Problem A Format Checking 5 points

One of the stated aims of the so-called 'streamless' education system is that students' academic records follow them from one institution to another. This sounds fine except that the formats, the number of fields and even the characters used to separate fields can all differ. Thus one institution may have 6 fields separated by commas while another may have 8 fields separated by tabs. In the general case this can be a very difficult problem, but for this situation we will simplify it considerably.

Write a program that will read in a number of lines containing fields separated by commas. There should be 6 fields in each line and hence 5 commas, no more and no less. Some fields may be blank; we wish to know how many of them.

Input consists of a number of lines containing characters. The input is terminated by a line consisting of a single '#'. This line should not be processed.

For each line determine how many commas it contains. If it does not contain exactly 5 commas write 'Invalid' and continue to the next line. If it does, determine how many empty fields there are and write this number to standard output.

Sample Input

```
Name,Address1,Address2,Address3,City,Country
Name,Address1,Address2,City,Country
Name,Address1,Address2,,City,Country
Name,Address1,Address2, ,City,Country
#
```

Sample Output

```
0
Invalid
1
1
```

Problem B Words Words 5 points

It is surprisingly common for people to repeat a word in writing, even when it makes no sense. Here is an example from the documentation of a well-known and valuable tool-kit:

> If a command pipeline is opened for writing, keystrokes entered into the console are not visible until the the pipe is closed.

Write a program which reads some text from standard input and reports the adjacent repeated words. All the letters will have been converted to lower case first. To count as adjacent, there must be no letters between the words, but there may be any number of spaces, line terminations, and punctuation marks.

Input will be a sequence of up to 50 lines, terminated by a line consisting of a single '#'. The lines may contain 0 to 80 printing characters, including spaces, punctuation marks, and lower case letters. A word is a maximal non-empty sequence of letters.

Sample Input

```
if a command pipeline is opened for writing,
keystrokes entered into the console are not
visible until the the pipe is closed.

closed pipes cannot be reopened.  row, row,
row your boat, gently down the stream.
#
```

Sample Output

```
the
closed
row
row
```

Problem D Choices 5 points

Typically, when an organisation gets too big for all the members to be involved in its organisation, the solution is to form a committee to run it for them. There are many ways in which a committee can be formed, in fact $N!/(k!(N-k)!)$, where N is the size of the organisation and k is the size of the committee. The ! operator refers to the factorial function, defined as:

$N! = 1 \times 2 \times 3 \times \ldots \times N$.

Thus $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$.

Write a program that will determine the number of ways a committee of size k can be formed from a group of size N.

Input consists of a number of lines containing pairs of integers (N and k) separated by spaces. N will be in the range 1 to 12 and k will be in the range 1 to N. The list will be terminated by a line containing two zeroes (0 0).

For each line in the input (each N, k pair), output the number of ways in which the committee of size k can be formed from N members.

Sample Input

```
5 3
6 2
0 0
```

Sample Output

```
10
15
```

Problem C Registration Plates 5 points

New Zealand, in common with many countries, has a centralised car registration system with a simple, incremental, system of generating new plates. At least I believe so, I have never actually seen a vehicle with a number 'next to' mine — close admittedly, but never next to.

This then raises the problem of what we mean by close. Which of these is closest to NZ3868: NZ3878? NZ3870? NY3868? MZ3868? What about NA3868?

For the purposes of this problem we will define 'closeness' to be the overall numeric similarity. To determine this we count the number of differences and form their absolute sum. The first criterion is the number of differences (fewer is closer); if these are equal then the smaller sum determines the closeness. The examples should make this clear. The differences are between the characters as they stand, without wrap-around, so the difference between 'A' and 'Z' is 25 not 1.

Input consists of a number of lines, each containing three strings representing New Zealand registration plates, i.e. two upper case letters followed by 4 digits. These strings will be separated by exactly one space. Your task is to determine which of the second two strings is closest (under the above definitions) to the first. The list is terminated by a line consisting of only the character #.

For each line except the terminal line print the closest plate to the first one. If both are equally close, print both.

Sample Input

```
NZ3868 NZ3768 NZ3859
NZ3868 NY3869 NZ3969
NZ3868 NZ3969 NZ3669
AA0001 AA0002 AB0001
#
```

Sample Output

```
NZ3768
NY3869 NZ3969
NZ3969
AA0002 AB0001
```

Problem E Simple String Comparison 5 points

Most programming languages make it easy for you to compare characters and strings according to some computer character set. Unfortunately, this doesn't match *human* conventions very well. For example,

Context	Order
Dictionary	Aardvark < elephant < Zebra
ASCII	Aardvark < Zebra < elephant
EBCDIC	elephant < Aardvark < Zebra

The full ISO specification for how to sort according to human conventions runs to 150 printed pages, requires large tables (not least because it has to deal with nearly 40 000 characters), and allows up to seven passes. It needs four passes for English. Here's how they go:

1. Compare characters left to right, ignoring everything that isn't a letter, and paying no attention to alphabetic case or accents. Some letters (such as 'æ') are mapped to several letters (such as 'ae') before comparison. If a difference is found, report it, otherwise continue.
2. Compare characters left to right, ignoring everything that isn't a letter, and paying no attention to alphabetic case. Accents are noticed, and ligatures (such as 'æ') remain single letters. If a difference is found, report it, otherwise continue.
3. Compare characters left to right, ignoring everything that isn't a letter. This time, pay attention to alphabetic case as well as accents. If a difference is found, report it, otherwise continue.
4. Compare all the characters left to right. If a difference is found, report it, otherwise the strings are equal.

You have a much easier task: implement just pass 1, assuming that the character set is ASCII, so you only need to consider the letters 'A' ...'Z' and 'a' ... 'z'.

Input consists of pairs of lines each containing at least 1 and no more than 72 characters and is terminated by a pair of lines each consisting of a single '#'.

Output one of the following codes on a line by itself for each pair of lines in the input, except the terminating pair.

LT if the first line is less than the second line (according to pass 1 above);
EQ if the first line is equal to the second line (according to pass 1 above);
GT if the first line is greater than the second line (according to pass 1 above);

Sample input

```
Aardvark
elephant
elephant
Aardvark
elephant
Zebra
Eland
elephant
**this will FOX them!
This will fox   THEM.
#
#
```

Sample Output

```
LT
GT
LT
LT
EQ
```

Problem H Bank Statement 15 points

A bank statement is an every-day document that most people will be familiar with. In this problem, you need to produce a series of human readable bank statements from some raw input data.

Each set of raw data begins with the letter 'O' followed by the opening balance of the account. This is followed by the transactions that have occurred in the current month, terminated by a line starting with 'E'. Four pieces of information are given for each transaction:

- the transaction type ('D' for deposit or 'W' for withdrawal).
- the day of the month on which the transaction occurred (it is assumed that each statement covers a single calendar month).
- the amount of the transaction (withdrawals reduce the balance; deposits increase it).
- a text field that describes the transaction.

A single space character appears between input fields. Input is terminated by a line containing a single '#'. This line must immediately follow an 'E' line.

Write a program to produce a bank statement for each set of raw data (between an 'O' and an 'E'). A bank statement consists of a line for the opening balance, followed by the various transactions terminated by a closing balance, followed by a blank line. The transactions are listed in the order they appear in the input.

Each non-blank line in a statement consists of 6 or 7 fields:

- a two character field for the day of the month, followed by a space.
- a 16 character field for the transaction type
- a 23 character field for the transaction description.
- an 8 character field for the deposit amount.
- an 8 character field for the withdrawal amount.
- an 8 character field for the current balance.
- if the balance is negative there is a final field containing " OD" (if the balance is positive then the last character in the line is the final digit of the balance).

The justification of each field is demonstrated in the sample output below, where the digits on the first line indicate spacing only and are **not** to be reproduced. All spacing is to be done using space characters (tabs must not be output). Note that the input values are such that the specified field widths are never exceeded.

Sample Input

```
O 123.45
D 2 0.53 Riccarton
W 7 200.96 Pak'n'save
D 16 16.20 Some Money Machine
E
O -99.99
E
#
```

Sample Output

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
   Opening balance                                                  123.45
 2 Deposit         Riccarton                    0.53                123.98
 7 Withdrawal      Pak'n'save                            200.96      76.98 OD
16 Deposit         Some Money Machine          16.20                 60.78 OD
   Closing balance                                                   60.78 OD

   Opening balance                                                   99.99 OD
   Closing balance                                                   99.99 OD
```

Problem I Crossword Helper 15 points

Cryptic crossword compilers often embed the answer in the clue. For example, the answer to the clue "Liquid stew at Eric's" might be water. You have been asked to write a program that will help people find answers of this type.

Input consists of a number of pairs of lines. The first line of each pair contains the text of the clue. This line contains at least one character and no more than 80. The second line of each pair contains a "pattern" that represents what is known about the answer (this line also contains at least one character and no more than 80). The characters that can appear in a pattern are the lower case letters and the ? character. The end of input is indicated by a clue line that consists of just the # character.

Output for each pair of lines is the heading "Output for clue N" (where N is a counter that starts at 1) followed by the 0 or more matches for the specified pattern in the specified clue, followed by a blank line. For a match to occur, a sequence of letters from the clue must match the characters specified in the pattern (a question mark matches any character). All non-letters in the clue (spaces, digits, punctuation, and so on) are ignored in the matching process. If there are two or more matches, they must be printed in the order encountered when scanning the clue from left to right. All matches are output in lower case.

Sample Input

```
Liquid stew at Eric's
w??e?
aaaaaaaaaa
b?
Abba's baby (boys)!
b?b?
#
```

Sample Output

```
Output for clue 1
water

Output for clue 2

Output for clue 3
baby
bybo
```

Problem J Tramping by torchlight 15 points

The Out-in-the-Wop-Wops Tramping Club (OTC) has a reputation for running trips in which parties end up tramping in the dark. For some reason, it always turns out that exactly one person in the party is in possession of a torch. Things become tricky when such a party has to cross a bridge in the dark. DOC assigns a maximum capacity to each bridge. If the party is too big to cross the bridge together then it must cross the bridge in groups. When a group crosses the bridge one of the members of the group must have the torch (so they can see where they are going). Also, the group must stick together, which means that it must travel at the speed of the slowest member of the group. Of course once a group reaches the other side, one member of the group must cross the bridge again to convey the torch back to the remaining members of the party.

The OTC have commissioned you to write a program that, given details of a particular party and bridge capacity, computes the shortest possible time in which the party could cross the bridge. The input for each scenario consists of several positive integers:

- C, the bridge capacity (≥ 2).
- N, the number of members of the party (in the range 1 to 30)
- N integers that specify for each party member how long it takes them to cross the bridge.

Input is terminated by C = 0.

Output consists of the shortest possible time it takes the entire party to cross the bridge.

Sample Input

```
2 3 1 2 3
3 4 10 6 7 4
0
```

Sample Output

```
6
20
```

Problem K Squash Ladder 15 points

A ladder is an ordered list of players (individuals or teams) that specifies (hopefully) their ranking. That is, someone high on the ladder is better than someone low on the ladder. Thus a typical ladder could look be:

Alfred
Brian
Carol
Desdemona
Evita
Frank

To move up the ladder, players can challenge anyone up to 3 places above them. If the higher ranked player wins, then nothing changes. If the lower ranked player wins then (s)he moves above her/his opponent, and everyone moves down one place. Thus, for instance, if Evita beats Brian (she could have challenged Desdemona or Carol) the ladder becomes:

Alfred
Evita
Brian
Carol
Desdemona
Frank

You are to write a program to maintain the ladder. Note that a game which was valid when it was set up, may be invalid by the time it is finished because of movements in the ladder. Assume that Desdemona challenges Alfred while Evita and Brian are playing. If Evita wins, Brian is 'out of reach' of Desdemona and hence their game becomes invalid. If Evita loses then the Desdemona-Brian game is still valid.

Input will consist of a number on a line by itself, giving the number of players in the ladder (more than 5 and fewer than 100), followed by that many names, each name on a line by itself, representing the initial state of the ladder. Each name will be no more than 20 characters in length, with no embedded spaces.

This will be followed by a series of transaction lines, either games or requests to show the ladder. A game line consists of the letter 'G' followed by the winner and loser in the game. All names will have appeared in the list already and all fields will be separated by 1 space. A print request is the letter 'P' on a line by itself. The file is terminated by a # on a line by itself.

Output will consist of the current state of the ladder whenever a print request is read. Separate successive printouts by a blank line.

Sample Input

```
6
Alfred
Brian
Carol
Desdemona
Evita
Frank
G Evita Brian
P
G Desdemona Alfred
P
#
```

Sample Output

```
Alfred
Evita
Brian
Carol
Desdemona
Frank

Alfred
Evita
Brian
Carol
Desdemona
Frank
```

Problem L Time Difference 15 points

In Unix, the standard way of recording a timestamp is as the number of complete seconds that have elapsed since 00:00:00 on January 1st 1970, thus any file created at or after 11:30:00 but before 11:30:01 on January 1st 1970 will be time stamped 41400. In calculating the number of seconds that have elapsed since 00:00 on 1970-01-01 it is assumed that every day is 86400 seconds long. Unfortunately, this is not entirely true. Because modern atomic clocks measure time very accurately, and because the earth's speed of rotation is gradually decreasing (which means that the solar day is slowly lengthening), leap seconds are occasionally inserted to keep the standard timescale used for civil purposes (UTC) synchronised with the solar day.

When a leap second is added, it is added as the last second of June or December. The following table summarizes the 23 leap seconds added to date:

June:	1972, 1981-3, 1985, 1992-4, 1997
December:	1971-9, 1987, 1989, 1990, 1995, 1998

As can be seen from the table, irregularities in the rotation speed of the earth mean that intervals between successive leap seconds vary somewhat. When the first leap second was added to the end of December 1971, the following sequence of time stamps occurred:

UTC timestamps	Unix timestamps
1971-12-31 23:59:58	63071998
1971-12-31 23:59:59	63071999
1971-12-31 23:59:60	
1972-01-01 00:00:00	63072000
1972-01-01 00:00:01	63072001

As you can see from the table, the Unix time scale doesn't allow for leap seconds. Another way to think of this is that some "Unix seconds" last for two "physical seconds". For example, the Unix second that began at 1971-12-31T23:59:59 lasted two physical seconds. This causes problems if you want to know the number of physical seconds that elapsed between two UNIX timestamps. Two seconds actually elapsed between 63071999 and 63072000, but the difference between the two Unix timestamps is only 1 second.

Input takes the form of a series of lines that each contain two integers that represent Unix timestamps (the end of input is indicated by a line that contains two -1 values). Each integer lies in the range 0 to 946684799 (which represents UTC time 1999-12-31 23:59:59).

The output consists of one integer for each pair of integers in the input. The output is the number of physical seconds in the interval that starts when the first Unix timestamp starts and ends just when the second Unix timestamp starts, i.e if a pair is ut1 ut2 in that order, then the output integer is:

complete physical seconds elapsed before ut2 – complete physical seconds elapsed before ut1.

Sample Input

```
63071999 63072000
20   10
-1 -1
```

Sample Output

```
2
-10
```

Problem O The Messy Professor 50 points

Professor Blarg has laboured long and hard over many pages of multiplications. Unfortunately, he is very untidy, and many of the figures he has written are illegible. Consequently, he has asked you to assist him in reconstructing the calculations he has done.

Input consists of a number of multiplications (there is one multiplication per line; end of input is marked by a line containing only a #). The format of each multiplication is:

num1 × num2 = product

The spacing is exactly as shown (in other words, each line contains four spaces, two around the '×' and two around the '='). The three 'integers' (*num1*, *num2* and *product*) consist of digits and '?' characters. The digits represent information that could be read from the original manuscripts. The *num1* and *num2* strings are between 1 and 3 characters in length and the *product* string is between 1 and 6 characters in length.

Your program must attempt to find digits to replace all ? characters in the multiplication so that when *num1* and *num2* are multiplied the answer is indeed *product*. Note that the first digit in each number, whether supplied in the input or computed by your program is never 0. If you can find a valid substitution, then the details of the multiplication should simply be printed. So far no case examined has had more than one solution, however, if that situation should occur then print any of them. Otherwise, the string "No solution found for " should be printed followed by the original multiplication you were trying to solve.

Sample Input

```
? x 3 = 9
?? x 11 = 12?
?? x ?? = ?
#
```

Sample Output

```
3 x 3 = 9
11 x 11 = 121
No solution found for ?? x ?? = ?
```

Problem P Long Patience 50 points

This card game is played with a standard deck of 52 cards. Each card is a unique combination of one of 4 suits (spades, clubs, diamonds, hearts) and one of 13 face-values (Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king). The game is played as follows. The top 48 cards of the deck are laid face down in a grid that has 4 rows each of 12 cards. The top card in the deck is put at the top left hand corner of the grid. The next 11 cards are used to fill positions 2 through to 12 of the top row (row 1). Cards 13 to 24 then form the second row, and so on until the 48th card is placed in the bottom right hand corner, leaving 4 cards over which form a 'spares' pile.

Play starts by turning over the top card of the pile of four spares. This card is then placed in its rightful place on the grid. Hearts go in the top row, diamonds in the second, clubs in the third and spades in the bottom row. Aces go in the left hand column, 2s to 10s in the column of the same number, jacks in column 11, queens in column 12 and kings in column 13 (after the original deal only columns 1 to 12 contain any cards; column 13 is initially empty). To place the first card in position, a face down card will have to removed to make room for the first card (unless the first card is a king — see later). The displaced card is then placed in its correct position which in turn displaces another card and so the game continues. This process continues until a king (of any suit) is encountered. A king is placed in the empty column 13, which means there is no card to displace. At this point the second card in the spares pile is turned over, and the placement sequence begins again.

Once the second king has been placed, the third card in the spares pile is turned over, and when the third king is turned up the fourth and final card in the spares pile is turned over. The game ends once the 4th king has been placed. You are to write a program to play this game.

The input consists of a series of card deck specifications. Each card deck specification gives the order in which the cards appear in the deck (the first card in the input is the one on the top of the deck when the deal begins). Each card deck is specified in two lines. Each line lists 26 cards, with a single space between card specifications. Cards are represented as a two character code. The first character is the face-value (A=Ace, 2--9, T=10, J=Jack, Q=Queen, K=King) and the second character is the suit (C=Clubs, D=Diamonds, H=Hearts, S=Spades). The final line of the input file contains a # as its first character.

For each card deck in the input, your program simulates the game and then prints the grid as it appears at the end of a game. A single space separates card specifiers, and there is a single blank line after each grid. Each face down card is represented by the sequence ##.

Sample Input

```
QD AD 8H 5S 3H 5H TC 4D JH KS 6H 8S JS AC AS 8D 2H QS TS 3S AH 4H TH TD 3C 6S
8C 7D 4C 4S 7S 9H 7C 5D 2S KD 2D QH JD 6D 9D JC 2C KH 3D QC 6C 9S KC 7H 9C 5C
AH KH QH JH TH 9H 8H 7H 6H 5H 4H 3H 2H 2S AS KS QS JS TS 9S 8S 7S 6S 5S 4S 3S
3C 2C AC KC QC JC TC 9C 8C 7C 6C 5C 4C 4D 3D 2D AD KD QD JD TD 9D 8D 7D 6D 5D
#
```

Sample Output

```
## 2H 3H ## 5H 6H 7H 8H 9H TH JH QH KH
AD 2D 3D 4D 5D 6D 7D 8D ## ## JD ## KD
AC 2C 3C 4C 5C 6C 7C 8C 9C TC JC QC KC
AS 2S 3S 4S ## 6S 7S 8S 9S TS JS QS KS

## 2H ## ## ## ## ## ## ## ## ## ## KH
AD 2D 3D 4D 5D 6D 7D 8D 9D TD JD QD KD
AC ## ## ## 5C 6C ## ## ## ## ## ## KC
AS 2S ## 4S 5S 6S 7S 8S 9S TS JS QS KS
```

Problem Q The psychic robot 50 points

A group of psychology students has grown tired of getting rats to run mazes, and plans to get robots to navigate them instead. You are assisting in writing the navigation software for one of the robots---the Guru 2000. This robot comes equipped with the latest development in computer technology---a chip that gives the robot psychic powers. This chip, the IC12, provides the robot complete knowledge of the maze, including the locations of the start and target points, before the robot begins its maze run. Armed with this knowledge, you have been asked to write a program to determine the smallest number of moves needed to get from the start point to the target point.

The input consists of a number of problems to solve. Each problem describes a maze defined on a rectangular grid with the origin in the top left corner. Each maze is specified by a line containing 6 integers followed by lines containing the layout of the maze. The 6 integers are: *mazerows, mazecols, startrow, startcol, endrow, endcol* where:

- *mazerows* and *mazecols* give the dimensions of the maze that follows, in the range 1 to 80.
- *startrow* and *startcol* give the starting position (row, column) of the robot. *Startrow* must be in the range 1 to *mazerows*, and *startcol* in the range 1 to *mazecols*.
- *endrow* and *endcol* give the target position (row, column) of the robot. *Endrow* must be in the range 1 to *mazerows*, and *endcol* in the range 1 to *mazecols*.

Following the 6 integers are *mazerows* lines each containing *mazecols* characters specifying the layout of the maze. A '#' character represents a cell in the maze that the robot cannot enter, a '_' character represents a cell that the robot can enter. Note that the robot will be able to enter both the start and target cells. The end of input is indicated by zeroes (0 0 0 0 0 0).

A single line is output for each maze specified in the input. If there is a legal path of moves between the start and target points, the line printed has the form "Shortest path has length" followed by the number of moves involved in the shortest path between the start and target points. Note that a legal move can only be made to one of the 4 cells above, below, left or right of the current cell, and then only if the cell moved to is a '_' cell within the maze (the robot cannot move outside the cells of the maze). If no path connects the starting and target points, print the string "No path".

Sample Input

```
4 5 2 1 1 4
_##__
____#_
____#_
#____
1 20 1 12 1 1
____#_________________
0 0 0 0 0 0
```

Sample Output

```
Shortest path has length 10
No path
```

Problem R Solitaire 50 points

You have received a new solitaire (one player game) for your birthday. Feeling lazy, you have decided to write a program to solve the problems. The game comes with a number of *layouts*. Each layout contains a number of squares, with each square identified by a unique integer greater than or equal to 1. One of the squares is designated the initial square and one of the squares (perhaps the initial square) is designated as the final square. There are a number of lines with arrowheads that connect the squares. Each line goes from one square to another, and is labelled with an upper case letter.

A problem involves a layout and a string of uppercase characters (of length 1 or more) representing an attempted solution. The game starts with a counter in the initial square. The first character in the string is inspected. If there is one line leaving the initial square labelled with that character then the counter can be moved to the square at the other end. If there are several possible lines then you can move the counter along any one of them, although you may need to revisit that decision later. The game continues in this fashion, moving the counter along appropriately marked paths until you either cannot move or you get to the end of the string. If you cannot move (there is no path marked with the next character in the input string) and you have made at least one choice earlier, then you will need to back up to that position and make a different choice. If it is possible to make a sequence of moves that uses all of the input characters and that leaves the counter in the designated target square then you win the game, otherwise you lose

The input contains a number of layouts, and for each layout a number of strings. Each layout begins with two integers giving the numbers of the initial and final squares. This line is followed by one or more lines that specify the connections between squares. Each connection is specified by a letter and two numbers specifying respectively the label and the start and end points of a line. Note that there can be two or more lines from a square labelled with the same character as well as lines from different squares labelled with the same character, but there will never be two lines with the same label and the same initial and final squares. The end of the list of edges is signified by a line that contains '# 0 0'. Following each layout is a list of one or more strings. Each string consists of between 1 and 80 uppercase letters terminated by a '#'. A string that consists of only a '#' signifies the end of input strings for the current layout. The entire input is terminated by a line containing two zeroes (0 0).

Output consists of a single line for each string to test containing either "Won" or "Lost".

Sample Input

```
1 3
A 1 2
A 2 2
B 2 3
B 3 1
B 3 3
# 0 0
ABBB#
AAABBBBBBAABB#
ACBBBBB#
AAAABBAA#
#
1 2
A 1 2
B 1 2
C 2 1
# 0 0
ACBCA#
ACBC#
ACDC#
#
0 0
```

Sample Output

```
Won
Won
Lost
Lost
Won
Lost
Lost
```

Problem S Alien detection 50 points

A group looking for signs of extraterrestrial life has asked you to write some software to assist them in their search. They are convinced that a race of alien super-beings is capable of moving stars, and has been re-arranging star positions so that when viewed from Earth there will be many instances of groups of 4 stars arranged in a perfect square.

They have written software that translates an image from a telescope into a list of coordinates of the positions of all stars present in the image. Your program must take a list of coordinates, and return the number of squares present in the image.

Input consists of a one or more data sets. Each data set begins with a line containing a single integer (N) which is the number of points in the data set. N will be at least 1 and no more than 200 (0 is used to indicate the end of input). Each of the N following lines contains two integer values that are the X and Y coordinates of a single star. Each of these integers will be at least 0 and no more than 1000000.

Output for each data set is a single line containing the string "Squares found = " followed by the number of squares found.

Sample Input

```
4
0 0
1000000 1000000
1000000 0
0 1000000
9
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
0
```

Sample Output

```
Squares found = 1
Squares found = 6
```

Problem V Directions 150 points

Car trialling is a test of navigational and driving skills involving a set of misleading instructions designed to get you from A to B but by a variety of different routes, depending on which traps you fall into. In the real situation instructions can be amended by placing various signs along the route. However are not available in a computer model, hence what follows is only a pale imitation of the real situation.

The following rules are loosely based on real car trialling instructions. **BOLD-TEXT** indicates text as it appears in the instruction (case sensitive) and | separates options of which exactly one must be chosen.

instruction = start | inst.
start = **START ON** where **BETWEEN** numth **AND** numth **AT** num **KMH**
inst = navigational | time-keeping | navigational **AND** time-keeping
navigational = directional | navigational **AND THEN** directional
directional = turn | do
turn = **TURN** direction | **TURN** when direction | **TURN** direction **AT** where
direction = **RIGHT** | **LEFT**
when = **FIRST** | **SECOND** | **THIRD**
do =**CROSS** where | **STOP**
where = num**th** road
road = **St** | **Ave**
time-keeping = record | change
record = **RECORD TIME**
change = cas **TO** num **KMH**
cas = **CHANGE AVERAGE SPEED** | **CAS**
num = digit | digit digit | **100**
digit = **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Note that num is a sequence of digits with no intervening spaces. There will be one or more spaces between items except before a period (.). Check the examples below.

For convenience assume that this trial takes place in an American city which is laid out on a grid with all intersections at right angles. **Streets** run east-west and are numbered from south to north starting from 1 and **Avenues** run north-south and are numbered from west to east starting from 1. The city is bounded to the south and west by InterState highways (notionally 0th Street and 0th Avenue) but these highways are out of bounds to car trialists. City blocks are exactly 1km long. You can assume that the starting and stopping places are car parks with the entrance/exit exactly in the middle of a block. Thus the starting instruction "START ON 23th St BETWEEN 50th AND 51th AT 35 KMH." means that you will start at time zero as you leave the car park heading towards 51st Avenue and attempt to maintain an average speed of 35 km/hour.

Instructions can be incorrect in that the words or syntax can be wrong or they can be semantically invalid, for example, "CROSS 60th St." is syntactically correct but is semantically invalid if you are travelling on 20th St. The input will not contain instructions that are syntactically valid but semantically invalid.

Write a program that will read a START instruction (which will always be correct) and a series of instructions (not including a START instruction), which may not be correct (as above), terminated by a valid STOP instruction. If an instruction is invalid, simply ignore all of it (even any speed changes) and continue with the next instruction. If an instruction includes a change of speed, determine how far you have travelled at the current speed and hence how long it should have taken you and accumulate the total elapsed time. Note that directional instructions can only be obeyed at intersections whereas time-keeping instructions can be obeyed immediately. If a valid instruction contains a RECORD instruction then write your current position (street, avenue) right justified in a field of width 4, a space, your heading (North, East, South, West) left justified in a field of width 5 and total elapsed time, rounded to the nearest second, right justified in a field of width 6. The last instruction will be "STOP". At this point you should specify your stopping position and time in a similar format to a start instruction. See example below.

Sample Input

```
START ON 23th St BETWEEN 50th AND 51th AT 36 KMH.
TURN SECOND LEFT.
TURN LEFT AT 24th St AND CAS TO 60 KMH.
TURN RIGHT AT 50th Ave AND RECORD TIME.
TURN RIGHT AND THEN TURN RIGHT AND THEN TURN SECOND RIGHT.
CROSS 50th St AND THEN CAS TO 40 KMH.
TURN THIRD RIGHT AND CAS TO 36 KMH.
TURN SECOND RIGHT AND RECORD TIME.
TURN SECOND RIGHT AND THEN TURN SECOND LEFT AND THEN STOP.
```

Sample Output

```
  24  50 North   370
  25  48 East    990
STOP ON 23th St BETWEEN 50th AND 51th AFTER 1440.
```

Problem W Backing Up 150 points

This problem has been withdrawn as a result of technical difficulties

Problem X 3D Noughts and Crosses 150 points

The ordinary (two dimensional) game of noughts and crosses is played by two players on a three by three grid. One player is assigned the nought symbol (which we will represent with the "o" character) and the other is assigned the cross symbol (which we will represent with the "x" character). The players take it in turns to write their symbol in an empty cell until one player manages to get three characters in a vertical, horizontal or diagonal line (in which case that player wins) or until the grid is full and no player has three symbols in a line (in which case it is a draw).

By stacking three 3×3 grids one on top of the other, we get a three dimensional version of noughts and crosses played on a 3×3×3 cube. Once again the aim is to be the first to get three symbols in a line. There are many possible lines, including ones that consist of the central cell and any two opposite corners of the cube. However even this game is too easy for some people. These superpeople play with cubes up to 10×10×10 and lines up to the size of the cube.

Your task is to write a program that can determine the state of a game of three-dimensional noughts and crosses given the size of the cube N ($3 \le N \le 10$), the length of a winning line l ($3 \le l \le$ N) and the contents of the N^3 cells. Each cell is either empty (represented by '–') or it contains a nought ('o') or cross ('x'). The state of a game is one of:

1. "Out of turn" if the difference between the number of noughts and the number of crosses is greater than one (for this to have happened the turn sequence must have been violated).
2. "Long line" if there is a line longer than $2l$–1.
3. "Illegal" if there are two or more different lines (the game should have stopped when the first line was completed).
4. "Won by o" or "Won by x" if there is exactly one line of length l or greater.
5. "Draw" if all cells are occupied, but there are no lines.
6. "In progress" if there are no lines but empty cells remain.

If more than one of these conditions is true then report the one with the lowest number.

Input consists of a number of game situations. The first line of each game situation consists of the pair of integers N and l as described above, separated by at least one space. This is followed by N sets of N lines, each line containing N characters. Each group of N lines represents one plane of the cube starting from the bottom. Input is terminated by a line containing two zeroes (0 0).

For each game situation, output the state of the game by printing the appropriate string (of the 7 strings quoted above) on a line by itself.

Sample Input

```
3 3
--x
-o---
---
-x-
-o-
o--
---
x--
4 3
xxxo
ooox
----
----
----
----
----
----
----
----
----
----
----
----
0 0
```

Sample Output

```
Won by x
Illegal
```

Problem Y Cordless Telephones 150 points

Cordless telephones consist of two units — a base unit which connects directly to the telephone network in the normal way and the telephone unit which can be carried around the house and even into the garden. These two units communicate by means of radio waves and each unit has a fixed frequency which is set at the time of purchase. This frequency may need to be changed if you move house, but that will not concern us here. The base unit typically has a reasonably powerful transmitter with a range of possibly several hundred metres whereas the telephone unit has a much more limited range. Because of the larger range of the base unit, it is essential that 'neighbouring' units operate on a different frequency, since otherwise you could listen in on your neighbour's calls and vice versa.

TelCon, the local mobile phone provider, is about to enter the next round of frequency negotiations. They obviously wish to limit the number of frequencies they buy since they have to pay for them, but they have to balance this against the cost of resetting the frequencies of existing units. In an attempt to get a feeling for the problem they have hired you as a consultant to determine some crucial parameters for them. They have selected a sample area 20 km square and determined the positions of all the base units in it; they want to know, for a given number of frequencies k, the radius of the largest circle they can draw around any base unit that will include no more than k–1 other base units.

Write a program that will determine this information.

Input will start with an integer N ($N \leq 60{,}000$) giving the number of sites, followed by N lines containing pairs of integers representing the coordinates of each base unit to the nearest metre. This part of the input will followed by a series of values of k ($5 \leq k \leq 20$) terminated by the integer zero.

Output will be the value of the radius in metres, truncated to a whole number, of the largest circle that can be drawn around any base unit so that no more than k-1 other base units are included.

Sample Input

```
15
99 101
99 100
99 99
100 100
100 101
100 102
101 99
101 100
101 101
110 90
110 100
110 110
120 90
120 100
120 110
5
0
```

Sample Output

```
1
```