COMPLEJIDAD ALGORITMICA

Jaime Lorenzo Sánchez

30 de enero de 2022

1.- $x \leftarrow x + 1$

O(t(n)) = O(c) = O(1) ->Complejidad lineal

2.- **para** i **de** 1 **hasta** n **hacer**
$x \leftarrow x + 1$
finpara

O(t(n)) = O($\sum_{i=1}^{n} c$) = O(c*n) = O(n) ->Complejidad lineal

3.- **para** i **de** 1 **hasta** n **hacer**
para j **de** 1 **hasta** n **hacer**
$suma \leftarrow suma + a(i, j)$
finpara
finpara

O(t(n)) = O($\sum_{i=1}^{n} \sum_{j=1}^{n} c$) = O(n*n*c) = O($n^2$) ->Complejidad cuadrática

4.- **si** $(n \ mod \ 2 = 0)$ **entonces**
para i **de** 1 **hasta** n **hacer**
$a(i) \leftarrow -a(i)$
finpara
finsi

O(t(n)) = O(t(condición)+t(consecuente)) = O(max(t(condición),t(consecuente))) = O(max(c,$\sum_{i=1}^{n} c$)) = O(max(c,c*)) = O(c*n) = O(n) ->Complejidad lineal

5.- $i \leftarrow 1$
mientras $(i \leq n)$ **hacer**
$x \leftarrow x * i$
$i \leftarrow i + 2$
finmientras

O(t(n)) = O(t(asignacion)+t(mientras)) = O(max(t(asignacion),t(mientras))) = O(max(c,$\sum_{i=1}^{n*div2} 2*c$)) = O(max(c,2c*n*div2)) = O(2c*n*div2) = O(n) ->Complejidad lineal

6.- $x \leftarrow 1$
mientras $(x \leq n)$ **hacer**
$x \leftarrow 2 * x$
finmientras

El cálculo de las iteraciones del bucle es equivalente al valor de t en las siguientes instrucciones:

x <- 1 t <- 1 mientras (x $\leq$n) hacer

x <- 2*x

t <- t+1

fin mientras

Si n=32 ->t = 6. Se cumple que $t = log_2 n$.

Por tanto, t(mientras) = 2c*log n

O(t(n)) = O(t(asignacion)+t(mientras)) = O(max(2c,2c*log n)) = O(2c*log n) = O(log n) ->Complejidad logarítmica

7.- Evaluación de la complejidad de un algoritmo iterativo que calcula el n-ésimo término de la serie de Fibonacci.

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

```
Algoritmo Fibonacci(n; −; fib)
 inicio
    si (n ≤ 1) entonces fib ← 1
        si no
                     mayor ← 1
                     menor ← 1
                     para i de 2 hasta n hacer
                          auxiliar ← menor
                          menor ← mayor
                          mayor ← mayor + auxiliar
                     finpara
                     fib ← mayor
    finsi
 fin
```

t(Fibbonacci) = t(condición) + t(consecuente) + t(alternativa)

=t(condición)+max(t(consecuente,t(alternativa)))

=c+max(c,2c+$\sum_{i=2}^{n-1}$3c)

= c + 2c+(n-1)*3c= 3c+(n-1)*3c = 3c*(1+n-1) = 3c*n

O(t(n)) = O(3c*n) = O(n) ->Complejidad lineal

8.- Evaluación de la complejidad de un algoritmo que calcula un número combinatorio.

Se desea calcular $\binom{n}{m} = \frac{n!}{m!(n-m)!}$

Primero se mostrará una función que calcula el factorial de un número.

```
Función Factorial(n; - ; -):entero
inicio
  fact ← 1
  si (n > 1) entonces
      para i de 2 hasta n hacer
          fact ← fact * i
      finpara
  finsi
  devolver fact
fin
```

t(Factorial) = t(asignacion) + t(condición)+t(consecuente)

$= c + c + \sum_{i=2}^{n-1} c = 2c + c*(n-1) = c * (2+n-1) = c*(n+1)$

O(Factorial) = O(c*(n+1)) = O(n) ->Complejidad lineal

El algoritmo parametrizado que calcula el número combinatorio es:

```
Algoritmo Combinatorio(n, m; −; c)
inicio
  si (n < m)   entonces c ← 0
       si no     c ← Factorial(n)/(Factorial(m) * Factorial(n − m))
  finsi
fin
```

t(Combinatorio) = t(condición)+t(consecuente)+t(alternativa)

= c + max(t(consecuente),t(alternativa))

= c + max(c,t(Factorial(n))+t(Factorial(m))+t(Factorial(n-m)))

= c + max(c,c*(n+1)+c*(m+1)+c*(n-m+1))

=c+max(c,c*(n+n+m-m+1+1+1)) = c+max(c,c*(2n+3))

= c + c*(2n+3) = c*(1+2n+3) = c*(2n+4)

O(Combinatorio) = O(n) ->Complejidad lineal

9.- Multiplicación de matrices.

```
Algoritmo Producto(A, B, n, p, m; −; C)
inicio
  para i de 1 hasta n hacer
        para j de 1 hasta m hacer
            C(i, j) ← 0
            para k de 1 hasta p hacer
                        C(i, j) ← A(i, k) ∗ B(k, j) + C(i, j)
            finpara
        finpara
  finpara
fin
```

t(Producto) = $\sum_{i=1}^{n} *(\sum_{j=1}^{m}(c + \sum_{i=1}^{p} c))$ = n*m*(c+c*p) = n*m*c*(1+p)

Si consideramos n=max(n,m,p) ->O(Producto) = O(n*n*n) = O(n^3) ->Complejidad cúbica

Ejercicio 10. Resolver mediante comparaciones y desplazamientos en los casos mejor,peor y medio

```
Algoritmo OrdenacionInsercion(n; v; )
inicio
  para i de 2 hasta n hacer
     v[0] ← v[i]
     j ← i − 1
     mientras v[j] > v[0]
          v[j + 1] ← v[j]
          j ← j − 1
     finmientras
     v[j + 1] ← v[0]
  finpara
fin
```

Consideramos $c_{max}, c_{min}, c_{medio}$ las comparaciones en los casos peor,mejor y medio respectivamente:

c_{max} = i; c_{min}=1; $c_{medio} = \frac{c_{max}+c_{min}}{2} = \frac{i+1}{2}$

Sean $C_{max}, C_{min}, C_{medio}$ las comparaciones totales en los casos peor, mejor y medio respectivamente:

$C_{max} = \sum_{i=2}^{n} c_max = \sum_{i=2}^{n} i = \frac{(primero+ultimo)*numeroTerminos}{2}$

$= \frac{(2+n)*(n-1)}{2} = \frac{2n-2+n^2-n}{2} = \frac{n^2+n-2}{2}$

$C_{min} = \sum_{i=2}^{n} 1 = n - 1$

$C_medio = \frac{C_{max}+C_{min}}{2} = \frac{n^2+n-2+2n-2}{4} = \frac{n^2+3n-4}{4}$

Sean $d_{max}, d_{min}, d_{medio}$ los desplazamientos en los casos peor, mejor y medio respectivamente:

$d_{max} = i - 1$; $d_{min} = 0$; $d_{medio} = \frac{i-1}{2}$

Sean $D_{max}, D_{min}, D_{medio}$ los desplazamientos totales en los casos peor, mejor y medio:

$D_{max} = \sum_{i=2}^{n}(i-1) = \frac{n^2-n}{2}$

$D_{min} = \sum_{i=2}^{n} 0 = 0$

$D_{medio} = \frac{n^2-n}{4}$

En el caso medio, el más probable, las comparaciones y los desplazamientos totales tienen una complejidad O(n^2), complejidad cúbica.

Los casos mejor y peor se da cuando el vector está ordenado inicialmente o en sentido inverso, respectivamente.

Ejercicio 11. Resolver mediante comparaciones y desplazamientos en los casos mejor, peor y medio

```
Algoritmo OrdenacionSeleccion(n; v; )
inicio
   para i de 1 hasta n − 1 hacer
     posicionMenor ← i
     para j de i+1 hasta n hacer
        si v[j] < v[posicionMenor] entonces
           posicionMenor ← j
        finsi
     finpara
     aux ← v[i]
     v[i] ← v[posicionMenor]
     v[posicionMenor] ← aux
   finpara
fin
```

Sean $c_{max}, c_{min}, c_{med}$ las comparaciones en los casos peor, mejor y medio en la pasada i.

Sean $d_{max}, d_{min}, d_{med}$ los desplazamientos en los casos peor, mejor y medio en la pasada i.

$c_{max} = c_{min} = c_{med} = n - i;$

$d_{max} = d_{min} = d_{med} = 1$

Sean $C_{max}, C_{min} y C_{med}$ las comparaciones totales en los casos peor, mejor y medio respectivamente:

$C_{max} = \sum_{i=1}^{n-i} c_{max} = \sum_{i=1}^{n-1} (n - i)$
$= \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i$
$= n * (n - 1) - \frac{(1+n-1)*(n-1)}{2} = \frac{n^2-n}{2}$

$C_{min} = C_{max} = C_{med} = \frac{n^2-n}{2}$

Sean $D_{max}, D_{min} y D_{med}$ los desplazamientos totales en los casos peor, mejor y medio respectivamente:

$D_{max} = D_{min} = D_{min} = \sum_{i=1}^{n-1} 1 = n - 1$

En las comparaciones la complejidad algorítmica es O(n^2) y en los desplazamientos la complejidad algorítmica es O(n). Por tanto, son independientes del orden inicial del vector.

Ejercicio 12. Resolver mediante comparaciones y desplazamientos en los casos mejor, peor y medio

```
Algoritmo OrdenacionBurbuja(n; v; )
inicio
   para i de 1 hasta n − 1 hacer
      para j de 1 hasta i hacer
         si v[j] > v[j + 1] entonces
            aux ← v[j]
            v[j] ← v[j + 1]
            v[j + 1] ← aux
         finsi
      finpara
   finpara
fin
```

Sean $c_{max}, c_{min}, c_{med}$ las comparaciones en los casos peor, mejor y medio cuando se inserta el elemento i:

$c_{max} = c_{min} = c_{med} = n - i$

Sean $d_{max}, d_{min}, d_{med}$ los desplazamientos en los casos peor, mejor y medio cuando se inserta el elemento i:

$d_{max} = n - i, d_{min} = 0, d_{med} = \frac{n-i}{2}$

Sean $C_{max}, C_{min}, C_{med}$ las comparaciones totales en los casos peor, mejor y medio:

$C_{min} = C_{med} = C_{max} = \sum_{i=1}^{n-1} c_{max} = \sum_{i=1}^{n-1} (n - i)$
$= \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = n * (n - 1) - \frac{(1+n-1)*(n-1)}{2} = \frac{n^2-n}{2}$

Sean $D_{max}, D_{min}, D_{med}$ los desplazamientos totales en los casos peor, mejor y medio:

$D_{max} = C_{max} = \frac{n^2-n}{2}$

$D_{min} = 0$

$D_{med} = \frac{n^2-n}{4}$

En el caso medio, que es más probable el mejor, el número de comparaciones y desplazamientos es de orden $O(n^2)$. Los casos mejor y peor se dan cuando el vector está ordenado inicialmente o de forma inversa, respectivamente.

Ejercicio 13. Calculo de la complejidad de un factorial usando expansión de recurrencias:

```
Función Factorial(n; - ; -):entero
 inicio
   si ((n = 0) o (n = 1))
          entonces devolver 1
          si no     devolver n * Factorial(n - 1)
     finsi
 fin
```

La versión iterativa de esta función posee complejidad lineal. Parece razonable que esta versión recursiva posea la misma complejidad.

El tiempo $t(n)$ se calcula usando la siguiente ecuación de recurrencia:

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

Aplicando expansión de recurrencias, tenemos lo siguiente:

$t(n) = t(n-1) + 3 =$

$(t(n-2) + 3) + 3 =$

$t(n-2) + 6$

$= (t(n-3) + 6) + 3 = t(n-3) + 9$

...

$= t(n-k) + 3 * k$

Para k=n, tenemos que: $t(n) = t(0) + 3n = 1 + 3n$

Por tanto, la complejidad del algoritmo es: $O(t(n)) = O(1 + 3n) = O(n)$ ->Complejidad lineal

Ejercicio 14. Calculo de la complejidad algorítmica usando expansión de recurrencias

Función $Recursiva1(n; -; -) : entero$
inicio
si $(n \leq 1)$ **entonces** **devolver** 5
si no **devolver** $Recursiva1(n-1) + Recursiva1(n-1)$
finsi
fin

Planteando las ecuaciones de recurrencia:

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

Aplicando expansión de recurrencias, tenemos que:

$t(n) = 2t(n-1) + 4$

$= 2(2t(n-2) + 4) + 4 = 4t(n-2) + 12 = 4t(n-2) + 3 * 4$

$= 4(4t(n-3) + 3 * 4) + 3 * 4 = 16t(n-3) + 5 * (3 * 4)$

$= ... = 2^k t(n-k) + (2^k - 1) * 4$

Para k=n, tenemos que: $t(n) = 2^n * t(0) + (2^n - 1) * 4 = 2^n + (2^n - 1) * 4 = 5 * (2^n) - 4$

Por tanto, la complejidad del algoritmo es: O(t(n)) = O(5∗(2^n)−4) = O(2^n) ->Complejidad exponencial

Ejercicio 15. Calculo de la complejidad algorítmica usando expansión de recurrencias

Función $Recursiva2(n; -; -) : entero$
inicio
si $(n \leq 1)$
entonces **devolver** 1
si no **devolver** $2 * Recursiva2(ndiv2)$
finsi
fin

Se tiene que

$$t(n) = \begin{cases} 1 & si\ n \leq 1 \\ t(n/2) + 3 & si\ n > 1 \end{cases}$$

Aplicando expansión de recurrencias, tenemos que:

$t(n) = t(\frac{n}{2}) + 3$

$= (t(\frac{n}{4}) + 3) + 3 = t(\frac{n}{2^2}) + 3 * 2$

$= (t(\frac{n}{2^3}) + 3 * 2) + 3 * 2 = t(\frac{n}{2^3}) + 3 * 3$

$= ...$

$= t(n) = \frac{n}{2^k} + 3k$

Cuando $\frac{n}{2^k} = 1$, tenemos que:

$\frac{n}{2^k} = 1- > n = 2^k- > k * log2 = logn- > k = \frac{logn}{log2} = log_2 n$

Por tanto, $t(n) = t(1) + 3 * log_2 n = 1 + 3 * log_2 n$

Luego, la complejidad del algoritmo es: $O(t(n)) = O(1+3*log_2 n) = O(log_2 n)$ ->Complejidad logarítmica

Ejercicio 16. Calculo de la complejidad algorítmica usando expansión de recurrencias

```
Algoritmo Recursiva3(n; x, r; -)
 inicio
   si (n = 0)    entonces r ← x
      si no
           para i de 1 hasta n hacer
                     x ← 2 * x
                     r ← r + 1
           finpara
           Recursivo3(n - 1, x, r)
   finsi
 fin
```

La ecuación de recurrencias que se plantea es:

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

Aplicando expansión de recurrencias, tenemos que:

$t(n) = 4n + 2 + t(n-1)$

$= 4n + 2 + (4(n-1) + 2 + t(n-2)) = 4(n + (n-1)) + 4 + t(n-2)$

$= 4(n + (n-1) + 4 + (4(n-2) + 2 + t(n-3))) = 4(n + (n-1) + (n-2)) + 6 + t(n-3)$

$= ...$

$= 4(n + (n-1) + (n-2) + ... + (n-k+1)) + 2k + t(n-k)$

Para k=n se tiene que t(0) = 2. Por tanto:

$t(n) = 4(n+(n-1)+(n-2)+...+1))+2n+t(0) = 4(n+(n-1)+(n-2)+...+1))+2n+2$
$= 4*(\frac{1+n}{2}*n)+2n+2 = 2*(n+n^2)+2n+2 = 2*(n^2)+4n+2$

La complejidad del algoritmo es O(n^2) ->Complejidad cuadrática

Ejercicio 17. Calculo de la complejidad algorítmica usando expansión de recurrencias

$$t(n) = \begin{cases} c & si\ n = 1 \\ 2 \times t(n/2) + bn + a & si\ n > 1\ y\ n = 2^k \end{cases}$$

Expandiéndola

$t(n) = 2t(n/2) + a + bn$

$= 2(2t(n/2^2) + a + bn/2) + a + bn = 2^2t(n/2^2) + a + 2a + 2bn$

$= 2^2(2t(n/2^3) + bn/2^2 + a) + a + 2a + 2bn = 2^3t(n/2^3) + a + 2a + 2^2a + 3bn$

...

$= 2^kt(n/2^k) + a(1 + 2 + 2^2 + ... + 2^{k-1}) + kbn = 2^kt(n/2^k) + a(2^k - 1) + kbn$

asumiendo que $n = 2^k$

$= nt(1) + a(n-1) + bnlog_2n = cn + a(n-1) + bnlog_2n$

$t(n) = cn + a(n-1) + bnlog_2n$

Luego el orden de complejidad es

$$O(t(n)) = O(cn + a(n-1) + bnlog_2n) = O(nlog_2n).$$

Ejercicio 18. Algoritmo de Fibonacci usando acotación

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

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

Se cumple que $g(n) \leq t(n) \leq f(n)$

Expandiendo $g(n)$

$$g(n) = 2g(n-2) + 4 = 2(2g(n-4) + 4) + 4 = 2^2 g(n-4) + 4 * (2+1)$$
$$= 2^2(2g(n-6) + 4) + 4 * (2+1) = 2^3 g(n-6) + 4 * (2^2 + 2 + 1)$$
$$...$$
$$= 2^k g(n-2k) + 4 * (2^{k-1} + 2^{k-2} + ... + 1) = 2^k g(n-2k) + 4 * (2^k - 1)$$

Si $n - 2k = 0$ entonces $g(n-2k) = g(0) = 0$

Luego $k = n/2$ y

$$g(n) = 2^{n/2} g(0) + 4 * (2^{n/2} - 1) = 4 * 2^{n/2} - 4$$
$$= 4 * 2^{n/2} - 4$$

y su orden de complejidad es

$O(g(n)) = O(4 * 2^{n/2} - 4) = O(2^{n/2}) = O(1{,}4142^n)$, orden exponencial.

Expandiendo $f(n)$

$$f(n) = 2f(n-1) + 4 = 2(2f(n-2) + 4) + 4 = 2^2 f(n-2) + 4 * (2+1)$$
$$= 2^2(2f(n-3) + 4) + 4 * (2+1) = 2^3 f(n-3) + 4 * (2^2 + 2 + 1)$$
$$...$$
$$= 2^k f(n-k) + 4 * (2^{k-1} + 2^{k-2} + ... + 1) =$$
$$= 2^k f(n-k) + 4 * (2^k - 1)$$

Si $n - k = 0$, entonces $f(n-k) = f(0) = 0$

Luego $k = n$ y

$$f(n) = 2^n f(0) + 4 * (2^n - 1)$$
$$= 4 * 2^n - 4$$

y su orden de complejidad es

$O(f(n)) = O(4 * 2^n - 4) = O(2^n)$, orden exponencial.

Debido a la acotación, $O(g(n)) \subset O(t(n)) \subset O(f(n))$, luego $O(t(n) \approx O(2^n)$ y la complejidad de Fibonacci resulta ser exponencial, que es mucho peor que la versión iterativa que posee una complejidad lineal.

Ejercicio 19. Algoritmo de Fibonacci usando el método de ecuación característica

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

La ecuación de recurrencia será: $t(n-1) + t(n-2)$, de modo que se cumple la ecuación característica: $x^2 - x - 1 = 0$, cuyas soluciones serán: $x1 = \frac{1+\sqrt{5}}{2}, x2 = \frac{1-\sqrt{5}}{2}$

La solución general será de la forma: $t(n) = c1 * x1^n + x2 * x2^n$

Cuando n=0 ->t(0) = 0 ->t(n) = c1 + c2 = 0

Si n=1 ->t(1) = 1 ->$c1 * x1 + c2 * x2 = c1 * \frac{1+\sqrt{5}}{2} + c2 * \frac{1-\sqrt{5}}{2} = 1$

Resolviendo el sistema, tenemos que $c1 = \frac{1}{\sqrt{5}}, c2 = -\frac{1}{\sqrt{5}}$

Por tanto, tenemos que: t(n) = $\frac{1}{\sqrt{5}} * (\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}} * (\frac{1-\sqrt{5}}{2})^n$