Tarea 7

Saul Ivan Rivas Vega

Diseño y análisis de algoritmos

Equipo Completo:
Yadira Fleitas Toranzo
Diego de Jesús Isla Lopez
Saul Ivan Rivas Vega

12 de noviembre de 2019

1. Ejercicio 1.

Sea $A[1, ..., n]$ un arreglo con enteros tanto positivos como negativos en sus entradas. La *suma de un subarreglo* $A[i, ..., j]$ es la suma de sus entradas: $\sum_{k=i}^{j} A[k]$. El problema del *subarreglo de suma máxima* consiste en recibir A como entrada y devolver el subarreglo de A cuya suma sea la máxima posible de entre todos los subarreglos de A.
Haz lo siguiente:

1.1. a) Diseña un algoritmo tipo *divide y vencerás* que solucione el problema. Demuestra la corrección de tu algoritmo y haz un análisis de tiempo y espacio.

1.1.1. Algoritmo

Data: Arreglo A, los índices s y t con valores s=1, y t=n.
Result: Tupla (max_sum, i, j) Que representa el valor de la suma máxima y los índices i, j del subarreglo $A[i, ..., j]$ con dicha suma máxima.

`/* Evaluamos el caso base. */`
if $s = t$ **then**
  return (A[s], s, t);
end
`/* Tomamos el valor medio en el rango actual. */`
mid=(s+t)/2;
`/* Obtener la respuesta para el rango izquierdo. */`
$(suma_{izq}, i_{izq}, j_{izq})$ = MAXSUM(A, s, mid);
`/* Obtener la respuesta para el rango derecho. */`
$(suma_{der}, i_{der}, j_{der})$ = MAXSUM(A, mid + 1, t);
`/* Obtener la respuesta para el rango que pasa por la mitad` *mid*. `*/`
$(suma_{mid}, i_{mid}, j_{mid})$ = MIDSUM(A, s, mid, t);
return las respuestas máximas de entre las tres;

Algorithm 1: Algoritmo MAXSUM.

Data: Arreglo A, los índices s, mid y t.

Result: Tupla (max_sum, i, j) Que representa el valor de la suma máxima y los índices i, j del subarreglo $A[i, ..., j]$ con dicha suma máxima con $s \leq i, j \leq t$ y además $i \leq mid \leq j$.

/* Inicializamos variable de la suma a la izquierda de mid y el índice. */

$max\_suma_{izq} = -\infty$;
$indice_{izq} = 0$;

/* Recorremos desde mid hasta s reduciendo i en uno en cada iteración. */

$suma = 0$;
for i = mid, $i \geq s$ **do**
    $suma += A[i]$;
    if $suma > max\_suma_{izq}$ **then**
        $max\_suma_{izq} = suma$;
        $indice_{izq} = i$;
    end
end

/* Inicializamos variable de la suma a la derecha de mid y el índice. */

$max\_suma_{der} = -\infty$;
$indice_{der} = 0$;

/* Recorremos desde $mid + 1$ hasta t incrementando i en uno en cada iteración. */

$suma = 0$;
for i = mid+1, $i \leq t$ **do**
    $suma += A[i]$;
    if $suma > max\_suma_{der}$ **then**
        $max\_suma_{der} = suma$;
        $indice_{der} = i$;
    end
end
return $(max\_suma_{izq} + max\_suma_{der}, indice_{izq}, indice_{der})$;

Algorithm 2: Algoritmo MIDSUM.

1.1.2. Correctitud.

El algoritmo MIDSUM termina. Los dos ciclos en el algoritmo son finitos y terminan al recorrer los elementos de (s, mid) y los elementos en (mid+1, t), es decir los elementos en el rango (s, t). Siendo n la cantidad máxima de elementos en un rango s y t, podemos decir que a lo más hará n iteraciones y como en cada iteración hacemos operaciones constantes el algoritmo termina en n pasos.

El algoritmo MAXSUM termina. En cada invocación adicional en la ejecución del algoritmo se reduce el rango de búsqueda. En este caso el rango (s, t), se divide por la mitad en (s, mid) y ($mid + 1$, t). El rango no puede dividirse mas que $log_2(t - s + 1)$, que es a su vez igual a $log_2(n)$, pues es la cantidad de elementos en A. Así llegamos a que se realizaran $log_2(n)$ invocaciones. Y en cada invocación se realizan operaciones constantes y además se llama el algoritmo MIDSUM, el cual en el parrafo anterior vimos que termina en n pasos, por lo tanto el algoritmo MAXSUM termina en $log_2(n) \times n$ pasos.

El algoritmo MIDSUM devuelve el subarreglo de suma máxima tal que incluye el elemento en mid en el rango (s, t). El algoritmo empieza directamente con mid siendo mayor siempre en un caso práctico $-\infty$ se tomará al menos ese elemento. En cada paso del primer loop lo que hace es checar la suma del subarreglo que termina en mid y empieza en algún índice en el rango (s, mid), pues de hecho revisa cada posición en ese rango llevando la suma acumulada y actualizando el mayor. De igual forma en el segundo loop se revisa las posibles posiciones finales del subarreglo. Como revisamos todos los posibles inicios y finales para el subarreglo y nos quedamos con los mayores respectivamente, la suma de ambos será la mayor. Así llegamos a que el la suma y los indices que devuelve MIDSUM corresponden al subarreglo de suma máxima que incluye al elemento mid en el rango (s, t).

El algoritmo MAXSUM devuelve el subarreglo de suma máxima de A en el rango (s, t). Por inducción.
Invariante: El subarreglo de suma máxima se encuentra en alguno de los 3 siguientes casos:

- Esta completamente en la parte a la izquierda de mid.
- Esta completamente en la parte a la derecha de mid.
- Esta cruzando por mid.

Los cuales son revisados en MAXSUM partiendo en las invocaciones a la izquierda y derecha de mid, así como el subarreglo cruzando por mid con MIDSUM.
Caso base: s es igual a t, así mismo $mid = s$, por lo tanto el único rango es de (s, t), que es igual a (s, s) y a su vez (mid, mid), lo cual cae en el caso 3 de la invariante. **Hipótesis inductiva:** El algoritmo devolverá el subarreglo de suma máxima que se encuentre en alguno de los 3 casos mencionados.
Suponemos cierto para una i-ésima invocación del algoritmo. Esto implica que ya tenemos la respuesta para cuando buscamos en el rango (s, mid), así como el rango $(mid + 1, t)$, los cuales corresponden a los primeros dos casos. Para el tercero y último caso debemos

considerar los rangos que contienen al elemento en *mid* y como vimos anteriormente el algoritmo MIDSUM nos devolverá el mayor. Entonces como revisamos esos tres casos y devolvemos el mayor podemos decir finalmente que el mayor rango que se encuentre en los únicos 3 casos mencionados, será revisado por el algoritmo y devuelto como respuesta.

1.1.3. Complejidad en Tiempo.

MIDSUM es de complejidad $O(n)$. Como vimos en la sección anterior el algoritmo de MIDSUM termina después de n iteraciones a lo más, por que solo recorre dos ciclos cuyo tamaño total es n.

MAXSUM es de complejidad $O(nlog(n))$. Como vimos en la sección anterior el algoritmo de MAXSUM termina después de $nlogn$ operaciones, que es de realizar $logn$ invocaciones con a lo más n operaciones en cada una por la llamada a MIDSUM, por lo que la complejidad total es $O(nlog(n))$. Con una relación de recurrencia podemos ver la complejidad como:

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 4T(n/4) + 2(n/2) + n \\ &= 8T(n/8) + 4(n/4) + n + n \\ &\text{se pueden tener a lo mas } log_2 n \text{ niveles en la recurrencia...} \\ &= T(1) + n + ... + n + n \\ &= 1 + n + ... + n + n \\ &= 1 + n(log_2 n) \end{aligned} \tag{1}$$

1.1.4. Complejidad en Espacio.

El algoritmo MIDSUM ocupa memoria adicional de O(1). En MIDSUM solo se inicializan de manera adicional a las entradas las variables para las sumas e índices, lo cual es de orden $O(1)$.

El algoritmo MAXSUM ocupa memoria adicional de $O(log_2 n)$. En cada invocación solo usamos variables adicionales de $O(1)$, sin embargo la pila de llamadas recursivas también ocupa memoria adicional, esta es equivalente a la profundidad de las invocaciones y como vimos en la sección anterior esta es de $log_2 n$, por lo que ademas de las variables de $O(1)$ utilizamos $O(log_2 n)$ de la pila de llamadas recursivas. Finalmente como el factor dominante es $O(log_2 n)$ decimos que en espacio MAXSUM es de orden $O(log_2 n)$.

1.2. b) Diseña un algoritmo tipo *programación dinámica* que solucione el problema. Demuestra la corrección de tu algoritmo y haz un análisis de tiempo y espacio.

1.3. c) Haz una comparación de tus dos algoritmos, explicando sus ventajas y desventajas del uno con respecto al otro.

2. Diseña un algoritmo que tome como entrada una gráfica dirigida G con pesos en sus aristas, w : E → Z, y responda a lo siguiente:

- Si G no tiene ciclos de pesos negativo, entonces responde FALSE.
- De otra forma, el algoritmo responde TRUE y además devuelve un ciclo C de G de peso negativo.