Análise de Complexidade de Algoritmos - Notação *Big O*

Universidade Federal do Pará
Intituto de Tecnologia
Faculdade de Engenharia da Computação de Telecomunicações
Teoria da Computação II
Danilo Henrique Costa Souza - 201006840008

Belém, 13 de Novembro de 2014

Resumo

Este artigo tem comoo objetivo principal enumerar as notações *Big O* bem como explicar suas principais funcionalidades. E também dar uma breve introdução sobre esta notação e análise de complexidade de Algoritmos

Palavras-chaves: *Big O*, Complexidade, Algoritmos.

Introdução

Projetar um algoritmo não consiste apenas em planejar seu objetivo e escrever o código que irá realizar a tarefa, seu desenvolvimento continua após sua concepção através de seu melhoramento, um algoritmo deve estar em constante evolução não somente corrigindo pequenos bugs mas também aprimorando e modificando quando necessário a sua lógica. A análise da complexidade de algoritmos utiliza a notação Big O, que vem justamente para ajudar a medir a eficácia de um dado algoritmo em termos de custos de espaço (e.g, memória, disco) e tempo, entretanto a medida de tempo absoluto varia de máquina para máquina (i.e, poder de processamento e arquiteturas podem variar) e isso causa uma diferença no tempo medido para um mesmo algoritmo. A solução é medir o número de oprações realizadas por um algoritmo (n) e exibir sua complexidade em função deste parâmetro.

1 Notação *Big O*

A notação Big O utiliza este método para analisar a complexidade de algoritmos e esta análise consiste em eleger uma entrada n para o algoritmo e verificar quantas operações são necessárias para obter um resultado válido. A notação Big O toma como base uma função f, por exemplo, $f(n) = 4n^3 + 200n + 1000$, para definir $O(f(n))$ faz-se n $\rightarrow \infty$ eliminando assim os termos de menor importância, para este exemplo teríamos O(f(n)) = 4n, com isso é possível definir os cenários de pior, médio e melhor caso que serão descritos a seguir.

1.1 Diferentes Complexidades

Para entender melhor a notação Big O é necessário entender primeiramente os principais tipos de complexidade usados hoje em dia, elas serão listadas abaixo. A Figura 1 (BARRICO, 2014) mostra um gráfico comparativo entre as complexidades citadas .

- O(1): O tempo de execução do programa não muda conforme a entrada aumenta de tamanho.
- O(log n): O tempo cresce em proporções logarítmicas conforme o crescimento da entrada, dobrando somente quando n vai para n^2.
- O(n): Tempo de exexução linear, cresce à na mesma proporção do crescimento de n.
- O(n log n): Usado em problemas divididos em problemas menores e depois suas soluções são combinadas.
- O(n^2): Tempo de execução quadrático, à medida que n dobra, o tempo é multiplicado por 4 por exemplo.
- O(n^3): Tempo de execução cúbico, segue o mesmo princípio de O(n^2).
- O(2^n): Tempo de execução exponencial, típico em soluções de força bruta e não muito encontrado na prática pois os algoritmos com esta complexidade normalmente demoram, na prática, anos para terminarem sua execução.

1.2 Notação *Big O*

A seguir serão mostradas as definições de pior, médio e melhor caso. Para melhor entendimento será considerado como exemplo o algoritmo Quick-Sort (ALBUQUERQUE, 2014).

- Pior caso: n é o melhor conjunto de dados possível para o algoritmo (e.g, quando o vetor está em ordem invertida ou já ordenado).

- Médio Caso: n é um conjunto de valores aleatórios (e.g, quando os valores do vetor estão distribuídos de maneira uniforme).
- Melhor caso: n é o melhor conjunto de dados possível para o algoritmo (e.g, quando o pivô divide o vetor no meio).

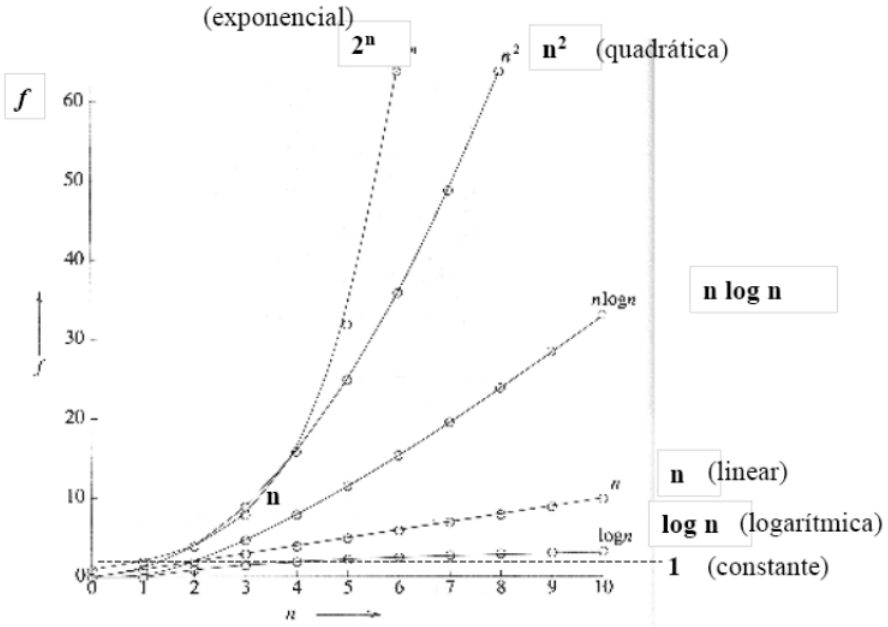

Figura 1 – Comparativo entre as complexidades citadas

Referências

ALBUQUERQUE, J. *Análise de Complexidade de Algoritmos*. 2014. `http://www.cin.ufpe.br/~joa/menu_options/school/cursos/ppd/aulas/complexidade.pdf`. Citado na página 2.

BARRICO, C. *Análise de COmplexidade*. 2014. `http://www.di.ubi.pt/~cbarrico/Disciplinas/AlgoritmosEstruturasDados/Downloads/Teorica_AnaliseComplexidade.pdf`. Citado na página 2.