TALLER 1 ALGORITMOS. | 2018-1.

UNIVERSIDAD NACIONAL DE COLOMBIA. PROFESOR: GERMAN HERNANDEZ.
NOMBRE: JHON JAIRO MUESES Q.
CODIGO: 2879355.

① Desarrolle los siguientes ejercicios del libro [Cormen 09]

(a) Ejercicio 3.1-2 (Pag 52). Para cualquier constante real a y b, donde $b > 0$, muestre que

$$(n+a)^b = \Theta(n^b)$$

R/.

Sea $c = 2^b$ y $n_0 \geq 2a$. Entonces para todo $n \geq n_0$ tenemos $(n+a)^b \leq (2n)^b = cn^b$
así que $(n+a)^b = O(n^b)$

Sea $n_0 \geq \dfrac{-a}{1 - \frac{1}{2^{1/b}}}$ y $c = \frac{1}{2}$ entonces:

$$(n \geq n_0) \geq \frac{-a}{1 - \frac{1}{2^{1/b}}} \quad \text{si y solo si}$$

$$\left(n - \frac{n}{2^{1/b}}\right) \geq -a \quad \text{si y solo si} \quad (n+a) \geq \left(\tfrac{1}{2}\right)^{a/b},$$

si y solo si $(n+a)^b \geq cn^b$. Por lo tanto

$(n+a)^b = \Omega(n^b)$. por teorema 3.1.

$(n+a)^b = \Theta(n^b)$

(b) Ejercicio 3.1-7 (pag 53)

probar que $o(g(n)) \cap \omega(g(n))$ es el conjunto Vacío.

Supongamos que tenemos $f(n) \in o(g(n)) \cap \omega(g(n))$, Entonces

$$0 = \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

una contradicción.

(c) Problema 3.3 (pag 61)

a) Ordenamiento por tasas de crecimiento asintotico.

Clasificar las siguientes funciones por orden de crecimiento, esto es encontrar un arreglo de funciones $g_1, g_2 \dots g_{30}$ satisfaciendo que $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3), \dots$ $g_{29} = \Omega(g_{30})$. las particiones de la lista en clases de equivalencia tal que las funciones $f(n)$ y $g(n)$ estan en la misma clase si y solo si $f(n) = \Theta(g(n))$.

$\lg(\lg^* n)$	$2^{\lg^* n}$	$(\sqrt{2})^{\lg n}$	n^2	$n!$	$(\lg n)!$
$\left(\frac{3}{2}\right)^n$	n^3	$\lg^2 n$	$\lg(n!)$	2^{2^n}	$n^{1/\lg n}$
$\ln \ln n$	$\lg^* n$	$n . 2^n$	$n^{\lg \lg n}$	$\ln n$	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	$n \lg n$	$2^{2^{n+1}}$

5/.

los terminos estan en una taza de crecimiento decreciente. las funciones en la misma fila son Θ de cada otra.

$2^{2^{n+1}}$	
2^{2^n}	
$(n+1)!$	
$n!$	
$n2^n$	
e^n	
2^n	
$\left(\frac{3}{2}\right)^n$	
$(\lg n)!$	
$n^{\lg(\lg n)}$	$\lg(n)^{\lg(n)}$
n^3	
n^2	$4^{\lg(n)}$
$n\lg(n)$	$\lg(n!)$
$2^{\lg(n)}$	n
$(\sqrt{2})^{\lg n}$	
$2^{\sqrt{2\lg n}}$	
$\lg^2(n)$	
$\ln(n)$	
$\sqrt{\lg(n)}$	
$\ln(\ln(n))$	
$2^{\lg^* n}$	
$\lg^*(n)$	$\lg^*(\lg(n))$
$\lg(\lg^* n)$	
1	$n^{\frac{1}{\lg n}}$

(b) dar un ejemplo de una sola función no negativa $f(n)$ tal que para todas las funciones $g_i(n)$ de la parte (a), $f(n)$ no es $O(g_i(n))$ ni $\Omega(g_i(n))$.

R/. Si nosotros definimos la función $f(n) = \begin{cases} g_1(n)! & n \bmod 2 = 0 \\ \frac{1}{n} & n \bmod 2 = 1 \end{cases}$

Note que $f(n)$ cumple los requerimientos asintoticamente positivos.

Entonces para todo n dado, se tiene: (n par)

$$\lim_{n\to\infty} \frac{f(2n)}{g_i(2n)} \geq \lim_{n\to\infty} \frac{f(2n)}{g_1(2n)}$$

$$= \lim_{n\to\infty} (g_1(2n)-1)!$$

$$= \infty$$

y para n impar se tiene:

$$\lim_{n\to\infty} \frac{f(2n+1)}{g_i(2n+1)} \leq \lim_{n\to\infty} \frac{f(2n+1)}{1}$$

$$= \lim_{n\to\infty} \frac{1}{2n+1}$$

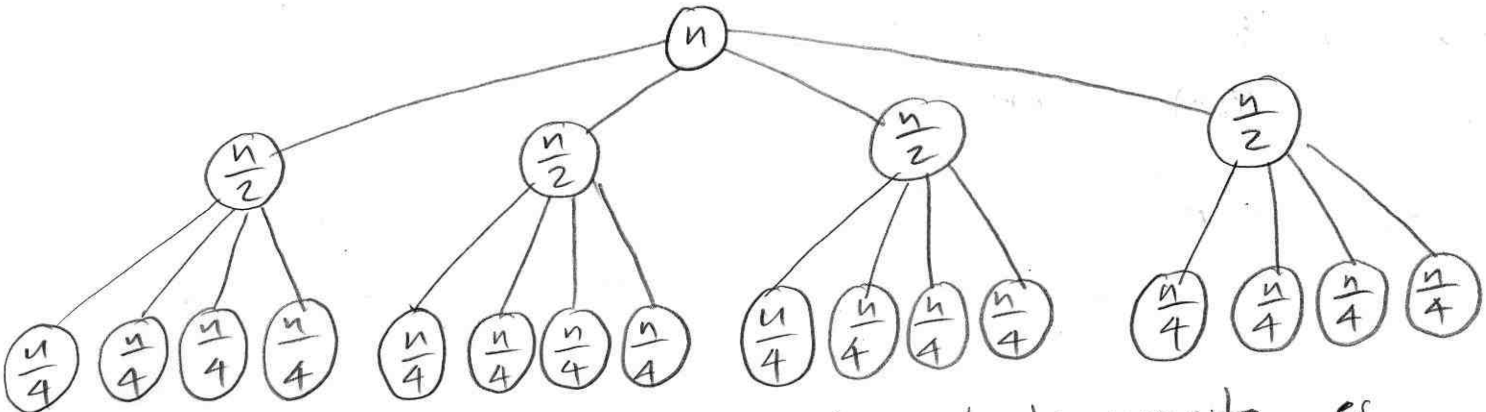
$$= 0.$$

Al mirar n par tenemos que $f(n)$ no es $O(g_i(n))$ para cualquier n pero mirando para n impar, nosotros tenemos que $f(n)$ no es $\Omega(g_i(n_i))$ para cualquier n.

(d) Ejercicio 4.4-7 (pag 93)

Dibujar el arbol de recursión para $T(n) = 4T(\lfloor n/2 \rfloor) + cn$, donde c es una constante y proporciona un limite asintotico, ligado en esta solución. Verifique su limite por el metodo de sustitución.

Este es un ejm para $n=4$



podemos ver que por una sustitución facil que la la respuesta es $\Theta(n^2)$. Suponga que $T(n) \le c'n^2$ entonces

$$T(n) = 4T(\lfloor n/2 \rfloor) + cn$$
$$\le c'n^2 + cn$$

El cual es $\le c'n^2$ cuando nosotros tenemos que $c' + \frac{c}{n} \le 1$, las cuales un n suficientemente grande es verdadero así como $c' < 1$. podemos hacer algo similar para mostrar que también esta limitado debajo de n^2

(c) Use el metodo maestro para dar cotas ajustadas para las siguientes recurrencias.

pag. (95)

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

$a = 8$

$b = 2$

$c = 1$

$\log_2 8 = x \rightarrow 2^x = 8$

$\log_2 8 = 3 \rightarrow 2^3 = 2 \times 2 \times 2 = 8.$

$1 < \log_2 8$

asi que $T(n) = \Theta(n^{\log_2 8})$

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

$a = 8$

$b = 2$

$c = 3$

$k = 1$

$3 \geq \log_2 8 \Rightarrow T(n) = \Theta(n^3 \log_2 n)$

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

$T(n) = aT(\frac{n}{b}) + n^c \quad a \geq 1, b \geq 1, c > 0$

$\Downarrow$

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & a > b^c \\ \Theta(n^c \log_b n) & a = b^c \\ \Theta(n^c) & a < b^c \end{cases}$$

$a = 8$

$b = 2$

$c = 5$

$T(n) = \Theta(n^5)$

② Dado el siguiente pseudocodigo:

```
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)

$\boxed{n=1}$

def misterio (1) $n \leq 1$
retur 1. $1 \leq 1$ ✓

$\boxed{n=2}$

r= misterio $\left(\frac{2}{2}\right)$

r= misterio (1)

r= 1.

i= 1.

while 2 > 1

i = 2

r= 1 + 1

r= 2.

$\boxed{n=3}$ r= mist(3/2) = 1

i= 1

while 3 > 1 3>4

i= 2

r= 1 + 1

r= 2.

$\boxed{n=4}$ r= misterio (2)

r= 2

i= 1

while 4 > 1 4>4

i= 2

r= 2 + 2

r= 4.

$$T(n) = \begin{cases} \Theta(1) & \text{si } n \leq 1 \\ T(n/2) + \Theta(n^2) \end{cases}$$

③. Ejercicio 22-3-1 (pag 610)
Haga un cuadro de 3X3 con etiquetas de fila y columna Blanco, gris, y negro. En cada celda i, j indicar si en cualquier punto durante una busqueda en profundidad de un grafo dirigido, puede haber un borde desde un vertice de color i hasta un vertice de color j. Para cada borde posible, indique que tipos de borde puede ser. Haga una segunda tabla de este tipo para la busqueda en profundidad de un grafo no dirigido.

para grafo dirigido

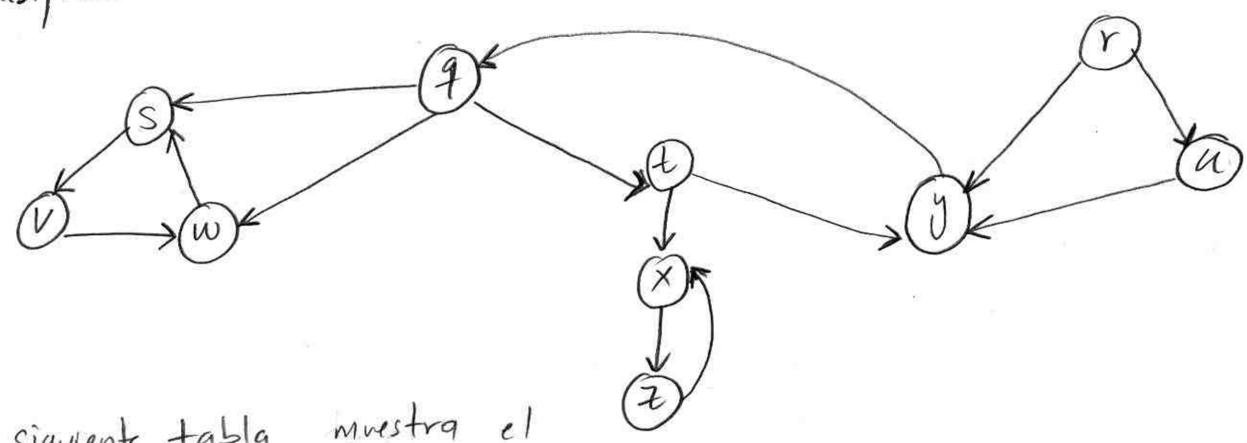
DE \ hasta	NEGRO	GRIS	BLANCO
NEGRO	todos los tipos	Atras cruzar	Atraz cruzar
GRIS	arbol, adelante cruzar	arbol adelante atras	atras cruzar
BLANCO	atras, arbol, adelante	cruzar atras	todos los tipos

para un grafo no dirigido

DE \ HASTA	NEGRO	GRIS	BLANCO
NEGRO	Todos los tipos	Todos los tipos	Todos los tipos
GRIS	—	Arbol, adelante, atras	Todos los tipos
BLANCO	—	—	Todos los tipos

(4) Ejercicio 22.3-2 (pag 611)

Muestre como funciona la busqueda de profundidad en el grafico de la figura 22.6. Supongamos que para el bucle de las lineas 5-7 del procedimiento DFS considera los vertices en orden alfabetico y supone que cada linea de adyacencia se ordena alfabeticamente. Muestre los tiempos de descubrimiento y finalización para cada vertice y muestre la clasificación de cada borde:



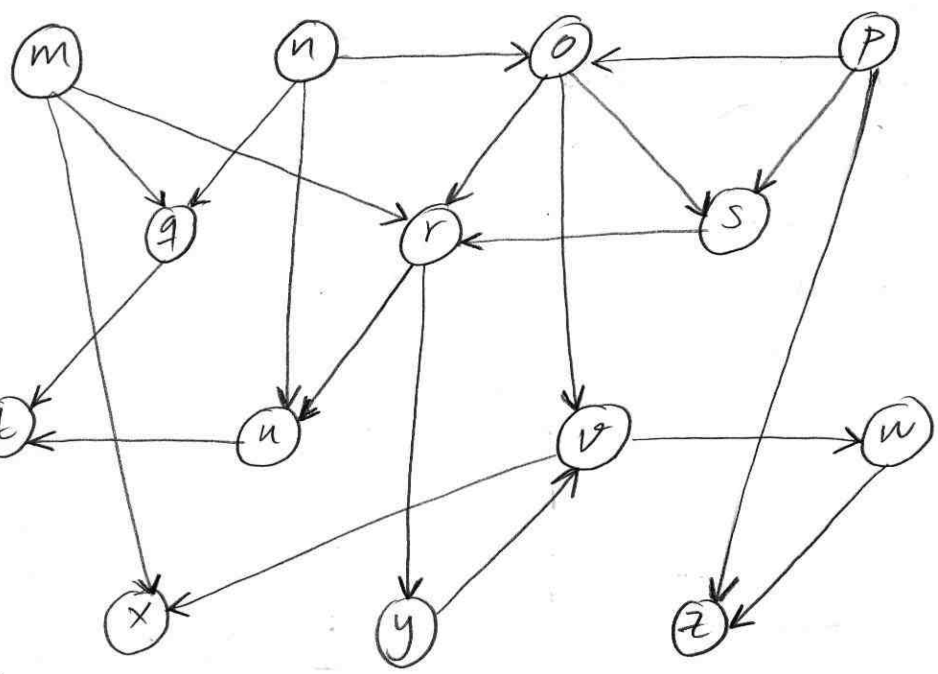
la siguiente tabla muestra el tiempo de descubierto y tiempo final para cada vertice en el grafo.

Vertice	Descubierto	finalizado
q	1	16
r	17	20
s	2	7
t	8	15
u	18	19
v	3	6
w	4	5
x	9	12
y	13	14
z	10	11

las aristas del arbol son: (q, s), (s, v), (v, w), (q, t), (t, x), (x, z), (t, y), (r, u). las aristas posteriores son. (w, s), (y, q), (z, x) las aristas delanteras son: (u, y), (r, y).

5) Ejercicio 22.4-2 (Pag 614)

Dar un algoritmo en tiempo lineal que toma como entrada un grafo aciclico dirigido $G = (V, E)$ y 2 vertices s y t, y retorna el numero de simples rutas de s a t en G. Por ejemplo el grafo aciclico dirigido de la figura 22.8 contiene exactamente 4 rutas simples del vertice p al vertice v: pov, $poryv$, $posryv$ y $psryv$. (tu algoritmo necesita solo contar las simples rutas)



El algoritmo trabaja de la siguiente forma: El atributo u.paths de nodo u le dice el numero simple de rutas u a v, donde nosotros asumimos que v es fijo a lo largo de todo el proceso. Contar el numero de rutas, podemos sumar el numero de rutas que salen de cada uno de sus vecinos. Como no tenemos ciclos, nunca correremos el riesgo de agregar un numero parcialmente completado de rutas. Ademas nunca podemos considerar el mismo limite 2 veces entre las llamadas recursivas. Por lo tanto, el numero total de ejecusiones del ciclo for sobre todas las llamadas recursivas es $O(V+E)$. llamar a SIMPLE-PATHS(s, t) produce el resultado deseado.

```
SIMPLE_PATHS (u, v)
 if  u == v then
       Return 1
 else if   u.paths ≠ NULL     then
       Return u.paths

 else
       for each  w ∈ Adj[u] do.
            u.paths = u.paths + SIMPLE_PATHS(w, v)
       end for
       Return  u.paths
 end if.
```