Problem 1: Breaking a Dollar

Using only the U. S. coins worth 1, 5, 10, 25, 50, and 100 cents, there are exactly 293 ways in which one U. S. dollar can be represented. Canada has no coin with a value of 50 cents, so there are only 243 ways in which one Canadian dollar can be represented. Suppose you are given a new set of denominations for the coins (each of which we will assume represents some integral number of cents less than or equal to 100, but greater than 0). In how many ways could 100 cents be represented?

Input

The input will contain multiple cases. The input for each case will begin with an integer *N* (at least 1, but no more than 10) that indicates the number of unique coin denominations. By *unique* it is meant that there will not be two (or more) different coins with the same value. The value of *N* will be followed by *N* integers giving the denominations of the coins.

Input for the last case will be followed by a single integer –1.

Output

For each case, display the case number (they start with 1 and increase sequentially) and the number of different combinations of those coins that total 100 cents. Separate the output for consecutive cases with a blank line.

Sample Input	Output for the Sample Input

`6 1 5 10 25 50 100`
`5 1 5 10 25 100`
`-1`
	`Case 1: 293 combinations of coins`
	
	`Case 2: 243 combinations of coins`

Problem 2: Rate of Return

Jill has been investing in a mutual fund for a while. Since her income has varied, the amount of money she has added to the investment has varied, and she hasn't always added to the investment at regular intervals.

Periodically Jill gets a report that indicates the total value of her investment. She wonders if she would have done better by investing her money in a savings account that pays a fixed interest rate. But to determine the answer to this question, she needs to know the equivalent interest rate that would have been paid on the mutual fund, had it paid a fixed rate. You are going to help her.

For simplicity we will assume that Jill added money to her mutual fund only at the beginning of a month, and that all months have the same length. We will further assume that the interest she would have been paid had she invested in a savings account would have been paid at the end of the month, and would have been compounded monthly.

Let's consider a simple example. Suppose Jill invested $100 at the beginning of January and another $100 in March. At the end of April she finds that the value of her mutual fund is $210. If the equivalent fixed monthly interest rate was i, then we know that at the end of January the value would have been $100 \times (1 + i)$. At the end of February the value would have been $100 \times (1 + i) \times (1 + i)$, or $100 \times (1 + i)^2$. At the end of March, the value would have been $100 \times (1 + i)^3 + 100 \times (1 + i)$, and at the end of April, the value would have been $100 \times (1 + i)^4 + 100 \times (1 + i)^2$. So the question to be answered in this case is this: what is the value of i such that $100 \times (1 + i)^4 + 100 \times (1 + i)^2 = 210$? The answer for this case is close to 0.01635179523.

Input Data

The input will contain multiple cases. The input for each case will begin with an integer N (no larger than 100) that indicates the number of times Jill invested in her mutual fund. This will be followed by $N + 1$ pairs, each pair containing an integer and a real number. The integer represents a month number (1 or larger) and the real number represents a positive non-zero dollar amount. The first N pairs give the month and amount of each of Jill's N investments in the mutual fund, and the last pair gives the value of the investment at the end of the specified month. There will be one or more whitespace characters (blanks, tabs, and/or ends of lines) between the input numbers. You may assume that the month numbers are given in ascending order.

Input for the last case will be followed by a single integer –1.

Output

For each case, display the case number (they start with 1 and increase sequentially) and the equivalent fixed monthly interest rate Jill's mutual fund would have paid. Display this number with five fractional digits. You may assume the interest rate will be no less than 0 and no larger than 1. Separate the output for consecutive cases by a blank line.

Sample Input	Output for the Sample Input
2 1 100.00 3 100.00 4 210.00 3 1 100.00 2 50.00 5 200.00 7 358.41 -1	Case 1: 0.01635 Case 2: 0.00520

Problem 3: The Zipper

Given three strings, you are to determine whether the third string can be formed by combining the characters in the first two strings. The first two strings can be mixed arbitrarily, but the characters from each must stay in their original order in the third string.

For example, consider forming "tcraete" from "cat" and "tree":

String *A*: **c a t**
String B: **t r e e**
String *C*: **t c r a e t e**

As you can see, we can form string C by alternately selecting characters from strings A and B, starting with string B. As a second example, consider forming "catrtee" from "cat" and "tree":

String *A*: **c a t**
String B: **t r e e**
String *C*: **c a t r t e e**

Finally, notice that it is impossible to form "cttaree" from "cat" and "tree".

Input

There will be multiple cases to consider. For each case the input consists of three lines, each containing a string, with the string possibly preceded and followed by one or more whitespace characters (blanks and tabs). Each string contains only upper and lower case letters. The length of the third string is always the sum of the lengths of the first two strings. The first two strings will have lengths between 1 and 200 characters, inclusive. The last case is followed by a single line that contains a period in the first column.

Output

For each case display a line containing the case number (starting with 1 and increasing sequentially), and the word "yes" or "no" to indicate whether the first two strings can be "zippered" to produce the third string. The output format is illustrated in the sample shown below.

Sample Input	Output for the Sample Input

cat
  tree
  tcraete
  cat
  tree
catrtee
cat
tree
cttaree
.
	Case 1: yes
	Case 2: yes
	Case 3: no

Problem 4: Lenny's Lucky Lotto Lists

Lenny likes to play the game of lotto. In the lotto game, he picks a list of *N* unique integers in the range from 1 to *M*. If his list matches the list of *N* integers that are selected randomly, he wins.

Lenny has a scheme that he thinks is likely to be lucky. He likes to choose his list so that each integer in it is at least twice as large as the one before it. So, for example, if *N* = 4 and *M* = 10, then the possible lucky lists Lenny could like are:

1 2 4 8
1 2 4 9
1 2 4 10
1 2 5 10

Thus Lenny has four lists from which to choose.

Your job, given *N* and *M*, is to determine from how many lucky lists Lenny can choose.

Input

There will be multiple cases to consider. The input for each is a pair of integers giving values for *N* and *M*, in that order. You are guaranteed that 1 <= *N* <= 10, 1 <= *M* <= 2000, and *N* <= *M*. The input for the last case will be followed by a pair of zeroes.

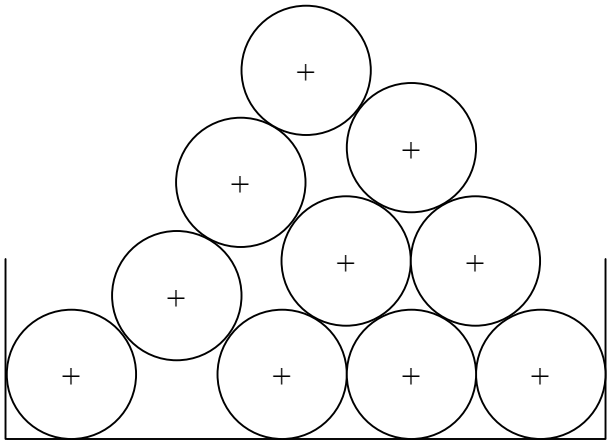
Output

For each case display a line containing the case number (starting with 1 and increasing sequentially), the input values for *N* and *M*, and the number of lucky lists meeting Lenny's requirements. The desired format is illustrated in the sample shown below.

Sample Input	Output for the Sample Input
4 10 2 20 2 200 0 0	Case 1: n = 4, m = 10, # lists = 4 Case 2: n = 2, m = 20, # lists = 100 Case 3: n = 2, m = 200, # lists = 10000

Problem 5: Stacking Cylinders

Cylinders (for example, oil drums) with a radius of one foot are stacked in a rectangular bin so that in a side view (as shown below) they look like a set of stacked circles. The cylinders in the bottom row (row 0) rest on the floor of the bin. Each cylinder in row k (for k > 0) rests on two cylinders in row k-1, which is the row below row k. Each row has one fewer cylinders than the row below it.



Given the position of the centers (show as "+" in the preceding illustration) of each cylinder on the bottom row, you are to find the position of the center of the cylinder on the top row.

Input

The input will contain multiple cases. The input for each case will begin with a positive non-zero integer *N* that indicates the number of cylinders on the bottom row. This will be followed by *N* real numbers giving the X coordinates of the centers of the cylinders on the bottom row. The Y coordinates of these cylinders are all 1.0 since the cylinders are resting on the floor of the bin, at which point Y is 0.0. The cylinders cannot overlap, and no cylinder in row *k* touches a cylinder in row *k – 2*. The last input case is followed by a single integer 0.

Output

For each case display the case number (starting with 1 and increasing sequentially), and the X and Y coordinates of the topmost cylinder. Display each coordinate with four fractional digits.

Sample Input	Output for the Sample Input
4 1.0 4.4 7.8 11.2 1 1.0 6 1.0 3.0 5.0 7.0 9.0 11.0 10 1.0 3.0 5.0 7.0 9.0 11.0 13.0 15.0 17.0 20.4 5 1.0 4.4 7.8 11.2 14.6 0	Case 1: 6.1000 4.1607 Case 2: 1.0000 1.0000 Case 3: 6.0000 9.6603 Case 4: 10.7000 15.9100 Case 5: 7.8000 5.2143

Problem 6: Going Home

A little region of the world is divided into equal-sized rectangular areas. In this little region there are *n* little men and *n* little houses. Every little man can move horizontally or vertically but not diagonally to an adjacent area, being paid a $1.00 travel fee for every move he makes between adjacent areas until he enters a little house. Your task is to compute the minimum travel fees required to get these *n* little men into those *n* little houses. The task is complicated by the restriction that each little house can accommodate only one little man.

The input is a map of the region, and has one of the characters '`.`', '`H`', or '`m`' in each area. A '`.`' identifies an empty area, an '`H`' identifies an area containing a little house, and an '`m`' identifies an area containing a little man.

Each area is quite large; it can hold up to *n* little men and a little house at the same time. A little man can also enter an area containing a little house without necessarily entering the little house. Initially, however, each area will hold at most one little man or one little house.

Input

The input will contain multiple cases. Each case starts with a line having two integers *N* and *M*; *N* is the number of rows (of areas) in the grid map, and *M* is the number of columns (of areas). The remainder of the input for the case will be *N* lines giving the map, one line for each row of areas. *N* and *M* are each between 2 and 100, inclusive. There may be one or more trailing whitespace (blank or tab) characters on a line. The number of 'H's on the map will equal the number of 'm's on the map, and there will be at most 100 houses. The last case will be followed by a line containing two integer zeroes.

Output

For each case, display the case number (they start with 1 and increase sequentially) and the minimum number of dollars required for travel fees. The output format should resemble that shown in the sample output.

Sample Input	Output for the Sample Input
2 2 .m H. 5 5 HH..m .... mm..H 7 8 ...H.... ...H.... ...H.... mmmHmmmm ...H.... ...H.... ...H.... 0 0	Case 1: $2 Case 2: $10 Case 3: $28

Problem 7: Jill's Tour Paths

Every year, Jill takes a bicycle tour between two villages. There are different routes she can take between these villages, but she does have an upper limit on the distance that she wants to travel. Given a map of the region indicating the cities and the roads between them (and their distances), Jill would like to have a list of the various routes between the selected cities that will meet her distance requirements. Your task is to write a program that will produce a list of these routes, in increasing order of distance.

We make the following assumptions.

- At most one road connects any pair of villages, and this road is two-way and has a non-zero positive distance.
- There are no roads that lead directly from a village back to the same village.
- Jill is only concerned about a one-way trip. That is, she is not concerned about returning to the village from which she starts her tour.
- Jill will not visit any village more than once during the tour.

Input

The input will contain several possible cases, each including a route map, identification of the start and destination villages, and the maximum distance Jill is willing to travel.

Each case appears in the input as a set of integers separated by blanks and/or ends of lines. The order and interpretation of these integers in each case is as follows:

- *NV* – the number of villages in the route map. This number will be no larger than 20.
- *NR* – the number of roads that appear in the route map. Each road connects a distinct pair of villages.
- *NR* triples, one for each road, containing *C1*, *C2*, and *DIST* – *C1* and *C2* identify two villages connected by a road, and *DIST* gives the distance between these villages on that road.
- *SV, DV* – the numbers associated with the start and destination villages; the villages are numbered 1 to *NV*.
- *MAXDIST* – the maximum distance Jill is willing to travel (one way).

Data for the last case will be followed by a single integer with the value –1.

Output

For each case, display the case number (1, 2, …) on the first line of output. Then, each on a separate additional line, list the routes that Jill might take preceded by the length of the route. Order the routes first by length, from shortest to longest. Within routes having the same length, order them in increasing lexicographic order. The sample input and output provide suitable examples, and the formatting shown there should be followed closely.

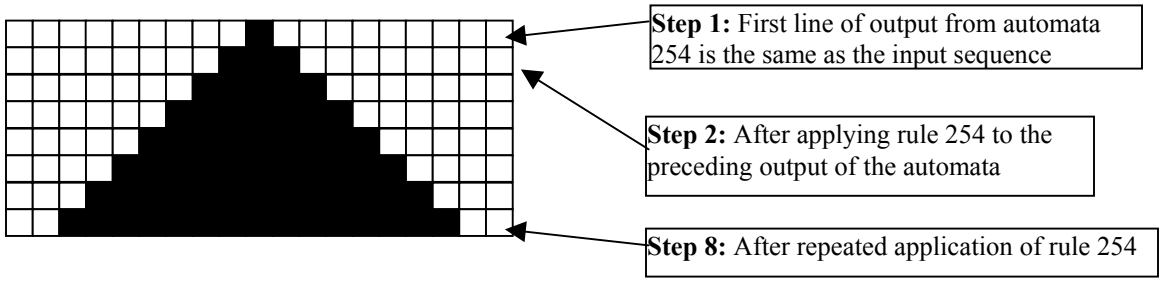
Separate the output for consecutive cases by a single blank line.

Sample Input	Output for the Sample Input
4 5 1 2 2 1 3 3 1 4 1 2 3 2 3 4 4 1 3 4 4 5 1 2 2 1 3 3 1 4 1 2 3 2 3 4 4 1 4 10 5 7 1 2 2 1 4 5 2 3 1 2 4 2 2 5 3 3 4 3 3 5 2 1 3 8 -1	Case 1: 3: 1 3 4: 1 2 3 Case 2: 1: 1 4 7: 1 3 4 8: 1 2 3 4 Case 3: 3: 1 2 3 7: 1 2 4 3 7: 1 2 5 3 8: 1 4 2 3 8: 1 4 3

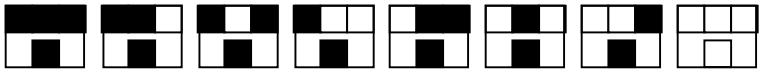
Problem 8: Boundaries on "A New Kind of Science"

Stephen Wolfram in "A New Kind of Science" describes a special kind of cellular automata. They consist of rows of blocks, where blocks are either filled or not filled depending on the previous row. To generate a new row of blocks, the automata looks at the preceding row and then follows a pre-set "rule" to either color or not color a square on the output row.

The diagram below illustrates the "output" from one of these special kinds of cellular automata.



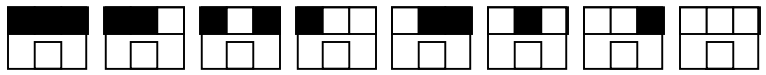
This was generated by repeated application of "Rule 254". The automaton was initialized with an input line that consisted of a single black square. Repeated application of rule 254 to the output line from the preceding step generated the black triangle shown above. Each "rule" can be described graphically using a set of eight input/output patterns, as illustrated below for rule 254.



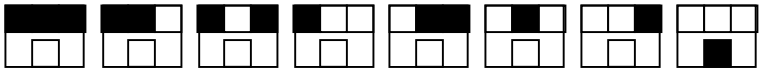
1 1 1 1 1 1 1 0 = RULE 254 (base 10)

The top row in each pattern gives one of eight possible color combinations for three adjacent cells (the middle cell and its left and right neighbors). The bottom row in a pattern specifies the color that the middle cell should be given on the next step for each of the 8 possible cases.

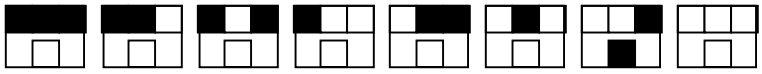
Given this arrangement, there are 256 generating rules or automata numbered as follows:



0 0 0 0 0 0 0 0 = RULE 0 (base10)

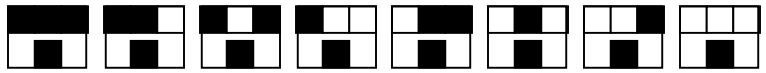


0 0 0 0 0 0 0 1 = RULE 1 (base10)



0 0 0 0 0 0 1 0 = RULE 2 (base10)

: : :



1 1 1 1 1 1 1 1 = RULE 255 (base10)

The automata for this problem operate in a bounded space. That is, each line output by an automaton consists of exactly *n* squares, and the first square and the last square of the output are always white, regardless of the rule being used for the automata. While an automaton can examine the first and last square of its input line when computing the next line, it cannot change the first or last square when it outputs the next line. These two squares must always remain white.

Bounded automata will always start life on a line with an odd number of squares, and all of the squares except the middle square are white. The middle square for step 1 is always black.

For every input case, your program must determine which, if any, of the 256 possible automata could have generated the particular line given for that input case. If none of the automata can generate the specified line by the given maximum step number (also given in the input), you must display "NONE". If one or more automata can generate the specified line, then you must identify them (as described below).

Input

The input will contain multiple cases, with the input for each case occupying a single input line. For each case the input line will contain a positive integer which specifies the maximum step number for the automata to run. After one or more whitespace (blank or tab) characters there will appear an odd-length string of characters which represents the "squares" on the line. The character "W" represents a white square and "B" represents a black square. If this string contains characters other than "W" or "B", or if the string is invalid (not a properly bounded string as previously described), then the string cannot be generated by any of the automata, and the output will be "NONE" (as illustrated below). The string may possibly be followed by one or more whitespace characters before the end of line. The input line for the last case will be followed by a line containing only the integer –1.

Output

For each input case, display the case number (1, 2, …) followed by pairs of numbers (*rule*, *step*), where *rule* is the rule number of the automata (0 through 255) that generated a particular output and *step* is the first step in the sequence of outputs at which the automata generated the desired output. If more than one rule generated the desired output sequence before or on the maximum step number, list a pair for each rule number that generated the desired output, in order, from the lowest automata number to the highest automata number, with a single space between each output pair. If none of the automata generate the particular line from the input file before or on the maximum step number, then display the case number and the word NONE. Your output should closely resemble that shown in the sample output provided.

Sample Input

```
3 WBWBWBWBW
1000 WBWBWBWBBBW
5235 WBWWBWWBBBBWWBBWWWWWWBBWWWBBWWWWWWBBWWBBBBWWBWWBW
5 WBWBDCWBW
-1
```

Output for the Sample Input

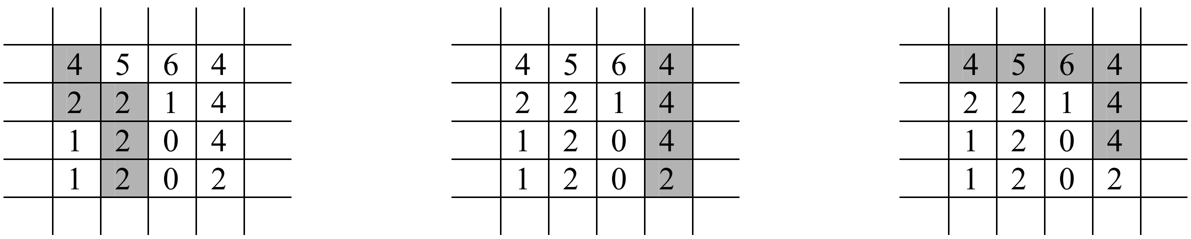
```
Case 1: (91,3)
Case 2: (15,8)(158,11)(159,14)(243,8)
Case 3: (129,84)(161,84)
Case 4: NONE
```

Problem 9: Detecting Explosives

There are many different kinds of explosives, each having slightly different properties. One method of detecting concealed explosives in luggage is to take an X-ray and look for areas where the density of the image falls into the appropriate range for a recognized explosive signature. Your job is to automate this detection, given the signatures of the explosives and an X-ray image.

The X-ray image will be provided as a two-dimensional array of integers in the range 0 to 2000, each representing the density of a small region. Each explosive signature is a triple (*L*, *H*, *M*), where *L* and *H* represent the lowest and highest values in the image that can appear in a particular explosive type, and *M* represents the minimum total "mass" (or cumulative density) of a connected region that could possibly contain that type of explosive. "Connected" means that the cells in the X-ray image of the explosive are adjacent horizontally or vertically, but not diagonally.

For example, assume we have two explosive signatures (2, 4, 12) and (4, 6, 22), and part of an X-ray image as shown in the three illustrations below. In the left and middle illustrations explosives of type one (2, 4, 12) have been detected (as shown by the shaded cells). An explosive of type two (4, 6, 22) was detected in the third illustration. Note that the cells occupied by different types of explosives may overlap.



Each time an explosive is detected, an "alarm" is generated. You are to process images to determine how many alarms for each explosive type will be generated, if any.

Input

The input will contain multiple cases. Each case will begin with a series of no more than 10 integer triples, each giving the L, H, and M values for a type of explosive to be detected; this is followed by three integer zeroes. Then there will appear 100 integers representing a 10-row, 10-column X-ray image, in row-major order (the first 10 integers are the first row, the second 10 are the second row, and so forth). The last case is followed by three integer zeroes.

Output

The output for each case should begin with a line giving the input case number (starting with 1 and increasing sequentially). If there are no alarms for an input case, the output should then contain a line containing the text "No alarms generated." If, however, the X-ray image does trigger one or more alarms, you should display one line for each type of alarm generated indicating the type and number of such alarms. These should be displayed in increasing order of alarm type. All lines except the line giving the case number should be indented four columns, and the output for consecutive cases must be separated by a blank line. Use the sample as a guide to the desired output format.

Sample Input

```
2 4 12
4 6 22
0 0 0
4 5 6 4 0 0 0 0 0 0
2 2 1 4 0 0 0 0 0 0
1 2 0 4 0 0 0 0 0 0
1 2 0 2 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
9 9 27
20 20 40
1 9 15
0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 5 0 0 0 0 0
5 1 1 1 0 0 0 0 0 0
0 0 0 1 5 0 0 0 9 9
0 0 0 0 0 0 0 0 5 5
0 0 0 0 0 0 0 0 9 9
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 9 9 9
1 5 8
3 10 4
0 0 0
1 1 0 1 1 1 0 1 0 7
1 1 0 1 1 1 0 1 0 1
1 1 0 1 1 1 0 1 1 1
1 1 0 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 1 1 1
1 1 1 1 0 9 0 1 1 1
1 1 1 1 0 0 0 1 1 1
1 1 1 1 1 1 1 1 1 1
0 0 0
```

Output for the Sample Input

```
Case 1:
    2 alarms of type 1
    1 alarm of type 2

Case 2:
    1 alarm of type 1
    3 alarms of type 3

Case 3:
    2 alarms of type 1
    2 alarms of type 2
```

Problem 10: The Take-Grant Protection Model

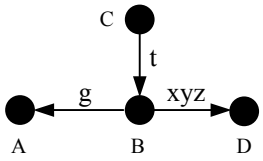
In this problem we consider a limited form of the "take-grant protection model." In this model, a system is represented as a graph. Vertices are normally either subjects (users or processes, the active agents in the system) or objects (for example, files). Directed edges between vertices are labeled to indicate the rights that the source vertex has with respect to the destination vertex.

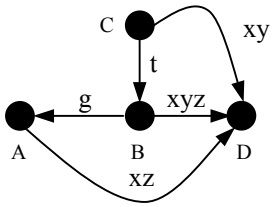
In this limited form of the model, we will consider only subject vertices.

Two rights of particular interest give this model its name. The "take" right, indicated by the letter "t", means a subject can take (a copy of, as its own) a right possessed by the vertex pointed to by the edge labeled "t". The "grant" right, identified by the letter "g", means a subject can give (a copy of) a right relative to a vertex to another subject. When the "take" or "grant" occurs, a new edge is created in the graph if necessary, or the additional rights are added to the label on an existing edge. Note that rights are always specified with respect to a particular vertex.

As an example, consider the graph shown on the left below. Subject B possesses rights x, y and z to D as indicated by the edge from B to D labeled "xyz". Likewise, B has the right to grant to A some or all of the rights it has; this is indicated by the edge from B to A labeled "g". Finally, subject C has the right to take (copy) from B any of the rights it possesses (indicated by the edge from C to B labeled "t").

Suppose B should grant the rights "xz to D" to subject A, and C should take the rights "xy to D" from B. Then the resulting graph would be as shown on the right. Note that C could also have taken the "grant to A" right possessed by B, which would have added an edge from C to A labeled "g".





Rights can be exercised at any time, and the same right can be exercised repeatedly. Subjects may also perform two additional actions. "Create" can be used by a subject *S* to add a new vertex *V* to the graph and an edge from *S* to *V*; this new edge can be labeled with any rights desired. "Remove" allows a subject *S* to remove some or all of the rights it possesses to a subject *V*; if all of the rights from S to *V* are removed, so is the edge from S to V. If a vertex has no edges to or from it, then it is removed. Every subject always has the right to create a new subject vertex with arbitrary rights to it, and to remove any rights it already possesses to vertices.

Given the graph for an initial system state, and a graph showing a possible future state, we wish to know whether that future system state could be reached using only the four types of actions (take, grant, create, and remove) just described. Rights are represented by a set of 26 or fewer lowercase letters (with 'g' and 't' indicating "grant" and "take" rights). The subject vertices in the possible future state are the same as those in the initial state (which does not preclude the possibility that additional subject vertices may be created and removed between the initial and possible future state).

Input

There will be multiple cases to consider. The input for each case consists of the graph describing the initial system state, followed by the graph for the possible future system state. Each of these graphs is specified by several input lines. The first line contains an integer *NV* giving the number of vertices (never more than 10) and an integer *NE* giving the number of edges. The labels for the vertices are the first *NV* uppercase letters (A, B, …). Then there is a sequence of *NE* lines, one for each edge. Each of these lines begins with a pair of uppercase letters *S* and *D* (separated by a space) identifying the source and destination vertices for an edge in the graph, a space, and a string containing between 1 and 26 lowercase alphabetic characters identifying the rights *S* has with respect to *D*.

The last case is followed by a line containing two integer zeroes.

Output

For each case, display the case number (1, 2, …) and the word "yes" or "no" to indicate if the second graph could, or could not be obtained from the first graph by some sequence of take, grant, create, and remove rule applications. Separate the output for consecutive cases with a blank line.

Sample Input	Output for the Sample Input
4 3 B A g C B t B D xyz 4 5 B A g C B t B D xyz C D xyz A D xyz 3 2 A B x C B ty 3 2 A B x C B tyx 0 0	Case 1: yes Case 2: no