PLU February 2016 Programming Contest

Advanced Problems

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. Problems will have either no input or will read input from standard input (stdin, cin, System.in -- the keyboard). All output should be to standard output (the monitor).

3. All input is as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Names of Problems

Number	Name
Problem 1	Absolutely
Problem 2	C3PO
Problem 3	Complexity
Problem 4	Complimentary
Problem 5	Egg Drop
Problem 6	Hangman
Problem 7	Judy
Problem 8	Mutint
Problem 9	Neighbor
Problem 10	O Fortuna
Problem 11	Rhonda
Problem 12	Semmy

1. Absolutely

We will call the distance between any two values on the number line the **absolute line distance**. Given two values, find the **absolute line distance** and output it, rounded and formatted to one place of precision.

Input: The first line contains a single positive integer, n, indicating the number of data sets. Each data set is on a separate line and contains two values on the number line, separated by a single space.

Output: The **absolute line distance** between the two points, rounded to the nearest tenth, and print exactly one decimal digit.

Sample input:

```
3
3 5
8.1 -9
-6.4 -18.34
```

Sample output:

```
2.0
17.1
11.9
```

2. C3PO

C3PO is in trouble. To escape his current predicament you must broadcast his ASCII picture to the Resistance.

Input: None

Output: ASCII picture exactly has shown below. Each eye is the lowercase letter `'o'`.

.

Sample output:

```
     /~\
    ( oo|
     _\=/_
   /  _  \
  //|/"\|\\
|| |\"/| ||
++++++++++
```

3. Complexity

Define the complexity of a string to be the number of distinct letters in it. For example, the string **`string`** has complexity 6 and the string **`letter`** has complexity 4.
You like strings which have complexity either 1 or 2. Your friend has given you a string and you want to turn it into a string that you like. You have a magic eraser which will delete one letter from any string. Compute the minimum number of times you will need to use the eraser to turn the string into a string with complexity at most 2.

Input: The first line of input contains a positive integer n. Each of the following n lines contains a single string of at most 100 lowercase ASCII letters (‘a’-‘z’).

Output: For each input string print, on a single line, the minimum number of times you need to use the eraser.

Sample input:

```
5
string
letter
aaabbb
aaabbbccc
uncopyrightable
```

Sample output:

```
4
2
0
3
13
```

4. Complimentary

Two's complement is used to express negative numbers in binary. For example, the value 30 in signed 8-bit binary is 00011110, and the signed 8-bit two's complement representation of -30 is 11100010. An easy way to convert from 00011110 to 11100010 is simply reverse 00011110 to become 11100001, and then add 1, which produces 11100010.

Your job is to read in two positive integers, express them in 8-bit signed binary, then their negative values in two's complement, and then the difference between the two in 8-bit signed binary.

For example, the difference between 30 and 18 is 12, which in 8-bit signed binary is 00001100, or if you subtract 18 – 30, you get -12, which in two's complement (signed 8-bit) is the reverse of 00001100 + 1, or 11110011 + 1, or 11110100.

Input: Several pairs of positive integers X and Y, each pair on one line, with 0<X<=127 and 0<Y<=127. The end of input is signaled with two zeros on the last line.

Output: For each pair of values, five 8-bit signed strings, representing X, Y, -X, -Y, and X-Y, with a blank line following each output set.

Sample input:

```
30 18
18 30
100 50
0 0
```

Sample output:

```
30 = 00011110
18 = 00010010
-30 = 11100010
-18 = 11101110
12 = 00001100

18 = 00010010
30 = 00011110
-18 = 11101110
-30 = 11100010
-12 = 11110100

100 = 01100100
50 = 00110010
-100 = 10011100
-50 = 11001110
50 = 00110010
```

5. Egg Drop

There is a classic riddle where you are given two eggs and a k-floor building and you want to know the highest floor from which you can drop the egg and not have it break.
It turns out that you have stumbled upon some logs detailing someone trying this experiment! The logs contain a series of floor numbers as well as the results of dropping the egg on those floors. You need to compute two quantities -- the lowest floor that you can drop the egg from where the egg could break, and the highest floor that you can drop the egg from where the egg might not break.
You know that the egg will not break if dropped from floor 1, and will break if dropped from floor k. You also know that the results of the experiment are consistent, so if an egg did not break from floor x, it will not break on any lower floors, and if an egg did break from floor y, it will break on all higher floors.

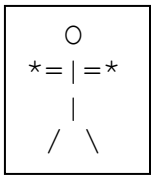
Input: The input is comprised of data sets. Each set starts with a line that contains two space-separated integers n and k ($1 \leq n \leq 100$, $3 \leq k \leq 100$), the number of egg drops and the number of floors of the building, respectively. The next n lines contain a floor number and the result of the egg drop, separated by a single space. The floor number will be between 1 and k, and the result will be either SAFE or BROKEN. The end of input is signaled with two zeros on the last line.

Output: For each data set print, on a single line, two integers separated by a single space. The first integer should be the number of the lowest floor from which you can drop the egg and it could break and still be consistent with the results. The second integer should be the number of the highest floor from which you can drop the egg and it might not break.

Sample input :	**Sample output :**
2 10 4 SAFE 7 BROKEN 3 5 2 SAFE 4 SAFE 3 SAFE 4 3 2 BROKEN 2 BROKEN 1 SAFE 3 BROKEN 0 0	5 6 5 4 2 1

6. HangMan

In the classic blackboard game HangMan you are given a word and attempt to guess the word by choosing a series of letters. In this program, you must determine if the provided letters are contained in the word prior to a body figure being drawn. One body part is added for each letter not contained in the word. The body figure below consists of the head shown by the letter "O", two body parts "|", 2 arms shown by equal signs "=", 2 hands shown by asterisks "*", and legs shown using a forward slash "/"and a back slash "\", 9 parts in all, drawn in order, top to bottom - from left to right, as shown below. Once all the letters of the word are found, the game ends and no further letters are read. The game also ends once the figure is completed. If no body parts are drawn, print the word “SAFE”.



Input: The first line contains a single positive integer, k, indicating the number of data sets. Each data set is on one line, and consists of a single word S (all capital letters), an integer N, followed by N distinct capital letters.

Output: The word and resulting hangman figure, according to the series of guesses provided by the letters. The body parts MUST align as shown, with the "*" on the left edge of the screen, and all other parts aligned accordingly. Exactly one blank line must follow each output.

Sample input:

```
4
APLUS 8 A S E U G P I J
UIL 9 A B C D E F G H J
SALTLICK 9 E F G H I J K M N
GOOGLE 4 G O L E
```

Sample output:

```
APLUS
  O
+=|

UIL
  O
+=|=+
  |
 / \

SALTLICK
  O
+=|=+
  |

GOOGLE
SAFE
```

7. Judy

Judy, the office robot, communicates only in numbers. To understand “her”, Jeff, your immediate supervisor, has asked you to write a translator. To play a prank on Jeff, especially since he did the same to you earlier in the week, you decide to translate Judy’s messages into a simple form of Pig Latin.

Judy only speaks in numbers that translate into alphabet characters according to the ASCII standard. Sometimes “she” hiccups and gives a number that does not represent a letter. You decide to represent these hiccups with a '-' (dash) and hopefully Jeff can figure it out by context. The numbers may indicate upper or lower case letters, but Jeff only wants the translated message in lowercase.

In this simple form of Pig Latin, the first character of the word is placed at the end translated word, with an ‘ay’ added to the end of the string. There are more complex aspects of Pig Latin you could include, but you decide to keep it simple for now.

Input: The first line of input contains a single positive integer, n, indication the number of data sets. Each data set is comprised of a single line representing one word of “Judy” speak. The values on one line are separated by spaces.

Output: The “Judy speak” lowercased, then translated to Pig Latin. Any value that does not translate to an alphabet character should be represented as a '-' (dash).

Sample Input:

```
3
84 104 101
81 85 7300 67 75
102 111 120
```

Sample Output:

```
hetay
u-ckqay
oxfay
```

8. Mutint

A “Mutint” is an integer M that is changed according to certain criteria, such as in this problem. Given a positive integer, change M according to the following rules.

1. Find the leftmost largest digit D of M.
2. If D is odd, change it to a zero.
3. If D is even, add 4 to that digit. If the sum exceeds 9, change D to the one’s place of the sum.

Input: Several integers, each on one line. The end of input is signaled with a zero on the last line. All integers, except the last integer, are positive.

Output: M, according to the rules above. M cannot have leading zeros.

Sample input:

```
132
1421
18234
923
0
```

Sample output:

```
102
1821
12234
23
```
(note: 023 is not correct)

9. Neighbor

Two fractions are considered “neighbors” if the numerator of the reduced positive difference between the two fractions is 1. For example, the fractions 1/11 and 1/12 subtract to be 1/132, and are therefore “neighbors”. However, 2/5 and 4/5 differ by 2/5, and are not neighbors. Determine and output either the “neighbor” fraction value, or output “NOT NEIGHBORS”.

Input: The first line of input contains a single positive integer, n, indication the number of data sets. Each data set is comprised of the positive integer values for two fractions, all on one line, in the order numerator, denominator, numerator, denominator, separated by single spaces.

Output: The positive “neighbor” difference between the two fractions or the phrase “NOT NEIGHBORS”.

Sample input:

```
3
1 11 1 12
20 19 19 19
2 5 4 5
```

Sample output:

```
1/132
1/19
NOT NEIGHBORS
```

10. O Fortuna

In mathematics there are numbers that are fortunate, and some less fortunate. A fortunate number is defined as Q – P, where P is the product of the first N prime numbers (where N>1), and Q is the smallest prime greater than P+1. For example, if N = 2, P = 2*3 (the first two prime numbers), or 6. Q would be the smallest prime greater than 7, which is 11. Q – P is therefore 11 – 6, which is 5, another prime number. **It has been conjectured by some mathematicians that this Fortunate number value will always be prime.**

Likewise, a less fortunate number is defined as P – Q, using the same above scenario, except that Q is the largest prime less than P-1. Therefore, with N = 2, P again is 6, which makes Q = 3 (the largest prime less than 5), and so the LESS fortunate number in this case is 3. **A similar conjecture for LESS fortunate numbers also asserts that they will always be prime.**

Input: A row of integer values N in the range 1 < N <= 14.

Output: The original input value N, followed by the fortunate and less fortunate number, as described above, and shown below, with a single space separating each output item.

Sample input:

```
2 3 4
```

Sample output:

```
N = 2 FORTUNATE = 5 LESS = 3
N = 3 FORTUNATE = 7 LESS = 7
N = 4 FORTUNATE = 13 LESS = 11
```

11. Rhonda

Rhonda is printing processor chips, which can be made from fabricated components inscribed onto thin layers. As she prints the chips, Rhonda stacks two or more layers together to create a final chip made up of multiple layers.

She has several unique layers that can be combined into various numbers of layers, which allows her to create a variety of chips. Each layered component is represented by a 10x10 grid of integers, with each cell containing an integer ranging from 0 to 9. The final chip is also 10x10 grid, with each cell being the sum of all the cells in that position for the layers that were used to create the chip.

For example, if Rhonda has fabricated 3 layers:

```
0123456789          1111111111          0011001100
0123456789          1111111111          1100110011
0123456789          1111111111          0011001100
0123456789          1111111111          1100110011
0123456789          1111111111          0011001100
0123456789          1111111111          1100110011
0123456789          1111111111          0011001100
0123456789          1111111111          1100110011
0123456789          1111111111          0011001100
0123456789          1111111111          1100110011
```

and decides to create a final chip that combines only the **first** and **second layers**, the resulting chip is represented by the 10x10 grid below:

```
01 02 03 04 05 06 07 08 09 10
01 02 03 04 05 06 07 08 09 10
01 02 03 04 05 06 07 08 09 10
01 02 03 04 05 06 07 08 09 10
01 02 03 04 05 06 07 08 09 10
01 02 03 04 05 06 07 08 09 10
01 02 03 04 05 06 07 08 09 10
01 02 03 04 05 06 07 08 09 10
01 02 03 04 05 06 07 08 09 10
01 02 03 04 05 06 07 08 09 10
```

Each layer is indexed by the order given (starting at zero). The data set for this chip would be:
`0 1`
since this chip is made by combining the first and second layer in the order given.

Input: The first line of the input will an integer, `i`, representing the number of unique layers Rhonda has fabricated. This is followed by `i` sets of `10x10` grid of integers, each followed by a new line, representing the `i` fabricated layers. The next line will be a single positive integer, n, indication the number of data sets that follow. Each data set is a list of integers, all on one line, separated by spaces, which represent the layers added to create a chip.

Output: A `10x10` integer grid with each of the `10x10` cells represented by 2 digits and a space, the whole grid representing the final summed chip.

Assumptions: The cell values of each individual layer will be in the range 0-9, and the cell values of the final chip combined chips will in the range 0-99. All original fabricated layers will be unique.

Sample input and output on next page.

Rhonda (continued)

Sample Input:

```
3
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101

0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789

7386459728
3748244739
8420475820
1938475283
3847520734
5826752039
4230489329
4594890835
2188305727
4320834420

3
0 1
0 1 2
0 0 0 0 0 0 0 0 0 0
```

Sample Output:

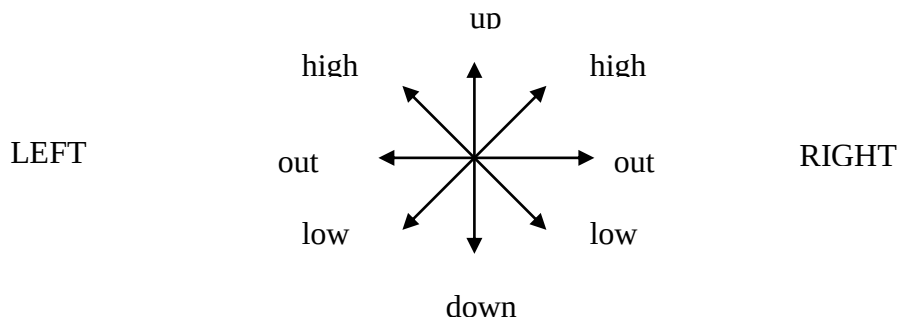
```
00 02 02 04 04 06 06 08 08 10
00 02 02 04 04 06 06 08 08 10
00 02 02 04 04 06 06 08 08 10
00 02 02 04 04 06 06 08 08 10
00 02 02 04 04 06 06 08 08 10
00 02 02 04 04 06 06 08 08 10
00 02 02 04 04 06 06 08 08 10
00 02 02 04 04 06 06 08 08 10
00 02 02 04 04 06 06 08 08 10

07 05 10 10 08 11 15 15 10 18
03 09 06 12 06 10 10 15 11 19
08 06 04 04 08 13 11 16 10 10
01 11 05 12 08 13 11 10 16 13
03 10 06 11 09 08 06 15 11 14
05 10 04 10 11 11 08 08 11 19
04 04 05 04 08 14 15 11 10 19
04 07 11 08 12 15 06 16 11 15
02 03 10 12 07 06 11 15 10 17
04 05 04 04 12 09 10 12 10 10

00 10 00 10 00 10 00 10 00 10
00 10 00 10 00 10 00 10 00 10
00 10 00 10 00 10 00 10 00 10
00 10 00 10 00 10 00 10 00 10
00 10 00 10 00 10 00 10 00 10
00 10 00 10 00 10 00 10 00 10
00 10 00 10 00 10 00 10 00 10
00 10 00 10 00 10 00 10 00 10
00 10 00 10 00 10 00 10 00 10
```

12. Semmy

In the semaphore signaling system, two flags are held, one in each hand, with arms extended, in various positions representing the letters of the alphabet. The pattern resembles a compass rose divided into eight positions: up (U), down (D), out (O), high (H) and low (L), for each hand. The left hand signal is always read first. Additionally, six letters require a hand to be brought across the body so that both flags are on the same side. As an example the letter H has the left hand across the body and held low (AL).



ALPHA	LEFT	RIGHT	ALPHA	LEFT	RIGHT
A	**D**	**L**	N	**L**	**L**
B	**D**	**O**	O	**AH**	**O**
C	**D**	**H**	P	**U**	**O**
D	**D**	**U**	Q	**H**	**O**
E	**H**	**D**	R	**O**	**O**
F	**O**	**D**	S	**L**	**O**
G	**L**	**D**	T	**U**	**H**
H	**AL**	**O**	U	**H**	**H**
I	**AL**	**U**	V	**L**	**U**
J	**O**	**U**	W	**O**	**AH**
K	**U**	**L**	X	**L**	**AH**
L	**H**	**L**	Y	**O**	**H**
M	**O**	**L**	Z	**O**	**AL**

Input: A data file that contains all letter/signal combinations on the first 26 lines, followed by two sets of data, each starting with an integer N. The first group contains N coded expressions that represent words or phrases, and the second group contains N English words or phrases. In the first group, a # indicates a space between words.

Output: Decode the first group of coded signal expressions into the English word or phrase, and encode the second group into the appropriate signal expression, with any spaces between words represented by the # sign.

Sample input:

```
ADL          EHD          IALU         MOL          QHO          UHH          YOH
BDO          FOD          JOU          NLL          ROO          VLU          ZOAL
CDH          GLD          KUL          OAHO         SLO          WOAH
DDU          HALO         LHL          PUO          TUH          XLAH
2
DLUOHLHHLO
HHALUHL#OOAHODHULLO
2
JAVA JIVE
FUN
```

Sample output:

```
APLUS
UIL ROCKS
OUDLLUDL#OUALULUHD
ODHHLL
```