uao

03

ESTRUCTURA DE DATOS Y ALGORITMOS 1

Análisis de Algoritmos

Profesor
Orlando Arboleda Molina

uao

Análisis Computacional

- Los computadores son **rápidos** y la **memoria** es **barata** pero no gratis, sin embargo, es **necesario** que los **algoritmos** sean **eficientes** en términos del **tiempo** y el **espacio** requerido.

- Los **algoritmos** para resolver el mismo problema a menudo **difieren** dramáticamente en su **eficiencia**. Estas **diferencias** pueden ser mucho **más significativas** que las diferencias debidas al **hardware** y **software**.

Ejemplo: determinar el tiempo de ejecución de un par de algoritmos que realizan la misma labor, para una entrada n=106 datos, en las maquinas X y Y.

Maquina X	Maquina Y
Ejecuta 10^9 IPS (instrucciones/seg) Algoritmo con $2n^2$ instrucciones **Ojo: 100 veces mas rápida que la Maquina Y**	Ejecuta 10^7 IPS (instrucciones/seg) Algoritmo con $50n\log_2 n$ instrucciones

Análisis Computacional

- Si n=106 datos.

Maquina X	Maquina Y
Ejecuta 10^9 IPS (instrucciones/seg) Algoritmo con $2n^2$ instrucciones **Total instrucciones del algoritmo:** $2(10^6)^2$ instrucciones = $2(10^{12})$ instrucciones Tiempo: $t = \frac{2(10^{12}) instruciones * 1seg}{(10^9) instruciones} = 2(10^3)\ seg$ $t = 2000seg$ $t = 33.33\ minutos$	Ejecuta 10^7 IPS (instrucciones/seg) Algoritmo con $50n\log_2 n$ instrucciones **Total instrucciones del algoritmo:** $50(10^6)\log_2(10^6)$ instrucciones=$5(10^7)\log_2(10^6)$ instrucciones Tiempo: $t = \frac{5(10^7)\log_2(10^6) instruciones * 1seg}{(10^7) instruciones} = 5\log_2(10^6)\quad seg$ $t = 5(19.93156..)\ seg$ $t = 1.6\ minutos$

Análisis Computacional

- El **tiempo** tomado por un algoritmo **crece** con el **tamaño de la entrada**.

- Es tradicional **describir** el **tiempo de ejecución** de un programa como una. **función del tamaño de su entrada**

- El **tiempo de ejecución** de un **algoritmo** sobre una entrada particular **depende del numero de operaciones primitivas** o **pasos ejecutados** (se requiere un tiempo constante ci para ejecutar cada línea).

- Se considerará **solo** el **peor caso**. Este caso establece el **tiempo máximo de computo**.

Tiempo de Ejecución

Ejemplo: determinar la función del tiempo de ejecución del siguiente fragmento (tener en cuenta que d y e se ejecuta una vez menos que c)

Nota: las instrucciones al interior del ciclo, no alteran el numero de ejecuciones

```
        a (1 vez)
(1)   f=1;  b(1 vez)  c        d
(2)   for (j=2 ; j<= n ; j++)
(3)       f*=j;  e
```

j	veces
2	1
3	2
.	.
n	n-1
n+1	n

$$T(n) = T_a(n) + T_b(n) + T_c(n) + T_d(n) + T_e(n)$$

$$T(n) = c_1*1 + c_2*1 + c_3*n + c_4*(n-1) + c_5*(n-1)$$

$$T(n) = (c_3+c_4+c_5)n+(c_1+c_2-c_4-c_5) = a*n+b$$

Instrucciones Ejecutadas

- Para **simplificar** nuestros **cálculos**, definiremos la función del **numero total de instrucciones ejecutadas** (donde la ejecución de cada instrucción es de tiempo 1).

Ejemplo: computo de la función del numero total de instrucciones ejecutadas para el ejemplo anterior y su relación con el tiempo de ejecución.

```
          a (1 vez)
(1)   f=1;   b(1 vez)  c       d
(2)   for (j=2 ; j<= n ; j++)
(3)       f*=j;  e
```

$$T(n) = c_1*1 + c_2*1 + c_3*n + c_4*(n-1) + c_5*(n-1)$$

$$T_{instrucciones}(n) = 1 + 1 + n + (n-1) + (n-1)$$

$$T_{instrucciones}(n) = 3n$$

Instrucciones Ejecutadas

Ejemplo: determinar el numero total de instrucciones ejecutadas de los siguientes

```
(1) k=0;
(2) s=0;
(3) for ( j=0; j < n-1; j ++ ) {
(4)       if ( A[ j ] < A[ j + 1 ] )
          {
(5)         temp = A[ j ];
(6)            A[  j ] = A[ j + 1 ];
(7)            A[ j + 1 ] = temp;
            }
          else
(8)          k ++;
(9)       s+=A[ j ];
     }
```

j	veces
0	1
1	2
.	.
n-2	n-1
n-1	

```
(1) j = b;
(2) for( i = 1; i <= n + 2; i ++ ) {
(3)       if( j > i ) {
(4)          div = 1;
(5)          break;
          }
(6)       j *= i;
(7)       div ++ ;
    }
```

Instrucciones Ejecutadas

Ejemplo: determinar el numero total de veces que se ejecuta la instrucción indicada

```
(1) for (i=0 ; i<=n-1 ; i++){
(2)    men=i;
(3)    for (j=i+1 ; j < n ;j++)
(4)        if (A[j] < A [men])
(5)            men=j;
(6)    temp=A[men];
(7)    A[men]=A[i];
(8)    A[i]=temp;
    }
```

Reto: determinar el numero total de instrucciones ejecutadas en todo el fragmento

Instrucciones Ejecutadas (con métodos)

- Se debe tener en cuenta la **invocación** e **instrucciones ejecutadas en el método**. La **creación del contexto** y las **variables locales** se realiza en **una sola instrucción**.

Ejemplo1: se indica $T_{instrucciones}(n)$ para la siguiente línea, en la que hay una invocación, asignación y método

```
r1 = f1( 0, n );
```

$T(n) = 1 + 1 + T_{instrucciones\_f1}(n) = 2 + T_{instrucciones\_f1}(n)$

Ejemplo2: determinar $T_{instrucciones}(n)$ del siguiente programa

```
    principal() {
      int r1, n;
(1)   leer( n );
(2)   r1 = f1( 0, n );
(3)   escribir( f2( r1, n ) );
    }
```

```
    int f1( int x, int n ) {
      int i;
(4)   for( i = 1; i <= n; i ++ )
(5)         x += f2( i, n );
(6)   return x;
    }
```

```
    int f2( int x, int n ) {
      int i;
(7)   for( i = 1; i <= n; i ++ )
(8)         x += i;
(9)   return x;
    }
```

$T(n) = 1+1+(2+T_{f1}(n))+(2+T_{f2}(n))$

$T(n) = 6+T_{f1}(n)+T_{f2}(n)$

Orden de Complejidad

- El **orden de crecimiento describe** como el **tiempo de ejecución** o el **consumo de recursos** (como la memoria) de un algoritmo **aumenta en función del tamaño de la entrada.**

- Tener en cuenta los siguientes **ordenes de crecimiento** de los **tiempos de ejecución** (o de la función del numero total de instrucciones ejecutadas).

Function	Name	
c	Constant	menor tiempo de ejecución
$\log N$	Logarithmic	
$\log^2 N$	Log-squared	
N	Linear	
$N \log N$		
N^2	Quadratic	
N^3	Cubic	
2^N	Exponential	mayor tiempo de ejecución

- En los ordenes de crecimiento se **ignoran** los **términos de orden mas bajo** (entre mas grande sea el valor de n mas se percibe la diferencia).

Ejemplo: las siguientes funciones del numero total de instrucciones ejecutadas son de orden n2 (cuadráticos)

$T_{instrucciones1}(n) = 10n^2$
$T_{instrucciones2}(n) = 0.01n^2+100n$

$T_{instrucciones3}(n) = n^2+100n+5000\log_2 n$
$T_{instrucciones4}(n) = n^2+1000n+500000$

Orden de Complejidad

- Los **ordenes de crecimiento** del tiempo de ejecución de un algoritmo da una **caracterización** de la **eficiencia del algoritmo** y **permite comparar** el **desempeño** relativo de **algoritmos alternos**.

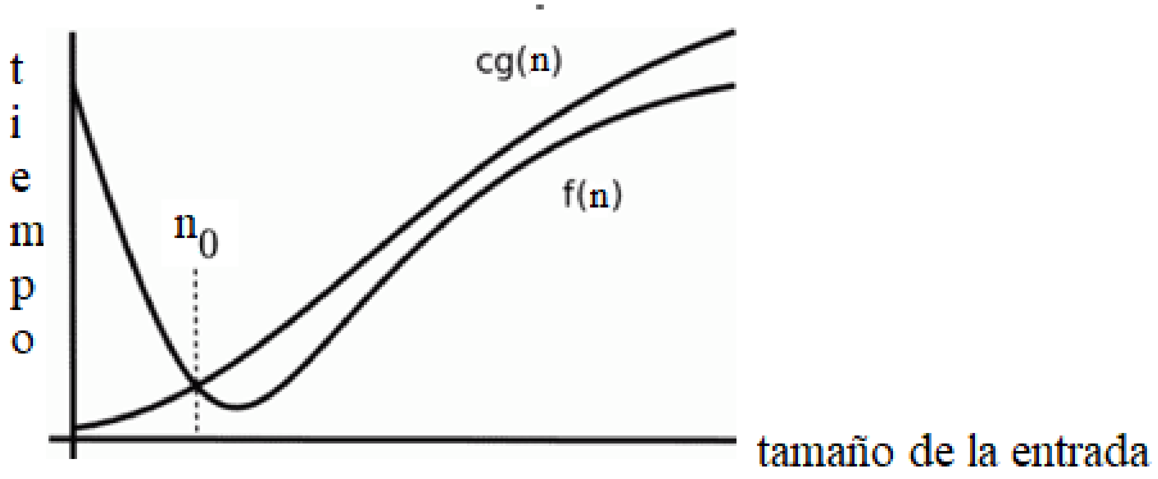
- Con el **orden de crecimiento** del tiempo de ejecución de un algoritmo se **estudia** su **eficiencia asintótica** (medida teórica que **describe** su **comportamiento** cuando el **tamaño de la entrada tiende a ser muy grande**).

- Existen las siguientes notaciones asintóticas:
  - Notación Ω (notación omega), para el orden inferior (mejor caso)
  - Notación Θ (notación theta), para el orden exacto (caso medio)
  - Notación O (notación O grande), para el orden superior (peor caso)

Notación O-Grande

- La **notación *O*** (**Big O**) se corresponde a la **cota superior** de un algoritmo.

- T(n) es O(g(n)), Léase “T(n) es de orden O(g(n))”, si:



$$O(g(n)) = \left\{ \begin{array}{l} f(n) : \text{existen } n_0, c > 0 \text{ tales que} \\ \forall n \geq n_0 > 0 : 0 \leq |f(n)| \leq c|g(n)| \end{array} \right\}$$

Ejemplo: si para un algoritmo $T_{\text{instrucciones}}$ (n) = $3n^2+11n+13$. Este $T_{\text{instrucciones}}$ (n) = $O(n^2)$, por que es posible encontrar un n_0 y c, que cumplan la definición.

Notación O-Grande

notación	nombre
$O(1)$	orden constante
$O(\log \log n)$	orden sublogarítmica
$O(\log n)$	orden logarítmica
$O(\sqrt{n})$	orden sublineal
$O(n)$	orden lineal o de primer orden
$O(n \cdot \log n)$	orden lineal logarítmica
$O(n^2)$	orden cuadrática o de segundo orden
$O(n^3)$, ...	orden cúbica o de tercer orden, ...
$O(n^c)$	orden potencial fija
$O(c^n)$, $n > 1$	orden exponencial
$O(n!)$	orden factorial
$O(n^n)$	orden potencial exponencial

- Complejidad +

\+ Eficiencia -

Notación O-Grande

Ejercicios: determinar los orden O para las siguiente funciones del numero total de instrucciones ejecutadas

$T_1(n) = 1000$

$T_2(n) = 2000+\log_5 n$

$T_3(n) = 0.05n^3+10n^2+3000n$

$T_4(n) = 10n^2+0.1(2^n)+800n$

$T_5(n) = 4n\log_5 n+80n$

$T_6(n) = 200n+4n^2+0.1n^{2.5}$

Propiedades Notación O-Grande

- Si $f_1=O(g_1)$ y $f_2=O(g_2)$, entonces $f_1+f_2=O(\ |g_1|+|g_2|\ )$

Ejemplo: calculo de la complejidad del siguiente fragmento

{ **A(n)** **B(n)** **}**	$T(n)=T_A(n) + T_B(n)$ (es la suma de $T_A(n)$ y $T_B(n)$) Suponiendo que $T_A(n)=O(n)$ y $T_B(n)=O(n^2)$ Entonces: $T(n)=T_A(n) + T_B(n) = O(n+n^2) = O(\ n^2\ )$

Ejemplo2: calculo de la complejidad de la línea en que se indica el ciclo *for*

for (let var=1; var<n; var++){
// instrucciones
}
	$T(n) = T_{inicio}(n)+T_{condición}(n)+T_{incremento}(n)$
	
	Dado que $T_{inicio}(n)=O(1)$ y $T_{condición}(n)=T_{incremento}(n)=O(n)$
	Entonces:
	$T(n) = T_{inicio}(n)+T_{condición}(n)+T_{incremento}(n) = O(1+n+n)= O(\ n\ )$

Propiedades Notación O-Grande

- Si $f_1=O(g_1)$ y $k>0$, entonces $k*f_1=O(g_1)$

Ejemplo: calculo de la complejidad del siguiente fragmento

{ **A(n)** **A(n)** **A(n)** **}**	$T(n) = 3*T_A(n)$ (3 veces el tiempo de $T_A(n)$) Suponiendo que $T_A(n) = O(n)$ Entonces: $T(n) = 3*T_A(n) = 3*O(n) = O(3*n) = O(n)$

Ejemplo2: calculo de la complejidad por todas las invocaciones del método al interior del ciclo

for (let var=1; var<=5; var++){ **A(n)** }	$T(n) = 5*T_A(n)$ (5 veces el tiempo de $T_A(n)$) Suponiendo que $T_A(n) = O(n^2)$ Entonces: $T(n) = 5*T_A(n) = 5*O(n^2) = O(5*n^2) = O(n^2)$

Propiedades Notación O-Grande

- Si $f_1=O(g_1)$ y $g_1=O(g_2)$, entonces $f_1=O(g_2)$

Ejemplo: calculo de la complejidad del método A, a partir de la complejidad de B

A(n){ **B(n)** **}**	$T(n) = T_B(n)$ (es la complejidad interna) Suponiendo que $T_B(n) = O(n^2)$ Entonces: $T_A(n) = O(T_B(n)) = O(O(n^2)) = O(n^2)$

- Si $f_1=O(g_1)$ y $f_2=O(g_2)$, entonces $f_1 * f_2 =O(g_1*g_2)$

Ejemplo: calculo de la complejidad de las instrucciones al interior del ciclo

Ciclo (condición){ **instrucciones** **}**	$T_{instrucciones}(n) = T_{ciclo}(n) * T_{individual\_instrucciones}(n)$ (esta al interior) Suponiendo que $T_{ciclo}(n)=O(\log n)$ y $T_{individual\_instrucciones}(n)=O(n)$ Entonces: $T_{instrucciones}(n) = T_{ciclo}(n) * T_{individual\_instrucciones}(n) = O((\log n) * n) = O(n*\log n)$

Complejidad de Algoritmos

Ejercicios: determinar la notación O en el peor caso de los siguientes fragmentos y los ejercicios propuestos en el taller del caso.

```
(1) k=0;
(2) s=0;
(3) for ( j=0; j < n-1; j ++ ) {
(4)      if ( A[ j ] < A[ j + 1 ] )
         {
(5)       temp = A[ j ];
(6)            A[  j ] = A[ j + 1 ];
(7)            A[ j + 1 ] = temp;
         }
         else
(8)         k ++;
(9)      s+=A[ j ];
     }
```

```
(1) for( i = 0; i < n - 1; i ++ ) {
(2)    men = i;
(3)    for( j = i + 1; j < n; j ++ )
(4)        if( A[j] < A[men] )
(5)            men = j;
(6)    temp = A[men];
(7)    A[men] = A[i];
(8)    A[i] = temp;
     }
```

```
(1) limite = Math.sqrt(n);
(2) for( i = 0; i < limite; i ++ ) {
(3)    res = 0;
(4)    for( j = 0; j < n; j ++ )
(5)        res += A[j];
(6)    A[i] = res;
     }
```

03

BUEN VIENTO Y BUENA MAR !!!

uao

uao