

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

Introduccion

En este trabajo vamos a analizar, por medio de la experimentación, la complejidad de un algoritmo dado, proponiendo una función $g(n)$ tal que $\text{Suma} \in O(g(n))$ y $g(n)$ sea mínima, en el sentido de que si $\text{Suma} \in O(h(n))$, entonces $g(n) \in O(h(n))$. Observando como se comporta, se comparará 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 quizá no estemos empleando la mejor solución 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 ocasiones, por lo tanto es importante el estudio de estos algoritmos, en este trabajo, por medio de la experimentación vamos a analizar que tan eficiente y complejo puede ser un algoritmo, para así poder encontrar una mejor solución a este lo que puede significar ahorrar tanto tiempo, como esfuerzo y quizá dinero.

2. Conceptos Básicos

Para la correcta comprensión de este trabajo, es necesario definir algunos términos tales como θ , O y Ω .

$\theta(n)$:

Sea $g(n)$ una función. Se define $\theta(g(n))$ como:

$$\theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0 \ \& \ n_0 > 0 \mid \forall n \geq n_0 \ 0 <= c_1 g(n) <= f(n) <= c_2 g(n)\}$$

$O(n)$:

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

$$O(n) = \{f(n) \mid \exists c > 0 \ \& \ n_0 > 0 \mid f(n) \leq Cg(n) \ \forall n \geq n_0\}$$

$\Omega(n)$:

Sea $g(n)$ una función. Se define $\Omega(g(n))$ como:

$$\Omega(g(n)) = \{f(n) \mid \exists c > 0 \ \& \ n_0 > 0 \mid 0 \leq cg(n) \leq f(n) \ \forall n \geq n_0\}$$

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 binaria considerando 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 \leftarrow 1 to i \leftarrow 17 do
2. n \leftarrow potencia(2,i)
3. suma(A,B,n,c)
4. print(c)

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

Figura 2

La gráfica $x=y$ es:

Figura 3

Ahora nuestra gráfica con los datos obtenidos es:

Figura 4

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

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

-Conclusiones

En este experimento.....

3.2. Euclides

1.- Pseudocódigo Algoritmo de Euclides

Euclides (A,B,c]

Entrada: fibonacci(n),fibonacci(n-1) con $1 \leq n < 50$

Salida: Tiempo computacional de $1 \leq n < 50$

1. for $i \leftarrow 1$ to $i \leftarrow 79$ do
2. $a \leftarrow \text{fibonacci}(i)$
3. $b \leftarrow \text{fibonacci}(i+1)$
4. $e \leftarrow \text{euclides}(b,a,c)$
5. print(c)

-Mostrar diversas 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

Padilla Calderon Jose Manuel:

Salgado Alarcon Genaro:

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:

$$t_i = n - i \text{ y}$$

$$R_i = n - i - 1 = t_i - 1$$

Por lo tanto tenemos que:

$$T(n) = An + B(n-1) + C(\sum_{i=0}^{n-1} (n-i)) + D(\sum_{i=0}^{n-1} (n-i-1)) + E(\sum_{i=0}^{n-1} (n-i-1)) + F(n-1)$$

$$T(n) = An + Bn - B + C(\sum_{i=0}^{n-1} n) - C(\sum_{i=0}^{n-1} i) + D(\sum_{i=0}^{n-1} n) - D(\sum_{i=0}^{n-1} i) - D(\sum_{i=0}^{n-1} 1) + E(\sum_{i=0}^{n-1} n) - E(\sum_{i=0}^{n-1} i) - E(\sum_{i=0}^{n-1} 1) + Fn - F$$

$$T(n) = (A+B+F)n + (C+D+E)(n^2) - (C+D+E)(\sum_{i=1}^n(i-1)) - (D+E)n - (B+F)$$

$$T(n) = (C+D+E)n^2 - (C+D+E)(\frac{n(n+1)}{2} - n) + (A+B-D-E+F)n - (B+F)$$

$$T(n) = (\frac{C+D+E}{2})n^2 + (A+B+C+F)n - (B+F)$$

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

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