Fundamentos de Algoritmos – Examen Final

Master in Management + Analytics, Escuela de Negocios, UTDT

Primer semestre 2020

- El examen es individual y se aprueba con 60 puntos o más, sobre un total de 100.
- Está permitido usar Python y Spyder para hacer pruebas, consultar materiales de la materia y documentación online.
- No está permitido comunicarse por ningún medio con otros estudiantes ni con otras personas durante el examen, excepto con los docentes de la materia.
- Puede consultarse a los docentes solo por aclaraciones específicas del enunciado, a través del chat de Zoom.
- La resolución de los ejercicios debe realizarse en un único archivo de texto plano llamado "`examen.txt`". Ese archivo debe subirse a la "Entrega del examen" en el campus virtual.

Problema 1. (20 puntos) [IQJ11]
Escribir una función en Python que, dada una lista de strings, devuelva un nueva lista de enteros formada por las longitudes de aquellos elementos que tienen la letra 'p'.

En este problema no está permitido usar ciclos (`while`, `for`, etc.). Debe resolverse usando **listas por comprensión**, completando solamente la línea punteada del siguiente código:

```
1  def procesar(lista):
2     return [ ........ ]
3  }
```

Ejemplos:
`procesar(['hola', 'pepe', 'ppppp', 'chau', 'pu'])` debe devolver `[4, 5, 2]`
`procesar([])` debe devolver `[]`
`procesar(['hola', 'chau'])` debe devolver `[]`
`procesar(['pepe', 'ppppp'])` debe devolver `[4, 5]`

Problema 2. (25 puntos) [NK4G3]
Considerar el Tipo Fecha con las operaciones que se muestran a continuación:

- `Fecha(dia,mes,anio)`: Crea un nuevo elemento de tipo Fecha con el día, mes y año especificados.
- `f.dia()`: Devuelve el día de la fecha `f`.
- `f.mes()`: Devuelve el mes de la fecha `f`.
- `f.anio()`: Devuelve el año de la fecha `f`.
- `f.avanzar(dias)`: Hace avanzar a `f` la cantidad de días especificada.
- `f.retroceder(dias)`: Hace retroceder a `f` la cantidad de días especificada.

A continuación puede observarse un ejemplo de uso y el comportamiento esperado:

```
f = Fecha(25,5,1977)
print(f)                    # imprime '25/5/1977'
f.avanzar(3)
print(f)                    # imprime '28/5/1977'
f.retroceder(8)
print(f)                    # imprime '20/5/1977'
```

Se cuenta con una implementación sobre la cual se verificó formalmente que las operaciones `avanzar` y `retroceder` están correctamente implementadas. El código de las restantes operaciones es el siguiente:

```
class Fecha:
    def __init__(self, d, m, a):
        self._anio = a
        self._mes = a
        self._dia = a

    def dia(self):
        return self._dia

    def mes(self):
        return self._mes

    def anio(self):
        return self._anio
```

Para poner a prueba la implementación se escribieron los siguientes tests:

```
  def test_avanzando(self):
      f = Fecha(1,2,2000)
      f.avanzar(3)                    # f debiera representar '4/2/2000'
      self.assertEqual(f.dia(), 4)

  def test_retrocediendo(self):
      f = Fecha(4,4,2000)
      f.retroceder(4)                 # f debiera representar '31/3/2000'
      self.assertEqual(f.mes(), 3)
```

a) Para cada test, hacer un seguimiento cuidadoso e indicar si falla o no.

b) Para cada test que falla (si es que hay), encontrar y explicar el error que lo hace fallar.

Problema 3. (30 puntos) [MWN33]
Se tiene una lista con S cofres y en cada cofre se encuentra un mazo de T cartas. Un mazo es simplemente una lista de cartas y las cartas se pueden comparar por $<$. Leer el siguiente código y responder las preguntas que se presentan a continuación:

```
1  def f_aux(mazo):
2      i = 1
3      b = 0
4      while i < len(mazo):
5          if mazo[i-1] < mazo[i]:
6              b = b + 1
7          i = i + 1
8      return b == len(mazo)-1
9
10 def f(cofres):
11     r = 0
12     for c in cofres:
13         t = obtener_el_mazo_del_cofre(c)
14         if f_aux(t):
15             r = r + 1
16     return r
```

(a) Sabiendo que la función `obtener_el_mazo_del_cofre(c)` tiene complejidad temporal $O(1)$ en el peor caso, determinar cuál es la complejidad temporal de `f` en el peor caso en función de S (la cantidad de cofres) y T (la cantidad de cartas en un mazo).

(b) Renombrar las funciones `f` y `f_aux` por nombres más descriptivos que den cuenta de lo que hacen (máximo 5 palabras cada nombre). Justificar la elección de los nuevos nombres.

(c) Responder las siguientes preguntas y justificar:

(I) ¿Cuál es el valor de la variable `i` después de salir del ciclo `while`?

(II) ¿Qué devuelve la función `f` si todos los mazos tienen 1 sola carta?

Problema 4. (25 puntos) [2G2EH]

Contamos con un tipo de datos `Pieza` ya definido, con las siguientes operaciones:

- `Pieza(peso, altura)`: Crea un nuevo elemento de tipo `Pieza`, con los valores de peso y altura especificados (enteros mayores que 0).
- `p.peso()`: Devuelve el valor de peso de la pieza `p` (un entero mayor que 0).
- `p.altura()`: Devuelve el valor de altura de la pieza `p` (un entero mayor que 0).

Se pide definir una función que toma como argumentos una lista de elementos de tipo `Pieza` y un valor entero `umbral`, y calcula el peso máximo de los elementos que tienen altura mayor que `umbral`. Si ningún elemento cumple esa condición, debe devolver -1.

En este problema no está permitido usar listas por comprensión, ni tampoco la función `max`. Debe resolverse usando un ciclo (`while` o `for`).

Ejemplos:

```
piezas = []
piezas.append(Pieza(1, 2))
piezas.append(Pieza(5, 5))
piezas.append(Pieza(7, 5))
piezas.append(Pieza(40, 2))
piezas.append(Pieza(3, 5))
print(maximo(piezas, 3))  # imprime 7
print(maximo(piezas, 2))  # imprime 7
print(maximo(piezas, 1))  # imprime 40
print(maximo(piezas, 7))  # imprime -1
```