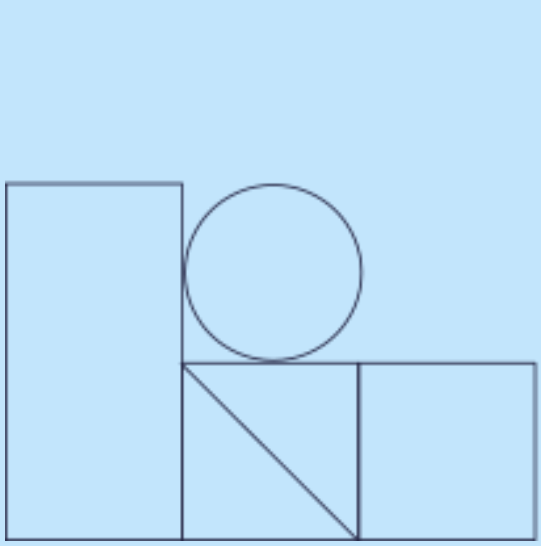
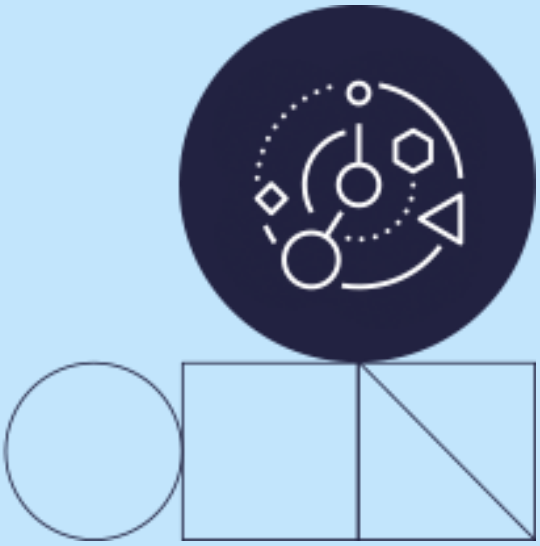
IBM SkillsBuild

IBM

Programação em Python

Análise assimptótica notação Big-O





Índice

Introdução

Vimos no tópico anterior que a eficiência de um algoritmo depende da quantidade de tempo, armazenamento e outros recursos necessários para executar o algoritmo. A eficiência é medida com a ajuda de notações assimptóticas.

Um algoritmo pode não ter o mesmo desempenho para diferentes tipos de insumos. Com o aumento do tamanho da entrada, o desempenho mudará.

O estudo da mudança no desempenho do algoritmo com a mudança na ordem de tamanho de entrada é definido como **análise assimptótica**.

Notações assimptóticas

A idéia principal da análise assimptótica é ter uma medida da eficiência dos algoritmos que não dependem de constantes específicas da máquina e não requerem a implementação de algoritmos ou o tempo que leva para que os programas os comparem. Notações assimptóticas são ferramentas matemáticas para representar a complexidade temporal dos algoritmos para análise assimptótica.

Há principalmente três notações assimptóticas:

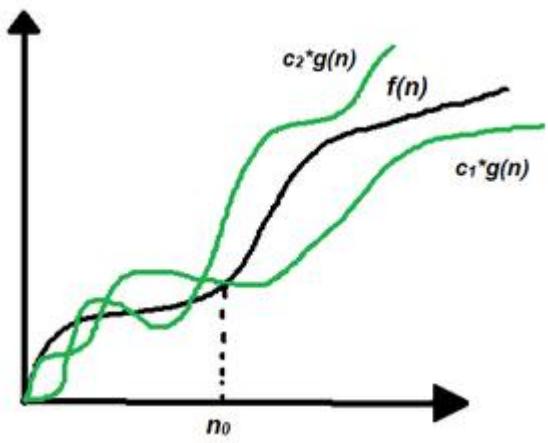
- Notação theta (notação Θ)
- Notação omega (notação Ω)
- Notação Big-O (notação O)

Notação Theta (Notação Θ) :

A notação theta encerra a função por cima e por baixo. Como representa o limite superior e inferior do tempo de execução de um algoritmo, ele é usado para analisar a complexidade do caso médio de um algoritmo.

Sejam **g** e **f** a função do conjunto de números naturais para si mesmo. Diz-se que a função **f** é **Θ(g)**, se houver constantes **c1**, **c2 > 0** e um número natural **n0** tal que

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



Representação matemática da notação Theta:

Θ (g(n)) = {f(n): há constantes positivas **c1**, **c2** e **n0** de tal forma que

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

Nota: **Θ(g)** é um conjunto

A expressão acima pode ser descrita como se **f(n)** fosse theta de **g(n)**, então o valor **f(n)** está sempre entre **c1 * g(n) y c2 * g(n)** para grandes valores de **n (n ≥ n0)**. A definição de theta também exige que **f(n)** seja não-negativo para valores de **n** superiores a **n0**.

Uma maneira simples de obter a notação Theta de uma expressão é eliminar os termos de ordem inferior e ignorar as constantes iniciais. Por exemplo, considere a expressão **3n 3 + 6n 2 + 6000 = Θ(n 3)** , a eliminação de termos de ordem inferior é sempre boa porque sempre haverá um número (**n**) após o qual **Θ(n 3)** tem valores superiores a **Θ (n 2)** independentemente das constantes envolvidas. Para uma função **g(n)** dado, denota-se que **Θ(g(n))** é o seguinte conjunto de funções.

Exemplos:

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

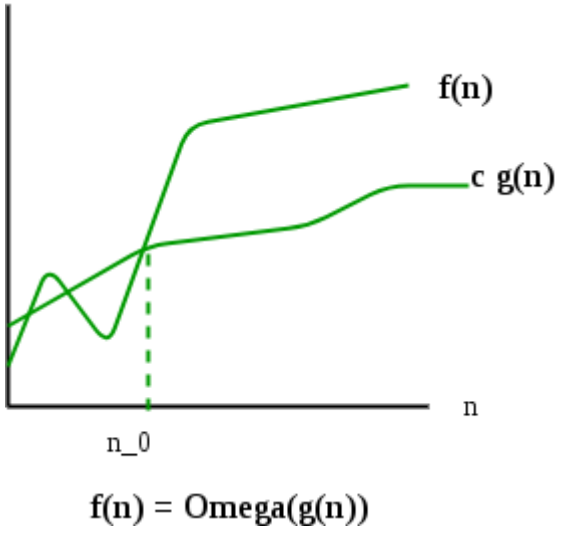
Nota: **Θ** fornece limites exatos.

Notação Omega (Notação Ω)

A notação Omega representa o limite inferior do tempo de execução de um algoritmo. Portanto, ele fornece a melhor complexidade de um algoritmo.

Sejam **g** e **f** a função do conjunto de números naturais para si mesmo. Diz-se que a função **f** é **Ω(g)**, se existir uma constante **c > 0** e um número natural **n0** tal que

c*g(n) ≤ f(n) para todo **n ≥ n0**



Representação matemática da notação Omega

Ω(g(n)) = { f(n): há constantes positivas **c** e **n0** de tal forma que **0 ≤ cg(n) ≤ f(n)** para todo

n ≥ n0 }

Considere aqui o mesmo exemplo de ordenação por inserção. A complexidade temporal do tipo de inserção pode ser escrita como **Ω(n)**, mas esta não é uma informação muito útil sobre o tipo de inserção, pois normalmente estamos interessados no pior caso e às vezes no caso médio.

Exemplos:

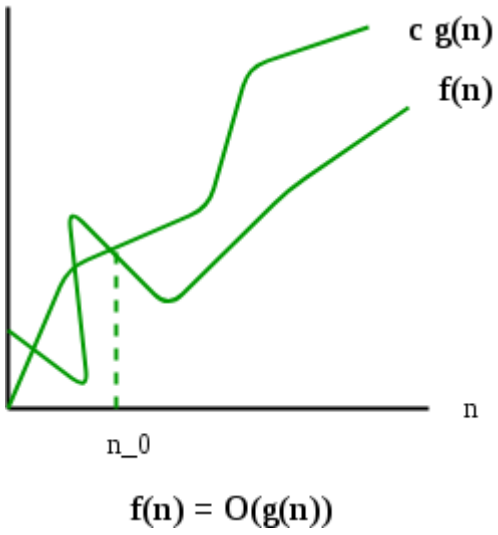
{ (n^2+n) , (2n^2) , (n^2+log(n))} pertence a **Ω(n^2)**
U { (n/4) , (2n+3) , (n /100 + log(n)) } pertence a **Ω(n)**
U { 100 , log (2000) , 10^4 } pertence a **Ω(1)**

Nota: Aqui, **U** representa o sindicato, podemos escrevê-lo desta forma porque **Ω** fornece dimensões exatas ou inferiores.

Notação Big-O (Notação O)

A notação **Big-O** representa o limite superior no tempo de execução de um algoritmo. Portanto, ele dá o pior cenário de complexidade de um algoritmo.

Se **f(n)** descreve o tempo de execução de um algoritmo, **f(n)** é **O(g(n))** se existe uma constante positiva **C** e **n0** tal que, **0 ≤ f(n) ≤ cg(n)** para todo **n ≥ n0**



Representação matemática da notação Big-O

O(g(n)) = { f(n): há constantes positivas **c** e **n0** de tal forma que **0 ≤ f(n) ≤ cg(n)** para todo

n ≥ n0 }

Por exemplo, considere o caso do Ordenação por inserção. Leva tempo linear na melhor das hipóteses e tempo quadrático na pior das hipóteses. Podemos dizer com segurança que a complexidade temporal do tipo de inserção é **O(n 2)**.

Nota : **O(n 2)** também cobre o tempo linear.

Se usarmos a notação **Θ** para representar a complexidade temporal do pedido de inserção, temos que usar duas declarações para o melhor e o pior cenário:

- A complexidade do tempo no pior cenário de Gerenciamento de Inserção é **Θ(n 2)**.
- A complexidade temporal da classificação da melhor caixa de inserção é **Θ(n)**.

A notação **Big-O** é útil quando temos apenas um limite superior na complexidade temporal de um

algoritmo. Muitas vezes encontramos facilmente um limite superior simplesmente olhando para o algoritmo.

Exemplos:

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

Nota: aqui, **U** representa a união, podemos escrevê-la desta forma porque **O** fornece limites exatos ou superiores.