Práctica 1: Determinación experimental de la complejidad temporal de un algoritmo

Hernández Castellanos César Uriel, Aguilar Garcia Mauricio

Escuela Superior de Cómputo
Instituto Politécnico Nacional, México
*uuriel*12009*u*@*gmail.com*, *mauricio.aguilar.garcia*,90@*gmail.com*

Resumen: El trabajo consiste en mostrar la importancia del análisis de algoritmos, así como la manera experimental de obtener su complejidad.
Palabras Clave: Algoritmo, Complejidad, Euclides, Suma binaria.

1. Introducción

Los algoritmos son de gran importancia tanto en la computación como en nuestra vida diaria, estamos rodeados de ellos, como su definición lo dice: .Es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite llevar a cabo una actividad mediante pasos sucesivos que no generen dudas a quien deba hacer dicha actividad."[1]

El análisis de algoritmos se utiliza para conocer la eficacia con la cual el algoritmo que se está analizando realiza su función, esta eficacia se mide respecto al tamaño de la entrada a la que se le ingrese el algoritmo y así obteniendo una función que permita generalizar el tiempo ocupado para terminar su proceso.
Es de gran importancia el análisis, puesto que permite escoger de manera eficaz (y ahorrando la mayor cantidad de recursos posibles) el algoritmo a utilizar dado un problema a resolver.

El objetivo de la práctica es reafirmar los conocimiento vistos en clase sobre los métodos de análisis de algoritmos y en específico para esta practica se espera obtener los resultados de manera experimental.

2. Conceptos básicos

Definición de algoritmo

Un algoritmo es una secuencia de pasos lógicos necesarios para llevar a cabo una tarea especifica, como la solución de un problema. Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta.

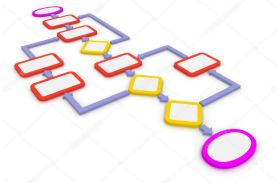

Figura 1: Algoritmo

Algoritmo de Euclides

El algoritmo de Euclides es un método antiguo y eficaz para calcular el máximo común divisor (MCD). Fue originalmente descrito por Euclides en su obra Elementos. El algoritmo de Euclides extendido es una ligera modificación que permite además expresar al máximo común divisor como una combinación lineal. Este algoritmo tiene aplicaciones en diversas áreas como álgebra, teoría de números y ciencias de la computación, entre otras. Con unas ligeras modificaciones suele ser utilizado en computadoras electrónicas debido a su gran eficiencia.

Figura 2: Euclides

Algoritmo de ordenamiento por selección

Consiste en encontrar el menor de todos los elementos del arreglo o vector e intercambiarlo con el que está en la primera posición. Luego el segundo mas pequeño, y así sucesivamente hasta ordenarlo todo. Su implementación requiere $O(n^2)$ comparaciones e intercambios para ordenar una secuencia de elementos. Su funcionamiento se muestra a continuación.

Algoritmo para sumar dos números binarios

Para hacer la suma binaria se suma columna por columna los ceros o unos y se lleva el acarreo dependiendo del resultado de la suma a continuacion se usa un ejemplo en al fig 3 en el cual se demuestra de color negro las pocas operaciones que se usan al sumar bit por bit y como afecta el acarreo:

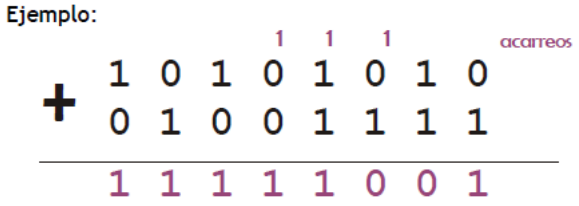

Figura 3: Ejemplo suma binaria

Notación Θ

En análisis de algoritmos una cota ajustada asintotica es una función que sirve de cota tanto superior como inferior de otra función cuando el argumento tiende a infinito. Usualmente se utiliza la notación $\Theta(g(x))$ para referirse a las funciones acotadas por la función $g(x)$.[2]

Notación O

En análisis de algoritmos una cota superior asintotica es una función que sirve de cota superior de otra función cuando el argumento tiende a infinito.

Usualmente se utiliza la notación de Landau: $O(g(x))$, Orden de $g(x)$, coloquialmente llamada Notación O Grande, para referirse a las funciones acotadas superiormente por la funcion $g(x)$.[2]

Notación Ω

En análisis de algoritmos una cota inferior asintotica es una función que sirve de cota inferior de otra función cuando el argumento tiende a infinito. Usualmente se utiliza la notación $\Omega(g(x))$ para referirse a las funciones acotadas inferiormente por la función $g(x)$.[2]

3. Experimentación y Resultados

Problema uno

Desarrollar e implementar un algoritmo Suma que sume dos enteros en notacion binaria bajo las siguientes consideraciones: Dos arreglos unidimensionales A de tama˜no n y B de tamaño m con k = log2(n) y t = log2(m) almacenarán los números a sumar. La suma se almacenará en un arreglo C.

Algorithm 1 Algoritmo para sumar dos números binarios

```
1: procedure ADDBINARYNUMBER(A, B)
2:     sum ← false
3:     carry ← false
4:     for i ← 0, nbits do
5:         constant ← nbits − i
6:         bitOne ← A[constant]
7:         bitTwo ← B[constant]
8:         if carry then
9:             if bitOne && bitTwo then
10:                carry ← true
11:                sum ← true
12:            else if bitOne || bitTwo then
13:                carry ← true
14:                sum ← false
15:            else
16:                carry ← false
17:                sum ← true
18:            end if
19:        else
20:            if bitOne && bitTwo then
21:                carry ← true
22:                sum ← false
23:            else if bitOne || bitTwo then
24:                carry ← false
25:                sum ← true
26:            else
27:                carry ← false
28:                sum ← false
29:            end if
30:        end if
31:        if sum then
32:            C[constant] ← true
33:        end if
34:    end for
35: end procedure
```

El algoritmo desarrollado anteriormente nos realiza la suma de dos números binarios de n bits, se hace uso de dos banderas (sum y carry), en primer lugar iteramos de uno en uno hasta el número de bits de nuestro par de arreglos. Después se comprueba si carry es verdadero, en tal caso que se cumpla realizaremos las condiciones para los diferentes casos, en el caso de que el carry sea falso de igual forma se realizan condiciones para cada caso, pero con el cambio de que las banderas tomarán un distinto valor. Al implementar el algoritmo se le

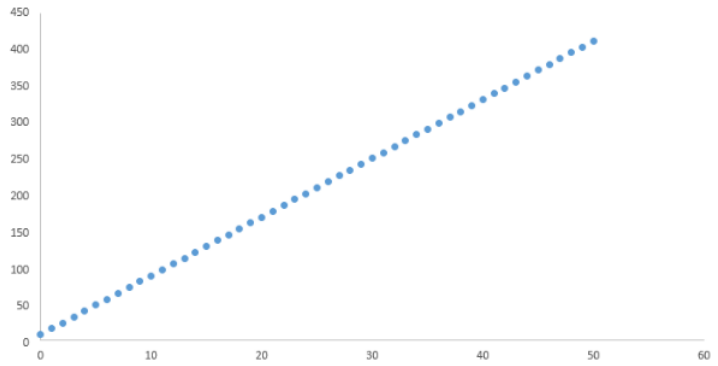

Figura 4: Gráfica del algoritmo de suma de dos números binarios

agregó una variable k para cada linea, con el fin de poder obtener el orden de complejidad por el método de linea por linea, por lo que nuestro resultado fue el mostrado en la figura 4

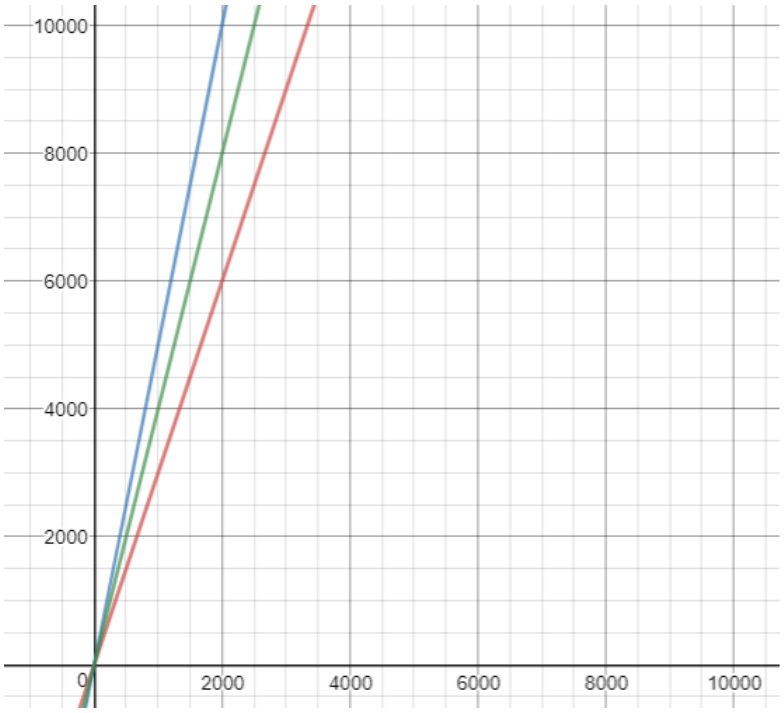

Figura 5: Funciones propuestas f1(n) = 5n y f2(n) = 3n

En la figura 5 se muestran las funciones propuestas que acotan como por la parte inferior y superior a la función que se obtuvo de manera experimental

```
run:
Array  A:  1011001000000000001100111101000101010000111010001100000001000100
Array  B:  1011000000000011011111000010111010000010101001010000100100011100
Array  C: 10110001000000011101011111111111111101001110001101110010010110000
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 6: Ejecucion del algoritmo de suma de dos números binarios

En la figura 6 se muestra la ejecución del programa con dos arreglos de 64 bits que fueron poblados de manera aleatoria.

Problema dos

Implementar el algoritmo de Euclides para encontrar el mcd de dos números enteros positivos m y n.

Algorithm 2 Algoritmo de Euclides

```
1: procedure EUCLIDES(m, n)
2:     while n ≠ 0 do
3:         m mód n
4:         m ← n
5:         n ← r
6:     end while
7: end procedure
```

El algortimo de euclides ya que se encarga el calcular el maximo comun divisor,como se observa en la figura 7 recibe 2 valores y nos devuelve su MCD con los pasos que hizo el algoritmo para mostrar los resultados en cada linea en la que hubo una operación.

```
Ingresa el primer valor: 4181
Ingresa el segundo valor: 6765
MCD: 1
Tiempo: 76
Seguir: 0
Ingresa el primer valor: 5
Ingresa el segundo valor: 10
MCD: 5
Tiempo: 8
Seguir:
```

Figura 7: Ejecucion algoritmo de euclides

Al implementar el algoritmo ingresamos el peor de sus casos el cual seria la sucesion de fibonacci en el cual se puede apreciar en la figura 8 en la cual se aprecia de color azul el peor caso y con naranja un caso para acotarlo en el cual tambien es la función logaritmica $4\log_2(x^2)$.

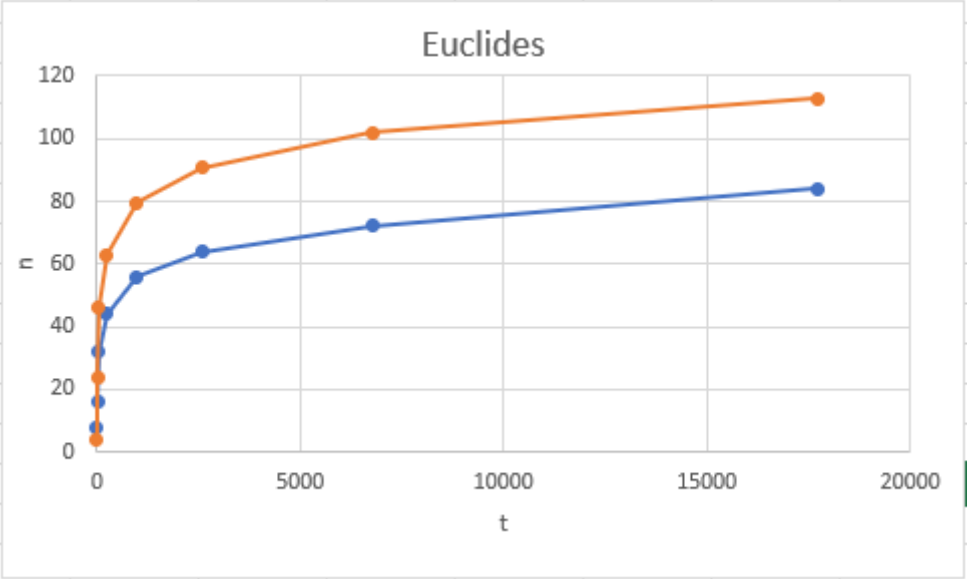

Figura 8: Resultados algoritmo de Euclides

4. Conclusiones

Conclusión general

Para cada algoritmo, son interesantes un par de medidas de complejidad: en el peor caso (que señalamos con la notación O, y en el mejor, para el que utilizamos la notación omega. En especial, suele ser más imprescindible conocer el peor caso, ya que nos da una idea de qué es lo que puede pasar cuando las cosas van realmente mal.

Es de gran importancia diseñar nuestros algoritmos de manera adecuada para evitar el uso de recursos computacionales de manera innecesaria.

Aguilar Garcia Mauricio

En esta práctica me costo mucho trabajo calcular la complejidad del algorimo de Euclides debido a como funciona el modulo, también me sirvió para repasar los temas y me ayudo a comprender la importancia de analizar la complejidad de los algoritmos.

Me permitió pensar en como se pueden llegar a generar el peor de los casos y lo útil que es, saber como generar los casos de prueba correctos, lo que nos permite checar el tiempo que deberia de tomar en correr de acuerdo con su complejidad.

Hernández Castellanos César Uriel

El algoritmo de Euclides, a pesar de su edad, es un algoritmo que prueba su valor gracias al buen rendimiento que tiene, ya que su complejidad no es la peor de este tipo de algoritmos en el caso del algoritmo de Euclides nos resultó una función logarítmica, además se propuso una función que acotara por la parte superior a la función obtenida de manera experimental.

El algoritmo para la suma de dos números binarios que me tocó implementar no representó mayor problema al implementarlo, excepto en el acarreo del último bit que fue donde se me presentaron algunas dificultades. El algoritmo que implementé resultó tener complejidad lineal además, se propusieron dos funciones que acotaron a la función obtenida tanto en la parte inferior como superior.

La práctica me permitió conocer la importancia que debemos de tener como ingenieros en diseñar nuestros algoritmos, ya que un mal planteamiento podría conllevar a un mayor costo computacional innecesario.

5. Anexo

El siguiente algoritmo, es un algoritmo de ordenamiento llamado por selección (SelectSort(A)). El peor caso se presenta cuando A está ordenado de manera decreciente.

for $j \leftarrow 0$ to $j \leq n - 2$ do	c_1	n
$k \leftarrow j$	c_2	$n-1$
for $i \leftarrow j + 1$ to $i \leq n - 1$ do	c_3	$\sum_{i=0}^{n-1} t_i$
if A[i] $\leq$ A[k] then	c_4	$\sum_{i=0}^{n-1} (t_i - 1)$
$k \leftarrow i$	c_5	$\sum_{i=0}^{n-1} (t_i - 1)$
Intercambia(A[j],A[k])	c_6	$n-1$

Sea t_i la cantidad de veces que se ejecuta el for interno.
Se tiene que

i	t_i
1	n–1
2	n–2
3	n–3
4	n–4
i	n–i

$\Rightarrow t_i = n - i$

$$T(n) = c_1 n + c_2(n-1) + c_3 \sum_{i=0}^{n-1} t_i + c_4 \sum_{i=0}^{n-1} (t_i - 1) + c_5 \sum_{i=0}^{n-1} (t_i - 1) + c_6(n-1)$$

sustituyendo t_i

$$T(n) = c_1 n + c_2(n-1) + c_3 \sum_{i=0}^{n-1} (n-i) + c_4 \sum_{i=0}^{n-1} (n-i-1) + c_5 \sum_{i=0}^{n-1} (n-i-1) + c_6(n-1)$$

simplificando y renombrando las constantes, se tiene
$T(n) = an^2 = bn + c = O(n^2)$
Cuando el arreglo esta ordenado de manera decreciente, Selection Sort $\in O(n^2)$.

6. Bibliografía

[1]Brassard, Gilles; Bratley, Paul (1997). Fundamentos de Algoritmia. Madrid: PRENTICE HALL.

[2] Introduction to Algorithms, Second Edition by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein