Análise de Algoritmos

Estruturas de Dados e Algoritmos – Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília, Campus Taguatinga

Sumário

Sumário

Análise de Algoritmos

Análise de Algoritmos

- Analisar algoritmos é essencial para que se possa prever os recurso, tais como tempo e espaço, a serem utilizados.
- Temos que definir um modelo computacional, visto que diferentes modelos computacionais possuem diferentes complexidades nas operações:
  - Máquina de Turing.
  - Cálculo-λ.
  - Assembly Mips.
  - Código em C.
- É necessário fixar um modelo computacional para a análise de algoritmos.

Modelo RAM

Modelo RAM

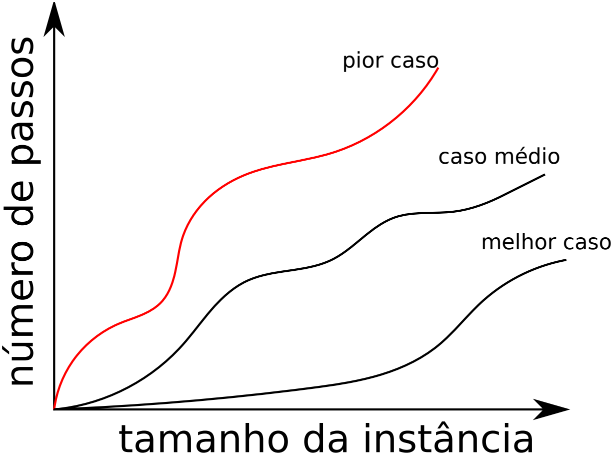
- O Modelo RAM corresponde bem à maioria das Arquiteturas encontradas no mundo real:
  - Instruções aritméticas (add, div,...);
  - Instruções de dados (load,str,cpy);
  - Instruções de controle(branches, loops, ...);
- Cada operação básica leva um passo, e, neste modelo, o tempo é mensurado pelo número de **passos**, já o espaço pelo número de **posições** de memória utilizados.

Análise de Algoritmos

Pior Caso, Caso Médio e Melhor Caso

- O pior caso de complexidade de um algoritmo é a função definida pelo maior número de passos com tamanho de instância n.
- O melhor caso de complexidade é dada pela função definida pelo menor número de passos com tamanho de instância n.
- O caso médio de complexidade do algoritmo é dado por uma função definida pela média do número de passos sobre todas as instâncias de tamanho n.

Análise de Algoritmos



Inspeção Linear

```c
int acha_menor(int* v,int n){
    int i;
    int menor = INT_MAX;
    for(i=0;i<n;i++){
        menor = menor > v[i] ? v[i] : menor;
    }
    return menor;
}
```

Análise de Algoritmos

- Qual o número de passos no pior caso de acharmos o menor elemento em um vetor arbitrário?
- Qual o número de passos no melhor caso de acharmos o menor elemento em um vetor arbitrário?

Análise de Algoritmos

- Não sabemos ao certo.
- Mas é proporcional ao tamanho do vetor, isto é, é $c \cdot n$ para alguma constante n.

Notação Assintótica

Notação Assintótica

- Contar precisamente o número de passos executados numa máquina RAM é muito trabalhoso.
- Depende muitas vezes de detalhes específicos de implementação.
- É interessante colocar tudo em função de uma cota superior e uma cota inferior de complexidade de tempo e espaço.
- Ferramenta: Notação Assintótica.

Sumário

Notação Assintótica

- Chegar na fórmula fechada do número de passos do pior caso de um algoritmo é muito complicado.
- Por exemplo:

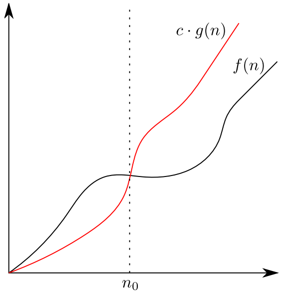
$$2n^2 + 10n + 300$$

- Nos dá quase a mesma informação que a função de pior caso cresce de maneira quadrática em função do tamanho da entrada.
- Além disso, a fórmula fechada depende de como as instruções foram usadas, e isto dificulta bastante o processo.
- A notação assintótica despreza esses detalhes e se concentra somente no crescimento da função.

Notação O

Notação O

$$O(g(n)) = \{f(n) | 0 \leq f(n) \leq c \cdot g(n), c \in \mathbb{R}^{+}, \forall n > n_0 \in \mathbb{R}^{+}\}$$



Notação O

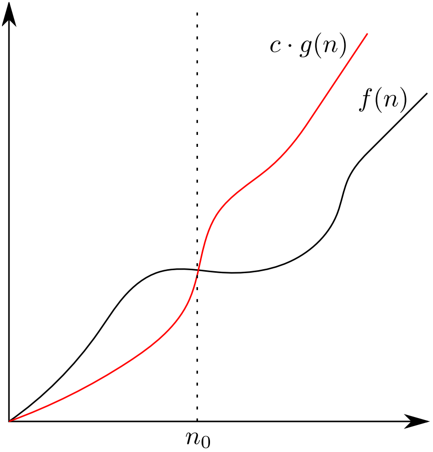


Figura: Notação O.

Notação O

Notação O

- A notação O nos dá uma cota superior, em termos assintóticos.
- Intuitivamente, ela nos diz que se uma função $f \in O(g(n))$, então f não cresce mais rápido que $g(n)$, em termos assintóticos.

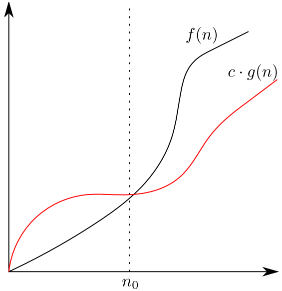
Exemplo

$$
\begin{array}{ll}
n^2 \in O(n^3) & n^2 \notin O(1) \\
n \in O(2^n) & 2n^2 \notin O(n) \\
\log n \in O(\log^2 n) & \log n \notin O(\log \log n)
\end{array}
$$

Notação Ω

Notação Ω

$$\Omega(g(n)) = \{f(n) | 0 \leq c \cdot g(n) \leq f(n), c \in \mathbb{R}^+, \forall n > n_0 \in \mathbb{R}^+\}$$



Notação Ω

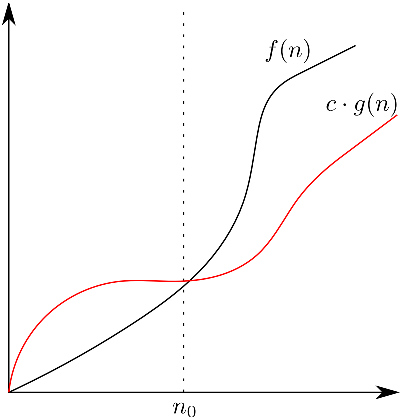


Figura: Notação Ω.

Notação Ω

Notação Ω

- A notação Ω nos dá uma cota inferior, em termos assintóticos.
- Intuitivamente, ela nos diz que se uma função $f \in \Omega(g(n))$, então f cresce mais rápido que $g(n)$, em termos assintóticos.

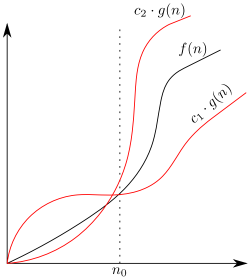
Exemplo

$$\begin{array}{ll} n^2 \in \Omega(n) & n^2 \notin \Omega(n^3) \\ 2n^2 \in \Omega(n^2) & 2n^2 \notin \Omega(2^n) \\ \log n \in \Omega(\log \log n) & \log n \notin \Omega(n) \end{array}$$

Notação Θ

Notação Θ

$$\Theta(g(n)) = \{f(n) | 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n),$$
$$c_1, c_2 \in \mathbb{R}^+, \forall n > n_0 \in \mathbb{R}^+\}$$



Notação Θ

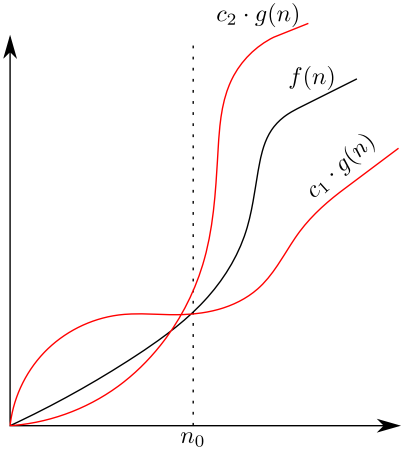


Figura: Notação Θ.

Notação Θ

Notação Θ

- A notação Θ nos dá uma cota **justa**, em termos assintóticos.
- Intuitivamente, ela nos diz que se uma função $f \in \Theta(g(n))$, então f cresce tanto quanto $g(n)$, em termos assintóticos.

Exemplo

$$
\begin{array}{ll}
n^2 \in \Theta(n^2) & n^2 \notin \Theta(n^3) \\
10^{900}n^2 \in \Theta(n^2) & 200n^2\Theta(n) \\
\frac{1}{2}\log n \in \Theta(\log n) & \frac{1}{2}\log n \notin \Theta(\sqrt{n})
\end{array}
$$

Notação Assintótica

Ajustando as Cotas

- É imprescindível buscar sempre uma cota justa para a função de custo de pior caso de um algoritmo, isto é:
  - $O(g(n)) \downarrow$
  - $\Omega(g(n)) \uparrow$
  - $\Theta(g(n))$
- Superestimar cotas superiores ou subestimar cotas inferiores levam uma estimativa inadequada dos recursos necessários para a execução do algoritmo.

Notação Assintótica

- Voltando a nosso algoritmo de achar o menor.
- Qual o número de passos necessários no pior caso?

Inspeção Linear

```c
int acha_menor(int* v,int n){
    int i;
    int menor = INT_MAX;
    for(i=0;i<n;i++){
        menor = menor > v[i] ? v[i] : menor;
    }
    return menor;
}
```

Notação Assintótica

$$\Theta(n)$$