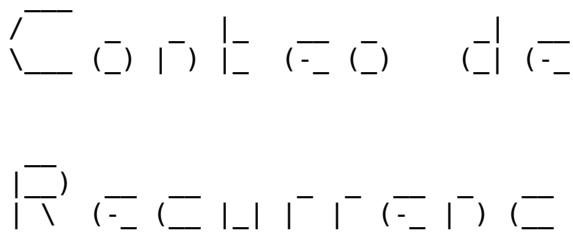
"Análisis de Algoritmos"
José Helo Guzmán



Contenido

▬▬▬ 1. Introducción.

Presentamos en esta sección algunos ejemplos de pseudocódigo con su respectivo conteo por recurrencias. Esta técnica de conteo es especialmente útil cuando se tienen algoritmos o programas recursivos.

▬▬▬ 2. Construcción de Relaciones de Recurrencia.

Supongamos que tenemos el siguiente algoritmo, que recibe un valor “n” entero como entrada.

```
test(n):=block
(
   if (n > 0) then
   (
      print(n),
      test(n-1)
   )
);
```

Veamos que ocurre cuando se invoca:

```
test(5)
5...test(4)
5...4...test(3)
5...4...3...test(2)
5...4...3...2...test(1)
5...4...3...2...1...test(0)
5...4...3...2...1...false
```

Podemos suponer que print(n) toma 1 unidad de tiempo.
Por lo tanto la relación de T(n) estará dada por:

$T(0) = 1$

$T(n) = 1 + T(n-1)$

Que es equivalente a:

$T(0) = 1$

$T(n) = T(n-1) + 1$

Al resolver esta relación tenemos:

$T(n) = n + 1$

Por lo tanto tenemos que su tiempo de ejecución es:

$O(n)$

●● Debemos tomar en cuenta el “if”?

Observe que para la construcción de esta relación se ha ignorado el valor del “if” para la relación de recurrencia. Si se toma en cuenta se tendría:

T(0) = 1

T(n) = 1+ T(n-1) + 1

T(n) = T(n-1) + 2

Como 2 es un valor constante, se sustituir por 1. Además el resultado de la relación de recurrencia nos dará el mismo valor de la BigOh. Resolvamos esta nueva relación de recurrencia:

T(0) = 1

T(n) = n + 1

Por lo tanto es:

O(n)

>>> En Maxima

```
load(solve_rec);
solve_rec(
t[n]=t[n-1]+1,
t[n],
t[0]=1
);
```

▬▬▬ 3. Ejemplo.

Supongamos que tenemos el siguiente algoritmo:

```
test(n):=block
(
   if (n > 0) then
   (
      for i:1 thru n do
      (
         print(i)
      ),
      test(n-1)
   )
);
```

Es más sencillo ver qué realiza esta función de la siguiente manera.

```
test(n):=block
(
```

```
   if (n > 0) then
   (
      for i:1 thru n do
      (
         print(i)
      ),
      print("---"),
      test(n-1)
   )
);
```

Veamo el tiempo de la función:

```
test(n):=block
(
   if (n > 0) then --------> 1
   (
      for i:1 thru n do ---> n
      (
         print(i) --> 1
      ),
      print("---"), -------> 1
      test(n-1) -----------> T(n-1)
   )
);
```

Por lo tanto el tiempo estará dado por:

$T(0) = 1$

$T(n) = T(n-1) + n + 2$

Y tenemos :

$$T(n) = \left(\frac{n*(n+5)}{2} \right) + 1$$

$$T(n) = \left(\frac{n^2 + 5*n}{2} \right) + 1$$

$$T(n) = \left(\frac{n^2}{2} \right) + \left(\frac{5*n}{2} \right) + 1$$

Y por lo tanto es

$O(n^2)$

>>> En Maxima:

```
load(solve_rec);
solve_rec(
t[n]=t[n-1]+n+2,
t[n],
t[0]=1
);
expand(t[n]=(n*(n+5))/2+1);
```

▬▬▬ 4. Ejemplo.

Se tiene el siguiente código:

```
test(n):=block
(
   if (n > 0) then
   (
      for i:1 thru n step i*2 do
      (
         print(i)
      ),
      print("---"),
      test(n-1)
   )
);
```

Se tiene el tiempo dado por:

```
test(n):=block
(
   if (n > 0) then   ----------------> 1
   (
      for i:1 thru n step i*2 do ----> log(n)
      (
         print(i)
      ),
      print("---"),  ----------------> 1
      test(n-1) ---------------------> T(n-1)
   )
);
```

Por lo tanto:

$T(0) = 1$

$T(n) = T(n-1) + \log(n) + 2$

Y tenemos la solución dada por:

$T(n) = \log(n!) + 2n + 1$

Por lo tanto:

O(log(n!))

Podemos resolver esta relación de recurrencia analíticamente.

Para ello la simplificamos a:

T(0) = 1

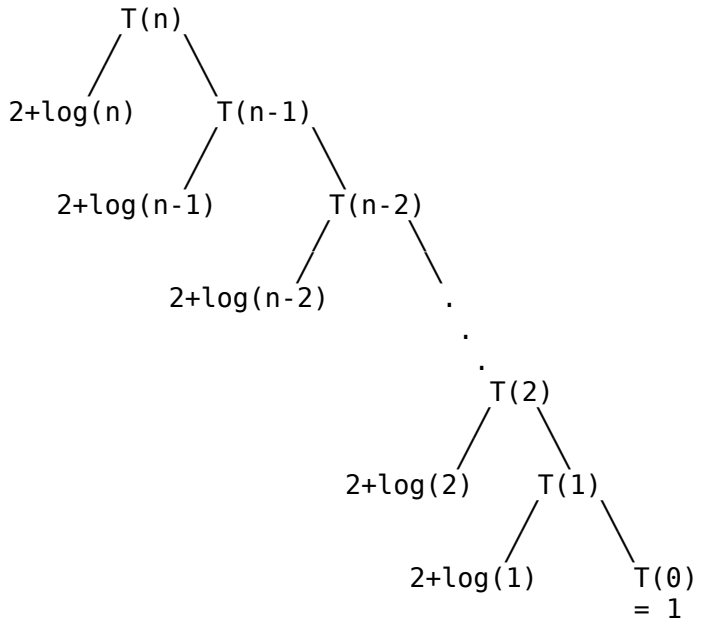
T(n) = T(n-1) + log(n) + 2

Pasamos a:

T(0) = 1

T(n) = 2 + log(n) + T(n-1)

Desarrollemos el árbol de la realción:



Que es equivalente a:

T(n) = 2+log(n) + 2+log(n-1) + 2+log(n-2) + ... + 1

T(n) = log(n) + log(n-1) + log(n-2) + ... + log(2) + log(1) + 2*n + 1

T(n) = log(n*(n-1)*(n-2)* ...*2*1) + 2*n + 1

T(n) = log(n!) + 2*n + 1

Cuyo valor es

O(log(n!))

Este resultado no suele presentarse de esta forma, usualmente se presenta de otra forma:

log(n!)

= log(n * (n-1) * (n-2) * ... * 1)

≤ log(n * n * n * ... * n)

= log(n^n)

= n * log(n)

Por lo tanto se dice que

O(log(n!)) = O(n*log(n))

>>> En Wxmaxima

```
load(solve_rec);

log2(x):=log(x)/log2;

solve_rec
(
t[n]=t[n-1] + log(n) + 2,
t[n],
t[0]=1
);
```

>>> En WolframAlpha

https://www.wolframalpha.com/examples/mathematics/discrete-mathematics/recurrences

https://www.wolframalpha.com/input?i=a%5Bn%5D+%3D+a%5Bn-1%5D+%2B+Log2%5Bn%5D%2B2%2C+a%5B0%5D%3D1

t[n] = t[n-1] + log2[n]+ 2, t[0]=1

▬▬▬▬ 5. Un Alto en el camino.

Hasta este momento hemos visto las siguientes relaciones:

T(n) = T(n-1) + 1 entonces O(n)

T(n) = T(n-1) + n entonces O(n^2)

T(n) = T(n-1) + n^2 entonces O(n^3)

$T(n) = T(n-1) + n^3$ entonces $O(n^4)$

$T(n) = T(n-1) + \log(n)$ entonces $O(n*\log(n))$

En general se tiene que:

$T(n) = T(n-1) + f(n)$ es $O(n*f(n))$

Observemos que en estas relaciones, siempre se multiplica el segundo término por "n". Este resultado se puede generalizar. Por ejemplo si tenemos la relación:

$T(n) = T(n-1) + n^6$ entonces será $O(n^7)$

Otra generalización que se puede realizar, es en la cantidad que se disminuye de "n". Por ejemplo:

$T(n) = T(n-2) + 1$

En lugar de disminuir en una unidad por iteración, disminuye en dos:

$T(n) = T(n-1) + 1$ entonces $O(n)$

$T(n) = T(n-2) + 1$ entonces $O(n)$

$T(n) = T(n-3) + 1$ entonces $O(n)$

Por lo tanto:

$T(n) = T(n-100) + 1$ es $O(n)$

$T(n) = T(n-100) + n$ es $O(n^2)$

En general:

$T(n) = T(n-k) + f(n)$ es $O(n*f(n))$

▬▬▬ 6. Seguimos con Otras Relaciones de Recurrencia.

Se tiene el siguiente algoritmo:

```
test(n):=block
(
   if (n > 0) then
  (
      print(n),
      test(n-1),
      test(n-1)
```

```
  )
);
```

Para analizar este código se tiene que:

```
test(n):=block
(
   if (n > 0) then ---> 1
  (
      print(n),  -----> 1
      test(n-1), -----> T(n-1)
      test(n-1)  -----> T(n-1)
  )
);
```

Entonces tenemos que:

$T(0) = 1$

$T(n) = 2*T(n-1) + 2$

Al resolverla se obtiene:

$T(n) = 3*2^n - 2$

Y tenemos

$O(2^n)$

>>> En Maxima:

```
load(solve_rec);

solve_rec(
t[n]=2*t[n-1]+2,
t[n],
t[0]=1
);
```

▬▬▬ 7. Master Theorem for Decreasing Functions.

Hasta ahora tenemos los siguientes resultados:

$T(n) = T(n-1) + 1 \quad = O(n)$

$T(n) = T(n-1) + n \quad = O(n^2)$

$T(n) = T(n-1) + n^2 \quad = O(n^3)$

$T(n) = T(n-1) + n^3 \quad = O(n^4)$

$T(n) = T(n-1) + \log(n) = O(n*\log(n))$

$T(n) = T(n-1) + f(n) = O(n*f(n))$

$T(n) = T(n-b) + f(n) = O(n*f(n))$

$T(n) = 2*T(n-1) + 1 = O(2^n)$

$T(n) = 3*T(n-1) + 1 = O(3^n)$

$T(n) = 4*T(n-1) + 1 = O(4^n)$

$T(n) = a*T(n-1) + 1 = O(a^n)$

$T(n) = 2*T(n-1) + n = O(2^n)$

$T(n) = 3*T(n-1) + n = O(3^n)$

$T(n) = 4*T(n-1) + n = O(4^n)$

$T(n) = a*T(n-1) + n = O(a^n)$

En general se puede plantear el teorema para funciones decrecientes de la siguiente manera. CUIDADO!!! Observe que este teorema tiene un “<=” por lo que la cuota puede ser muy alta.

$T(n) \leq a*T(n-b) + f(n)$

con $a>0$, $b>0$, $f(n)=O(n^k)$, $k\geq 0$

Entonces

Si $a<1$ entonces $O(n^k)$

Si $a=1$ entonces $O(n^{(k+1)})$

Si $a>1$ entonces $O(n^k * a^{(n/b)})$

Veamos algunas cuotas demasiado altas producidas por el teorema para las funciones decrecientes.

T(n)		TMFD
$T(n) = 2*T(n-1) + n = O(2^n)$	$\leq$	$O(n*2^n)$
$T(n) = 3*T(n-1) + n = O(3^n)$	$\leq$	$O(n*3^n)$
$T(n) = 4*T(n-1) + n = O(4^n)$	$\leq$	$O(n*4^n)$
$T(n) = a*T(n-1) + n = O(a^n)$	$\leq$	$O(n*a^n)$

▬▬▬▬ 8. Seguimos con Otras Relaciones de Recurrencia.

Se tiene el siguiente algoritmo:

```
test(n):=block
(
   if (n>1) then
   (
      print(n),
      test(n/2)
   )
);
```

Analicemos el algoritmo:

```
test(n):=block
(
   if (n>=1) then
   (
      print(n),  ----> 1
      test(n/2)  ----> T(n/2)
   )
);
```

Por lo tanto:

$T(0) = 1$

$T(n) = T(n/2) + 1$

Al resolver esta relación se tiene:

$T(n) = \log(n) = O(\log(n))$ cuando $n \to +\infty$

>>> Hay que resolverlo
    En Wolfram!

$t(0)=1,\ t(n)=t(n/2) + 1$

Respuesta:

$t(n) \sim \log_2(n)$, cuando $n \to \infty$

▬▬▬▬ 9. Otro Ejemplo.

Tenemos el siguiente programa:

```
test(n):=block
(
   if (n>=1) then
   (
      for i:1 thru n do
      (
         print(i)
      ),
      print("---"),
      test(n/2),
      test(n/2)
   )
);
```

Veamos el análisis de tiempo:

```
test(n):=block
(
   if (n>=1) then ----------> 1
   (
      for i:1 thru n do  ---> n
      (
         print(i) ---> 1
      ),
      print("---"), --------> 1
      test(n/2), -----------> T(n/2)
      test(n/2)  -----------> T(n/2)
   )
);
```

Tenemos que:

$T(0) = 1$

$T(n) = T(n/2) + T(n/2) + n + 2$

$T(n) = 2*T(n/2) + n$

Por lo tanto:

$T(0) = 1$

$T(n) = 2*T(n/2) + n + 2$

Cuya solución está dada por:

$T(n) = n * \log(n) = O(n*\log(n))$ cuando $n \to +\infty$

>>> Hay que resolverlo
    En Wolfram

$t(0)=1, t(n)=2*t(n/2)+n+2$

$t(n) = n * \log(n) = O(n*\log(n))$ cuando $n \to +\infty$

▬▬▬▬ 10. Resumen de Algunas Recurrencias.

Tenemos los siguientes resultados:

$T(n) = 2*T(n/2) + 1 = O(n)$

$T(n) = 4*T(n/2) + 1 = O(n^2)$

$T(n) = 8*T(n/2) + 1 = O(n^3)$

$T(n) = 16*T(n/2) + 1 = O(n^4)$

Tenemos también que:

$T(n) = 4*T(n/2) + n = O(n^2)$

$T(n) = 8*T(n/2) + n^2 = O(n^3)$

$T(n) = 16*T(n/2) + n^3 = O(n^4)$

Tenemos estos otros resultados:

$T(n) = 1*T(n/2) + n = O(n)$

$T(n) = 2*T(n/2) + n^2 = O(n^2)$

$T(n) = 3*T(n/2) + n^3 = O(n^3)$

$T(n) = 4*T(n/2) + n^4 = O(n^4)$

Y este último caso:

$T(n) = 1*T(n/2) + 1 = O(\log n)$

$T(n) = 2*T(n/2) + n = O(n * \log n)$

Con base en estos resultados, podemos enunciar el siguiente teorema maestro.

▬▬▬▬ 11. Master Theorem For Dividing Functions.

Tenemos:

$T(1) = c$

T(n) = a*T(n/b) + f(n)

Donde a ≥ 1, b ≥ 1, c > 0

Si f(n) tiene la forma:

f(n) = O(n^d)

Entonces:

• Si a < (b^d), entonces T(n) es O(n^d)

• Si a = (b^d), entonces T(n) es O(n^d * log(n))

• Si a > (b^d), entonces T(n) es O(n^(log_b(a))

▬▬▬ 12. Ejemplo del Teorema Maestro.

Sea

T(n) = a*T(n/b) + f(n)

T(n) = 2*T(n/2) + 1

Tenemos:
a=2
b=2
d=0

f(n) = O(1) = O(n^0) = O(n^d) entonces d=0

Tenemos

a=2

b^d = 2^0 = 1,

Como a > b^d entonces

O(n^(log_b(a))

= O(n^log_2(2))

= O(n^1)

= O(n)

▬▬▬▬ 13. Ejemplo del Teorema Maestro.

Sea

T(n) = a*T(n/b) + f(n)

T(n) = 4*T(n/2) + n

Tenemos:
a=4
b=2
d=1

f(n) = O(n) = O(n^d) = O(n^1) entonces d=1

Tenemos:

a=4

b^d = 2^1 = 2

Como a > b^d, entonces

O(n^(log_b(a))

= O(n^(log_2(4))

= O(n^2)

▬▬▬▬ 14. Ejemplo del Teorema Maestro.

Sea

T(n) = a*T(n/b) + f(n)

T(n) = 8*T(n/2) + n

Tenemos:
a=8
b=2
d=1

f(n) = O(n) = O(n^1)

Tenemos

a=8

b^d = 2^1 = 2

Como a > b^d, entonces:

O(n^(log_b(a))

= O(n^(log_2(8))

= O(n^3)

▬▬▬▬ 15. Ejemplo del Teorema Maestro.

Sea

T(n) = a*T(n/b) + f(n)

T(n) = 2*T(n/2) + n

Tenemos:
a=2
b=2
d=1

Pues
f(n) = O(n) = O(n^1)

Tenemos:

a = 2

b^d = 2^1 = 2

Como a=b^d entonces:

O(n^d * log(n))

= O(n^1 * log(n))

= O(n * log(n))

▬▬▬▬ 16. Ejemplo del Teorema Maestro.

Sea

T(n) = a*T(n/b) + f(n)

T(n) = 4*T(n/2) + n^2

Tenemos:
a=4
b=2
d=2

Entonces:

a=4

b^d = 2^2 = 4

Como a=b^d, se tiene:

O(n^d * log(n))

= O(n^2 * log(n))

▬▬▬▬ 17. Ejemplo del Teorema Maestro.

Sea

T(n) = a*T(n/b) + f(n)

T(n) = 8*T(n/2) + n^3

Tenemos:
a=8
b=2
d=3

Como:
a=8
b^d = 2^3 = 8

Como a=b^d, entonces:

O(n^d * log(n))

= O(n^3 * log(n))

▬▬▬▬ 18. Ejemplo del Teorema Maestro.

Sea

T(n) = a*T(n/b) + f(n)

T(n) = 1*T(n/2) + n^2

Tenemos:
a=1
b=2
d=2

Como:
a=1
b^d = 2^2 = 4

Como:
a < b^d

O(n^d)

= O(n^2)

▬▬▬▬ 19. Comentario sobre el Teorema Maestro.

Aunque el teorema maestro es muy útil para calcular una serie de valores, tiene sus limitaciones. En particular no se puede usar si:

• T(n) no es monótona, por ejemplo T(n)=sin(n)

• f(n) no es un polinomio, por ejemplo f(n)= 2^n

• “a” no es una constante, por ejemplo a=2*n

• Si a<1

▬▬▬▬ 20. Resumen.

• Se puede obtener el tiempo de ejecución de una función recursiva mediante relaciones de recurrencia.

• Algunas relaciones de recurrencia son muy frecuentes, por lo que existe el Teorema Maestro para Funciones Decrecientes y el Teorema Maestro General.

▬▬▬▬ 21. Ejercicios.

●● Ejercicio 1.

A continuación se da una relación de recurrencia interesante, realice un proceso manual para encontrar su valor:

T(2) = 1

T(n) = T(√n) + 1

●● Ejercicio 2.

Utilice el Teorema Maestro para resolver la siguiente relación de recurrencia:

T(n) = a*T(n/b) + f(n)

T(n) = 4*T(n/2) + n^3