

**The Northeast Regional Preliminary Competition
of the
2005-2006 ACM International Collegiate Programming Contest
sponsored by IBM
Western New England College Site, Springfield, MA
October 29, 2005**

Problem #1: Fractions...Continued

A continued fraction of order k is a fraction of the form

$$n_0 + \frac{1}{n_1 + \frac{1}{n_2 + \dots + \frac{1}{n_k}}}$$

where n_i is a positive integer for each $i > 0$. Any rational number has an equivalent expansion as a continued fraction. For example, the fraction $\frac{42}{11}$ can be written as a continued fraction of order 3 as follows:

$$\frac{42}{11} = 3 + \frac{1}{1 + \frac{1}{4 + \frac{1}{2}}}$$

This continued fraction can be described by the order $k = 3$, followed by the integers n_0, n_1, n_2 and n_3 . Thus we describe the expansion of the above continued fraction as $(3; 3, 1, 4, 2)$. In general, the expansion of a continued fraction will be described as

$$(k; n_0, n_1, n_2, \dots, n_k)$$

where k is the order and $n_0, n_1, n_2, \dots, n_k$ are the integers in the expansion of the continued fraction. You must write a program that determines the expansion of a given rational number as a continued fraction. To ensure uniqueness, whenever $k \geq 1$, make $n_k > 1$.

The first line of input will contain the number of rational numbers N ($1 \leq N \leq 20$). Each of the next N lines will contain two integers, the numerator and the denominator, respectively, separated by a space. For each rational number, output the description of the corresponding continued fraction, one description per line.

Sample Input:

```
4
42 11
1 2
3 1
-6 5
```

Sample Output:

```
(3; 3, 1, 4, 2)
(1; 0, 2)
(0; 3)
(2; -2, 1, 4)
```

**The Northeast Regional Preliminary Competition
of the
2005-2006 ACM International Collegiate Programming Contest
sponsored by IBM
Western New England College Site, Springfield, MA
October 29, 2005**

Problem #2: A Smart Mouse

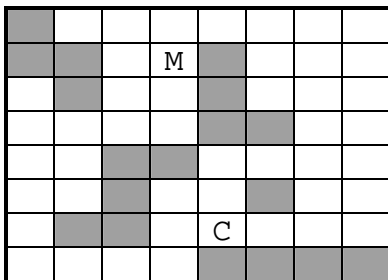
A classical experiment in psychology is the mouse-and-cheese experiment. In this experiment a mouse and a cheese are placed in a maze, and the mouse is observed while it finds its way through the maze until the cheese is found. You are to write a program to find all shortest paths from the mouse to the cheese. (A path is shortest if it consists of a minimum number of moves.) Note that the mouse can move vertically and horizontally, but not diagonally. The maze will be represented by a square matrix, where asterisks (*) are used to represent the walls, blanks are used to represent free paths, M is the initial position of the mouse, and C is the position of the cheese. Your program should read the maze from input, and print the maze with all shortest paths found. Use digits 1, 2, 3, ..., 8, 9, 1, 2, 3, ... to represent each path as shown in the sample output below.

The first line of the input is a positive integer representing the size of the maze. The remaining lines represent the maze, and each line is enclosed in a pair of quotes ("). You may assume that the size of the maze will be 20 or less. Also, assume that there is an invisible wall around the maze, so the mouse cannot step outside of the maze.

Sample Input

```
8
" *      "
" ** M*  "
" *  *   "
"   **   "
"  **    "
" *  *   "
" ** C   "
"   **** "
```

The input represents the following maze:



It has 6 shortest paths:

			1	2	3	4	
			M			5	
						6	
						7	
						8	
						9	
				C	2	1	

			1	2	3		
			M		4		
					5	6	
						7	
						8	
						9	
				C	2	1	

			1	2	3		
			M		4	5	
						6	
						7	
						8	
						9	
				C	2	1	

			1	2	3	4	
			M			5	
						6	
						7	
					1	9	8
					2		
				C			

			1	2	3		
			M		4		
					5	6	
						7	
					1	9	8
					2		
				C			

			1	2	3		
			M		4	5	
						6	
						7	
					1	9	8
					2		
				C			

Your program should display the total number of shortest paths, and combine all shortest paths into one as shown below in the sample output.

Sample Output

6 shortest paths found

```
*--1234-
**-M*45-
-*--*56-
----**7-
--**198-
--*-2*9-
-**-C21-
----*****
```

Note that each empty cell should be represented by a dash ('-') in your output.

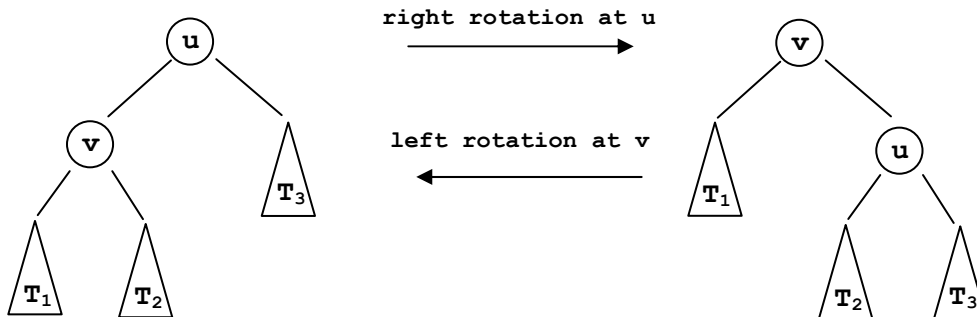
If there is no path from the mouse to the cheese, your program should display the message

No path found

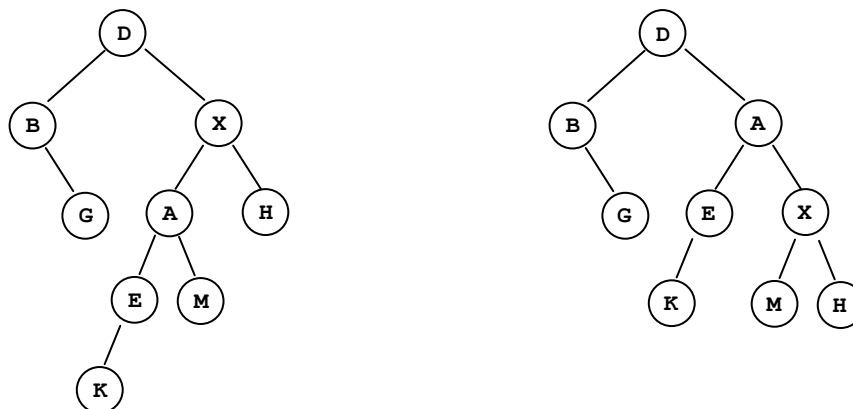
The Northeast Regional Preliminary Competition
of the
2005-2006 ACM International Collegiate Programming Contest
sponsored by IBM
Western New England College Site, Springfield, MA
October 29, 2005

Problem #3: Pivot!

A **binary tree** is a tree data structure in which each node has at most two children. The children are called the **left child** and the **right child**. To improve efficiency of algorithms, it is often desirable to keep a binary tree nearly balanced, that is, keep the heights of the subtrees at any node nearly equal. One common balancing operation is a rotation. The diagram below shows the operation of a **right rotation** at the node u from the binary tree on the left to the binary tree on the right. Similarly, it shows a **left rotation** at the node v from the tree on the right to the tree on the left.



Given the data for two binary trees, you must determine if the second tree can be obtained from the first by either a left rotation or a right rotation. In the example shown below, the binary tree on the right can be obtained from the binary tree on the left by a right rotation at the node with key X.



The first line of the input contains an integer n , $1 \leq n \leq 40$, representing the number of nodes in the first tree. Each of the next n lines represents one node with the following information:

- the node number (between 1 and n)
- the data in the node (a character)
- the node number of the left child or 0 if no left child exists
- the node number of the right child or 0 if no right child exists

There is a single space between two pieces of data. Note that nodes may appear in any order in the input.

The next line of input contains an integer m , where $1 \leq m \leq 40$, representing the number of nodes in the second tree. Each of the next m lines represents the data for the second tree in the same format as listed above.

If the second tree can be obtained from the first tree by a rotation, output "left" or "right" accordingly, along with the node number and data in the first tree at which the rotation occurs. If no such rotation occurs, output "no rotation".

Sample Input:

```
9
5 D 2 1
2 B 0 4
8 K 0 0
9 E 8 0
4 G 0 0
6 H 0 0
1 X 7 6
3 M 0 0
7 A 9 3
9
3 A 7 9
9 X 6 8
2 D 4 3
8 H 0 0
4 B 0 1
6 M 0 0
1 G 0 0
7 E 5 0
5 K 0 0
```

Sample Output:

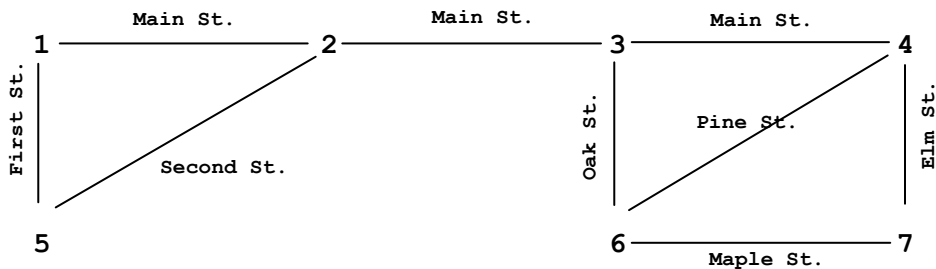
```
right 1 X
```

**The Northeast Regional Preliminary Competition
of the
2005-2006 ACM International Collegiate Programming Contest
sponsored by IBM
Western New England College Site, Springfield, MA
October 29, 2005**

Problem #4: Water Over the Bridge

Suppose you are given a map of roads between various intersections in a city. If a certain road is flooded between 2 intersections (and is thus impassable), are you still able to travel between all pairs of intersections in the city?

For example, consider the following city map. There are 7 intersections, numbered 1, 2, 3, 4, 5, 6, 7, and there are roads between the various intersections. Notice that if the road segment marked Main St. between intersections 2 and 3 was flooded, then there would be no way to travel from any of the intersections marked 1, 2 or 5 to any of the intersections marked 3, 4, 6 or 7.



The first line of input contains the number of intersections, M . The maximum number of intersections will be 20. Each intersection is assigned an integer 1, 2, ..., M .

The next line of input will contain the number of road segments, N . The maximum number of road segments will be 190. The next N lines will contain a description for each road segment. Each road segment is described by a name 20 characters long, followed by the two intersections between which the road segment runs, separated by spaces. The road segment names are not necessarily unique because a road may run between more than one pair of intersections. You may assume that all roads are two-way streets and, thus, the order that the intersections are listed does not matter. You may also assume that, to begin with (prior to any flooding), you are able to travel between any pair of intersections.

Output all road segments and corresponding intersections, for which flooding along the road segment would prevent travel between all pairs of intersections of the city. If there are no such road segments, output "none". If there is more than one such road segment, output the list in alphabetical order. If there is more than one such road segment with the same name, the order of the listing of these road segments is immaterial.

Sample Input:

```
7
9
Main St.      1 2
Main St.      2 3
Main St.      3 4
First St.     1 5
Second St.    2 5
Oak St.       3 6
Pine St.      4 6
Elm St.       4 7
Maple St.     6 7
```

Sample Output:

```
Main St.      2 3
```

Sample Input:

```
6
7
Center St.    1 2
Sprinkle Rd.  2 3
Chester Ct.   3 4
Hagadorn Rd.  4 5
Allen St.     2 4
Center St.    2 5
Center St.    5 6
```

Sample Output:

```
Center St.    1 2
Center St.    5 6
```

Sample Input:

```
4
4
Elmcrest Dr.  1 3
Harwich St.   1 2
Sprague St.   2 4
First Ave.    3 4
```

Sample Output:

```
none
```

**The Northeast Regional Preliminary Competition
of the
2005-2006 ACM International Collegiate Programming Contest
sponsored by IBM
Western New England College Site, Springfield, MA
October 29, 2005**

Problem #5: The Domino Effect

The Post correspondence problem is a well-known problem in theoretical computer science. It can be described as a simple game of manipulating dominoes. Each domino has two strings, one on the top half of the domino and the other on the bottom half, such as

AB
BAC

The game is to select a subset from a given set of dominoes and place them one next to the other so that the top and bottom strings are identical. For example, consider the problem defined by the following three dominoes

AA	BCA	BA
A	C	ABAA

It has a solution for which both the top and bottom contain the string AABAAA:

AA	BA	AA
A	ABAA	A

Note that we assume that there is an unlimited number of dominoes of the same type. So the same domino can be used several times in constructing a solution. It can be proved that this problem is unsolvable. That is, there is no general algorithm that can determine whether an arbitrary Post correspondence system has a solution.

However, if we require that no domino can be used more than once, then the problem becomes solvable. You are to write a program to solve this modified version of the Post correspondence problem.

Input to your program is an unknown number of data sets. Each data set begins with an integer N indicating the number of dominoes in the problem. Each of the next N lines contains two strings that are on the top and the bottom of a domino. Assume that $N \leq 20$ and all strings are 0 - 5 letters long. The first string (string on the top) starts on column 1, and the second string (string on the bottom) starts on column 7. Only uppercase letters are used. Note that a string may be null, and we use a @ to represent a null string (see sample input below). Your program should read these strings, print the data set number, then find and print all solutions of the problem. For each solution, print the dominoes in their correct order used in constructing the solution, followed by 5 blank spaces then the string generated by the solution. Use a comma (,) to separate two domino numbers. Each solution should be printed on a separate line. If a data set has no solution, then print the message "No solution found" to indicate so. Use two blank lines to separate solutions of two different data sets. Continue to read and process each problem until the sentinel line containing a zero is reached.

Sample Input

```
4
BAB  ABAB
AABA BAA
AB   A
B    BAB
4
@    C
AD   AD
B    C
BC   B
2
AB   B
AA   CC
0
```

Sample Output

```
Data set 1:
3, 2, 4    ABAABAB
```

```
Data set 2:
4, 1, 2    BCAD
2, 4, 1    ADBC
4, 1      BC
2         AD
```

```
Data set 3:
No solution
```

Note that you must find all solutions of each problem. When a problem has multiple solutions, the order in which you print them is not important.

**The Northeast Regional Preliminary Competition
of the
2005-2006 ACM International Collegiate Programming Contest
sponsored by IBM
Western New England College Site, Springfield, MA
October 29, 2005**

Problem #6: Meaningful Pretty Printer

Write a pretty printer that formats simple algebraic expressions to fit within a given width without altering the expressions' semantics. Expressions have the following syntax:

```
expression ::= expression spaces operator (spaces | new-line) expression
            | [0-9a-zA-Z]+
operator    ::= [+*/]
spaces      ::= [ ]*
new-line    ::= [\n]
```

Here are two examples:

```
3 +Count/2 * j-   n   - 1

a + b -
c + d
```

When expressions are broken across multiple lines, mathematically, each line has an implicit set of parentheses. For example,

```
a + b -
c + d *
e + f
```

mathematically means

```
(a + b) - (c + d) * (e + f)
```

Like typical algebraic expressions, * and / have a higher precedence than + and -, and operators with the same precedence evaluate in left-to-right order.

Your pretty printer will be given a maximum number of characters a line can hold and a one-line expression. Your pretty printer must convert this one-line expression into a multi-line expression that fits in the given width without changing the expression's semantics. Starting with the first line, your pretty printer will try to fit as much of the expression on the current line as it can before starting a new line. However, it can only break expressions where the above grammar allows (immediately after operators), and only at places where doing so will not change the expression's semantics. A **legal break point** is one where the remaining expression can be formatted according to these same rules such that the meaning of the entire expression is maintained. If no legal breaking point can be found at, or before, the end of the line, then the expression must overflow the line. If an expression

must overflow, minimize the overflow by using the next legal breaking point following the end of the line.

Also, since the input expression may have arbitrary spacing, we'd like your pretty printer to tidy them up. In particular, resulting expressions must have exactly one space before each operator, and exactly one space or new-line (not both) after each operator.

Note, new-lines do not count toward the width of a line. Of course, spaces do.

Input: The first line contains the number of width-expression pairs your program will be given. The subsequent lines each contain a width ($0 \leq \text{width} \leq 70$) and a one-line expression. The length of the one-line expression will be no more than 70 characters.

Output: Print each formatted expression in the same order that they were given, and labeled with "Expression K" where K is the number of input expression starting with 1.

Sample input:

```
3
4 a   +b + c+d *           e
7 a + b + c + d * e
7 a * b - c + d + e / f * g
```

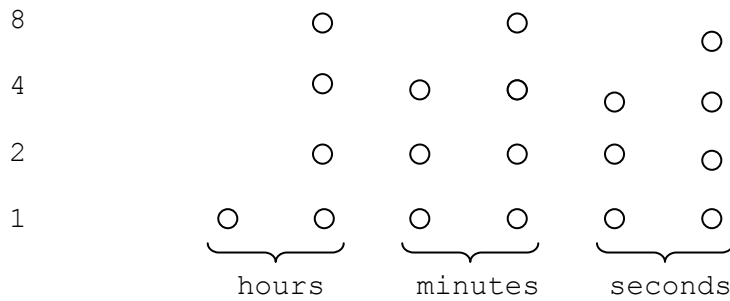
Sample output:

```
Expression 1
a +
b +
c +
d *
e
Expression 2
a + b +
c +
d * e
Expression 3
a * b -
c +
d +
e / f *
g
```

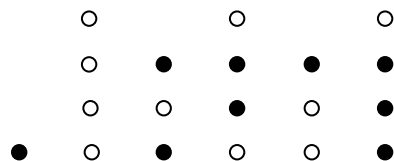
The Northeast Regional Preliminary Competition
of the
2005-2006 ACM International Collegiate Programming Contest
sponsored by IBM
Western New England College Site, Springfield, MA
October 29, 2005

Problem #7: Time for Binary

As an avid computer programmer, you insist that all clocks in your house display each digit of the current time in binary. Your favorite binary clock has a series of lights that illuminate to show the current time. The lights on the clock are in the following formation:



Beginning from the bottom row, the lights stand for place values 1, 2, 4 and 8. Each column of lights represents a different digit in the time. The following example shows the light display for the time 10:56:47.



You must write a program that accepts a time and produces a series of dots as shown in the display. The first line of the input will be the number N of times that will be given ($1 \leq N \leq 50$). Each of the next N lines contains a time in standard 12-hour format (e.g., 10:56:47 or 9:04:13).

The output for each time consists of 4 lines, showing the binary display representing the given time. Use the character '*' to represent a light that is illuminated and the character 'O' for a light that is not illuminated. Output a blank space between characters within a row. Print two blank lines between each group of output lines.

Sample Input:

2

10:56:47

9:04:13

Sample Output:

```
  O   O   O
  O * * * *
  O O * O *
* O * O O *
```

```
  *   O   O
  O O * O O
  O O O O *
O * O O * *
```