Problem A

ICPC Score Totalizer Software

The International Clown and Pierrot Competition (ICPC), is one of the most distinguished and also the most popular events on earth in the show business.

One of the unique features of this contest is the great number of judges that sometimes counts up to one hundred. The number of judges may differ from one contestant to another, because judges with any relationship whatsoever with a specific contestant are temporarily excluded for scoring his/her performance.

Basically, scores given to a contestant's performance by the judges are averaged to decide his/her score. To avoid letting judges with eccentric viewpoints too much influence the score, the highest and the lowest scores are set aside in this calculation. If the same highest score is marked by two or more judges, only one of them is ignored. The same is with the lowest score. The average, which may contain fractions, are truncated down to obtain final score as an integer.

You are asked to write a program that computes the scores of performances, given the scores of all the judges, to speed up the event to be suited for a TV program.

Input

The input consists of a number of datasets, each corresponding to a contestant's performance. There are no more than 20 datasets in the input.

A dataset begins with a line with an integer n, the number of judges participated in scoring the performance ($3 \leq n \leq 100$). Each of the n lines following it has an integral score s ($0 \leq s \leq 1000$) marked by a judge. No other characters except for digits to express these numbers are in the input. Judges' names are kept secret.

The end of the input is indicated by a line with a single zero in it.

Output

For each dataset, a line containing a single decimal integer indicating the score for the corresponding performance should be output. No other characters should be on the output line.

Sample Input

```
3
1000
```

```
342
0
5
2
2
9
11
932
5
300
1000
0
200
400
8
353
242
402
274
283
132
402
523
0
```

Output for the Sample Input

```
342
7
300
326
```

Problem B

Analyzing Login/Logout Records

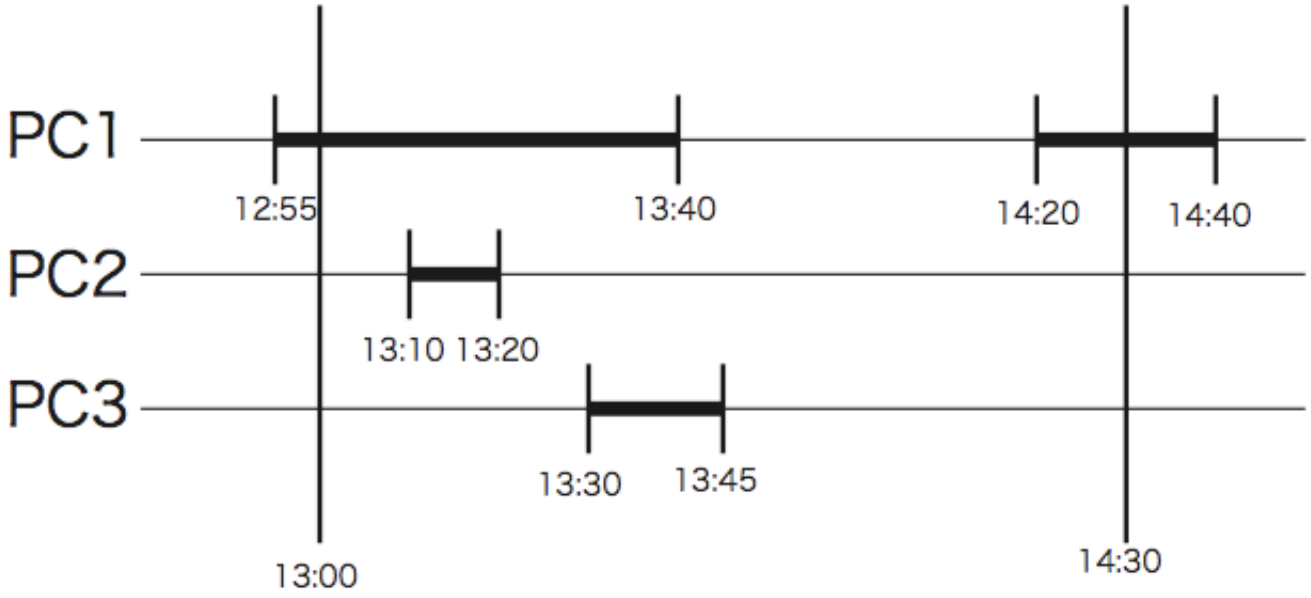
You have a computer literacy course in your university. In the computer system, the login/logout records of all PCs in a day are stored in a file. Although students may use two or more PCs at a time, no one can log in to a PC which has been logged in by someone who has not logged out of that PC yet.

You are asked to write a program that calculates the total time of a student that he/she used at least one PC in a given time period (probably in a laboratory class) based on the records in the file.

The following are example login/logout records.

- The student 1 logged in to the PC 1 at 12:55
- The student 2 logged in to the PC 4 at 13:00
- The student 1 logged in to the PC 2 at 13:10
- The student 1 logged out of the PC 2 at 13:20
- The student 1 logged in to the PC 3 at 13:30
- The student 1 logged out of the PC 1 at 13:40
- The student 1 logged out of the PC 3 at 13:45
- The student 1 logged in to the PC 1 at 14:20
- The student 2 logged out of the PC 4 at 14:30
- The student 1 logged out of the PC 1 at 14:40

For a query such as "Give usage of the student 1 between 13:00 and 14:30", your program should answer "55 minutes", that is, the sum of 45 minutes from 13:00 to 13:45 and 10 minutes from 14:20 to 14:30, as depicted in the following figure.



Input

The input is a sequence of a number of datasets. The end of the input is indicated by a line containing two zeros separated by a space. The number of datasets never exceeds 10.

Each dataset is formatted as follows.

$N\ M$
r
record_1
...
record_r
q
query_1
...
query_q

The numbers N and M in the first line are the numbers of PCs and the students, respectively. r is the number of records. q is the number of queries. These four are integers satisfying the following.

$$1 \leq N \leq 1000, 1 \leq M \leq 10000, 2 \leq r \leq 1000, 1 \leq q \leq 50$$

Each record consists of four integers, delimited by a space, as follows.

$t\ n\ m\ s$

s is 0 or 1. If s is 1, this line means that the student m logged in to the PC n at time t. If s is 0, it means that the student m logged out of the PC n at time t. The time is expressed as elapsed minutes from 0:00 of the day. t, n and m satisfy the following.

$$540 \leq t \leq 1260, 1 \leq n \leq N, 1 \leq m \leq M$$

You may assume the following about the records.

- Records are stored in ascending order of time t.
- No two records for the same PC has the same time t.
- No PCs are being logged in before the time of the first record nor after that of the last record in the file.
- Login and logout records for one PC appear alternatingly, and each of the login-logout record pairs is for the same student.

Each query consists of three integers delimited by a space, as follows.

$t_s\ t_e\ m$

It represents "Usage of the student m between t_s and t_e". t_s, t_e and m satisfy the following.

$$540 \leq t_s < t_e \leq 1260, 1 \leq m \leq M$$

Output

For each query, print a line having a decimal integer indicating the time of usage in minutes. Output lines should not have any character other than this number.

Sample Input

```
4 2
10
775 1 1 1
780 4 2 1
790 2 1 1
800 2 1 0
810 3 1 1
820 1 1 0
825 3 1 0
860 1 1 1
870 4 2 0
880 1 1 0
1
780 870 1
13 15
12
540 12 13 1
600 12 13 0
650 13 15 1
660 12 15 1
665 11 13 1
670 13 15 0
675 11 13 0
680 12 15 0
1000 11 14 1
1060 12 14 1
1060 11 14 0
1080 12 14 0
3
540 700 13
600 1000 15
1000 1200 11
1 1
2
600 1 1 1
700 1 1 0
5
540 600 1
550 650 1
610 620 1
650 750 1
700 800 1
0 0
```

Output for the Sample Input

```
55
70
30
0
0
50
10
50
0
```

Problem C

Cut the Cake

Today is the birthday of Mr. Bon Vivant, who is known as one of the greatest *pâtissiers* in the world. Those who are invited to his birthday party are *gourmets* from around the world. They are eager to see and eat his extremely creative cakes. Now a large box-shaped cake is being carried into the party. It is not beautifully decorated and looks rather simple, but it must be delicious beyond anyone's imagination. Let us cut it into pieces with a knife and serve them to the guests attending the party.

The cake looks rectangular, viewing from above (Figure C-1). As exemplified in Figure C-2, the cake will iteratively be cut into pieces, where on each cut exactly a single piece is cut into two smaller pieces. Each cut surface must be orthogonal to the bottom face and must be orthogonal or parallel to a side face. So, every piece shall be rectangular looking from above and every side face vertical.

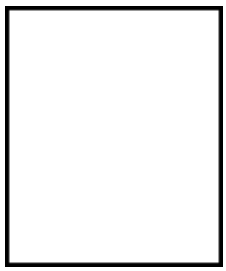


Figure C-1: The top view of the cake

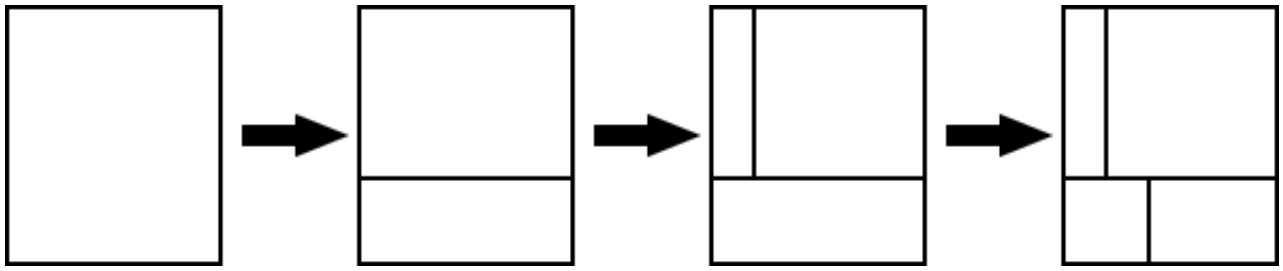


Figure C-2: Cutting the cake into pieces

Piece sizes in Figure C-2 vary significantly and it may look unfair, but you don't have to worry. Those guests who would like to eat as many sorts of cakes as possible often prefer smaller pieces. Of course, some prefer larger ones.

Your mission of this problem is to write a computer program that simulates the cutting process of the cake and reports the size of each piece.

Input

The input is a sequence of datasets, each of which is of the following format.

n w d
p_1 s_1
...
p_n s_n

The first line starts with an integer n that is between 0 and 100 inclusive. It is the number of cuts to be performed. The following w and d in the same line are integers between 1 and 100 inclusive. They denote the width and depth of the cake, respectively. Assume in the sequel that the cake is placed so that w and d are the lengths in the east-west and north-south directions, respectively.

Each of the following n lines specifies a single cut, cutting one and only one piece into two. p_i is an integer between 1 and i inclusive and is the identification number of the piece that is the target of the i-th cut. Note that, just before the i-th cut, there exist exactly i pieces. Each piece in this stage has a unique identification number that is one of $1, 2, ..., i$ and is defined as follows:

- The earlier a piece was born, the smaller its identification number is.
- Of the two pieces born at a time by the same cut, the piece with the smaller area (looking from above) has the smaller identification number. If their areas are the same, you may define as you like the order between them, since your choice in this case has no influence on the final answer.

Note that identification numbers are adjusted after each cut.

s_i is an integer between 1 and 1000 inclusive and specifies the starting point of the i-th cut. From the northwest corner of the piece whose identification number is p_i, you can reach the starting point by traveling s_i in the clockwise direction around the piece. You may assume that the starting point determined in this way cannot be any one of the four corners of the piece. The i-th cut surface is orthogonal to the side face on which the starting point exists.

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

Output

For each dataset, print in a line the areas looking from above of all the pieces that exist upon completion of the n cuts specified in the dataset. They should be in ascending order and separated by a space. When multiple pieces have the same area, print it as many times as the number of the pieces.

Sample Input

```
3 5 6
1 18
2 19
1 2
3 4 1
1 1
2 1
3 1
0 2 5
0 0 0
```

Output for the Sample Input

```
4 4 6 16
1 1 1 1
10
```

Problem D

Cliff Climbing

At 17:00, special agent Jack starts to escape from the enemy camp. There is a cliff in between the camp and the nearest safety zone. Jack has to climb the almost vertical cliff by stepping his feet on the blocks that cover the cliff. The cliff has slippery blocks where Jack has to spend time to take each step. He also has to bypass some blocks that are too loose to support his weight. Your mission is to write a program that calculates the minimum time to complete climbing.

Figure D-1 shows an example of cliff data that you will receive. The cliff is covered with square blocks. Jack starts cliff climbing from the ground under the cliff, by stepping his left or right foot on one of the blocks marked with 'S' at the bottom row. The numbers on the blocks are the "slippery levels". It takes t time units for him to safely put his foot on a block marked with t, where $1 \le t \le 9$. He cannot put his feet on blocks marked with 'X'. He completes the climbing when he puts either of his feet on one of the blocks marked with 'T' at the top row.

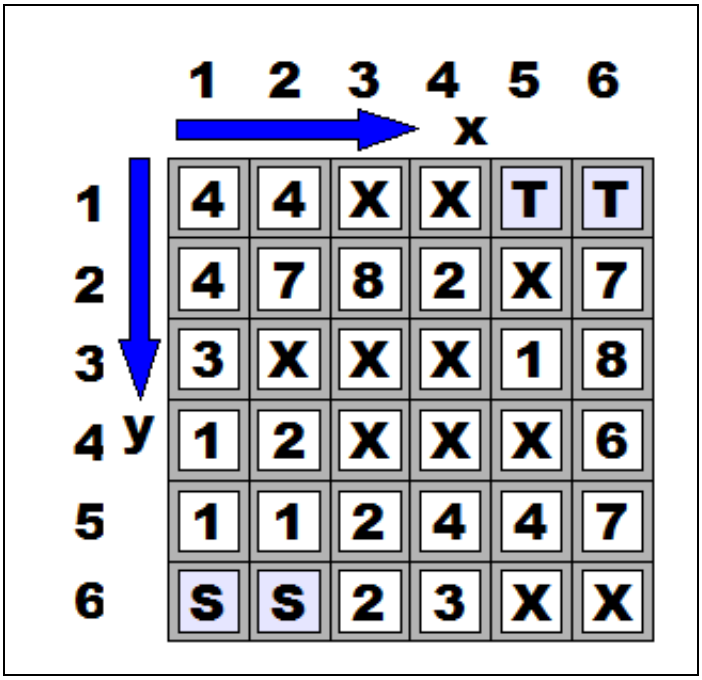


Figure D-1: Example of Cliff Data

Jack's movement must meet the following constraints. After putting his left (or right) foot on a block, he can only move his right (or left, respectively) foot. His left foot position (lx, ly) and his right foot position (rx, ry) should satisfy $lx < rx$ and $| lx - rx | + | ly - ry | \le 3$. This implies that, given a position of his left foot in Figure D-2 (a), he has to place his right foot on one of the nine blocks marked with blue color. Similarly, given a position of his right foot in Figure D-2 (b), he has to place his left foot on one of the nine blocks marked with blue color.

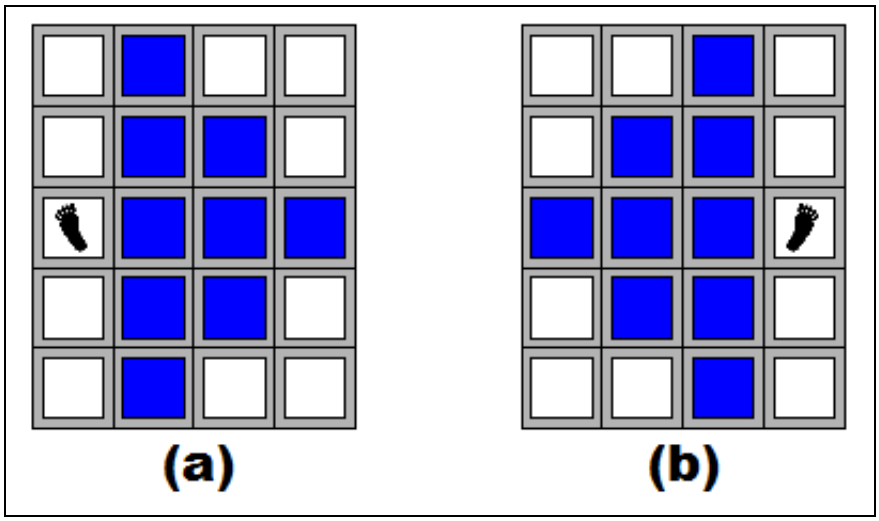


Figure D-2: Possible Placements of Feet

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:

w h
s(1,1) ... s(1,w)
s(2,1) ... s(2,w)
...
s(h,1) ... s(h,w)

The integers *w* and *h* are the width and the height of the matrix data of the cliff. You may assume $2 \leq w \leq 30$ and $5 \leq h \leq 60$. Each of the following *h* lines consists of *w* characters delimited by a space. The character *s(y, x)* represents the state of the block at position (x, y) as follows:

- 'S': Jack can start cliff climbing from this block.
- 'T': Jack finishes climbing when he reaches this block.
- 'X': Jack cannot put his feet on this block.
- '1' - '9' (= *t*): Jack has to spend *t* time units to put either of his feet on this block.

You can assume that it takes no time to put a foot on a block marked with 'S' or 'T'.

Output

For each dataset, print a line only having a decimal integer indicating the minimum time required for the cliff climbing, when Jack can complete it. Otherwise, print a line only having "-1" for the dataset. Each line should not have any characters other than these numbers.

Sample Input

```
6 6
4 4 X X T T
```

```
4 7 8 2 X 7
3 X X X 1 8
1 2 X X X 6
1 1 2 4 4 7
S S 2 3 X X
2 10
T 1
1 X
1 X
1 X
1 1
1 X
1 X
1 1
1 X
S S
2 10
T X
1 X
1 X
1 X
1 1
1 X
1 X
1 1
1 X
S S
10 10
T T T T T T T T T T
X 2 X X X X X 3 4 X
9 8 9 X X X 2 9 X 9
7 7 X 7 3 X X 8 9 X
8 9 9 9 6 3 X 5 X 5
8 9 9 9 6 X X 5 X 5
8 6 5 4 6 8 X 5 X 5
8 9 3 9 6 8 X 5 X 5
8 3 9 9 6 X X X 5 X
S S S S S S S S S S
10 7
2 3 2 3 2 3 2 3 T T
1 2 3 2 3 2 3 2 3 2
3 2 3 2 3 2 3 2 3 4
3 2 3 2 3 2 3 2 3 5
3 2 3 1 3 2 3 2 3 5
2 2 3 2 4 2 3 2 3 5
S S 2 3 2 1 2 3 2 3
0 0
```

Output for the Sample Input

```
12
5
-1
22
12
```

Problem E

Twirl Around

Let's think about a bar rotating clockwise as if it were a twirling baton moving on a planar surface surrounded by a polygonal wall (see Figure 1).

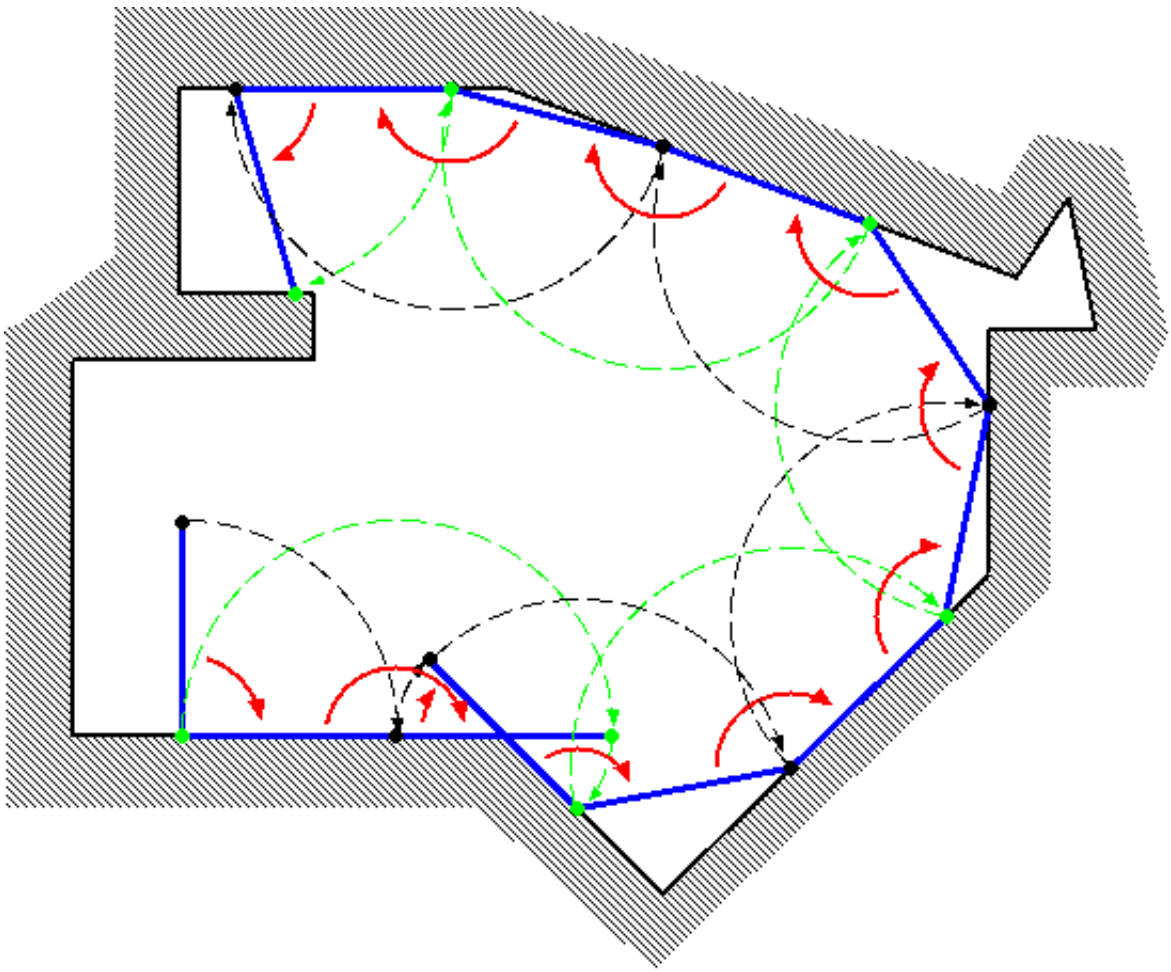


Figure 1. A bar rotating in a polygon

Initially, an end of the bar (called "end A") is at (0,0), and the other end (called "end B") is at (0,L) where L is the length of the bar. Initially, the bar is touching the wall only at the end A.

The bar turns fixing a touching point as the center. The center changes as a new point touches the wall.

Your task is to calculate the coordinates of the end A when the bar has fully turned by the given count R.

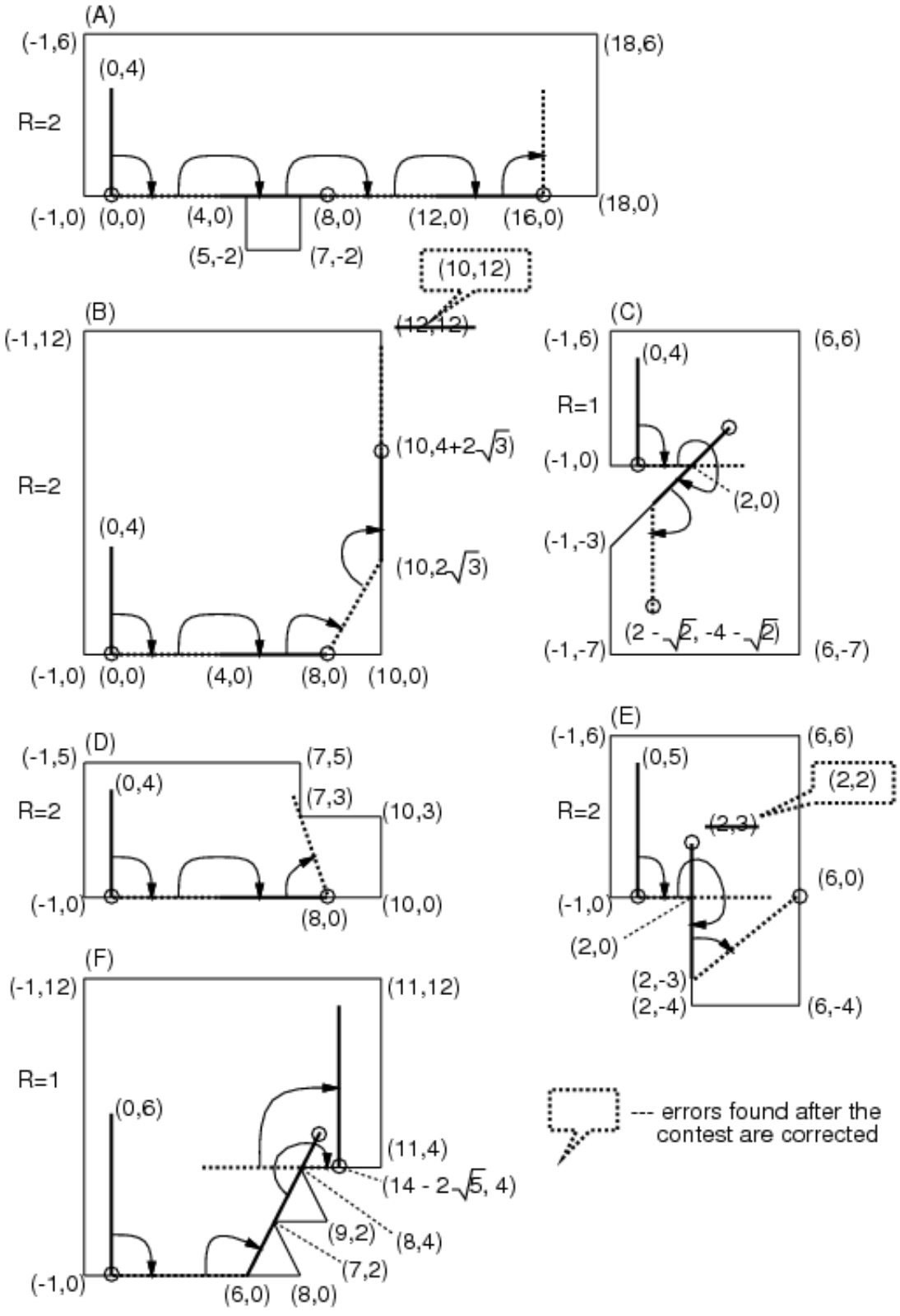


Figure 2. Examples of turning bars

In Figure 2, some examples are shown. In cases (D) and (E), the bar is stuck prematurely (cannot rotate clockwise anymore with any point touching the wall as the center) before *R* rotations. In such cases, you

should answer the coordinates of the end A in that (stuck) position.

You can assume the following:

> When the bar's length L changes by ε ($|\varepsilon| < 0.00001$), the final (x,y) coordinates will not change more than 0.0005.

Input

The input consists of multiple datasets. The number of datasets is no more than 100. The end of the input is represented by "0 0 0".

The format of each dataset is as follows:

L R N
X_1 Y_1
X_2 Y_2
...
X_N Y_N

L is the length of the bar. The bar rotates $2\pi \times R$ radians (if it is not stuck prematurely). N is the number of vertices which make the polygon.

The vertices of the polygon are arranged in a counter-clockwise order. You may assume that the polygon is *simple*, that is, its border never crosses or touches itself.

N, X_i and Y_i are integer numbers; R and L are decimal fractions. Ranges of those values are as follows:

$1.0 \le L \le 500.0$,
$1.0 \le R \le 10.0$,
$3 \le N \le 100$,
$-1000 \le X_i \le 1000$,
$-1000 \le Y_i \le 1000$,

$X_1 \le -1$, $Y_1 = 0$,
$X_2 \ge 1$, $Y_2 = 0$.

Output

For each dataset, print one line containing x- and y-coordinates of the final position of the end A, separated by a space. The value may contain an error less than or equal to 0.001. You may print any number of digits after the decimal point.

Sample Input

```
4.0 2.0 8
-1 0
5 0
5 -2
```

```
7 -2
7 0
18 0
18 6
-1 6
4.0 2.0 4
-1 0
10 0
10 12
-1 12
4.0 1.0 7
-1 0
2 0
-1 -3
-1 -8
6 -8
6 6
-1 6
4.0 2.0 6
-1 0
10 0
10 3
7 3
7 5
-1 5
5.0 2.0 6
-1 0
2 0
2 -4
6 -4
6 6
-1 6
6.0 1.0 8
-1 0
8 0
7 2
9 2
8 4
11 4
11 12
-1 12
0 0 0
```

Output for the Sample Input

```
16.0 0.0
9.999999999999998 7.4641016151377535
0.585786437626906 -5.414213562373095
8.0 0.0
6.0 0.0
9.52786404500042 4.0
```

Note that the above sample input corresponds to the cases in Figure 2. ~~For convenience, in Figure 3, we will show an animation and corresponding photographic playback for the case (C).~~

Problem F

Dr. Podboq or: How We Became Asymmetric

(An error found after the contest has been corrected.)

After long studying how embryos of organisms become asymmetric during their development, Dr. Podboq, a famous biologist, has reached his new hypothesis. Dr. Podboq is now preparing a poster for the coming academic conference, which shows a tree representing the development process of an embryo through repeated cell divisions starting from one cell. Your job is to write a program that transforms given trees into forms satisfying some conditions so that it is easier for the audience to get the idea.

A tree representing the process of cell divisions has a form described below.

- The starting cell is represented by a circle placed at the top.
- Each cell either terminates the division activity or divides into two cells. Therefore, from each circle representing a cell, there are either no branch downward, or two branches down to its two child cells.

Below is an example of such a tree.

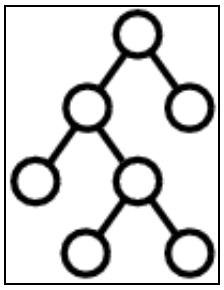


Figure F-1: A tree representing a process of cell divisions

According to Dr. Podboq's hypothesis, we can determine which cells have stronger or weaker asymmetricity by looking at the structure of this tree representation. First, his hypothesis defines "left-right similarity" of cells as follows:

1. The left-right similarity of a cell that did not divide further is 0.
2. For a cell that did divide further, we collect the partial trees starting from its child or descendant cells, and count how many kinds of structures they have. Then, the left-right similarity of the cell is defined to be the ratio of the number of structures that appear both in the right child side and the left child side. We regard two trees have the same structure if we can make them have exactly the same shape by interchanging two child cells of ~~arbitrary~~ some cells.

For example, suppose we have a tree shown below:

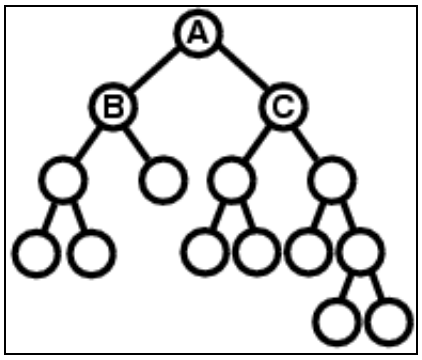


Figure F-2: An example tree

The left-right similarity of the cell A is computed as follows. First, within the descendants of the cell B, which is the left child cell of A, the following three kinds of structures appear. Notice that the rightmost structure appears three times, but when we count the number of structures, we count it only once.

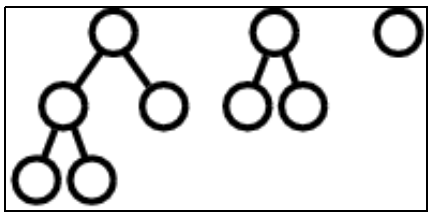


Figure F-3: Structures appearing within the descendants of the cell B

On the other hand, within the descendants of the cell C, which is the right child cell of A, the following four kinds of structures appear.

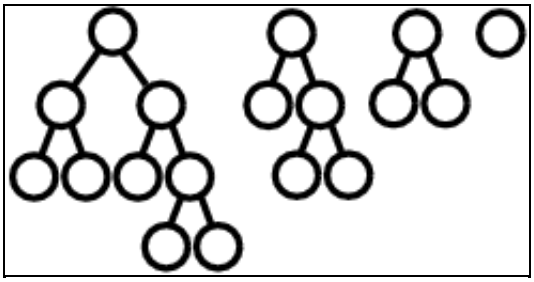


Figure F-4: Structures appearing within the descendants of the cell C

Among them, the first, second, and third ones within the B side are regarded as the same structure as the second, third, and fourth ones within the C side, respectively. Therefore, there are four structures in total, and three among them are common to the left side and the right side, which means the left-right similarity of A is 3/4.

Given the left-right similarity of each cell, Dr. Podboq's hypothesis says we can determine which of the cells *X* and *Y* has stronger asymmetricity by the following rules.

1. If *X* and *Y* have different left-right similarities, the one with lower left-right similarity has stronger asymmetricity.
2. Otherwise, if neither *X* nor *Y* has child cells, they have completely equal asymmetricity.
3. Otherwise, both *X* and *Y* must have two child cells. In this case, we compare the child cell of *X* with stronger (or equal) asymmetricity (than the other child cell of *X*) and the child cell of *Y* with stronger

(or equal) asymmetricity (than the other child cell of *Y*), and the one having a child with stronger asymmetricity has stronger asymmetricity.
4. If we still have a tie, we compare the other child cells of *X* and *Y* with weaker (or equal) asymmetricity, and the one having a child with stronger asymmetricity has stronger asymmetricity.
5. If we still have a tie again, *X* and *Y* have completely equal asymmetricity.

When we compare child cells in some rules above, we recursively apply this rule set.

Now, your job is to write a program that transforms a given tree representing a process of cell divisions, by interchanging two child cells of arbitrary cells, into a tree where the following conditions are satisfied.

1. For every cell *X* which is the starting cell of the given tree or a left child cell of some parent cell, if *X* has two child cells, the one at left has stronger (or equal) asymmetricity than the one at right.
2. For every cell *X* which is a right child cell of some parent cell, if *X* has two child cells, the one at right has stronger (or equal) asymmetricity than the one at left.

In case two child cells have equal asymmetricity, their order is arbitrary because either order would results in trees of the same shape.

For example, suppose we are given the tree in Figure F-2. First we compare B and C, and because B has lower left-right similarity, which means stronger asymmetricity, we keep B at left and C at right. Next, because B is the left child cell of A, we compare two child cells of B, and the one with stronger asymmetricity is positioned at left. On the other hand, because C is the right child cell of A, we compare two child cells of C, and the one with stronger asymmetricity is positioned at right. We examine the other cells in the same way, and the tree is finally transformed into the tree shown below.

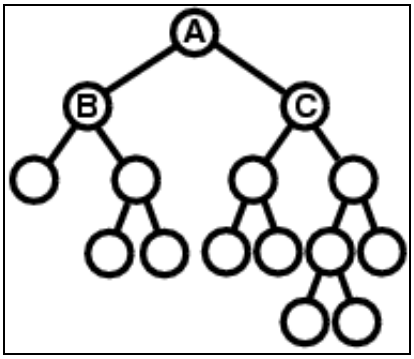


Figure F-5: The example tree after the transformation

Please be warned that the only operation allowed in the transformation of a tree is to interchange two child cells of some parent cell. For example, you are not allowed to transform the tree in Figure F-2 into the tree below.

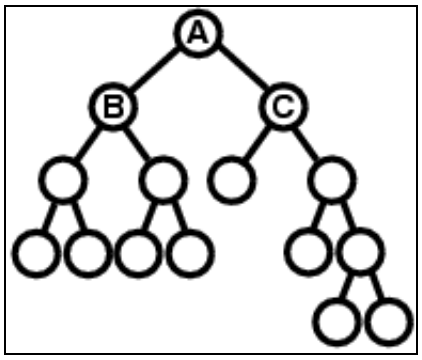


Figure F-6: An example of disallowed transformation

Input

The input consists of *n* lines (1≤*n*≤100) describing *n* trees followed by a line only containing a single zero which represents the end of the input. Each tree includes at least 1 and at most 127 cells. Below is an example of a tree description.

```
((x (x x)) x)
```

This description represents the tree shown in Figure F-1. More formally, the description of a tree is in either of the following two formats.

> "(" <description of a tree starting at the left child> <single space> <description of a tree starting at the right child> ")"

or

> "x"

The former is the description of a tree whose starting cell has two child cells, and the latter is the description of a tree whose starting cell has no child cell.

Output

For each tree given in the input, print a line describing the result of the tree transformation. In the output, trees should be described in the same formats as the input, and the tree descriptions must appear in the same order as the input. Each line should have no extra character other than one tree description.

Sample Input

```
(((x x) x) ((x x) (x (x x))))
(((x x) (x x)) ((x x) ((x x) (x x))))
(((x x) ((x x) x)) (((x (x x)) x) (x x)))
(((x x) x) ((x x) (((((x x) x) x) x) x)))
(((x x) x) ((x (x x)) (x (x x))))
((((x (x x)) x) (x ((x x) x))) ((x (x x)) (x x)))
((((x x) x) ((x x) (x (x x)))) (((x x) (x x)) ((x x) ((x x) (x x)))))
0
```

Output for the Sample Input

```
((x (x x)) ((x x) ((x x) x)))
(((x x) ((x x) (x x))) ((x x) (x x)))
(((x ((x x) x)) (x x)) ((x x) ((x x) x)))
(((x ((x ((x x) x)) x)) (x x)) ((x x) x))
((x (x x)) ((x (x x)) ((x x) x)))
(((x (x x)) (x x)) ((x ((x x) x)) ((x (x x)) x)))
(((x (x x)) ((x x) ((x x) x))) (((x x) (x x)) (((x x) (x x)) (x x))))
```