Guía de ejercicios C1

COM4502 Programación Avanzada

Pregunta 1

Defina los siguientes conceptos, proveyendo ejemplos en cada caso:

1. Sentencia condicional,
2. Programación orientada a objetos,
3. Dominación asintótica de funciones,
4. Operación elemental,
5. Scope local y global.

Pregunta 2

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

1. Si $f(n) \in \Theta(g(n))$, entonces $2^{f(n)} \in \Theta(2^{g(n)})$.
2. Al utilizar una seguidilla de comandos `if-elif-elif-...`, si una condición es satisfecha entonces el resto de la secuencia no es verificado.
3. Si se define la variable `x=4`, el resultado de `(2*x+4)/2` es `6`.
4. El siguiente código nunca termina:

```
i=32
while i>0:
  i//2
print(i)
```

5. Si `x=(1,2,3,4)`, el siguiente código imprime $(4, 3, 2, 1)$:

```
for i in range(2):
  aux = x[i]
  x[i] = x[3-i]
  x[3-i] = aux
print(x)
```

6. La única forma de poder recuperar algún valor definido en el scope local definido por una función es declarar la variable correspondiente como global.

Solución propuesta

1. Falso. Consideremos por ejemplo las funciones $f(n) = \log_2(n)$ y $g(n) = 2\log_2(n)$. En este caso tenemos que $f(n) \in \Theta(g(n))$, pero $2^{f(n)} = n$ y $2^{g(n)} = n^2$ y no tienen el mismo orden de crecimiento.

2. Verdadero. Python evalúa las condiciones una a una de arriba hacia abajo, pero cuando una se cumple ejecuta el bloque correspondiente y se salta el resto de las condiciones completas.

3. Falso. Por defecto, el resultado de la división ("/") es un dato de tipo `float`, por lo que el resultado sería `6.0`.

4. Verdadero. El valor de `i` nunca se actualiza, por lo que siempre es `32`. De esta forma, la condición del ciclo `while` siempre se satisface, entrando en un loop infinito.

5. Falso. `x` corresponde a una tupla, las que son inmutables, por lo que no podemos modificar sus valores a través de índices como en el código. Su ejecución arrojaría un error.

6. Falso. Es posible también recuperar un valor del scope local retornándolo como salida de la función.

Pregunta 3

Analice la complejidad en términos asintóticos de los siguientes códigos en función de los datos que recibe como entrada:

1.
```
def func1(n):
  x=1
  for i in range(n):
    for j in range(0,n,2):
      x*=i*j
  return x
```

2.
```
def func2(n):
  x=1
  for i in range(0,n,3):
    for j in range(i,n):
      x*=i*j
  return x
```

3.
```
def func3(n):
  x=n
  while x>0:
    x//2
  return x
```

4.
```
def func4(n,m):
  x=1
  y=1
  while x<=n:
    for i in range(m):
      y*=i
    x*=2
  return x,y
```

5.
```
def func5(n):
  x=n
  while x>0
    x%=5
    x-=1
  return x
```

Solución propuesta

1. En este caso tenemos una primera asignación de valor a la variable `x`, que toma tiempo $\Theta(1)$, y luego dos ciclos `for` anidados, donde el ciclo externo tiene tamaño n mientras que el ciclo interno tiene tamaño $n/2$ (pues escaneamos los elementos entre 0 y $n-1$ con saltos de tamaño dos); con esto, la parte correspondiente a los ciclos involucra $\Theta(n^2)$ operaciones, pues un ciclo interno con $\Theta(n)$ instrucciones se ejecuta $\Theta(n)$ veces. En total, la complejidad asintótica del código es $\Theta(n^2)$.

2. Nuevamente tenemos una primera asignación de valor a la variable `x`, que toma tiempo $\Theta(1)$. Después de eso tenemos dos ciclos anidados, pero el largo del ciclo interno depende del iterador del ciclo externo, por lo que debemos mirar más en detalle el cálculo. Para un valor fijo de `i`, el ciclo

interno toma $\Theta(n-i)$ operaciones, por lo que en total tendremos una complejidad asintótica del orden de

$$\sum_{i=1}^{n/3} n - 3i = \frac{n^2}{3} - 3\sum_{i=1}^{n/3} i = \frac{n^2}{3} - \frac{n(n+1)}{6} \in \Theta(n^2)$$

4. En este caso la función depende de dos parámetros, y por ende la complejidad asintótica será una función de dos variables. Las primeras dos instrucciones son asignaciones de valores, por lo que toman complejidad $O(1)$. Luego de eso, tenemos un ciclo `while` que tiene una condición dependiente de x, que dentro tiene un ciclo `for` dependiente de m (e independiente del ciclo externo), por lo que la complejidad estará definida por el producto del largo de ambos ciclos. El ciclo interno es siempre de largo $\Theta(m)$, pero el ciclo externo es más difícil de entender. La pregunta es cuántas veces debemos dividir n a la mitad para que llegue a valer 1, y la respuesta es $\Theta(\log_2(n))$ veces (pues $2^{\log_2(n)} = n$). Con esto, la complejidad asintótica del código es $\Theta(m\log(n))$.

5. En este caso debemos primero recordar qué hace el comando `%`: al ejecutar `x%y`, donde `x` e `y` son números enteros, el resultado obtenido es el resto de dividir `x` por `y`. En este caso, independiente de qué tan grande sea n, el resto de dividir n por 5 es un número entre 0 y 4, por lo que el ciclo siempre se ejecuta una cantidad constante de veces (pues el valor después de aplicar el resto va disminuyendo en 1 cada vez hasta llegar a cero). De esta forma, la complejidad asintótica de este código es $\Theta(1)$.

Pregunta 4

Ordene las siguientes funciones en términos de dominación asintótica, desde la menor hasta la mayor. En caso de equivalencias, puede decidir un orden arbitrario entre ellas.

- $f(n) = 2^n + 10$,
- $g(n) = 5n^2 + 15n + 2$,
- $h(n) = n! + n$,
- $q(n) = n\log^2(n)$,
- $r(n) = 1000000$,
- $s(n) = 100n^2 + 1$,
- $t(n) = \log(n^3)$,
- $v(n) = 50n\log(n)$.

Solución propuesta

En general, el orden creciente de funciones en términos de dominación asintótica es constante - logarítmico - polinomial - exponencial - factorial. En ese orden de cosas, la función r es constante, la función t es logarítmica, las funciones g, q, s y v son polinomiales, la función f es exponencial y la función h es factorial. Entre las funciones polinomiales, debemos ver si hay empates o no: la función g es $\Theta(n^2)$, la función q es $\Theta(n \log^2(n))$ (es decir, más grande que lineal pero más pequeña que cuadrática), la función s es $\Theta(n^2)$ y la función v es $\Theta(n \log(n))$ (también más grande que lineal pero más pequeña que $n \log^2(n)$). Con esto, el orden sería:

$$r \to t \to v \to q \to g, s \to f \to h.$$

Notar que g y s empatan en términos de dominación asintótica pues tienen el mismo comportamiento.

Pregunta 5

Implemente una función en Python que reciba como entrada un string `x` y que:

1. Retorne el string `x` invertido. Por ejemplo, si `x=hola mundo`, la salida debe ser `odnum aloh`.
2. Retorne cada palabra del string invertida. Por ejemplo, si `x=hola mundo`, la salida debe ser `aloh odnum`.
3. Retorne el orden de las palabras invertido. Por ejemplo, si `x=hola mundo`, la salida debe ser `mundo hola`.