Thirteenth Annual University of Oregon Programming Competition

Saturday, April 11, 2009

Supported in part by  and 

Problem Contributors

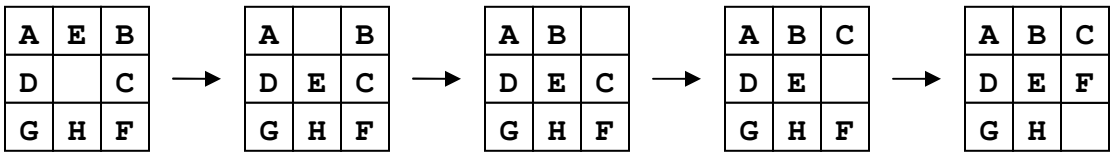
Jim Allen
David Atkins
Peter Boothe
Scott Burich
Michal Young

PUZZLED?

An 8-puzzle is a 3 x 3 grid with 8 tiles (leaving one space unoccupied). Typically the tiles are labeled with the digits 1–8, but any other sequence is possible and our puzzles use the letters A through H. A *solved* 8-puzzle is shown to the right.

A	B	C
D	E	F
G	H	

A *move* in an 8-puzzle consists of sliding a tile horizontally or vertically adjacent to the empty space into the empty space. This puzzle shown can be solved in four moves:



The moves are: move the E down, move the B left, move the C up, move the F up.

For this problem, you must write a program to find a shortest sequence of moves that solves an 8-puzzle. The first line of the program input is a non-negative integer indicating the number of puzzles to be solved. Each puzzle that follows consists of 3 lines of 3 characters each, and then is followed by an empty line. Each of the capital letters 'A' through 'H' appears exactly once in a puzzle and the character '_' (underscore) represents the unoccupied space and also occurs exactly once. Each of the input puzzles can be solved in at most 5 moves. Some can be solved in fewer moves.

Your program must output one line for each puzzle in the input. Each line of output describes a minimum-length solution of the 8-puzzle. (Minimum-length means that, for example, if a puzzle could be solved in 3 moves or in 5 moves, the program must print a solution of length 3 and not 5.) The output for each puzzle is the number of moves required, and the sequence of tiles moved in a solution. For example, the solution to the puzzle described above would be output as: "4 moves: EBCF".

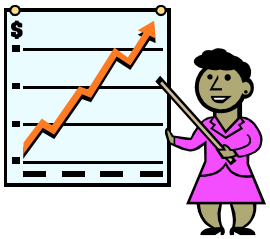
Sample Input

```
5
ABC
DE_
GHF

AB_
DEC
GHF

ABC
_DF
GEH

ABC
E_F
DGH

ABC
DH_
GFE
```

Sample Output

```
1 moves: F
2 moves: CF
3 moves: DEH
4 moves: EDGH
5 moves: EFHEF
```

RECOVERY ACT



Jessica is fourteen years old, and plans to go to college after high school. To that end, her parents have been saving money in the Oregon College Savings Plan where it is invested to grow until Jessica goes to college. Unfortunately, the recession has taken a deep toll on the Plan's investments over the last year, and the value of Jessica's college account has dropped by 50%, from $10,000 to $5,000. Jessica knows that investments go up and down, and hopes the economy will recover. Although the investments dropped by 50%, she hopes that the government stimulus will cause them to rise by 50%. But as she thinks about this, she realizes that rising by 50% will only bring her account value to $7,500, not the $10,000 it was a year ago. In fact, it will take a rise of 100% to bring the current value of $5,000 back to the original value of $10,000.

Since Jessica has several years till she goes to college, there is more time for the value to recover, but she wonders what yearly rate of increase will be required over that period. For example, if the investments grow at an annual rate of 20% over the next four years, then the value would have grown to $6,000 after one year, $7,200 after two years, $8,640 after three years, and $10,368 after four years. Although four years of average growth of 20% is very optimistic, it seems more plausible than a rebound of 100% in one year.

	Beginning	*End*
1 year	$5000	$6000 = $5000 + 20%
2 years	$6000	$7200 = $6000 + 20%
3 years	$7200	$8640 = $7200 + 20%
4 years	$8640	$10368 = $8640 + 20%

In this problem, you are to compute the rate of yearly increase required to restore an account value back to its original value in a given number of years after an initial decrease. The first line of input will be a problem count that specifies how many values you are to process. Each subsequent line will have three numbers: the original value, the percentage decrease, and the number of years over which it can grow to restore the value. The percentage decrease will be less than 100, and the number of years will range from 1 to 100. For each value, your program should output the yearly percentage increase required, accurate to one decimal place.

Sample Input

```
4
10000 50 1
10000 25 1
1000 50 4
1000 50 3
```

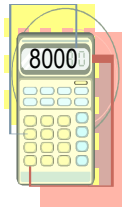
Sample Output

```
100.0%
33.3%
18.9%
26.0%
```

DISAPPEARING FACTS

The factorial of a number N, denoted *N!*, is the product of all the numbers from one through N, inclusive. Factorials are important in combinatorics and in particular, *N!* is the number of permutations of N items. Factorials grow very fast, faster than polynomial or exponential functions, and so *N!* is a very large number, even for relatively small values of N, as the table shows.

N	*N!*
1	1
2	2
3	6
4	24
5	120
10	3,628,800
15	1,307,674,368,000
18	6,402,373,705,728,000
20	2,432,902,008,176,640,000



We would like to build a calculator that computes factorials. Our calculator will be able to handle the computation of large numbers, but the physical display is limited to displaying the least significant digits. As the factorial values get larger, they will end in some zeroes, and we are curious about when this limited display of the least significant digits will turn to all zeroes. For example, if our display had just four digits, then 18! would display as 8000, and 10! would display as four zeroes.

In this problem, you are given a list of numbers. For each number, your program must compute the last non-zero digit in the factorial of the number, and also the number of zeroes that follow.

The first line of input will be a problem count that specifies how many numbers you are to process. Each number will be on a single line and will range from 1 to 10,000. For each of these numbers, your program must output the last non-zero digit and the number of trailing zeroes for the factorial, in the form shown.

Sample Input

```
6
1
2
5
10
20
125
```

Sample Output

```
  1! ends with 1 and 0 zeroes
  2! ends with 2 and 0 zeroes
  5! ends with 2 and 1 zeroes
 10! ends with 8 and 2 zeroes
 20! ends with 4 and 4 zeroes
125! ends with 8 and 31 zeroes
```

CRYPTARITHMS

Jeremy has been appointed entertainment editor of the ACM club newsletter, and he wants to include a cryptarithm with each edition. Cryptarithms are fun mathematical puzzles in which the digits in a valid equation have been replaced with letters.

Consider the example:

```
  SEND
+ MORE
 MONEY
```

Each letter must correspond to a unique digit, and no number in the resulting numerical equation may have an initial digit of zero. Some possible solutions to this example are:

D=7, E=5, M=1, N=6, O=0, R=8, S=9, Y=2	**Correct** since 9567 + 1085 = 10652
D=6, E=3, M=0, N=1, O=8, R=2, S=7, Y=9	**Wrong** since 7316 + 0823 = 08139 may not be written that way (with leading zeros)
D=5, E=0, M=4, N=7, O=6, R=2, S=1, Y=3	**Wrong** since 1075 + 4620 ≠ 46703
D=0, E=9, M=1, N=0, O=1, R=9, S=9, Y=9	**Wrong**, even though 9900 + 1199 = 11099, duplicate assignments are not allowed (E, R, S and Y cannot all be 9)

Jeremy wants you to help him by writing a program that, when given a cryptarithm puzzle, will print out a valid number substitution for that equation. Jeremy intends to use your program to check if his puns are solvable cryptarithms, so if more than one valid number substitution exists, it doesn't matter which one you print out.

The first line of the input will be the number of puzzles to check. Each puzzle will be on a single line. For each puzzle, the output should be the numerical version of the equation, or "Not valid" if no numeric equivalent exists. There will always be three numbers involved, one binary operation (+,-,*, or /), and one equals sign. Spaces may appear anywhere in the input, but do not alter the solution, and there should be no spaces in your output. Lowercase letters are treated as different from their uppercase equivalents.

Sample Input

```
6
binary=zeroes+ones
FOUR is = two * two
RED * RED = MOSCOW
Square + Dance = Dancer
Square - Dance = Dancer
Square / Dance = Dancer
```

Sample Output

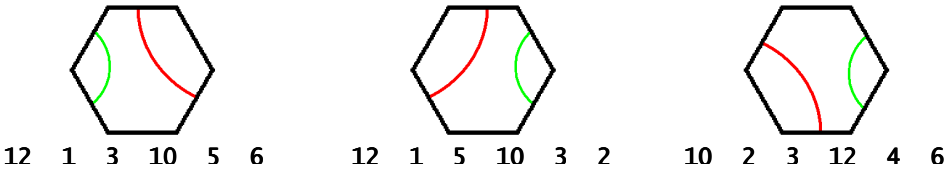
```
701584=698392+3192
321489=567*567
904*904=817216
824163+91573=915736
631708-57428=574280
Not valid
```

BOXED SETS

Pathways is a game that uses hexagonal tiles with lines painted on them. The players must place their tiles so that the lines are continuous. Each tile is made of see-through Plexiglas and has two lines painted on it. The lines never cross, and are in two distinct colors. Each game box may have different combinations of lines and colors. For example, the Ducks vs. Beavers set has only tiles with green and yellow or orange and black; the rainbow set has all the possible combinations of sixteen colors.

The tile manufacturer has screwed up, and each game box in the warehouse may not have a complete set of tiles. However, you have a robot at your disposal that can look through the boxes and identify each tile using an electric eye. The robot cannot determine which set is supposed to be in the box, but you can assume that if the box contains equal counts of each tile type, then it is a complete set. If there are more of some types than others, then it is clearly an incomplete set.

The robot outputs a description of a tile as two triples, where each triple describes one of the lines on the tile. Each triple starts with an integer from 0 to 15 for the color of the line, followed by two numbers indicating the start and end edges of the line. The top edge is numbered 1 and the numbering proceeds clockwise around the tile to 6. Unfortunately, two identical tiles may be coded differently, depending on how the robot picked them up. For example, here are three views and codes for the same tile type:



For this problem, you must write a program that determines if the tiles in a box are a complete set. The first line of input is a number that indicates how many boxes you are to process. Each box starts with a line containing the number of tiles in the box, followed by a line for each tile. Each tile line consists of six non-negative integers which are the two triples described above. For each box, your output should be "complete set" or "incomplete set" depending on whether the box contains the same number of each type of tile.

Sample Input

```
2
6
12 1 3 10 4 6
10 1 3 12 4 6
12 1 5 10 2 4
10 2 4 12 1 5
10 1 4 12 2 3
10 1 4 12 5 6
8
11 4 6 15 1 3
8 2 5 15 1 6
0 1 3 15 5 6
0 2 4 15 1 6
0 2 4 8 1 5
8 1 3 0 4 6
11 1 3 15 4 6
8 1 4 15 2 3
```

Sample Output

```
incomplete set
complete set
```