EDB II

Andriel Vinicius

October 6, 2024

1 Análise de Algoritmos

É fundamental estudarmos a complexidade de nossos algoritmos pois:

- Podemos verificar sua eficácia e eficiência em diversas instâncias;
- Adequamos-o à capacidade de processamento de diversas máquinas diferentes;
- Podemos verificar se o algoritmo é viável.

O tempo de execução (eficiência) de um algoritmo depende de fatores como:

- Infraestrutura utilizada;
- Tamanho do problema (parâmetros grandes, ...);
- Estruturas de dados utilizadas;
- Linguagem e compilador usados;
- Programador.

Por isso, a análise dos algoritmos deve ser feita de forma independente dos recursos de hardware, já que rodar o mesmo programa em computadores diferentes (por ex.) pode ter tempos de resposta diferentes.

Então, para realizar esta análise, utilizamos o **Modelo de computação RAM**. Esse modelo abstrato é capaz de realizar todas as tarefas em um tempo t constante.

1.1 Modelo de computação RAM

Suas principais características são:

- Cada instrução I possui um tempo associado $t(i)$ para ser operacionalizada em RAM.
- O tempo total de execução do algoritmo $t(n)$ será dado por $\sum_{j=1}^{m} r_j t(I_j)$, onde r_j é uma instrução de tipo I_j) e $t(I_j)$ é o tempo de execução da instrução.

1.2 Complexidade em Tempo

Def. A função $t_A : 0, 1^* \to \mathbb{N}$ é dita complexidade do algoritmo A se, para toda entrada x, A termina em exatamente $t_a(x)$ passos.

Podemos analisar os seguintes casos:

- **Melhor caso**: analisamos o número *mínimo* de passos para executar o algoritmo;
- **Pior caso**: analisamos o número *máximo* de passos para executar o algoritmo;

A complexidade média não costuma ser tão utilizada, tendo em vista que envolve a soma das probabilidades de cada caso acontecer, algo bem complexo de se calcular.

- Nota: instruções que dependem do tamanho da entrada são escritas como $t(n)$, e as constantes, t. Por ex., um algoritmo de complexidade $6t(n) + 4t$ tem 4 instruções constantes e independentes, e 6 que dependem da entrada.

2 Complexidade assintótica

A complexidade assintótica fornece limites para a função de complexidade. É representada em:

- Big O: representa o pior dos casos;
- Ômega: representa o melhor dos casos;
- Theta: representa o caso onde o pior e o melhor caso tem o mesmo desempenho.

2.1 Big O

Suponha duas funções $f(n), g(n)$. Diz-se que $f(n)$ é de ordem $g(n)$, ou $f(n)$ é $O(g(n))$, se o crescimento de $f(n)$ é, no máximo, tão rápido quanto o crescimento de $g(n)$.

Def.: Diz-se que $f(n)$ é $O(g(n))$ se existem as constantes reais positivas C e n_0 tais que $f(n) \leq cg(n)$, para todo $n > n_0$.

Esta def. é o mesmo que dizer que $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ existe e é finito. Note que:

- Se $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R}_+^*$, então $f(n) \in O(g(n))$ ($f(n)$ é de ordem $O(g(n))$);
- Se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, então $f(n) \in O(g(n))$ ($f(n)$ é de ordem $O(g(n))$);
- Se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$, então $f(n) \notin O(g(n))$ ($f(n)$ não é de ordem $O(g(n))$).

2.1.1 Regras

Básicas:

- Termos de ordem menores não importam;
- Constantes não importam;
- Dar ênfase onde é gasto mais tempo (repetição de código);
- Verificar a complexidade de funções nativas da linguagem.

Regra da Soma: se um algoritmo se divide em duas ou mais partes independentes, a complexidade $t(n)$ total será dada por $t(n) = t_1(n) + t_2(n) + \cdots + t_i(n)$ e o algoritmo será de ordem $O(max f_1(n), f_2(n), \cdots, f_i(n))$.

Regra do Produto: se um algoritmo se divide em duas ou mais partes dependentes, a complexidade $t(n)$ total será dada por $t(n) = t_1(n) \cdot t_2(n) \cdot \cdots \cdot t_i(n)$ e o algoritmo será de ordem $O(f_1(n) \cdot f_2(n) \cdot \cdots \cdot f_i(n))$.