Algoritmos: Seminario 3

Aplicación del Teorema de resolución de recurrencias Divide y Vencerás

José María Casanova, Elena Hernández, Alberto Valderruten

Dept. de Computación, Universidade da Coruña

jcasanova@udc.es, elena.hernandez@udc.es, alberto.valderruten@udc.es

UNIVERSIDADE DA CORUÑA

- **Teorema de resolución de recurrencias Divide y Vencerás**

$$T(n) = \ell T(n/b) + cn^k, n > n_0 \qquad (1)$$

Con $\ell \geq 1, b \geq 2, k \geq 0, n_0 \geq 1 \in \mathbb{N}$ y $c > 0 \in \mathbb{R}$,
cuando n/n_0 es potencia exacta de b ($n \in \{bn_0, b^2 n_0, b^3 n_0 \dots\}$).

- **Teorema Divide y Vencerás**:
Si una recurrencia es de la forma (1), se aplica

$$T(n) = \begin{cases} \theta(n^k) & si \quad \ell < b^k \\ \theta(n^k logn) & si \quad \ell = b^k \\ \theta(n^{log_b \ell}) & si \quad \ell > b^k \end{cases} \qquad (2)$$

En análisis de algoritmos, se suelen manejar desigualdades:
$T(n) \leq \ell T(n/b) + cn^k, n > n_0$ con n/n_0 potencia exacta de b

$$\Rightarrow T(n) = \begin{cases} O(n^k) & si \quad \ell < b^k \\ O(n^k logn) & si \quad \ell = b^k \\ O(n^{log_b \ell}) & si \quad \ell > b^k \end{cases}$$

1. operación elemental = 1 $\leftrightarrow$ Modelo de Computación
2. **secuencia**: $S_1 = O(f_1(n)) \wedge S_2 = O(f_2(n))$
   $\Rightarrow \boxed{S_1; S_2} = O(f_1(n) + f_2(n)) = O(max(f_1(n), f_2(n)))$
   - También con Θ
3. **condición**: $B = O(f_B(n)) \wedge S_1 = O(f_1(n)) \wedge S_2 = O(f_2(n))$
   $\Rightarrow \boxed{\textbf{si } B \textbf{ entonces } S_1 \textbf{ sino } S_2} = O(max(f_B(n), f_1(n), f_2(n)))$
   - Si $f_1(n) \neq f_2(n)$ y $max(f_1(n), f_2(n)) > f_B(n) \leftrightarrow$ **Peor caso**
   - ¿Caso medio? $\rightarrow f(n)$: promedio de f_1 y f_2 ponderado con las frecuencias de cada rama $\rightarrow O(max(f_B(n), f(n)))$
4. **iteración**: $B; S = O(f_{B,S}(n)) \wedge$ n^o iter$= O(f_{iter}(n))$
   $\Rightarrow \boxed{\textbf{mientras } B \textbf{ hacer } S} = O(f_{B,S}(n) * f_{iter}(n))$
   ssi el coste de las iteraciones no varía, sino: $\sum$ costes indiv.
   $\Rightarrow \boxed{\textbf{para } i \leftarrow x \textbf{ hasta } y \textbf{ hacer } S} = O(f_S(n) *$n^o iter$)$
   ssi el coste de las iteraciones no varía, sino: $\sum$ costes indiv.
   - B es comparar 2 enteros $= O(1)$; n^o iter $= y - x + 1$

- **Problema de la Suma de la Subsecuencia Máxima**
  $a_1..a_n \rightarrow \sum_{k=i}^{j} a_k$ máxima?
  Ejemplo: $SSM(-2, 11, -4, 13, -5, -2) = 20$ [2..4]
- *SSM* **recursiva**: estrategia $\boxed{\text{Divide y Vencerás}}$
  Divide la entrada en 2 *mitades* $\rightarrow$ 2 soluciones recursivas
  Vence usando las 2 soluciones $\rightarrow$ solución para entrada original

  $$\text{La } SSM \text{ puede estar: } \begin{cases} \text{- en la 1}^{\text{a}} \text{ mitad} \\ \text{- en la 2}^{\text{a}} \text{ mitad} \\ \text{- entre las 2 mitades} \end{cases}$$

  Las dos primeras soluciones son las obtenidas recursivamente.
  La 3ª solución se obtiene sumando:
  - la *SSM* de la 1ª mitad *que incluye el extremo derecho*, y
  - la *SSM* de la 2ª mitad *que incluye el extremo izquierdo*.

Suma de la Subsecuencia Máxima (2) - *SSM* recursiva

```
      función SSM ( a[1..n] ) : valor              función interfaz
        devolver SSM recursiva (a, 1, n)
      fin función

      función SSM recursiva (var a[1..n], izq, der) : valor
{1}     si izq = der entonces
{2}        si a[izq] > 0 entonces
{3}           devolver a[izq]                      caso base: si >0, es SSM
           sino
{4}           devolver 0
           fin si
        sino
{5}        Centro := (izq + der) div 2 ;
{6}        Primera solución := SSM recursiva (a, izq, Centro) ;
{7}        Segunda solución := SSM recursiva (a, Centro + 1, der) ;
```

Suma de la Subsecuencia Máxima (3) - *SSM* recursiva

```
        Suma máxima izquierda := 0 ; Suma izquierda := 0 ;
        para i := Centro hasta izq paso -1 hacer
          Suma izquierda := Suma izquierda + a[i] ;
          si Suma izquierda > Suma máxima izquierda entonces
             Suma máxima izquierda := Suma izquierda
        fin para ;

        Suma máxima derecha := 0 ; Suma derecha := 0 ;
        para i := Centro + 1 hasta der hacer
          Suma derecha := Suma derecha + a[i] ;
          si Suma derecha > Suma máxima derecha entonces
             Suma máxima derecha := Suma derecha
        fin para ;

        devolver max (Primera solución, Segunda solución,
                      Suma máxima izquierda + Suma máxima derecha)
   fin si
fin función
```

Suma de la Subsecuencia Máxima - Ejercicio

- Entender y ejecutar el algoritmo SSM recursivo con un ejemplo y dibujar el arbol de recursividad.
- Analizar el algoritmo SSM planteando la relación de recurrencia y aplicando teorema de resolución de recurrencias divide y vencerás.

- **Análisis**:
  Caso base: $\{1\text{-}4\} \Rightarrow T(1) = \Theta(1)$
  Ciclos $\{9\text{-}12\}$ y $\{14\text{-}17\}$: $\Theta(n)$ *en conjunto*: $a_1..a_n$
  Llamadas recursivas $\{6\}$ y $\{7\}$: $T(n/2)$ cada una (aprox.)
  Resto = $\Theta(1)$: se puede ignorar frente a $\Theta(n)$
  Relación de recurrencia: $\begin{cases} T(1) = 1 \\ T(n) = 2T(n/2) + n, n > 1 (*) \end{cases}$
- **Aplicando teoremas:**

Teorema de resolución de recurrencias Divide y Vencerás

$T(n) = lT(n/b) + cn^k, n > n_0,$
con $l \geq 1$, $b \geq 2$, $c > 0 \in \mathbb{R}$, $k \geq 0 \in \mathbb{N}$, $n_0 \geq 1 \in \mathbb{N}$
$\{l = 2, b = 2, c = 1, k = 1, n_0 = 1\}$: caso $l = b^k$
$\Rightarrow T(n) = \Theta(n^k logn) \Rightarrow T(n) = \Theta(nlogn)$

- **Observación**: pasar el vector *a* **por referencia** (var), sino:

  Sea $R(n)$: nº de copias de *a*: $\begin{cases} R(1) = 0 \\ R(n) = 2R(n/2) + 2, n > 1 \end{cases}$

  $\Rightarrow R(n) = 2n - 2$ copias $*\Theta(n)$ cada una $\Rightarrow T(n) = \Theta(n^2)$

  También la complejidad espacial sería cuadrática!

Búsqueda Binaria (1)

- Ejemplo de *algoritmo logarítmico*
- Dados x y un vector *ordenado* $a_1, a_2, \ldots a_n$ de enteros,

$$\text{devolver: } \begin{cases} i \text{ si } \exists a_i = x \\ \text{“elemento no encontrado”} \end{cases}$$

$\rightarrow$ Comparar x y a_{medio}, con $medio = (i + j) div2$, siendo $a_i..a_j$ el *espacio de búsqueda*:

1. $x = a_{medio}$: terminar (interrupción)
2. $x > a_{medio}$: seguir buscando en $a_{medio+1}..a_j$
3. $x < a_{medio}$: seguir buscando en $a_i..a_{medio-1}$

- ¿n^{o} iter? $\leftrightarrow$ evolución del tamaño d del espacio de búsqueda

Invariante: $d = j - i + 1$

$$\text{¿Cómo decrece d? } \begin{cases} i \leftarrow medio + 1 \\ j \leftarrow medio - 1 \end{cases}$$

- *Peor caso*: se alcanza la terminación “normal” del bucle $\equiv i > j$

Búsqueda Binaria (2)

```
función Búsqueda Binaria (x, a[1..n]) : posición
{a: vector ordenado de modo no decreciente}

   i := 1 ; j := n ;                        {espacio de búsqueda: i..j}
   mientras i <= j hacer
      medio := (i + j) div 2 ;
      si a[medio] < x entonces
         i := medio + 1
      sino si a[medio] > x entonces
         j := medio - 1
      sino devolver medio                   {se interrumpe el bucle}
   fin mientras;
   devolver "elemento no encontrado"        {fin normal del bucle}
fin función
```

Búsqueda Binaria (3) - Análisis del peor caso

- Sea $< d, i, j >$ iteración $< d', i', j' >$:
  1. $i \leftarrow medio + 1$:
     $$i' = (i + j) div 2 + 1$$
     $$j' = j$$
     $$\begin{aligned} d' = j' - i' + 1 \quad & = j - (i + j) div 2 - 1 + 1 \\ & \leq j - (i + j - 1)/2 \\ & = (j - i + 1)/2 \\ & = d/2 \end{aligned}$$
     $$\boxed{\rightarrow d' \leq d/2}$$
  2. $j \leftarrow medio - 1$:
     $$i' = i$$
     $$j' = (i + j) div 2 - 1$$
     $$\begin{aligned} d' = j' - i' + 1 \quad & = (i + j) div 2 - i - 1 + 1 \\ & \leq (i + j)/2 - i \\ & < (j - i + 1)/2 \\ & = d/2 \end{aligned}$$
     $\boxed{\rightarrow d' < d/2}$ *(decrece más rápido)*

Búsqueda Binaria (4) - Análisis del peor caso

- ¿$T(n)$? Sea d_l: d *después* de la l-ésima iteración

  $$\begin{cases} d_0 = n \\ d_l \leq d_{l-1}/2 \ \forall l \geq 1 \end{cases} \quad \text{(inducción)} \rightarrow d_l \leq n/2^l$$

  hasta $d < 1 \rightarrow l = \lceil log_2 n \rceil + 1 = O(logn)$ iteraciones

  Cada iteración es $\Theta(1)$ (reglas) $\Rightarrow T(n) = O(logn)$

- **Razonamiento alternativo**: *pensar en versión recursiva*

  $$T(n) = \begin{cases} 1 & \text{si } n = 0, 1 \\ T(n/2) + 1 & \text{si } n > 1 \end{cases}$$

  Teorema Divide y Vencerás: $l = 1, b = 2, c = 1, k = 0, n_0 = 1$

  Caso $l = b^k \Rightarrow T(n) = \Theta(n^k logn) \rightarrow T(n) = \Theta(logn)$

- **Conclusiones**:
  - Pensar en versión recursiva puede ser otro recurso útil
  - es Divide y Vencerás? $\rightarrow$ $\boxed{\text{Algoritmos de reducción}}$ $(l = 1)$
  - $T(n) = \Theta(logn) \leftrightarrow$ los datos ya están en memoria (Modelo de Computación)

Búsqueda Binaria - Ejercicio

- Diseñar una versión recursiva del algoritmo de búsqueda binaria.
- Analizar el algoritmo recursivo resultante aplicando el teorema de resolución de recurrencias divide y vencerás.

Búsqueda Binaria Recursiva (1)

```
     función Búsqueda Binaria (x, a[1..n]): posición {función interfaz}
     {a: vector ordenado de modo no decreciente}
       devolver Búsqueda Binaria Recursiva (x, a, 1, n)
     fin función

     función Búsqueda Binaria Recursiva (x, var a[1..n], i, j): posición
     {espacio de búsqueda: i..j}
{1}     si i > j entonces devolver 0 {"elemento no encontrado"}
{2}     sino si i = j entonces
{3}        si a[i] = x entonces devolver i
{4}        sino devolver 0 {"elemento no encontrado"}
        sino {i < j}
{5}        medio := (i + j) div 2 ;
{6}        si a[medio] < x entonces
{7}           devolver Búsqueda Binaria Recursiva (x, a, medio+1, j)
{8}        sino si a[medio] > x entonces
{9}           devolver Búsqueda Binaria Recursiva (x, a, i, medio-1)
{10}       sino devolver medio {a[medio] = x}
     fin función
```

Búsqueda Binaria Recursiva (2) - Análisis del peor caso

- **Peor caso**:
  ocurre cuando no se encuentra el elemento buscado
  $$T(n) = \begin{cases} 1 & \text{si } n = 0, 1 \\ T(n/2) + 1 & \text{si } n > 1 \end{cases}$$
  Importante: observar que sólo se produce una llamada recursiva
  Teorema de resolución de recurrencias Divide y Vencerás:
  $l = 1, b = 2, c = 1, k = 0, n_0 = 1$
  (se comprueban las condiciones)
  Caso $l = b^k \Rightarrow T(n) = \Theta(n^k logn) \rightarrow T(n) = \Theta(logn)$
- **Ejercicio**: mejor caso?

Bibliografía

- G. Brassard y P. Bratley,
  Fundamentals of Algorithmics, Prentice Hall 1996.
- G. Brassard y P. Bratley,
  Fundamentos de Algoritmia, Prentice Hall 1997.