Algoritmos y Estructuras de Datos II

Árboles binarios de búsqueda

24 de abril de 2013

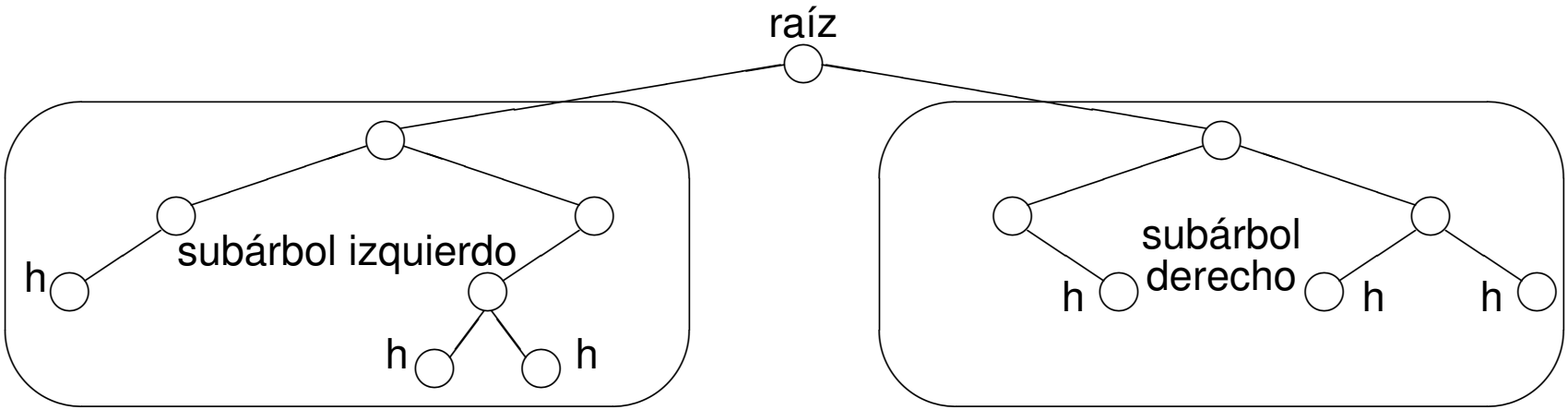
Clase de hoy

1. Repaso
   - Árboles binarios
   - Especificación
   - Terminología habitual
   - Implementación con punteros
   - Posiciones
2. Repaso para el parcial
   - Ordenación
   - Recurrencias

Repaso

- cómo vs. qué
- 3 partes
  1. análisis de algoritmos
     - algoritmos de ordenación
     - notación $\mathcal{O}$, Ω y Θ.
     - propiedades y jerarquía
     - recurrencias (D. y V., homogéneas y no homogéneas)
  2. tipos de datos
     - tipos concretos (arreglos, listas, tuplas, punteros)
     - tipos abstractos (TAD contador, TAD pila, TAD cola, TAD pcola)
     - implementaciones elementales
     - implementaciones utilizando listas enlazadas
     - árboles binarios
  3. técnicas de resolución de problemas

Intuición



- De cada circulo, cuelgan dos subárboles.
- En el caso de las hojas, los subárboles son vacíos.
- Todos los árboles pueden construirse con los constructores
  - $<>$, que construye un árbol vacío
  - $< \_, \_, \_ >$, que construye un árbol no vacío a partir de un elemento y dos subárboles

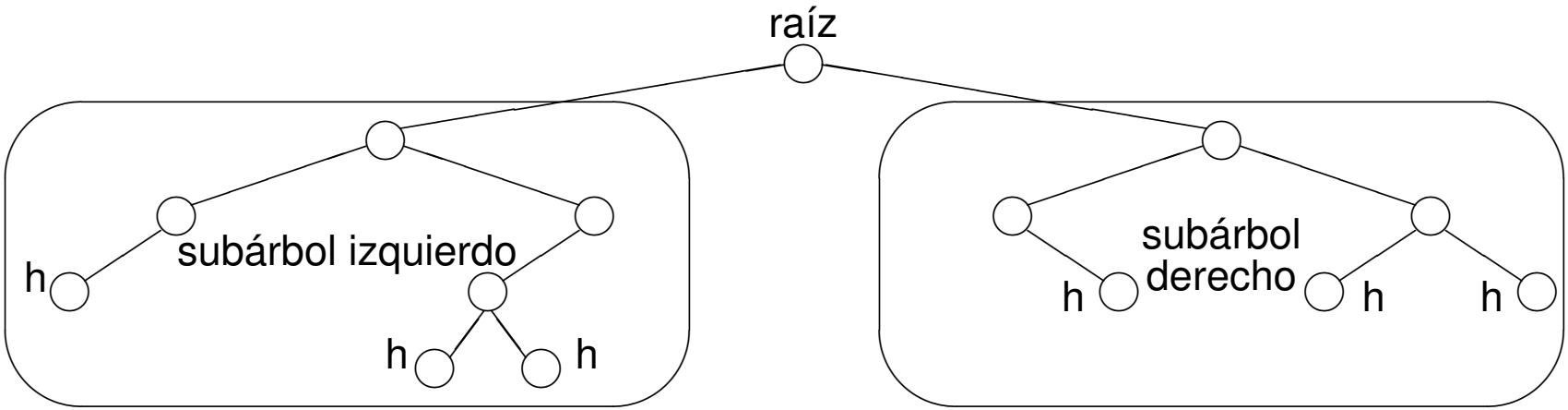
Especificación

TAD árbol_binario[elem]
constructores
$<>$: árbol_binario
$< \_, \_, \_ >$: árbol_binario $\times$ elem $\times$ árbol_binario $\rightarrow$ árbol_binario
operaciones
raíz : árbol_binario $\rightarrow$ elem {se aplica sólo a un árbol no vacío}
izquierdo : árbol_binario $\rightarrow$ árbol_binario {sólo a un árbol no vacío}
derecho : árbol_binario $\rightarrow$ árbol_binario {sólo a un árbol no vacío}
es_vacío : árbol_binario $\rightarrow$ booleano
ecuaciones
raíz($<$i,r,d$>$) = r
izquierdo($<$i,r,d$>$) = i
derecho($<$i,r,d$>$) = d
es_vacío($<>$) = verdadero
es_vacío($<$i,r,d$>$) = falso

Notación $<>$

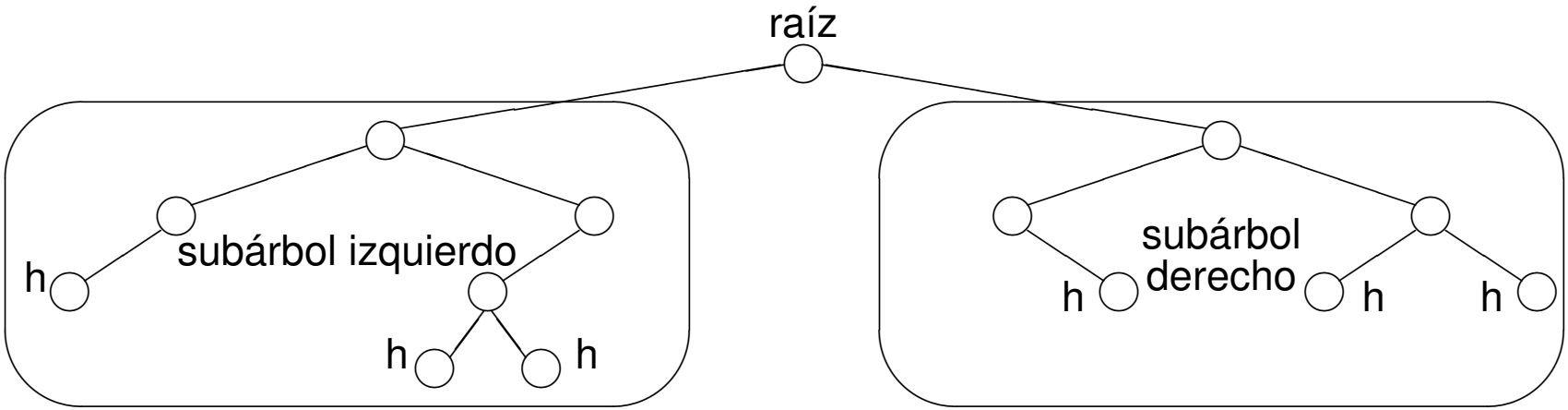
- Notar la sobrecarga de la notación $<>$:
- cuando no tiene ningún argumento, $<>$ es el árbol vacío,
- cuando tiene tres argumentos, $< i, r, d >$ es el árbol no vacío cuya raíz es r, subárbol izquierdo es i y subárbol derecho es d.
- Una hoja es un árbol de la forma $<<>, r, <>>$. Se la abrevia $< r >$.
- Conclusión: la notación $<>$ puede tener 0, 1 ó 3 argumentos.

Botánica y genealogía



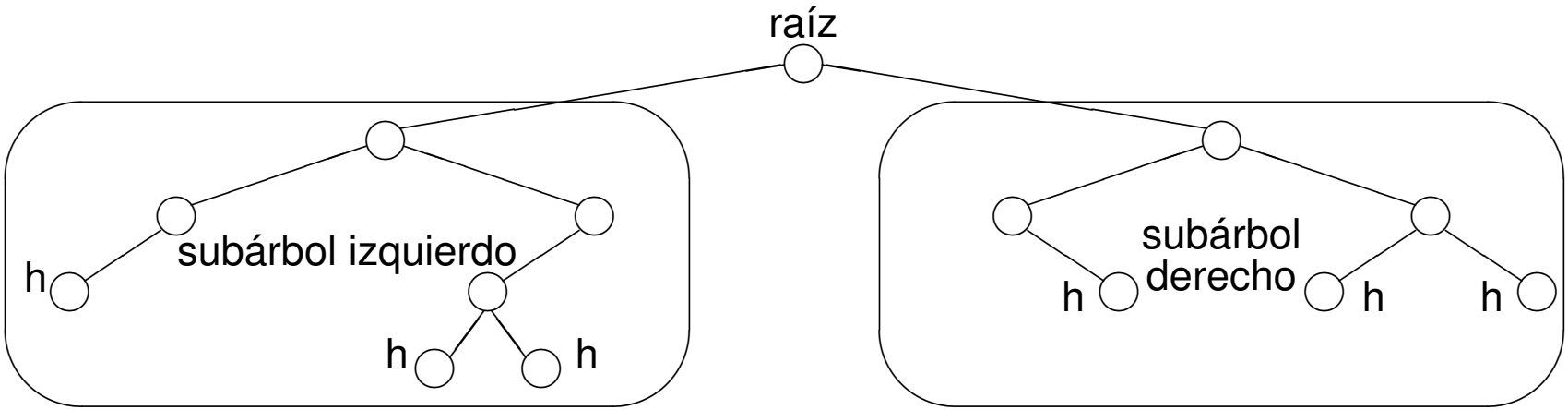
- Un **nodo** es un árbol no vacío.
- Tiene **raíz**, **subárbol izquierdo** y **subárbol derecho**.
- A los subárboles se los llama también **hijos** (izquierdo y derecho).
- Y al nodo se le dice **padre** de sus hijos.
- Una **hoja** es un nodo con los dos hijos vacíos.

Más genealogía



- Dos **hermanos** son hijo izquierdo y derecho del mismo padre.
- Un **camino** es una secuencia $A_1, \ldots, A_n$ donde cada árbol es hijo del anterior.
- A_i es **ancestro** de A_j sii $i \leq j$,
- en ese caso A_j es **descendiente** o **subárbol** de A_i
- un camino se puede identificar con un recorrido descendente del árbol.

Altura, profundidad, nivel



- Decimos que siempre hay un camino de longitud 0 de un árbol a sí mismo.
- La **altura** de un árbol es la longitud del camino que va desde él hasta la hoja más lejana.
- La **profundidad** de un subárbol es la longitud del camino que va desde el árbol hasta dicho subárbol.
- Se llama **nivel** al conjunto de los subárboles de igual profundidad.

Implementación con punteros

```
type node = tuple
                lft: pointer to node
                value: elem
                rgt: pointer to node
            end
type bintree = pointer to node

fun empty() ret t:bintree
    t:= null
end
{Post: t ~<> }
```

Implementación con punteros

{Pre: l $\sim$ L $\wedge$ e $\sim$ E $\wedge$ r $\sim$ R}
fun node(l:bintree,e:elem,r:bintree) **ret** t:bintree
    alloc(t)
    t$\rightarrow$lft:= l
    t$\rightarrow$value:= e
    t$\rightarrow$rgt:= r
{Post: t $\sim$ <L,E,R>} **end**

{Pre: t $\sim$ T $\wedge\neg$ is_empty(t)}
fun root(t:bintree) **ret** e:elem
    e:= t$\rightarrow$value
end
{Post: e $\sim$ raíz(T)}

Implementación con punteros

{Pre: t $\sim$ T $\wedge\neg$ is_empty(t)}
fun left(t:bintree) **ret** l:bintree
    l:= t$\rightarrow$lft
end
{Post: l $\sim$ izquierdo(T)}

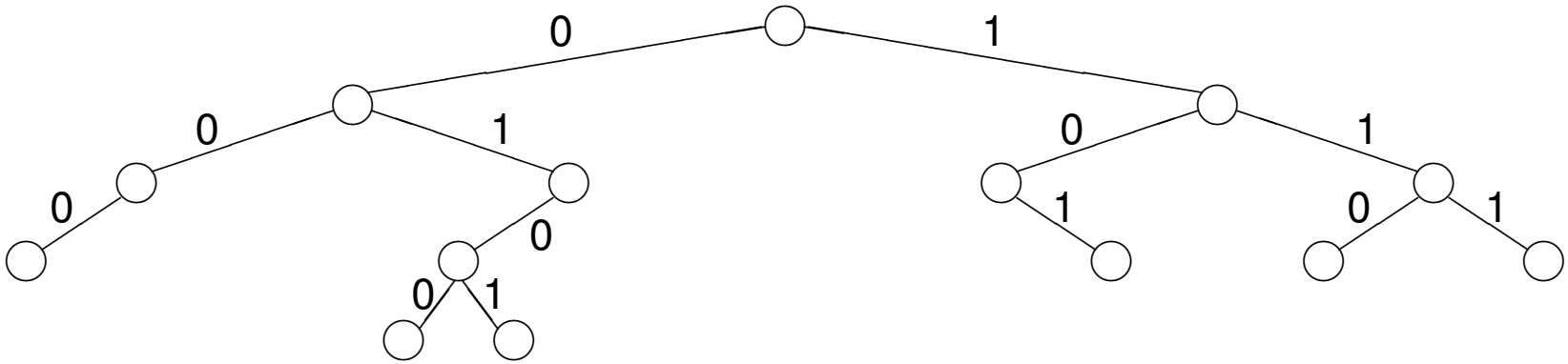
{Pre: t $\sim$ T $\wedge\neg$ is_empty(t)}
fun right(t:bintree) **ret** r:bintree
    r:= t$\rightarrow$rgt
end
{Post: r $\sim$ derecho(T)}

Implementación con punteros

{Pre: t $\sim$ T}
fun is_empty(t:bintree) **ret** b:**bool**
    b:= (t = **null**)
end
{Post: b $\sim$ es_vacío(T)}

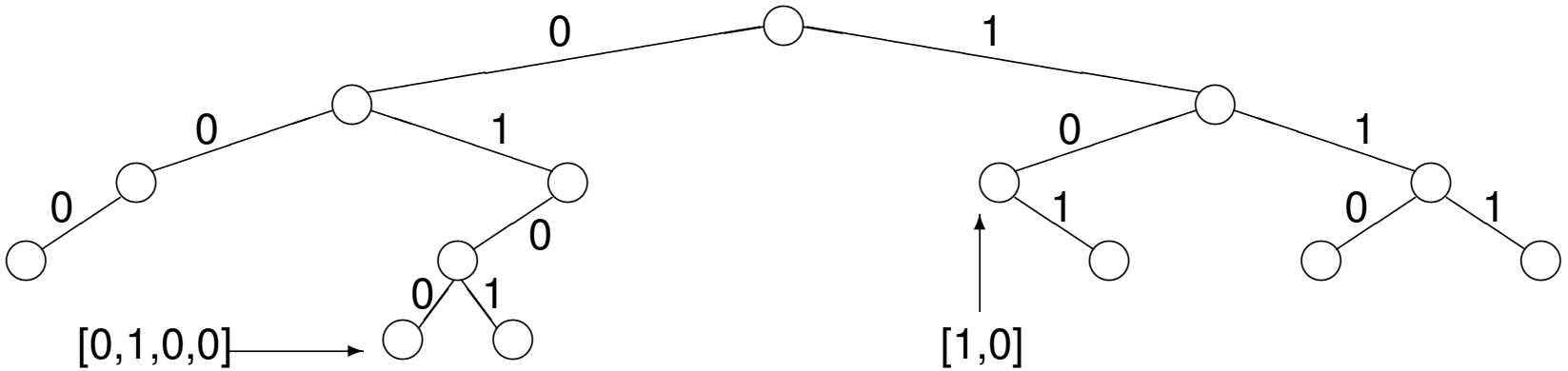
proc destroy(**in**/**out** t:bintree)
    if $\neg$ is_empty(t) **then** destroy(left(t))
                destroy(right(t))
                free(t)
                t:= **null**
    fi
end

Indicaciones



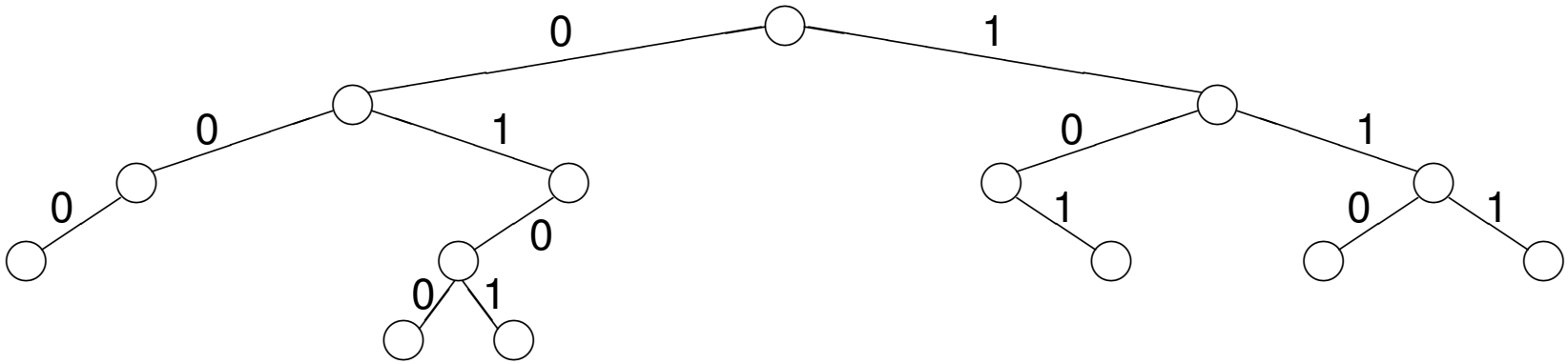
- A cada arista que conecta un padre con su hijo se la rotula 0 si es con el hijo izquierdo y 1 si es el derecho,
- Este 0 ó 1 puede entenderse como dando **indicaciones**
- 0 es ir a la izquierda
- 1 es ir a la derecha

Posiciones



- Una lista de 0's y 1's sirve para desplazarse desde la raíz hacia las hojas.
- Cada subárbol queda señalado por una lista de 0's y 1's.
- Estas listas de 0's y 1's marcan **posiciones** dentro del árbol.
- Definimos $pos = [\{0, 1\}]$.
- Es el conjunto de todas las posiciones.

Selección de subárbol

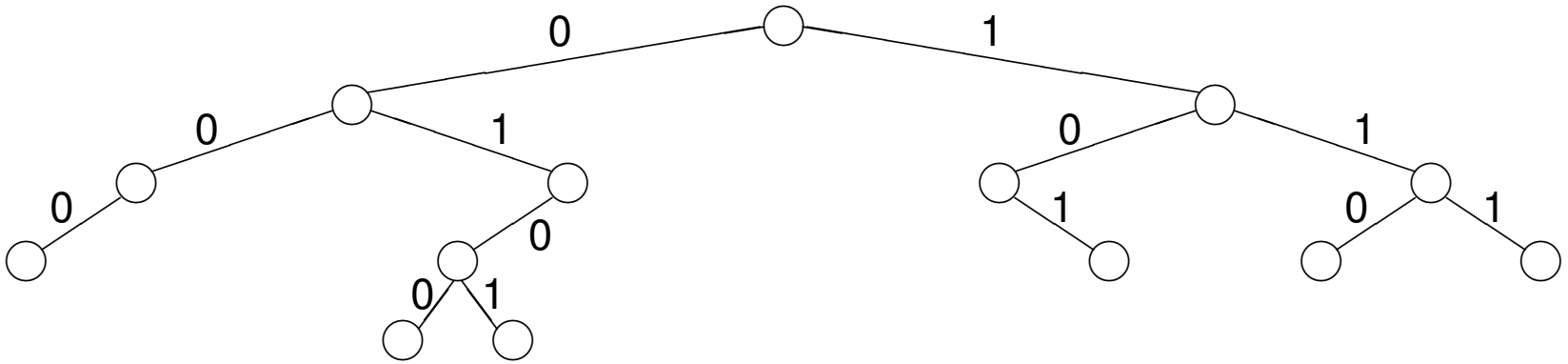


Dado un árbol t y una posición $p \in pos$, $t \downarrow p$ es el subárbol de t que se encuentra en la posición p:

$<>\downarrow p = <>$
$< i, e, d >\downarrow [] = < i, e, d >$
$< i, e, d >\downarrow (0 \triangleright p) = i \downarrow p$
$< i, e, d >\downarrow (1 \triangleright p) = d \downarrow p$

Se define $pos(t) = \{p \in pos \mid t \downarrow p \neq <>\}$. Es el conjunto de las posiciones del árbol binario t.

Selección de elemento



Dado un árbol t y una posición $p \in pos(t)$, $t.p$ es el elemento de t que se encuentra en la posición p:

$< i, e, d > .[] = e$
$< i, e, d > .(0 \triangleright p) = i.p$
$< i, e, d > .(1 \triangleright p) = d.p$

o equivalentemente $t.p = raiz(t \downarrow p)$.

¿Este algoritmo ordena?

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j:= i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

¿Este algoritmo ordena?

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= i
            for j:= i+1 to n do
                if a[j] < a[minp] then minp:= j fi
            od
            swap(a,i,minp)
        od
end proc
```

¿Este algoritmo ordena?

```
proc selection_sort (in/out a: array[1..n] of T)
        var maxp: nat
        for i:= n downto 2 do
            maxp:= 1
            for j:= 2 to i do
                if a[j] > a[maxp] then maxp:= j fi
            od
            swap(a,i,maxp)
        od
end proc
```

¿Este algoritmo ordena?

```
proc cocktail_sort (in/out a: array[1..n] of T)
    var minp, maxp: nat
    for i:= 1 to n ÷ 2 do
        minp:= i
        maxp:= i
        for j:= i+1 to n-i+1 do
            if a[j] < a[minp] then minp:= j
            else if a[j] > a[maxp] then maxp:= j fi
            fi
        od
        swap(a,i,minp)
        swap(a,n-i+1,maxp)
    od
end proc
```

¿Este algoritmo ordena?

```
proc bubble_sort (in/out a: array[1..n] of T)
        for i:= 1 to n-1 do
            for j:= n-1 downto i do
                if a[j] > a[j+1] then swap(a,j,j+1) fi
            od
        od
end proc
```

¿Este algoritmo ordena?

```
proc bubble_sort (in/out a: array[1..n] of T)
    var done: bool
    done:= false
    i:= 1
    do i ≤ n-1 ∧ ¬ done →
        done:= true
        for j:= n-1 downto i do
            if a[j] > a[j+1] then swap(a,j,j+1)
                                  done:= false
            fi
        od
        i:= i+1
    od
end proc
```

¿Este algoritmo ordena?

```
proc bubble_sort (in/out a: array[1..n] of T)
    var done: nat
    i:= 1
    do i ≤ n-1 →
        done:= n
        for j:= n-1 downto i do
            if a[j] > a[j+1] then swap(a,j,j+1)
                                  done:= j
            fi
        od
        i:= done+1
    od
end proc
```

¿Este algoritmo ordena?

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

¿Este algoritmo ordena?

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            j:= i
            do j > 1 ∧ a[j] < a[j − 1] → swap(a,j,j-1)
                                          j:= j-1
            od
        od
end proc
```

¿Este algoritmo ordena?

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= n-1 downto 1 do
            j:= i
            do j < n ∧ a[j] > a[j + 1] → swap(a,j,j+1)
                                         j:= j+1
            od
        od
end proc
```

¿Este algoritmo ordena?

```
proc merge_sort (in/out a: array[1..n] of T)
        merge_sort_rec(a,1,n)
end proc

proc merge_sort_rec (in/out a: array[1..n] of T, in izq,der: nat)
        var med: nat
        if der > izq → med:= (der+izq) ÷ 2
                       merge_sort_rec(a,izq,med)
                       merge_sort_rec(a,med+1,der)
                       merge(a,izq,med,der)
        fi
end proc
```

¿Este algoritmo ordena?

```
proc merge_sort (in/out a: array[1..n] of T)
    var gap: nat
    gap:= 1
    do gap < n →
        izq:= 1;    med:= gap;    der:= 2*gap
        do der ≤ n →
            merge(a,izq,med,der)
            izq:= der + 1;    med:= med + 2*gap;    der:= der + 2*gap
        od
        if med < n then merge(izq,med,n)
        gap:= 2*gap
    od
end proc
```

¿Este algoritmo ordena?

Acá quería traer variantes del quicksort, no tuve tiempo:

- variar la manera de elegir el pivot,
- variar la manera de usar el pivot (por ejemplo, como en el baile húngaro)
- pivoteando en tres bloques: menores, iguales y mayores al pivot.

Ejemplo

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

$\{pre : n \geq 0\}$

Contando las ejecuciones de la acción A

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

$$t(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ 3t(n-1) + 4t(n-2) & \text{si } n > 1 \end{cases}$$

Es una recurrencia

- Es una recurrencia.
- ¿Divide y vencerás? ¿Homogénea? ¿No homogénea?
- Hay que mirar la ecuación interesante.
- Es homogénea.

Método de resolución

Paso 1: ecuación característica

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

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

,

- En el ejemplo, $t(n) = 3t(n-1) + 4t(n-2)$ puede llevarse a $t_n - 3t_{n-1} - 4t_{n-2} = 0$.
- Entonces, $k = 2$, $a_k = a_2 = 1$, $a_1 = -3$ y $a_0 = -4$.

Método de resolución

Paso 2: polinomio característico

- Considerar el **polinomio característico asociado** $a_k x^k + \ldots + a_0$,
- En el ejemplo el polinomio es $x^2 - 3x - 4$.

Método de resolución

Paso 3: raíces y multiplicidades

- determinar las raíces $r_1, \ldots, r_j$ del polinomio característico, de multiplicidad $m_1, \ldots, m_j$ respectivamente (se tiene $m_i \geq 1$ y $m_1 + \ldots + m_j = k$),
- En el ejemplo, las raíces del polinomio son $r = \frac{3 \pm \sqrt{9+16}}{2} = \frac{3 \pm \sqrt{25}}{2}$.
- Entonces, $j = 2$, $r_1 = \frac{3+5}{2} = 4$, $r_2 = \frac{3-5}{2} = -1$, $m_1 = m_2 = 1$.

Método de resolución

Paso 4: forma general de la solución

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

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

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

- En el ejemplo, la forma general es

$$t(n) = c_1 r_1^n + c_2 r_2^n = c_1 4^n + c_2 (-1)^n$$

Método de resolución

Paso 5: sistema de ecuaciones

- con las k **condiciones iniciales** $t_{n_0}, \ldots, t_{n_0+k-1}$ (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

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

Método de resolución

Paso 6: cálculo de incógnitas

- obtener de este sistema los valores de las k incógnitas $c_1, \ldots, c_k$,
- En el ejemplo, de la primera ecuación, se obtiene $c_1 = -c_2$, reemplazando en la segunda:

$$1 = -4c_2 - c_2 = -5c_2$$

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

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,

$$\begin{aligned} t_n &= t(n) \\ &= c_1 4^n + c_2(-1)^n \\ &= \tfrac{1}{5}4^n - \tfrac{1}{5}(-1)^n \end{aligned}$$

Método de resolución

Paso 8: comprobación

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

$$\begin{aligned} t_2 &= 3t_1 + 4t_0 = 3 * 1 + 4 * 0 = 3 \\ t'(2) &= \frac{1}{5} * 4^2 - \frac{1}{5} * (-1)^2 \\ &= \frac{16}{5} - \frac{1}{5} \\ &= \frac{15}{5} \\ &= 3 \end{aligned}$$

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, $t'(n) = \frac{1}{5}4^n - \frac{1}{5}(-1)^n \in \Theta(4^n)$.

Ejemplo

$\{pre : n \geq 0\}$

proc p (**in** n: **nat**)
    if n $=$ 0 $\rightarrow$ **skip**
      n $>$ 0 $\rightarrow$ p(n-1)
                selection_sort_to(a,n)
    fi
end proc

Contando las comparaciones

Sea $t(n)$ = número de comparaciones que realiza $p(n)$.

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

Es una recurrencia

- Es una recurrencia.
- ¿Divide y vencerás? ¿Homogénea? ¿No homogénea?
- Hay que mirar la ecuación interesante.
- Es no homogénea.

Método de resolución

Paso 1: ecuación característica

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

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

  donde $p(n)$ es un polinomio no nulo de grado d.
- En el ejemplo, $t(n) = t(n-1) + \frac{n^2}{2} - \frac{n}{2}$ puede llevarse a $t_n - t_{n-1} = 1^n(\frac{n^2}{2} - \frac{n}{2})$.
- Entonces, $k = 1$, $a_k = a_1 = 1$, $a_0 = -1$, $b = 1$ y $d = 2$.

Método de resolución

Paso 2: polinomio característico

- Considerar el **polinomio característico asociado** $(a_k x^k + \ldots + a_0)(x - b)^{d+1}$,
- En el ejemplo el polinomio es $(x - 1)(x - 1)^3$,
- O sea, $(x - 1)^4$,

Método de resolución

Paso 3: raíces y multiplicidades

- determinar las raíces $r_1, \ldots, r_j$ del polinomio característico, de multiplicidad $m_1, \ldots, m_j$ respectivamente (se tiene $m_i \geq 1$ y $m_1 + \ldots + m_j = k + d + 1$),
- En el ejemplo, hay una sola raíz del polinomio $r_1 = 1$ con multiplicidad $m_1 = 4$. Y $j = 1$.

Método de resolución

Paso 4: forma general de la solución

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

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

  como $m_1 + \ldots + m_j = k + d + 1$, tenemos $k + d + 1$ incógnitas: $c_1, \ldots, c_{k+d+1}$,
- En el ejemplo, la forma general es

$$\begin{aligned} t(n) &= c_1 r_1^n + c_2 n r_1^n + c_3 n^2 r_1^n + c_4 n^3 r_1^n \\ &= c_1 + c_2 n + c_3 n^2 + c_4 n^3 \end{aligned}$$

Método de resolución

Paso 5: cálculo de más condiciones adicionales

- a partir de las k condiciones iniciales $t_{n_0}, \ldots, t_{n_0+k-1}$ (n_0 es usualmente 0 ó 1), obtener usando la ecuación característica, los valores de $t_{n_0+k}, \ldots, t_{n_0+k+d}$,
- En el ejemplo, $n_0 = 0$, $t_0 = 0$ y necesitamos t_{n_0+k}, t_{n_0+k+1} y t_{n_0+k+2} (o sea, t_1, t_2 y t_3):

$$\begin{aligned} t_1 &= t_0 + \frac{1^2}{2} - \frac{1}{2} \\ &= 0 \\ t_2 &= t_1 + \frac{2^2}{2} - \frac{2}{2} \\ &= 1 \\ t_3 &= t_2 + \frac{3^2}{2} - \frac{3}{2} \\ &= 4 \end{aligned}$$

Método de resolución

Paso 6: sistema de ecuaciones

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

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

Método de resolución

Paso 6: sistema de ecuaciones

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

$$\begin{aligned}
c_1 &= c_1 + c_2 0 + c_3 0 + c_4 0 = t(0) = t_0 = 0 \\
c_2 + c_3 + c_4 &= c_1 + c_2 1 + c_3 1 + c_4 1 = t(1) = t_1 = 0 \\
2c_2 + 4c_3 + 8c_4 &= c_1 + c_2 2^1 + c_3 2^2 + c_4 2^3 = t(2) = t_2 = 1 \\
3c_2 + 9c_3 + 27c_4 &= c_1 + c_2 3 + c_3 3^2 + c_4 3^3 = t(3) = t_3 = 4
\end{aligned}$$

Método de resolución

Paso 7: cálculo de incógnitas

- obtener de este sistema los valores de las $k + d + 1$ incógnitas $c_1, \ldots, c_{k+d+1}$,

Método de resolución

Paso 7: cálculo de incógnitas

- De la primera ecuación, se obtiene $c_1 = 0$, y de la segunda se obtiene $c_2 = -c_3 - c_4$, reemplazando en la tercera:

$$\begin{aligned} 1 &= -2c_3 - 2c_4 + 4c_3 + 8c_4 \\ &= 2c_3 + 6c_4 \end{aligned}$$

Entonces $c_3 = \frac{1-6c_4}{2}$, reemplazando en la cuarta ecuación:

$$\begin{aligned} 4 &= -3c_3 - 3c_4 + 9c_3 + 27c_4 \\ &= 6c_3 + 24c_4 \\ &= 6\frac{1-6c_4}{2} + 24c_4 \\ &= 3 - 18c_4 + 24c_4 \\ &= 3 + 6c_4 \end{aligned}$$

Entonces $c_4 = \frac{1}{6}$, $c_3 = 0$, $c_2 = -\frac{1}{6}$ y $c_1 = 0$.

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,

$$\begin{aligned} t_n &= t(n) \\ &= c_1 + c_2 n + c_3 n^2 + c_4 n^3 \\ &= \tfrac{1}{6} n^3 - \tfrac{1}{6} n \end{aligned}$$

Método de resolución

Paso 9: comprobación

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

$$\begin{aligned} t_4 &= t_3 + \frac{4^2}{2} - \frac{4}{2} = 4 + 8 - 2 = 10 \\ t'(4) &= \frac{1}{6}4^3 - \frac{1}{6}4 \\ &= \frac{64}{6} - \frac{4}{6} \\ &= \frac{60}{6} \\ &= 10 \end{aligned}$$

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, $t'(n) = \frac{1}{6}n^3 - \frac{1}{6}n \in \Theta(n^3)$.