TecnoMate 2017
COMPETENCIA DE PROGRAMACIÓN

# Nivel “AEDD”
# Problemas

*13 de Octubre de 2017*

## Problema 1: Máquina de Café

El nuevo edificio de los Trescientos Matemáticos Tecnológicos (TMT) tiene 3 pisos. En ciertas épocas del año, los funcionarios del TMT beben mucho café. Por ello, la presidencia del TMT decidió regalar a los funcionarios una nueva máquina de expreso. Esta máquina debe ser instalada en uno de los 3 pisos, pero la instalación debe ser hecha de forma que las personas no pierdan mucho tiempo subiendo y bajando escaleras.

Cada empleado del TMT bebe 1 café expreso por día. Cada uno necesita ir del piso donde trabaja hasta el piso donde está la máquina y volver a su puesto de trabajo. Todo funcionario tarda 1 minuto para subir o bajar un piso. Como el TMT se preocupa mucho por la eficiencia, quiere posicionar la máquina para minimizar el tiempo total gastado subiendo y bajando escaleras.

Tu tarea es ayudar a la dirección a posicionar la máquina para minimizar el tiempo total gastado por los empleados subiendo y bajando escaleras.

### Entrada

La entrada consiste de 3 números $\mathbf{A_1}$ , $\mathbf{A_2}$ , $\mathbf{A_3}$ ($0 \leq \mathbf{A_1}$ , $\mathbf{A_2}$ , $\mathbf{A_3} \leq 1000$), cada uno en una línea diferente. El número $\mathbf{A_k}$ representa el número de personas que trabajan en el **k**-ésimo piso.

### Salida

Su programa debe imprimir una única línea que contenga la cantidad de minutos que serán gastados si la máquina se coloca en el mejor lugar posible.

### Ejemplos

| Entrada | Salida |
|---|---|
| 10<br>20<br>30 | 80 |
| 10<br>30<br>20 | 60 |
| 30<br>10<br>20 | 100 |

## Problema 2: Palíndromo Real

Un palíndromo es una palabra, frase o cualquier otra secuencia de unidades (como una cadena de ADN; restricción de enzimas) que tiene la propiedad de ser leída de derecha a izquierda igual que de izquierda a derecha. Un número capicúa o palíndromo es un número entero cuyo inverso es él mismo. Este problema quiere que analices un número real y verifiques cual es el menor valor que puede ser sumado para que se transforme en un “palíndromo real” (sin un nombre específico hasta ahora). Por ejemplo, si el número es 101.099, al sumarle 0.002, obtenemos el palíndromo real 101.101. Otro ejemplo sería el número 13.31, que ya es un palíndromo real y solo se le debe sumar 0 para que quede de esa manera. Un ejemplo final es el número 100.9, al que se le debe sumar 0.1 para que el resultado sea 101.

Escribe un programa que, dado un número real, dé el menor valor a ser sumado tal que se convierta en un palíndromo real.

### Entrada

Habrá varios casos de prueba. El primer número en ser leído **C** es un entero representando el número de casos de prueba.

Cada caso de prueba tiene un número real **R** ( 0 ≤ **R** ≤ 999,999.999999 ). Recordando que la entrada tendrá un máximo de 6 lugares decimales, y que el separador decimal es un punto en vez de una coma.

### Salida

Para cada caso de prueba mostrar el valor requerido para convertir el número en un palíndromo real.

### Ejemplo

| Entrada | Salida |
|---|---|
| 3<br>101.099<br>13.31<br>100.9 | 0.002<br>0<br>0.1 |

## Problema 3: Ayuda a Jirafales!

Minutos antes de que la clase termine, el profesor Jirafales pasa la lista de asistencia. Un día decidió verificar las materias y noto que algunos estudiantes firman diferente en algunas clases y sospechó que alguien haya estado firmando por ellos.

Como el profesor tiene muchos estudiantes y no suficiente tiempo (el café con Doña Florinda es prioridad), te ha preguntado a tí para que lo ayudes a validar la firma de los estudiantes. Una firma es considerada falsa si hay más de una diferencia entre la original y la que se está queriendo chequear. Considerando como diferencia una letra mayúscula como minúscula y viceversa.

### Entrada

Habrá varios casos de prueba. La primera línea de cada caso de prueba comienza con un entero **N** ($1 \leq N \leq 50$) representando la cantidad de estudiantes en la clase. Las siguientes **N** líneas serán de la siguiente manera:

***Nombre del Estudiante  Firma Original del Estudiante***

Despues vendrá un entero M ($0 \leq M \leq N$), representando el número de estudiantes que asisten a clases. Las **M** líneas siguientes en el siguiente formato:

***Nombre del Estudiante  Firma del dia***

Todos los estudiantes tienen sólo el primer nombre en la lista, todos los nombres son distintos y no contienen más de 20 letras (**a-z A-Z**).

La entrada finaliza con **N**=0, que no debe ser procesado.

### Salida

Para cada caso, imprimir una sola línea, con la cantidad de firmas falsas encontradas.

## Ejemplo

| Entrada | Salida |
|---|---|
| 4<br>Chaves ChAvEs<br>Kiko kikO<br>Nhonho NHONHO<br>Chiquinha<br>CHIquinHa<br>3<br>Chaves ChAvEs<br>Kiko kIKO<br>Chiquinha<br>CHIquinHA<br>2<br>Jadson jadsON<br>Crishna Crishna<br>2<br>Crishna CRISHNA<br>Jadson JADson<br>0 | 1<br>2 |

## Problema 4: Imagen

Rafael encontró un nuevo hobby: escribir usando caracteres del teclado. Aunque este nuevo tipo de arte es simple y limitado, la creatividad es suficiente para hacer muy buenos dibujos.

Después de hacer un par de dibujos, Rafael se preguntó cómo podrían ser si eran redimensionados, pero tener que rehacer todos los dibujos parecía un poco cansador y te preguntó si lo ayudabas.
Cuando quieres redimensionar algo, una imagen de **N** líneas y **M** columnas tendrá **A** líneas y **B** columnas, y, dado que las nuevas dimensiones son más grandes que las dimensiones originales, algunos caracteres tendrán que repetirse.

Digamos que **A** es 3 veces más grande que **N**. En ese caso, cada línea tendrá que repetirse 3 veces, así la imagen es redimensionada correctamente.

Dado un dibujo hecho por Rafael, muestra como sería el dibujo si fuera redimensionado a una nueva dimensión dada

### Entrada

Habrá varios casos de prueba.
Cada caso de prueba comienza con dos enteros **N** y **M** ($1 \leq N, M \leq 50$), representando respectivamente, la altura y el ancho del dibujo de Rafael.
A continuación vienen N líneas, cada una conteniendo M caracteres, representando el dibujo hecho por Rafael. Después, vienen dos enteros **A** y **B** ($N < A \leq 100$, $M < B \leq 100$, **A** es múltiplo de **N**, y **B** es múltiplo de **M**), representando, respectivamente, la nueva altura y el nuevo ancho que Rafael quiere que el dibujo tenga.
El último caso de prueba es indicado cuando **N**=**M**=0, que no deberá ser procesado.

### Salida

Para cada caso de prueba, imprimir **A** líneas, conteniendo **B** caracteres cada una, representando el dibujo de Rafael redimensionado

### Ejemplo

| Entrada | Salida |
|---|---|
| 3 3<br>###<br>#__<br>###<br>6 9<br>0 0 | #########<br>#########<br>###______<br>###______<br>#########<br>######### |

## Problema 5: Intercambio de tren

En una vieja estación de ferrocarril, todavía se puede encontrar uno de los últimos “swappers de tren”. Un swapper de tren es un empleado del ferrocarril, cuyo único trabajo es reordenar los carros del tren.

Una vez que los carros están reorganizados en el orden óptimo, todo lo que el conductor del tren debe hacer, es desenganchar los carros, uno por uno, en las estaciones donde deben ser cargados.

El título de “swapper de tren” se deriva de la primer persona que realizó esta tarea, en una estación cercana a un puente de ferrocarril. En vez de abrirse verticalmente, el puente giraba alrededor de un pilar en el centro del río. Luego de rotar el puente 90 grados, los barcos podían pasar por la izquierda o derecha.

El primer swapper de tren descubrió que el puente puede ser operado teniendo como máximo dos carros encima. Rotando el puente 180 grados, los carros cambian de lugar, permitiéndole reordenar los carros (como un efecto secundario, los carros luego “ven” para direcciones opuestas, pero los carros del tren se pueden mover para ambos lados, así que a quien le importa).

Ahora que casi todos los swappers de tren han fallecido, la compañía de ferrocarril quiere poder automatizar su operación. Parte del programa ha ser desarrollado, es una rutina que decide para un tren dado el menor número de intercambios de carros adyacentes para ordenar el tren. Tu tarea es crear esta rutina.

### Entrada

La entrada contiene en la primer línea el número de casos de prueba *(**N**)*. Cada caso de prueba consiste de dos líneas. La primer línea de un caso de prueba contiene un entero ***L***, determinando la longitud del tren ($0 \leq L \leq 50$). La segunda línea de un caso de prueba contiene una permutación de los números 1 al ***L***, indicando el orden actual de los carros. Los carros deben ser ordenados tal que el carro 1 viene primero, luego el 2, etc. con el carro **L** llegando ultimo.

### Salida

Para cada caso de prueba imprimir la oración: “Optimal train swapping takes ***S*** swaps” donde ***S*** es un entero.

## Ejemplo

| Entrada | Salida |
|---|---|
| 3<br>3<br>1 3 2<br>4<br>4 3 2 1<br>2<br>2 1 | Optimal train swapping takes 1 swaps.<br>Optimal train swapping takes 6 swaps.<br>Optimal train swapping takes 1 swaps. |

### Problema 6: Cazando Digletts

Un Diglett es un Pokémon de tipo tierra que cava túneles subterráneos y casi nunca se deja ver. De vez en cuando sale a la superficie a través de un agujero en la tierra, y entonces se le puede ver sólo la cabeza.



Los túneles construídos por ellos son unidireccionales y siempre conectan un agujero de origen con uno de destino, por ejemplo: si hay un túnel conectando el agujero **A** con el agujero **B**, es posible ir desde **A** hacia **B** pero no al revés.

Cada Diglett posee su propio agujero y si hay **N** agujeros habrá **N** Digletts. Cada agujero tiene exactamente dos túneles: uno que va desde él hacia otro agujero y otro que viene desde otro hacia él.

Los Digletts viajan de agujero en agujero en cada momento del tiempo, por ejemplo: considere que el agujero **A** tiene un túnel que lo conecta con el agujero **B**, si un Diglett se encuentra en **A** en el momento **T**, en el momento **T+1** se encontrará en **B**. Cuando un Diglett llega a su agujero, se asoma a la superficie inmediatamente. Cuando no está en su agujero, se queda bajo tierra esperando al siguiente momento del tiempo para ir al siguiente agujero. Está garantizado que cada Diglett siempre retorna a su agujero en algún momento.

Xisto es un maestro Pokémon y quiere capturar la mayor cantidad posible de Digletts con una sola pokebola, que es capaz de capturar todos los Digletts visibles en un área determinada. Él necesita tu ayuda para saber cuál es el menor momento del tiempo en el que todos los Digletts aparecerán en la superficie simultáneamente, así puede tirar la pokebola y capturarlos a todos.

**Nota:** en el momento cero los Digletts están cada uno en su agujero y no salen a la superficie.

#### Entrada

La primer línea contiene un entero **N** ($2 \leq N \leq 100$) que representa la cantidad de agujeros. La siguiente línea contiene **N** enteros $B_i$ ($1 \leq B_i \leq N$), donde el ***i-ésimo*** entero representa al ***i-ésimo*** agujero, e indica que hay un túnel unidireccional desde el ***i-ésimo*** agujero hacia el agujero $B_i$.

#### Salida

Imprima el menor momento del tiempo en el que todos los Digletts aparecerán en la superficie al mismo tiempo.

**Ejemplo**

| Entrada | Salida |
|---|---|
| 2<br>2 1 | 2 |
| 4<br>4 3 2 1 | 2 |
| 6<br>2 1 5 3 6 4 | 4 |

**Problema 7: Ordenar!, Ordenar! y ordenar!!!**

Hmm! Acá sólo tienes que hacer un simple ordenamiento. Serán dados **N** números y un entero positivo **M**. Debes ordenar los **N** números de forma ascendente según su valor módulo **M**. Si hay un empate entre un número par y otro impar (sus valores módulo **M** son iguales), entonces el número impar debe preceder al número par. Si hay un empate entre dos número impares (sus valores módulo **M** son iguales), el mayor de los números debe preceder al menor, y si hay un empate entre dos números pares (sus valores módulo **M** son iguales), entonces el menor de los números debe preceder al mayor. Para calcular el valor del resto de números negativos sigan la regla del lenguaje de programación C: el módulo de un número negativo nunca puede ser mayor que cero. Por ejemplo: -100 MOD 3 = -1; -100 MOD 4 = 0; etc.

**Entrada**

El archivo de entrada contiene varios casos de prueba. Cada caso de prueba comienza con dos enteros **N** (0 < **N** ≤ 10000) y **M** (0 < **M** ≤ 10000), que representan lo indicado en el enunciado. Cada una de las siguientes **N** líneas contiene un número cada una. Estos números caben en un entero de 32 bits con signo. La entrada termina con una línea que contiene dos ceros.

**Salida**

La primer línea de salida de cada caso de prueba deben ser los números **N** y **M** correspondientes. Las siguientes **N** líneas deben contener los números ordenados según las reglas especificadas previamente. Imprima también los últimos dos ceros del archivo de entrada.

## Ejemplo

| Entrada | Salida |
|---|---|
| 15 3<br>1<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>9<br>10<br>11<br>12<br>13<br>14<br>15<br>3 3<br>9<br>12<br>10<br>0 0 | 15 3<br>15<br>9<br>3<br>6<br>12<br>13<br>7<br>1<br>4<br>10<br>11<br>5<br>2<br>8<br>14<br>3 3<br>9<br>12<br>10<br>0 0 |

## Problema 8: El mayor número “Un-dígito”

Los habitantes del planeta Uno tiene un terrible problema de detección de números con más de un dígito, por eso transforman los números enteros en números “Un-dígito” realizando sucesivas sumas del número hasta que se reduzca a un solo dígito. Por ejemplo: el número 999999999991 resulta en 9 + 9 + 9 + 9 + 9 + 9 + 9 + 9 + 9 + 9 + 9 + 1 = 100. Como el número 100 tiene más de un dígito, el proceso se repite, resultando 1 + 0 + 0 = 1. Una de las mayores dificultades que tiene la gente es comparar dos números y ver cuál es mayor bajo las reglas de este planeta.

Escriba un programa que, dados dos enteros, identifique cuál es el mayor número “Un-dígito”.

### Entrada

Habrá varios casos de prueba.
Cada caso de prueba posee dos enteros **N** y **M** ( $0 \leq \mathbf{N} \leq 10^{100}$, $0 \leq \mathbf{M} \leq 10^{100}$ ), indicando los dos números a comparar. El último caso contiene **N** = **M** = 0, y no debe ser procesado.

### Salida

Para cada caso de prueba, imprima una línea que contenga un entero: 1 si el primer número es mayor que el segundo, 2 si el segundo número es mayor que el primero, ó 0 si ambos números tienen el mismo valor.

### Ejemplo

| Entrada | Salida |
|---|---|
| 111 2<br>22 55<br>123 222<br>12 4<br>0 0 | 1<br>1<br>0<br>2 |

## Problema 9: Super Primos: Ataquen!

La Asociación de Primos Indivisibles eligio una categoría de números primos llamada Super Primos. Un número es considerado Super Primo si además de ser primo, todos sus dígitos son primos. La Asociación te solicitó que hagas un programa que caracterice los números.

### Entrada

La entrada contiene varios casos de prueba, cada caso de prueba comienza con un entero **N** ($0 < \mathbf{N} < 10^5$) en una sola línea. La entrada finaliza en el último caso de prueba.

### Salida

Para cada caso de prueba, la clasificación del número de entrada se muestra en una sola línea, que puede ser: **"Super"**, si el número es un Super Primo; **"Primo"** si el número sólo es un primo; ó **"Nada"** si el número tiene divisores más allá del 1 y él mismo.

### Ejemplo

| Entrada | Salida |
|---|---|
| 23<br>33<br>43 | Super<br>Nada<br>Primo |

## Problema 10: Construyendo Paredes

Después de que el colosal titán destruyera la pared de Maria, la Tropa de Exploración ha decidido construir una nueva, esta pared será tan dura que ningún titán podrá destruirla.

Pero si el titán es demasiado alto este simplemente puede saltar por encima de la pared, a causa de esto el Ejército de Exploración te contrató para que escribas un programa, que dada la altura de la pared y el tamaño de los titanes conocidos, responda cuales titanes pueden pasar por encima de la pared.

Un titán puede saltar por encima de la pared sólo si es más alto que ella.

### Entrada

La primer línea de entrada contiene dos enteros **N** ($1 \leq N \leq 100$) y **W** ($1 \leq W \leq 1000$) representando cuantos titanes la Tropa de Exploración conoce y el tamaño de la pared que intenta construir.

Cada una de las siguientes **N** líneas contiene un string **S** ($1 \leq |S| \leq 100$) representando el nombre del titán, seguido por un entero **H** ($1 \leq H \leq 1000$) representando la altura del titán. El string está compuesto por minúsculas, mayúsculas y espacios.

El nombre del titán nunca comienza o termina con espacio.

### Salida

Tu programa debe mostrar cuáles titanes podrían pasar por encima de la pared, los titanes deben ser mostrados en el orden que aparecieron en la entrada.

### Ejemplo

| Entrada | Salida |
|---|---|
| 3 50<br>Titan Colossal 60<br>Titan Encoracado 15<br>Titan Femea 14 | Titan Colossal |