```
3
...
.@.
...
3
.##
.#.
@##
3
##.
#..
@..
5
....@
##...
#....
...#.
##.##
5
#...#
...#.
#....
...##
..@..
5
#....
.....
.....
.....
..@..
5
#..#.
#.#.#
.#.#.
....#
.#@##
```

```
5
..##.
..#..
#....
#....
.#@..
0
```

Output for the Sample Input

```
0
10
-1
3
2
1
6
4
```