Apuntes de Teórico

PROGRAMACIÓN 3

Análisis de Algoritmos

Version 1.2

Índice

Análisis de algoritmos

Introducción

Algoritmo:

i. Secuencia finita de pasos
ii. Cada paso correctamente definido
iii. Cada paso debe ejecutarse en un tiempo finito
iv. Termina en algún momento
v. Devuelve el resultado esperado
vi. Tiene un dominio de definición

Tiempo de ejecución

n^3 ; n = 45 $\rightarrow$ 0.09 s
2^n ; n = 45 $\rightarrow$ 1.1 años

Conceptos Básicos

Interesa calcular de antemano cuanto tiempo puede llevar la ejecución de un algoritmo (aproximadamente): Análisis del algoritmo.
Se tendrán en cuenta sólo ciertas operaciones simples de los algoritmos, p.e.: asignación, comparación, suma, resta, etc.- Cada operación simple tiene un costo asociado.

Se verán tres posibles puntos de vista para el análisis:

1) Contar las operaciones simples con sus costos
2) Contar las operaciones simples sin tener en cuenta costos
3) Contar sólo una operación predefinida y/o básica

1. Contar las operaciones que realiza el algoritmo y tener el costo de cada operación
   a. x:= x+y
      2op :
      suma : costo c1
      asignación: costo c2
      T = c1 + c2

   b. for i:= 1 to n
         x:=x+y
      2 sumas; 2 asignaciones
      T = 2(c1+c2)*n
      variable de control : $\rightarrow$ 2

c. caso general:
Algoritmo con $p_1, p_2, p_3, \ldots, p_m$ operaciones
costo(p_i) = c_i
cantidad(p_i) = k_i

$$T \leq \sum_{i=1}^{m} ci * ki$$

2. Contar las operaciones

$$T \leq \sum_{i=1}^{n} ki$$

a. Caso ejemplo a:
T = 2

b. Caso ejemplo b:
T = 4n (5n con la comparación)

Comparación de 2 algoritmos: A1 y A2
A1: T1 = 10000 n
A2: T2 = n^2 (A2 es mejor que A1 para n < 10000)

Ejemplo: Mínimo de una secuencia

```
// a array of int [0...N-1]
// n cant elementos;

int minimo(int* a,int n){
  int min;
  min = a[0];
  int i;
  for( i=1;i<n;i++ ){
       if (a[i] < min){
            min = a[i];
       }
  }
  return min;
}
```

* operaciones: asignación, comparación de elementos, comparación de variable de control, incremento de la variable de control.

T = 2 + n + 2*(n-1) + X*(n-1) donde X ∈ [0..1], $X = \frac{\sum_{1}^{n-1} x_i}{n-1}$ donde si la comparación ***a[i] < min*** es verdadera entonces x_i=1 si no x_i=0

X es 0 si ***min*** está en el primer lugar → Mejor caso
X es 1 si la secuencia esta ordenada de forma decreciente → Peor caso.

$$T = \begin{cases} 2 + 3n - 2 = 3n \\ 2 + 4n - 2 - 1 = 4n - 1 \end{cases}$$

T es el costo en tiempo de ejecución de un algoritmo.
T depende del tamaño de la entrada: T es función del tamaño de la entrada.
T depende del ordenamiento de la secuencia (en el ejemplo anterior)

3. Complejidad de un algoritmo

Contar una operación específica/básica.
operación básica: es una operación que determina la forma del algoritmo.
ej:

```
A[i] < min
```

comparación de elementos de la secuencia
No cuenta otro tipo de comparaciones como variables de control.

T(n) = n -1

Se hacen n -1 comparaciones en el peor y mejor caso.

Dominio de definición y

Análisis en Peor Caso, Mejor Caso y Caso Medio

Dominio de definición D, es el conjunto de TODAS las entradas posibles.
Mantiene la correctitud del algoritmo.

Dn = {E / E ∈ D, Tamaño(E) = n}

T(E) = tiempo/cantidad/complejidad (costo según la forma de estudio 1, 2 ó 3) para la entrada E del algoritmo.

Se busca determinar, analizando un algoritmo dado, el costo del mismo expresándolo como una función **T** del tamaño ***n*** de la entrada: T(n).

Análisis:

No es viable estudiar en caso genérico, se tratará de obtener la función T(n) en:

- peor caso: se obtendrá el mayor costo posible
- mejor caso: se tendrá el menor costo posible
- caso promedio: costo en un caso basado en ciertas hipótesis y probabilidades

Peor Caso:

$T_w(n) = \max \{ T(E) / E \in D_n \}$ // costo en el peor caso
$\{E / T(E) = T_w(n), E \in D_n\}$ // conjunto de entradas que son peor caso

Mejor Caso:

$T_B(n) = \min \{ T(E) / E \in D_n \}$ // costo en el mejor caso
$\{E / T(E) = T_B(n), E \in D_n\}$ // conjunto de entradas en el mejor caso

Caso Medio:

$T_A(n)$= promedio de TODAS las T(E), $E \in D_n$

$$T_A(n) = \sum_{E \in Dn} p(E) * T(E)$$ // costo en caso promedio

donde p(E) es la probabilidad que se de la entrada E, probabilidad EXPERIMENTAL

Propio caso medio:
$\{E / T(E) = T_A(n), E \in Dn\}$

Ejemplo: Find

Encontrar un elemento `x` dado, en una secuencia `A` dada, (puede considerarse que todos los elementos de la secuencia son distintos), si el elemento está devuelve la posición, sino devuelve el valor -1.

```
// a array of int [0...N-1]
// n cant elementos;

int find(int* A, int n, int x){
   int i = 0;
   while (i<n) && (A[i]!=x){
      i++
   }
   if (i<n)
      return i;
   else
      return -1;
}
```

Análisis:

Mejor caso:

Secuencia donde x esta en el 1er lugar
$T_B(n) = 1$

Peor caso:

Hay 2 configuraciones en las que se da el peor caso, estas son cuando:

- X no esta en A o
- X es el último elemento de A

$T_w(n) = n$

Caso medio:

Asumimos que si x ∈ A, todas las posiciones son equiprobables.

Se parte de y se debe obtener resultados para la expresión:

$$T_A(n) = \sum_{E \in Dn} p(E) * T(E)$$

esta expresión debe ser transformada para que resulte práctica.

Observación:
D_n es el conjunto de todas las secuencias de largo n.
En la práctica cabe notar que el algoritmo dado se comporta (ejecuta) de la misma forma para todas las secuencias que tengan a ***x*** en un lugar específico ***i***: No interesa el resto de los elementos que compongan la secuencia, si ***x*** está en el lugar i-ésimo se realizan i comparaciones y el algoritmo termina. Sucede algo similar si el elemento no está en la secuencia.

Caso 1: x ∈ A

En base a la observación anterior, es posible subdividir D_n en subconjuntos $D_{ni} \subset D_n$, donde D_{ni} está formado por las secuencias que tienen a ***x*** en la posición i-ésima, $1 \leq i \leq n$.
Resulta posible modificar el enfoque del problema y considerar que se tienen sólo ***n*** entradas $\boldsymbol{E}_i$, $1 \leq i \leq n$. Serán secuencias que sólo difieren en la posición del elemento ***x***.

En ese caso se tendrá:

- $p(E_i) = 1/n$ debido a la hipótesis de equiprobabilidad de ubicación de ***x*** en ***A***
- $T(E_i) = i$

$$Ta(n) = \sum_{i=1}^{n} p(Ei) * T(Ei) = \sum_{i=1}^{n} (1/n) * i = (1/n) * \sum_{i=1}^{n} i = (1/n) * n(n+1)/2$$

$$\Rightarrow T_A(n) = (n+1)/2$$

Caso 2: x puede no estar en A

Probabilidad de que **x ∈ A = q**

En forma análoga al caso anterior:

1) si $1 \leq i \leq n$ E_i representa las instancias $x \in A$ en el lugar i
2) E_{n+1} representa a las instancias de $\mathbf{x} \notin \mathbf{A}$

Se tendrá:

- $p(E_i) = 1/n$ si $1 \leq i \leq n$
- $p(E_{n+1}) = 1-q$
- $T(E_i) = i$ si $1 \leq i \leq n$, $T(E_{n+1}) = n$

$$T_A(n) = \sum_{i=1}^{n+1} p(Ei) * T(Ei) = (1-q) * n + \sum_{i=1}^{n} i * (q/n) = (1-q) * n + (q/n) * \sum_{i=1}^{n} i$$

$$T_A(n) = (q/n) * n * (n+1)/2 + (1-q) * n$$
$$T_A(n) = q * (n+1)/2 + (1-q) * n$$

notar que si ***q=1***, estamos en el caso 1 y el resultado es idéntico al obtenido antes; si ***q=0*** entonces $\mathbf{x} \notin \mathbf{A}$ y el costo es n como en el peor caso visto.

Comportamiento asintótico

Se denomina comportamiento asintótico cuando se analiza el tiempo/costo de ejecución T(n) en el caso $n \to \infty$

Interesa determinar la *tasa de crecimiento* de T(n).

Ejemplos: log(n), $n^3 + 3n^2 + 2n + 2$; 2^n; n!

Comparación de tasas de crecimiento

Sean A1 y A2 dos algoritmos con costos:
$T_{A1}(n) = 10^{-6}2^n$ $T_{A2}(n) = 10^{-6}n^3$

n	$T_{A1}(n)$	$T_{A2}(n)$
5	$3.2 * 10^{-5}$ s	$1.25 * 10^{-4}$ s
10	10^{-3} s	10^{-3} s
20	1s	$8* 10^{-3}$ s
30	18 min	$27 * 10^{-3}$ s
40	13 dias	$64 * 10^{-3}$s
45	4.1 años	0.09 s
100		1 s

Con un procesador 10^6 veces más rápido:

n	$T_{A1}(n)$
45	35 s
65	1.2 años

Noción Informal

Ejemplo: Sea un algoritmo con $T(n) = 20n^2 + 15n + 6$
y Sea $f(n) = n^2$

Se dirá:
T(n) es del ***orden*** de f(n) si T(n) es acotada por un múltiplo real positivo de f(n).

O sea: $T(n) \leq k.f(n)$ donde $k \in R^+$
Notar en el ejemplo:

$n^2 \geq n$ para $n \geq 1$
$n^2 \geq 1$ para $n \geq 1$

$T(n) = 20n^2 + 15n + 6$

$T(n) \leq 20n^2 + 15\, n^2 + 6\, n^2$

$T(n) \leq 41\, n^2$ para $n > 1$

Por lo tanto es posible decir que T(n) es de orden n^2.

Observar que:

- Funciones como n^3, $3n^2$, $27n^2$, n^4 también cumplen lo antedicho por lo tanto se tiene un conjunto de funciones "f" que "acotan" a T(n).
- Funciones g(n) como $3n^2$, $27n^2$, 5n, 4 son todas del orden de $f(n)=n^2$ porque siempre se puede encontrar un $k \in R^+$ de forma que se cumpla $g(n) \leq k.f(n)$. Por lo antedicho las funciones ***g*** que cumplan esta desigualdad forman un conjunto de funciones, cada una de las cuales será del orden de ***f(n).***

Notación Asintótica

Recordatorio:

N naturales.
R reales
R^+ reales positivos
$R^* = R^+ \cup \{0\}$

Definición: Orden: cota superior

$O(f) = \{ g: N \rightarrow R^* / \exists c \in R^+, \exists n_0 \in N, \forall n \in N, n > n_0 , g(n) \leq c*f(n)\}$

Si una función T(n) es una de las g que cumple la definición, se dirá que **$T(n) \in O(f)$**

Definición: Omega: cota inferior

$\Omega(f) = \{ g: N \rightarrow R^* / \exists c \in R^+, \exists n_0 \in N, \forall n \in N, n > n_0 , g(n) \geq c*f(n)\}$

análogamente a lo anterior se dirá **$T(n) \in \Omega(f)$**

Definición: Orden exacto.

$\Theta(f) = O(f) \cap \Omega(f)$

Notar:

O y Ω no son conceptos simétricos:
Sea $T_W(n)$ costo en el peor caso y T(n) el costo para una entrada cualquiera (genérica).
Se cumplirá que $T(n) \leq T_W(n)$ (por definición de costo en peor caso)
Si $T_W \in O(f) \Rightarrow \exists c, \exists n_0, T_W(n) \leq c*f(n)$ y por lo tanto f(n) acota TODA ejecución del algoritmo (para todas las entradas posibles)
Si $T_W \in \Omega(f) \Rightarrow \exists c, \exists n0, T_W(n) \geq c*f(n)$ se cumple para el peor caso pero no se puede afirmar nada respecto a un T(n) genérico.

Propiedades

1. $f \in O(g)$, $g \in O(h) \Rightarrow f \in O(h)$
2. $f \in O(g) \Leftrightarrow g \in \Omega(f)$
3. Si $f \in \Theta(g) \Rightarrow g \in \Theta(f)$
4. Θ define una relación de equivalencia
5. $O(f + g) = O(\max\{f,g\})$

Def:

$f > g \Leftrightarrow \exists n_0 \in N$ tal que $\forall n \in N$ que cumple $n > n_0$ entonces $f(n) > g(n)$

Demostraciones

Prop. 1

$f \in O(g) \Rightarrow \exists c_1 \in R^+, \exists n_1 \in N, \forall n \in N, n > n_1, f(n) \leq c_1*g(n)$
$g \in O(h) \Rightarrow \exists c_2 \in R^+, \exists n_2 \in N, \forall n \in N, n > n_2, g(n) \leq c_2*h(n)$

Se debe demostrar que:

$\exists c \in R^+, \exists n_0 \in N, \forall n \in N, n > n_0, f(n) \leq c*h(n)$

Con los valores vistos de c_1, c_2, n_1 y n_2 se sabe que se cumplen

$f(n) \leq c_1*g(n)$ $\quad\quad$ $g(n) \leq c_2*h(n) \Rightarrow$

$f(n) \leq c_1*g(n) \leq c_1*c_2*h(n)$ a partir de un valor de n

Entonces eligiendo:
$n_0 \in N$, $n_0 = \max(n_1, n_2)$ y
$c \in R^+$, $c=c_1*c_2$

Prop. 3

Si $f \in \Theta(g) \Rightarrow g \in \Theta(f)$

Como $f \in \Theta(g) \Rightarrow f \in O(g)$ y $f \in \Omega(g)$ por definición de Θ

Por propiedad 2,
si $f \in O(g) \Rightarrow g \in \Omega(f)$ y
si $f \in \Omega(g) \Rightarrow g \in O(f)$
Por lo tanto $g \in \Theta(f)$ por definición de Θ