

# Dividir y Conquistar

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

3 de Noviembre de 2017

# Dividir y conquistar

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

- 1 **Dividir** el problema en  $k$  subproblemas del mismo tipo, pero más chicos.
- 2 **Conquistar** resolviendo los subproblemas, recursivamente o directamente (si son lo suficientemente fáciles o chicos).
- 3 **Combinar** las soluciones obtenidas para resolver el problema original.

# Merge Sort

- 1 **Dividir** el arreglo en dos mitades.
- 2 **Conquistar** Si los arreglos son de un elemento, ya están ordenados, sino, hacer recursión sobre ellos.
- 3 **Combinar** Merge para obtener el arreglo ordenado a partir de las dos mitades ordenadas.

DC(X)

- Si X es suficientemente chico (o simple):
  - ADHOC(X)
- En caso contrario:
  - Descomponer X en subinstancias  $X_1, X_2, \dots, X_k$
  - 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

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, o sea  $\frac{n}{c}$  es el tamaño de los subproblemas a resolver
- $f(n)$  es el costo de todo lo que se hace en cada paso además de la recursión

# Teorema maestro

Tres casos:  $T(n) =$

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

# Teorema Maestro

Ahora bien, ¿qué pasa cuando coinciden  $a$  y  $c$ ? O sea, resolvemos cada uno de los subproblemas en los que particionamos el problema original

# Teorema Maestro

Ahora bien, ¿qué pasa cuando coinciden  $a$  y  $c$ ? O sea, resolvemos cada uno de los subproblemas en los que particionamos el problema original

$\log_c a = 1$  entonces...



# Teorema Maestro

Ahora bien, ¿qué pasa cuando coinciden  $a$  y  $c$ ? O sea, resolvemos cada uno de los subproblemas en los que particionamos el problema original

$\log_c a = 1$  entonces...

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

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

¿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**, que nos puede dar un indicio de cual es la forma cerrada de la ecuación (y luego demostrar que efectivamente la complejidad es la que supusimos). O directamente **demostrar por inducción la 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**, que nos puede dar un indicio de cual es la forma cerrada de la ecuación (y luego demostrar que efectivamente la complejidad es la que supusimos). O directamente **demostrar por inducción la complejidad**.

¿Y entonces por qué está 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**, que nos puede dar un indicio de cual es la forma cerrada de la ecuación (y luego demostrar que efectivamente la complejidad es la que supusimos). O directamente **demostrar por inducción la complejidad**.

¿Y entonces por qué está el teorema maestro?

¡¡Porque no tenemos que demostrar nada!! Simplemente ver en qué caso cae y listo. Algún genio ya demostró que funciona, y nos lo regaló.

# 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  $O(n)$

- 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.



# 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:

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

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  $O(n^2)$ .

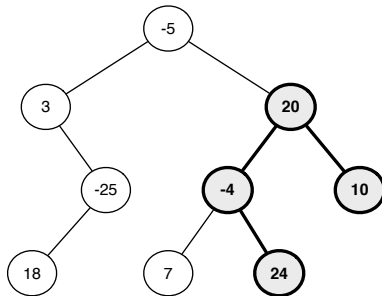
- (b) Calcular y justificar la complejidad del algoritmo propuesto. Para simplificar el cálculo, se puede suponer que  $n$  es potencia de dos.

## 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. Suponer que el árbol está balanceado.

## Ejercicio de parcial

Se pide dar un algoritmo  $\text{MÁXIMASUMACAMINO}(a : \text{ab}(\text{int})) \rightarrow \text{int}$  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.



**Figura:** Ejemplo de un camino de máxima suma en un posible  $\text{ab}(\text{int})$ . Resultado correcto: 50.

# Ejercicio de parcial

Y... lo podremos hacer  $O(n)$



Figura: wubalubadubdub

- ¿Preguntas?

