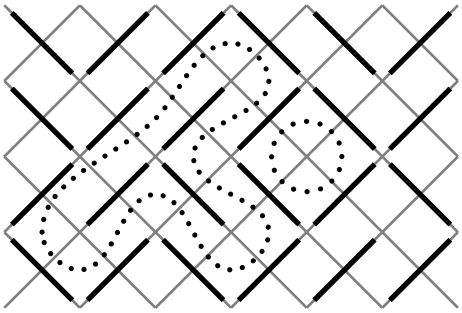
ACM International Collegiate Programming Contest 1999/2000
Mid-Central European Regional Contest

Problem A

Maze

Source: `maze.(c|cc|pas|java)`
Input: `maze.in`

By filling a rectangle with slashes (/) and backslashes (\), you can generate nice little mazes. Here is an example:



As you can see, paths in the maze cannot branch, so the whole maze only contains cyclic paths and paths entering somewhere and leaving somewhere else. We are only interested in the cycles. In our example, there are two of them.

Your task is to write a program that counts the cycles and finds the length of the longest one. The length is defined as the number of small squares the cycle consists of (the ones bordered by gray lines in the picture). In this example, the longer cycle has length 16 and the shorter one has length 4.

Input

The input contains several maze descriptions. Each description begins with one line containing two integers w and h ($1 \leq w, h \leq 75$), the width and the height of the maze. The next h lines represent the maze itself, and contain w characters each; all these characters will be either "/" or "\".

The input is terminated by a test case beginning with $w = h = 0$. This case should not be processed.

Output

For each maze, first output the line "`Maze #`n`:`", where n is the number of the maze. Then, output the line "k `Cycles; the longest has length` l`.`", where k is the number of cycles in the maze and l the length of the longest of the cycles. If the maze does not contain any cycles, output the line "`There are no cycles.`".

Output a blank line after each test case.

Sample Input

```
6 4
\//\\/
\///\/
//\\/\
\/\///
3 3
///
\//
\\\
0 0
```

Sample Output

```
Maze #1:
2 Cycles; the longest has length 16.

Maze #2:
There are no cycles.
```

ACM International Collegiate Programming Contest 1999/2000
Mid-Central European Regional Contest

Problem B

Date bugs

Source: `y2k.(c|cc|pas|java)`
Input: `y2k.in`

There are rumors that there are a lot of computers having a problem with the year 2000. As they use only two digits to represent the year, the date will suddenly turn from 1999 to 1900. In fact, there are also many other, similar problems. On some systems, a 32-bit integer is used to store the number of seconds that have elapsed since a certain fixed date. In this way, when 2^{32} seconds (about 136 Years) have elapsed, the date will jump back to whatever the fixed date is.

Now, what can you do about all that mess? Imagine you have two computers C_1 and C_2 with two different bugs: One with the ordinary Y2K-Bug (i. e. switching to $a_1 := 1900$ instead of $b_1 := 2000$) and one switching to $a_2 := 1904$ instead of $b_2 := 2040$. Imagine that the C_1 displays the year $y_1 := 1941$ and C_2 the year $y_2 := 2005$. Then you know the following (assuming that there are no other bugs): the real year can't be 1941, since, then, both computers would show the (same) right date. If the year would be 2005, y_1 would be 1905, so this is impossible, too. Looking only at C_1, we know that the real year is one of the following: 1941, 2041, 2141, etc. We now can calculate what C_2 would display in these years: 1941, 1905, 2005, etc. So in fact, it is possible that the actual year is 2141.

To calculate all this manually is a lot of work. (And you don't really want to do it each time you forgot the actual year.) So, your task is to write a program which does the calculation for you: find the first possible real year, knowing what some other computers say (y_i) and knowing their bugs (switching to a_i instead of b_i). Note that the year a_i is definitely not after the year the computer was built. Since the actual year can't be before the year the computers were built, the year your program is looking for can't be before any a_i.

Input

The input file contains several test cases, in which the actual year has to be calculated. The description of each case starts with a line containing an integer n ($1 \leq n \leq 20$), the number of computers. Then, there is one line containing three integers y_i, a_i, b_i for each computer ($0 \leq a_i \leq y_i < b_i < 10000$). y_i is the year the computer displays, b_i is the year in which the bug happens (i. e. the first year which can't be displayed by this computer) and a_i is the year that the computer displays instead of b_i.

The input is terminated by a test case with $n = 0$. It should not be processed.

Output

For each test case, output output the line "`Case #`k`:`", where k is the number of the situation. Then, output the line "`The actual year is` z`.`", where z is the smallest possible year

(satisfying all computers and being greater or equal to u). If there is no such year less than 10000, output "`Unkown bugs detected.`".

Output a blank line after each case.

Sample Input

```
2
1941 1900 2000
2005 1904 2040
2
1998 1900 2000
1999 1900 2000
0
```

Sample Output

```
Case #1:
The actual year is 2141.

Case #2:
Unknown bugs detected.
```

ACM International Collegiate Programming Contest 1999/2000
Mid-Central European Regional Contest

Problem C

Robbery

Source: `robbery.(c|cc|pas|java)`
Input: `robbery.in`

Inspector Robstop is very angry. Last night, a bank has been robbed and the robber has not been caught. And this happened already for the third time this year, even though he did everything in his power to stop the robber: as quickly as possible, all roads leading out of the city were blocked, making it impossible for the robber to escape. Then, the inspector asked all the people in the city to watch out for the robber, but the only messages he got were of the form "We don't see him."

But this time, he has had enough! Inspector Robstop decides to analyze how the robber could have escaped. To do that, he asks you to write a program which takes all the information the inspector could get about the robber in order to find out where the robber has been at which time.

Coincidentally, the city in which the bank was robbed has a rectangular shape. The roads leaving the city are blocked for a certain period of time t, and during that time, several observations of the form "The robber isn't in the rectangle R_i at time t_i" are reported. Assuming that the robber can move at most one unit per time step, your program must try to find the exact position of the robber at each time step.

Input

The input file contains the description of several robberies. The first line of each description consists of three numbers W, H, t ($1 \leq W, H, t \leq 100$) where W is the width, and H the height of the city and t is the time during which the exits are blocked.

The next line contains a single integer n ($0 \leq n \leq 100$), the number of messages the inspector received. The next n lines (one for each of the messages) consist of five integers t_i, L_i, T_i, R_i, B_i each. The integer t_i is the time at which the observation was made ($1 \leq t_i \leq t$), and L_i, T_i, R_i, B_i are the left, top, right and bottom coordinates respectively of the (rectangular) area which has been observed. ($1 \leq L_i \leq R_i \leq W, 1 \leq T_i \leq B_i \leq H$; the point $(1, 1)$ is the upper left hand corner, and (W, H) is the lower right hand corner of the city.) The messages mean that the robber was not in the given rectangle at time t_i.

The input is terminated by a test case starting with $W = H = t = 0$. This case should not be processed.

Output

For each robbery, first output the line "`Robbery #`k`:`", where k is the number of the robbery. Then, there are three possibilities:

If it is impossible that the robber is still in the city considering the messages, output the line "`The robber has escaped.`"

In all other cases, assume that the robber really is in the city. Output one line of the form "`Time step` τ`: The robber has been at` x`,`y`.`" for each time step, in which the exact location can be deduced. (x and y are the column resp. row of the robber in time step τ.) Output these lines ordered by time τ.

If nothing can be deduced, output the line "`Nothing known.`" and hope that the inspector will not get even more angry.

Output a blank line after each processed case.

Sample Input

```
4 4 5
4
1  1 1 4 3
1  1 1 3 4
4  1 1 3 4
4  4 2 4 4
10 10 3
1
2  1 1 10 10
0 0 0
```

Sample Output

```
Robbery #1:
Time step 1: The robber has been at 4,4.
Time step 2: The robber has been at 4,3.
Time step 3: The robber has been at 4,2.
Time step 4: The robber has been at 4,1.

Robbery #2:
The robber has escaped.
```

ACM International Collegiate Programming Contest 1999/2000
Mid-Central European Regional Contest

Problem D

Dreisam Equations

Source: `dreisam.(c|cc|pas|java)`
Input: `dreisam.in`

During excavations in the *Dreisamwüste*, a desert on some far away and probably uncivilized planet, sheets of paper containing mysterious symbols had been found. After a long investigation, the project scientists have concluded that the symbols might be parts of equations. If this were true, it would be proof that the *Dreisamwüste* was civilized a long long time ago.

The problem, however, is that the only symbols found on the sheets are digits, parantheses and equality signs. There is strong evidence that the people living in the *Dreisamwüste* knew only of three arithmetic operations: addition, subtraction, and multiplication. It is also known that the people of the *Dreisamwüste* did not have prioritization rules for arithmetic operations – they evaluate all terms strictly left to right. For example, for them the term $3 + 3 * 5$ would be equal to 30, and not 18.

But currently, the sheets do not contain any arithmetic operators. So if the hypothesis is true, and the numbers on the sheets form equations, then the operators must have faded away over time.

You are the computer expert who is supposed to find out whether the hypothesis is sensible or not. For some given equations (without arithmetic operators) you must find out if there is a possibility to place $+$, $-$, and $*$ in the expression, so that it yields a valid equation. For example, on one sheet, the string "`18=7 (5 3) 2`" has been discovered. Here, one possible solution is "`18=7+(5-3)*2`". But if there was a sheet containing "`5=3 3`", then this would mean that the *Dreisamwüste* people did not mean an equation when writing this.

Input

Each equation to deal with occupies one line in the input file. Each line begins with a positive integer (less than 2^{30}) followed by an equality sign "`=`". (For your convenience, the *Dreisamwüste* inhabitants used equations with trivial left sides only.) This is followed by up to 12 positive integers forming the right side of the equation. (The product of these numbers will be less than 2^{30}.) There might be some parentheses around groups of one or more numbers. There will be no line containing more than 80 characters. There is no other limit for the amount of the parentheses in the equation. There will always be at least one space or parenthesis between two numbers, otherwise the occurrence of white space is unrestricted.

The line containing only the number 0 terminates the input, it should not be processed.

Output

For each equation, output the line "`Equation #`n`:`", where n is the number of the equation. Then, output one line containing a solution to the problem, i. e. the equation with the missing $+$, $-$, and $*$ signs inserted. Do not print any white space in the equation.

If there is no way to insert operators to make the equation valid, then output the line "`Impossible.`".

Output one blank line after each test case.

Sample Input

```
18 = 7 (5 3) 2
30 = 3 3 5
18 = 3 3 5
5 = 3 3
0
```

Sample Output

```
Equation #1:
18=7+(5-3)*2

Equation #2:
30=3+3*5

Equation #3:
Impossible

Equation #4:
Impossible
```

ACM International Collegiate Programming Contest 1999/2000
Mid-Central European Regional Contest

Problem E

LC-Display

Source: `lcd.(c|cc|pas|java)`
Input: `lcd.in`

A friend of you has just bought a new computer. Until now, the most powerful computer he ever used has been a pocket calculator. Now, looking at his new computer, he is a bit disappointed, because he liked the LC-display of his calculator so much. So you decide to write a program that displays numbers in an LC-display-like style on his computer.

Input

The input file contains several lines, one for each number to be displayed. Each line contains two integers s, n ($1 \leq s \leq 10, 0 \leq n \leq 99\,999\,999$), where n is the number to be displayed and s is the size in which it shall be displayed.

The input file will be terminated by a line containing two zeros. This line should not be processed.

Output

Output the numbers given in the input file in an LC-display-style using s "-" signs for the horizontal segments and s "|" signs for the vertical ones. Each digit occupies exactly $s + 2$ columns and $2s + 3$ rows. (Be sure to fill all the white space occupied by the digits with blanks, also for the last digit.) There has to be exactly one column of blanks between two digits.

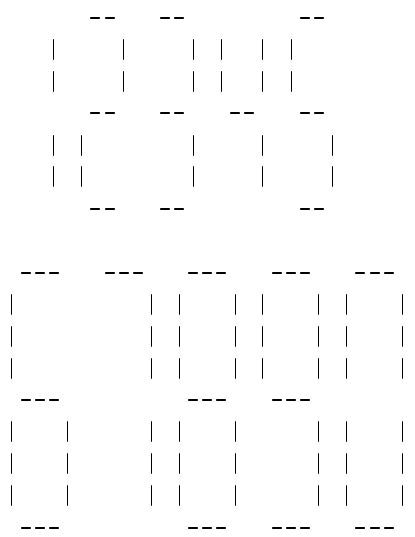
Output a blank line after each number.

You will find a sample of each digit in the sample output.

Sample Input

```
2 12345
3 67890
0 0
```

Sample Output



ACM International Collegiate Programming Contest 1999/2000
Mid-Central European Regional Contest

Problem F

Formatting Text

Source: `formatting.(c|cc|pas|java)`
Input: `formatting.in`

Writing e-mails is fun, but, unfortunately, they often do not look very nice, mainly because not all lines have the same lengths. In this problem, your task is to write an e-mail formatting program which reformats a paragraph of an e-mail (e. g. by inserting spaces) so that, afterwards, all lines have the same length (even the last line of each paragraph).

The easiest way to perform this task would be to insert more spaces between the words in lines which are too short. But this is not the best way. Consider the following example:

```
**************************
This is the example you are
actually considering.
```

Let us assume that we want to get lines as long as the row of stars. Then, by simply inserting spaces, we would get

```
**************************
This is the example you  are
actually        considering.
```

But this looks rather odd because of the large gap in the second line. By moving the word "`are`" from the first to the second line, we get a better result:

```
**************************
This  is  the  example   you
are  actually   considering.
```

Of course, this has to be formalized. To do this, we assign a *badness* to each gap between words. The badness assigned to a gap of n spaces is $(n-1)^2$. The goal of the program is to minimize the sum of all badnesses. For example, the badness of the first example is $1 + 7^2 = 50$, whereas the badness of the second one is only $1 + 1 + 1 + 4 + 1 + 4 = 12$.

In the output, every line has to start and to end with a word. (I. e. there cannot be a gap at the beginning or the end of a line.) The only exception to this is the following:

If a line contains only one word this word shall be put at the beginning of the line, and a badness of 500 is assigned to this line if it is shorter than it should be. (Of course, in this case, the length of the line is simply the length of the word.)

Input

The input file contains a text consisting of several paragraphs. Each paragraph is preceded by a line containing a single integer n, the desired width of the paragraph ($1 \leq n \leq 80$).

Paragraphs consist of one or more lines which contain one or more words each. Words consist of characters with ASCII codes between 33 and 126, inclusive, and are separated by spaces (possibly more than one). No word will be longer than the desired width of the paragraph. The total length of all words of one paragraph will not be more than 10000 characters.

Each paragraph is terminated by exactly one blank line. There is no limit on the number of paragraphs in the input file.

The input file will be terminated by a paragraph description starting with $n = 0$. This paragraph should not be processed.

Output

Output the same text, formatted in the way described above (processing each paragraph separately).

If there are several ways to format a paragraph with the same badness, use the following algorithm to choose which one to output: Let A and B be two solutions. Find the first gap which has not the same length in A and B. Do *not* output the solution in which this gap is bigger.

Output a blank line after each paragraph.

Sample Input

```
28
This is the example you are
actually considering.

25
Writing e-mails is fun, and with this program,
they even look nice.

0
```

Sample Output

```
This  is  the  example   you
are  actually   considering.

Writing e-mails  is  fun,
and  with  this  program,
they  even   look   nice.
```

ACM International Collegiate Programming Contest 1999/2000
Mid-Central European Regional Contest

Problem G

The Game

Source: `game.(c|cc|pas|java)`
Input: `game.in`

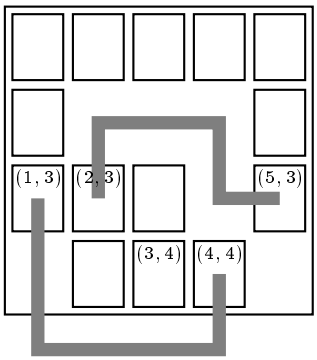
One morning, you wake up and think: "I am such a good programmer. Why not make some money?" So you decide to write a computer game.

The game takes place on a rectangular board consisting of $w \times h$ squares. Each square might or might not contain a game piece, as shown in the picture.

One important aspect of the game is whether two game pieces can be connected by a path which satisfies the two following properties:

1. It consists of straight segments, each one being either horizontal or vertical.
2. It does not cross any other game pieces.

It *is* allowed that the path leaves the board temporarily!
Here is an example:



The game pieces at $(1, 3)$ and at $(4, 4)$ can be connected. The game pieces at $(2, 3)$ and at $(5, 3)$ can be connected, too. On the other hand, the game pieces at $(2, 3)$ and $(3, 4)$ cannot be connected; each path would cross at least one other game piece.

The part of the game that you have to write now is the one testing whether two game pieces can be connected according to the rules above.

Input

The input file contains descriptions of several different game situations. The first line of each description contains two integers w and h ($1 \leq w, h \leq 75$), the width and the height of

the board. The next h lines describe the contents of the board; each of these lines contains exactly w characters: a "`X`" if there is a game piece at this location, and a space if there is no game piece.

Each description is followed by several lines containing four integers x_1, y_1, x_2, y_2 each satisfying $1 \leq x_1, x_2 \leq w, 1 \leq y_1, y_2 \leq h$. These are the coordinates of two game pieces. (The upper left corner has the coordinates $(1, 1)$.) These two game pieces will always be different. The list of pairs of game pieces for a board will be terminated by a line containing "`0 0 0 0`".

The entire input is terminated by a test case starting with $w = h = 0$. This test case should not be processed.

Output

For each board, output the line "`Board #`n`:`", where n is the number of the board. Then, output one line for each pair of game pieces associated with the board description. Each of these lines has to start with "`Pair` m`:  `", where m is the number of the pair (starting the count with 1 for each board). Follow this by "k `segments.`", where if k is the minimum number of segments for a path connecting the two game pieces, or "`impossible.`", if it is not possible to connect the two game pieces as described above.

Output a blank line after each board.

Sample Input

```
5 4
XXXXX
X   X
XXX X
 XXX
2 3 5 3
1 3 4 4
2 3 3 4
0 0 0 0
0 0
```

Sample Output

```
Board #1:
Pair 1: 4 segments.
Pair 2: 3 segments.
Pair 3: impossible.
```

ACM International Collegiate Programming Contest 1999/2000
Mid-Central European Regional Contest

Problem H

Dividing

Source: `divide.(c|cc|pas|java)`
Input: `divide.in`

Marsha and Bill own a collection of marbles. They want to split the collection among themselves so that both receive an equal share of the marbles. This would be easy if all the marbles had the same value, because then they could just split the collection in half. But unfortunately, some of the marbles are larger, or more beautiful than others. So, Marsha and Bill start by assigning a value, a natural number between one and six, to each marble. Now they want to divide the marbles so that each of them gets the same total value.

Unfortunately, they realize that it might be impossible to divide the marbles in this way (even if the total value of all marbles is even). For example, if there are one marble of value 1, one of value 3 and two of value 4, then they cannot be split into sets of equal value. So, they ask you to write a program that checks whether there is a fair partition of the marbles.

Input

Each line in the input file describes one collection of marbles to be divided. The lines contain six non-negative integers $n_1, \ldots, n_6$, where n_i is the number of marbles of value i. So, the example from above would be described by the input-line "`1 0 1 2 0 0`". The maximum total number of marbles will be 20000.

The last line of the input file will be "`0 0 0 0 0 0`"; do not process this line.

Output

For each collection, output "`Collection #`k`:`", where k is the number of the test case, and then either "Can be divided." or "Can't be divided.".

Output a blank line after each test case.

Sample Input

```
1 0 1 2 0 0
1 0 0 0 1 1
0 0 0 0 0 0
```

Sample Output

```
Collection #1:
Can't be divided.

Collection #2:
Can be divided.
```

ACM International Collegiate Programming Contest 1999/2000
Mid-Central European Regional Contest

Problem I

S-Trees

Source: `stree.(c|cc|pas|java)`
Input: `stree.in`

A Strange Tree (S-tree) over the variable set $X_n = \{x_1, x_2, \ldots, x_n\}$ is a binary tree representing a Boolean function $f : \{0,1\}^n \to \{0,1\}$. Each path of the S-tree begins at the *root* node and consists of $n+1$ nodes. Each of the S-tree's nodes has a *depth*, which is the amount of nodes between itself and the root (so the root has depth 0). The nodes with depth less than n are called *non-terminal* nodes. All non-terminal nodes have two children: the *right child* and the *left child*. Each non-terminal node is marked with some variable x_i from the variable set X_n. All non-terminal nodes with the same depth are marked with the same variable, and non-terminal nodes with different depth are marked with different variables. So, there is a unique variable x_{i_1} corresponding to the root, a unique variable x_{i_2} corresponding to the nodes with depth 1, and so on. The sequence of the variables $x_{i_1}, x_{i_2}, \ldots, x_{i_n}$ is called the *variable ordering*. The nodes having depth n are called *terminal* nodes. They have no children and are marked with either 0 or 1. Note that the variable ordering and the distribution of 0's and 1's on terminal nodes are sufficient to completely describe an S-tree.

As stated earlier, each S-tree represents a Boolean function f. If you have an S-tree and values for the variables $x_1, x_2, \ldots x_n$, then it is quite simple to find out what $f(x_1, x_2, \ldots x_n)$ is: start with the root. Now repeat the following: if the node you are at is labelled with a variable x_i, then depending on whether the value of the variable is 1 or 0, you go its right or left child, respectively. Once you reach a terminal node, its label gives the value of the function.

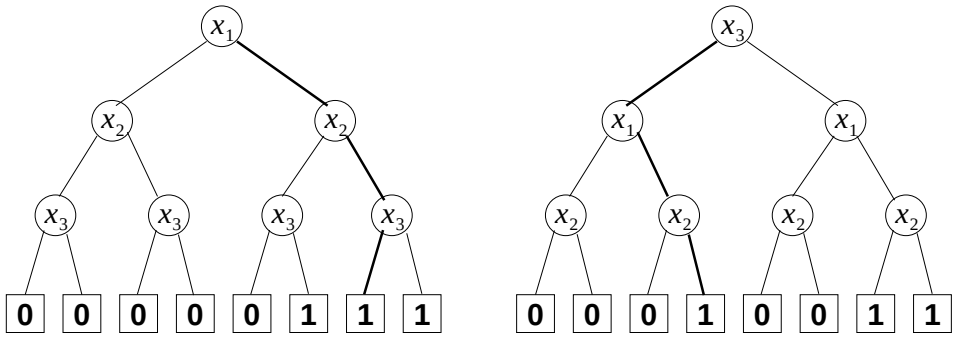


Figure 1: S-trees for the function $x_1 \wedge (x_2 \vee x_3)$

On the picture, two S-trees representing the same Boolean function, $f(x_1, x_2, x_3) = x_1 \wedge (x_2 \vee x_3)$, are shown. For the left tree, the variable ordering is x_1, x_2, x_3, and for the right tree it is x_3, x_1, x_2.

The values of the variables $x_1, x_2, \ldots, x_n$ are given as a *Variable Values Assignment* (VVA)

$$(x_1 = b_1, x_2 = b_2, \ldots, x_n = b_n)$$

with $b_1, b_2, \ldots, b_n \in \{0,1\}$ For instance, $(x_1 = 1, x_2 = 1, x_3 = 0)$ would be a valid VVA for $n = 3$, resulting for the sample function above in the value $f(1,1,0) = 1 \wedge (1 \vee 0) = 1$. The corresponding paths are shown bold in the picture.

Your task is to write a program which takes an S-tree and some VVAs and computes $f(x_1, x_2, \ldots x_n)$ as described above.

Input

The input file contains the description of several S-trees with associated VVAs which you have to process. Each description begins with a line containing a single integer n, $1 \leq n \leq 7$, the depth of the S-tree. This is followed by a line describing the variable ordering of the S-tree. The format of that line is `x`i_1 `x`i_2 `...x`i_n. (There will be exactly n different space-separated strings). So, for $n = 3$ and the variable ordering x_3, x_1, x_2, this line would look as follows:

```
x3 x1 x2
```

In the next line the distribution of 0's and 1's over the terminal nodes is given. There will be exactly 2^n characters (each of which can be 0 or 1), followed by the new-line character. The characters are given in the order in which they appear in the S-tree, the first character corresponds to the leftmost terminal node of the S-tree, the last one to its rightmost terminal node.

The next line contains a single integer m, the number of VVAs, followed by m lines describing them. Each of the m lines contains exactly n characters (each of which can be 0 or 1), followed by a new-line character. Regardless of the variable ordering of the S-tree, the first character always describes the value of x_1, the second character describes the value of x_2, and so on. So, the line

```
110
```

corresponds to the VVA $(x_1 = 1, x_2 = 1, x_3 = 0)$.

The input is terminated by a test case starting with $n = 0$. This test case should not be processed.

Output

For each S-tree, output the line "`S-Tree #`j`:`", where j is the number of the S-tree. Then print a line that contains the value of $f(x_1, x_2, \ldots, x_n)$ for each of the given m VVAs, where f is the function defined by the S-tree.

Output a blank line after each test case.

Sample Input

```
3
x1 x2 x3
00000111
4
000
010
111
110
3
x3 x1 x2
00010011
4
000
010
111
110
0
```

Sample Output

```
S-Tree #1:
0011

S-Tree #2:
0011
```