THE CODERS CUP

JUNIOR ROUND

QUESTION SET

Competition Rules:

Participation Guidelines:

- The Freshmen Coders Cup round spans for 1 hour. If you have completed the problem set before the allocated time, you may leave the competition room quietly, but inform the coordinator must.
- If you leave the room, you cannot return.
- You can discuss only with your team mates. If you discuss with anyone else, both team will be disqualified from the competition.

Submission Guidelines:

- Find sample inputs from net-storage.
- The problem submission will be through PC^2
- Clarifications to any problem can be obtained using PC^2. No in room managers will be responsible for problems/confusions in problem set given.
- You are allowed to use language
    - C, C++, C#.NET, JAVA.
- IDEs allowed are:
    - Bloodshed Dev C++ for C and C++
    - Visual Studio 2008 or 2010 for C#.NET
    - NetBeans 6.8 for JAVA.
- Make console projects for all afore mentioned IDEs.
- Show output on console; don't write on a text file.
- Do not prompt for input from console in the program.
- Remove system ("pause")/getch()/package inclusion statements from your choice before submitting.

Additional Guidelines:

- The solution will be judged by multiple input files and execution time.
- The decision of judge will stand unchallenged.
- Books, manuals, and any sort of guide materials are not allowed.
- Your team can be dis-qualified, if found hard coding for solutions.
- Your team can be dis-qualified, if found using internet.
- Your team can be dis-qualified, if found unfair in anyway.

Note: Save your work continuously, ACM NUCES is not responsible for any loss of work due to power failure or any other reason.

Question 1:

WORD PUZZLE

Remember those grids of random letters with words hidden in them? As kids, we all used to spend hours finding the hidden words in them. Well, now you have a chance to speed up that task by writing a program for it. Words may be forwards or backwards in a straight line in any direction (vertical, horizontal, diagonal).

INPUT:

The input starts with a single integer T indicating the number of test cases. T test-cases follow after this line. Each test case starts with an integer N, defining the dimensions of the grid of letters. The next N lines will each have a string, N characters long, thus making an NxN grid. After this is an integer W which tells the length of the word list. The next W lines will have a single word each.

OUTPUT:

For each word in each test case, output the starting point and ending point of the word. Each point should be printed as a pair of integers where first integer is row and second is column. Grid is numbered 0 to N left to Right and top to bottom. Be sure to format it EXACTLY like shown in the sample output.

Sample:

Input	Output
1 5 ABDLF ACAYA BTAFS OARTT XYQWT 3 BOX FAST CAT	Case #1: BOX 2,0 to 4,0 Case #1: FAST 0,4 to 3,4 Case #1: CAT 1,1 to 3,3

Question 2:

TEXTING

We all love texting. Even though most of us have smart phones now that use QWERTY keyboards, we can never really forget the classical numpad typing. In memory of that now fading, brilliant design, this task requires you to write a program that converts key presses on a mobile keypad into text.

INPUT:

The input starts with a single integer T indicating the number of test cases. T test-cases follow after this line. Each test consists of a single line containing a string where each character represents a keypress. A '-' character is used to indicate a pause to distinguish between 66 = N and 6-6 = MM. Following the Nokia convention, a space is placed using the 0.

OUTPUT:

For each test case, output the string formed from the key presses in CAPS. Be sure to format it EXACTLY like shown in the sample output

Sample:

Input	Output
3	Case #1: ACM
2-2226	Case #2: FAST
333277778	Case #3: CODERS CUP
222666333777-77770222887	

Question 3:

TEXTING T9

We all love texting. Even though most of us have smart phones now that use QWERTY keyboards, we can never really forget the classical numpad typing. One form of numpad typing was dictionary mode (aka T9) where the cellphone guessed what word you wanted to type but matching your key presses against a set of words called a dictionary.

This task requires you to create that system. Given a dictionary of words, you have to output text against each numeric input.

INPUT:

The input starts with a single integer W indicating the number of words in the dictionary. Next W lines contain one word each. All test-cases will use this dictionary. The next line will have a single integer T indicating the number of test cases. T test-cases follow after this line. Each test consists of a single line containing a string where each character represents a keypress. Following the Nokia convention, a space is placed using the 0. Also by that convention, where the same key presses results in different words, the user presses * to select next word. Assume same key-press words are in the same order as they occur in the dictionary.

OUTPUT:

For each test case, output the string formed from the key presses in CAPS. Be sure to format it EXACTLY like shown in the sample output

Input	Output
4	Case #1: HOUSE
HOUSE	Case #2: BAT BALL
CAT	
BAT	
BALL	
2	
46873	
228*02255	

Question 4:

CAT IN THE HAT

A clever cat walks into a messy room which he needs to clean. Instead of doing the work alone, it decides to have its helper cats do the work. It keeps its (smaller) helper cats inside its hat. Each helper cat also has helper cats in its own hat, and so on. Eventually, the cats reach a smallest size. These smallest cats have no additional cats in their hats. These unfortunate smallest cats have to do the cleaning.

The number of cats inside each (non-smallest) cat's hat is a constant, N. The height of these cats-in-a-hat is tex2html_wrap_inline35 times the height of the cat whose hat they are in.

The smallest cats are of height one; these are the cats that get the work done. All heights are positive integers.

Given the height of the initial cat and the number of worker cats (of height one), find the number of cats that are not doing any work (cats of height greater than one) and also determine the sum of all the cats' heights (the height of a stack of all cats standing one on top of another).

INPUT:

The input consists of a sequence of cat-in-hat specifications. Each specification is a single line consisting of two positive integers, separated by white space. The first integer is the height of the initial cat, and the second integer is the number of worker cats.

A pair of 0's on a line indicates the end of input.

OUTPUT:

For each input line (cat-in-hat specification), print the number of cats that are not working, followed by a space, followed by the height of the stack of cats. There should be one output line for each input line other than the ``0 0'' that terminates input.

Input	Output
216 125	31 671
5764801 1679616	335923 30275911
0 0	