Contro de cátedra 1

COM4502 Programación Avanzada

Pregunta 1

Decida si cada afirmación siguiente es verdadera o falsa, argumentando cada respuesta.

1. Si la variable L es un diccionario con n pares de llave-valor, cualquier algoritmo que requiera leer los elementos de L tendrá complejidad $\Omega(n)$.

2. Al ejecutar el código

```
x="arrecife"
x.replace('r','l')
print(x)
```

   en pantalla se imprime el string `allecife`.

3. Si en la variable X hay guardada una lista, entonces la línea de comando $X[-1] =$ "a" no arroja error y asigna el valor "a" a la última celda de `X`.

4. El siguiente código nunca termina:

```
x=100
while x!=0:
  x-=3
print(x)
```

5. Al ejecutar el código:

```
arch = open("texto.txt",'r')
x=arch.readline()
if x=='hola':
  print("A")
else:
  print("B")
```

   si el archivo "texto.txt" contiene dos líneas, la primera con el texto `hola` y la segunda con el texto `HOLA`, en pantalla entonces se imprime "A".

6. Al ejecutar

```
x=5
def func(arg):
  global x
  x=10
  return arg*x
print(x)
```

en pantalla se imprimirá `10`.

Solución Propuesta

1. Verdadero. Si un algoritmo requiere leer cada elemento del diccionario al menos una vez, entonces el número de operaciones elementales que realiza es al menos n, y por ende su complejidad es $\Omega(n)$.

2. Falso. Ya que los strings son inmutables, la función replace no modifica el texto guardado en la variable x sino que crea una copia con los cambios. Ya que la variable x no se actualiza en ningún momento, x sigue siendo `"arrecife"`.

3. Verdadero. Esta línea de comando modifica la entrada en posición `len(X)-1` dándole el valor `"a"`. La única forma que tiene de fallar es que X sea una lista sin elementos.

4. Verdadero. La condición de término es que x sea igual a cero, lo que nunca ocurre pues los posibles valores de x a través de las iteraciones son de la forma $100 - 3 \cdot k$ para $k \in \mathbb{N}$, y 100 no es divisible por 3.

5. Falso. Ya que el archivo tiene dos líneas, al ejecutar el comando `arch.readline()` se recupera el string `"hola\n"`, y por ende se imprime `"B"` en pantalla.

6. Falso. La función modificaría el valor de x ya que está declarando sus cambios como globales, pero la función nunca es invocada y por ende esas líneas de código no se ejecutan, por lo que x sigue valiendo 5.

Pregunta 2

1. Describa en detalle el significado de la frase, proveyendo algún ejemplo concreto:

   (a) La complejidad del algoritmo $\mathcal{A}$ en términos de n es $O(f(n))$.

   (b) La complejidad del algoritmo $\mathcal{B}$ en términos de n es $\Theta(f(n))$.

2. Analice la complejidad de los siguientes algoritmos en función de las entradas que reciben, proveyendo resultados tan precisos como sea posible en términos de dominación asintótica.

   (a)
```
def func(n):
  x=n
  while x>0:
    x//2
  return x
```

   (b)
```
def func2(n):
  x=1
  for i in range(0,n,2):
    for j in range(0,i):
      x+=i+j
  return x
```

   (c)
```
  def func3(n):
    x=n
    if x>1000:
      x-=n+1000
    else:
      while x>0:
        x-=1
    return x
```

Solución propuesta

1. (a) La frase "La complejidad del algoritmo $\mathcal{A}$ en términos de n es $O(f(n))$" hace alusión a que la función $T(n)$ correspondiente al número de operaciones elementales que realiza el algoritmo $\mathcal{A}$ al recibir una entrada de tamaño n en el peor caso, está dominada superiormente en el regimen asintóticamente por $f(n)$. En otras palabras, el número de operaciones que realiza el algoritmo $\mathcal{A}$ crece a lo más tan rápido como la función f respecto a n. Esta noción provee sólo garantías de cota superior, pues queda la puerta abierta a que la complejidad sea mucho menor que $f(n)$.

   Un posible ejemplo sería el siguiente algoritmo.

```
def funcion(n):
  x=1
  sol=1
  while x<n:
    sol=sol*x
    x+=1
  return sol
```

El número de operaciones elementales que realiza es aproximadamente $4(n-1)+2$, que podemos decir está en $O(n)$, en $O(n^2)$ o en $O(2^n)$.

(b) La frase "La complejidad del algoritmo $\mathcal{B}$ en términos de n es $\Theta(f(n))$" hace alusión a que la función $T(n)$ correspondiente al número de operaciones elementales que realiza el algoritmo $\mathcal{A}$ al recibir una entrada de tamaño n en el peor caso, está dominada asintóticamente tanto superior como inferiormente por $f(n)$. En otras palabras, el número de operaciones que realiza el algoritmo $\mathcal{A}$ crece a a la misma velocidad que la función f respecto a n. Esta noción provee garantías tanto de cota superior como de cota inferior, proveyendo las descripción más precisa posible en el regimen asintótico.

Siguiendo con el ejemplo anterior, la complejidad de dicho algoritmo está en $\Theta(n)$.

2. (a) Para $n > 0$ el código entra en un loop de largo indefinido, y por ende su complejidad no está acotada por ninguna función. En caso de actualizarse el valor de x dentro del ciclo `while`, la complejidad del algoritmo sería $\Theta(\log_2(n))$ pues en cada iteración el valor de x se divide al menos a la mitad hasta llegar a 1, lo que toma una cantidad logarítmica de iteraciones.

(b) En este caso tenemos dos ciclos `for` anidados, donde el ciclo externo realiza del orden de $n/2$ iteraciones y el interno realiza i iteraciones (donde i es la variable del ciclo externo). De esta forma, la complejidad asintótica queda descrita por

$$\sum_{i=1}^{n/2} 2i = 2\frac{n/2)(n/2+1)}{2} \in \Theta(n^2).$$

(c) En este caso, si $n > 1000$ se realiza sólo una cantidad constante de operaciones. Ya que nos interesa el regimen asintótico, podemos concluir que la complejidad del algoritmo es $\Theta(1)$.

Pregunta 3

1. Cree una clase `Vector3` para representar vectores de tres dimensiones. Cada objeto de la clase está definido por tres datos de tipo float `x`, `y` y `z`. La clase debe soportar las operaciones:

   - Recibir tres números y construir un vector con dichos tres números, asignándolos en orden a `x`, `y` y `z`.
   - Sumar dos vectores, calculando la suma de cada una de sus coordenadas.
   - Imprimir un vector, que debe imprimir las coordenadas del vector separadas por comas, entre corchetes (de la forma `[x,y,z]`).

2. Utilice la clase anterior para definir la clase `complejo` que representa números complejos (definidos por dos datos de tipo float, correspondientes a la parte real y la parte imaginaria). Debe soportar las operaciones de la clase anterior, pero modificar la impresión para tener el formato de número complejo (si tiene coordenadas a y b, entonces imprime `a+i*b`), y además debe soportar la multiplicación de complejos.

Solución propuesta

1.
```
class vector3():

    def __init__(self,x,y,z):
        self.x=x
        self.y=y
        self.z=z

    def __add__(self,vector):
        new = vector3(self.x+vector.x,self.y+vector.y,self.z+vector.z)
        return new

    def __str__(self):
        rep = "[" + str(self.x) + ", " + str(self.y) + ", " + str(self.z) +
        return rep
```

2.
```
class complejo(vector3):

    def __init__(self,real,imag):
        vector3.__init__(self,real,imag,0)

    def __str__(self):
        rep = str(self.y) + "*i + " + str(self.x)
        return rep
```

```
def __mul__(self,comp):
    real_new = self.x*comp.x - self.y*comp.y
    imag_new = self.x*comp.y + self.y*comp.x
    new = complejo(real_new,imag_new)
    return new
```