Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

CI2691 - Laboratorio de Algoritmos y Estructuras I

Enero-Marzo 2024

Laboratorio (Semana 6)

(5 puntos)

1. Descripción de la Actividad

Se quiere que realice programas en Python que resuelvan los siguientes problemas. Todas las soluciones deberán estar en un solo script `LabSem6.py`. Serán necesarias verificaciones de los argumentos (y lanzar su correspondiente Excepción si no las cumplen), en caso de ser necesario, excepto las verificaciones de tipo.

1. `numero_de_kaprekar` *(0.5 puntos)*: Un número es un número de Kaprekar si la representación decimal de su cuadrado puede ser dividida en dos partes, de tal manera que la suma de las partes sea igual al número original. Por ejemplo, $45^2 = 2025$ y $20 + 25 = 45$. Dado un número entero n, determina si n es un número de Kaprekar.

   Ejemplos de ejecución:

   ```
   numero_de_kaprekar(45)  # True
   numero_de_kaprekar(46)  # False
   ```

2. `numero_feliz` *(0.5 puntos)*: Un número no negativo es feliz si al reemplazar el número por la suma de los cuadrados de sus dígitos, y repetir el proceso, se llega al número 1[1]. Dado un número entero n, determina si n es un número feliz. Por ejemplo: $19 \to 1^2 + 9^2 = 82 \to 8^2 + 2^2 = 68 \to 6^2 + 8^2 = 100 \to 1^2 + 0^2 + 0^2 = 1$.

   Verificaciones:

   a) El número debe ser no-negativo.

   Ejemplos de ejecución:

   ```
   numero_feliz(19)  # True
   numero_feliz(20)  # False
   ```

3. `anagrama` *(0.5 puntos)*: Se quiere determinar, dadas dos cadenas, si una es un anagrama de la otra. Un anagrama es un string que se forma reorganizando las letras de otro string.

   Ejemplos de ejecución:

   ```
   anagrama("amor", "roma")  # True
   anagrama("hola", "mundo") # False
   ```

4. `comprimir` *(0.5 puntos)*: Dado un string, se quiere obtener su versión comprimida usando Run-Length Encoding (RLE). Esta consiste en sustituir tokens que se repiten consecutivamente por el token y la cantidad de ocurrencias. Para este caso, cada token será un carácter y todos los strings deberán tener valores no numéricos.

   Verificaciones:

[1] Si se repite algún número, significa que nunca llegará a 1.

a) El string no debe contener caracteres numéricos.

Ejemplos de ejecución:

```
comprimir("aaaaabbbcccc")  # "5a3b4c"
comprimir("ab")            # "1a1b"
```

5. `puede_colocar_flores` *(1 punto)*: Tienes un macizo de flores en el que algunas parcelas están plantadas y otras no. Sin embargo, no se pueden plantar flores en parcelas adyacentes. Dada una lista de números enteros que contiene ceros y unos, donde 0 significa vacío y 1 significa que hay una flor plantada, y dado un número entero no-negativo `n`, se quiere saber si se pueden plantar `n` nuevas flores en el macizo de flores sin violar la regla de no flores adyacentes.

   Verificaciones:

   a) El número de flores a plantas es no-negativo.
   b) Los valores de la maceta deben ser 0 o 1.

   Ejemplos de ejecución:

```
puede_colocar_flores([1, 0, 0, 0, 1], 1)  # True
puede_colocar_flores([1, 0, 0, 0, 1], 2)  # False
```

6. `sopa_de_letras` *(1 punto)*: Dado un string y una matriz $M \times N$ de caracteres, se quieres sabes si el string aparece en la matriz como en una sopa de letras.

   Verificaciones:

   a) La matriz debe ser $M \times N$ para algún M, N enteros.
   b) La matriz sólo puede contener caracteres (strings de longitud 1).

   Ejemplos de ejecución:

```
sopa_de_letras(
    "HOLA",
    [["a", "w", "a" ,"q", "s", "t"],
     ["m", "z", "b" ,"a", "6", "1"],
     ["<", "a", "H" ,"d", "h", "G"],
     ["s", " ", "*" ,"O", "a", "$"],
     ["a", "y", "a" ,"u", "L", "F"],
     ["v", "j", ";" ,"@", "j", "A"]]
) # True
sopa_de_letras(
    "HOLA",
    [["a", "w", "a" ,"q", "s", "t"],
     ["m", "z", "b" ,"a", "6", "1"],
     ["<", "a", "H" ,"d", "h", "G"],
     ["s", " ", "*" ,"8", "a", "$"],
     ["a", "y", "a" ,"u", "L", "F"],
     ["v", "j", ";" ,"@", "j", "A"]]
) # False
```

7. `jaque` *(1 punto)*: En un tablero de ajedrez, dada la posición del rey, una lista de posiciones de torres y una lista de posiciones de alfiles, se quiere saber si las torres y los alfiles hacen jaque al rey. Las posiciones del tablero se enumeran igual que las posiciones en una matriz, es decir, la posición `(0, 0)` representa la esquina superior izquierda, y la posición `(7, 7)` representa la esquina inferior derecha. Las torres y alfiles pueden estar superpuestas pero el rey si tiene que estar en una posición no ocupada.

   Verificaciones:

   a) Todas las posiciones deben pertenecer al conjunto $[0, 8) \times [0, 8)$.
   b) El rey no puede ocupar la misma posición que otra pieza.

Ejemplos de ejecución:

```
jaque((0, 0), [(0, 5)], [])          # True
jaque((0, 0), [(3, 3)], [])          # False
jaque((0, 0), [], [(3, 3)])          # True
jaque((0, 0), [(3, 3)], [(3, 3)])    # True
jaque((0, 0), [], [(0, 0)])          # ValueError
```

2. Evaluación

Se implementará un script para la evaluación de cada solución propuesta, el cual aplicará una variedad de casos de prueba, y generará un resultado final. Se evaluarán las firmas de las funciones. Si no se pueden importar las funciones, se penalizará con -2 puntos. Las funciones que no se puedan correr sin realizar ninguna modificación tendrán la mitad de los puntos obtenidos. Aunque no se realizará una evaluación del estilo de codificación, se efectuarán correcciones en este aspecto, lo cual les será de utilidad en evaluaciones futuras, donde dicho estilo podría ser objeto de evaluación.

3. Condiciones de Entrega

El programa correspondiente al laboratorio `LabSem6.py` y la declaración de autenticidad debidamente firmada, deben ser incluidos en un archivo comprimido, en formato `tar.xz` (o `zip`), denominado `LabSem6_X.tar.xz` (o `LabSem4_X.zip`), donde `X` representa el número de carné del estudiante. La entrega del archivo debe realizarse a través de Gmail, enviándolo al correo electrónico **16-10072@usb.ve** antes de las 11:59 PM del día domingo 25 de febrero de 2024. Las entregas que se realicen después de la hora límite tendrán una penalización de un (1) punto, más un punto adicional por cada hora de retraso. Por lo tanto, después de 4 horas de retraso, la evaluación será anulada.