Tarea 1 Problemas conceptuales

Iván David Valderrama Corredor
Ingeniería de Sistemas y Ciencias de la Computación
Pontificia Universidad Javeriana, Cali

1 de febrero de 2019

Índice

1. Problemas conceptuales

1.1. Escribir el código de honor del curso

Como miembro de la comunidad académica de la Pontificia Universidad Javeriana Cali me comprometo a seguir los más altos estándares de integridad académica.

1.2. Ejercicios 2 y 4: Orden asintótico y eficiencia algorítmica (Kleinberg and Tardos páginas 67 y 68)

Ejercicio 2

Supongamos que tiene algoritmos con los seis tiempos de ejecución enumerados a continuación.(Suponga que este es el número exacto de operaciones realizadas en función del tamaño de entrada n.) Suponga que tiene una computadora que puede realizar 10^{10} operaciones por segundo, y necesita calcular un resultado como máximo una hora de cálculo. Para cada uno de los algoritmos, ¿cuál es el tamaño de entrada más grande n para el que podría obtener el resultado en una hora?
(a), n^2
(b), n^3
(c), $100n^2$
(d), $nlog(n)$
(e), 2^n
(f), 2^{2^n}

Respuesta

10^{10} operaciones por segundo, 1 hora tiene 60 minutos y 1 minuto tiene 60 segundos. Por lo que nuestro computador puede realizar 3.6x10^{13}(60x60x10^{10}) operaciones por hora.

$n^2 = 3{,}6 * 10^{13}$
$n = \sqrt{3{,}6x10^{13}}$
R//: n=6'000,000

$n^3 = 3{,}6 * 10^{13}$
$n = \sqrt[3]{3{,}6 * 10^{13}}$
R//: n=33,019

$100 * n^2 = 3{,}6 * 10^{13}$
$n^2 = \frac{3{,}6*10^{13}}{100}$
$n = \sqrt{\frac{(3{,}6*10^{13})}{100}}$
R//: n=600.000

$n * log(n) = 3{,}6 * 10^{13}$
Resultado generado mediante WolframAlpha:
R//: n=$1{,}29095 * 10^{12}$

Usando bisección:

```
import math

def nlogn(operations):
    low,hi = 0.0, 10e15
    while True:
        mid = (low+hi)/2
        if low == mid or mid == hi:
            return mid
        if mid*math.log(mid, 10) > operations:
            hi = mid
        else:
            low = mid

print(nlogn(3.6*10e13))
```

[1]
R//: n=$2{,}7 * 10^3$

$2^n = 3{,}6 * 10^{13}$
$log(2^n) = log(3{,}6 * 10^{13})$
$n * log(2) = log(3{,}6 * 10^{13})$
$n = \frac{log(3{,}6*10^{13})}{log(2)} = 45{,}033$

R//: n=45

$2^{2^n} = 3{,}6 * 10^{13}$
$log(2^{2^n}) = log(3{,}6 * 10^{13})$
$2^n * log(2) = log(3{,}6 * 10^{13})$

$2^n = \frac{log(3{,}6*10^{13})}{log(2)}$
$log(2^n) = log(\frac{log(3{,}6*10^{13})}{log(2)})$
$log(2) * n = log(\frac{log(3{,}6*10^{13})}{log(2)})$
$n = \frac{log(\frac{log(3{,}6*10^{13})}{log(2)})}{log(2)} = 5{,}492$
R//: n=5

Ejercicio 4

Tome la siguiente lista de funciones y organícelas en orden ascendente de la tasa de crecimiento. Es decir, si la función g(n) sigue inmediatamente a la función f(n) en su lista, entonces debería ser el caso de que f(n) sea O(g(n)).
$g_1(n) = 2\sqrt{log(n)}$
$g_2(n) = 2^n$
$g_3(n) = n(log(n))^3$
$g_4(n) = n^{4/3}$
$g_5(n) = n^{log(n)}$
$g_6(n) = 2^{2^n}$
$g_7(n) = 2^{n^2}$

Respuesta

Mediante desmos (https://www.desmos.com/calculator/) pude visualizar el comportamiento de cada función.
$g_1(n) = 2\sqrt{log(n)}$
$<$
$g_3(n) = n(log(n))^3$
$<$
$g_4(n) = n^{4/3}$
$<$
$g_5(n) = n^{log(n)}$
$<$
$g_2(n) = 2^n$
$<$
$g_7(n) = 2^{n^2}$
$<$
$g_6(n) = 2^{2^n}$

1.3. Ejercicio 6:Array Sums(Kleinberg and Tardos página 68 y 69).

Considere el siguiente problema básico. Se le da una matriz A que consta de n enteros A [1], A [2], ..., A [n]. Le gustaría generar una matriz bidimensional n-por-n en la que B [i, j] (para i <j) contiene la suma de las entradas de matriz A [i] a A [j]. la suma A [i] + A [i + 1]] ... + A [j]. (El valor de la entrada de la matriz B [i, j] se deja sin especificar siempre que i>= j, por lo que no importa cual es la salida de estos valores.)

(a) Para alguna función f que debe elegir, indique un límite de la forma O (f (n)) en el tiempo de ejecución de este algoritmo en una entrada de tamaño n (es decir, un límite en el número de operaciones realizadas por el algoritmo) .

Respuesta

```
For i=1, 2,... n-----------------------------------------------|
  For j= i+1, i+2,...n-------------------------------|         |
    Add up array entries A[i] through A[j]--->[n] |>[n]  |>[n]
    Store the result in B[i,j] ---------------->[1] |         |
  Endfor-------------------------------------------|         |
Endfor-------------------------------------------------------|
```

n (ciclo principal)*n(ciclo inmerso)*(n(añadir entradas desde A[i] hasta A[j] y guardar resultados)
Por lo que el peor de los casos seria O(n^3)

(b)Para esta misma función f, muestre que el tiempo de ejecución del algoritmo en un tamaño de entrada n también es Ω(f(n)).(Esto muestra un límite estrechamente asintótico de Θ(f(n)) en el tiempo de ejecución).

Respuesta
El anteroir algoritmo se puede ver de la forma $n * (\sum_{i=1}^{n} i + 1)$
$n * (\frac{(n+1)*(n+2)}{2})$
$(\Omega) = \frac{n^3+3n^2+2n}{2}$
por lo tanto $F(n) \in O(n^3)$ $F(n) >= g(n) * c$
$F(n) >= n^3 * c$
$F(n) \in \Omega(n^3)$
el c que serviria en este caso seria $\frac{1}{2}$
$\frac{n^3+3n^2+2*n}{2} >= \frac{n^3}{2}$
$n^3 + 3n^2 + 2 * n >= n^3$

$3n^2 + 2 * n >= 0$
para $n >= 1$

(c)Aunque el algoritmo que se analizó en las partes (a) y (b) es la forma más natural de resolver el problema, simplemente itera a través de las entradas relevantes de la matriz B, completando un valor para cada una de ellas. Proporcione un algoritmo diferente para resolver este problema, con un tiempo de ejecución mejor asintótico. En otras palabras, debe diseñar un algoritmo con tiempo de ejecución O(g(n)), donde $\lim_{n\to\infty} g(n)/f(n) = 0$.

Respuesta
Una manera de optimizar el codigo, sería a través de una suma que llevara A[i] hasta A[j], ya que nos ahorrariamos el comando (Add up array) que tiene un costo de n. logrando asi unicamente el costo de los 2 ciclos que seria O(n^2)

```
 sumaActual = 0

For i=1, 2, ..., n

    sumaActual = A[i]

     For j = i+1, i+2, ..., n

         sumaActual = sumaActual + A[j]

          Store the sumaActual in B[i, j]

     Endfor

Endfor
```

[2]

1.4. Ejercicio 7:Folk Songs(Kleinberg and Tardos página 69).

Hay una clase de canciones populares y festivas en las que cada verso consta del verso anterior, con una línea adicional agregada. "Los doce días de Navidad"tiene esta propiedad; por ejemplo, cuando llegas al quinto verso, cantas sobre los cinco anillos de oro y luego, repitiendo las líneas del cuarto verso, también cubres los cuatro pájaros que llaman, las tres gallinas francesas, las dos palomas tortugas y, por supuesto, el Perdiz

en el peral. La canción aramea "Had gadya"de PassoVer Haggadah también funciona así, al igual que muchas otras canciones. Estas canciones tienden a durar mucho tiempo, a pesar de tener guiones relativamente cortos. En particular, puede transmitir las palabras más instrucciones para una de estas canciones especificando solo la nueva línea que se agrega en cada verso, sin tener que escribir todas las líneas anteriores cada vez. (Por lo tanto, la frase "[cinco anillos de oro]"solo se debe escribir una vez, aunque aparecerá en los versículos cinco y adelante). Hay algo asintótico que se puede analizar aquí. Supongamos, para concretar, que cada línea tiene una longitud que está delimitada por una constante "[c]", y supongamos que la canción, cuando se canta en voz alta, corre para "n"palabras en total. Muestra cómo codificar una canción de este tipo utilizando un guión que tiene la longitud f(n), para una función f(n) que crece lo más lentamente posible.

Respuesta

```
For  i=1,2,...,n
    For  j=1,...,i
        reproducirLinea(j)
    Endfor
Endfor
```

Para hallar la complejidad del algoritmo tendremos en cuenta el comportamiento de los ciclos, los cuales se pueden ver como una sumatoria ($\sum_{i=1}^{n} i$) la cual es una progresión aritmética que por medio de la formula de "[n primeros numeros naturales]"podemos verla como ($\frac{n*(n+1)}{2}$). Teniendo en cuenta el total de palabras (w), tendriamos ($\frac{n*(n+1)}{2} <= w$) para hacer mas facil el calculo del limite, $\frac{n^2}{2} <= \frac{n*(n+1)}{2} <= w$
$\frac{n^2}{2} <= w$
$n^2 <= 2 * w$
$n <= \sqrt{2 * w}$
por lo que la complejidad es de $O\sqrt{n}$

1.5. Ejercicio 13:Inversions(Erickson, Capítulo 1).

Una inversión en la matriz A [1...n] es un par de índices (i, j) tales que i <j y A [i]>A [j]. El número de inversiones en una matriz de n elementos está entre 0 (si la matriz está ordenada) y $\binom{n}{2}$ (si la matriz está ordenada hacia atrás). Describa y analice un algoritmo para calcular el número de inversiones en una matriz de n elementos en O(n*log(n)) tiempo. [Sugerencia: modificar mergesort.]

Respuesta
En la funcion del merge, podemos poner un contador(i) que nos permita recorrer la

matriz izquierda y otro contador(j) para recorrer la sub-matriz derecha. Cuando se da el proceso de combinacion en el merge, si A[i] es mayor que A[j], entonces hay (len(arreglo)//2 - i) inversiones. Esto es debido a que los subarreglos de la izquierda y la derecha están ordenados.

1.6. Ejercicios 4a, 4b y 13: Demostraciones por inducción (Erickson, Apéndice I.1)

Ejercicio 4a y 4b Recuerde la definición recursiva estándar de los números de fibonacci: $F_0 = 0$, $F_1 = 1$, y $F_n = F_{n-1} + F_{n-2}$ para todos $n >= 2$. Demuestre las siguientes identidades para todos los enteros no negativos n y m.

(a)$\sum_{i=0}^{n} F_i = F_{n+2} - 1$

Respuesta
Caso base: Probamos para n=0, por el lado izquierdo tenemos que $F_0 = 0$ y por el lado derecho es $F_2 = 0$, los 2 lados son iguales por lo tanto es verdadero y se cumple para n=0.
hipotesis inductiva: $\sum_{i=0}^{k} F_i = F_{k+2} - 1$
... $= F_{k+2+1} - 1$
... $= F_{k+3} - 1$
Paso inductivo: Tenemos un $k \in \mathbb{N}$ y suponemos que $k = n$ es verdad, luego:
$\sum_{i=0}^{k+1} F_i = \sum_{i=0}^{k} F_i + F_{k+1}$
... $= F_{k+2} - 1 + F_{k+1}$ (Por hipotesis inductiva con n=k)
... $= F_{k+3} - 1$ (Por la recurrencia de F_n)
Esto se cumple para n= k + 1 por lo que podemos concluir que mediante el principio de induccion, se cumple para todos los n tal que $n \in \mathbb{N}$.
[3]

(b)$F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$

Respuesta
Caso base: Probamos para n=1, por el lado izquierdo tenemos que $F_1^2 - F_2F_0 = 1$ y por el lado derecho es $(-1)^{1+1} = 1$, los 2 lados son iguales por lo tanto es verdadero y se cumple para n=1.
Tenemos un $k \in \mathbb{N}$ y suponemos que $k = n$ es verdad, luego:
$(-1)^{k+2}$

Ejercicio 13
El hipercubo d-dimensional es el gráfico definido a continuación. Hay 2d vértices, cada

uno etiquetado con una cadena diferente de d bits. dos vértices están unidos por un borde si y solo si sus etiquetas difieren exactamente en un bit.
Recuerde que un ciclo hamiltoniano es una caminata cerrada que visita cada vértice en una gráfica exactamente una vez. Probar que para cada entero d >=, el hipercubo d-dimensional tiene un ciclo hamiltoniano

Respuesta

1.7. Citación de bibliografía

Referencias

[1] liori, *How to calculate n log n = c*. https://stackoverflow.com/questions/3847327/how-to-calculate-n-log-n-c.

[2] pdiniz, *Algorithms Design – Chapter 2, Exercise 6*. https://itsiastic.wordpress.com/2013/07/19/algorithms-design-chapter-2-exercise-6/.

[3] illinois, *Practice Problems—Solutions*. https://faculty.math.illinois.edu/ hilde-br/347.summer14/induction2sol.pdf