Estrutura de Dados Avançada

Daniel de Sousa Moraes
danielmoraes14@gmail.com

Análise de Algoritmos

- Prever os recursos que o algoritmo precisa

- Ocasionalmente, recursos como memória, largura de banda, hardware são a principal preocupação, porém o **tempo de execução** é o mais frequentemente medido.

- Para analisar um algoritmo, precisa-se de um modelo de tecnologia como base. Consideraremos um modelo genérico de ***random-access machine***, com um único processador onde as instruções são executadas uma após a outra

Análise de Algoritmos

- O modelo RAM possui um conjunto de instruções básicas bem definidas.
- Instruções aritméticas: soma, subtração, etc. Cada uma custa uma quantidade de tempo constante
- O guia para o modelo é como os computadores reais são projetados

Análise do pior caso

- Na análise de um algoritmo concentraremos sempre no tempo de execução do **pior caso**.
  - O tempo de execução do pior caso estabelece o limite superior para o tempo de execução para qualquer entrada.
  - Em alguns algoritmos o pior caso ocorre com bastante frequência
  - Muitas vezes o “caso médio” é quase tão ruim quanto o pior caso

Ordem de crescimento

- Algumas abstrações são usadas para facilitar a análise de um algoritmo
  - O custo real de cada instrução é ignorado, utilizando-se constantes, porém até mesmo essas constantes trazem mais detalhes do que precisamos

- No caso do InsertionSort, expressamos o tempo de execução do pior caso como $an^2 + bn + c$ para constantes a,b,c que dependem de um custo c_i. Desse modo não só os custos reais são ignorados como também os custos abstratos c_i.

Ordem de crescimento

- A ordem de crescimento ou taxa de crescimento é uma abstração ainda mais simplificada

- Ela considera apenas o elemento de maior ordem (por exemplo, an^2), pois os termos de ordem mais baixa se tornam relativamente insignificantes para grande valores de n.

- Além disso, também ignora-se o termo constante, visto que fatores constantes também são menos significativos.

Ordem de crescimento

- No InsertionSort, por exemplo, ao ignorar-se esses termos, afirma-se que o tempo de execução do pior caso é $\Theta(n^2)$ ou $O(n^2)$

- Em geral, consideramos um algoritmo mais eficiente que o outro se seu tempo de execução do pior caso apresenta uma ordem de crescimento mais baixa.

Notação Assintótica

- Embora, as vezes, seja possível determinar o tempo exato de execução de um algoritmo, o ganho em precisão em geral não vale o esforço do cálculo.

- Para entradas suficientemente grandes, as constantes multiplicativas e os termos de ordem mais baixa de um tempo de execução exato são dominado pelos efeitos do tamanho da entrada.

Notação Assintótica

- Quando observa-se entradas muitos grandes para tornar relevante a ordem de crescimento do tempo de execução de um algoritmo, estuda-se a eficiência **assintótica** dos algoritmos.

- Ou seja, preocupa-se com o modo como o tempo de execução aumenta com o tamanho da entrada *no limite*, à medida que a entrada aumenta sem limitação.

Notação Assintótica

- As notações usadas para descrever o tempo de execução assintótico são definidas em termos de funções cujos domínios são o conjunto dos números naturais.

- Tais notações são convenientes para descrever a função *T(n)* do tempo de execução do pior caso, que em geral é definida somente para tamanhos de entrada inteiros

Notação Assintótica

- Usaremos notação assintótica para descrever o tempo de execução de algoritmos, porém na realidade a notação aplica-se a funções.

- Então quando dizemos que o tempo de ordenação do InsertSort é $\Theta(n^2)$, significa que $\Theta(n^2)$ é a função $an^2 + bn + c$

- As funções às quais aplicamos a notação assintótica caracterizam o tempo de execução, porém pode-se aplicar à funções que caracterizam outros aspectos.

Notação Assintótica

- Mesmo usando a notação assintótica para o tempo de execução, é preciso entender a qual tempo estamos nos referindo (na maioria das vezes o do pior caso).

- Algumas vezes precisamos propor um enunciado abrangente que aplique a todas as entradas e não apenas ao pior caso. As notações assintóticas a seguir prestam-se a esse objetivo.

Notação *O*

- A notação *O* fornece um **limite assintótico superior** a uma função, dentro de um fator constante.

- Para uma função $g(n)$, denotamos por $O(g(n))$ (lê-se “Ó grande de *g* de *n*” ou apenas “ó de *g* de *n*) o conjunto de funções:

  $f(n) = O(g(n))$, se existe um número positivo c e um número n_0 tais que $f(n) \leq c \cdot g(n)$ para todo n maior que n_0.

Notação Ω

- A notação Ω fornece um **limite assintótico inferior** a uma função, dentro de um fator constante.

- Para uma função $g(n)$, denotamos por $\Omega(g(n))$ (lê-se “ômega grande de g de n” ou, às vezes, “ômega de g de n) o conjunto de funções:

  $f(n) = \Omega(g(n))$, se existe um número positivo c e um número n_0 tais que $f(n) \geq c \cdot g(n)$ para todo n maior que n_0.

Notação Θ

- A notação Θ fornece tanto um **limite assintótico inferior** como um **superior** a uma função, dentro de um fator constante.

- Para uma função $g(n)$, denotamos por $\Theta(g(n))$ (lê-se “teta de g de n”) o conjunto de funções:
  $f(n) = \Theta(g(n))$, se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$. Ou seja, existe números positivos c e d tais que $c\, g(n) \leq f(n) \leq d\, g(n)$ para todo n suficientemente grande.

Bibliografia

Cormen, Thomas H. et al. Algoritmos.; [tradução Arlete Simille]. 3ª ed - Rio de Janeiro - Elsevier, 2011.