UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Algoritmos
Germán J. Hernández

Taller 1
Cormen

Miguel Angel Ortiz Marin - 1032504680

1. Desarrolle los siguientes ejercicios del libro [Cormen09]

a. 3.1-2 Show that for any real constants a and b,where b>0, $(n + a)^b = \Theta(n^b)$

Respuesta:

Sea $c = 2^b$ y $n_0 \geq 2a$. Entonces para todo $n \geq n_0$. se tiene que $(n+a)^b \leq (2n)^b = cn^b$ y por lo tanto $(n+a)^b = O(n^b)$.

Ahora sea $n_0 \geq \frac{-a}{1 - 1/2^{1/b}}$ y $c = 1/2$.

Entonces $n \geq n_0 \geq \frac{-a}{1 - 1/2^{1/b}}$

ssi $n - \frac{n}{2^{1/b}} \geq -a$

ssi $n + a \geq (\frac{1}{2})^{a/b} n$

ssi $(n + a)^b \geq cn^b$.

Por lo tanto $(n+a)^b = \Omega(n^b)$.

By Theorem 3.1, $(n + a)^b = \Theta(n^b)$.

b. 3.1-7 Prove that $o(g(n)) \cap \omega(g(n))$ is the empty set.

Podemos definir formalmente $o(g(n)) \; y \; \omega(g(n))$ de la siguiente manera:

1. Por definición $o(g(n)) = \{f(n) : \forall c, \; n_0 \; tal \; que \; 0 \leq f(n) \leq cg(n), \; \forall n > n_0\}$
2. Por definición $\omega(g(n)) = \{f(n) : \forall c, \; n_0 \; tal \; que \; 0 \leq cg(n) \leq f(n), \; \forall n > n_0\}$

Podemos entonces definir $o(g(n)) \cap \omega(g(n))$ como:

$o(g(n)) \; y \; \omega(g(n)) = \{f(n) : \forall c, \; n_0 \leq cg(n) \leq f(n) \leq cg(n), \; \forall n > n_0\}$

Esta desigualdad no se cumple para ninguna función f (n) por lo tanto la intersección de o(g(n)) y ω(g(n)) debe ser el conjunto vacío.

c. Problema **3.3 a)** Rank the following functions by order of growth; that is, find an arrangement g1, g2, …. , g30 of the functions satisfying $g1 = \Omega(g2),\ g2 = \Omega(g3),\ \ldots\ , g29 = \Omega(g30)$ .Partition your list into equivalence classes such that functions f(n) and g(n) are in the same class if and only if $f(n) = \Theta(g(n))$.

$\lg(\lg^* n)$	$2^{\lg^* n}$	$(\sqrt{2})^{\lg n}$	n^2	$n!$	$(\lg n)!$
$(\frac{3}{2})^n$	n^3	$\lg^2 n$	$\lg(n!)$	2^{2^n}	$n^{1/\lg n}$
$\ln\ln n$	$\lg^* n$	$n \cdot 2^n$	$n^{\lg\lg n}$	$\ln n$	1
$2^{\lg n}$	$(\lg n)^{\lg n}$	e^n	$4^{\lg n}$	$(n+1)!$	$\sqrt{\lg n}$
$\lg^*(\lg n)$	$2^{\sqrt{2\lg n}}$	n	2^n	$n \lg n$	$2^{2^{n+1}}$

Respuesta:

1	$2^{2^{n+1}}$	
2	2^{2^n}	
3	$(n+1)!$	
4	$n!$	
5	$n2^n$	
6	e^n	
7	2^n	
8	$\left(\frac{3}{2}\right)^n$	
9	$(\lg(n))!$	
10	$n^{\lg(\lg(n))}$	$\lg(n)^{\lg(n)}$
11	n^3	
12	n^2	$4^{\lg(n)}$

13	$n\lg(n)$	$\lg(n!)$
14	$2^{\lg(n)}$	n
15	$(\sqrt{2})^{\lg(n)}$	
16	$2^{\sqrt{2\lg(n)}}$	
17	$\lg^2(n)$	
18	$\ln(n)$	
19	$\sqrt{\lg(n)}$	
20	$\ln(\ln(n))$	
21	$2^{\lg^*(n)}$	
22	$\lg^*(n)$	$\lg^*(\lg(n))$
23	$\lg(\lg^*(n)$	
24	1	$n^{1/\lg(n)}$

3.3 b) Give an example of a single nonnegative function f(n) such that for all functions gi (n) in part (a), f(n) is neither O(gi (n)) nor $\Omega(gi\,(n))$.

Respuesta:

Si definimos la función

$$f(n) = \begin{cases} g_1(n)! & n \mod 2 = 0 \\ \frac{1}{n} & n \mod 2 = 1 \end{cases} :$$

Teniendo en cuenta que f(n) es asintóticamente positiva.

Para n par tenemos:

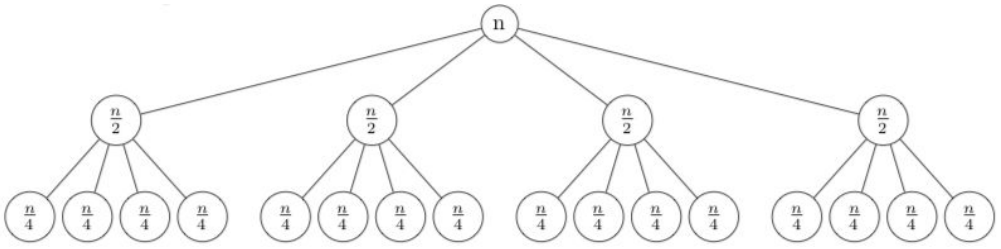
$$\begin{aligned} \lim_{n\to\infty} \frac{f(2n)}{g_i(2n)} &\geq \lim_{n\to\infty} \frac{f(2n)}{g_1(2n)} \\ &= \lim_{n\to\infty} (g_1(2n)-1)! \\ &= \infty \end{aligned}$$

Para n impar tenemos:

$$\begin{aligned}\lim_{n\to\infty}\frac{f(2n+1)}{g_i(2n+1)} &\leq \lim_{n\to\infty}\frac{f(2n+1)}{1}\\ &= \lim_{n\to\infty}\frac{1}{2n+1}\\ &= 0\end{aligned}$$

d. 4.4-7 Draw the recursion tree for T(n) = 4T([n/2]) + cn, where c is a constant, and provide a tight asymptotic bound on its solution. Verify your bound by the substitution method.

Respuesta:



Se incrementa el número de subproblemas por cada recursión en 4. Se disminuye el tamaño del subproblema en 2 más 2 adicionales. De esa manera, en cada nivel del árbol hay 4i nodos cada uno de coste c(n/2+2) a una profundidad i, que sigue la secuencia i = 0,1,2,...,lgn.

El costo total del árbol es:

$$\begin{aligned}T(n) &= \sum_{i=0}^{\lg n} 4^i \cdot c\left(\frac{n}{2^i}+2\right)\\ &= \sum_{i=0}^{\lg n} 4^i \cdot c\frac{n}{2^i} + \sum_{i=0}^{\lg n} 4^i \cdot c \cdot 2\\ &= cn\sum_{i=0}^{\lg n}\frac{4^i}{2^i} + 2c\sum_{i=0}^{\lg n} 4^i\\ &= cn\sum_{i=0}^{\lg n} 2^i + 2c\sum_{i=0}^{\lg n} 4^i\\ &= cn\frac{2^{\lg n+1}-1}{2-1} + 2c\frac{4^{\lg n+1}-1}{4-1}\\ &= cn(2^{\lg n+1}-1) + \frac{2c}{3}(4^{\lg n+1}-1)\\ &= cn(2\cdot 2^{\lg n}-1) + \frac{2c}{3}(4\cdot 4^{\lg n}-1)\\ &= cn(2\cdot n-1) + \frac{2c}{3}(4\cdot n^2-1)\\ &= 2cn^2 - 2n + \frac{8cn^2}{3} - \frac{2c}{3}\\ &= O(n^2)\end{aligned}$$

Se verifica el resultado del problema por el método de sustitución.

$$
\begin{aligned}
T(n) &= 4T(n/2+2)+n \\
&\leq 4d((n/2+2)^2 - b(n/2+2)) + n \\
&= 4d(n^2/4 + 2n + 4 - bn/2 - 2b) + n \\
&= dn^2 + 8dn + 16d - 2dbn - 8db + n \\
&= dn^2 - dbn + 8dn + 16d - dbn - 8db + n \\
&= d(n^2 - bn) - (db - 8d - 1)n - (b-2)8d \\
&\leq d(n^2 - bn)
\end{aligned}
$$

Esto se mantiene para db-1-8d>=0.

e. Use el método maestro para dar cotas ajustadas para las siguientes recurrencias:

- T (n) = 8T (n/2) + n
- T (n) = 8T (n/2) + n^3
- T (n) = 8T (n/2) + n^5

En estos problemas se puede ver que a=8,b=2 y f(n)=n,n^2,n^3 respectivamente. Se compara f(n) con $n^{log_b a} = n^{log_2 8}$. Las recurrencias es cada una de un caso diferente del teorema maestro.

Por lo tanto:

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

$n^{log_2 8}$ = n^3 = Θ(n^3), la función $f(n)$ tiene una cota Θ($n^{log_b(a)-e}$) $con\ \epsilon$ = 2

por lo tanto $T(n) = O(n^3)$

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

$n^{log_2 8}$ = n^3 = Θ(n^3), la función $f(n)$ tiene una cota Θ($n^{log_b a}$)

por lo tanto $T(n) = O(n^3 log(n))$

T(n) = 8T(n/2) + n^5

$n^{log_2 8}$ = n^3 = Θ(n^3), la función $f(n)$ tiene una cota Θ($n^{log_b(a)+e}$) $con\ \epsilon$ = 2

por lo tanto $T(n)$ = Θ(n^5)

2. Dado el siguiente pseudocódigo:

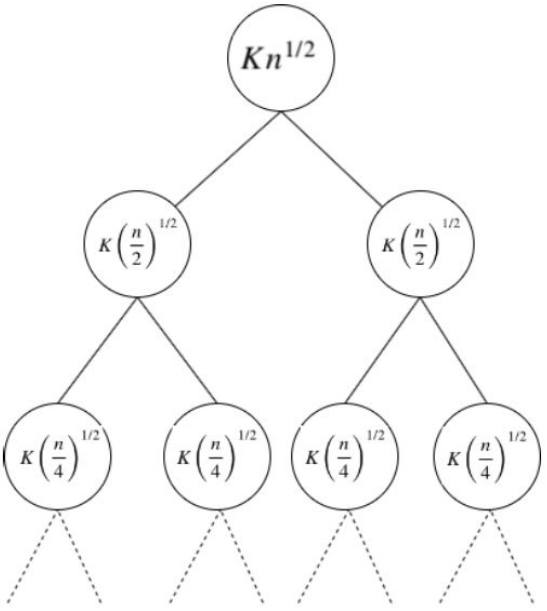
```
def misterio(n):
      if n <= 1:
            return 1
      else:
            r = misterio(n / 2)
            i = 1
            while n > i*i:
                  i = i + 1
            r = r + misterio(n / 2)
            return r
```

a. Plantee una ecuación de recurrencia para T (n), el tiempo que toma la función misterio(n).

T(n)=2T(n/2)+Θ(n^(1/2))

b. Dibuje el árbol de recursión y calcule:

i. La altura del mismo
ii. El número de nodos por cada nivel
iii. La suma de los nodos de cada nivel
iv. . La suma total



v. La altura del mismo: $log_2(n)$
vi. El número de nodos por cada nivel: $2^{log_2(n)}$
vii. La suma de los nodos de cada nivel: $2^i k(\frac{n}{2^i})^{1/2}$
viii. La suma total: $\Theta(n)$

c. Determine el comportamiento asintótico de T (n) justificándolo de manera detallada.

Por el método maestro podemos estudiar el comportamiento asistólico de la función de recurrencia T(n):

Donde es de la forma $O(n)$ con ϵ = 0,5
por lo tanto $T(n)$ = Θ(n)

3. **22.3-1** (pág 610) Realizar un cuadro 3x3 con filas y columnas WHITE, GRAY, BLACK. En cada celda (i,j), indique si en algún momento durante la búsqueda en la profundidad de un grafo dirigido, puede haber una arista o un vértice de color i a un vértice de color j. Por cada arista posible indique que tipo puede ser. Haga otro cuadro pero utilizando un grado no dirigido

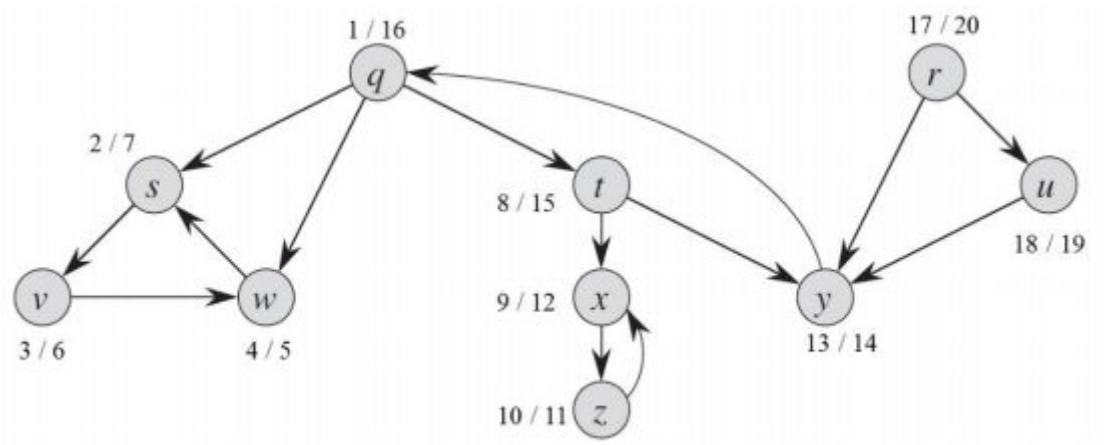
Para el grafo dirigido.

i/j	White	Gray	Black
White	Allkinds	Cross, Back	Cross, Tree, Forward.
Gray	Back, Cross	Tree, Forward, Back	Tree, forward and cross.
Black	Back, Cross	Back, Cross.	Allkinds

Para el grafo no dirigido.

i/j	White	Gray	Black
White	Allkinds.	-	-
Gray	Allkinds.	Tree,Forward,Back.	-.
Black	Allkinds.	Allkinds	Allkinds

4. **22.3-2** (Pág 611) Muestre como DFS funciona en el grafo de la figura 22.6. Asuma que el ciclo for de la línea 5-7 del DFS considera los vértices en orden alfabético, y asuma que cada lista de adyacencia está ordenada alfabéticamente. Muestre el descubrimiento y los tiempos de cada vértice y muestre la clasificación de cada arista.



- Forward edges: (q,w).
- Back edges: (z,x), (w,s), (y,q).
- Tree edges: (q,s),(s,v),(v,w),(q,t),(t,x),(x,z),(t,y),(r,u).
- Cross edges: (u,y),(r,y)

5. **22.4-2** Dé un algoritmo de tiempo lineal que tome como input un grado dirigido acíclico $G = (V, E)$ y dos vértices s y t, y retorna el número de caminos simples de s a t en G.

   Añada un campo a la representación de vértices para mantener un número entero. Inicialmente, establece la cuenta del vértice t en 1 y la de otros vértices en 0. Comienza ejecutando DFS con s como vértice de inicio. Cuando se descubre t, debe marcarse inmediatamente como terminado (NEGRO), sin ningún otro tratamiento a partir de él. Posteriormente, cada vez que DFS finaliza un vértice v, establece el conteo de v como la suma de los conteos de todos los vértices adyacentes a v. Cuando DFS finalice el vértice s, deténgase y devuelva el conteo calculado para s.

```
def SIMPLE-PATHS(u,v):
    if u == v:
        return 1
    elif u.paths == None:
        return u.paths
    else
    for w in adj[u]:
        paths.append(SIMPLE-PATHS(w,v))
    return u.paths
```