Tema 01:
Fundamentos del Análisis Asintótico de Algoritmos

Noviembre, 2003

CS0218: Algoritmos y Programación II

Introducción

En Ciencias de la Computación se presenta con frecuencia la situación de analizar dos o más algoritmos que resuelven el mismo problema para determinar cuál de ellos requiere menos recursos. Los enfoques para resolver este problema son los siguientes:

- *Benchmarking*
- Análisis Matemático de Algoritmos
- Análisis Asintótico de Algoritmos

Introducción

El análisis de algoritmos tiene los siguientes objetivos:

- Mejorar (si fuese posible) las características estructurales de los algoritmos
- Facilitar el proceso de selección de un algoritmo (entre varios disponibles) para un problema
- Predecir la cantidad de recursos requeridos por un algoritmo para la resolución de un problema

Introducción

Los principales criterios para realizar el análisis de algoritmos son los siguientes:

- Correctitud
- Cantidad de trabajo realizado
- Cantidad de espacio utilizado
- Simplicidad
- "Optimalidad"

Introducción

Énfasis en la cantidad de trabajo realizado por un algoritmo:

- Permite determinar la eficiencia del método utilizado por un algoritmo para la resolución de un problema

- Permite comparar dos o más algoritmos en términos de eficiencia, por lo que debe facilitar el proceso de selección de un algoritmo (entre varios disponibles) para la resolución de un problema

- Es independiente del computador, del lenguaje de programación, de las habilidades del programador y de los detalles de implementación

Benchmarking

Consiste en implementar los algoritmos en cuestión, ejecutarlos y determinar cuál de ellos requiere menos recursos. Esta solución parece ser la más sencilla e intuitiva, sin embargo tiene varios inconvenientes:

- Pueden existir muchos algoritmos para resolver un mismo problema, y resulta muy costoso y tedioso implementarlos todos para poder llevar a cabo la comparación.

- Es una técnica muy específica, ya que determina el desempeño de un programa particular dependiendo de varios factores: el computador, el lenguaje de programación, el compilador, las habilidades del programador y el conjunto de datos de entrada. A partir de un *benchmark* es difícil predecir el comportamiento de un programa en otro ambiente.

Análisis Matemático de Algoritmos

Consiste en tratar de asociar con cada algoritmo una función matemática exacta que mida su "eficiencia" dependiendo sólo de los siguientes factores:

- Las características estructurales del algoritmo.
- El tamaño del conjunto de datos de entrada: El número de datos con los cuales trabaja el algoritmo. Esta medida se interpreta según el tipo de algoritmo sobre el cual se esté trabajando.

Se define la función $T_A(n)$ como la cantidad de trabajo realizado por el algoritmo A para procesar una entrada de tamaño n y producir una solución al problema.

Análisis Matemático de Algoritmos

El análisis matemático de algoritmos es independiente de la implementación, sin embargo tiene varios inconvenientes:

- La imposibilidad de determinar, para muchos problemas y cada una de las posibles entradas, la cantidad de trabajo realizado
- La dificultad para determinar $T_A(n)$ exactamente.

Análisis Asintótico de Algoritmos

Técnica derivada del análisis matemático de algoritmos basada en dos conceptos fundamentales: la caracterización de datos de entrada y la complejidad asintótica.

Peor caso: Los casos de datos de entrada que maximizan la cantidad de trabajo realizado por un algoritmo.

Mejor caso: Los casos de datos de entrada que minimizan la cantidad de trabajo realizado por un algoritmo.

Caso promedio: El valor medio de la cantidad de trabajo realizado por un algoritmo. Se debe tener en cuenta la distribución probabilística de los datos de entrada que se manejan.

Análisis Asintótico de Algoritmos

Definición: Sea D_n el conjunto de datos de entrada de tamaño n para un algoritmo, y sea $I \in D_n$ un elemento cualquiera. Sea $t(I)$ la cantidad de trabajo realizado por el algoritmo para procesar la entrada I. Se definen entonces las siguientes funciones:

Complejidad del peor caso: $W(n) = max\{t(I) : I \in D_n\}$

Complejidad del mejor caso: $B(n) = min\{t(I) : I \in D_n\}$

Complejidad del caso promedio: $A(n) = \sum_{I \in D_n} Pr(I) * t(I)$, donde $Pr(I)$ es la probabilidad de que ocurra la entrada I

Análisis Asintótico de Algoritmos

- Énfasis en el peor caso, ya que representa una cota superior para la cantidad de trabajo realizado por un algoritmo
- La idea fundamental consiste en tratar de encontrar una función $W(n)$, fácil de calcular y conocida, que acote asintóticamente el orden de crecimiento de la función $T_A(n)$
- Se estudia la eficiencia asintótica de algoritmos: Cómo se incrementa la cantidad de trabajo realizado por un algoritmo a medida que se incrementa (con valores "suficientemente grandes") el tamaño de la entrada
- Para realizar este procedimiento se necesitan herramientas especiales: las notaciones asintóticas

La Notación O-grande

Definición: Sean $f, g : \mathbb{N} \rightarrow \mathbb{R}^*$ dos funciones arbitrarias de los números naturales en los números reales no negativos. Se dice entonces que f está en O-grande de g, y se escribe $f \in O(g)$, si y sólo si existe una constante real positiva M y un número natural n_0 tales que para todo número natural $n \geq n_0$ se tiene que $f(n) \leq M * g(n)$. Simbólicamente:

$$f \in O(g) \Leftrightarrow$$

$$\exists M \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} : \forall n \in \mathbb{N} : [n \geq n_0 \Rightarrow f(n) \leq M * g(n)]$$

La Notación O-grande

Definición: Sea $g : \mathbb{N} \rightarrow \mathbb{R}^*$ una función arbitraria de los números naturales en los números reales no negativos. $O(g)$ representa el conjunto de todas las funciones $t : \mathbb{N} \rightarrow \mathbb{R}^*$ tales que existe una constante real positiva M y un número natural n_0 de manera tal que para todo número natural $n \geq n_0$ se tiene que $t(n) \leq M * g(n)$. Simbólicamente:

$$O(g) = \{t : \mathbb{N} \rightarrow \mathbb{R}^* | \exists M \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} : \\ \forall n \in \mathbb{N} : [n \geq n_0 \Rightarrow f(n) \leq M * g(n)]$$

La Notación O-grande

Observaciones:

- Puede ocurrir que $f \in O(g)$ aún cuando $\forall n \in \mathbb{N} : f(n) > g(n)$
- De la definición se tiene que algún múltiplo constante de g es una cota superior asintótica de f (para valores "suficientemente grandes" de n)
- No se considera la relación entre f y g para valores "pequeños" de n

La Notación O-grande

Ejemplo: Considere la función $f(n) = 8n + 128$, y suponga que se quiere mostrar que $f(n) \in O(n^2)$. Según la definición, se debe encontrar una constante real positiva M y un número natural n_0 tales que para todo número natural $n \geq n_0$ se verifique que $f(n) \leq M * n^2$. Suponga que se selecciona $M = 1$. Se tiene entonces que:

$$\begin{aligned} f(n) \leq M * n^2 &\Leftrightarrow 8n + 128 \leq n^2 \\ &\Leftrightarrow 0 \leq n^2 - 8n - 128 \\ &\Leftrightarrow 0 \leq (n - 16) * (n + 8) \end{aligned}$$

Como $\forall n \in \mathbb{N} : n + 8 > 0$, se concluye que $n - 16 \geq 0$. Por lo tanto, $n_0 = 16$. Así que, para $M = 1$ y $n_0 = 16$, $f(n) \leq M * n^2$. Luego, $f(n) \in O(n^2)$.

La Notación O-grande

Ejemplo:

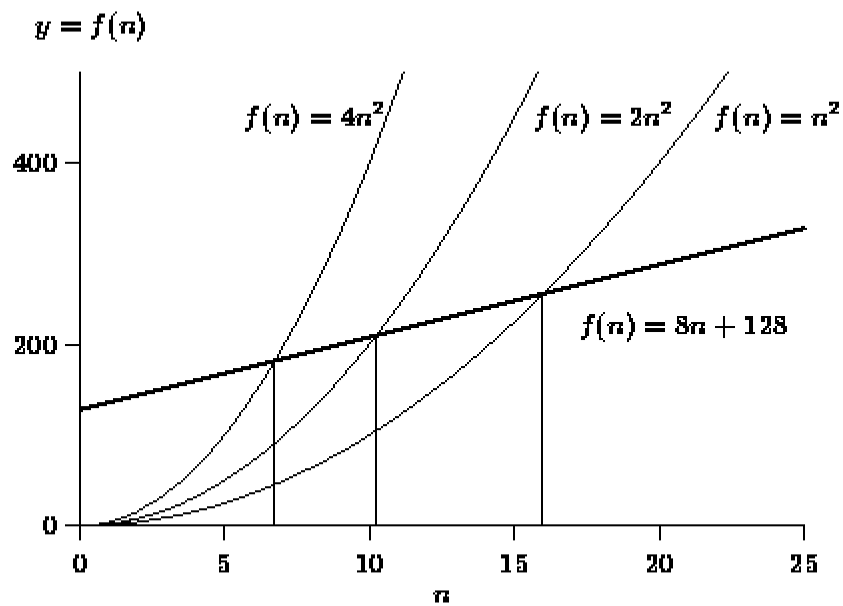

Figura 1: Orden de crecimiento de $f(n) = 8n + 128$

La Notación O-grande

Regla del límite: Sean $f, g : \mathbb{N} \rightarrow \mathbb{R}^*$ dos funciones arbitrarias de los números naturales en los números reales no negativos, y sea $L = lim_{n \rightarrow \infty}[f(n)/g(n)]$. Se tiene entonces que:

- Si $L \in \mathbb{R}^+$ entonces $f \in O(g)$ y $g \in O(f)$
- Si $L = 0$ entonces $f \in O(g)$ y $g \notin O(f)$
- Si $L \rightarrow \infty$ entonces $f \notin O(g)$ y $g \in O(f)$

La Notación O-grande

Definición: Sean $f, g : \mathbb{N} \rightarrow \mathbb{R}^*$ funciones arbitrarias de los números naturales en los números reales no negativos, y suponga que $f \in O(g)$. Se dice que g es una cota superior "ajustada" para f si y sólo si, para toda función $h : \mathbb{N} \rightarrow \mathbb{R}^*$, se verifica que:

$$f \in O(h) \Rightarrow g \in O(h)$$

De manera informal, g es una cota superior "ajustada" para f si no existe una función h que se "encuentre" en el *gap* existente entre f y g.

La Notación O-grande

Proposición: Sean $f, g, h : \mathbb{N} \rightarrow \mathbb{R}^*$ funciones arbitrarias de los números naturales en los números reales no negativos. La notación O-grande verifica las siguientes propiedades:

- Reflexividad: $f \in O(f)$
- Transitividad: $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$

La Notación O-grande

Propiedades: Sean $f, g : \mathbb{N} \rightarrow \mathbb{R}^*$ dos funciones arbitrarias de los números naturales en los números reales no negativos. La notación O-grande verifica las siguientes propiedades:

- $\lambda * O(f) = O(f)$
- Si $\lambda > 0$ entonces $O(\lambda * f) = O(f)$
- $O(f) + O(g) = O(max(f, g))$ (Regla de la suma)
- $O(f) * O(g) = O(f * g)$ (Regla del producto)

La Notación O-grande

Orden	**Nombre**
$O(1)$	Constante
$O(log(n))$	Logarítmico
$O(n)$	Lineal
$O(nlog(n))$	nlog(n)
$O(n^2)$	Cuadrático
$O(n^3)$	Cúbico
$O(n^k)$, $k > 3$	Polinomial
$O(2^n)$	Exponencial

Tabla 1: Órdenes de crecimiento comunes

Cálculo del tiempo de ejecución de un algoritmo

Declaraciones: Las declaraciones de constantes, tipos, variables, procedimientos y funciones no se toman en cuenta en el análisis asintótico de un algoritmo.

Operaciones elementales: El tiempo de ejecución de una operación elemental puede tomarse como O(1).

Secuencias de instrucciones: El tiempo de ejecución de una secuencia de instrucciones se determina utilizando la regla de la suma

Cálculo del tiempo de ejecución de un algoritmo

Condicionales: El tiempo de ejecución de una proposición condicional:

```
if cond then
   S1
else
   S2
endif
```

es el tiempo necesario para evaluar la condición (que generalmente es $O(1)$) más el máximo entre el tiempo de ejecución de la secuencia de proposiciones cuando la condición es verdadera y el tiempo de ejecución de la secuencia de proposiciones cuando la condición es falsa.

Cálculo del tiempo de ejecución de un algoritmo

Ciclos: El tiempo de ejecución de un ciclo:

```
while cond do
{
   S
}

for k in [min .. max] do
{
   S
}
```

es la suma, sobre todas las iteraciones del ciclo, del tiempo de ejecución de las instrucciones que están en el interior del ciclo y del tiempo utilizado para evaluar la condición de terminación (que generalmente puede tomarse como $O(1)$)