CAPÍTULO 3: NOTACIÓN ASINTÓTICA

3.1. Introducción

Un aspecto importante de este libro es el que concierne a la determinación de la eficiencia de algoritmos. En la Sección 2.3, veíamos que este conocimiento puede ayudarnos, por ejemplo, a elegir uno entre varios algoritmos en pugna. Recordemos que deseamos determinar matemáticamente la cantidad de recursos que necesita el algoritmo como función del tamaño (o a veces del valor) de los casos considerados. Dado que no existe una computadora estándar con la cual se puedan comparar todas las medidas del tiempo de ejecución, vimos también en la Sección 2.3 que nos contentaremos con expresar el tiempo requerido por el algoritmo salvo por una constante multiplicativa.

Con este objetivo, presentaremos ahora formalmente la notación asintótica que se utiliza a lo largo de todo el libro. Además, esta notación permite realizar simplificaciones sustanciales aun cuando estemos interesados en medir algo más tangible que el tiempo de ejecución, tal como el número de veces que se ejecuta una instrucción dada dentro de un programa.

Esta notación se denomina **asintótica** porque trata acerca del comportamiento de funciones en el límite, esto es, para valores suficientemente grandes de su parámetro. En consecuencia, los argumentos basados en la notación asintótica pueden no llegar a tener un valor práctico cuando el parámetro adopta valores «de la vida real». Sin embargo, las enseñanzas de la notación asintótica suelen tener una relevancia significativa. Esto se debe a que, como regla del pulgar, un algoritmo que sea superior asintóticamente suele ser (aunque no siempre) preferible incluso en casos de tamaño moderado.

Diversas notaciones asintóticas:

- Notación para “el orden de”
- Notación omega
- Notación theta
- **Notación asintótica condicional**

Hay muchos algoritmos que resultan más fáciles de analizar si en un principio limitamos nuestra atención a aquellos casos cuyo tamaño satisfaga una cierta condición, tal como ser una potencia de 2.

Considérese por ejemplo el algoritmo **divide y vencerás** para multiplicar enteros grandes. Sea n el tamaño de los enteros que hay que multiplicar (supongamos que eran de un mismo tamaño). El algoritmo se ejecuta directamente si n=1n = 1n=1, lo cual requiere a microsegundos para una constante apropiada a. Si n>1, el algoritmo se ejecuta recursivamente multiplicando cuatro parejas de enteros de tamaño [n/2].

Además, es precisa una cantidad lineal de tiempo para efectuar tareas adicionales. Por sencillez, digamos que el trabajo adicional requiere de n microsegundos para una constante adecuada b (para ser exactos, sería necesario un tiempo entre $b_1 n$ y $b_2 n$ microsegundos para las constantes adecuadas b_1 y b_2.

- **Notación asintótica con varios parámetros**

Puede suceder, cuando se analiza un algoritmo, que su tiempo de ejecución dependa simultáneamente de más de un parámetro del ejemplar en cuestión. Esta situación es típica de ciertos algoritmos para problemas de grafos, por ejemplo, en los cuales el tiempo depende tanto del número de nodos como del número de aristas. En tales casos, la noción de «tamaño del ejemplar» que se ha utilizado hasta el momento puede perder gran parte de su significado. Por esta razón, se generaliza la notación asintótica de forma natural para funciones de varias variables.

- **Operaciones sobre notación asintótica**

Para simplificar algunos cálculos, podemos manipular la notación asintótica usando operadores aritméticos. Por ejemplo, O(f(n))+O(g(n)) representa el conjunto de operaciones obtenidas al sumar, punto por punto, cualquier función de O(f(n)) con cualquier función de O(g(n)).

Intuitivamente, este conjunto representa el orden del tiempo requerido por un algoritmo compuesto por dos fases:

1. Una primera fase con tiempo del orden de f(n).
2. Una segunda fase con tiempo del orden de g(n).

Aunque las constantes ocultas que multiplican a f(n) y g(n) pueden ser diferentes, esto no afecta el resultado, ya que se puede demostrar que:

O(f(n))+O(g(n)) es idéntico a O(f(n) + g(n)).

Además, por la regla del máximo, esto equivale a: O(max(f(n),g(n))), o, si se prefiere: max(O(f(n)),O(g(n))).

En resumen, estas simplificaciones permiten analizar algoritmos de manera más eficiente sin perder rigor matemático.