Análisis de Algoritmos, Sem: 2018-1, 3CV2 Práctica 1, 24-08-2017

Práctica 1: Determinación experimental de la complejidad de un algoritmo

Salgado Alarcon Genaro, Padilla Calderon Jose Manuel
Escuela Superior de Cómputo
Instituto Politécnico Nacional
isomaelking@gmail.com, genaro_yen13@hotmail.com

Primero

En este trabajo vamos a analizar, por medio de la experimentación, la complejidad de un algoritmo dado, proponiendo una función g(n) tal que Suma $\in$ O(g(n)) y g(n) sea mínima, en el sentido de que si Suma $\in$ O(h(n)), entonces g(n) $\in$ O(h(n)). Obserando como se comporta, se compará el comportamiento entre el tiempo de ejecución y un número de veces que se ejecuta la llamada a la función.

Palabras Clave

- Algoritmo
- Funcion
- Experimentacion
- Eficiencia

1. Introducción

Todos los días nos enfrentamos a problemas en los que usamos un algoritmo para resolverlos, es importante observar que quiza no estemos empleando la mejor solucion a estos problemas. Tanto en la vida cotidiana como en la industria se usan algoritmos que nos ayudan a optimizar nuestro trabajo, trabajo que significa esfuerzo y dinero en algunas ocaciones, por lo tanto es importante el estudio de estos algoritmos, en este trabajo, por medio de la experimentacion vamos a analizar que tan eficiente y complejo puede ser un algoritmo, para así poder encontrar una mejor soluciona este lo que puede significar ahorrar tanto tiempo, como esfuerzo y quiza dinero.

2. Conceptos Básicos

Para la correcta comprension de este trabajo, es necesario definir algunos terminos tales como θ, O y Ω.
θ(n):
Sea g(n) una función. Se define θ (g(n)) como:

$\theta(\text{g(n)}) = \{f(n) \quad | \quad \exists c1, c2 > 0 \quad \& \quad n_0 > 0 \quad | \quad \forall n >= n_0 \quad 0 <= c1g(n) <= f(n) <= c2g(n)\}$

O(n):
Sea g(n) una función, O(n) se define como:

$$\text{O(n)}=\{f(n) \quad | \quad \exists c > 0 \quad \& \quad n_0 > 0 \quad | \quad f(n) <= Cg(n) \quad \forall n >= n_0\}$$

Ω(n):
Sea g(n) una función. Se define Ω (g(n)) como:

$$\Omega(\text{g(n)}) =\{f(n) \quad | \quad \exists c > 0 \quad \& \quad n_0 > 0 \quad | \quad 0 <= cg(n) <= f(n) \quad \forall n >= n_0\}$$

Se muestra experimentalmente el tiempo computacional de dos algoritmos y un tercero de manera teórica, el primer algoritmo de desarrollo experimental consiste en sumar dos números enteros en notación binariaconsiderando que: Dos arreglos unidimensionales A de tamaño n y B de tamaño m con k = $\log_2(n)$ y t = $\log 2(m)$ almacenarán los números a sumar. La suma se almacenará en un arreglo C. El propósito es obtener el tiempo que se tarda en realizar dicha operación con un tamaño r que se le asigna al arreglo.

Figura 1

La Figura 1 ejemplifica la operación que se realiza en una suma binaria, se toma las siguientes cuestiones:

- 0+0=0
- 0+1=1
- 1+0=1
- 1+1=0 y se lleva uno

El segundo algoritmo experimental consiste en implementar el algoritmo de Euclides para encontrar el mcd de dos números enteros positivos m y n. El propósito al igual que el anterior algoritmo es obtener el tiempo de ejecución cuando se mande a llamar la función de Euclides, pero con la condición que los números a los que se obtendrá el m.c.d serán los que se obtienen de la serie de Fibonacci, los cuales se obtienen de la suma de los dos números que le preceden 1 ,1,2,3,5,8,13,21,34 ,etc, se toma por ejemplo el 8,13 y se llama a la función Euclides(8,13) el cual dará como resultado un número "n" tal que 8 %n=0 y 13 %n=0.

Para el algoritmo teórico se encontrará la complejidad de un programa de ordenamiento.

3. Experimentación y Resultados

3.1. Suma Binaria

-Pseudocódigo Suma Binaria

Suma (A,B,n[1,...,17],c)
Entrada: i[1,..., 17]
Salida: Tiempo computacional de n[1,...,17]
1. for i ← 1 to i ← 17 do
2. n ← potencia(2,i)
3. suma(A,B,n,c)
4. print(c)

-Mostrar gráficas para la función Suma que muestre tiempo vs r con r = m = n (considere diversos valores de r).

Figura 2

La tabla p[ara esta grafica es:

Figura 3

-Proponer una función g(n) tal que S ∈ O(g(n)) y g(n) sea mínima, en el sentido de que si S ∈ O(h(n)), entonces g(n) ∈ O(h(n)).

Por los resultados obtenidos en la gráfica proponemos que la complejidad del algoritmo S es de tipo lineal, y se propone la función O(n).

-Conclusiones

Genaro Salgado Alarcon:.
A mi perspectiva el ejercicio con los temas vistos en clase fue algo sencillo de analizar, porque si, cuando lo leímos por primera vez en verdad se complico un poco el entenderlo, pero, si tienes los apuntes completos y los consultas mientras desarrollas el ejercicio, se hará mas sencillo de entender, al igual que con la suma binaria, tuvimos que consultar los apuntes para entenderlo por completo
Padilla Calderon Jose Manuel:.
La suma fue un poco complicado complicada, al menos un poco mas que el de Euler si, tanto en el nivel de análisis como en programación, porque entender exactamente lo que se solicitaba era algo que nos preocupaba para cumplir completamente con la practica, nuestro primer problema fue que terminábamos el script y nos dábamos cuenta que teníamos que hacer otras cosas, entonces, obviamente teníamos que completar el script.

3.2. Euclides

1.- Pseudocódigo Algoritmo de Euclides

Euclides (A,B,c]
Entrada: fibonacci(n),fibonacci(n-1) con $1 <= n < 50$
Salida: Tiempo computacional de $1 <= n < 50$
1. for i ← 1 to i ← 79 do
2. a ← fibonacci(i)
3. b ← fibonacci(i+1)
4. e ← euclides(b,a,c)
5. print(c)

-Mostrar gráficas para la función Euclides que muestre tiempo vs diferentes valores de m y n.

Para valores del fibonacci de 1 a 79

Figura 5

La gráfica es:

Figura 6

-Proponer una función g(n) tal que Euclides $\in$ O(g(n)) y g(n) sea mínima, en el sentido de que si Euclides $\in$ O(h(n)), entonces g(n) $\in$ O(h(n)).

Por los resultados obtenidos en la última gráfica proponemos que la complejidad del algoritmo Euclides es de tipo logarítmica y propones la función $O(\log_2(n))$ ya que $O(\log_2(n))$ pasa ser cota superior cumpliendo así la definición de $O(\log_2(n))$.

-Conclusiones

Genaro Salgado Alarcon:. El ejercicio de Euler se nos complico en el inciso de Fibonacci porque estábamos programando todo para poder compararlo, pero nos dimos cuenta que teníamos que analizar lo que nos pedía que era entender el mcd y su relación con la serie de fibonacci mas no el código, después de entender esto se facilito el análisis teórico, ya con esto pudismos completar el ejercicio.

Padilla Calderon Jose Manuel:.

Este ejercicio fue un poco mas sencillo al momento de programar ya que era un problema sencillo, peor en el análisis nos fue un poco mas complicado resolverlo ya que su curva fue logarítmica según los resultados que obtuvimos, no fue tan sencillo como en la suma donde todo era lineal, es importante mencionar que el pseudocodigo que vimos en el pdf de la practica nos fue de gran ayuda para empezar a programar y entender.

4. Conclusión General

Esta primer practica como ya lo hemos explicado a lo largo del desarrollo de esta, nos ha presentado una nueva visión no solo de los algoritmos sino de la programación en general, es cierto que hoy en día existen muchísimas maneras para hacer un correcto programa, tanto documentación como otras, que radican desde el logaritmo mismo, esta materia nos ha abierto el panorama a los primeros métodos de análisis de algoritmos, para así, más tarde poder decir si nuestro algoritmo es válido o no, la practica nos presentó un tema de digitales, el cual nos fue más fácil de lo normal ya que estos temas no nos son ajenos, el mayor problema fue el de entender las características del reporte para que cumpliera con los rubros de la entrega.

5. Anexos

Select-Sort

Select-Sort(A[0...n-1])	Costo	#Pasos
1.- for j $\leftarrow$ 0 to n-2 do	C1	n
2.- k $\leftarrow$ j	C2	n-1
3.- for i $\leftarrow$ j+1 to n-1 do	C3	$\sum_{i=0}^{n-1} t_i$
4.- if A[i] < A[k] then	C4	$\sum_{i=0}^{n-1}(t_i - 1)$
5.- k $\leftarrow$ i	C5	$\sum_{i=0}^{n-1} R_i$
6.- intercambia(A[j],A[k])	C6	n-1

El peor de los casos se presenta cuando el arreglo esta ordenado en forma decreciente por lo que:
$t_i = n - i$ y
$R_i = n - i - 1 = t_i - 1$

Por lo tanto tenemos que:
T(n) = C1n + C2(n-1) + C3($\sum_{i=0}^{n-1}(n-i)$) + C4($\sum_{i=0}^{n-1}(n-i-1)$) + C5($\sum_{i=0}^{n-1}(n-i-1)$) + C6(n-1)

T(n) = C1n + C2n -C2 + C3($\sum_{i=0}^{n-1} n$) - C3($\sum_{i=0}^{n-1} i$) + C4($\sum_{i=0}^{n-1} n$) - C4($\sum_{i=0}^{n-1} i$) - C4($\sum_{i=0}^{n-1} 1$) + C5($\sum_{i=0}^{n-1} n$) -C5 ($\sum_{i=0}^{n-1} i$) - C5($\sum_{i=0}^{n-1} 1$) + C6n -C6

T(n) = (C1+C2+C6)n + (C3+C4+C5)(n^2) - (C3+C4+C5)($\sum_{i=1}^{n}(i-1)$) - (C4+C5)n -(C2+C6)

T(n) = (C3+C4+C5)n^2 -(C3+C4+C5)($\frac{n(n+1)}{2}$ - n) +(C1+C2-C4-C5+C6)n - (C2+C6)

T(n) = ($\frac{C3+C4+C5}{2}$)n^2 + (C1+C2+C3+C6)n - (C2+C6)

$\therefore T(n) \quad \in \quad O(n^2)$

6. Bibliografía

- Brassard, G. (1997). Fundamentos de Algoritmia. España: Ed. Prentice Hall. ISBN 848966000X
- Harel, D. (2004). Algorithmics: The spirit of Computing (3rd. Ed). Estados Unidos de América: Addison Wesley. ISBN-13: 978-0321117847