Análisis de Algoritmos, Sem: 2018-1, 3CV2 Práctica 6, 19 de Octubre del 2017

Práctica 6: Problema del Maximo Subarreglo

Martínez Berumen Luis Daniel

Escuela Superior de Computo
Instituto Politecnico Nacional
dany.berumen@gmail.com

Abstract

En esta practica analizaremos la complejidad del algoritmo de maximo subarreglo, de igual manera la complejidad del maximo subarreglo cruzado, estos seran tratados tanto de manera grafica como de manera analitica, esperamos que los resultados sean de un algoritmo con complejidad lineal, para el caso del maximo subarreglo cruzado, en el caso del algortimo de maximo subarreglo una complejidad de $\theta(nlogn)$, ademas de considerar el caso cuando el arreglo esta lleno con numeros enteros negativos, en el cual sabemos se presenta un caso interesante para su analisis.

Palabras Clave

- Algoritmo
- Funcion
- Recursividad
- Orden
- Arreglo

1. Introducción

Divide y Vencerás es una frase quye hemos escuchado todos, al menos, una vez en nuestra vida, para nosotros es técnica de diseño de algoritmos, siendo de gran utilidad para nuestra carrera, ya que, los problemas a los que nos enfrentamos día con día son mas faciles de resolver si aplciamos una tecnica de este tipo. De hecho, suele ser considerada una filosofía general para resolver problemas, no solo del termino informatico, sino que también se utiliza en muchos otros ámbitos

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) (el pero de los casos) 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)) (el mejor de los casos) como:

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

El problema del subarreglo máximo consiste en encontrar un subarreglo de una determinada longitud m cuya suma sea máxima dentro de un arreglo de longitud n, con m mayor a n.

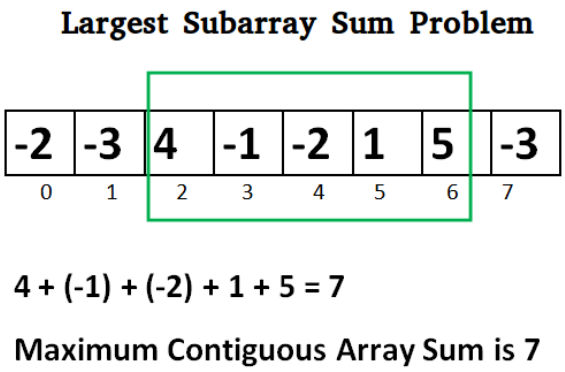

Figura 1 Maximo SubArreglo .

En la figura 1, podemos apreciar que tenemos un arreglo de 8 elementos, podemos ver que consta de numeros tanto negativos como positivos, pero siempre enteros.
Como podemos ver, la suma de los elementos en las posicions 2 a 6, nos da un resultado de 7, a este arreglo se le llama maximo subarreglo, ya que es la maxima suma obtenible con elementos secuenciales dentro del arreglo.

3. Experimentación y Resultados

3.1. Implemente el algortimo del maximo subarreglo.

i) Mediante gráficas, muestre que el algoritmo de maximosubarreglocruzado tiene complejidad lineal.

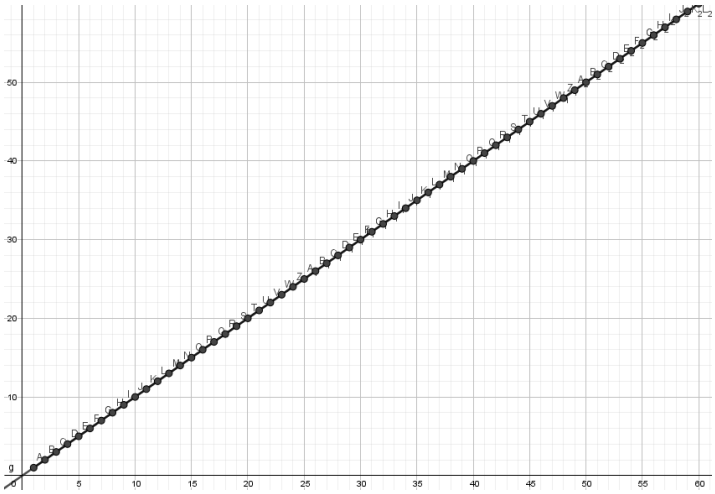

Figura 2. Gráfica del algoritmo del máximo subarreglo cruzado.

La curva de color negro es la función f(n) que propongo apartir de los resultados obtenidos de la ejecución,y la recta de color verde es g(n). Como se puede observar en la figura 2, tanto la función propuesta como la función obtenido por medio del algoritmo se enciman, es decir, se puede concluir de manera gráfica que el algoritmo del máximo subarreglo cruzado tiene de cota superior lineal. Dicha función es de tipo T(n)=$\theta(n)$.

ii) Demuestre analíticamente que el algoritmo maximosubarreglocruzado tiene complejidad lineal.

MSC(A, bajo, medio, alto)	Costo
1.-sumaIzq $\leftarrow \infty$	1
2.-suma$\leftarrow$ 0	1
3.-for i=medio to i=bajo	medio-bajo+2
4.- suma $\leftarrow$ suma +A[i]	1
5.- if suma sumaIzq	1
6.- sumaIzq $\leftarrow$ suma	1
7.- maxIzq $\leftarrow$ i	1
8.-sumaDer $\leftarrow \infty$	1
9.-suma$\leftarrow$ 0	1
10.-for i=medio+1 to i=alto	alto-(medio+1)+2
11.- suma $\leftarrow$ suma +A[i]	1
12.- if suma sumaDer	1
13.- sumaDer $\leftarrow$ suma	1
14.- maxDer $\leftarrow$ i	1

medio-bajo+2+alto-medio-1+2
= alto-bajo+3
= alto-bajo+1+2
= n+2
= $\theta(n)$

iii) Mediante gráficas, muestre que el algoritmo maximosubarreglo tiene complejidad θ(nlogn).

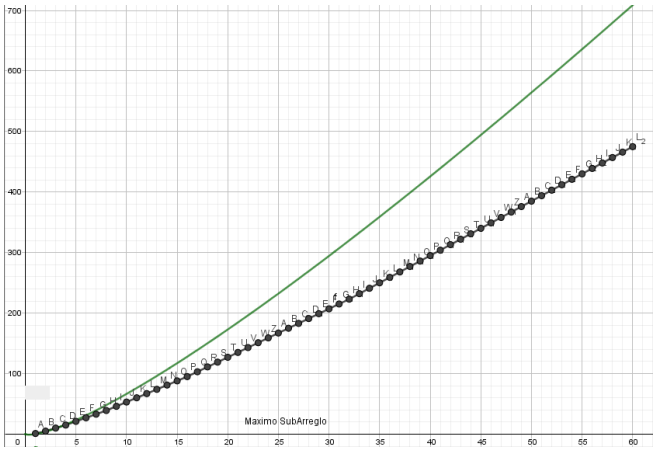

Figura 3. Gráfica del algoritmo del máximo subarreglo.

La curva de color negro es la función f(n) que propongo apartir de los resultados obtenidos de la ejecución,y tiene un comportamiento muy similar al de T(n)=$\theta(nlogn)$, mientras la recta de color verde será nuestra g(n) será muy similar a f(n) con la única diferencia que será multiplicado por alguna constante C. Observando la figura 3 se puede notar que nuestra cota superior propuesta para el algoritmo del máximo subarreglo es la correcta. Por lo tanto, es correcto suponer que el algoritmo tiene complejidad de la forma T(n)=$\theta(nlogn)$.

iV) Demuestre analíticamente que el algoritmo maximosubarreglotiene complejidad θ(nlogn).

1.-if bajo=alto	$\theta(1)$
2.- return(bajo, alto A[bajo])	$\theta(1)$
3.-else	
4.-medio $\leftarrow$ (bajo+alto)/2	1
5.-MS(A, bajo, medio)	$T(\frac{n}{2})$
6.-MS(A, medio+1, alto)	$T(\frac{n}{2})$
7.-MSC(A, bajo, medio, alto)	$\theta(n)$
8.-if(sumaIzq sumDer & sumaIzq sumaCruz)	$\theta(1)$
9.-return(mmIzq, maxIzq, sumaIzq)	$\theta(1)$

V) Implemente un algoritmo que resuelva el problema del máximo subarreglo utilizando fuerza bruta. Calculé su orden de complejidad analítica y experimentalmente.

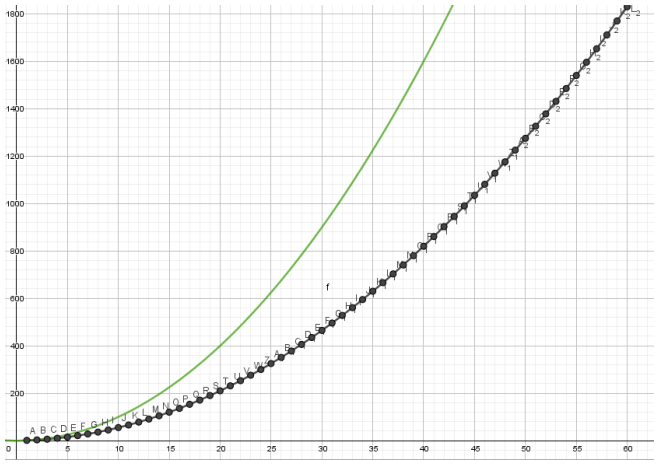

Figura 4. Gráfica del algoritmo implementado por l fuerza bruta.

La curva de color negro es la funcion f(n) que propongo apartir de los resultados obtenidos por medio de la experimentación y g(n) será la recta de color verde. El comportamiento que vamos a proponer será de T(n)=$\theta(n^2)$, tanto f(n) como g(n) se puede verificar que dicha complejidad sirve como cota superior de manera correcta. Por lo tanto, dicho algoritmo es de la forma T(n)=$\theta(n^2)$

4. Conclusión

Para estra practica regrese a utilizar C, ya que, al utilizar arreglos, la manera de C de manipularlos me sigue pareciendo mas sencilla que Python.
Me doy cuenta que en esta practica no es solamente necesario suponer que algun algoritmo tiene cierta complejidad, debemos de tener algun analisis, ya que es importante respaldar toda nuestra hipotesis con informacion verdadera.
Este algoritmo, cuando lo analise a .ºjo de buen cubero"no creia que fuera asi de complicada su implementacion, tuve que regresar a un lenguaje en el que me siento mas comodo como lo es C, esta vez Python no me ayudo tanto. Me gusto el trabajo de implementar este algoritmo, siento que fue la mejor manera de cerrar el tema de "Divide y venceras".

5. 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