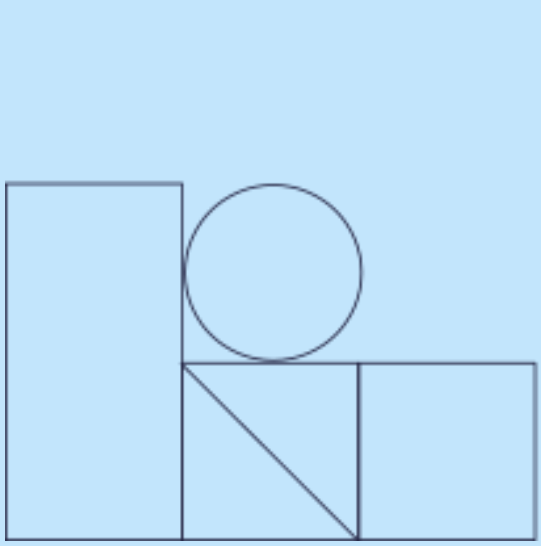
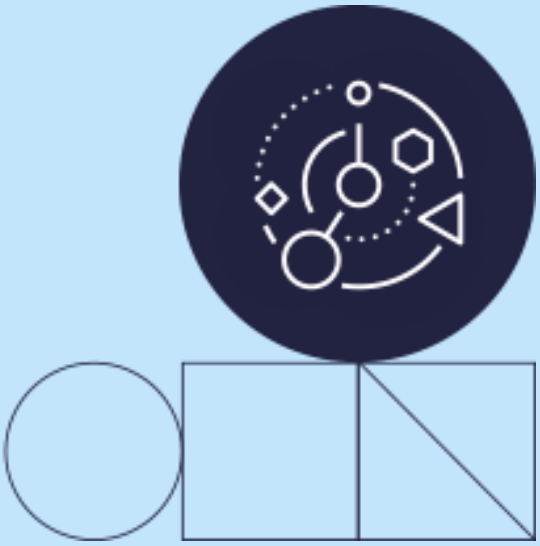
IBM SkillsBuild

IBM

Programando en Python

Análisis asintótico Notación Big-O





Índice

Introducción

Hemos visto en el tema anterior que La eficiencia de un algoritmo depende de la cantidad de tiempo, almacenamiento y otros recursos necesarios para ejecutar el algoritmo. La eficiencia se mide con la ayuda de notaciones asintóticas.

Un algoritmo puede no tener el mismo rendimiento para diferentes tipos de entradas. Con el aumento en el tamaño de entrada, el rendimiento cambiará.

El estudio del cambio en el rendimiento del algoritmo con el cambio en el orden del tamaño de entrada se define como **análisis asintótico**.

Notaciones asintóticas

La idea principal del análisis asintótico es tener una medida de la eficiencia de los algoritmos que no dependan de las constantes específicas de la máquina y no requieran la implementación de algoritmos ni el tiempo que toman los programas para compararlos. Las notaciones asintóticas son herramientas matemáticas para representar la complejidad temporal de los algoritmos para el análisis asintótico.

Hay principalmente tres notaciones asintóticas:

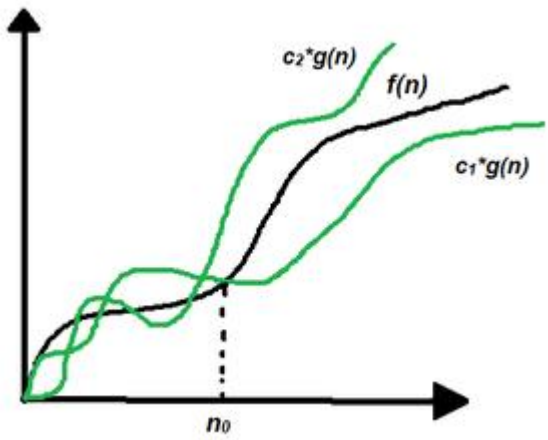
- Notación theta (notación Θ)
- Notación omega (notación Ω)
- Notación Big-O (notación O)

Notación Theta (Notación Θ) :

La notación theta encierra la función desde arriba y desde abajo. Dado que representa el límite superior e inferior del tiempo de ejecución de un algoritmo, se utiliza para analizar la complejidad del caso promedio de un algoritmo.

Sean **g** y **f** la función del conjunto de números naturales a sí mismo. Se dice que la función **f** es **Θ(g)**, si existen constantes **c1**, **c2 > 0** y un número natural **n0** tal que

c1* g(n) ≤ f(n) ≤ c2 * g(n) para todo **n ≥ n0**



Representación matemática de la notación Theta:

Θ (g(n)) = {f(n): existen constantes positivas **c1**, **c2** y **n0** tales que

0 ≤ c1 * g(n) ≤ f(n) ≤ c2 * g(n) para todo **n ≥ n0** }

Nota: **Θ(g)** es un conjunto

La expresión anterior se puede describir como si **f(n)** fuera theta de **g(n)**, entonces el valor **f(n)** siempre está entre **c1 * g(n) y c2 * g(n)** para valores grandes de **n (n ≥ n0)**. La definición de theta también requiere que **f(n)** no sea negativa para valores de **n** mayores que **n0**.

Una forma sencilla de obtener la notación Theta de una expresión es eliminar los términos de orden inferior e ignorar las constantes iniciales. Por ejemplo, considere la expresión **3n 3 + 6n 2 + 6000 = Θ(n 3)** , la eliminación de los términos de orden inferior siempre está bien porque siempre habrá un número (**n**) después del cual **Θ(n 3)** tiene valores más altos que **Θ (n 2)** independientemente de las constantes involucradas. Para una función **g(n)** dada, denotamos que **Θ(g(n))** es el siguiente conjunto de funciones.

Ejemplos:

{ 100 , log (2000) , 10^4 } pertenece a **Θ(1)**
{ (n/4) , (2n+3) , (n/100 + log(n)) } pertenece a **Θ(n)**
{ (n^2+n) , (2n^2) , (n^2+log(n))} pertenece a **Θ(n 2)**

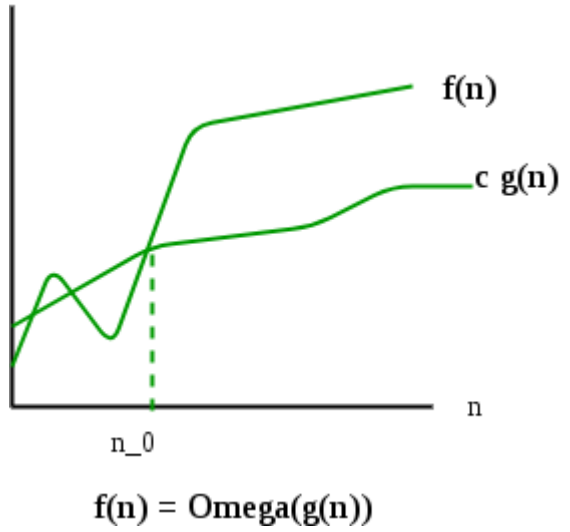
Nota: **Θ** proporciona límites exactos.

Notación Omega (Notación Ω)

La notación Omega representa el límite inferior del tiempo de ejecución de un algoritmo. Por lo tanto, proporciona la mejor complejidad de caso de un algoritmo.

Sean **g** y **f** la función del conjunto de números naturales a sí mismo. Se dice que la función **f** es **Ω(g)**, si existe una constante **c > 0** y un número natural **n0** tal que

c*g(n) ≤ f(n) para todo **n ≥ n0**



Representación matemática de la notación Omega

Ω(g(n)) = { f(n): existen constantes positivas **c** y **n0** tales que **0 ≤ cg(n) ≤ f(n)** para todo

n ≥ n0 }

Consideremos aquí el mismo ejemplo de clasificación por inserción. La complejidad temporal de la ordenación por inserción se puede escribir como **Ω(n)**, pero no es una información muy útil sobre la ordenación por inserción, ya que generalmente nos interesa el peor de los casos y, a veces, el caso promedio.

Ejemplos:

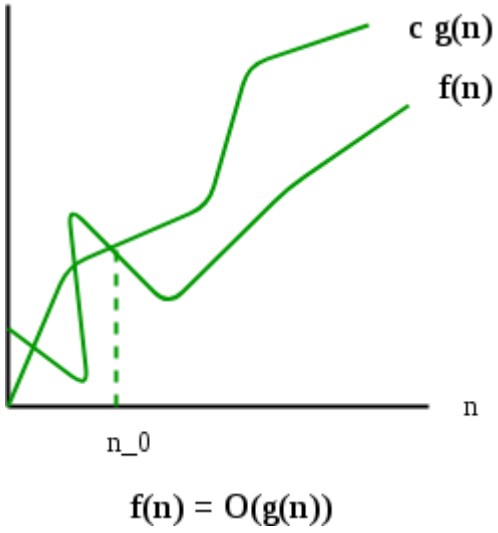
{ (n^2+n) , (2n^2) , (n^2+log(n))} pertenece a **Ω(n^2)**
U { (n/4) , (2n+3) , (n /100 + log(n)) } pertenece a **Ω(n)**
U { 100 , log (2000) , 10^4 } pertenece a **Ω(1)**

Nota: Aquí, **U** representa la unión, podemos escribirlo de esta manera porque **Ω** proporciona cotas exactas o inferiores.

Notación Big-O (notación O)

La notación **Big-O** representa el límite superior del tiempo de ejecución de un algoritmo. Por lo tanto, da la complejidad del peor de los casos de un algoritmo.

Si **f(n)** describe el tiempo de ejecución de un algoritmo, **f(n)** es **O(g(n))** si existe una constante positiva **C** y **n0** tal que, **0 ≤ f(n) ≤ cg(n)** para todo **n ≥ n0**



Representación matemática de la notación Big-O

O(g(n)) = { f(n): existen constantes positivas **c** y **n0** tales que **0 ≤ f(n) ≤ cg(n)** para todo

n ≥ n0 }

Por ejemplo, consideremos el caso de Ordenación por inserción. Toma tiempo lineal en el mejor de los casos y tiempo cuadrático en el peor de los casos. Podemos decir con seguridad que la complejidad temporal de la ordenación por inserción es **O(n 2)**.

Nota : **O(n 2)** también cubre el tiempo lineal.

Si usamos la notación **Θ** para representar la complejidad temporal del ordenamiento por inserción, tenemos que usar dos declaraciones para el mejor y el peor de los casos:

- La complejidad temporal en el peor de los casos de Ordenación por inserción es **Θ(n 2)**.
- La complejidad de tiempo del mejor caso de clasificación por inserción es **Θ(n)**.

La notación **Big-O** es útil cuando sólo tenemos un límite superior en la complejidad temporal de un algoritmo. Muchas veces encontramos fácilmente un límite superior simplemente mirando el algoritmo.

Ejemplos:

{ 100 , log (2000) , 10^4 } pertenece a **O(1)**
U { (n/4) , (2n+3) , (n/100 + log(n)) } pertenece a **O(n)**
U { (n^2+n) , (2n^2) , (n^2+log(n))} pertenece a **O(n^2)**

Nota: aquí, **U** representa unión, podemos escribirlo de esta manera porque **O** proporciona límites exactos o superiores.