Análisis de Algoritmos
Primer Problemario

Prof. Miguel A. Pizaña

24 de mayo de 2017

I Tareas

1. ¿Qué dice la paradoja de Zenón de Elea? ¿Qué significa "paradoja"? ¿Cómo se resuelve la paradoja de Zenón?

II Notación Asintótica.

1. Demostrar el criterio del límite:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty \text{ implica } f(n) = O(g(n))$$

2. Demuestre que

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \text{ con } 0 < c < \infty \text{ implica } f(n) = \Theta(g(n))$$

3. Demuestre que $f(n) = o(g(n))$ implica $f(n) = O(g(n))$.
4. Muestre que $f(n) = O(g(n))$ implica $|f(n)| + |g(n)| = \Theta(g(n))$.
5. Muestre que $f(n) = o(g(n))$ implica $g(n) \pm f(n) = \Theta(g(n))$.
6. Dar un ejemplo de funciones $T(n)$ y $A(n)$ tal que $T(n) = o(A(n))$, pero $A(n) = o(T(n+1))$. ¿Qué tiene que ver esto con Zenón, Aquiles y la Tortuga ?
7. Dar un ejemplo de funciones f y g tal que $f(n) \neq O(g(n))$ y $g(n) \neq O(f(n))$.
8. Dar un ejemplo tal que $f(n) = O(g(n))$, pero que f y g no cumplan con las hipótesis del criterio del límite.
9. Dar un ejemplo tal que $f(n) = O(g(n))$, $g(n) \neq O(f(n))$, pero $f(n) \neq o(g(n))$.

10. Suponga que tengo dos algoritmos A y B para resolver el mismo problema. El algoritmo A es de tiempo $O(n)$ y que el algoritmo B es de tiempo $O(n^2)$. ¿Esto significa que A es mejor que B? Explique. ¿Es posible que B sea *siempre* mejor que A? Explique.
11. Sea $T(n) = aT(n/b) + cn$, con $a \geq 1$, $b > 1$ y $c > 0$. Demuestre que:
$$T(n) = \begin{cases} \Theta(n) & \text{si } a < b \\ \Theta(n \log(n)) & \text{si } a = b \\ \Theta(n^{\log_b(a)}) & \text{si } a > b \end{cases}$$
12. Sean $a, b, c, d \in \mathbb{R}$ con $0 < a < b$ y $1 < c < d$. Para este problema, escribiremos $f \prec g$ en lugar de $f(n) = o(g(n))$. Muestre que: $\log(n)^a \prec \log(n)^b \prec n^a \prec n^a \log(n) \prec n^b \prec n^{\log(n)} \prec c^n \prec d^n \prec \log(n)^n \prec n! \prec n^n$
13. Muestre que $\log(n) = \Theta(\log(n+1))$.
14. Muestre que
$$\sum_{k=1}^{n} \frac{1}{k} = \Theta(\log(n))$$
15. Muestre que
$$\sum_{k=1}^{n} \log(k) = \Theta(n \log(n))$$

III Ordenación, Divide y vencerás.

1. Suponga que tengo dos algoritmos A y B para resolver el mismo problema. El algoritmo A es de tiempo $O(n)$ y que el algoritmo B es de tiempo $O(n^2)$. ¿Esto significa que A es más rápido que B? Explique. ¿Es posible que B sea *siempre* más rápido que A? Explique.
2. Sean $a, b > 1$. Suponga que tengo dos algoritmos A y B para resolver el mismo problema, el algoritmo A es de tiempo $\Theta(n^a)$ y el algoritmo B es de tiempo $\Theta(b^n)$. Muestre que $T_A(n) = o(T_B(n))$. ¿Esto significa necesariamente que A es mejor que B en la práctica? Explique.
3. Calcule asintóticamente (usando "$= \Theta$") el mejor tiempo, el peor tiempo y el tiempo promedio de los algoritmos de ordenación: Burbuja, MergeSort, QuickSort, RadixSort, BucketSort.

4. Muestre que $\Omega(n \log(n))$ es una cota inferior asintótica absoluta para cualquier algoritmo que ordene n elementos diferentes usando exclusivamente decisiones binarias. ¿Cómo es entonces que RadixSort y BucketSort pueden ordenar en tiempo lineal?

5. ¿Es posible ordenar n elementos en tiempo lineal? ¿Bajo que premisas? Explique.

6. Calcule una fórmula cerrada para los números de Fibonacci: $F(n) = F(n-1) + F(n-2)$ con $F(0) = 1$, $F(1) = 1$. Sugerencia: Suponga que existe una solución a la recurrencia de la forma $F(n) = q^n$, encuentre los dos valores q_1, q_2 que hacen que ese supuesto sea cierto, demuestre por inducción que cualquier combinación lineal de la forma $c_1 q_1^n + c_2 q_2^n$ es en efecto una solución a la recurrencia, use los valores iniciales ($F(0) = 1$, $F(1) = 1$) para determinar los valores exactos que c_1 y c_2 deben tener.

7. El tiempo que tarda, el algoritmo recursivo para calcular números de Fibonacci es $T_F(n) = T_F(n-1) + T_F(n-2) + 1$ con $T_F(0) = T_F(1) = 1$. Proporcione una fórmula cerrada exacta para el tiempo de ejecución de este algoritmo en términos de los números de Fibonacci $F(n)$. Sugerencia: primero itere la recurrencia, luego observe el patrón que emerge y finalmente pruebe su conjetura por inducción matemática.

8. Describa el método de Strassen para multiplicar matrices. ¿Cuál es el tiempo de ejecución del algoritmo obvio para multiplicar matrices? ¿Cuál es la complejidad del algoritmo de Strassen? ¿y la del algoritmo de Coppersmith-Winograd?

9. Considere el problema de multiplicar números enteros arbitrariamente grandes. Describa un algoritmo del tipo divide y vencerás para este problema. Cuál es la complejidad de los siguientes algoritmos: Multiplicación por suma iterada, el algoritmo de la primaria, divide y vencerás, Schönhage-Strassen. ¿Cuál es el mejor algoritmo conocido para multiplicar números enteros grandes? ¿Cuál es su tiempo de ejecución? ¿Quiénes son los autores de este algoritmo? ¿Qué técnica se utiliza en ese algoritmo?

10. Diseñe un algorimo divide y vencerás que dado un arreglo desordenado de n enteros (no necesariamente acotados) y un número k, $1 \leq k \leq n$, encuentre el k-ésimo elemento más chico en tiem-

po promedio lineal. (sugerencia: adapte QuickSort). ¿Puede ahora modificar su algoritmo para que también tenga tiempo lineal en el peor caso?

11. ¿En el contexto de diseño de algoritmos, qué significa "divide y vencerás"? Proporcione 3 ejemplos. Explique. ¿y que cosa es multiplica y perderás? Proporcione un ejemplo. ¿Cómo se hace para evitar este problema?

IV Programación Dinámica.

1. ¿Que es la programación dinámica? ¿para que sirve? ¿Cuándo se puede aplicar? Proporcione 3 ejemplos. Explique.
2. Proporcione un algoritmo que use programación dinámica para "arreglar" el algoritmo recursivo para los números de Fibonacci.
3. Proporcione un algoritmo polinomial para el problema de encontrar la parentización idónea para multiplicar una lista de matrices.
4. Proporcione un algoritmo polinomial para el problema de calcular la distancia entre palabras.
5. Vaya al juez en línea (https://uva.onlinejudge.org/), regístrese y resuelva el problema 100: "The 3n+1 problem". Envíe su solución. Asegúrese que el Juez la acepte.
6. Vaya al Juez en línea (https://uva.onlinejudge.org/) y resuelva el problema 116 "Unidireccional TSP". Envíe su solución y asegúrese que el Juez la acepte. ¿En que tiempo se ejecutó su algoritmo? ¿Cuál es el mejor tiempo registrado por el Juez en línea para ese problema? ¿¡Cómo es eso posible!?
7. Resuelva también los siguientes problemas del Juez en línea:
   10131 "Is Bigger Smarter?"
   10069 "Distinct Subsequences"
   10154 "Wheights and Measures"
   10003 "Cutting Sticks"
   10261 "Ferry Loading"
   10271 "Chopsticks"
   10201 "Adventures in Moving, part IV"