Algoritmos y Estructuras de Datos II

Resolución del 1er parcial

28 de abril de 2014

Clase de hoy

Ejercicio 1
proc A

```
proc A(in/out a: array[1..n,1..n] of nat, in b,c: nat)
        var f: nat
        f:= a[b,c]
        a[b,c]:= a[c,b]
        a[c,b]:= f
end proc
```

1. ¿qué hace?
2. ¿cómo?
3. ¿orden?
4. ¿casos?

Ejercicio 1

proc A, respuestas

1. ¿qué hace? intercambia los valores de las celdas a[b,c] y a[c,b]
2. ¿cómo? aloja temporariamente el valor de la primera en una variable auxiliar
3. ¿orden? constante
4. ¿casos? siempre constante

Ejercicio 1
proc B

```
proc B(in/out a: array[1..n,1..n] of nat)
        for i:= 1 to n do
            for j:= i+1 to n do
                A(a,i,j)
            od
        od
end proc
```

1. ¿qué hace?
2. ¿cómo?
3. ¿orden?
4. ¿casos?

Ejercicio 1

proc B, respuestas

1. ¿qué hace? traspone la matriz
2. ¿cómo? recorre todas las celdas del triángulo que está sobre la diagonal, intercambiando cada una de ellas por la celda correspondiente en el triángulo que está debajo de la diagonal.
3. ¿orden? $n-1+n-2+\ldots+1=\frac{n(n-1)}{2}\in\Theta(n^2)$, cuadrático
4. ¿casos? siempre cuadrático

Ejercicio 1
proc C

```
proc C(in/out a: array[1..n,1..n] of nat)
        for i:= 1 to n do
            for j:= i to n do
                A(a,i,j)
            od
        od
end proc
```

1. ¿qué hace?
2. ¿cómo?
3. ¿orden?
4. ¿casos?

Ejercicio 1

proc C, respuestas

1. ¿qué hace? lo mismo que el **proc** B: traspone la matriz
2. ¿cómo? recorre además la diagonal, innecesariamente ya que intercambia cada celda de la diagonal con sí misma.
3. ¿orden? $n + n - 1 + \ldots + 1 = \frac{n(n+1)}{2} \in \Theta(n^2)$, cuadrático
4. ¿casos? siempre cuadrático

Ejercicio 1
proc D

```
proc D(in/out a: array[1..n,1..n] of nat)
        for i:= 1 to n do
            for j:= 1 to n do
                A(a,i,j)
            od
        od
end proc
```

1. ¿qué hace?
2. ¿cómo?
3. ¿orden?
4. ¿casos?

Ejercicio 1

proc D, respuestas

1. ¿qué hace? nada: devuelve la misma matriz que recibe
2. ¿cómo? traspone dos veces cada celda, y por eso la matriz resultante es la misma que recibe.
3. ¿orden? $n * n \in \Theta(n^2)$, cuadrático
4. ¿casos? siempre cuadrático

Ejercicio 1
proc E

```
proc E (in/out a: array[1..n] of nat)
        var b,c: nat
        for i:= n-1 downto 1 do
            b:= i
            do b < n ∧ a[b] > a[b+1] ⟶ c:= a[b+1]
                                       a[b+1]:= a[b]
                                       a[b]:= c
                                       b:= b+1
            od
        od
end proc
```

1. ¿qué hace?
2. ¿cómo?
3. ¿orden?
4. ¿casos?

Ejercicio 1

proc E, respuestas

1. ¿qué hace? ordena ascendentemente el arreglo
2. ¿cómo? ordenación por inserción de atrás para adelante: inserta el penúltimo, luego el ante-penúltimo, luego el anterior, etc.
3. ¿orden? entre lineal (si nunca entra al cuerpo del ciclo **do**) y cuadrático (si ejecuta el cuerpo del ciclo siempre $n - b$ veces)
4. ¿casos? lineal cuando el arreglo ya está ordenado, cuadrático cuando el arreglo está ordenado en forma descendente

Ejercicio 1
proc F

```
fun F (a: array[1..n] of nat) ret b: array[1..3] of nat
    b[1]:= a[1]
    b[2]:= 1
    b[3]:= 1
    for i:= 2 to n do
        b[1]:= b[1] + a[i]
        if a[i] < a[b[2]] then b[2]:= i
        else if a[i] > a[b[3]] then b[3]:= i fi
        fi
    od
end fun
```

1. ¿qué hace?
2. ¿cómo?
3. ¿orden?
4. ¿casos?

Ejercicio 1

proc F, respuestas

1. ¿qué hace? devuelve en la primera celda de b, la suma de los valores de a, en la segunda celda de b, la posición del menor elemento de a, y en la tercera celda de b, la posición del mayor elemento de a.
2. ¿cómo? recorre el arreglo a de izquierda a derecha, sumando los valores encontrados con la suma parcial que se encuentra en b[1], comparando con el menor y el mayor provisorios, cuyas posiciones se encuentran en b[2] y b[3].
3. ¿orden? lineal, recorre una vez el arreglo a.
4. ¿casos? siempre lineal

Ejercicio 2a

Planteá la recurrencia que indica la cantidad de asignaciones a *m*

```
fun f (n: nat) ret m: nat
    if n ≤ 2 then
        m := n
    else
        m := 3*f(n/2) + n
    fi
end
```

Ejercicio 2a

Planteá la recurrencia que indica la cantidad de asignaciones a *m*, solución

```
fun f (n: nat) ret m: nat
    if n ≤ 2 then
        m := n
    else
        m := 3*f(n/2) + n
    fi
end
```

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

Ejercicio 2b y 2c

Resolvé la recurrencia

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

  Es divide y vencerás con $a = 4$, $b = 2$ y $k = 2$. Como $a = b^k$, $t(n) \in \Theta(n^2 \log n)$.

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

  Es divide y vencerás con $a = 1$, $b = 2$ y $k = 0$. Como $a = b^k$, $t(n) \in \Theta(\log n)$.

Ejercicio 3

Ordená según sus $\mathcal{O}$

1. $\log_2(3^{n!})$
2. $\log_2(n * (3^{n!})^2)$
3. $n * \log_2(3^{(n-1)!})$
4. $(\log_2(3^n))^n$

Ejercicio 3

Ordená según sus $\mathcal{O}$, solución

1. $\log_2(3^{n!}) = n! \log_2 3 \in \Theta(n!)$
2. $\log_2(n * (3^{n!})^2) = \log_2(n) + 2 * n! * \log_2 3 \in \Theta(n!)$
3. $n * \log_2(3^{(n-1)!}) = n * (n-1)! * \log_2 3 \in \Theta(n!)$
4. $(\log_2(3^n))^n = (n * \log_2 3)^n = n^n * (\log_2 3)^n$

Conclusión

$\mathcal{O}(\log_2(3^{n!})) = \mathcal{O}(\log_2(n * (3^{n!})^2)) = \mathcal{O}(n * \log_2(3^{(n-1)!}) = \mathcal{O}(n!)$

$\subset$

$\mathcal{O}((\log_2(3^n))^n)$

Ejercicio 4
Especificar el TAD natural

TAD natural
constructores
cero : natural
sucesor : natural $\rightarrow$ natural
operaciones
es_cero : ...
predecesor : ... { pre: argumento $\neq$ cero }
suma : ...
multiplicación : ...
exponenciación : ... { pre: sus dos argumentos no pueden ser c
ecuaciones
...
...
...

Ejercicio 4

Especificar el TAD natural, primera parte

TAD natural
constructores
cero : natural
sucesor : natural $\rightarrow$ natural
operaciones
es_cero : natural $\rightarrow$ booleanos
predecesor : natural $\rightarrow$ natural
suma : natural $\times$ natural $\rightarrow$ natural
multiplicación : natural $\times$ natural $\rightarrow$ natural
exponenciación : natural $\times$ natural $\rightarrow$ natural
ecuaciones
...
...
...

Ejercicio 4

Especificar el TAD natural, segunda parte

TAD natural

. . .

ecuaciones

es_cero(cero) = verdadero
es_cero(sucesor(n)) = falso
predecesor(sucesor(n)) = n
suma(cero,m) = m
suma(sucesor(n),m) = sucesor(suma(n,m))
multiplicación(cero,m) = cero
multiplicación(sucesor(n),m) = suma(m,multiplicación(n,m))
exponenciación(n,cero) = sucesor(cero)
exponenciación(n,sucesor(m))
= multiplicación(n,exponenciación(n,m))

Ejercicio 5a

En que se diferencia el TAD colable del TAD cola

En dos cosas

- El TAD colable permite observar el último elemento de la cola.
- El TAD colable permite agregar un elemento al principio de la cola.

Ejercicio 5b

Implementar el TAD colable con listas enlazadas circulares, función last

$\{$Pre: p $\sim$ Q $\wedge \neg$is_empty(p)$\}$
fun first(p:queue) **ret** e:elem
    e:= p$\rightarrow$next$\rightarrow$value
end fun
$\{$Post: e $\sim$ primero(Q)$\}$

$\{$Pre: p $\sim$ Q $\wedge \neg$is_empty(p)$\}$
fun last(p:queue) **ret** e:elem
    e:= p$\rightarrow$value
end fun
$\{$Post: e $\sim$ último(Q)$\}$

Ejercicio 5b

Implementar el TAD colable con listas enlazadas circulares, procedimiento jump

{Pre: p $\sim$ Q $\wedge$ e $\sim$ E}
proc enqueue(**in**/**out** p:queue,**in** e:elem)
    var q: **pointer to** node
    alloc(q)
    q$\rightarrow$value:= e
    if p $=$ **null** $\rightarrow$ p:= q
                    q$\rightarrow$next:= q
      p $\neq$ **null** $\rightarrow$ q$\rightarrow$next:= p$\rightarrow$next
                    p$\rightarrow$next:= q
                    p:= q
    fi
end proc
{Post: p $\sim$ encolar(Q,E)}

Ejercicio 5b

Implementar el TAD colable con listas enlazadas circulares, procedimiento jump

{Pre: p $\sim$ Q $\wedge$ e $\sim$ E}
proc jump(**in/out** p:queue,**in** e:elem)
    var q: **pointer to** node
    alloc(q)
    q$\rightarrow$value:= e
    if p $=$ **null** $\rightarrow$ p:= q
                    q$\rightarrow$next:= q
      p $\neq$ **null** $\rightarrow$ q$\rightarrow$next:= p$\rightarrow$next
                    p$\rightarrow$next:= q
    fi
end proc
{Post: p $\sim$ colar(E,Q)}