Semana 5 control de lectura

Tema 3 – Notación Asintótica

¿Por qué usamos notación asintótica?

En algoritmos, no basta con que un programa funcione correctamente. Es necesario saber qué tan eficiente es, especialmente cuando el tamaño de la entrada crece mucho. Para esto, se utiliza la notación asintótica, que permite describir el comportamiento del algoritmo a gran escala, sin depender de detalles como el lenguaje de programación, la máquina o pequeñas constantes.

¿Qué mide la notación asintótica?

Principalmente dos aspectos:

1. Tiempo de ejecución: cuántas operaciones realiza un algoritmo.
2. Espacio de memoria: cuánta memoria adicional necesita.

En este tema se hace énfasis en el análisis del tiempo de ejecución, y en cómo este crece en función del tamaño de entrada nnn.

Tipos de notación asintótica

1. O grande – O(f(n))O(f(n))O(f(n))

Representa una cota superior. Indica que el algoritmo, como máximo, realiza una cantidad de trabajo proporcional a f(n)f(n)f(n). Es útil para analizar el peor caso.

Ejemplo: Si un algoritmo necesita a lo más 3n2+5n+23n^2 + 5n + 23n2+5n+2 operaciones, entonces:

T(n)=O(n2)T(n) = O(n^2)T(n)=O(n2)

2. Omega – Ω(f(n))\Omega(f(n))Ω(f(n))

Representa una cota inferior. Indica que el algoritmo, como mínimo, hace un trabajo proporcional a f(n)f(n)f(n). Se usa para analizar el mejor caso.

Ejemplo: Si un algoritmo tarda al menos nnn operaciones, entonces:

T(n)=Ω(n)T(n) = \Omega(n)T(n)=Ω(n)

3. Theta – Θ(f(n))\Theta(f(n))Θ(f(n))

Representa una cota ajustada. Indica que el algoritmo hace trabajo proporcional a f(n)f(n)f(n), tanto en el límite superior como en el inferior. Se utiliza cuando el peor y el mejor caso coinciden en orden de magnitud.

Ejemplo: Si el número de operaciones está acotado por arriba y por abajo por nnn, entonces:

T(n)=Θ(n)T(n) = \Theta(n)T(n)=Θ(n)

Comparación de funciones comunes

Brassard y Bratley presentan una jerarquía de funciones que aparecen frecuentemente en análisis de algoritmos:

1<log⁡n<n<nlog⁡n<n2<2n<n!1 < \log n < n < n \log n < n^2 < 2^n < n!1<logn<n<nlogn<n2<2n<n!

Estas funciones indican cómo aumenta el tiempo de ejecución según el tamaño de la entrada. Por ejemplo:

- O(1)O(1)O(1): constante, no depende de la entrada.
- O(log⁡n)O(\log n)O(logn): crece muy lentamente (como en la búsqueda binaria).
- O(n)O(n)O(n): crece linealmente con la entrada.
- O(nlog⁡n)O(n \log n)O(nlogn): aparece en algoritmos eficientes de ordenación.
- O(n2)O(n^2)O(n2): típico de algoritmos con bucles anidados.
- O(2n)O(2^n)O(2n) y O(n!)O(n!)O(n!): aparecen en problemas combinatorios, y no escalan bien.

¿Cómo se ignoran detalles?

En la notación asintótica:

- Se ignoran constantes multiplicativas, ya que no afectan el crecimiento a gran escala.
- Solo se considera el término dominante, el que más contribuye al crecimiento.

Ejemplo:

T(n)=5n2+100n+7⇒O(n2)T(n) = 5n^2 + 100n + 7 \Rightarrow O(n^2)T(n)=5n2+100n+7⇒O(n2)

Aplicación práctica

Los autores muestran cómo esta notación ayuda a analizar y comparar algoritmos como la búsqueda lineal, búsqueda binaria y algoritmos de ordenamiento. Con ella se puede prever cuál será más eficiente a medida que aumente el tamaño de entrada, sin tener que implementarlos o medirlos experimentalmente.