1998 Denison Spring Programming Contest
Problem 1
Who Turned Out the Lights?

Single Row Lights Out is a puzzle that consists of a single row of n lighted buttons, numbered 1 through n. Each button can be either on (lit) or off (dark). When a button is pressed, it and the button(s) immediately adjacent to it change state — either on to off or off to on. You will be given a configuration of the n lighted buttons and try to find which buttons to press to turn all the lights out. The buttons you press are called a *solution* to the puzzle.

Note two things: pressing a button twice is like not pressing it at all and the order you press the buttons doesn't matter. Therefore, we'll only consider solutions where each button is pressed at most once. You're asked to find a solution, if one exists and list the buttons pressed in the solution in ascending order.

For example, if 1 indicates a lit button and 0 indicates a dark button, then the configuration 000110 has solution 1, 2, 4. Since after pressing button 1, the row of lights becomes 110110. Then pressing button 2, we get 001110. And finally pressing button 4 yields 000000. (Note there may be more than one possible solution; you are asked to find only one, if one exists.) On the other hand, 10000 has no solution.

You will be given many puzzle instances. Each puzzle instance is given in two lines. The first line contains n ($n \leq 50$), the number of lights in the row, and the second line contains a string of n 0's and 1's with no spaces separating the characters. (0 indicates the light is off and 1 indicates the light is on.) $n = 0$ indicates end of input.

For each puzzle instance generate a single line of output. If a solution exists, you are to print out the buttons pressed to solve the puzzle, listed in increasing order separated by a single space. If there is no solution, print the message "`NO SOLUTION.`".

WARNING: Generating all possible solutions is not a computationally viable strategy; your time limit will be exceeded. You must find a more efficient way to solve the problem.

EXAMPLE:

Input:

```
6
000110
5
10000
20
10101010101010101011
0
```

should generate output:

```
1 2 4
NO SOLUTION.
1 3 5 7 9 11 13 15 17 19
```

Of course the first and last puzzle instances might have other solutions.

1998 Denison Spring Programming Contest
Problem 2
I Think That I Shall Never See A Poem Lovely As A Tree

The *Tree of All Fractions* is a binary tree with an infinite number of nodes. The root node is labelled 1/1 and left and right children are labelled as follows: if i/j is the label of a node then its left child is labelled $i/(i+j)$ and its right child is labelled $(i+j)/j$.

So, the first couple of levels of the Tree of All Fractions looks like:

```
                1/1

       1/2               2/1

  1/3       3/2      2/3       3/1
```

Interestingly, every positive rational number will appear in this tree exactly once and in lowest terms! (Neat, huh?) Your problem is to find the path back to the root node from a given rational number, printing the labels as you go. For example, the path from 13/8 back to the root node is:

$$13/8 \rightarrow 5/8 \rightarrow 5/3 \rightarrow 2/3 \rightarrow 2/1 \rightarrow 1/1$$

Input for this problem will be a series of fractions, one per line, in the form `i j`, where `i` and `j` are relatively prime positive integers (both less than 1000) separated by a single space, representing the fraction i/j. The last line of input will be `1 1` and should generate no output. Each input line should generate one output line giving the path from i/j back to the root 1/1 with successive fractions separated by `->` (that's two characters: "minus" and "greater than") with one space one either side of these two characters. (Some output lines may exceed 80 characters; do not worry about that.) For example the above path should be printed out as:

```
13/8 -> 5/8 -> 5/3 -> 2/3 -> 2/1 -> 1/1
```

EXAMPLE:

Input:	should generate output:
`13 8`	`13/8 -> 5/8 -> 5/3 -> 2/3 -> 2/1 -> 1/1`
`6 1`	`6/1 -> 5/1 -> 4/1 -> 3/1 -> 2/1 -> 1/1`
`2 3`	`2/3 -> 2/1 -> 1/1`
`1 1`	

1998 Denison Spring Programming Contest
Problem 3
Oh What A Tangled Web

A page on the World Wide Web may contain hypertext references, or links, to other pages on the web. Each page of the Web has a unique string address (known as its URL) and each link specifies the address of the page to be visited. Web page authors are allowed to use either fully qualified addresses or a variety of shorter addresses. You are to translate some shorter addresses to fully qualified ones according to some translation rules.

Input consists of one up to ten nonempty lines containing translation rules, followed by an empty line, followed by one or more nonempty lines containing strings to be translated. End of file terminates the input. Each translation rule has the form

$$matchtype\ x\ action\ y$$

where *matchtype* is either `prefix`, `suffix` or `substring`; x and y are nonempty strings of characters not containing blanks and *action* is either `prepend`, `append` or `replace`. The four parts of each translation rule are separated by single blanks. The string x will be no longer than three characters and the string y will be no longer than 30 characters.

For each string z to be translated, successively apply translation rules *in the order given* and output the resultant string. The string z, the resultant string and all intermediate strings will be no longer than 60 characters. A `prefix` rule applies only if x is a prefix of z. A `suffix` rule applies only if x is a suffix of z. A `substring` rule applies only if x is a substring of z. If the *action* is `prepend` , concatenate y to the beginning of the string. If the *action* is `append` , concatenate y to the end of the string. If the *action* is `replace` , replace the first occurrence of x in the string with y.

EXAMPLE:

Input:

```
prefix // prepend http:
prefix / prepend http://jazz.sci.funu.edu
suffix / append index.html
substring /~ replace /user/

http://jazz.sci.funu.edu/
/~joe/
/this/one.html
//jazz.sci.funu.edu/
```

should generate output:

```
http://jazz.sci.funu.edu/index.html
http://jazz.sci.funu.edu/user/joe/index.html
http://jazz.sci.funu.edu/this/one.html
http://jazz.sci.funu.edu/index.html
```

1998 Denison Spring Programming Contest
Problem 4
Trying To Keep Too Many Things In The Air At Once

Believe it or not, there is a mathematical theory of juggling. When doing two-handed asyncronous juggling (that is, alternating throws between the right hand and the left), a pattern of length n can be notated by the n *throw heights* of each successive ball: $t_0, t_1, \ldots, t_{n-1}$, where each t_i is a positive integer. What this pattern denotes is how long each ball stays in the air and when it is thrown. Think of the indices $0, 1, \ldots$ as counts and t_i as how long the ball thrown at count i stays in the air. (Obviously, the higher the ball is thrown the longer it takes for it to come down.) What a juggler wants to avoid is two balls landing in the same hand at the same time. If that were the case, the given pattern would not be doable. It turns out that for a pattern to be doable with b balls, the mean of the throw heights must be equal to b. This condition is not sufficient, however. Surprisingly, a necessary and sufficient condition for this pattern is that $\{t_i + i(\bmod n)\} = \{0, 1, \ldots, n-1\}$. (We'll call this the *Fundamental Theorem of Juggling*, or *FTJ*. This is not an obvious result.) Note that when calculating the numbers in the first set ($\{t_i + i(\bmod n)\}$), the residues modulo n do not have to appear in order $0, 1, \ldots, n-1$.

For example, notice that $7, 4, 4, 1$ is a pattern of length 4 for 4 balls. Note that $(7+4+4+1)/4 = 4$ and $7 + 0(\bmod 4) = 3$, $4 + 1(\bmod 4) = 1$, $4 + 2(\bmod 4) = 2$ and $1 + 3(\bmod 4) = 0$. Thus the condition for FTJ is met, and so this is a legitimate juggling pattern for 4 balls of length 4.

In this problem, you will receive as input three non-negative integers b, n and m (on one line each integer separated by a space) and print out all legitimate juggling patterns of length n for b balls, where no throw height exceeds m. You will be given multiple problem instances and should halt when either b, n or m is 0. You may assume that $b \leq 6$, $n \leq 6$ and $m \leq 6$ and that $b \leq n$ and $n \leq m$. Each input line should generate one output line giving *all* the legitimate juggling patterns of length n for b balls. These patterns may be displayed in any order. The throw heights in each pattern should be separated by single spaces and each pattern (including the last one on your output line) should end with a comma and a space. Some output lines may be longer than 80 characters. (Don't worry about this.)

EXAMPLE:

Input:

```
3 3 5
2 3 4
0 1 3
```

should generate output:

```
3 3 3, 2 3 4, 3 4 2, 4 2 3, 4 1 4, 4 4 1, 1 4 4, 5 3 1, 1 5 3, 3 1 5, 5 2 2, 2 5
 2, 2 2 5,
2 2 2, 3 1 2, 1 2 3, 2 3 1, 4 1 1, 1 4 1, 1 1 4,
```

Note that the first output line "spilled over" to the next line (assuming an 80 character line, here) and your output might give the patterns in another order.

1998 Denison Spring Programming Contest
Problem 5
Land 'O Lakes

A geological survey receives satelite photos of various regions of the world and wishes to automate some of the more mundane bookkeeping chores. One is to count the number of islands in lakes. The photo comes as an n by m array of pixels. Each pixel is either 0 (indicating water) or 1 (indicating land). Number these pixels starting at the northwest corner as coordinate $(0,0)$ and proceed in the usual fashion to the southeast corner at coordinate $(n-1, m-1)$. You will be given the coordinates of a position on the map that will be water. You are to count the number of islands in the lake that contains this pixel. You are guaranteed that this lake is completely surrounded by land. Two pixels are connected if they are adjacent either horizontally or vertically. An island is a maximally connected set of land pixels in the lake that is not connected to the land surrounding the lake.

Input for this problem will be multiple problem instances. Each problem instance starts with a line containing the dimensions of the map, n and m, positive integers, both < 100, separated by a space, followed by a space, then the integers p and q, indicating the position of some water pixel on the map. (Again, you are guaranteed (p,q) is indeed in a lake surrounded by land.) The condition $n = m = p = q = 0$ indicates the end of input. If not the end of input, next follows n rows, each with m 0's and 1's, with no intervening spaces. The first such line gives positions $(0,0)$ through $(0, m-1)$. The next line gives positions $(1,0)$ through $(1, m-1)$ and so on. Each problem instance should generate one output line simply giving the number of islands in the lake indicated.

EXAMPLE:

Input:

```
16 20 2 5
11111111111111111111
10000000000110000011
11000000000010001001
11100000011000001101
11110000011100011101
11100000100100000001
11000001110110000111
11110001110000001111
11000000000000000111
11111111111101111111
11111111111000111111
11111111100000001111
11111111000011000111
11111111100011100011
11111111000000000011
11111111111111111111
6 8 3 3
11111111
10000001
10011001
11001101
11000001
11111111
0 0 0 0
```

should generate output:

```
4
1
```