Pontifícia Universidade Católica de Minas Gerais
Trabalho Algoritmos e Estruturas de Dados II

Noções de complexidade

Daniel Vitor de Oliveira Santos

Belo Horizonte
2021

1 Introdução

Dada a definição de um algoritmo como uma sequência de passos para chegar a uma solução, sabe-se que podem existir diversas, entretando precisamos medir aquela que será a melhor de todas em uma determi nada situação. Para isso, utilizamos a Complexidade de Algoritmos para projetar algoritmos eficientes e prever a quantidade de recursos que ele irá demandar a medida que o tamanho do problema cresce, principalmente o tempo de execução do algoritmo.

2 Análise de Algoritmos

Conta-se o número de operações relevantes realizadas por um algoritmo e expressa-se esse número como uma função de n. Essas operações podem ser comparações, operações aritméticas, movimento de dados, etc. Sobre a quantidade de recursos utilizados no algoritmo, podemos destacar três casos:

- **Melhor caso:**

  É o menor custo possível na execução de um algoritmo, normalmente as funções de melhor caso podem ser delimitadas inferiormente usando a notação assintótica Ω.

- **Pior caso:**

  Indica o maior tempo de execução de um algoritmo qualquer. A ordem de crescimento da complexidade de pior caso normalmente é usada pra compara a eficiência de dois algoritmos. O pior caso é comumente mais utilizado pois o tempo de execução dele estabelece um limite supeior para o tempo de execução para qualquer entrada, conhecê-lo nos garante que o algoritmo nunca demorará mais do que esse tempo esperado.

- **Caso médio:**

  É a quantidade de algum recurso computacional utilizado pelo algoritmo, numa média sobre todas as entradas possíveis.

3 Notações O, Ω e Θ

As propriedades do somatório facilitam o desenvolvimento das expressões algébricas, com o objetivo de chegar às somas simples ou somas de quadrados.

- **Notação O:**

  É dada quando temos apenas um **limite assuntótico superior**. Para uma dada função $g(n)$, denotamos por $O(g(n))$ o conjunto de funções

$$O(g(n)) = f(n) : \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que}$$
$$0 \leq f(n) \leq cg(n) \text{ para todo } n \geq n_0$$

Usamos a notação O para dar um limite supeior a uma função, dentro de um fator constante. Para todos os valores n em n_0 ou à direita de n_0, o valor da função $f(n)$ está abaixo de $cg(n)$. Com a notação O, podemos descrever frequentemente o tempo de execução de um algoritmo apenas inspecionando a estrutura global do algoritmo.

- **Notação Ω:**

  Da mesma maneira que a notação O fornece um limite assuntótico superior para uma funçao, a notação Ω nos dá um **limite assintótico inferior**. Para uma dada função $g(n)$, denotamos por $\Omega(g(n))$ o conjunto de funções

$$\Omega(g(n)) = f(n) : \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que}$$
$$0 \leq cg(n) \leq f(n) \text{ para todo } n \geq n_0$$

  Para todos os valores n em n_0 ou à direita de n_0, o valor de $f(n)$ encontra-se em $g(n)$ ou acima de $g(n)$.

- **Notação Θ:**

  A notação Θ fornece uma simbologia simplificada para representar um limite justo de desempenho para um algoritmo. Um limite exato de tempo que um algoritmo leva para ser executado. Ou seja, a notação Θ representa o **ponto de encontro entre as notações** Ω (limite inferior) e Big O (limite superior). Para uma dada função $g(n)$, denotamos por $\Theta(g(n))$ o conjunto de funções

$$\Theta(g(n)) = f(n) : \text{existem constantes positivas } c_1, c_2 \text{ e } n_0 \text{ tais que}$$
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ para todo } n \geq n_0$$

  Uma função $f(n)$ pertence ao conjunto $\Theta(g(n))$ se existirem cosntantes positivas c_1 e c_2 tais que ela possa ser "encaixada" entre $c_1 g(n)$ e $c_2 g(n)$, para um valor de n suficientemente grande.