amsmath amssymb

Tarea: Complejidad computacional y máquinas de estados finitos

Estimados estudiantes,

Resuelvan los ejercicios utilizando las fórmulas matemáticas y conceptos adecuados.

Respecto a Complejidad computacional.

Definiciones de Omicron, Omega y Theta en el contexto de la complejidad temporal de algoritmos, junto con ejemplos y algunos teoremas básicos.

- Omicron (Notación O grande):

  La notación O grande describe una cota superior asintótica en el tiempo de ejecución de un algoritmo. Representa el crecimiento máximo de una función en términos de su tamaño de entrada. Formalmente, $f(n)$ está en $O(g(n))$ si existe una constante positiva c y un tamaño de entrada n_0 tal que $f(n) \leq c \cdot g(n)$ para todo $n \geq n_0$.

  Ejemplo: Si un algoritmo tiene una complejidad temporal de $O(n^2)$, significa que su tiempo de ejecución crece cuadráticamente con el tamaño de entrada.

  Teorema básico: Si $f(n)$ está en $O(g(n))$ y $g(n)$ está en $O(h(n))$, entonces $f(n)$ también está en $O(h(n))$.

- Omega (Notación Ω grande):

  La notación Ω grande establece una cota inferior asintótica en el tiempo de ejecución de un algoritmo. Representa el crecimiento mínimo de una función en términos de su tamaño de entrada. Formalmente, $f(n)$ está en $\Omega(g(n))$ si existe una constante positiva c y un tamaño de entrada n_0 tal que $f(n) \geq c \cdot g(n)$ para todo $n \geq n_0$.

  Ejemplo: Si un algoritmo tiene una complejidad temporal de $\Omega(n)$, significa que su tiempo de ejecución crece al menos linealmente con el tamaño de entrada.

  Teorema básico: Si $f(n)$ está en $\Omega(g(n))$ y $g(n)$ está en $\Omega(h(n))$, entonces $f(n)$ también está en $\Omega(h(n))$.

- Theta (Notación Θ):

  La notación Θ combina las ideas de O grande y Omega grande. Representa una cota ajustada asintóticamente en el tiempo de ejecución de un algoritmo. Formalmente, $f(n)$ está en $\Theta(g(n))$ si existe una constante positiva c_1, una constante positiva c_2, y un tamaño de entrada n_0 tal que $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ para todo $n \geq n_0$.

  Ejemplo: Si un algoritmo tiene una complejidad temporal de $\Theta(n)$, significa que su tiempo de ejecución crece linealmente con el tamaño de entrada de manera ajustada.

  Teorema básico: Si $f(n)$ está en $\Theta(g(n))$, entonces $f(n)$ está tanto en $O(g(n))$ como en $\Omega(g(n))$

- En resumen, estas notaciones nos ayudan a comprender cómo crece el tiempo de ejecución de un algoritmo a medida que aumenta el tamaño de entrada.

- Referencia para profundizar:

  Introducción a los algoritmos (por Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein): - Este libro es un clásico en el campo de los algoritmos. Ofrece una cobertura completa de algoritmos y estructuras de datos, incluyendo análisis de complejidad temporal. Es ampliamente utilizado en cursos académicos y es una excelente referencia.

  Algorithms (por Robert Sedgewick y Kevin Wayne): - Este libro aborda algoritmos y estructuras de datos desde una perspectiva práctica. Incluye ejemplos de código en Java y ofrece una sólida base para comprender la complejidad temporal y otros conceptos relacionados.

Ejercicio:
Si $T(n) = 3^{8n}$ y $U(n) = 5^n$, determine si U es O(T) o viceversa, o U es $\Theta(T)$.

Ejercicio:
En general decida si $T(n) \in O(U(n))$ y viceversa, implica que U es $\Theta(T)$ y viceversa.

Ejercicio: Determine si la complejidad temporal del siguiente algoritmo es $O(n^n), \Theta(n^3), O(n!), \Omega(e^n)$, cuando a[] es una lista de números:

```
algoritmo0(a[1],....a[n],n)
  j = a[1]+...+a[n]+1
  k = (n!!!)-(n!!)
  imprimir j+k
```

Ejercicio:
Verifique que el siguiente algoritmo es $O(n \log(n))$ y determine el valor final de la variable j y cuántas veces se imprime el mensaje entre comillas:

```
algoritmo1(n)
para i = 0, ..., n
  j = 1
  mientras j < i
    j = j * 2
  imprimir "Make an effort studying while others sleep, then you will achieve what others dream of."
```

Ejercicio:

Escriba un algoritmo $\Theta(n^4)$, o bien escriba una función de complejidad de ejecución T(n) que sea de esa clase.

Ejercicio:

Escriba un algoritmo $O(n^3)$ que imprima en pantalla enteros en los casos: (a) Con una entrada de tamaño n. (b) Con una entrada libre.

Ejercicio:

Determine la menor clase $\Omega(.)$ a la que pertenece el siguiente algoritmo y cuántas veces se imprime "bye" y cuántas "odd":

```
algoritmo2(n)
for i=2,...,n
   if mod(i,2) ≠ 0
      print "odd"
   while n*i > n
      i=mod(i,3)-1
      print "bye"
```

Ejercicio:

Escriba un algoritmo $\Omega(\log(n))$, o bien escriba una función de complejidad de ejecución T(n) que sea de esa clase.

Ejercicio:

Escriba un algoritmo $O(\log_7(n))$ que sume enteros.

Ejercicios sobre máquinas de estados:

Para las siguientes tablas de máquinas de estados finitos, determine la secuencia de estados transitados y la salida de la cadena I dada si el estado inicial es el indicado, además determine los alfabetos de entrada y de salida así como el grafo de transición y la función de transición como lista de pares ordenados.

$init = s_1, I = aaacbbb \rightarrow$

$Estado \backslash In \rightarrow$	a	b	c
$s1$	$s2(x)$	$s3(x)$	$s1(y)$
$s2$	$s3(x)$	$s2(z)$	$s3(x)$
$s3$	$s1(x)$	$s2(z)$	$s3(z)$

,

$init = s_3, I = abacbcc \rightarrow$

$Estado \backslash In \rightarrow$	a	b	c
$s1$	$s2(x)$	$s3(x)$	$s1(y)$
$s2$	$s3(x)$	$s1(z)$	$s2(x)$
$s3$	$s1(x)$	$s2(z)$	$s3(z)$

,

$init = s_2, I = 01010011 \rightarrow$

$Estado \backslash In \rightarrow$	0	1
$s1$	$s2(x)$	$s3(x)$
$s2$	$s3(x)$	$s4(0)$
$s3$	$s1(x)$	$s1(0)$
$s3$	$s1(0)$	$s4(0)$

Ejercicio: Genere la máquina de estados finitos capaz de de automatizar el cobro de parqueo en un mall, cobrando 20Lps por estancia de a lo mucho 3 horas y 100Lps para los otros casos, capaz también de devolver cambio con billetes de 10,50,100 aceptando solo billetes de 20,50, 100 y 200 imprimiendo un menasje de error en otro caso.