Divide y vencerás

January 14, 2011

- Idea nueva: noción de tamaño de un problema.
- Idea clave: resolución de un problema a partir de soluciones del mismo problema pero de menor tamaño.
- Existe un tamaño crítico del problema que podemos resolver "directamente".
- Son típicamente recursivos:
  - Código sencillo y legible: poco coste de mantenimiento.
  - Mayor tiempo de ejecución que su equivalente iterativo.
  - Posibilidad de problemas a causa de la complejidad espacial $\longrightarrow$ desbordamiento de pila.
  - Los k subproblemas no deben solaparse $\longrightarrow$ En caso contrario, complejidad exponencial.

Esquema de la metodología de divide y vencerás:

$$\textit{Problema} \leftrightarrow \textit{tamaño } n$$
$$\Downarrow$$
$$\textit{Fase de división: } k \textit{ problemas} \leftrightarrow \textit{tamaño } n_k \ (\forall k : 0 \leq n_k < n)$$
$$\Downarrow$$
$$\textit{Fase de resolución: } k \textit{ problemas} \rightarrow \textit{solución al problema } k\textit{: } S_k$$
$$\Downarrow$$
$$\textit{Fase de combinación: } S_k \rightarrow S_{global}$$

Consideremos los números de Fibonacci:

$$F(N) = \begin{cases} N, \text{ si } N = 0 \text{ ó } N = 1 \\ \\ F(N-1) + F(N-2) \text{ , si } N > 1 \end{cases}$$

Algoritmo

fun $fib(N)$
  si $N = 0 \vee N = 1$ **entonces**
    $Fib \leftarrow N$
  si no
    $Fib \leftarrow Fib(N-1) + Fib(N-2)$
  fin si
  devolver Fib
fin fun

Tiempo de ejecución:

$$T(N) = \begin{cases} cte, \text{ si } N = 0 \text{ ó } N = 1 \\ \\ T(N-1) + T(N-2)\text{ , si } N > 1 \end{cases}$$

$$T(n) - T(n-1) - T(n-2) = 0$$
$$\Downarrow$$
$$X^2 - X - 1 = 0$$
$$\Downarrow$$
$$X_1 = \frac{1+\sqrt{5}}{2} \; X_2 = \frac{1-\sqrt{5}}{2}$$
$$\Downarrow$$
$$T(n) = cte(\frac{1+\sqrt{5}}{2})^n + cte(\frac{1-\sqrt{5}}{2})^n$$
$$\Downarrow$$
$$T(n) \in \Theta(cte^n) \Rightarrow \text{Ruinoso!!!}$$

Cálculo del factorial:

$$Fact(N) = \begin{cases} 1, \text{ si } N = 0 \\ \\ N * Fact(N-1) \text{ , si } N > 0 \end{cases}$$

Algoritmo

fun $fact(N)$
  si $N = 0$ **entonces**
    $fact \leftarrow 1$
  si no
    $fact \leftarrow N * fact(N - 1)$
  fin si
  devolver $fact$
fin fun

Tiempo de ejecución:

$$T(N) = \begin{cases} cte, \text{ si } N = 0 \\ \\ T(N-1) + cte \text{ , si } N > 0 \end{cases}$$

$$T(n) - T(n-1) = 0$$
$$\Downarrow$$
$$X - 1 = 0$$
$$\Downarrow$$
$$X = 1$$
$$\Downarrow$$
$$T(n) = cte * 1^n$$
$$\Downarrow$$
$$T(n) \in \Theta(1) \Rightarrow \text{Prodigioso?!?!}$$

Conclusiones:

- No podemos despreciar la constante $\rightarrow$ Ya no tenemos una recurrencia homogénea.
- Entonces tenemos que $b = 1$ y $d = 0$. Debemos resolver el polinomio:

$$(X-1)(X-1) = 0 \rightarrow r_i = 1 \text{ con } m = 2.$$

- Entonces tenemos:

$$T(n) = cte * n^0 * 1^n + cte * n^1 * 1^n = cte + cte * n \Rightarrow T(n) \in \Theta(n)$$

Dado una lista de N elementos ordenados, determinar en que posición se el elemento X.

- **Planteamiento**:
  - Mirando el valor de la posición central podemos descartar la búsqueda en una mitad de la lista $\longrightarrow$ Reducimos el problema a un subproblema de menor tamaño.
  - Para una sublista de tamaño 1 el problema es trivial.
- La primera llamada es: $busqueda - binaria(1, N, X, elem)$.

Algoritmo

fun $busqueda - binaria(iz, de, X, elem[1..N])$
  //iz,de: extremos de la búsqueda
  $k \leftarrow div((iz + de)/2)$ *//parte entera*
  si $elem[K] = X$ **entonces**
    $busqueda \leftarrow k$
  si no
    si $iz = de$ **entonces**
      $busqueda \leftarrow 0$ *//no existe*
    si no
      si $elem[k] > X$ **entonces**
        $busqueda - binaria(iz, k - 1, x, elem)$
      si no
        si $elem[k] < X$ **entonces**
          $busqueda - binaria(k + 1, de, x, elem)$
        fin si
      fin si
    fin si
  fin si
  devolver $busqueda$
fin fun

Estudio de la complejidad:
En el peor de los casos se ejecuta la llamada recursiva. Sea M el número de elementos de la búsqueda ($M = de - iz + 1$):

$$T(M) = T(M/2) + cte$$

Hacemos el cambio de variable: $M = 2^q \iff q = log_2 M$.

$$T(2^q) = T(2^{q-1}) + cte$$
$$\Downarrow$$
$$\hat{T}(q) = \hat{T}(q-1) + cte$$
$$\Downarrow$$
$$\hat{T}(q) = q * cte + cte$$
$$\Downarrow$$
$$T(M) = cte * log_2 M + cte$$

Problema

Dado un vector ordenado de N elementos, $V[1..N]$*. Diseña un algoritmo que determine si existe algún elemento del vector que cumpla:* $V[i] = i$ *donde:* $1 \leq i \leq N$

Algoritmo

fun $coincide(v[1..N], c, f)$
  //c,f: límites de la búsqueda
  casos
    $c > f$:
      $coincide \leftarrow falso$
    $c = f$:
      $coincide \leftarrow v[c] = c$
    $c < f$:
      $m \leftarrow div(c + f)/2$
      casos
        $m < v[m]$:
          $coincide \leftarrow coincide(v, c, m - 1)$
        $m = v[m]$:
          $coincide \leftarrow cierto$
        $m > v[m]$:
          $coincide \leftarrow coincide(v, m + 1, f)$
      fin casos
  fin casos
  devolver $coincide$
fin fun

- **Objetivo**: Ordenar una lista de n elementos.
- Sea una lista de elementos $\{L_i\}$ $i : 1..n$. Diremos que está parcialmente ordenada si:

$$\{L_i\} = \{L_q\} \cup \{L_r\}$$
$$i : 1..n$$
$$q : 1..k$$
$$r : k+1..n \text{ con } k \neq n$$

donde $L_q \leq L_t$ y $L_t \leq L_r$ con $L_t \in \{L_i\}$.
- Podemos obtener listas parcialmente ordenadas fácilmente para cada sublista.
- Existe un tamaño crítico ($n = 2$ o $n = 3$) para el cual una lista parcialmente ordenada es, además, un lista ordenada.

- La ordenación de una lista se puede realizar a través de la ordenación parcial de sus sublistas.
- No hay solapamiento.
- Fase de división:
  - Elegimos un pivote al azar: L_t.
  - Movemos todos los elementos menores que él a la parte baja de la lista y los mayores a la parte alta.
  - Mediante dos índices recorremos la lista realizando los cambios oportunos.
  - Cuando se crucen los índices tendremos una lista parcialmente ordenada.
  - Repetimos las operaciones con cada sublista.

Algoritmo

fun $quicksort(L[1..N], iz, der)$
  //Tomamos como pivote el elemento de la mitad
  $x \leftarrow L[DIV(iz + der)/2]$
  $I \leftarrow iz$
  $J \leftarrow der$
  repetir
    mientras $L[I] < x$ **hacer**
      $I++$
    fin mientras
    mientras $L[J] > x$ **hacer**
      $J--$
    fin mientras
    si $I \leq J$ **entonces**
      $aux \leftarrow L[I]$
      $L[I] \leftarrow L[J]$
      $L[J] \leftarrow aux$
      $I++$
      $J--$
    fin si
  hasta $I > J$
  si $iz < J$ **entonces**
    $quicksort(L, iz, J)$
  fin si
  si $I < der$ **entonces**
    $quicksort(L, I, der)$
  fin si
fin fun

La primera llamada sería: $quicksort(L, 1, N)$

Traza. Caso eficiente:

Lista inicial:

39 50 20 41_p 80 79 65 23

Fin fase de división:

39 23 20 || 41_p || 80 79 65 50

39 23_p 20 || 41 || 80 79_p 65 50

Fin fase de división:

20 || 23_p|| 39 || 41 || 50 65_p || 79 80

Fin fase de división:

20 ||23|| 20 || 41 || 50_p 65|| 79_p 80

Final:

20 ||23|| 39 || 41 || 50 || 65 ||79|| 80

Rendimiento:

- Valores de la mediana cercanos al valor medio: dos llamadas recursivas de longitud mitad.
- $T(n) = cte * n + 2T(n/2)$

$$T(n) - 2T(n/2) = cte * n \text{ cambio de variable: } n = 2^q$$
$$T(2^q) - 2T(2^{q-1}/2) = cte * 2^q$$
$$\Updownarrow$$
$$\widehat{T}(q) - 2\widehat{T}(q-1) = cte * 2^q$$
Recurrencia no homogénea con $b = 2$ y $p(q) = cte$

Polinomio característico: $(x-2)^2$
$$\widehat{T}(q) = cte * 2^q + cte * q * 2^q$$
$$T(n) = cte * n + cte * log(n) * n \text{ luego: } T(n) = \Theta(n * log(n))$$

Traza. Caso ineficiente:

Lista inicial:

20 60 39 90_p 70 75 30 89

Fin fase de división:

20 60 39 89 70 75 30 || 90

20 60 39 89_p 70 75 30 || 90

Fin fase de división:

20 60 39 30 70 75 || 89 || 90

20 60 39 30_p 70 75 ||89 || 90

Fin fase de división:

20 30 ||39|| 60 70 75 || 89 || 90

20_p 30 ||39|| 60 70_p 75 || 89 || 90

Final:

20 ||30 ||39|| 60 || 70 || 75 || 89 || 90

Rendimiento:

- Valores de la mediana cercanos al valor medio: una llamada recursiva de longitud $n-1$.
- $T(n) = cte * n + T(n-1)$

$$T(n) - T(n-1) = cte * n$$

Recurrencia no homogénea con $b = 1$ y $p(q) = cte * n$

Polinomio característico: $(x-1)^3$

$T(n) = cte * 1^n + cte * n * 1^n + cte * n^2 * 1^n$ luego:

$$T(n) = \Theta(n^2)$$

Problema

Sean N tuercas y N tornillos. Queremos emparejar cada tuerca con su tornillo. Como su tamaño es tan parecido no podemos ordenarlos por tamaños. La única acción posible con un tornillo (o tuerca) es probar con una tuerca (o tornillo) obteniendo tres posibles resultados: encajan, tornillo mayor que la tuerca, tornillo menor que la tuerca. Diseña un algoritmo de divide y vencerás que solucione el problema.

Planteamiento:

- Tomar un tornillo (o tuerca) y probarlo con todas las tuercas (o tornillos) estableciendo tres conjuntos:
    - El de los iguales.
    - El de los mayores.
    - El de los menores.
- Tomar una tuerca que encaje con el tornillo y probar con todos los tornillos estableciendo los conjuntos anteriores para los tornillos.
- De esta forma podemos descartar emparejamientos entre distintas parejas de conjuntos.
- Aplicamos la misma operación a las parejas de conjuntos "menores" y "mayores".
- Tenemos definida la solución en términos de subproblemas de menor tamaño.
- El caso base se corresponde con el caso en el que en cada subconjunto solo hay 1 elemento.

Variables:

- Para cada tipo de pieza tenemos una lista: $L[1..N]$.
- En cada momento del proceso podemos considerar la lista dividida en en los siguientes subconjuntos según su relación con el pivote:

$$\underbrace{L_c \ldots L_{i-1}}_{menores}, \underbrace{L_i \ldots L_{k-1}}_{iguales}, \underbrace{L_k \ldots L_j}_{no\ procesados}, \underbrace{L_{j+1} \ldots L_f}_{mayores}$$

  - $L[c..i-1]$: elementos menores.
  - $L[i..k-1]$: elementos iguales.
  - $L[k..j]$: elementos no clasificados.
  - $L[j+1..f]$: elementos mayores.
- Inicialmente, para el conjunto de los no procesados: $k = 1$ $j = N$.
- Al final: el conjunto de los no procesados es vacío.

Algoritmo

proc $partition(L[1..N], c, f, pivote, I, J)$
//c,f: límites del procesamiento
//i,j: resultado de la partición
$I \leftarrow c$
$K \leftarrow c$
$J \leftarrow f$
mientras $K \leq J$ **hacer**
//K: elemento a procesar
casos
$L[K] < pivote$ **:**
$aux \leftarrow L[I]$
$L[I] \leftarrow L[K]$
$L[K] \leftarrow aux$
$I++$
$K++$
$L[K] = pivote$ **:**
$K++$
$L[K] > pivote$ **:**
$aux \leftarrow L[J]$
$L[J] \leftarrow L[K]$
$L[K] \leftarrow aux$
$J--$
fin casos
fin mientras
fin proc

Algoritmo

proc $emparejar(tornillos[1..N], tuercas[1..N], c, f)$
  si $c < f$ **entonces**
    // fase de particion:
    $particion(tuercas, c, f, tornillos[c], I, J)$
    // Tomamos como pivote el primer tornillo:
    $particion(tornillos, c, f, tuercas[I], p, q)$
    // Emparejamos:
    $emparejar(tornillos, tuercas, c, I - 1)$
    $emparejar(tornillos, tuercas, J + 1, f)$
  fin si
fin proc

Traza:

Tuercas: 40 18 62 25 10 5 7 82 99
Tornillo pivote: 40
Tuercas: 40_{ik} 18 62 25 10 5 7 82 99_j
Tuercas: 40_i 18_k 62 25 10 5 7 82 99_j
Tuercas: 18 40_i 62_k 25 10 5 7 82 99_j
Tuercas: 18 40_i 99_k 25 10 5 7 82_j 62
Tuercas: 18 40_i 82_k 25 10 5 7_j 99 62
Tuercas: 18 40_i 7_k 25 10 5_j 82 99 62
Tuercas: 18 7 40_i 25_k 10 5_j 82 99 62
Tuercas: 18 7 25_i 40_i 10_k 5_j 82 99 62
Tuercas: 18 7 25 10 40_i 5_{jk} 82 99 62
Tuercas: 18 7 25 10 5 40_{ij} 82_k 99 62

Definición

Dado un vector de N elementos, la mediana es el valor del elemento que ocupa la posición: $(N+1) div\ 2$.

Problema

Dados dos vectores A y B de tamaño N ordenados de forma no decrecientemente, implementa un algoritmo que obtenga la mediana de la fusión de ambos lo más eficientemente posible.

Planteamiento:

- Algoritmo inmediato: realizar la fusión de ambos y mirar en la posición $(2N+1) div\ 2$. Complejidad: $\Theta(N)$.
- Solamente podemos rebajar la complejidad si renunciamos a realizar la fusión de los dos vectores $\longrightarrow$ Nos referiremos a la mediana de la fusión "hipotética" o posible.
- Consideraciones de la mediana de la fusión hipotética, m, a partir de las medianas de los dos vectores

  (m_a y m_b):
  - Si $m_a = m_b$ entonces: $m = m_a = m_b$.
  - Si $m_a > m_b$ entonces: $m_a \geq m \geq m_b$.
- Idea clave: Si quitamos el mismo número de elementos por delante y detrás de la mediana no varía $\longrightarrow$ Remitimos la solución de un problema a un subproblema.
- Casos base (tamaños críticos): $N = 1$ y $N = 2$.

Algoritmo

proc $mediana(A[1..N], B[1..N], c_a, f_a, c_b, f_b)$
  $//c_a, f_a, c_b, f_b$: *límites del procesamiento*
  si $(c_a = f_a) \wedge (c_b = f_b)$ **entonces**
    $mediana \leftarrow min\{A[f_a], B[f_b]\}$
  si no
    $num \leftarrow f_a - c_a + 1$
    si $num = 2$ **entonces**
      si $A[f_a] < B[c_b]$ **entonces**
        $mediana \leftarrow A[f_a]$
      si no
        si $B[f_b] < A[c_a]$ **entonces**
          $mediana \leftarrow B[f_b]$
        si no
          $mediana \leftarrow max\{A[c_a], B[c_b]\}$
        fin si
      fin si
    si no
      ...
    fin si
  fin si
fin proc

Algoritmo

proc $mediana(A[1..N], B[1..N], c_a, f_a, c_b, f_b)$
...
//Desplazamiento respecto del primer elemento de M_a *y* M_b
$desp \leftarrow (num - 1)div2$
$M_a \leftarrow c_a + desp$
$M_b \leftarrow c_b + desp$
si $A[M_a] = B[M_b]$ **entonces**
$mediana \leftarrow A[M_a]$
si no
si $A[M_a] < B[M_b]$ **entonces**
$mediana(A, B, f_a - desp, f_a, c_b, c_b + desp)$
si no
$mediana(A, B, c_a, c_a + desp, f_b - desp, f_b)$
fin si
fin si
fin proc