

## Tema 3: Divide y Vencerás

A. Salguero, F. Palomo, I. Medina

Universidad de Cádiz

Diseño de Algoritmos  
Curso 2013/14

# Introducción

«Divide y vencerás» no es más que una técnica particular de diseño recursivo. Un ejemplar se descompone en varios subejemplares de tamaño menor.

Pero, a veces, la forma natural de dividir un ejemplar conduce a otros que se solapan. Al resolverlos independientemente aparecen finalmente muchos subejemplares idénticos. El enfoque de la «programación dinámica» consiste en evitar esto resolviendo cada subejemplar exactamente una vez. Para ello es necesario guardar su solución para un uso posterior.

Existen determinados tipos de problemas en los que no aparecen subejemplares solapados al descomponer el problema. Un enfoque de tipo «divide y vencerás» produce en ese caso algoritmos más eficientes.

# Esquema general

- Descomposición.  
Se descompone el ejemplar a resolver en subejemplares del mismo problema.
- Resolución recursiva.  
Se resuelve cada uno de ellos recursivamente, dado un cierto tamaño umbral y un algoritmo que resuelva el problema para subejemplares cuyo tamaño no supere el umbral.
- Combinación.  
Se obtiene la solución del ejemplar original a partir de las de cada subejemplar.

Existen distintas variantes de esta técnica general, principalmente: *simplificación o reducción, equilibrado y partición.*

# Búsqueda binaria

Este algoritmo funciona por *simplificación binaria* y busca la posición de la primera aparición de un elemento en un vector ordenado.

*búsqueda-binaria* :  $x \times v \times i \times j \rightarrow p$

$n \leftarrow j - i + 1$

si  $n = 1$

    si  $x \leq v[i]$

$p \leftarrow i$

    si no

$p \leftarrow i + 1$

si no

$k \leftarrow i - 1 + n \text{ div } 2$

    si  $x \leq v[k]$

$p \leftarrow \text{búsqueda-binaria}(x, v, i, k)$

    si no

$p \leftarrow \text{búsqueda-binaria}(x, v, k + 1, j)$

# Búsqueda binaria

La precondition es:

$$i \leq j \wedge \forall \alpha \in [i, j) \ v[\alpha] \leq v[\alpha + 1]$$

La postcondición es:

$$i \leq p \wedge (p = i \vee v[p - 1] < x) \wedge \\ p \leq j + 1 \wedge (p = j + 1 \vee x \leq v[p])$$

Si  $x$  está en  $[v_i, \dots, v_j]$ ,  $p$  será su posición. Si no,  $p$  contendrá la primera posición donde  $x$  podría insertarse sin alterar la ordenación. Si se desea obtener  $n + 1$  cuando el elemento no esté, se puede utilizar el algoritmo de la siguiente forma:

```
posición :  $x \times v \times n \rightarrow p$ 
 $p \leftarrow \text{búsqueda-binaria}(x, v, 1, n)$ 
si  $p \leq n \wedge x \neq v[p]$ 
   $p \leftarrow n + 1$ 
```

# Búsqueda binaria

El número de comparaciones entre elementos del vector que realiza *búsqueda-binaria* es:

$$t(n) = \begin{cases} 1, & n = 1 \\ t(\lfloor \frac{n}{2} \rfloor) + 1, & n > 1 \wedge x \leq v_k \\ t(\lceil \frac{n}{2} \rceil) + 1, & n > 1 \wedge x > v_k \end{cases}$$

En el *mejor caso*, se entra por la primera rama sistemáticamente. En el *peor caso*, se entra siempre por la segunda. Cuando  $n = 2^k$ , las dos se funden en:

$$t(n) = t\left(\frac{n}{2}\right) + 1$$

En ambos casos  $t(n) = \log_2 n + 1$ ,  $n = 2^k$ . Esto implica que también en el *caso promedio*  $t(n) \in \Theta(\log n \mid n = 2^k)$ . Se demuestra que la restricción se puede eliminar del orden.

# Potencia rápida

Sea  $\langle A, \cdot, 1 \rangle$  un monoide (es decir, un semigrupo con identidad). Se desea calcular la  $n$ -ésima potencia de un elemento  $a \in A$ .

El siguiente algoritmo funciona por *simplificación binaria* del exponente.

```
potencia :  $a \times n \rightarrow r$   
si  $n = 0$   
   $r \leftarrow 1$   
si no  
  si  $n = 1$   
     $r \leftarrow a$   
  si no  
    si  $n \bmod 2 = 0$   
       $r \leftarrow potencia(a \cdot a, n \div 2)$   
    si no  
       $r \leftarrow a \cdot potencia(a, n - 1)$ 
```

# Potencia rápida

El número de productos realizados es:

$$t(n) = \begin{cases} 0, & n \leq 1 \\ t\left(\frac{n}{2}\right) + 1, & n > 1 \wedge 2 \mid n \\ t(n-1) + 1, & n > 1 \wedge 2 \nmid n \end{cases}$$

En el *mejor caso*,  $n$  es par sistemáticamente. Luego  $n = 2^k$  y  $t(n) = \log_2 n$ ,  $n \geq 1$ .

En el *peor caso*,  $n$  alterna entre impar y par sistemáticamente. Si  $n$  es impar,  $n-1$  es par:

$$t(n) = t(n-1) + 1 = t\left(\frac{n-1}{2}\right) + 2$$

Luego  $n = 2^k - 1$  y  $t(n) = 2 \log_2(n+1) - 2$ ,  $n \geq 1$ . Así, en ambos casos  $t(n) \in \Theta(\log n)$ .



# Potencia rápida

Se puede generalizar fundiendo ambas ramas y realizando un análisis más detallado.

$$t(n) = t\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1 + n \bmod 2$$

# Mínimo y máximo

Este algoritmo funciona por *equilibrado binario*. Calcula recursivamente el mínimo y el máximo de dos subvectores de tamaños similares y los combina para formar el resultado.

$\text{mín-máx} : v \times i \times j \rightarrow a \times b$

$n \leftarrow j - i + 1$

si  $n = 1$

$\langle a, b \rangle \leftarrow \langle v[i], v[i] \rangle$

si no

$k \leftarrow i - 1 + n \text{ div } 2$

$\langle c, d \rangle \leftarrow \text{mín-máx}(v, i, k)$

$\langle e, f \rangle \leftarrow \text{mín-máx}(v, k + 1, j)$

$\langle a, b \rangle \leftarrow \langle \text{mín}(c, e), \text{máx}(d, f) \rangle$

Cada *mín* o *máx* realiza una comparación.

$$t(n) = \begin{cases} 0, & n = 1 \\ t(\lfloor \frac{n}{2} \rfloor) + t(\lceil \frac{n}{2} \rceil) + 2, & n > 1 \end{cases}$$

# Mínimo y máximo

No es mejor que el algoritmo clásico: se obtiene que  $t(n) = 2n - 2$ ,  $n = 2^k$ . Esto es mejorable, ya que para  $n = 2$  basta una sola comparación en lugar de dos. Elevando el umbral:

```
...
si  $n = 1$ 
   $\langle a, b \rangle \leftarrow \langle v[i], v[i] \rangle$ 
si no
  si  $n = 2$ 
    si  $v[i] \leq v[j]$ 
       $\langle a, b \rangle \leftarrow \langle v[i], v[j] \rangle$ 
    si no
       $\langle a, b \rangle \leftarrow \langle v[j], v[i] \rangle$ 
  si no
    ...
```

No hay mejora asintótica, pero disminuyen las comparaciones:

$$t(n) = \frac{3}{2}n - 2, \quad n = 2^k \geq 2.$$

# Ordenación por fusión

Este algoritmo funciona por *equilibrado binario*. Descompone el vector en dos subvectores de tamaños similares, los ordena recursivamente y combina los resultados en un vector ordenado.

*ordenación-fusión* :  $v \times i \times j \rightarrow v$

$n \leftarrow j - i + 1$

si  $n \leq n_0$

*ordenación-inserción*( $v, i, j$ )

si no

$k \leftarrow i - 1 + n \text{ div } 2$

*ordenación-fusión*( $v, i, k$ )

*ordenación-fusión*( $v, k + 1, j$ )

*fusión*( $v, i, k, j$ )

Como algoritmo base se emplea una versión del algoritmo de ordenación por inserción directa capaz de trabajar con subvectores.

# Ordenación por fusión

En la combinación se funden dos subvectores ordenados  $[v_i, \dots, v_k]$  y  $[v_{k+1}, \dots, v_j]$  en un único subvector ordenado  $[v_i, \dots, v_j]$ .

*fusión* :  $v \times i \times k \times j \rightarrow v$

$n \leftarrow j - i + 1$

$\langle p, q \rangle \leftarrow \langle i, k + 1 \rangle$

desde  $l \leftarrow 1$  hasta  $n$

    si  $p \leq k \wedge (q > j \vee v[p] \leq v[q])$

$\langle w[l], p \rangle \leftarrow \langle v[p], p + 1 \rangle$

    si no

$\langle w[l], q \rangle \leftarrow \langle v[q], q + 1 \rangle$

desde  $l \leftarrow 1$  hasta  $n$

$v[i - 1 + l] \leftarrow w[l]$

La fusión emplea  $n$  comparaciones entre elementos del vector en el peor caso y un espacio auxiliar de  $n$  elementos.

# Ordenación por fusión

Analicemos el número de comparaciones entre elementos del vector.  
Sea  $t(n)$  su valor en el *peor caso*:

- La ordenación por inserción directa realiza  $n(n-1)/2$  comparaciones.
- Los subvectores, de tamaños  $\lfloor \frac{n}{2} \rfloor$  y  $\lceil \frac{n}{2} \rceil$ , se ordenan recursivamente con  $t(\lfloor \frac{n}{2} \rfloor)$  y  $t(\lceil \frac{n}{2} \rceil)$  comparaciones, respectivamente.
- La combinación (fusión de dos vectores ordenados) emplea  $n$  comparaciones.

El tiempo en el peor caso viene entonces dado por la siguiente ecuación de recurrencia:

$$t(n) = \begin{cases} \frac{n(n-1)}{2}, & n \leq n_0 \\ t(\lfloor \frac{n}{2} \rfloor) + t(\lceil \frac{n}{2} \rceil) + n, & n > n_0 \end{cases}$$

# Ordenación por fusión

Restringimos  $n = n_0 2^k$ , así para  $n > n_0$ :

$$t(n_0 2^k) = 2t(n_0 2^{k-1}) + n_0 2^k$$

$$T(k) = 2T(k-1) + n_0 2^k$$

El polinomio característico ampliado de  $T(k)$  es:

$$c(x) = (x-2)(x-2) = (x-2)^2$$

Por lo tanto:

$$T(k) = (\alpha k + \beta) 2^k$$

y hallamos  $\alpha$ , que no es libre, por sustitución:

$$\alpha k 2^k = 2\alpha(k-1)2^{k-1} + n_0 2^k$$

resultando  $\alpha = n_0$ , y deshaciendo el cambio:

$$T(k) = (n_0 k + \beta) 2^k$$

$$t(n) = \left( n_0 \log_2 \frac{n}{n_0} + \beta \right) \frac{n}{n_0}$$

# Ordenación por fusión

Ya se puede obtener el orden en el peor caso:

$$t(n) \in O(n \log n \mid n = n_0 2^k)$$

Se demuestra que la restricción se puede eliminar del orden.  
Calculemos la solución particular. Aplicando la condición inicial  $t(n_0) = n_0(n_0 - 1)/2$ :

$$\left( n_0 \log_2 \frac{n_0}{n_0} + \beta \right) \frac{n_0}{n_0} = \frac{n_0(n_0 - 1)}{2}$$

con lo que  $\beta = n_0(n_0 - 1)/2$ , quedándonos:

$$\begin{aligned} t(n) &= n \log_2 n + \left( \frac{\beta}{n_0} - \log_2 n_0 \right) n \\ &= n \log_2 n + \left( \frac{n_0 - 1}{2} - \log_2 n_0 \right) n \end{aligned}$$

siempre que  $n = n_0 2^k$ .



# Ordenación por fusión

Calculemos el *umbral óptimo*, un  $n_0 \in \mathbb{N}^*$  que minimice  $u(n_0) = (n_0 - 1)/2 - \log_2 n_0$ :

$$u'(x) = \frac{1}{2} - \frac{1}{x \ln 2}$$

$$u'(x) = 0 \implies x = \frac{2}{\ln 2} \approx 2.89$$

pero el umbral ha de ser un natural:

$$u(2) = -\frac{1}{2} = -0.5$$

$$u(3) = 1 - \log_2 3 \approx -0.58$$

con esto minimizamos localmente, queda comprobar los extremos para que sea global:

$$u(1) = 0$$

$$u(\infty) = \infty$$

luego, finalmente, el umbral óptimo es  $n_0 = 3$ .

# Ordenación rápida

Este algoritmo funciona por *partición*. Descompone el vector adecuadamente para que al ordenar recursivamente las partes el resultado quede ya ordenado.

*ordenación-rápida* :  $v \times i \times j \rightarrow v$

$n \leftarrow j - i + 1$

si  $n \leq n_0$

*ordenación-inserción*( $v, i, j$ )

si no

$p \leftarrow \text{pivote}(v, i, j)$

*ordenación-rápida*( $v, i, p - 1$ )

*ordenación-rápida*( $v, p + 1, j$ )

Como algoritmo base se emplea una versión del algoritmo de ordenación por inserción directa capaz de trabajar con subvectores (incluso vacíos).

# Ordenación rápida

La descomposición toma un elemento como pivote y reorganiza el subvector para que a la izquierda del pivote queden los menores o iguales, y a su derecha los mayores. Se devuelve la posición final del pivote.

$pivot : v \times i \times j \rightarrow v \times p$

$\langle p, x \rangle \leftarrow \langle i, v[i] \rangle$

desde  $k \leftarrow i + 1$  hasta  $j$

si  $v[k] \leq x$

$p \leftarrow p + 1$

$v[p] \leftrightarrow v[k]$

$v[i] \leftarrow v[p]$

$v[p] \leftarrow x$

Esta versión pivota sobre el primer elemento, realizando  $n - 1$  comparaciones entre elementos del vector, con  $n = j - i + 1$ .

# Ordenación rápida

Sea  $t(n)$  el número de comparaciones entre elementos del vector.

Como  $p \in [i, j]$ , usaremos  $q = p - i + 1 \in [1, n]$ . En el caso  $n_0 = 1$ ,  $t(0) = t(1) = 0$  y para  $n > 1$ :

$$t(n) = t(q - 1) + t(n - q) + n - 1$$

El *mejor caso* se produce cuando la posición final del pivote es siempre la del elemento central: el tamaño de los subvectores está totalmente equilibrado.

$$q - 1 = n - q \implies q = \frac{n + 1}{2}$$

Esto sólo es exacto cuando  $n$  es impar, en tal caso

$q - 1 = n - q = (n - 1)/2$ . En general, se obtiene que:

$$t(n) = t\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) + t\left(\left\lceil \frac{n-1}{2} \right\rceil\right) + n - 1$$

# Ordenación rápida

Restringiendo el análisis a  $n = 2^k - 1$ :

$$t(2^k - 1) = 2t(2^{k-1} - 1) + 2^k - 2$$

$$T(k) = 2T(k-1) + 2^k - 2$$

Esto es similar a la ordenación por fusión. Se obtiene que  $t(n) \in \Omega(n \log n \mid n = 2^k - 1)$ . La restricción se puede eliminar del orden.

El *peor caso* se produce cuando la posición final del pivote es siempre un extremo: el tamaño de los subvectores está totalmente desequilibrado.

$$t(n) = t(n-1) + n - 1$$

Esta ecuación es idéntica a la de la ordenación por inserción directa. Como  $t(1) = 0$ , queda:

$$t(n) = \sum_{i=1}^n (i-1) = \frac{n(n-1)}{2} \in O(n^2)$$

# Ordenación rápida

Para analizar el *caso promedio*, suponemos que los elementos del vector son todos distintos y que cada permutación es equiprobable. Se demuestra que  $q$  tiene la misma probabilidad de tomar cualquier valor entre 1 y  $n$ :

$$\mathcal{P}(q = 1) = \dots = \mathcal{P}(q = n) = \frac{1}{n}$$

por lo que, cuando  $n > 0$ :

$$\begin{aligned} t(n) &= \sum_{i=1}^n \frac{1}{n} [t(i-1) + t(n-i) + n-1] \\ &= n-1 + \frac{1}{n} \sum_{i=1}^n [t(i-1) + t(n-i)] \\ &= n-1 + \frac{2}{n} \sum_{i=1}^{n-1} t(i) \end{aligned}$$

# Ordenación rápida

Restando  $nt(n)$  y  $(n-1)t(n-1)$  y despejando  $t(n)$ , se obtiene para  $n > 1$ :

$$t(n) = \frac{n+1}{n}t(n-1) + \frac{2(n-1)}{n}$$

Pero esta ERL no es de coeficientes constantes. Transformémosla:

$$t(n) = \left[ \prod_{i=1}^n \frac{i+1}{i} \right] u(n) = (n+1)u(n)$$

Aplicando el cambio:

$$u(n) = u(n-1) + \frac{2(n-1)}{(n+1)n}$$

y como  $u(1) = 0$ :

$$u(n) = \sum_{i=1}^n \frac{2(i-1)}{(i+1)i}$$

# Ordenación rápida

El término general es una función racional. Conviene descomponerla en fracciones simples:

$$\begin{aligned}\frac{a}{i+1} + \frac{b}{i} &= \frac{2(i-1)}{(i+1)i} \\ ai + b(i+1) &= 2(i-1) \\ (a+b)i + b &= 2i - 2\end{aligned}$$

Basta identificar términos en la identidad para obtener:

$$\begin{cases} a + b = 2 \\ b = -2 \end{cases} \implies a = 4$$

Por lo tanto, nos queda:

$$u(n) = 4 \sum_{i=1}^n \frac{1}{i+1} - 2 \sum_{i=1}^n \frac{1}{i}$$



# Ordenación rápida

Sea  $H_n = \sum_{i=1}^n \frac{1}{i}$ , entonces:

$$\begin{aligned}\sum_{i=1}^n \frac{1}{i+1} &= \sum_{i=1}^n \frac{1}{i} - 1 + \frac{1}{n+1} \\ &= H_n - 1 + \frac{1}{n+1}\end{aligned}$$

Por lo tanto:





$$\begin{aligned}u(n) &= 4 \left( H_n - 1 + \frac{1}{n+1} \right) - 2H_n \\ &= 2H_n - \frac{4n}{n+1}\end{aligned}$$

y deshaciendo el cambio:

$$t(n) = (n+1)u(n) = 2(n+1)H_n - 4n$$

Como  $H_n \in \Theta(\log n)$ , se obtiene que, en el caso promedio,  
 $t(n) \in \Theta(n \log n)$ .

# Referencias

-  Brassard, Gilles & Bratley, Paul.  
*Fundamentos de Algoritmia*.  
Prentice-Hall. 1997.
-  Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. & Stein, Clifford.  
*Introduction to Algorithms*.  
MIT Press. 2001. 2ª ed.
-  Manber, Udi.  
*Introduction to Algorithms. A Creative Approach*.  
Addison-Wesley. 1989.
-  Sedgewick, Robert.  
*Algorithms*.  
Addison-Wesley. 1988. 2ª edición.