Algoritmos e Estrutura de Dados II

Análise Assintótica de Algoritmos

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia

Agenda

Introdução

- Na última aula aprendemos a calcular a complexidade de um algoritmo contando o nº de vezes que as instruções são executadas.
- No entanto, é inviável medir a complexidade exata de um algoritmo, visto que alguns outros fatores podem influenciar nessa complexidade.
  - Linguagem de programação
  - Compilador
  - Hardware
- Precisamos saber a **complexidade assintótica** definida pelo termo dominante de uma função de crescimento.

Ao analisar valores grandes de n percebe-se que as constantes faze pouca diferença...

Tabela: Comparativo MaxMin

Algoritmo	Melhor Caso	Pior Caso	Caso Médio
MinMax1	4n	4n	4n
MinMax2	3n+1	5n-1	$\frac{(7n-1)}{2}$
MinMax3	3n+6	4n+4	$\frac{7n}{2}+4$
MinMax4	3n+1	4n-3	$\frac{7n}{2}-2$

Introdução

Comportamento Assintótico

Quando observamos tamanhos de entrada grandes o suficiente para tornar relevante apenas a ordem de crescimento do tempo de execução, estamos estudando a **eficiência assintótica** dos algoritmos (Cormen et al., 2009).

- Na notação assintótica, representa-se uma função pelo termo que cresce mais rapidamente, ignorando fatores constantes.
- Analisa-se o algoritmo quando o valor tende ao *infinito*.
- Permite "simplificar" expressões, focando apenas nas **ordens de grandeza**.

Introdução

Exemplo: Qual os termos dominates das equações a seguir:

- $T_1(n) = \frac{1}{2}n^2 - 3n$
- $T_2(n) = n + 1$
- $T_3(n) = n^5 + n^3 + 100$
- $T_4(n) = 2^n - \frac{2}{3}n^3 + 5n^2 - 3n + 5000$
- $T_5(n) = \sqrt{n} + 1$
- $T_6(n) = n \log_2 n + 2n - 5$

Relações de Domínio

$$n! \ggg 2^n \ggg n^3 \ggg n^2 \ggg n \log n \ggg n \ggg logn \ggg 1$$

Introdução

Exemplo: Qual os termos dominates das equações a seguir:

- $T_1(n) = \frac{1}{2}n^2 - 3n$
  - Resposta: n^2
- $T_2(n) = n + 1$
  - Resposta: n
- $T_3(n) = n^5 + n^3 + 100$
  - Resposta: n^5
- $T_4(n) = 2^n - \frac{2}{3}n^3 + 5n^2 - 3n + 5000$
  - Resposta: 2^n
- $T_5(n) = \sqrt{n} + 1$
  - Resposta: $\sqrt{n}$
- $T_6(n) = n \log_2 n + 2n - 5$
  - Resposta: $n \log_2 n$

Introdução

Utiliza-se uma notação específica:

- Notação O
  - Representa um limite superior, relacionado ao **pior caso**.
- Notação Ω
  - Representa um limite inferior, relacionado ao **melhor caso**.
- Theta: Θ
  - Representa casos em que os **limites superiores e inferiores são iguais**.

Comportamento Assintótico

BuscaVetor

Relembrando o tempo de execução do algoritmo que **busca um elemento em um vetor**:

- Melhor caso: $T(n) = 3$.
  - Portanto o algoritmo é $\Omega(1)$.
- Pior caso: $T(n) = 2n + 3$.
  - Portanto o algoritmo é $O(n)$

Comportamento Assintótico

MaxVetor

Relembrando o tempo de execução do algoritmo que encontra o elemento **máximo de um vetor**:

- Melhor caso: $T(n) = 2n + 1$.
  - Portanto o algoritmo é $\Omega(n)$.
- Pior caso: $T(n) = 3n$
  - Portanto o algoritmo é $O(n)$.
- Como o algoritmo é $\Omega(n)$ e $O(n)$, ele também é $\Theta(n)$.

Comportamento Assintótico

MaxMin

Relembre as funções:

Tabela: Comparativo MaxMin

Algoritmo	Melhor Caso	Pior Caso	Caso Médio
MinMax1	4n	4n	4n
MinMax2	3n+1	5n-1	$\frac{(7n-1)}{2}$
MinMax3	3n+6	4n+4	$\frac{7n}{2}+4$
MinMax4	3n+1	4n-3	$\frac{7n}{2}-2$

- Observe que todos os algoritmos tem a mesma complexidade assintótica.
- Todos são $O(n)$ e $\Omega(n)$. Portanto, são $\Theta(n)$.

Classes Assintóticas

- Em geral, é interessante agrupar os algoritmos/problemas em **Classes de Comportamento Assintótico**.
- Quando dois algoritmos fazem parte da mesma classe de comportamento assintótico, eles são ditos equivalentes

Classes Assintóticas

$O(1)$

$T(n) = O(1)$

- Algoritmos de complexidade $O(1)$ são ditos de **complexidade constante**.
- Uso do algoritmo independe de n.
- As instruções do algoritmo são executadas um número fixo de vezes

Classes Assintóticas

$O(\log n)$

$T(n) = O(\log n)$

- Um algoritmo de complexidade $O(\log n)$ é dito ter **complexidade logarítmica**.
- Típico em algoritmos que transformam um problema em outros menores.
- Pode-se considerar o tempo de execução como menor que uma constante grande.
  - Quando $n = 10^3$, $\log_2 n \approx 10$; quando $n = 10^6$, $\log_2 n \approx 20$.
  - Para dobrar o valor de $\log n$ temos de considerar o quadrado de n.
  - A base do logaritmo muda pouco estes valores: quando $n = 10^3$, $\log_2 n \approx 20$ e $\log_{10} n$ é 6.

Classes Assintóticas

$O(n)$

$T(n) = O(n)$

- Um algoritmo de complexidade O(n) é dito ter complexidade linear.
- Em geral, um pequeno trabalho é realizado sobre cada elemento de entrada.
- É a melhor situação possível para um algoritmo que tem de processar/produzir n elementos de entrada/saída.
- Cada vez que ndobra de tamanho, o tempo de execução dobra

Classes Assintóticas

$O(n \log n)$

$T(n) = O(n \log n)$

- Típico em algoritmos que quebram um problema em outros menores, resolvem cada um deles independentemente e ajuntando as soluções depois.
- Quando $n = 1$ milhão, $n \log_2 n \approx 20$ milhões.
- Quando $n = 2$ milhões, $n \log_2 n$ é cerca de 42 milhões, pouco mais do que o dobro.

Classes Assintóticas

$O(n^2)$

$T(n) = O(n^2)$

- Um algoritmo de complexidade $O(n^2)$ é dito ter **complexidade quadrática**.
- Ocorrem quando os itens de dados são processados aos pares, muitas vezes em um anel dentro de outro.
- Quando n é mil, o número de operações é da ordem de 1 milhão.
- Sempre que n dobra, o tempo de execução é multiplicado por 4.
- Úteis para resolver problemas de tamanhos relativamente pequenos.

Classes Assintóticas

$O(n^3)$

$T(n) = O(n^3)$

- Um algoritmo de complexidade $O(n^3)$ é dito ter **complexidade cúbica**.
- Úteis apenas para resolver pequenos problemas.
- Quando $n = 100$, o número de operações é da ordem de 1 milhão.
- Sempre que n dobra, o tempo de execução fica multiplicado por 8.

Classes Assintóticas

$O(2^n)$

$T(n) = O(2^n)$

- Um algoritmode complexidade $O(2^n)$ é dito ter **complexidade exponencial**.
- Geralmente não são úteis sob o ponto de vista prático.
- Ocorrem na solução de problemas quando se usa força bruta para resolvê-los.
- Quando $n = 20$, o tempo de execução é cercade 1 milhão. Quando n dobra, o tempo fica elevado ao quadrado.

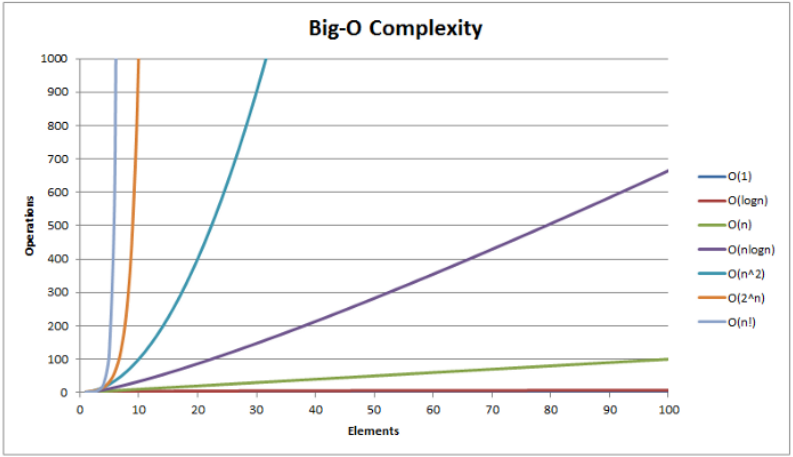
Classes Assintóticas

$O(n!)$

$T(n) = O(n!)$

- Um algoritmode complexidade $O(n!)$ é dito ter **complexidade exponencial**, apesar de $O(n!)$ ter comportamento muito pior do que $O(2^n)$.
- Geralmente ocorrem quando se usa força bruta para solução do problema.
- Ocorrem na solução de problemas quando se usa força bruta para resolvê-los.
- $n = 20 \rightarrow 20! = 2432902008176640000$, um número com 19 dígitos.
- $n = 40 \rightarrow$ um número com 48 dígitos.

Comportamento Assintótico



FIM

Fim

Algoritmos e Estrutura de Dados II

Análise Assintótica de Algoritmos

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia