Retos HACK A TON — JUECES — CONFIDENCIAL

Primer reto.

Se podría decir que es un reto “de calentamiento”. Es intencionadamente simple para que los estudiantes de cursos inferiores puedan obtener algunos puntos. La resolución del mismo es trivial.

Explicar el funcionamiento del código es extremadamente importante. Este reto es muy conocido y hay múltiples soluciones o variaciones del mismo por internet.

Casos Base:

Se considerará el ejemplo dado como caso base.

Casos Extremos:

Input	Output
42 39 48 1 10 19 22	30 39 48 1 10 19 28
38 8 7 9 18 23 29	38 47 7 9 18 27 29
46 6 47 17 24 35 37	46 6 8 17 26 35 37
5 14 16 25 34 36 45	5 14 16 25 34 36 45
13 15 26 33 20 44 4	13 15 24 33 42 44 4

```
21  27  32  41  43  30  12      21  23  32  41  43   3  12
28  31  40  49   2  11   3      22  31  40  49   2  11  20

Diagonal (Arriba Izq - Dcha) antes era: {42,  8, 47, 25, 2
0, 30,  3} ahora es {30, 47,  8, 25, 42,  3, 20}
 Diagonal (Abajo Izq - Dcha) antes era: {28, 27, 26, 25, 2
4, 23, 22} ahora es {22, 23, 24, 25, 26, 27, 28}
```

Modificación in-situ (5 minutos):

Hacer un “simulador” del programa. Imprimir uno de cada cinco estados intermedios por pantalla mostrando las sumas de las filas y las columnas de la matriz y el coste acumulado.

Problemas esperados:

No tener en cuenta la suma de las diagonales y el **cálculo erróneo de los costes**.

Segundo Reto.

El objetivo principal del reto es que los participantes descubran los patrones comunes en las formas de las letras, creen casos base y métodos para definir las letras restandes a partir de esos casos base. Por ejemplo:

```
    "o" en ASCII art es: "a" en ASCII art es: "c" en ASCII
```

```
 art es:
    -------------------- -------------------- ------------
--------
             ###                  ##                   ###
             # #                  # #                  #
             ###                  ###                  ###
```

Como se puede observar, la única diferencia entre “a” y “c” respecto a “o” se encuentra en una “#”. Con definir “o” y crear maneras de simular “a” y “c” en base a “o” se puede reducir de manera significante el tamaño de informacón guardada.

El objetivo final es que los participantes encuentren una manera de ahorrar memoria buscando patrones de los cuales partir. Aunque en este caso se empleen 10 letras, la eficiencia de este método es mucho mayor en sets masivos.

Los **participantes deberÃ¡n mostrar astucia a la hora de elegir las 10 letras del alfabeto y el formato del ASCII-ART**, dado que, en función de lo que elijan, la labor de encontrar un patrón común será más complicado o más simple.

Problemas esperados:

No encontrar patrones comunes, una mala elecciÃ³n de letras del alfabeto, no sincronizar la impresión de las letras por pantalla, etc.

Casos Base:

Un caso que incluya todas las letras definidas y espacios y un caso que incluya símbolos que no han definido ("|" o “” preferiblemente).

Casos Extremos:

Un caso que incluya input masivo “aaaaaaaa…” y un caso que incluya input multilinea.

ModificaciÃ³n In-Situ (5 minutos):

Incluir un modo “itálico” en el que el ASCII-ART se muestre por pantalla en *italics* en vez de estándar.

```
    Letras escogidas: {h, z, b, k, w, o, n, c, t, a}
    Input: hack a ton

    Output:
       #              #               #
      ###  ##   ###  # #     ##     ###  ###  ##
     # #  # #  #    ##      # #     #   # #  # #
    # #  ###  ###  #  #    ###     ##  ###  # #
```

Tercer Reto.

El objetivo principal del reto es que los competidores se vean forzados a

resolver el problema de una manera no clásica y poco común. Se considera que emplear **recursividad** para resolver el problema es la mejor opción, aunque cualquier otra manera de realizarlo que se considere lo suficientemente lógica y optimizada será aceptada.

Idealmente, los participantes deberán interpretar cada caracter del input como un token diferente y lograr que el programa sea capaz de predecir el siguiente token según la expresión regular dada. Debido a esto, el **backtracking queda terminantemente prohibido** a la hora de analizar el input.

Casos Base:

```
Expresión: [a-z][0-9]+[A-Z]
    Input: ss7A
   Output: "Incorrecto"
-------------------------------
Expresión: [a-z](yes|no)?[0-9]+
    Input: hackaton
   Output: "Correcto"
```

Casos Extremos:

```
Expresión: (hack|ton)+.[a-z]*
    Input: hackaton
   Output: "Correcto"
-------------------------------
```

```
Expresión: (.|hack)?[a-z]?
 NO Input:
   Output: "Correcto"
```

Modificación In-Situ (5 minutos):

Reprogramar el funcionamiento del símbolo “|”. Ahora este símbolo será representado por “-”.

```
Cambiar "|" -> "-"
Vieja Expresión: (hack|ton)+.[a-z]*
Nueva Expresión: (hack-ton)+.[a-z]*
          Input: hackaton
         Output: "Correcto"
```

Problemas esperados:

Se espera que los participantes tengan dificultad a la hora de conceptualizar el problema. Además, a la hora de imposibilitar el backtracking, es más que seguro que no tengan en cuenta algún que otro caso. La modificación in-situ también les obliga a considerar un triple valor para el mismo símbolo dependiendo de los símbolo de delimitación que lo rodeen.