Árboles y Grafos, 2024-2

Para entregar el lunes 5 de agosto de 2024
A las 23:59 en la arena de programación

Instrucciones para la entrega

- Para esta tarea y todas las tareas futuras en la arena de programación, la entrega de soluciones es *individual*. Por favor escriba claramente su nombre, código de estudiante y sección en cada archivo de código (a modo de comentario). Adicionalmente, cite cualquier fuente de información que utilizó. Los códigos fuente que suba a la arena de programación deben der de su completa autoría.
- En cada problema debe leer los datos de entrada de la forma en la que se indica en el enunciado y debe imprimir los resultados con el formato allí indicado. No debe agregar mensajes ni agregar o eliminar datos en el proceso de lectura. La omisión de esta indicación puede generar que su programa no sea aceptado en la arena de programación.
- Puede resolver los ejercicios en C/C++ y Python. Sin embargo, deben haber por los menos soluciones a dos problemas en cada lenguaje.
- Debe enviar sus soluciones a través de la arena. Antes de subir sus soluciones asegurese de realizar pruebas con los casos de pruebas proporcionados para verificar que el programa finalice y no se quede en un ciclo infinito.
- El primer criterío de evaluación será la aceptación del problema en la arena cumpliendo los requisitos indicados en los enunciados de los ejercicios y en este documento. El segundo criterio de evaluación será la complejidad computacional de la solución. Además, se tendrán en cuenta aspectos de estilo como no usar `break` ni `continue` y que las funciones deben tener únicamente una instrucción `return` que debe estar en la última línea.

Problemas prácticos

Hay cinco problemas prácticos cuyos enunciados aparecen a partir de la siguiente página. Debe resolver 4 de los 5 problemas. El problema restante se calificará como bonificación.

A - Problem A

Source file name: `accordian.cpp`
Time limit: 1 second

You are to simulate the playing of games of "Accordian" patience, the rules for which are as follows:

Deal cards one by one in a row from left to right, not overlapping. Whenever the card matches its immediate neighbour on the left, or matches the third card to the left, it may be moved onto that card. Cards match if they are of the same suit or same rank. After making a move, look to see if it has made additional moves possible. Only the top card of each pile may be moved at any given time. Gaps between piles should be closed up as soon as they appear by moving all piles on the right of the gap one position to the left. Deal out the whole pack, combining cards towards the left whenever possible. The game is won if the pack is reduced to a single pile.

Situations can arise where more than one play is possible. Where two cards may be moved, you should adopt the strategy of always moving the leftmost card possible. Where a card may be moved either one position to the left or three positions to the left, move it three positions.

Input

Input data to the program specifies the order in which cards are dealt from the pack. The input contains pairs of lines, each line containing 26 cards separated by single space characters. The final line of the input file contains a `'#'` as its first character. 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 input must be read from standard input.

Output

One line of output must be produced for each pair of lines (that between them describe a pack of 52 cards) in the input. Each line of output shows the number of cards in each of the piles remaining after playing "Accordian patience" with the pack of cards as described by the corresponding pairs of input lines.

The output must be written to standard output.

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

Sample Output

```
6 piles remaining: 40 8 1 1 1 1
1 pile remaining: 52
```

B - Problem B

Source file name: `entropy.cpp`
Time limit: 1 second

Claude Shannon, mathematician and computer scientist, born on April 30 1916, and died on February 24 2001, was the one who create the mathematical foundations which laid down the general rules of modern information theory. In his fundamental paper of 1948, *A Mathematical Theory of Communication*, a measure of the uncertainty associated with a random memoryless source, called *entropy*, is proposed. Here we are interested in the use of the entropy concept to analyze texts at the level of its words variety.

We define the entropy of a text T, with λ words and n different ones, by the formula

$$E_t(p_1,\ldots,p_n) = \frac{1}{\lambda}\sum_{i=1}^{n} p_i(log_{10}(\lambda) - log_{10}(p_i))$$

where p_i, $i = 1,\ldots,n$, is the frequency of each i-word in the text T, that is, p_i is the number of times that the i-word happens to occur in the given text. If we consider that a text of length λ (a text with λ words) is as much richer as much larger is the number n of different words and, among the texts with the same number λ of words and the same number n of different words, is richer the one where the words have less variation in frequency, one can easily conclude that the entropy is indeed a very useful measure to compare the richness of two or more texts. To compare texts with different number of words λ, we introduce a kind of "relative entropy" E_{rel}, defined as the quotient between the entropy E_T of the text and the maximum entropy E_{max}, and multiplying by 100 if one wants a percentage:

$$E_{rel} = \frac{E_T}{E_{max}} \times 100$$

The maximum entropy E_{max} is just the entropy of a text with the same number λ of words and in which each word occurs exactly once (i.e., $n := \lambda$, $p_i := 1$):

$$E_{max} = \frac{1}{\lambda}\sum_{i=1}^{\lambda} 1 \cdot (log_{10}(\lambda) - log_{10}(1)) = log_{10}(\lambda)$$

Given a text T, write a program that computes the total number λ of words in T, the entropy E_T of the text, and its relative entropy E_{rel}. In order to determine the required numbers, your program must be case insensitive (for example, words like "House", "house" or "HOUSE" must be considered to be the same). Also, in the context of this program, a word is a consecutive sequence of characters different of the punctuation marks `, . : ; ! ? " ( )` as well as spaces, tabs and newlines (`'\n'`).

Words with only one letter are to be considered.

Input

The input contains several texts T, each one necessarily with more than one word ($\lambda > 1$). You can assume that the maximum length of the words is 20 characters long and that a single text does not have more than 100,000 words.

A line containing only `'****END_OF_TEXT***'` marks the end of each text, and a line containing `'****END_OF_INPUT****'` marks the end of input. You can be certain that these reserved words will not appear inside a text. Besides thats, everything can appear on a text, including blank lines.

The input must be read from standard input.

Output

In the output write one line for each test, each one containing three numbers: the first with the total number λ of words in T; the second with the text entropy E_T rounded to one decimal digit; and the last one with the relative entropy E_{rel}, in percentage, and rounded to be an integer.

Note: The sample below is from Barbra Streisand, *Memory* (first two verses).

The output must be written to standard output.

Sample Input

```
Midnight, not a sound from the pavement
Has the moon lost her memory?
She is smiling alone
In the lamplight, the withered leaves collect at my feet
And the wind begins to moan
****END_OF_TEXT****
Memory, all alone in the moonlight
I can dream of the old days
Life was beautiful then
I remember the time I knew what happiness was
Let the memory live again
****END_OF_TEXT****
****END_OF_INPUT****
```

Sample Output

```
33 1.4 93
31 1.3 89
```

C - Problem C

Source file name: `expressions.cpp`
Time limit: 1 second

Arithmetic expressions are usually written with the operators in between the two operands (which is called infix notation). For example, $(x + y) * (z - w)$ is an arithmetic expression in infix notation. However, it is easier to write a program to evaluate an expression if the expression is written in postfix notation (also known as reverse polish notation). In postfix notation, an operator is written behind its two operands, which may be expressions themselves. For example, $'xy + zw - *'$ is a postfix notation of the arithmetic expression given above. Note that in this case parentheses are not required.

To evaluate an expression written in postfix notation, an algorithm operating on a stack can be used. A stack is a data structure which supports two operations:

1. **push**: a number is inserted at the top of the stack.
2. **pop**: the number from the top of the stack is taken out.

During the evaluation, we process the expression from left to right. If we encounter a number, we push it onto the stack. If we encounter an operator, we pop the first two numbers from the stack, apply the operator on them, and push the result back onto the stack. More specifically, the following pseudocode shows how to handle the case when we encounter an operator `'O'`:

```
a := pop();

b := pop();

push(b O a);
```

The result of the expression will be left as the only number on the stack.

Now imagine that we use a queue instead of the stack. A queue also has a push and pop operation, but their meaning is different:

1. **push**: a number is inserted at the end of the queue.
2. **pop**: the number from the front of the queue is taken out of the queue.

Can you rewrite the given expression such that the result of the algorithm using the queue is the same as the result of the original expression evaluated using the algorithm with the stack?

Input

The first line of the input contains a number T.The following T lines each contain one expression in postfix notation. Arithmetic operators are represented by uppercase letters, numbers are represented by lowercase letters. You may assume that the length of each expression is less than 10000 characters.

The input must be read from standard input.

Output

For each given expression, print the expression with the equivalent result when using the algorithm with the queue instead of the stack. To make the solution unique, you are not allowed to assume that the operators are associative or commutative.

The output must be written to standard output.

Sample Input	**Sample Output**
`2`	`wzyxIPM`
`xyPzwIM`	`gfCecbDdAaEBF`
`abcABdefgCDEF`	

D - Problem D

Source file name: `minning.cpp`
Time limit: 1 second

A mining base needs to build some robots to collect at least 10000 units of resource. Each robot will start from the base, reach the diggings in S minutes, work for W minutes, and then take C units of resource back to the base in S minutes.

To speed up this procedure, K robots will be built at the base. It takes M minutes to produce one robot. A robot will be set to start working immediately after it is built, and producing the next robot will be on line right after. This procedure continues untill all the robots are built.

Due to the limitation of the mining equipments, there can be only one robot digging at the working area. That is, it is only after the currently working robot finishes its collecting work and starts getting back to the base that the next robot can work at the diggings.

Now it is your job to write a program to simulate this procedure, and find out how many minutes it will take to collect at least 10000 units of resource.

Input

There are several lines of input. Each line contains a test case which consists of 5 integers, namely S, W, C, K, and M.

The input must be read from standard input.

Output

For each test case, you are asked to output an integer t, which is the number of minutes taken to collect at least 10000 units of resource.

The output must be written to standard output.

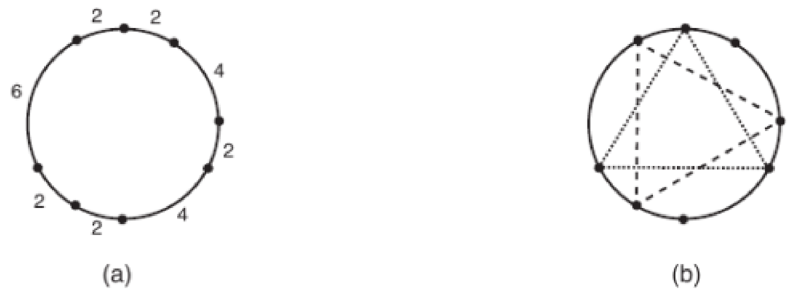
Sample Input	**Sample Output**
`10 20 10 1 5`	`40005`

E - Problem E

Source file name: `triangles.cpp`
Time limit: 1 second

You will be given N points on a circle. You must write a program to determine how many distinct equilateral triangles can be constructed using the given points as vertices.

The figure below illustrates an example: (a) shows a set of points, determined by the lengths of the circular arcs that have adjacent points as extremes; and (b) shows the two triangles which can be built with these points.



Input

The input contains several test cases. The first line of a test case contains an integer N, the number of points given. The second line contains N integers X_i, representing the lengths of the circular arcs between two consecutive points in the circle: for $1 \leq i \leq (N-1)$, X_i represents the length of the arc between between points i and $i+1$; X_N represents the length of the arc between points N and 1.

The input must be read from standard input.

Output

For each test case your program must output a single line, containing a single integer, the number of distinct equilateral triangles that can be constructed using the given points as vertices.

Restrictions

- $3 \leq N \leq 10^5$
- $1 \leq X_i \leq 10^3$, for $1 \leq i \leq N$

The output must be written to standard output.

Sample Input	Sample Output

```
8
4 2 4 2 2 6 2 2
6
3 4 2 1 5 3
```
	```
	2
	1
	```