Preamble

Please note the following very important details:

1) Read all input from the keyboard, i.e. use `stdin`, `System.in`, `cin` or equivalent. Input will be redirected from a file to form the input to your submission.

2) Write all output to the screen, i.e. use `stdout`, `System.out`, `cout` or equivalent. Do not write to `stderr`. Do NOT use, or even include, any module that allows direct manipulation of the screen, such as `conio`, `Crt` or anything similar. Output from your program is redirected to a file for later checking. Use of direct I/O means that such output is not redirected and hence cannot be checked. This could mean that a correct program is rejected!

3) Unless otherwise stated, all *integers* in the input will fit into a standard 32-bit computer word. If more than one integer appears on a line, they will be separated by white space, i.e. blanks or tabs.

4) An uppercase letter is a character in the sequence 'A' to 'Z', a lowercase letter is a character in the sequence 'a' to 'z', and a letter is either.

5) Unless otherwise stated, a *word* or a *name* is a continuous sequence of letters.

6) Unless otherwise stated, a *title* is a continuous sequence of letters and spaces starting and ending with a letter. A title will only ever occur on a line by itself.

7) Unless otherwise stated, a *string* is a continuous sequence of non whitespace characters.

8) Unless otherwise stated, words, names and titles will not contain more than 60 characters, and strings will not contain more than 250 characters. If the problem demands a different limit, then the actual limit will be placed in parentheses after the term in the problem description, thus 'title(20)' refers to a sequence of no more than 20 letters and spaces.

9) If it is stated that 'a line contains no more than *n* characters', this does not include the character(s) specifying the end of line.

Problem A Who’s First? 3 points

In this problem you are given a number of sets of words. In each set, you are to determine which word comes first alphabetically. You should ignore case, so “apPle” will come before “Bat” but after “AnT”, for example.

Each set starts with a single integer, *n* ($2 \leq n \leq 1000$), which gives the number of words in the set. The next *n* lines contain one word(20) per line. A set will never contain two words that differ only in case. The last line of input will contain 0 on its own – this line should not be processed.

For each set of words, output just the word that comes first alphabetically, on a line by itself, as it appears in the input.

Sample input

```
3
Cat
fat
bAt
4
call
ball
All
Hall
0
```

Sample output

```
bAt
All
```

Problem B Rugby Club 3 points

The All Golds Rugby Club categorises members as Senior or Junior. Anyone over 17 is a Senior, as is anyone weighing 80 kg or more. Everyone else is a Junior. Your task is to correctly classify club members.

Each line of input contains a name followed by two positive integers representing age and weight, in that order. The last line of input will be # 0 0. This line should not be processed.

For each line of input you must output a single line containing the name, followed by a space and the appropriate category. Note that the category must begin with an upper case letter.

Sample input

```
Joe 16 34
Bill 18 65
Billy 17 65
Sam 17 85
# 0 0
```

Sample output

```
Joe Junior
Bill Senior
Billy Junior
Sam Senior
```

Problem C Your Days Are Numbered 3 points

There are many situations where we need to calculate the number of days between two dates, for instance, how many days leave have you taken between the start and end of your holiday. The easiest way to do those calculations is to assign an integer number to each day by choosing a suitable starting date and counting the number of days since then. For this problem we will use the first of January of the relevant year as the starting date.

This means that the day number of today (13th August 2005) is 225. Last year, the day number of 13th August was 226 because 2004 was a leap year. A leap year occurs when the year is divisible by 4 with the exception that years divisible by 100 are not and with the further exception that years divisible by 400 are, thus 2000 and 1976 were leap years, but 1900 and 1977 weren't. Hopefully you also know that most months have 31 days, except for April, June, September and November which have 30, and February which has 28 (29 in a leap year).

Input will consist of a number of dates each on a separate line. Each date will consist of three positive integers representing the day, month and year respectively. The day and month will always be valid and the year will always lie between 1800 and 2200 (both dates inclusive). Input will be terminated by a line containing 0 0 0 — this line should not be processed.

Output will be one line for each line of input, the line containing just the day number.

Sample input

```
14 8 2004
1 1 2004
31 1 1976
1 3 1974
1 3 1976
0 0 0
```

Sample output

```
227
1
31
60
61
```

Problem D Digit Sums 3 points

A simple operation that you can perform on a number is to add up its digits. If the result has more than 1 digit, the process may be repeated until a single digit answer is given. For example, applying the operation to 673 gives 7; 6 + 7 + 3 = 16 and 1 + 6 = 7.

Input will consist of a number of lines, each containing a single positive number less than 100,000. The last line of input will contain 0 – this line should not be processed.

Output will consist of a line containing the digit sum (a single digit number) for each line of input.

Sample input

```
673
51
1000
99
0
```

Sample output

```
7
6
1
9
```

Problem E Magickology 10 points

Matheos the magician wishes to organise her considerable collection of squares into categories based on the amount of magical power that each one possesses. A square is a table of numbers with an equal number of rows and columns and each square has an associated sum (*S*), the sum of the first column. Matheos wishes to categorise them based on the following properties:

Not Magick	At least one row or column does not sum to *S*.
Semi-Magick Square	Each row and column sums to *S* but at least one of the diagonals doesn't.
Weakly-Magick Square	All the rows, columns and diagonals sum to *S*.
Strongly-Magick Square	All the rows, columns and diagonals sum to *S*, and the numbers are distinct (i.e. no number is duplicated).
Magically-Magick Square	All the rows, columns and diagonals sum to *S*, the numbers are distinct and they are consecutive (i.e. they form a sequence with no gaps).

Thus

2	3
3	2

is a Semi-Magick square because each row and column adds up to 5, but the diagonals don't.

and

8	1	6
3	5	7
4	9	2

is a Magically-Magick square because each row, column and diagonal adds up to 15, each number is distinct, and the numbers are consecutive (i.e. 1 2 3 4 5 6 7 8 9).

Your task is to help Matheos by writing a program to read and evaluate a sequence of squares. For each square your program is to output a message to say whether it is Magically-Magick, Strongly-Magick, Weakly-Magick, Semi-Magick, or has no magical properties at all.

Input will consist of a sequence of squares. Each square starts with an integer, *n*, specifying the square's size (number of rows and columns; $2 \le n \le 8$) on a line by itself, followed by *n* lines, each with *n* integers. The sequence of squares is terminated by a line containing a zero (0).

Output for each square is a single line starting with the word 'Square' followed by a space, a sequence number (starting at 1) and a colon (:). This is then followed by another space and then one of following massages 'Magically-Magick Square', 'Strongly-Magick Square', 'Weakly-Magick Square', 'Semi-Magick Square', or 'Not a Magick Square' as appropriate.

Sample input

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

Sample output

```
Square 1: Not a Magick Square
Square 2: Semi-Magick Square
Square 3: Magically-Magick Square
```

Problem F Super 12 10 Points

In a rugby-mad country such as New Zealand, fans eagerly await the ranking of the teams in league tables such as the Super 12 (soon to be the Super 14). These rankings are really only meaningful at the end of a round, when all teams have played the same number of games.

A team is awarded 4 points for each win, 2 points for each draw and 0 for each loss. Any team scoring 4 or more tries in a game is awarded a bonus point, as is any team that loses a game by less than 8 points. At the end of a round we can produce a table showing the relative standings of each team, i.e. a table sorted in descending order of points. If two or more teams have the same number of points, then apply the following rules there are no tied teams:

Sort in descending order of spread, i.e. the cumulative difference between total points scored and total points against.
Sort in descending order of the total number of tries they scored.
Sort in ascending alphabetic sequence.

Input will consist of the results for a single league and will consist of the names of the competing teams and details of each game. The list of competing teams will be a series of no more than 20 lines each containing a single name(20) and terminated by a line consisting of a single '#'. This will be followed by a series of lines giving the results of each game, also terminated by a line consisting of a single '#'. Each game will start with the names of the two teams (home side first) followed by 4 integers representing, in order: the score of the home team, the score of the away team, the number of tries scored by the home team, and the number of tries scored by the away team (see the sample input). Note that there will not be any indication of the end of a round, it is determined by the number of competing teams (which will always be even), i.e. a round will end when every team has played exactly one game.

Output will consist of a league table for each round. The first line of the league table will consist of the word 'Round' followed by a space and the number of the round (a running number starting at 1). This will be followed by the teams listed in order of their standing, according to the rules outlined above. Each line consists of the name of the team, followed by, starting in column 22 and right justified in fields of widths as specified in parentheses, their points (2), their cumulative score since the beginning of the league (4), the cumulative score of the teams they have played (4), the total number of tries that they have scored (3) and the total number of tries scored against them (3). Leave a blank line between rounds. See the sample output below.

Sample Input

```
Highlanders
Crusaders
Warriors
ThisIsTheLongestName
#
Highlanders Crusaders  25 24  3 2
Warriors ThisIsTheLongestName 32  16 5  1
ThisIsTheLongestName Highlanders 10 20 1 2
Crusaders Warriors 17 16 3 3
#
```

Sample Output

```
Round 1
Warriors              5  32  16  5  1
Highlanders           4  25  24  3  2
Crusaders             1  24  25  2  3
ThisIsTheLongestName  0  16  32  1  5

Round 2
Highlanders           8  45  34  5  3
Warriors              6  48  33  8  4
Crusaders             5  41  41  5  6
ThisIsTheLongestName  0  26  52  2  7
```

Problem G GPS Encoding 10 Points

Traditionally, simple codes have taken a permutation of the alphabet, numbered each character in the range 01 to 26 (or 00 to 25) and then encrypted the message as a string of digits. It is then relatively easy to conceal the structure of the text by breaking the string into groups of a fixed length.

This problem turns that encoding around and assumes that one has a sequence of digits that one wishes to encrypt (possibly a GPS location that you want to transmit to friend, telling her where a treasure is located). We could do this the simple way and use only 10 characters to encrypt the 10 digits, or we could use all 26 letters and use the additional letters to encrypt suitable pairs of digits. Thus the sequence `941177` could be encoded as `9 4 1 1 7 7` or as `9 4 11 7 7` or as `9 4 1 17 7`.

Given a permutation of the upper case letters (which implicitly defines an encoding of the numbers (0)0, (0)1, …, 25) and a sequence of digits, determine the shortest encryption of the sequence as a sequence of letters. Note that decryption of such a sequence may not be unique.

Input will consist of a series of encryption problems. Each problem will begin with a permutation of the uppercase letters, followed by a series of lines each containing a string of between 3 and 20 digits. The string of digits will be terminated by a single zero (‘0’) on a line by itself. The series of problems will be terminated by a line consisting of a ‘#’ on a line by itself. Neither of these lines should be processed.

Output for each problem will consist of a line starting with ‘Problem ’ followed by the problem number, a running number starting at 1. This will be followed by a series of lines, one for each digit sequence in the input for that problem, giving the shortest encoding for that sequence. If there are several such encodings, then choose the lexicographically greatest (i.e. the one that would appear nearest the end of a dictionary if they were to be considered as words). Leave a blank line between successive problems.

Sample Input

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
111
234
0
CUGEBRHKNXDMIQSLTFVWOYPJZA
111213
11121
212223
0
#
```

Sample Output

```
Problem 1
LB      //Could be BBB, BL or LB. Choose LB as shortest and lexicographically greatest
XE      //CDE is longer

Problem 2
MIQ
UMY
YPJ
```

Problem H Custom table sorter 10 Points

Web site users are often presented with data in tables. Different users may want table rows displayed in different orders. For a web site listing available hotels, possible orders include hotel name, hotel locality, hotel star rating and room rate. For this problem you are to write a piece of support software for possible inclusion in such a site.

Input will consist of a number of data sets. Each data set consists of a heading line, a table section and a sorter section. The heading line contains the title of the data set. The sequence of data sets is terminated by a (heading) line consisting of a single '#'. A table section consists of at least 1 and no more than 20 lines, terminated by a line consisting of a single '#'. Each line contains between 1 and 10 fields, separated by commas; each field contains a string(20). All lines have the same number of fields and there are no empty fields. A sorter section consists of several sorter lines. Each sorter line contains one or more field sorters separated by commas, each consisting of a field number (a distinct number in the range 1 to the number of fields), and a direction ('A' or 'D'). A sorter section is terminated by a line consisting of '0#'.

The output starts with the title of the data set, followed by several groups of lines, indented two spaces and separated by a blank line between groups. Each group consists of the contents of the table section, sorted according to the corresponding sort specification. Sorting is primarily done based on the first field sorter, and the second and subsequent field sorters are only used for those rows with the same value(s) in the field(s) used by previous field sorter(s). If there are still ties (equal elements), the tied elements should appear in the order of the original table. Leave a blank line between the output for successive data sets.

Sample input

```
Hotel rooms and locations
Lowton_Hotel,Airport,**
Hotel_foobar,CBD,*
Dug_Inn,Airport,*
#
3A,1D
2A
0#
General enquiry
One_value,here
#
1A
0#
#
```

Sample output

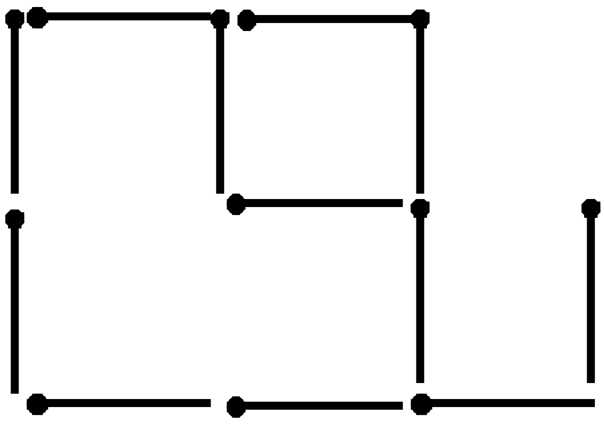
```
Hotel rooms and locations
  Hotel_foobar,CBD,*
  Dug_Inn,Airport,*
  Lowton_Hotel,Airport,**

  Lowton_Hotel,Airport,**
  Dug_Inn,Airport,*
  Hotel_foobar,CBD,*

General enquiry
  One_value,here
```

Problem I A game with matches 30 Points

Consider a grid of matches on a table, like so:



The problem involves counting the number of squares formed by these matches. The arrangement above, for example, forms two squares.

Input consists of a sequence of arrangements. Each arrangement starts with a line containing two integers representing the number of rows and columns in the arrangement, (r and c, $1 \le r\ c \le 20$), followed by $2r$+1 lines representing the arrangement. Odd numbered rows will consist of c characters where each character is either a hyphen ('–'), representing a horizontal match or an asterisk ('*'), representing a space. Even numbered rows will consist of c+1 characters where each character is either a bar ('|'), representing a vertical match, or an asterisk ('*'), representing a space. Input will be terminated by a line containing two zeroes (0 0).

Output will consist of one line for each arrangement giving the number of squares that can be found in the arrangement followed by a space and the word 'squares'.

Sample Input

```
2 3
--*
|||*
*-*
|*||
---
2 2
--
|||
--
|||
--
0 0
```

Sample output

```
2 squares
5 squares
```

Problem J Checking Causality 30 Points

The concept of causality is important when monitoring distributed systems. Consider the event of one computer sending a message and the event of that message being received at another computer. Causality tells us that the send event must occur before the receive event. If there were a global clock to timestamp all of the events that occur in a distributed system, then the timestamp of a message receive event would always be greater than the timestamp of a message send event.

However, distributed systems do not have such a global clock, all they have is a collection of idiosyncratic clocks, one in each computer, all ticking at different speeds. All we can say for certain is that each clock never stops and that its (apparent) granularity is fine enough that any sequence of events on a given computer will be given a series of timestamps that is strictly monotonically increasing. Thus it is perfectly feasible to have a message that left A at time 50 arrive at B at time 40 and for the reply to leave B at 45 and arrive at A at 51, where all times are given according to the relevant computer's clock.

Given send and receive timestamps from a collection of messages exchanged by a group of computers, it is possible to use the concept of causality to make some checks that will detect faulty clocks. Continuing the example above, assume that a second message is sent from B at time 45 on B's clock (i.e. after the arrival of the first message). If A receives this before 51 (on its clock) then at least one clock is faulty, because causality has been breached (apparently), since we know that all clocks increase by at least 1 tick between successive events on the same computer. It is also possible to detect causality violations in longer chains, e.g., A to B, B to C, C to A.

Write a program to process sets of message timestamp data, and for each set say whether causality is violated, i.e. whether there is a cycle of messages, starting and ending at the same computer, where one or more messages has apparently travelled back in time.

Input consists of a number of timestamp sets. Each timestamp set starts with the number of messages in the set (*n*), followed by *n* lines of timestamp data, where each line contains: sending computer, send time, receiving computer, and receive time, all separated by whitespace. Times are according to the relevant computer, computers are identified by a single upper case letter and times are integers. The sequence of timestamp sets will be terminated by a line containing a single zero (0).

Output consists of a single line for each timestamp set. This line contains the word 'OK' if there is no causality violation or the word 'Bad' if there is such a violation.

Sample input

```
2
A  50  B 40
B 45  A  49
2
A  50  B  40
B 45 A 51
2
A 50 B 40
B 39 A 49
0
```

Sample output

```
Bad
OK
OK
```

Problem K Olympic Ranking 30 Points

Every year since 1896, the nations of the world have competed in the Olympic Games for glory, medals and (particularly in recent years) media coverage. Every day eager fans will scan the medal table to see where their team is placed, which, particularly for teams from smaller countries, usually means badly. Part of the problem is that the medal table is sorted strictly in the order: number of gold medals, number of silvers, number of bronzes. This means that a team with 6 silvers and 10 bronzes but only 1 gold is placed well behind a team with 2 golds and nothing else. The situation could be alleviated by allocating a value to each medal type, calculating a total score for each team by multiplying the numbers of each medal type they have won by the appropriate value and then ranking the teams on that. This then raises the problem of determining the values, with the obvious proviso that a gold has to be worth more than a silver which, in turn, has to worth more than a bronze. To keep numbers manageable (remember we are dealing with the general population, not Computer Science students), values must be in the range 1 to 99.

For this problem you will be given a medal table, that is, a list of countries together with the numbers of medals they have won, from which your program is to determine, for each country in the table, what allocation of values to medal types will give that country the best possible placing, where the placing of a country is the number of rivals with a larger score, plus 1.

Input will consist of several scenarios (Games). Each will start with the name of the city where the Games were held, followed by an unsorted sequence of lines containing the name of a country followed by 3 integers representing the number of gold, silver, and bronze medals won by that country, and terminated by a line containing a single '#'. There will be no more than 100 teams in any one Games and no team will win more than 1000 medals. The sequence of Games will be terminated by a line consisting of only a single '#'.

Output (for each scenario) will consist of a header line followed by a line for each team in the input in the order they appeared in the input. The header line will consist of 'Olympic Games in ' followed by the name of the relevant city. Each team line will consist of the name of the team followed by 4 integers, separated by single spaces, giving the medal values that results in their highest placing, and that placing. Remember that the values must be such that gold > silver > bronze and that they must be in the range 1 to 99. If there are several allocations of values that produce the same best placing, then choose the one that produces the smallest 6 digit number *ggssbb*, where *gg*, *ss*, and *bb* are the relevant values (including leading zeroes if necessary). Leave a single blank line between scenarios (Games).

Sample Input

```
Rome
UnitedStates 100 200 300
China 102 198 297
NewZealand 1 2 3
#
Auckland
UnitedStates 100 200 300
China 102 198 301
NewZealand 1 2 3
#
#
```

Sample Output

```
Olympic Games in Rome
UnitedStates 3 2 1 1
China 4 2 1 1
NewZealand 3 2 1 3

Olympic Games in Auckland
UnitedStates 3 2 1 2
China 3 2 1 1
NewZealand 3 2 1 3
```

Problem L Bus Timetable 30 Points

A typical bus timetable has a column of stops down the left hand side, followed by a series of columns specifying a particular service and labelled with the number of a bus route. Each entry in the table, i.e. each intersection of the row for a given stop and a column for a given service, will be either blank or contain the time that that service is scheduled to leave that stop.

As you can imagine, producing these timetables by hand is very difficult and error prone, so this is where you come in. Write a program to produce a timetable, given details of the various routes and services.

Input will consist of a number of scenarios. Each scenario will start with the title of the scenario, followed by a number of lines, one for each route in the scenario. The sequence of scenarios will be terminated by a line consisting of a single ‘#’.

The line for a route will start with the time that the first service for that route leaves the depot (hours and minutes) followed by the interval between services (minutes). This will be followed by a series of pairs of integers representing travel times and bus stops. The list of routes will be terminated by a line containing two zeroes (0 0) and will never contain details of more than 10 routes. Note that routes are numbered implicitly, starting from 1, bus stops are also numbered from 1 and are always visited in order (apart from the return to the depot), there will never be more than 99 stops, and that the depot is effectively bus stop 0 (thus the last bus stop number in the list will always be 0). Note that no buses leave before 6:00 am and all services terminate (i.e. are back in the depot) by midnight, thus you should not generate a service that violates these constraints.

Output will consist of one timetable per scenario. Each timetable will start with the name of the scenario on a line by itself, followed by a heading line detailing the services (sorted in order of departure time from the depot, see example) followed by as many lines as there are bus stops mentioned in the input. Each line starts with the number of the stop (two columns) followed by an entry for each service. Each entry is 6 columns wide and starts with a ‘|’ followed either by 5 spaces (if that service does not stop there) or a departure time in the form hh:mm, using a 24 hour clock (i.e., including leading zeroes). The line is terminated by a ‘|’. Note that times increase monotonically downwards, but not necessarily across. Leave a blank line between scenarios.

Sample Input

```
MadeUpAsIWentAlong
7 0 240 20 1 10 2 20 3 10 0
9 0 240 20 1 10 2 20 3 10 0
0 0
#
```

Sample Output

```
MadeUpAsIWentAlong
  |    1|    2|    1|    2|    1|    2|    1|    2|    1|
 1|07:20|09:20|11:20|13:20|15:20|17:20|19:20|21:20|23:20|
 2|07:30|09:30|11:30|13:30|15:30|17:30|19:30|21:30|23:30|
 3|07:50|09:50|11:50|13:50|15:50|17:50|19:50|21:50|23:50|
```

Problem M Conflicting Strings 100 points

In 2417 archaeologists discovered a large collection of 20th century text documents of vital historical importance. Although there were many duplicated documents it was soon evident that, as well as the damage due to time making much of the text illegible, there were also some disagreements between them. However, it was noticed that groups of texts could be made consistent, i.e. consistency between texts could be achieved by leaving out some (small) number of texts. For example, the texts:

```
    ap***
    ab*le
    app*e
    *p**e
```

(where * denotes an illegible character) can be made consistent by removing just the second text.

Input will consist of a sequence of sets of texts. Each set will begin with a line specifying the number of texts in the set, and the maximum number of texts which can be removed. This will be followed by the individual texts, one per line. Each text consists of at least one and no more than 250 characters, either lower case letters or asterisks. All the texts in a set will be the same length and there will be no more than 10,000 texts in a set. The sequence of sets is terminated by a line containing two zeros (0 0).

Output for each set consists of a line containing one of the words 'Yes' or 'No' depending on whether or not the set can be made consistent by removing at most the specified number of texts.

Sample input

```
4 1
ap***
ab*le
app*e
*pple
3 1
a
b
c
4 2
fred
ferd
derf
frd*
0 0
```

Sample output

```
Yes
No
No
```

Problem N 100 Points

This problem has been withdrawn for technical reasons

Problem O Adventure Game 100 points

Part of an adventure game requires the adventurer to make her way through a magical maze of numbered rooms. Each room has a set of numbered doors that lead to the indicated rooms. Additionally, each room might contain either a leprechaun or a troll.

Whenever the adventurer visits a room containing a leprechaun, the leprechaun will ensure that she leaves with at least a certain number of gold pieces (GP) in her sack. That is, if she arrives with fewer GP than the prescribed amount, then the leprechaun will "top her up" to that amount; however, if she arrives with the prescribed amount or more, then her supply of GP will remain unchanged. Each time the adventurer visits a room containing a troll, the troll will demand a toll of a certain number of gold pieces which must be paid before continuing.

The adventurer begins with 0 GP and the problem is to determine whether the adventurer can make her way from room 1 to the highest numbered room. If the highest numbered room contains a troll, the adventurer must also be able to pay the toll on arrival.

Input will consist of a sequence of mazes. Each maze will begin with a line containing an integer specifying the number of rooms in the maze (n, $1 \le n \le 1000$). This will be followed by n lines specifying the rooms in ascending order. Each line will specify the contents of the room (empty (E), a leprechaun (L) or a troll (T)), followed by a number specifying the number of GP that the leprechaun will top up to, or that the troll will require from the adventurer. (For empty rooms this amount is 0.) The rest of the line consists of a list of 1 or more numbers in the range $1..n$ and terminated by a zero (0), representing the numbers on the doors of that room. The sequence of mazes is terminated by a line containing a single zero (0).

Output will consist of a sequence of lines, one for each maze — either "Yes" or "No" indicating whether it is possible to reach the final room from the first one.

Sample input

```
3
E 0 2 0
L 10 3 0
T 15 1 2 0
4
E 0 2 3 0
L 201 2 3 0
L 10 4 0
T 15 2 3 1 0
0
```

Sample output

```
No
Yes
```

Problem P I'm a Frayed Knot 100 points

On the table in front of you lie a bunch of pieces of coloured thread. All the threads are different colours and the ends have been arranged in a single line, for example red, blue, green, green, blue, red. Note that each colour appears exactly twice, once for each end of that thread. You've been asked to tie the threads into a single large loop by successively tying together ends of some pair of adjacent threads. In the above example you could start by tying end 1 to end 2 or end 2 to end 3 but not end 3 to end 4 since that would make a green loop. Likewise, if your first tie was red to blue, then your second tie could not join the remaining red and blue.

Finding the job a little boring, you decide instead to count the number of ways in which you could perform the task, i.e. the number of sequences of ties that you could do. For instance, suppose that the initial pattern was `rrbb`, then there is only one allowed sequence of ties (join the middle two, then join the two remaining). On the other hand if it were `rbrb`, then there are three allowed sequences (join any consecutive pair to begin with, then join the remaining two).

Likewise you can see that if the initial pattern were `rgrbgb`, then four of your initial ties lead to a subsequent pattern of the general form `abab`, while one leads to `abba`. Thus there are 4x3 + 1x2 = 14 ways to complete the ties in this case.

Input will consist of a sequence of colour patterns represented by words of length at most 22 consisting of lower case letters and will be terminated by a line containing the single character '#'.

Output will be a sequence of lines, one for each line in the input, each containing a the number of ways to tie off the pattern in the corresponding input line. If an input line is invalid (because it does not contain exactly two occurrences of each of its characters) the output for that line should be 0.

Sample input

```
rrbb
rbrb
rgrbgb
gbg
#
```

Sample output

```
1
3
14
0
```