

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. (25 puntos) [ZIF22]

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:

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

Problema 2. (20 puntos) [1IZ4W]

Escribir una función en Python que, dada una lista de enteros, devuelva una nueva lista de enteros formada por el triple de aquellos elementos que son mayores o iguales que 10.

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([1, 4, 10, 20, 21, 30, 9, 10])` debe devolver `[30, 60, 63, 90, 30]`

`procesar([])` debe devolver `[]`

`procesar([1, 4])` debe devolver `[]`

`procesar([10, 30])` debe devolver `[30, 90]`

Problema 3. (30 puntos) [VRJ33]

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) [IPF41]

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:

```
1 f = Fecha(25,5,1977)
2 print(f)                # imprime '25/5/1977'
3 f.avanzar(3)
4 print(f)                # imprime '28/5/1977'
5 f.retroceder(8)
6 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:

```
1 class Fecha:
2     def __init__(self, d, m, a):
3         self._anio = a
4         self._mes = d
5         self._dia = m
6
7     def dia(self):
8         return self._dia
9
10    def mes(self):
11        return self._mes
12
13    def anio(self):
14        return self._anio
```

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

```
1 def test_avanzando(self):
2     f = Fecha(1,2,2000)
3     f.avanzar(3)                # f debiera representar '4/2/2000'
4     self.assertEqual(f.dia(), 4)
5
6 def test_retrocediendo(self):
7     f = Fecha(4,4,2000)
8     f.retroceder(4)            # f debiera representar '31/3/2000'
9     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.