

Algoritmos y Estructuras de Datos II

Recurrencias homogéneas y no homogéneas

31 de marzo de 2014

Contenidos

- 1 Repaso
 - Algoritmos de ordenación
 - Notación \mathcal{O}
 - Algoritmos divide y vencerás
 - Recurrencias divide y vencerás
 - Ejemplo: búsqueda binaria
- 2 Recurrencias homogéneas
 - Método de resolución
- 3 Recurrencias no homogéneas
 - Método de resolución

Algoritmos de ordenación

- Algoritmos elementales:
 - Ordenación por selección
 - Bubble sort
 - Cocktail sort
 - Ordenación por inserción
 - Shell sort
- Algoritmos eficientes:
 - Ordenación por intercalación
 - versión iterativa
 - Ordenación rápida
 - variantes sobre el procedimiento pivot

Notación \mathcal{O}

Sea $g : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$,

- $\mathcal{O}(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} \mid \exists c > 0. \forall n \in \mathbb{N}. f(n) \leq cg(n)\}$
- $\Omega(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} \mid \exists c > 0. \forall n \in \mathbb{N}. f(n) \geq cg(n)\}$
- $\Theta(g(n)) = \mathcal{O}(g(n)) \cap \Omega(g(n))$

Algunas equivalencias

- $\mathcal{O}(f(n)) \subseteq \mathcal{O}(g(n))$ es equivalente a:
 - $g(n) \in \Omega(f(n))$
 - $f(n) \in \mathcal{O}(g(n))$
 - $\Omega(g(n)) \subseteq \Omega(f(n))$
- $\mathcal{O}(f(n)) \subset \mathcal{O}(g(n))$ es equivalente a:
 - $\Omega(g(n)) \subset \Omega(f(n))$
 - $f(n) \in \mathcal{O}(g(n))$ pero $g(n) \notin \mathcal{O}(f(n))$.
- $\mathcal{O}(f(n)) = \mathcal{O}(g(n))$ es equivalente a:
 - $\Omega(g(n)) = \Omega(f(n))$
 - $f(n) \in \mathcal{O}(g(n))$ y $g(n) \in \mathcal{O}(f(n))$.
 - $f(n) \in \Omega(g(n))$ y $g(n) \in \Omega(f(n))$.
 - $f(n) \in \Theta(g(n))$.

Propiedades

- Constantes multiplicativas no afectan.
- Términos de crecimiento despreciable no afectan.
- Sean $a, b > 1$, $\mathcal{O}(\log_a n) = \mathcal{O}(\log_b n)$.
- Regla del límite. Jerarquía.
- Sea $\forall^\infty n \in \mathbb{N}. f(n) > 0$. Entonces
 $g(n) \in \mathcal{O}(h(n)) \iff f(n)g(n) \in \mathcal{O}(f(n)h(n))$.
- Sea $\lim_{n \rightarrow \infty} h(n) = \infty$. Entonces
 $f(n) \in \mathcal{O}(g(n)) \implies f(h(n)) \in \mathcal{O}(g(h(n)))$.

Jerarquía

$$\begin{aligned} \mathcal{O}(1) &\subset \mathcal{O}(\log(\log(\log n))) \subset \mathcal{O}(\log(\log n)) \subset \mathcal{O}(\log n) \subset \mathcal{O}(n^{0.001}) \subset \\ &\subset \mathcal{O}(n) \subset \mathcal{O}(n \log n) \subset \mathcal{O}(n^{1.001}) \subset \mathcal{O}(n^{100}) \subset \mathcal{O}(1.01^n) \subset \\ &\subset \mathcal{O}(n^{100} * 1.01^n) \subset \mathcal{O}(1.02^n) \subset \mathcal{O}(100^n) \subset \mathcal{O}(10000^n) \subset \\ &\subset \mathcal{O}((n-1)!) \subset \mathcal{O}(n!) \subset \mathcal{O}((n+1)!) \subset \mathcal{O}(n^n) \end{aligned}$$

Algoritmo divide y vencerás

Forma general

```
fun DV(x) ret y
  if x suficientemente pequeño o simple then y:= ad_hoc(x)
  else descomponer x en  $x_1, x_2, \dots, x_a$ 
    for i:= 1 to a do  $y_i := DV(x_i)$  od
    combinar  $y_1, y_2, \dots, y_a$  para obtener la solución y de x
  fi
end
```

Normalmente los x_i son **fracciones** de x :

$$|x_i| = \frac{|x|}{b}$$

para algún b fijo.

Algoritmo divide y vencerás

Conteo

Si queremos contar el costo computacional (número de operaciones) $t(n)$ de la función DV obtenemos:

$$t(n) = \begin{cases} c & \text{si la entrada es pequeña o simple} \\ a * t(n/b) + g(n) & \text{en caso contrario} \end{cases}$$

si c es una constante que representa el costo computacional de la función `ad_hoc` y $g(n)$ es el costo computacional de los procesos de descomposición y de combinación.

Esta definición de $t(n)$ es recursiva (como el algoritmo DV), se llama **recurrencia**. Existen distintos tipos de recurrencia. Ésta se llama **recurrencia divide y vencerás**.

Potencias de b

- Para



$$t(n) = \begin{cases} c & \text{si la entrada es pequeña o simple} \\ a * t(n/b) + g(n) & \text{en caso contrario} \end{cases}$$

- con $g(n) \in \mathcal{O}(n^k)$,
- demostramos

$$t(n) \in \begin{cases} \mathcal{O}(n^{\log_b a} | n \text{ potencia de } b) & \text{si } a > b^k \\ \mathcal{O}(n^k \log n | n \text{ potencia de } b) & \text{si } a = b^k \\ \mathcal{O}(n^k | n \text{ potencia de } b) & \text{si } a < b^k \end{cases}$$

- Similar resultado vale reemplazando \mathcal{O} por Ω en las 4 ocurrencias.
- Similar resultado vale reemplazando \mathcal{O} por Θ en las 4 ocurrencias.

Recurrencias divide y vencerás

- Para



$$t(n) = \begin{cases} c & \text{si la entrada es pequeña o simple} \\ a * t(n/b) + g(n) & \text{en caso contrario} \end{cases}$$

si $t(n)$ es eventualmente no decreciente, y $g(n) \in \mathcal{O}(n^k)$, entonces



$$t(n) \in \begin{cases} \mathcal{O}(n^{\log_b a}) & \text{si } a > b^k \\ \mathcal{O}(n^k \log n) & \text{si } a = b^k \\ \mathcal{O}(n^k) & \text{si } a < b^k \end{cases}$$

- Similar resultado vale reemplazando \mathcal{O} por Ω en las 4 ocurrencias.
- Similar resultado vale reemplazando \mathcal{O} por Θ en las 4 ocurrencias.

Ejemplo: búsqueda binaria

{Pre: $1 \leq \text{izq} \leq n+1 \wedge 0 \leq \text{der} \leq n \wedge a$ ordenado}

fun binary_search_rec (a: **array**[1..n] **of** T, x:T, izq, der : **nat**) **ret** i:na

var med: **nat**

if izq > der \rightarrow i = 0

izq \leq der \rightarrow med:= (izq+der) \div 2

if x < a[med] \rightarrow i:= binary_search_rec(a, x, izq, med-1)

x = a[med] \rightarrow i:= med

x > a[med] \rightarrow i:= binary_search_rec(a, x, med+1, der)

fi

fi

end fun

{Post: (i = 0 \Rightarrow x no está en a[izq,der]) \wedge (i \neq 0 \Rightarrow x = a[i])}

Búsqueda binaria

Función principal

```
fun binary_search (a: array[1..n] of T, x:T) ret i:nat  
    i:= binary_search_rec(a, x, 1, n)
```

```
end fun
```

{Post: $(i = 0 \Rightarrow x$ no está en $a) \wedge (i \neq 0 \Rightarrow x = a[i])$ }

- Sea $t(n)$ = número de comparaciones que hace en el peor caso cuando el arreglo tiene n celdas.

-

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

- $a = 1$, $b = 2$ y $k = 0$.
- $a = b^k$.
- conclusión $t(n) \in \mathcal{O}(n^k \log n) = \mathcal{O}(\log n)$.

Ejemplo

Calculemos el número de veces que el siguiente programa ejecuta la acción A:

```
proc p (in n: nat) {pre : n ≥ 0}  
  if n = 0 → skip  
    n = 1 → A  
    n > 1 → p(n-1)  
           p(n-2)  
  fi  
end proc
```

Sea $t(n)$ = número de veces que $p(n)$ ejecuta la acción A.

Contando las ejecuciones de la acción A

```
proc p (in n: nat) {pre : n ≥ 0}  
  if n = 0 → skip  
    n = 1 → A  
    n > 1 → p(n-1)  
           p(n-2)  
  fi  
end proc
```

Sea $t(n)$ = número de veces que $p(n)$ ejecuta la acción A.

$$t(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ t(n-1) + t(n-2) & \text{si } n > 1 \end{cases}$$

Es una recurrencia

- Es una recurrencia.
- ¿Es recurrencia divide y vencerás?
-

$$t(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ t(n-1) + t(n-2) & \text{si } n > 1 \end{cases}$$

- ¿Les resulta familiar esta recurrencia?

Recurrencia homogénea

- No es divide y vencerás porque las llamadas recursivas no son de la forma $t(n/b)$ sino de la forma $t(n - i)$.
- Si pasamos todos los términos de la forma $t(n - i)$ de la ecuación $t(n) = t(n - 1) + t(n - 2)$ a la izquierda, queda $t(n) - t(n - 1) - t(n - 2) = 0$,
- Por eso se la llama **recurrencia homogénea**.

Método de resolución

Paso 1: ecuación característica

- Llevar la recurrencia a una **ecuación característica** de la forma

$$a_k t_n + \dots + a_0 t_{n-k} = 0$$

- En el ejemplo, $t(n) = t(n-1) + t(n-2)$ puede llevarse a $t_n - t_{n-1} - t_{n-2} = 0$.
- Por eso se la llama homogénea.
- Entonces, $k = 2$, $a_k = a_2 = 1$, $a_1 = -1$ y $a_0 = -1$.

Método de resolución

Paso 2: polinomio característico

- Considerar el **polinomio característico asociado**
 $a_k x^k + \dots + a_0$,
- En el ejemplo el polinomio es $x^2 - x - 1$.

Método de resolución

Paso 3: raíces y multiplicidades

- determinar las raíces r_1, \dots, r_j del polinomio característico, de multiplicidad m_1, \dots, m_j respectivamente (se tiene $m_j \geq 1$ y $m_1 + \dots + m_j = k$),
- En el ejemplo, las raíces del polinomio son $r = \frac{1 \pm \sqrt{1+4}}{2} = \frac{1 \pm \sqrt{5}}{2}$.
- Entonces, $j = 2$, $r_1 = \frac{1+\sqrt{5}}{2}$, $r_2 = \frac{1-\sqrt{5}}{2}$, $m_1 = m_2 = 1$.

Método de resolución

Paso 4: forma general de la solución

- considerar la forma general de las soluciones de la ecuación característica:

$$\begin{aligned}
 t(n) &= c_1 r_1^n + c_2 n r_1^n + \dots + c_{m_1} n^{m_1-1} r_1^n + \\
 &+ c_{m_1+1} r_2^n + c_{m_1+2} n r_2^n + \dots + c_{m_1+m_2} n^{m_2-1} r_2^n + \\
 &\vdots \\
 &+ c_{m_1+\dots+m_{j-1}+1} r_j^n + c_{m_1+\dots+m_{j-1}+2} n r_j^n + \dots + c_k n^{m_j-1} r_j^n
 \end{aligned}$$

como $m_1 + \dots + m_j = k$, tenemos k incógnitas: c_1, \dots, c_k ,

- En el ejemplo, la forma general es

$$t(n) = c_1 r_1^n + c_2 r_2^n$$

Método de resolución

Paso 5: sistema de ecuaciones

- con las k **condiciones iniciales** $t_{n_0}, \dots, t_{n_0+k-1}$ (n_0 es usualmente 0 ó 1) plantear un sistema de k ecuaciones con k incógnitas:

$$\begin{aligned}t(n_0) &= t_{n_0} \\t(n_0 + 1) &= t_{n_0+1} \\&\vdots \\t(n_0 + k - 1) &= t_{n_0+k-1}\end{aligned}$$

- En el ejemplo, $n_0 = 0$ y el sistema es

$$\begin{aligned}c_1 + c_2 &= c_1 r_1^0 + c_2 r_2^0 = t(0) = t_0 = 0 \\c_1 r_1 + c_2 r_2 &= c_1 r_1^1 + c_2 r_2^1 = t(1) = t_1 = 1\end{aligned}$$

Método de resolución

Paso 6: cálculo de incógnitas

- obtener de este sistema los valores de las k incógnitas

$c_1, \dots, c_k,$

- En el ejemplo, de la primera ecuación, se obtiene

$c_1 = -c_2$, reemplazando en la segunda:

$$\begin{aligned} 1 &= -c_2 r_1 + c_2 r_2 \\ &= c_2 (r_2 - r_1) \\ &= c_2 \left(\frac{1-\sqrt{5}}{2} - \frac{1+\sqrt{5}}{2} \right) \\ &= c_2 \left(\frac{1-\sqrt{5}-1-\sqrt{5}}{2} \right) \\ &= c_2 \left(-\frac{2\sqrt{5}}{2} \right) \\ &= -c_2 \sqrt{5} \end{aligned}$$

Entonces $c_2 = -\frac{1}{\sqrt{5}}$ y $c_1 = \frac{1}{\sqrt{5}}$.

Método de resolución

Paso 7: solución final

- escribir la **solución final** de la forma $t_n = t'(n)$, donde $t'(n)$ se obtiene a partir de $t(n)$ reemplazando c_i y r_i por sus valores y simplificando la expresión final.
- En el ejemplo,

$$\begin{aligned}t_n &= t(n) \\ &= c_1 r_1^n + c_2 r_2^n \\ &= \frac{1}{\sqrt{5}} * \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} * \left(\frac{1-\sqrt{5}}{2}\right)^n\end{aligned}$$

Método de resolución

Paso 8: comprobación

- La **solución final** obtenida puede demostrarse por inducción. Más sencillo que eso es corroborar que $t_{n_0+k} = t'(n_0 + k)$, donde $n_0 + k$ es un **valor nuevo, no utilizado en el sistema de ecuaciones** anterior
- En el ejemplo,

$$\begin{aligned}t_2 &= t_1 + t_0 = 1 + 0 = 1 \\t'(2) &= \frac{1}{\sqrt{5}} * \left(\frac{1+\sqrt{5}}{2}\right)^2 - \frac{1}{\sqrt{5}} * \left(\frac{1-\sqrt{5}}{2}\right)^2 \\&= \frac{1}{4\sqrt{5}} * ((1 + \sqrt{5})^2 - (1 - \sqrt{5})^2) \\&= \frac{1}{4\sqrt{5}} * (2\sqrt{5} + 2\sqrt{5}) \\&= \frac{1}{4\sqrt{5}} * 4\sqrt{5} \\&= 1\end{aligned}$$

Método de resolución

Paso 9: orden

- Se concluye que $t'(n)$ es la solución de la recurrencia. Si el objetivo era calcular el **orden** ya se pueden utilizar las propiedades conocidas.
- En el ejemplo, $t'(n) = \frac{1}{\sqrt{5}} * \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} * \left(\frac{1-\sqrt{5}}{2}\right)^n$.
- Como las constantes multiplicativas no afectan, $t'(n) \in \mathcal{O}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$.
- Como $\lim_{n \rightarrow \infty} \frac{\left(\frac{1-\sqrt{5}}{2}\right)^n}{\left(\frac{1+\sqrt{5}}{2}\right)^n} = 0$,
 $\mathcal{O}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right) = \mathcal{O}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$.
- Por lo tanto $t'(n) \in \mathcal{O}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$.

Ejemplo

Calculemos el número de veces que el siguiente programa ejecuta la acción A:

```
proc p (in n: nat) {pre : n ≥ 0}  
  if n = 0 → skip  
    n > 0 → p(n-1)  
      p(n-1)  
      for i:= 1 to n do A od  
  fi  
end proc
```

Sea $t(n)$ = número de veces que $p(n)$ ejecuta la acción A.

Contando las ejecuciones de la acción A

```
proc p (in n: nat) {pre : n ≥ 0}  
  if n = 0 → skip  
  n > 0 → p(n-1)  
    p(n-1)  
    for i:= 1 to n do A od  
  fi  
end proc
```

Sea $t(n)$ = número de veces que $p(n)$ ejecuta la acción A.

$$t(n) = \begin{cases} 0 & \text{si } n = 0 \\ 2t(n-1) + n & \text{si } n > 0 \end{cases}$$

Es una recurrencia

- Es una recurrencia.
- ¿Es recurrencia divide y vencerás?
-

$$t(n) = \begin{cases} 0 & \text{si } n = 0 \\ 2t(n-1) + n & \text{si } n > 0 \end{cases}$$

- ¿Es una recurrencia homogénea?

Recurrencia no homogénea

- No es divide y vencerás porque las llamadas recursivas no son de la forma $t(n/b)$ sino de la forma $t(n - i)$.
- Si pasamos todos los términos de la forma $t(n - i)$ de la ecuación $t(n) = 2t(n - 1) + n$ a la izquierda, queda $t(n) - 2t(n - 1) = n$,
- Por eso no es homogénea.
- Se la llama **recurrencia no homogénea**.

Método de resolución

Paso 1: ecuación característica

- Llevar la recurrencia a una **ecuación característica** de la forma

$$a_k t_n + \dots + a_0 t_{n-k} = b^n p(n)$$

donde $p(n)$ es un polinomio no nulo de grado d .

- En el ejemplo, $t(n) = 2t(n-1) + n$ puede llevarse a $t_n - 2t_{n-1} = 1^n n$.
- Por eso se la llama no homogénea.
- Entonces, $k = 1$, $a_k = a_1 = 1$, $a_0 = -2$, $b = 1$ y $d = 1$.

Método de resolución

Paso 2: polinomio característico

- Considerar el **polinomio característico asociado**
 $(a_k x^k + \dots + a_0)(x - b)^{d+1}$,
- En el ejemplo el polinomio es $(x - 2)(x - 1)^2$

Método de resolución

Paso 3: raíces y multiplicidades

- determinar las raíces r_1, \dots, r_j del polinomio característico, de multiplicidad m_1, \dots, m_j respectivamente (se tiene $m_j \geq 1$ y $m_1 + \dots + m_j = k + d + 1$),
- En el ejemplo, las raíces del polinomio son $r_1 = 1$ y $r_2 = 2$ con multiplicidades $m_1 = 2$ y $m_2 = 1$. Y $j = 2$.

Método de resolución

Paso 4: forma general de la solución

- considerar la forma general de las soluciones de la ecuación característica:

$$\begin{aligned}
 t(n) &= c_1 r_1^n + c_2 n r_1^n + \dots + c_{m_1} n^{m_1-1} r_1^n + \\
 &+ c_{m_1+1} r_2^n + c_{m_1+2} n r_2^n + \dots + c_{m_1+m_2} n^{m_2-1} r_2^n + \\
 &\vdots \\
 &+ c_{m_1+\dots+m_{j-1}+1} r_j^n + \dots + c_{k+d+1} n^{m_j-1} r_j^n
 \end{aligned}$$

como $m_1 + \dots + m_j = k + d + 1$, tenemos $k + d + 1$ incógnitas: c_1, \dots, c_{k+d+1} ,

- En el ejemplo, la forma general es

$$\begin{aligned}
 t(n) &= c_1 r_1^n + c_2 n r_1^n + c_3 r_2^n \\
 &= c_1 + c_2 n + c_3 2^n
 \end{aligned}$$

Método de resolución

Paso 5: cálculo de más condiciones adicionales

- a partir de las k condiciones iniciales $t_{n_0}, \dots, t_{n_0+k-1}$ (n_0 es usualmente 0 ó 1), obtener usando la ecuación característica, los valores de $t_{n_0+k}, \dots, t_{n_0+k+d}$,
- En el ejemplo, $n_0 = 0$, $t_0 = 0$ y necesitamos t_{n_0+k} y t_{n_0+k+1} (o sea, t_1 y t_2):

$$\begin{aligned}t_1 &= 2t_0 + 1 \\ &= 1 \\ t_2 &= 2t_1 + 2 \\ &= 4\end{aligned}$$

Método de resolución

Paso 6: sistema de ecuaciones

- a partir de las $k + d + 1$ condiciones iniciales $t_{n_0}, \dots, t_{n_0+k+d}$ plantear un sistema de $k + d + 1$ ecuaciones con $k + d + 1$ incógnitas:

$$\begin{aligned}
 t(n_0) &= t_{n_0} \\
 t(n_0 + 1) &= t_{n_0+1} \\
 &\vdots \\
 t(n_0 + k + d) &= t_{n_0+k+d}
 \end{aligned}$$

- En el ejemplo, $n_0 = 0$ y el sistema es

$$\begin{aligned}
 c_1 + c_3 &= c_1 + c_2 \cdot 0 + c_3 \cdot 2^0 = t(0) = t_0 = 0 \\
 c_1 + c_2 + 2c_3 &= c_1 + c_2 + c_3 \cdot 2^1 = t(1) = t_1 = 1 \\
 c_1 + 2c_2 + 4c_3 &= c_1 + c_2 \cdot 2 + c_3 \cdot 2^2 = t(2) = t_2 = 4
 \end{aligned}$$

Método de resolución

Paso 7: cálculo de incógnitas

- obtener de este sistema los valores de las $k + d + 1$ incógnitas c_1, \dots, c_{k+d+1} ,
- En el ejemplo, de la primera ecuación, se obtiene $c_1 = -c_3$, reemplazando en la segunda:

$$\begin{aligned}1 &= -c_3 + c_2 + 2c_3 \\ &= c_2 + c_3\end{aligned}$$

Entonces $c_2 = 1 - c_3$, reemplazando en la tercera ecuación:

$$\begin{aligned}4 &= -c_3 + 2(1 - c_3) + 4c_3 \\ &= 2 + c_3\end{aligned}$$

Entonces $c_3 = 2$, $c_1 = -2$ y $c_2 = -1$.

Método de resolución

Paso 8: solución final

- escribir la **solución final** de la forma $t_n = t'(n)$, donde $t'(n)$ se obtiene a partir de $t(n)$ reemplazando c_i y r_i por sus valores y simplificando la expresión final.
- En el ejemplo,

$$\begin{aligned}t_n &= t(n) \\ &= c_1 + c_2 n + c_3 2^n \\ &= 2 * 2^n - n - 2\end{aligned}$$

Método de resolución

Paso 9: comprobación

- La **solución final** obtenida puede demostrarse por inducción. Más sencillo que eso es corroborar que $t_{n_0+k+d+1} = t'(n_0 + k + d + 1)$, donde $n_0 + k + d + 1$ es un **valor nuevo, no utilizado en el sistema de ecuaciones anterior**
- En el ejemplo,

$$\begin{aligned}t_3 &= 2t_2 + 3 = 2 * 4 + 3 = 11 \\t'(3) &= 2 * 2^3 - 3 - 2 \\&= 16 - 5 \\&= 11\end{aligned}$$

Método de resolución

Paso 10: orden

- Se concluye que $t'(n)$ es la solución de la recurrencia. Si el objetivo era calcular el **orden** ya se pueden utilizar las propiedades conocidas.
- En el ejemplo, $t'(n) = 2 * 2^n - n - 2 \in \mathcal{O}(2^n)$.