Escuela Superior de Cómputo.
Instituto Politécnico Nacional, México.

Práctica 6: Problema del máximo subarreglo.

Blancas Pérez Bryan Israel
orionmunecaycanica@gmail.com

Resumen: Implementar un algoritmo para encontrar el máximo subarreglo de una cadena de enteros, utilizando el paradigma "Divide y Vencerás", y compararlo con el algoritmo de fuerza bruta.

Palabras Clave: Divide y Vencerás, Máximo Subarreglo, Máximo Subarreglo Cruzado, Complejidad Computacional.

1. Introducción

En esta práctica se implementará un algoritmo orientado al paradigma Divide y Vencerás, para encontrar el máximo subarreglo. Aunque, como se verá en el desarrollo de esta práctica, existen otros métodos para resolver este problema en específico, y depende de nosotros encontrar el óptimo.

2. Conceptos Básicos

Problema del máximo subarreglo.

El problema del subvector (o subarreglo), de suma máxima consiste en encontrar un subvector de una determinada longitud m cuya suma sea máxima dentro de un vector de longitud n, con $m \leq n$. La forma de aplicar este algoritmo, consiste en obtener los segmentos de suma máxima correspondientes a las mitades izquierda y derecha del vector y a la parte central para, una vez calculados, elegir el máximo de los tres [1].

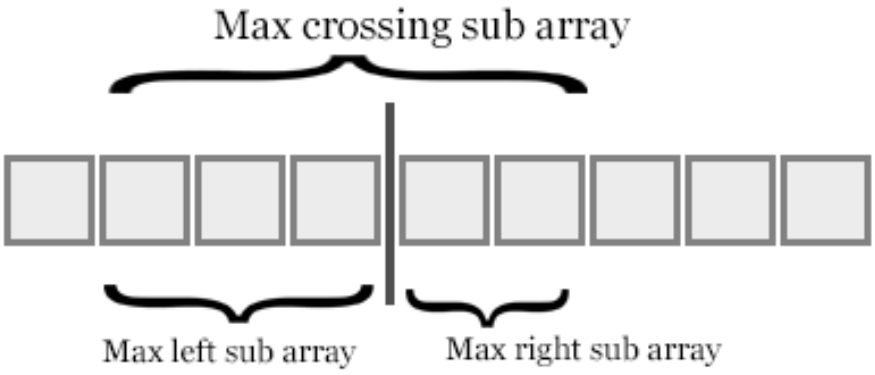

Figura 1: Maximum Subarray.

3. Experimentación y Resultados

Ejercicio 1.
Implementar el algoritmo de Máximo Subarreglo.

i) Mediante gráficas, muestre que el algoritmo del Máximo Subarreglo Cruzado tiene orden de complejidad lineal.

Pseudocódigo del algoritmo del Máximo Subarreglo Cruzado:

```
1  msc(A, bajo, medio, alto)
2    int sumaizq = A[medio]
3    suma = 0
4
5    for i = medio to i = bajo
6      suma += A[i];
7      if suma > sumaizq
8        sumaizq=suma;
9        min_izq=i;
10
11   int sumader=a[medio+1]
12   suma = 0
13
14   for i = medio+1 to i = alto
15     suma1 += A[i];
16     if suma1 > sumader
17       sumader = suma;
18       max_der=i;
19
20   return (max_izq, max_der, sumaizq + sumder);
```

Este algoritmo busca cual es la suma máxima que se puede lograr con los elementos del arreglo, considerando siempre que el subarreglo pase por en medio del arreglo original. En la figura 2, se observa el orden de complejidad del algoritmo, obtenido de manera experimental. La gráfica de color rojo, es la función $t(n) = 5n$, demostrando así que el algoritmo del Máximo Subarreglo Cruzado, tiene orden $\theta(n)$.

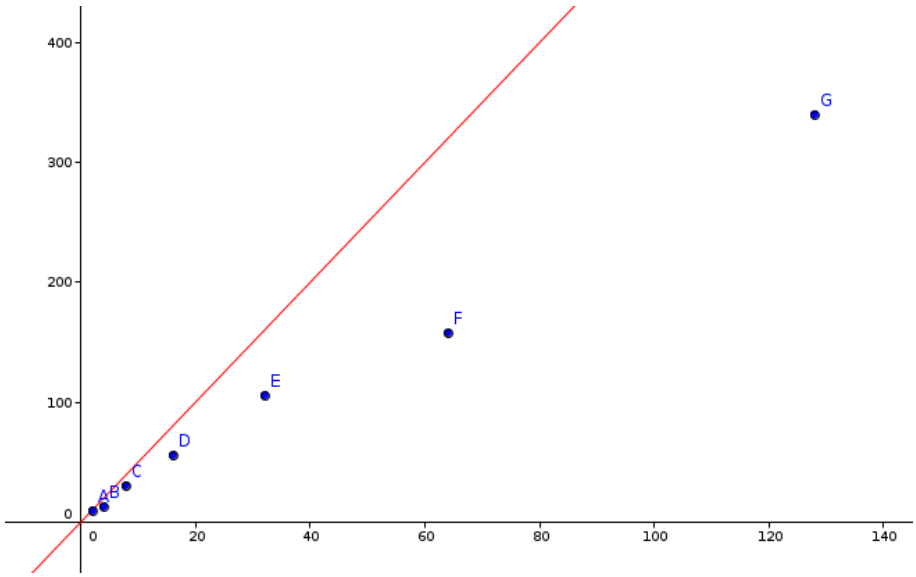

Figura 2: Máximo Subarreglo Cruzado.

ii)Demuestre analíticamente que el algoritmo del máximo subarreglo cruzado tiene complejidad lineal.

Análisis del Algoritmo	
	No. ejecuciones
$1er\ for$	$medio - bajo + 2$
$2do\ for$	$alto - (medio + 1) + 2$

Cuadro 1: Análisis del algoritmo.

Sumando el número de ejecuciones del 1er y 2do for, y considerando el tamaño total del arreglo igual a n, tenemos:
$1erfor + 2dofor = alto - bajo + 3 = (alto - bajo + 1) + 2 = n + 2$

Por lo tanto Maximo Subarreglo Cruzado $\epsilon\ \theta(n)$.

iii) Mediante gráficas, muestre que el algoritmo del Máximo Subarreglo tiene orden de complejidad $\theta(nlog(n))$.

Pseudocódigo del algoritmo del Máximo Subarreglo:

```
1  ms(int *a, int bajo, int alto){
2    if bajo = alto
3      return (bajo, alto, a[alto])
4    else
5      medio = (bajo + alto) / 2;
6      (min_izq, max_der, suma_izq) = ms(a, bajo, medio);
7      (min_izq, max_der, suma_der) = ms(a, medio+1, alto);
8      (min_izq, max_der, suma_cruz) = msc(a, bajo, medio, alto);
9
10     if(suma_izq >= suma_der and suma_izq >= suma_cruz){
11       return (min_izq, max_der, suma_izq)
12     else if(suma_der >= suma_izq and suma_der >= suma_cruz){
13       return (min_izq, max_der, suma_der)
14     else
15       return (min_izq, max_der, suma_cruz)
```

Como se puede ver en el pseudocódigo, este algoritmo implementa el paradigma Divide y Vencerás, puesto que va dividiendo el arreglo analizado una y otra vez, cada que se llama a si misma, hasta que solamente quede un arreglo de tamaño 1. Cuando se tengan las tres sumas de los subarreglos (suma_izq, suma_der y suma_cruz), se comparan para determinar cuál es el mayor de ellos y retornar el indicado.
En la figura 3, se muestra la gráfica de los valores obtenidos de la ejecución del algoritmo implementado en el lenguaje C. La gráfica de color rojo, corresponde a la función $t(n) = 4nlog(n)$, demostrando así, que el algoritmo esta acotado por dicha función, y por lo tanto tiene un orden de complejidad $\theta(nlogn)$

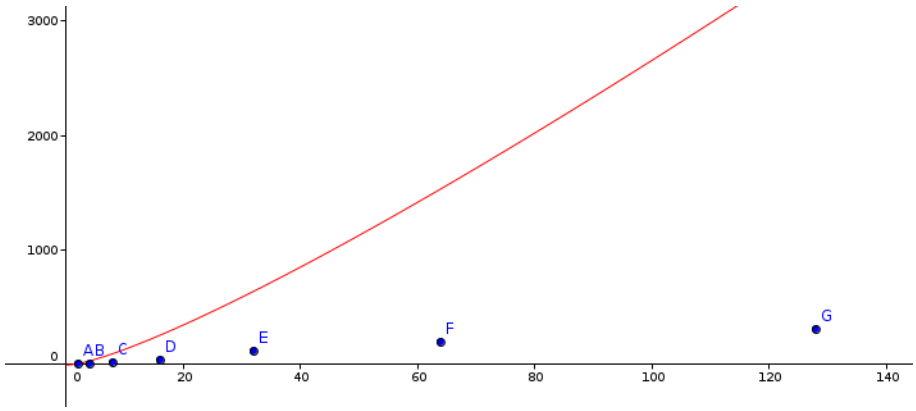

Figura 3: Máximo Subarreglo.

iv) Demuestre analíticamente que el algoritmo del Máximo Subarreglo tiene orden de complejidad $\theta(nlog(n))$.

Análisis del Algoritmo	
Línea de Código	**Orden de complejidad**
2-5	$\theta(1)$
6	$\theta(n/2)$
7	$\theta(n/2)$
8	$\theta(n)$ comprobado antes
10-15	$\theta(1)$

Cuadro 2: Análisis del algoritmo.

Por lo tanto, se tiene una ecuación recursiva de la siguiente forma:

$$T(n) = \begin{cases} C \ si \ n = 1 \\ 2T(n/2) + \theta(n) \ si \ n \ > \ 1 \end{cases}$$

Aplicando el Teorema Maestro, se tiene $a = 2, \ b = 2 \ y \ f(n) = \theta(n)$. Luego $n^{log_b a} = n^{log_2 2} = n$.
Entonces $f(n) = \theta(n)$.
Por lo tanto por el caso 2, del Teorema Maestro se tiene $T(n) \epsilon \ \theta(nlog_2(n))$

v) Implementar un algoritmo que resuelva el problema del máximo subarreglo utilizando fuerza bruta. Calcule su orden de complejidad analítica y experimentalmente.

Pseudocódigo del algoritmo del Máximo Subarreglo (Fuerza Bruta):

```
1  ms(A,n)
2    max=0
3    for i = 0 to i < n
4      suma=0
5      for   j = i to j < n
6        suma += a[j]
7        if suma > max
8          max=suma;
9
10 return max;
```

Este algoritmo es el más sencillo de implementar y de comprender, ya que se calcula todas las sumas posibles de todos los subarreglos que se pueden formar del arreglo original. El la figura 4, se muestra la gráfica de $T(n)$ vs n, la gráfica de color rojo, corresponde a la función $T(n) = 2n^2$, demostrando que este algoritmo tiene un orden de complejidad $\theta(n^2)$.

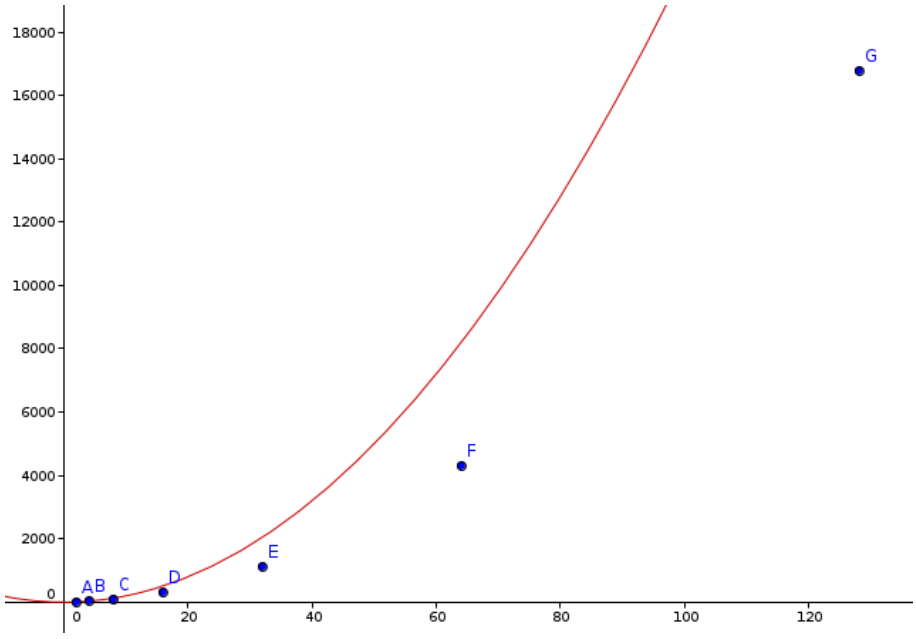

Figura 4: Máximo Subarreglo (Fuerza Bruta).

Demostración analítica.

Análisis del Algoritmo	
Línea de Código	**Orden de complejidad**
3	$\theta(n)$
5	$\theta(n)$
2,4,6,7,8,10	$\theta(1)$

Cuadro 3: Análisis del algoritmo de fuerza bruta.

Como las lineas de código dentro del segundo for, tiene orden de complejidad constante, ese bloque tiene complejidad lineal.
Como existen dos for anidados, se aplica el teorema de la multiplicación, quedando:
$T(n) = \theta(n * n)$, correspondientes al orden de complejidad del primer y segundo for. Así entonces: $T(n) = \theta(n^2)$

Por lo tanto , $T(n) \epsilon\ \theta(n^2)$

4. Conclusiones

Está práctica se me hizo muy sencilla, puesto que los algoritmos eran fáciles de implementar en un lenguaje de programación, y además, la lógica detrás de ellos no era tan compleja.
Lo que me parece más relevante de esta práctica, es identificar el algoritmo óptimo para la solución de un problema, porque como bien sabemos, existen muchos algoritmos para resolver una solo problemática.

5. Anexo

Resolver los siguientes problemas.

i) ¿Qué retorna la función de máximo subarreglo cuando todos los valores del arreglo son enteros negativos?.

R: Retorna siempre sólo dos elementos, el elemento en la posición $n/2$, y $(n/2)+1$. Esto es porque como el se desea obtener el máximo subarreglo, cualquier número que le sumemos a los iniciales, hará más pequeño el valor de la suma total del subarreglo, contradiciendo el objetivo del algoritmo.

6. Bibliografía

[1] https://es.wikipedia.org/wiki/Subvector_de_suma_máxima