Dividir y Conquistar

Algoritmos y Estructuras de Datos II,
Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

11 de junio de 2018

Dividir y conquistar

La resolución de un problema con D&C tiene tres partes:

1. **Dividir** el problema en k subproblemas del mismo tipo.
2. **Conquistar** resolviendo los subproblemas recursivamente.
3. **Combinar** las soluciones obtenidas para resolver el problema original.

Ejemplo

Merge Sort...

- Si el arreglo a ordenar tiene al menos 2 elementos:
  1. **Dividir** el arreglo en dos mitades.
  2. **Conquistar** (i.e., ordenar los subarreglos) aplicando recursivamente Merge Sort.
  3. **Combinar** los subarreglos ya ordenados para obtener el arreglo final ordenado.

Dividir y conquistar - El esquema general

DC(X)

- Si X es suficientemente chico (o simple):
  - Resolver X directamente
- En caso contrario:
  - Descomponer X en subinstancias X_1, X_2,...,X_k
  - Para i desde 1 hasta k hacer
    - Y_i = **DC(**X_i**)**
  - Combinar las soluciones Y_i para construir una solución Y para X

Teorema maestro

Podemos aplicarlo para resolver recurrencias de la forma:

$$T(n) = \begin{cases} aT\left(\frac{n}{c}\right) + f(n) & n > 1 \\ \theta(1) & n = 1 \end{cases}$$

Donde:

- a es la cantidad de subproblemas a resolver.
- c es la cantidad de particiones y $\frac{n}{c}$ es el tamaño de los subproblemas a resolver.
- $f(n)$ es el costo de todo lo que se hace en cada llamado además de los llamados recursivos.

Teorema maestro

Bajo estos supuestos, sabemos que $T(n)$ tiene las siguientes complejidades:

- $\theta(n^{\log_c a})$

  $\hookrightarrow$ Si $\exists \varepsilon > 0$ tal que $f(n) \in O(n^{\log_c a - \varepsilon})$
- $\theta(n^{\log_c a} \log n)$

  $\hookrightarrow$ Si $f(n) \in \theta(n^{\log_c a})$
- $\theta(f(n))$

  $\hookrightarrow$ Si $\exists \varepsilon > 0$ tal que $f(n) \in \Omega(n^{\log_c a + \varepsilon})$ y $\exists \delta < 1, \exists n_0 > 0$ tal que $\forall n \geq n_0$ se cumple $af\left(\frac{n}{c}\right) \leq \delta f(n)$

Teorema maestro

Bajo estos supuestos, sabemos que $T(n)$ tiene las siguientes complejidades:

- $\theta(n^{\log_c a})$

  $\hookrightarrow$ Si $\exists \varepsilon > 0$ tal que $f(n) \in O(n^{\log_c a - \varepsilon})$

- $\theta(n^{\log_c a} \log n)$

  $\hookrightarrow$ Si $f(n) \in \theta(n^{\log_c a})$

- $\theta(f(n))$

  $\hookrightarrow$ Si $\exists \varepsilon > 0$ tal que $f(n) \in \Omega(n^{\log_c a + \varepsilon})$ y
  $\exists \delta < 1, \exists n_0 > 0$ tal que $\forall n \geq n_0$ se cumple $af\left(\frac{n}{c}\right) \leq \delta f(n)$

Observación: Cuando coinciden a y c (o sea, resolvemos todos los subproblemas generados), entonces $\log_c a = $ ¿?

Observación: Cuando coinciden a y c (o sea, resolvemos todos los subproblemas generados), entonces... $\log_c a = 1$.

En este caso, $T(n)$ tiene las siguientes complejidades:

$$\begin{cases} \theta(n) & \text{Si } \exists \varepsilon > 0 \text{ tal que } f(n) \in O(n^{1-\varepsilon}) \\ \theta(n \log n) & \text{Si } f(n) \in \theta(n) \\ \theta(f(n)) & \text{Si } \exists \varepsilon > 0 \text{ tal que } f(n) \in \Omega(n^{1+\varepsilon}) \text{ y} \\ & \exists \delta < 1, \exists n_0 > 0 \text{ tal que } \forall n \geq n_0 \text{ se cumple } af\left(\frac{n}{c}\right) \leq \delta f(n) \end{cases}$$

Teorema Maestro

- ¿Es el Teorema Maestro la única herramienta que tenemos para ver que una ecuación de recurrencia está en una clase de complejidad?

Teorema Maestro

- ¿Es el Teorema Maestro la única herramienta que tenemos para ver que una ecuación de recurrencia está en una clase de complejidad?
- No, también podemos armar el **árbol de recurrencia** para tener un indicio de cuál es la forma cerrada de la ecuación (o una cota de ello) y luego demostrarlo (e.g., por inducción).

Teorema Maestro

- ¿Es el Teorema Maestro la única herramienta que tenemos para ver que una ecuación de recurrencia está en una clase de complejidad?
- No, también podemos armar el **árbol de recurrencia** para tener un indicio de cuál es la forma cerrada de la ecuación (o una cota de ello) y luego demostrarlo (e.g., por inducción).
- ¿Y entonces para qué queremos el teorema maestro?

Teorema Maestro

- ¿Es el Teorema Maestro la única herramienta que tenemos para ver que una ecuación de recurrencia está en una clase de complejidad?
- No, también podemos armar el **árbol de recurrencia** para tener un indicio de cuál es la forma cerrada de la ecuación (o una cota de ello) y luego demostrarlo (e.g., por inducción).
- ¿Y entonces para qué queremos el teorema maestro?
- ¡Porque no tenemos que demostrar nada! Alcanza con ver en qué caso cae y listo (si es que cae en alguno... (!)).

Ejemplo: Merge Sort

Algorithm 1 MERGE-SORT(A: arreglo(nat))

$n \leftarrow$ long(A)	$\theta(1)$
$izq \leftarrow$ subarreglo(A, 0, $n/2$)	$\theta(n)$
$der \leftarrow$ subarreglo(A, $n/2$, n)	$\theta(n)$
MERGE-SORT(izq)	$T(\frac{n}{2})$
MERGE-SORT(der)	$T(\frac{n}{2})$
merge(izq, der, A)	$\theta(n)$

Ejemplo: Merge Sort

Algorithm 2 MERGE-SORT(A: arreglo(nat))

$n \leftarrow$ long(A)	$\theta(1)$
$izq \leftarrow$ subarreglo(A, 0, $n/2$)	$\theta(n)$
$der \leftarrow$ subarreglo(A, $n/2$, n)	$\theta(n)$
MERGE-SORT(izq)	$T(\frac{n}{2})$
MERGE-SORT(der)	$T(\frac{n}{2})$
merge(izq, der, A)	$\theta(n)$

- $T(n) = 2T(\frac{n}{2}) + f(n)$, con $f(n) \in \theta(n)$
- $a = c = 2$

Ejemplo: Merge Sort

Algorithm 3 MERGE-SORT(A: arreglo(nat))

$n \leftarrow$ long(A)	$\theta(1)$
$izq \leftarrow$ subarreglo(A, 0, $n/2$)	$\theta(n)$
$der \leftarrow$ subarreglo(A, $n/2$, n)	$\theta(n)$
MERGE-SORT(izq)	$T(\frac{n}{2})$
MERGE-SORT(der)	$T(\frac{n}{2})$
merge(izq, der, A)	$\theta(n)$

- $T(n) = 2T(\frac{n}{2}) + f(n)$, con $f(n) \in \theta(n)$
- $a = c = 2$
- Entonces $T(n) = \theta(n \log n)$

Ejemplo 2: Búsqueda en arreglo ordenado

- Dado un arreglo de elementos ordenados y un elemento perteneciente a él, se quiere encontrar la posición en la que está.
- Si n es la cantidad de elementos del arreglo, encontrar un algoritmo que resuelva el problema en un tiempo estrictamente menor a $O(n)$

Máximo

- Un arreglo de enteros *montaña* está compuesto por una secuencia estrictamente creciente seguida de una estrictamente decreciente.
- Suponemos que hay al menos un elemento menor y uno mayor que el máximo (las secuencias creciente y decreciente tienen al menos 2 elementos)
- Por ejemplo, el arreglo (-1, 3, 8, 22, 30, 22, 8, 4, 2, 1)
- Dado un arreglo montaña de longitud n, queremos encontrar al máximo. La complejidad del algoritmo que resuelva el problema debe ser O(log n)

Subsecuencia de suma máxima

- Dada una secuencia de n enteros, se desea encontrar el máximo valor que se puede obtener sumando elementos consecutivos.
- Por ejemplo, para la secuencia (3, -1, 4, 8, -2, 2, -7, 5), este valor es 14, que se obtiene de la subsecuencia (3, -1, 4, 8).
- Si una secuencia tiene todos números negativos, se entiende que su subsecuencia de suma máxima es la vacía, por lo tanto el valor es 0.
- Se pide desarrollar un algoritmo que encuentre la suma de la máxima subsecuencia y tenga complejidad estrictamente menor que $O(n^2)$.

Matriz creciente

Se tiene una matriz A de $n * n$ números naturales, de manera que $A[i, j]$ representa al elemento en la fila i y columna j $(1 \leq i, j \leq n)$. Se sabe que el acceso a un elemento cualquiera se realiza en tiempo $O(1)$. Se sabe también que todos los elementos de la matriz son distintos y que todas las filas y columnas de la matriz están ordenadas de forma creciente (es decir, $i < n \Rightarrow A[i, j] < A[i + 1, j]$ y $j < n \Rightarrow A[i, j] < A[i, j + 1]$).

a. Implementar, utilizando la técnica de dividir y conquistar, la función:

   está(in n: nat, in A: matriz(nat), in e: nat) $\rightarrow$ bool

   que decide si un elemento e dado aparece en alguna parte de la matriz. Se debe dar un algoritmo que tome tiempo estrictamente menor que $O(n^2)$. Notar que la entrada es de tamaño $O(n^2)$.

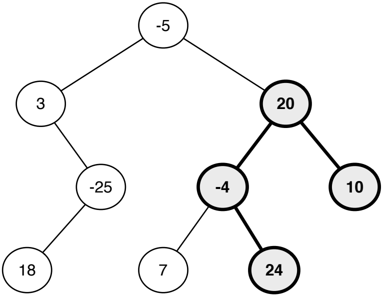
b. Calcular y justificar la complejidad del algoritmo propuesto. Para simplificar el cálculo, se puede suponer que n es potencia de dos.

Ejercicio de parcial

Dado un árbol binario de números enteros, se desea calcular la máxima suma de los nodos pertenecientes a un camino entre dos nodos cualesquiera del árbol. Un camino entre dos nodos n_1 y n_2 está formado por todos los nodos que hay que atravesar en el árbol para llegar desde n_1 hasta n_2, incluyéndolos a ambos. Un camino entre un nodo y sí mismo está formado únicamente por ese nodo.

Ejercicio de parcial

Se pide dar un algoritmo que resuelva el problema utilizando la técnica de *Dividir y Conquistar*, calculando y justificando claramente su complejidad. El algoritmo debe tener una complejidad temporal de peor caso igual o mejor que $O(n \log n)$ siendo n la cantidad de nodos del árbol y suponiendo que el árbol está balanceado.



Ejercicio de parcial

Ahora...

- ¿Lo podremos hacer $O(n)$?

Ejercicio de parcial

Ahora...

- ¿Lo podremos hacer $O(n)$?
- ¿Qué complejidad tiene nuestro algoritmo si el árbol no está balanceado?

Final

- ¿Preguntas?