Práctica de programación en Python

Programación para ciencia de datos 2024–2

Escribe programas de Python que resuelvan los siguientes problemas. Únicamente puedes usar la biblioteca estándar del lenguaje.

1 Múltiplos de 3 o 5

A partir de un entero no-negativo `n`, encuentra la suma de todos los múltiplos de 3 o 5 menores a n.

Ejemplo

Entrada `n = 10`
Salida `23`
Explicación Los múltiplos de 3 menores a 10 son $3, 6, 9$. Los múltiplos de 5 menores a 10 son 5. La suma resultante es $3 + 5 + 6 + 9 = 23$.

2 Combinar listas ordenadas

Considera dos listas de enteros `nums1` y `nums2` en orden no-decreciente, dos enteros `m` y `n` correspondientes a la cantidad de elementos en `nums1` y `nums2` respectivamente.

Combina `nums1` y `nums2` en una lista que contenga los elementos de ambas y también tenga orden no-decreciente.

Ejemplo 1

Entrada `nums1 = [1,2,3]`, `m = 3`, `nums2 = [2,5,6]`, `n = 3`
Salida `[1,2,2,3,5,6]`

Ejemplo 2

Entrada `nums1 = [1]`, `m = 1`, `nums2 = []`, `n = 0`
Salida `[1]`

Ejemplo 3

Entrada `nums1 = []`, `m = 0`, `nums2 = [1]`, `n = 1`
Salida `[1]`

3 Encuentra cúmulos de patrones en genomas

Considera que una secuencia de ADN se representa como una cadena de caracteres de Python con elementos `'A'` (Adenina), `'C'` (Citocina), `'T'` (Timina) y `'G'` (Guanina).

Dados dos enteros L y t, una secuencia de ADN `pattern` forma un (L, t)-cúmulo en una secuencia de ADN más grande `genome` si existe una región contigua de `genome` con longitud L donde `pattern` aparece al menos t veces.

Encuentra el conjunto de secuencias de ADN de longitud k que forman (L, t)-cúmulos en una secuencia `genome`.

Ejemplo

Entrada `genome = GATCAGCATAAGGGTCCCTGCAATGCATGACAAGCCTGCAGTTGTTTTAC`, `k = 4`, `L = 25`, `t = 3`

Salida `{'TGCA'}`

Explicación La secuencia `genome` tiene longitud 50, de entre todas sus regiones de longitud 25 hay únicamente un patrón que aparece al menos 3 veces en la región. El patrón es `'TGCA'` y la región es `genome[18:43]`.

Pruebas

Junto a este documento hay dos directorios `inputs` y `outputs`, los cuales usaremos para verificar tu solución.

Implementa la función `solve` en el archivo `dna.py` y corre el comando `python run_tests.py` para verificar empíricamente si tu solución es correcta o no.

Una vez que tu implementación funcione para las cuatro pruebas, edita el archivo `run_tests.py` para descomentar la quinta prueba. ¿Puedes hacer que sea rápida de resolver?