Problem A
Rational Irrationals
Input: rational.txt

Rational numbers are numbers represented by ratios of two integers. For a prime number p, one of the elementary theorems in the number theory is that there is no rational number equal to $\sqrt{p}$. Such numbers are called irrational numbers. It is also known that there are rational numbers arbitrarily close to $\sqrt{p}$.

Now, given a positive integer n, we define a set Q_n of all rational numbers whose elements are represented by ratios of two positive integers both of which are less than or equal to n. For example, Q_4 is a set of 11 rational numbers $\{1/1, 1/2, 1/3, 1/4, 2/1, 2/3, 3/1, 3/2, 3/4, 4/1, 4/3\}$. 2/2, 2/4, 3/3, 4/2 and 4/4 are not included here because they are equal to 1/1, 1/2, 1/1, 2/1 and 1/1, respectively.

Your job is to write a program that reads two integers p and n and reports two rational numbers x/y and u/v, where $u/v < \sqrt{p} < x/y$ and there are no other elements of Q_n between u/v and x/y. When n is greater than $\sqrt{p}$, such a pair of rational numbers always exists.

Input

The input consists of lines each of which contains two positive integers, a prime number p and an integer n in the following format.

p n

They are separated by a space character. You can assume that p and n are less than 10000, and that n is greater than $\sqrt{p}$. The end of the input is indicated by a line consisting of two zeros.

Output

For each input line, your program should output a line consisting of the two rational numbers x/y and u/v ($x/y > u/v$) separated by a space character in the following format.

x/y u/v

They should be irreducible. For example, 6/14 and 15/3 are not accepted. They should be reduced to 3/7 and 5/1, respectively.

Sample Input

```
2 5
3 10
5 100
0 0
```

Output for the Sample Input

```
3/2 4/3
7/4 5/3
85/38 38/17
```

Problem B
Square Coins
Input: coins.txt

People in Silverland use square coins. Not only they have square shapes but also their values are square numbers. Coins with values of all square numbers up to 289 (= 17^2), i.e., 1-credit coins, 4-credit coins, 9-credit coins, ..., and 289-credit coins, are available in Silverland.

There are four combinations of coins to pay ten credits:

> ten 1-credit coins,
> one 4-credit coin and six 1-credit coins,
> two 4-credit coins and two 1-credit coins, and
> one 9-credit coin and one 1-credit coin.

Your mission is to count the number of ways to pay a given amount using coins of Silverland.

Input

The input consists of lines each containing an integer meaning an amount to be paid, followed by a line containing a zero. You may assume that all the amounts are positive and less than 300.

Output

For each of the given amount, one line containing a single integer representing the number of combinations of coins should be output. No other characters should appear in the output.

Sample Input

```
2
10
30
0
```

Output for the Sample Input

```
1
4
27
```

Problem C
Die Game
Input: die.txt

Life is not easy. Sometimes it is beyond your control. Now, as contestants of ACM ICPC, you might be just tasting the bitter of life. But don't worry! Do not look only on the dark side of life, but look also on the bright side. Life may be an enjoyable game of chance, like throwing dice. Do or die! Then, at last, you might be able to find the route to victory.

This problem comes from a game using a die. By the way, do you know a die? It has nothing to do with "death." A die is a cubic object with six faces, each of which represents a different number from one to six and is marked with the corresponding number of spots. Since it is usually used in pair, "a die" is a rarely used word. You might have heard a famous phrase "the die is cast," though.

When a game starts, a die stands still on a flat table. During the game, the die is tumbled in all directions by the dealer. You will win the game if you can predict the number seen on the top face at the time when the die stops tumbling.

Now you are requested to write a program that simulates the rolling of a die. For simplicity, we assume that the die neither slips nor jumps but just rolls on the table in four directions, that is, north, east, south, and west. At the beginning of every game, the dealer puts the die at the center of the table and adjusts its direction so that the numbers one, two, and three are seen on the top, north, and west faces, respectively. For the other three faces, we do not explicitly specify anything but tell you the golden rule: the sum of the numbers on any pair of opposite faces is always seven.

Your program should accept a sequence of commands, each of which is either "north", "east", "south", or "west". A "north" command tumbles the die down to north, that is, the top face becomes the new north, the north becomes the new bottom, and so on. More precisely, the die is rotated around its north bottom edge to the north direction and the rotation angle is 90 degrees. Other commands also tumble the die accordingly to their own directions. Your program should calculate the number finally shown on the top after performing the commands in the sequence. Note that the table is sufficiently large and the die never falls off during the game.

Input

The input consists of one or more command sequences, each of which corresponds to a single game. The first line of a command sequence contains a positive integer, representing the number of the following command lines in the sequence. You may assume that this number is less than or equal to 1024. A line containing a zero indicates the end of the input. Each command line includes a command that is one of `north`, `east`, `south`, and `west`. You may assume that no white space occurs in any lines.

Output

For each command sequence, output one line containing solely the number on the top face at the time when the game is finished.

Sample Input

```
1
north
3
north
east
south
0
```

Output for the Sample Input

```
5
1
```

Problem D
Trapezoids
Input: trap.txt

If you are a computer user, you should have seen pictures drawn with ASCII characters. Such a picture may not look as good as GIF or Postscript pictures, but is much easier to handle. ASCII pictures can easily be drawn using text editors, and can convey graphical information using only text-based media. Programs extracting information from such pictures may be useful.

We are interested in simple pictures of trapezoids, consisting only of asterisk ('*') characters and blank spaces. A trapezoid (trapezium in the Queen's English) is a four-sided polygon where at least one pair of its sides is parallel. Furthermore, the picture in this problem satisfies the following conditions.

1. All the asterisks in the picture belong to sides of some trapezoid.
2. Two sides of a trapezoid are horizontal and the other two are vertical or incline 45 degrees.
3. Every side is more than 2 characters long.
4. Two distinct trapezoids do not share any asterisk characters.
5. Sides of two trapezoids do not touch. That is, asterisks of one trapezoid do not appear in eight neighbors of asterisks of a different trapezoid. For example, the following arrangements never appear.

```
   ****
   *  *
   ****
****
*  *
****
```

```
****
*   * ***
******* *
      ***
```

```
******
*   *
****
    ****
    *  *
    ****
```

Some trapezoids may appear inside others. For example, the following is a valid picture.

```
*********
*       *
* ***   *
* *  *  *
* ***** *
*       *
*********
```

Your task is to recognize trapezoids in the picture and to calculate the area of each trapezoid. The area of a trapezoid is the number of characters on or inside its four sides, including the areas of the trapezoids inside it, if any.

Input

The input contains several descriptions of pictures. Each of them starts with a line containing an integer h ($1 \leq h \leq 1000$), where h is the height (number of lines) of the picture. Each line of the picture consists only of asterisk and space characters, and contains less than 80 characters. The lines of the picture do not necessarily have the same length and may contain redundant space characters at the end. After the last picture, an integer zero terminates the input.

Output

For each picture, your program should produce output lines each containing two integers m and n in this order, which means there are n trapezoids of area m in the picture. Output lines for one picture should be in ascending order on m and count all the trapezoids in the picture.

Output lines for two pictures should be separated by a line containing ten hyphen ('`-`') characters. This separator line should not appear before the output for the first picture nor after the output for the last.

Sample Input

```
7
********
*      *
* ***  *
* * *  *
* ***  *
*      *
********
9

***
*  *
*****     *****
          *   *
    ***   *****
   * *
  * *
 ***
11
    ****                                      *******************
   *   *  *********                          *                  *
  ******  *       *    ****                 *  *********        *
          * ***   *   *  *                 *  *         *       *
***       * *  * *  ****  *******        *  *  *** ***  *      *
* *       * ***** *         *   *        *  *   * *  * *  *     *
***       *       *          ***        *  *    ***   ***  *    *
          *********                    *  *                 *   *
                                      *  *********************  *
                                     *                          *
                                    *****************************
0
```

(Spacing between lines in pictures is made narrower for better appearance. Note that a blank line exists as the first line of the second picture.)

Output for the Sample Input

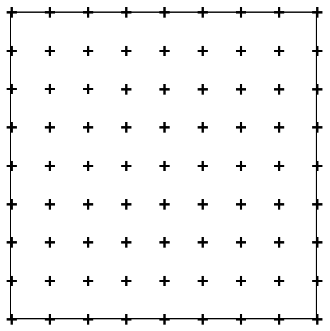
```
9 1
56 1
----------
12 2
15 1
----------
9 3
12 2
15 2
63 1
105 1
264 1
```

Problem E
Mirror Illusion
Input: mirror.txt

A rich man has a square room with mirrors for security or just for fun. Each side of the room is eight meters wide. The floor, the ceiling and the walls are not special; however, the room can be equipped with a lot of mirrors on the walls or as vertical partitions.

Every mirror is one meter wide, as tall as the wall, double-sided, perfectly reflective, and ultimately thin.

Poles to fix mirrors are located at the corners of the room, on the walls and inside the room. Their locations are the 81 lattice points at intervals of one meter. A mirror can be fixed between two poles which are one meter distant from each other. If we use the sign "+" to represent a pole, the overview of the room can be illustrated as follows.



Let us denote a location on the floor by (x, y) in a rectangular coordinate system. For example, the rectangular coordinates of the four corners of the room are (0,0), (8,0), (0,8) and (8,8), respectively. The location (x, y) is in the room if and only if the conditions $0 \leq x \leq 8$ and $0 \leq y \leq 8$ are satisfied. If i and j are integers satisfying the conditions $0 \leq i \leq 8$ and $0 \leq j \leq 8$, we can denote by (i, j) the locations of poles.

One day a thief intruded into this room possibly by breaking the ceiling. He stood at (0.75, 0.25) and looked almost toward the center of the room. Precisely speaking, he looked toward the point (1, 0.5) of the same height as his eyes. So what did he see in the center of his sight? He would have seen one of the walls or himself as follows.

- If there existed no mirror, he saw the wall at (8, 7.5).
- If there existed one mirror between two poles at (8, 7) and (8, 8), he saw the wall at (7.5, 8). (Let us denote the line between these two poles by (8, 7)–(8, 8).)

- If there were four mirrors on (8, 7)–(8, 8), (7, 8)–(8, 8), (0, 0)–(0, 1) and (0, 0)–(1, 0), he saw himself at (0.75, 0.25).
- If there were four mirrors on (2, 1)–(2, 2), (1, 2)–(2, 2), (0, 0)–(0, 1) and (0, 0)–(1, 0), he saw himself at (0.75, 0.25).

Your job is to write a program that reports the location at which the thief saw one of the walls or himself with the given mirror arrangements.

Input

The input contains multiple data sets, each representing how to equip the room with mirrors. A data set is given in the following format.

n
d_1 i_1 j_1
d_2 i_2 j_2
. . .
d_n i_n j_n

The first integer n is the number of mirrors, such that $0 \leq n \leq 144$. The way the k-th ($1 \leq k \leq n$) mirror is fixed is given by d_k and (i_k, j_k). d_k is either '`x`' or '`y`', which gives the direction of the mirror. If d_k is '`x`', the mirror is fixed on (i_k, j_k)–$(i_k + 1, j_k)$. If d_k is '`y`', the mirror is fixed on (i_k, j_k)–$(i_k, j_k + 1)$. The end of the input is indicated by a negative integer.

Output

For each data set, your program should output the location (x, y) at which the thief saw one of the walls or himself. The location should be reported in a line containing two integers which are separated by a single space and respectively represent x and y in centimeter as the unit of length. No extra lines nor spaces are allowed.

Sample Input

```
0
1
y 8 7
4
y 8 7
x 7 8
y 0 0
x 0 0
4
y 2 1
```

```
x 1 2
y 0 0
x 0 0
-1
```

Output for the Sample Input

```
800 750
750 800
75 25
75 25
```

Problem F
Heavenly Jewels
Input: jewels.txt

There is a flat island whose shape is a perfect square. On this island, there are three habitants whose names are IC, PC, and ACM. Every day, one jewel is dropped from the heaven. Just as the jewel touches the ground, IC, PC, and ACM leave their houses simultaneously, run with the same speed, and then a person who first touched the jewel can get the jewel. Hence, the person whose house is nearest to the location of the jewel is the winner of that day.

They always have a quarrel with each other, and therefore their houses are located at distinct places. The locations of their houses are fixed. This jewel falls at a random point on the island, that is, all points on the island have even chance.

When there are two or more persons whose houses are simultaneously nearest, the last person in the order of

IC, PC, ACM

obtains the jewel.

Our interest is to know the probability for IC to get the jewel under this situation.

Input

The input describes one problem instance per line. Each line contains the x- and y-coordinates of IC's home, those of PC's, and those of ACM's in this order. Note that the houses of IC, PC and ACM are located at distinct places. The end of the input is indicated by a line with six zeros.

The coordinates of the whole island are given by $(0, 10000) \times (0, 10000)$ and coordinates of houses are given in integer numbers between 1 and 9999, inclusive. It should be noted that the locations of the jewels are arbitrary places on the island and their coordinate values are not necessarily integers.

Output

For each input line, your program should output its sequence number starting from 1, and the probability for the instance. The computed probability values should have errors less than 10^{-5}.

The sequence number and the probability should be printed on the same line. The two numbers should be separated by a space.

Sample Input

```
2000  2000  8000  8000  9000  9500
2500  2500  7500  2500  2500  7500
0 0 0 0 0 0
```

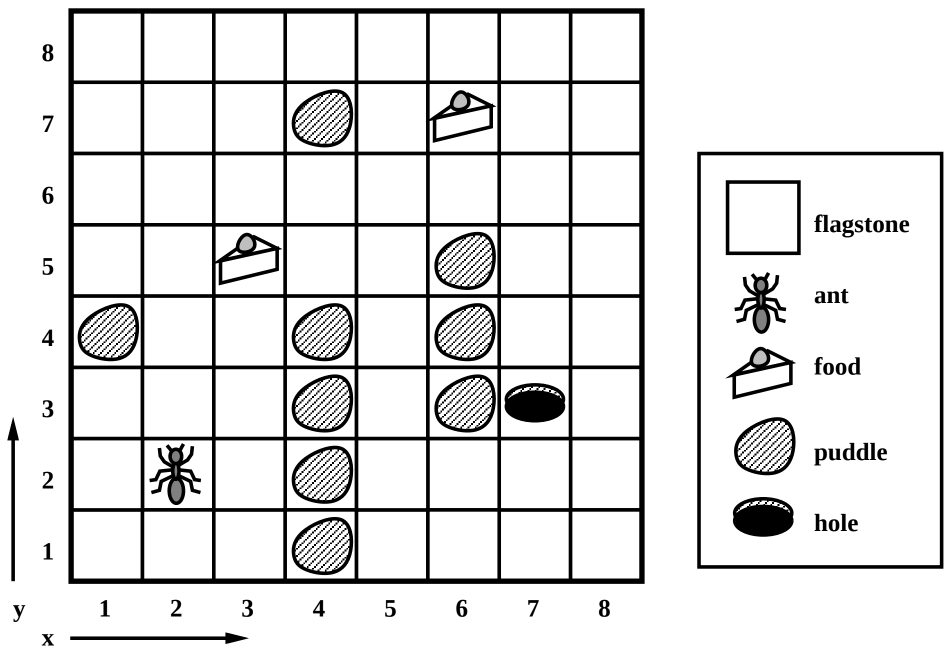
Output for the Sample Input

```
1 0.50000
2 0.25000
```

Problem G
Walking Ant
Input: ant.txt

Ants are quite diligent. They sometimes build their nests beneath flagstones.

Here, an ant is walking in a rectangular area tiled with square flagstones, seeking the only hole leading to her nest.



The ant takes exactly one second to move from one flagstone to another. That is, if the ant is on the flagstone with coordinates (x,y) at time t, she will be on one of the five flagstones with the following coordinates at time $t+1$:

(x, y), ($x+1$, y), ($x-1$, y), (x, $y+1$), (x, $y-1$).

The ant cannot go out of the rectangular area. The ant can visit the same flagstone more than once.

Insects are easy to starve. The ant has to go back to her nest without starving. Physical strength of the ant is expressed by the unit "HP". Initially, the ant has the strength of 6 HP. Every second, she loses 1 HP. When the ant arrives at a flagstone with some food on it, she eats a small piece of the food there, and recovers her strength to the maximum value, i.e., 6 HP, without taking any time. The food is plenty enough, and she can eat it as many times as she wants.

When the ant's strength gets down to 0 HP, she dies and will not move anymore. If the ant's strength gets down to 0 HP at the moment she moves to a flagstone, she does not effectively reach the flagstone: even if some food is on it, she cannot eat it; even if the hole is on that stone, she has to die at the entrance of her home.

If there is a puddle on a flagstone, the ant cannot move there.

Your job is to write a program which computes the minimum possible time for the ant to reach the hole with positive strength from her start position, if ever possible.

Input

The input consists of multiple maps, each representing the size and the arrangement of the rectangular area. A map is given in the following format.

$$\begin{array}{ccccc} w & h & & & \\ d_{11} & d_{12} & d_{13} & \cdots & d_{1w} \\ d_{21} & d_{22} & d_{23} & \cdots & d_{2w} \\ & & \cdots & & \\ d_{h1} & d_{h2} & d_{h3} & \cdots & d_{hw} \end{array}$$

The integers w and h are the numbers of flagstones in the x- and y-directions, respectively. w and h are less than or equal to 8. The integer d_{yx} represents the state of the flagstone with coordinates (x, y) as follows.

0: There is a puddle on the flagstone, and the ant cannot move there.

1, 2: Nothing exists on the flagstone, and the ant can move there. '2' indicates where the ant initially stands.

3: The hole to the nest is on the flagstone.

4: Some food is on the flagstone.

There is one and only one flagstone with a hole. Not more than five flagstones have food on them.

The end of the input is indicated by a line with two zeros.

Integer numbers in an input line are separated by at least one space character.

Output

For each map in the input, your program should output one line containing one integer representing the minimum time. If the ant cannot return to her nest, your program should output -1 instead of the minimum time.

Sample Input

```
3 3
2 1 1
1 1 0
1 1 3
8 4
2 1 1 0 1 1 1 0
1 0 4 1 1 0 4 1
1 0 0 0 0 0 0 1
1 1 1 4 1 1 1 3
8 5
1 2 1 1 1 1 1 4
1 0 0 0 1 0 0 1
1 4 1 0 1 1 0 1
1 0 0 0 0 3 0 1
1 1 4 1 1 1 1 1
8 7
1 2 1 1 1 1 1 1
1 1 1 1 1 1 1 4
1 1 1 1 1 1 1 1
1 1 1 1 4 1 1 1
4 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 3
8 8
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 4 4 1 1 1 1 1
1 4 4 2 1 1 0 0
1 1 0 0 0 0 0 3
1 1 0 4 1 1 1 1
1 1 1 1 1 1 1 1
8 8
1 1 1 1 1 1 1 1
1 1 2 1 1 1 1 1
1 1 4 4 4 1 1 1
1 1 1 4 4 1 0 1
1 1 1 1 1 1 0 1
1 1 1 1 1 1 0 3
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
0 0
```

Output for the Sample Input

```
4
-1
13
20
-1
-1
```

Problem H
Co-occurrence Search
Input: cooc.txt

A huge amount of information is being heaped on WWW. Albeit it is not well-organized, users can browse WWW as an unbounded source of up-to-date information, instead of consulting established but a little out-of-date encyclopedia. However, you can further exploit WWW by learning more about keyword search algorithms.

For example, if you want to get information on recent comparison between Windows and UNIX, you may expect to get relevant description out of a big bunch of Web texts, by extracting texts that contain both keywords "Windows" and "UNIX" close together.

Here we have a simplified version of this co-occurrence keyword search problem, where the text and keywords are replaced by a string and key characters, respectively. A character string S of length n $(1 \leq n \leq 1,000,000)$ and a set $\mathcal{K}$ of k distinct key characters $a_1, \ldots, a_k$ $(1 \leq k \leq 50)$ are given. Find every shortest substring of S that contains all of the key characters $a_1, \ldots, a_k$.

Input

The input is a text file which contains only printable characters (ASCII codes 21 to 7E in hexadecimal) and newlines. No white space such as space or tab appears in the input.

The text is a sequence of the shortest string search problems described above. Each problem consists of character string S_i and key character set $\mathcal{K}_i$ ($i = 1, 2, ..., p$). Every S_i and $\mathcal{K}_i$ is followed by an empty line. However, any single newline between successive lines in a string should be ignored; that is, newlines are not part of the string. For various technical reasons, every line consists of at most 72 characters. Each key character set is given in a single line. The input is terminated by consecutive empty lines; p is not given explicitly.

Output

All of p problems should be solved and their answers should be output in order. However, it is not requested to print all of the shortest substrings if more than one substring is found in a problem, since found substrings may be too much to check them all. Only the number of the substrings together with their representative is requested instead. That is, for each problem i, the number of the shortest substrings should be output followed by the first (or the leftmost) shortest substring s_{i1}, obeying the following format:

the number of the shortest substrings for the i-th problem
empty line
the first line of s_{i1}
the second line of s_{i1}
...
the last line of s_{i1}
empty line for the substring termination

where each line of the shortest substring s_{i1} except for the last line should consist of exactly 72 characters and the last line (or the single line if the substring is shorter than or equal to 72 characters, of course) should not exceed 72 characters.

If there is no such substring for a problem, the output will be a `0` followed by an empty line; no more successive empty line should be output because there is no substring to be terminated.

Sample Input

```
Thefirstexampleistrivial.

mfv

AhugeamountofinformationisbeingheapedonWWW.Albeititisnot
well-organized,userscanbrowseWWWasanunboundedsourceof
up-to-dateinformation,insteadofconsultingestablishedbutalittle
out-of-dateencyclopedia.However,youcanfurtherexploitWWWby
learningmoreaboutkeywordsearchalgorithms.Forexample,ifyou
wanttogetinformationonrecentcomparisonbetweenWindowsandUNIX,
youmayexpecttogetrelevantdescriptionoutofabigbunchofWeb
texts,byextractingtextsthatcontainbothkeywords"Windows"and"UNIX"
closetogether.

bWn

3.14159265358979323846264338327950288419716939937510582097494459230781640

pi

Wagner,Bach,Beethoven,Chopin,Brahms,Hindemith,Ives,Suk,Mozart,Stravinsky

Weary

ASCIIcharacterssuchas+,*,[,#,<,},_arenotexcludedinagivenstringas
thisexampleillustratesbyitself.Youshouldnotforgetthem.Onemorefact
youshouldnoticeisthatuppercaselettersandlowercaselettersare
distinguishedinthisproblem.Don'tidentify"g"and"G",forexmaple.
However,weareafraidthatthisexamplegivesyoutoomuchhint!

![GsC_l

ETAONRISHDLFCMUGYPWBVKXJQZ

ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Output for the Sample Input

```
1

firstexampleistriv

7

nWWW.Alb

0

1

Wagner,Bach,Beethoven,Chopin,Brahms,Hindemith,Ives,Suk,Mozart,Stravinsky

1

CIIcharacterssuchas+,*,[,#,<,},_arenotexcludedinagivenstringasthisexampl
eillustratesbyitself.Youshouldnotforgetthem.Onemorefactyoushouldnoticeis
thatuppercaselettersandlowercaselettersaredistinguishedinthisproblem.Don
'tidentify"g"and"G",forexmaple.However,weareafraidthatthisexamplegivesyo
utoomuchhint!

1

ETAONRISHDLFCMUGYPWBVKXJQZ
```