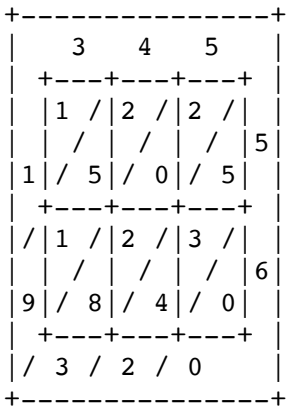
Problem A: (More) Multiplication

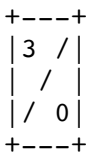
Source file: `multiply.{c, cpp, java}`
Input file: `multiply.in`

Educators are always coming up with new ways to teach math to students. In 2011, an educational software company, All Computer Math (ACM), developed an application to display products in a traditional grade school math format. ACM is now working on an updated version of the software that will display results in a lattice format that some students find to be easier when multiplying larger numbers.

An example would be when multiplying 345 * 56 = 19320 as given below, using a lattice grid with 2 rows and 3 columns, which appears inside a surrounding frame:



The first operand, 345, is displayed above the top of the grid with each digit centered horizontally above its column of the grid, and the second operand, 56, is displayed along the righthand side with each digit centered vertically at the center of its row in the grid. A single cell of the grid, such as



represents the product of the digit of the first operand that is above its column and the digit of the second operand that is to the right of its row. In our example, this cell represents the product 5 times 6 = 30 that results when multiplying the 5 in 345 and the 6 in 56. Note that the 10's digit of that product is placed in the upper left portion of this cell and the 1's digit in the lower right.

The overall product is then computed by summing along the diagonals in the lattice that represent the same place values in the result. For example, in our first problem the product 19320 was computed as:

1's digit = **0**
10's digit = 5 + 3 + 4 = 12, thus **2** with a carry of 1
100's digit = (1 carry) + 2 + 0 + 2 + 8 = 13, thus **3** with a carry of 1
1000's digit = (1 carry) + 2 + 5 + 1 = **9**
10000's digit = **1**

The resulting product is placed with the one's digit below the grid at the far right and, depending on its length, with the most significant digits wrapped around the left side of the grid. Each digit of the final product appears perfectly aligned with the corresponding diagonal summands.

To provide an aesthetic view, we use a series of minus (-) characters for horizontal lines, pipe (|) characters for vertical lines, and slash (/) characters for diagonal lines. Furthermore, we use a plus (+) character wherever a horizontal and vertical line meet. Each multiplication lattice is subsequently "boxed" by an outer border. There is a row containing the first operand which is between the topmost border and the top line of the grid, and a row between the bottom of the grid and the bottom border, which contains some portion of the resulting product. There is one column between the leading | and the left edge of the inner grid, which may contain a portion of the resulting product, and one column after the right edge of the inner grid but before the rightmost | border, which contains the second operand. If the product is not long enough to wrap around the bottom-left corner, the column between the left border and the left edge of the grid will containing only spaces. (See the later example of 3 x 3.)

Leading zeros should be displayed within lattice grid cells, but leading zeros should never be displayed in the product, nor should there ever be a slash (/) character prior to the leading digit of the product. For example, consider the product of 12 * 27 = 324 below:

```
+-----------+
|    1   2  |
| +---+---+ |
| |0 /|0 /| |
| | / | / |2|
| |/ 2|/ 4| |
| +---+---+ |
| |0 /|1 /| |
| | / | / |7|
|3|/ 7|/ 4| |
| +---+---+ |
|/ 2 / 4    |
+-----------+
```

Note that in the top-right grid of the lattice, the product 2 * 2 = 04 is displayed with the zero for the tens digit. However, there is no thousands digit displayed in the product 324, nor is there any slash displayed above the digit 3 in that product.

Input: The input contains one or more tests. Each test contains two positive integers, A and B, such that $1 \le A \le 9999$ and $1 \le B \le 9999$. The last data set will be followed by a line containing 0 0.

Output: For each data set, produce the grid that illustrates how to multiply the two numbers using the lattice multiplication technique.

Note: The tables must be formatted precisely as outlined by the rules and examples provided. Mistakes that involve solely errant whitespace will be categorized as **`Presentation Error`**; all other errors will be reported as **`Wrong Answer`**.

Example Input:

```
345 56
12 27
1 68
9999 7
3 3
0 0
```

Example Output:

```
+---------------+
|    3   4   5  |
| +---+---+---+ |
| |1 /|2 /|2 /| |
| | / | / | / |5|
|1|/ 5|/ 0|/ 5| |
| +---+---+---+ |
|/|1 /|2 /|3 /| |
| | / | / | / |6|
|9|/ 8|/ 4|/ 0| |
| +---+---+---+ |
|/ 3 / 2 / 0    |
+---------------+
+-----------+
|    1   2  |
| +---+---+ |
| |0 /|0 /| |
| | / | / |2|
| |/ 2|/ 4| |
| +---+---+ |
| |0 /|1 /| |
| | / | / |7|
|3|/ 7|/ 4| |
| +---+---+ |
|/ 2 / 4    |
+-----------+
+-------+
|    1  |
| +---+ |
| |0 /| |
| | / |6|
| |/ 6| |
| +---+ |
| |0 /| |
| | / |8|
|6|/ 8| |
| +---+ |
|/ 8    |
+-------+
+-------------------+
|    9   9   9   9  |
| +---+---+---+---+ |
| |6 /|6 /|6 /|6 /| |
| | / | / | / | / |7|
|6|/ 3|/ 3|/ 3|/ 3| |
| +---+---+---+---+ |
|/ 9 / 9 / 9 / 3    |
+-------------------+
+-------+
|    3  |
| +---+ |
| |0 /| |
| | / |3|
| |/ 9| |
| +---+ |
|  9    |
+-------+
```

[Page intentionally blank]

Problem B: Fun House

Source file: `fun.{c, cpp, java}`
Input file: `fun.in`

American Carnival Makers Inc. (ACM) has a long history of designing rides and attractions. One of their more popular attractions is a fun house that includes a room of mirrors. Their trademark is to set up the room so that when looking forward from the entry door, the exit door appears to be directly ahead. However, the room has double-sided mirrors placed throughout at 45 degree angles. So, the exit door can be on any of the walls of the room. The set designer always places the entry and mirrors, but can never seem to be bothered to place the exit door. One of your jobs as part of the construction crew is to determine the placement of the exit door for the room given an original design.

The final diagram for a sample room is given below. The asterisk (`*`) marks the entry way, lower case x's mark the walls, the mirrors are given by the forward and backward slash characters (`/ and \`), open spaces with no visual obstructions are marked by periods (`.`), and the desired placement of the exit is marked with an ampersand (`&`). In the input diagram, there is an 'x' in place of the '&', since the exit has not yet been located. You need to alter the input diagram by replacing the proper 'x' with an '&' to identify the exit. Note that entrances and exits can appear on any of the walls (although never a corner), and that it is physically impossible for the exit to be the same as the entrance. (You don't need to understand why this is so, although it may be fun to think about.)

```
xxxxxxxxxxx
x../..\...x
x..../....x
*../......x
x.........x
xxxxxx&xxxx
```

Input: Each room will be preceded by two integers, W and L, where $5 \le W \le 20$ is the width of the room including the border walls and $5 \le L \le 20$ is the length of the room including the border walls. Following the specification of W and L are L additional lines containing the room diagram, with each line having W characters from the alphabet: { `*` , `x` , `.` , `/` , `\` }. The perimeter will always be comprised of walls, except for one asterisk (`*`) which marks the entrance; the exit is not (yet) marked. A line with two zeros indicates the end of input data.

Output: For each test case, the first line will contain the word, HOUSE, followed by a space and then an integer that identifies the given fun house sequentially. Following that should be a room diagram which includes the proper placement of the exit door, as marked by an ampersand (`&`).

Note: In both Java and C++ the backslash character (`\`) has special meaning as an escape character within character and string literals. You must use the combination `\\` to express a single backslash within a character or string literal within source code.

Example Input:	Example Output:
11 6 xxxxxxxxxxx x../..\...x x..../....x *../......x x.........x xxxxxxxxxxx 5 5 xxxxx *...x x...x x...x xxxxx 5 5 xxxxx x./\x *./.x x..\x xxxxx 6 6 xxx*xx x/...x x....x x/./.x x\./.x xxxxxx 10 10 xxxxxxxxxx x.../\...x x........x x........x x.../\..\x *...\/../x x........x x........x x...\/...x xxxxxxxxxx 0 0	HOUSE 1 xxxxxxxxxxx x../..\...x x..../....x *../......x x.........x xxxxxx&xxxx HOUSE 2 xxxxx *...& x...x x...x xxxxx HOUSE 3 xxxxx x./\x *./.x x..\& xxxxx HOUSE 4 xxx*xx x/...x x....x x/./.& x\./.x xxxxxx HOUSE 5 xxxxxxxxxx x.../\...x x........x x........x &.../\..\x *...\/../x x........x x........x x...\/...x xxxxxxxxxx

ACM Mid-Central Programming Competition 2014

Problem C: Lexicography

Source file: `lex.{c, cpp, java}`
Input file: `lex.in`

An anagram of a string is any string that can be formed using the same letters as the original. (We consider the original string an anagram of itself as well.) For example, the string `ACM` has the following 6 anagrams, as given in alphabetical order:

```
ACM
AMC
CAM
CMA
MAC
MCA
```

As another example, the string `ICPC` has the following 12 anagrams (in alphabetical order):

```
CCIP
CCPI
CICP
CIPC
CPCI
CPIC
ICCP
ICPC
IPCC
PCCI
PCIC
PICC
```

Given a string and a rank K, you are to determine the K^{th} such anagram according to alphabetical order.

Input: Each test case will be designated on a single line containing the original word followed by the desired rank K. Words will use uppercase letters (i.e., `A` through `Z`) and will have length at most 16. The value of K will be in the range from 1 to the number of distinct anagrams of the given word. A line of the form "`# 0`" designates the end of the input.

Warning: The value of K could be almost 2^{45} in the largest tests, so you should use type `long` in Java, or type `long long` in C++ to store K.

Output: For each test, display the K^{th} anagram of the original string.

Example Input:	**Example Output:**

`ACM 5`
`ICPC 12`
`REGION 274`
`# 0`
	`MAC`
	`PICC`
	`IGNORE`

[Page intentionally blank]

Problem D: The Leprechaun Hunt

Source file: `hunt.{c, cpp, java}`
Input file: `hunt.in`

In Irish mythology, a Leprechaun is a small sprite who stores all his treasure in a hidden pot of gold at the end of the rainbow. If someone is able to catch the Leprechaun, he must give that person his pot of gold. In this problem, we explore the difficulty of capturing a Leprechaun.

We model a search with V villagers trying to catch a single Leprechaun as a game on a simple undirected graph having $N \geq 1+V$ nodes. To begin the game, the villagers position themselves at a subset of V distinct nodes. After that, the Leprechaun chooses a remaining node as a starting position. In each round of the game that follows, one villager moves from his or her current node to an adjacent node that is unoccupied by another villager. If that node has the Leprechaun, the villagers win the pot of gold. Otherwise, the Leprechaun now has the option of either staying at his current node, or moving to an adjacent, unoccupied node. Given a specific graph, and a fixed number of villagers, we are interested in the minimum number of turns the villagers need to capture the most clever of Leprechauns.

As examples, consider the two figures below. For the graph in Figure 1, a single villager can never capture a Leprechaun, as the Leprechaun can easily stay away from the villager. However, two villagers can capture the Leprechaun after at most 2 turns. For example, the villagers might begin at nodes `A` and `D`, in which case a clever Leprechaun will start at node `F`. But after the villager at `A` moves to `G` the villagers can capture the Leprechaun on their second turn, no matter whether the Leprechaun moves to `E` or remains at `F`.

For the graph in Figure 2, a single villager is unable to catch a clever Leprechaun. To see why this is the case, we describe a possible strategy of the Leprechaun, which is to always stay within the square made by BCDE, and opposite of the villager if the villager is in that square. If the villager were ever to go to A, the Leprechaun can remain still. In contrast, two villagers are able to capture the Leprechaun on their first move by picking initial positions such as B and E.

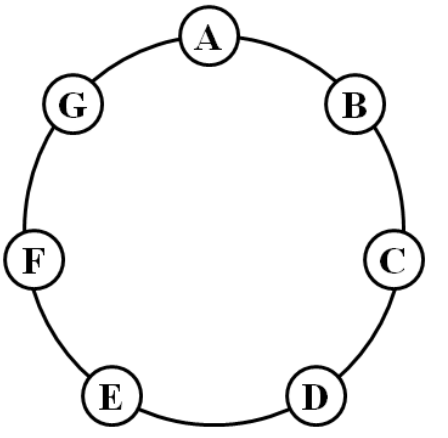


Figure 1

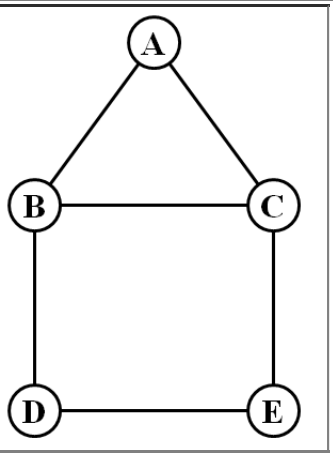


Figure 2

Input: Each tests begins with a line containing three integers: V N E. The value of V denotes the number of villagers such that $1 \le V \le 7$. The number of nodes N in the graph will satisfy $1 + V \le N \le 15$. The value $1 \le E \le 45$ designates the number of edges in the graph. Following the initial line of parameters will be one or more lines describing the edges of the graph, with up to 15 edges per line. Nodes of the graph are implicitly denoted with the first N uppercase letters (`A`, `B`, `C`, ...), and edges are explicitly denoted as two-character strings; for example the string `AC` denotes an edge connecting nodes `A` and `C` to each other. The E edges will be distinct, each edge connects two distinct nodes, and any node will have at most 6 incident edges. A line with the single value `0` designates the end of the input.

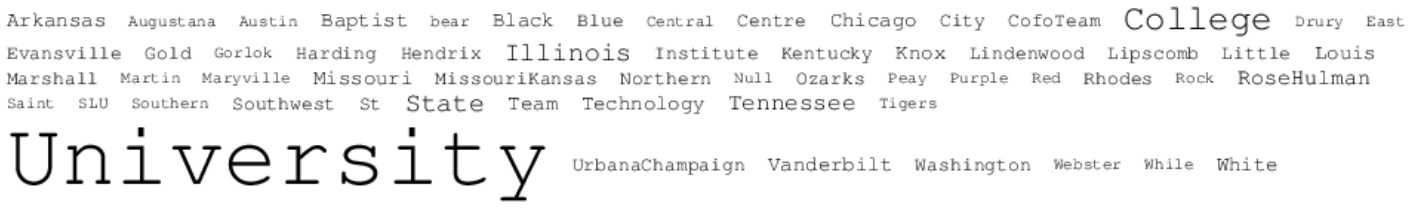
Output: For each test case, output a line, prefaced with the case number as shown in the example output below, followed by the minimum number of moves that the villagers need to guarantee capture of the Leprechaun, or the word `NEVER` if the villagers are unable to capture the Leprechaun.

Example Input:	**Example Output:**
1 7 7 AB BC CD DE EF FG GA 2 7 7 AB BC CD DE EF FG GA 1 5 6 AB AC BC BD DE EC 2 5 6 AB AC BC BD DE EC 2 10 15 AB BC CD DE EA AF BG CH DI EJ FH HJ JG GI IF 3 10 15 AB BC CD DE EA AF BG CH DI EJ FH HJ JG GI IF 3 14 10 AB BC CD EF FG GH IJ JK LM MN 4 14 10 AB BC CD EF FG GH IJ JK LM MN 0	CASE 1: NEVER CASE 2: 2 CASE 3: NEVER CASE 4: 1 CASE 5: NEVER CASE 6: 1 CASE 7: NEVER CASE 8: 2

ACM Mid-Central Programming Competition 2014

Problem E: Word Cloud

Source file: `cloud.{c, cpp, java}`
Input file: `cloud.in`



A word cloud (or tag cloud) is a visual representation of textual data based on a weighted metric. In the above cloud (which is based on this year's list of Mid-Central teams), the font size of each word is based on its number of occurrences in the data set. Tagg Johnson is a man obsessed with counting words that appear in online documents. On his computer, he keeps a spreadsheet of all the sites he visits, along with a list of words that appear on each site and the number of times such word appears. Tagg would like to generate word clouds based on the data he has collected.

Before describing the algorithm Tagg uses for generating clouds, we digress for a quick lesson in typography. The basic unit of measure is known as a *point* (typically abbreviated as *pt*). A font's size is described based on the vertical number of points from one line to the next, including any interline spacing. For example, with a 12pt font, the vertical space from the top of one character to the top of a character below it is 12 points. We assume that a character's height is precisely equal to the font's point size (regardless of whether the character is upper or lower case).

For this problem, we focus on a fixed-width font, such as Courier, in which each character of the alphabet is also given the same amount of width. The character width for such a font depends on the font size and the aspect ratio. For Courier, a word with t characters rendered in a font of size P has a total width of $\left\lceil \frac{9}{16} \cdot t \cdot P \right\rceil$ when measured in points. Note well the use of the ceiling operator, which converts any noninteger to the next highest integer. For example, a 5-letter word in a 20pt font would be rendered with a height of 20 points and a width equal to $\left\lceil \frac{900}{16} \right\rceil = \lceil 56.25 \rceil = 57$ points.

Now we can describe Tagg's algorithm for creating a word cloud. He pre-sorts his word list into alphabetical order and removes words that do not occur at least five times. For each word w, he computes a point size based on the formula $P = 8 + \left\lceil \frac{40(c_w - 4)}{(c_{max} - 4)} \right\rceil$, where c_w is the number of occurrences of the word, and c_{max} is the number of occurrences of the most frequent word in the data set. Note that by this formula, every word will be rendered with anywhere from a 9pt font to a 48pt font. He then places the words in rows, with a 10pt horizontal space between adjacent words, placing as many words as fit in the row, subject to a maximum width W for his entire cloud. The height of a given row is equal to the *maximum* font size of any word rendered in that row.

As a tangible example, consider the following data set and word cloud.

word	count
apple	10

banana	5
grape	20
kiwi	18
orange	12
strawberry	10

In this example, `apple` is rendered with 23pt font using width 65pt, `banana` is rendered with 11pt font using width 38pt, and `grape` is rendered with 48pt font and width 135pt. If the overall word cloud is constrained to have width at most 260, those three words fit in a row and the overall height of that row is 48pt (due to `grape`). On the second row `kiwi` is rendered with height 43pt and width 97pt, and `orange` is rendered with height 28pt and width 95pt. A third row has `strawberry` with height 23pt and width 130pt. The overall height of this word cloud is 114pt.

Input: Each data set begins with a line containing two integers: W and N. The value W denotes the maximum width of the cloud; $W \le 5000$ will be at least as wide as any word at its desired font size. The value $1 \le N \le 100$ denotes the number of words that appear in the cloud. Following the first line are N additional lines, each having a string S that is the word (with no whitespace), and an integer C that is a count of the number of occurrences of that word in the original data set, with $5 \le C \le 1000$. Words will be given in the same order that they are to be displayed within the cloud.

Output: For each data set, output the word `CLOUD` followed by a space, a serial number indicating the data set, a colon, another space, and the integer height of the cloud, measured in font points.

Example Input:	**Example Output:**
260 6 apple 10 banana 5 grape 20 kiwi 18 orange 12 strawberry 10 250 6 apple 10 banana 5 grape 20 kiwi 18 orange 12 strawberry 10 610 6 apple 10 banana 5 grape 20 kiwi 18 orange 12 strawberry 10 0 0	CLOUD 1: 114 CLOUD 2: 99 CLOUD 3: 48

ACM Mid-Central Programming Competition 2014

Problem F: The Maze Makers

Source file: `maze.{c, cpp, java}`
Input file: `maze.in`

The Maze Makers is a publisher of puzzle books. One of their most popular series is maze books. They have a program that generates rectangular two-dimensional mazes like the one shown in **Figure 1**. The rules for these mazes are: (1) A maze has exactly two exterior cell walls missing, opening to two distinct terminal cells, (2) starting from any one cell, all other cells are reachable, (3) between any two cells in the maze there is exactly one simple path. Formally, a path is a sequence of cells where each cell and its successor on the path share an edge without a wall. A *simple* path is a path that never repeats a cell.

The Maze Maker program uses hexadecimal digits to encode the walls and passages of a maze. For each cell in the maze there is a corresponding hex digit. As shown in **Figure 2**, the 1's and 0's in the 4 digit binary representation of a hex digit correspond to the walls (1's) and passages (0's) for each cell in the maze. For example, the binary encoding for the hex digit B is 1011. Starting at the top of the cell and moving clockwise around it, this digit represents a cell with a wall at the top, a passage to the right and walls at the bottom and to the left. A path between two maze cells successively moves one cell up, down, left or right, going through passages only.

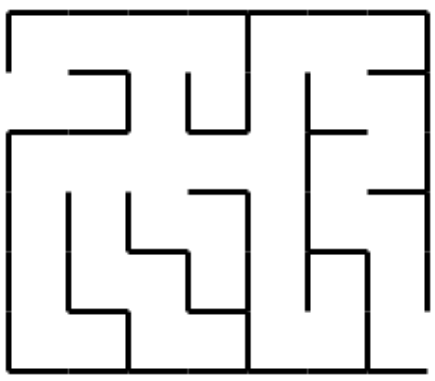


Figure 1: Sample Maze

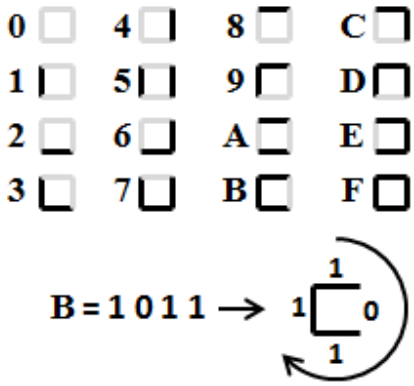


Figure 2: Hex Code for Walls and Passageways

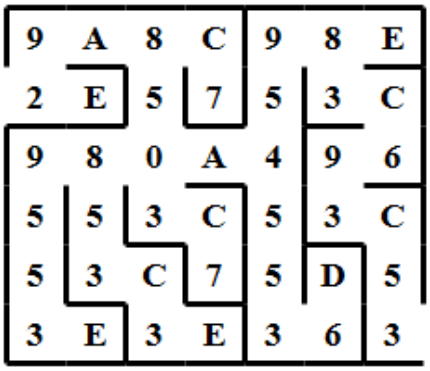


Figure 3: Maze with Cell Labels

Figure 3 shows the sample maze with the hexadecimal labels in each cell. For example, the hexadecimal digit E in the top-right cell indicates that it has a wall above it, to its right, below it, yet a passageway to its left. The hexadecimal digit 8 to its left indicates that its cell has only a wall above it. The inputs will always be self-consistent, in that the hexadecimal digits in neighboring cells will agree on whether they share a wall or passageway, and each input will always have precisely two terminal cells, each with one missing exterior wall.

Our sample maze is a legitimate maze in that all cells are reachable and there is a unique simple path between any pairs of cells in the maze. Your goal is to write a program that reads the hexadecimal descriptions of a potential maze and tests to determine if it is legitimate. If there is a problem, your program must report only the *first* problem, as detailed below in the section titled "Output".

Input: The input consists of the descriptions of one or more candidate mazes. Each maze description will start with two integers, H and W, indicating the height and width of the maze, respectively, such that $1 \le H \le 50$ and $2 \le W \le 50$. Following this first line will be H rows of hexadecimal digits, with each row consisting of W digits. The input is terminated with a line displaying a pair of zeros.

Output: For each candidate maze, the program should output the first one of the following statements that applies:

NO SOLUTION
UNREACHABLE CELL
MULTIPLE PATHS
MAZE OK

The classification statements are defined formally as follows:

NO SOLUTION - There is no path through the interior of the maze between the two exterior openings.

UNREACHABLE CELL - There is at least one cell in the maze that is not reachable by following passageways from either of the openings in the exterior walls of the maze.

MULTIPLE PATHS - There exists a pair of cells in the maze that have more than one simple path between them. Two simple paths are considered to be distinct if any part of the paths differ.

MAZE OK - None of the above problems exist.

Note well that for the second case given in the following examples, there is no path between the start and finish and there is an unreachable cell; the correct output should simply be NO SOLUTION, because that error message is listed first in the above list. Similarly, in the fourth example given, UNREACHABLE CELL is reported because that error has priority over the multiple paths.

Sample input and output is given on the following page.

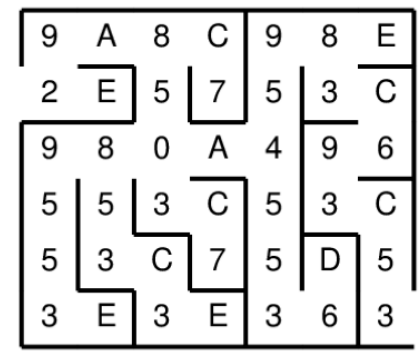
Example Input:

```
6 7
9A8C98E
2E5753C
980A496
553C53C
53C75D5
3E3E363
3 3
F9A
D3E
3AC
1 8
3AAA8AAE
6 3
9AC
3C5
A24
9A6
5BC
3C7
5 4
8A8E
592C
5186
161C
3A63
5 4
8AAE
59AC
5386
1E1C
3A63
0 0
```

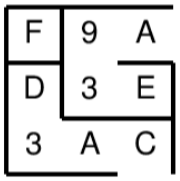
Example Output:

```
MAZE OK
NO SOLUTION
MAZE OK
UNREACHABLE CELL
MULTIPLE PATHS
MULTIPLE PATHS
```

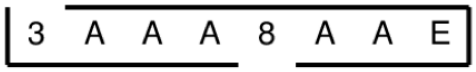
Maze Diagrams:



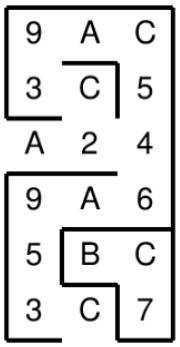
maze 1



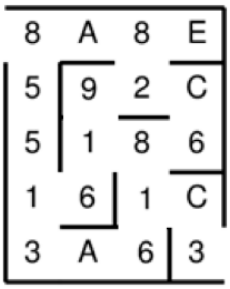
maze 2



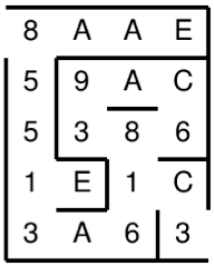
maze 3



maze 4



maze 5



maze 6

ACM Mid-Central Programming Competition 2014

[Page intentionally blank]

Problem G: Reverse Rot

Source file: `rot.{c, cpp, java}`
Input file: `rot.in`

A very simplistic scheme, which was used at one time to encode information, is to rotate the characters within an alphabet and rewrite them. ROT13 is the variant in which the characters A-Z are rotated 13 places, and it was a commonly used insecure scheme that attempted to "hide" data in many applications from the late 1990's and into the early 2000's.

It has been decided by Insecure Inc. to develop a product that "improves" upon this scheme by first reversing the entire string and then rotating it. As an example, if we apply this scheme to string `ABCD` with a reversal and rotation of 1, after the reversal we would have `DCBA` and then after rotating that by 1 position we have the result `EDCB`.

Your task is to implement this encoding scheme for strings that contain only capital letters, underscores, and periods. Rotations are to be performed using the alphabet order:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ_.
```

Note that underscore follows Z, and the period follows the underscore. Thus a forward rotation of 1 means 'A' is shifted to 'B', that is, 'A'→'B', 'B'→'C', ..., 'Z'→'_', '_'→'.', and '.'→'A'. Likewise a rotation of 3 means 'A'→'D', 'B'→'E', ..., '.'→'C'.

Input: Each input line will consist of an integer N, followed by a string. N is the amount of forward rotation, such that $1 \le N \le 27$. The string is the message to be encrypted, and will consist of 1 to 40 characters, using only capital letters, underscores, and periods. The end of the input will be denoted by a final line with only the number 0.

Output: For each test case, display the "encrypted" message that results after being reversed and then shifted.

Example Input:	**Example Output:**
1 ABCD 3 YO_THERE. 1 .DOT 14 ROAD 9 SHIFTING_AND_ROTATING_IS_NOT_ENCRYPTING 2 STRING_TO_BE_CONVERTED 1 SNQZDRQDUDQ 0	EDCB CHUHKWBR. UPEA ROAD PWRAYF_LWNHAXWH.RHPWRAJAX_HMWJHPWRAORQ. FGVTGXPQEAGDAQVAIPKTVU REVERSE_ROT

ACM Mid-Central Programming Competition 2014

[Page intentionally blank]

Problem H: Shrine Maintenance

Source file: `shrine.{c, cpp, java}`
Input file: `shrine.in`

A religious sect has holy sites with shrines placed around a circle of radius 1000. The circle is split into N equal length arcs and the endpoints are numbered in order, 1 through N. The first figure shows a circle where N is 12, with 12 gray tick marks like on a 12-hour analog clock. We can imagine the marks numbered, as on a clock, with 12 at the top. Each circle has one or more *sacred numbers* associated with it. The sacred numbers for the circle in the first figure are 2 and 3. A shrine, indicated by a black dot in the figure, is placed at each mark whose number is a multiple of at least one of the sacred numbers, so in this case the shrines are at positions 2, 3, 4, 6, 8, 9, 10, and 12.

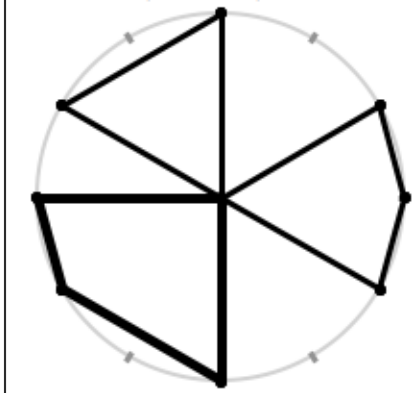


Figure 1

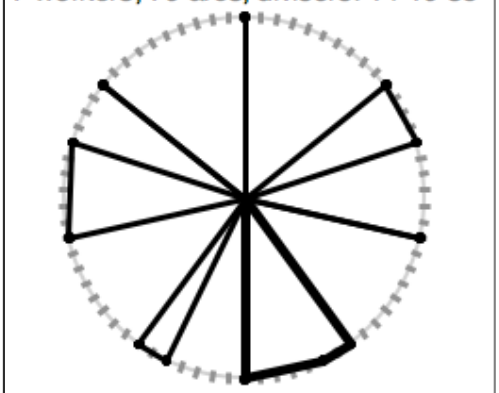


Figure 2

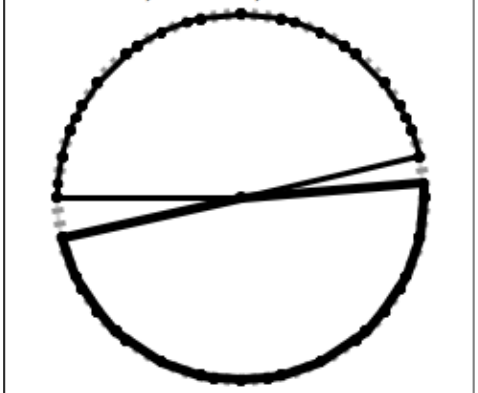


Figure 3

When it comes time to inspect and repair the shrines at a given site, the area is closed and a team of workers simultaneously fan out from a maintenance shed, located in the center of the circle, so that each shrine is visited by at least one worker. Once all workers have returned to the shed, the site is reopened to the public. Because these sites are in great demand, it is important that they be closed as briefly as possible. In order to minimize this time, they must figure out how to apportion the shrines among the current number of workers, so the maximum distance traveled by any one worker is as small as possible. Figure 1 shows one choice for the optimal solution paths for 3 workers. The lower left path has darker lines, to indicate that it is one with the longest length, which in this case is approximately 3517.6.

This sect has many circular sites with multiple shrines. The number of available workers at a site, W, the value of the number equal arcs, N, and the sacred numbers vary between sites. The sacred numbers are always divisors of N. Your job is to help figure out how much time is required for maintenance. Figures 2 and 3 show optimal solutions for other sites.

Input: The input consists of one or more data sets. Each data set is on a single line and consists entirely of positive integers. The first three entries are W, the number of workers, N, the number of equal arcs

around the circle, and D, the number of sacred divisors of N. At the end come the D divisors of N. W is no more than the total number of shrines; N ≤ 8600, and D ≤ 6; each listed divisor of N is smaller than N.

A single zero, 0, will be placed on the last line to indicate the end of the input.

Output: The output is a single line for each dataset: the maximum distance a worker must travel with an optimal assignment of the shrines. This number is displayed so that it is rounded to one decimal place, and always shows that decimal place, even if it is 0. To ensure unique answers with double arithmetic, the datasets are chosen so that if your answer is anywhere within .005 of the exact minimum distance, then the answer rounded to one decimal place will be the same.

The first three sample datasets correspond to the three figures.

Caution: Be careful with your algorithm so it finishes rapidly.

Example Input:	**Example Output:**
3 12 2 2 3 7 70 3 14 10 35 2 84 3 3 4 14 4 35 2 7 5 3 20 2 5 4 3 6 1 1 4 6 1 1 1 6 1 1 8600 8600 3 1 10 100 0	3517.6 2624.3 4987.7 3224.9 3488.4 3000.0 3000.0 7000.0 2000.0

ACM Mid-Central Programming Competition 2014

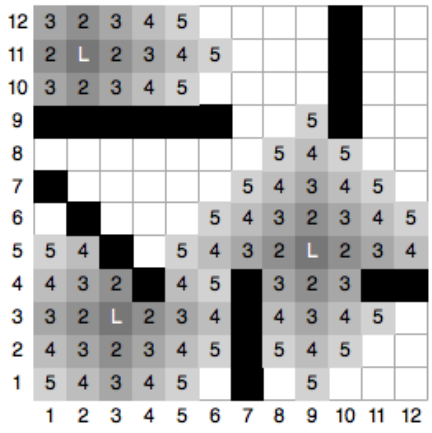
Problem I: Wet Tiles

Source file: `wet.{c, cpp, java}`
Input file: `wet.in`

Alice owns a construction company in the town of Norainia, famous for its unusually dry weather. In fact, it only rains a few days per year there. Because of this phenomenon, many residents of Norainia neglect to do roof repairs until leaks occur and ruin their floors. Every year, Alice receives a deluge of calls from residents who need the leaks fixed and floor tiles replaced. While exquisite in appearance, Norainia floor tiles are not very water resistant; once a tile becomes wet, it is ruined and must be replaced. This year, Alice plans to handle the rainy days more efficiently than in past years. She will hire extra contractors to dispatch as soon as the calls come in, so hopefully all leaks can be repaired as soon as possible. For each house call, Alice needs a program to help her determine how many replacement tiles a contractor team will need to bring to complete the job.

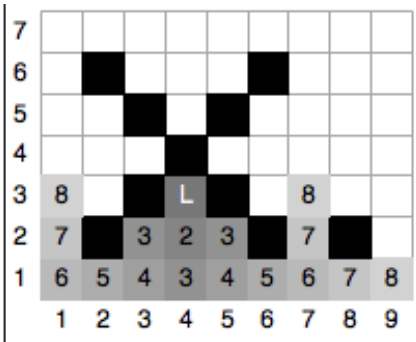
For a given house, square floor tiles are arranged in a rectangular grid. Leaks originate from one or more known source locations above specific floor tiles. After the first minute, the tiles immediately below the leaks are ruined. After the second minute, water will have spread to any tile that shares an edge with a previously wet tile. This pattern of spreading water continues for each additional minute. However, the walls of a house restrict the water; if a damaged area hits a wall, the water does not penetrate the wall. We assume there are always four outer walls surrounding the entire house. A house may also have a number of additional "inner" walls; each inner wall is comprised of a connected linear sequence of locations (which may or may not be connected to the outer walls or to each other).

As an example, Figure 1 shows water damage (in gray) that would result from three initial leaks (each marked with a white letter 'L') after each of the first five minutes of time. Tiles labeled '2' become wet during the second minute, tiles labeled '3' become wet during the third minute, and so forth. The black areas designate inner walls that restrict the flow of water. Note that after 5 minutes, a total of 75 tiles have been damaged and will need to be replaced. Figures 2 through 4 show other houses that correspond to the example inputs for this problem.



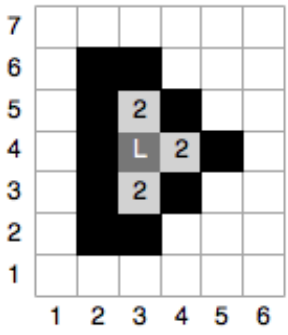
75 wet tiles

Figure 1



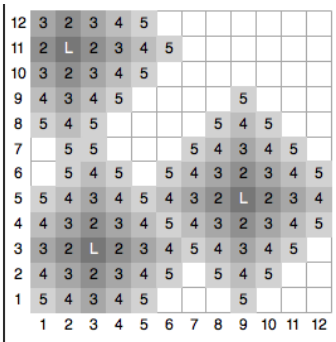
17 wet tiles

Figure 2



4 wet tiles

Figure 3



94 wet tiles

Figure 4

Input: Each house is described beginning with a line having five integral parameters: *X Y T L W*. Parameters *X* and *Y* designate the dimensions of the rectangular grid, with $1 \le X \le 1000$ and $1 \le Y \le 1000$. The coordinate system is one-indexed, as shown in the earlier figures. Parameter *T* designates the number of minutes that pass before a team of contractors arrives at a house and stops the leaks, with $1 \le T \le 200000$. The parameter *L* designates the number of leaks, with $1 \le L \le 100$. Parameter *W* designates the number of inner walls in the house, $0 \le W \le 100$.

The following *2L* integers in the data set, on one or more lines, are distinct (x y) pairs that designate the locations of the *L* distinct leaks, such that $1 \le x \le X$ and $1 \le y \le Y$.

If $W > 0$, there will be *4W* additional integers, on one or more lines, that describe the locations of the walls. For each such wall the four parameters $(x_1, y_1), (x_2, y_2)$ describe the locations of two ends of the wall. Each wall replaces a linear sequence of adjoining tiles and is either axis-aligned or intersects both axes at a 45 degree angle. Diagonal walls are modeled as a sequence of cells that would just be touching corner to corner. If the two endpoints of a wall are the same, the wall just occupies the single cell at that location. Walls may intersect with each other, but no leak is over a wall.

There will be one or more houses in the data file and a line with a single integer `-1` designates the end of the data set.

Output: For each house, display the total number of tiles that are wet after *T* minutes.

Example Input:	**Example Output:**
12 12 5 3 5 2 11 3 3 9 5 1 9 6 9 1 7 4 4 7 1 7 4 10 9 10 12 11 4 12 4 9 7 8 1 3 4 3 2 2 6 6 6 2 2 6 8 2 8 2 6 7 50 1 3 3 4 2 2 2 6 3 6 5 4 5 4 3 2 12 12 5 3 0 2 11 3 3 9 5 -1	75 17 4 94