Comportamento Assintótico de Funções

[redacted]

Adaptado dos slides da
Profª. Renata Oliveira

A notação “O”

- A ordem de crescimento do tempo de execução de um algoritmo nos dá uma caracterização da sua eficiência e nos permite comparar o desempenho relativo com outros algoritmos.
- Quando nos concentramos em tamanhos grandes de entrada, que torna relevante apenas a ordem de crescimento do tempo de execução do algoritmo, estamos estudando a EFICIÊNCIA ASSINTÓTICA do mesmo, ou seja, estamos preocupados em como o tempo de execução de um algoritmo cresce com o tamanho da entrada crescendo infinitamente.

  Ou ainda: Qual o comportamento do algoritmo para entradas cada vez maiores?

A notação “O”

- A ordem de magnitude de um algoritmo é dada pela soma das frequências de todas as suas declarações. Ela pode ser extraída diretamente do algoritmo, independente da máquina ou linguagem.
- Existe uma notação matemática adequada para tratar da ordem de magnitude. É chamada de notação O (“O” grande ou Big “O”), e foi sugerida por Knuth.

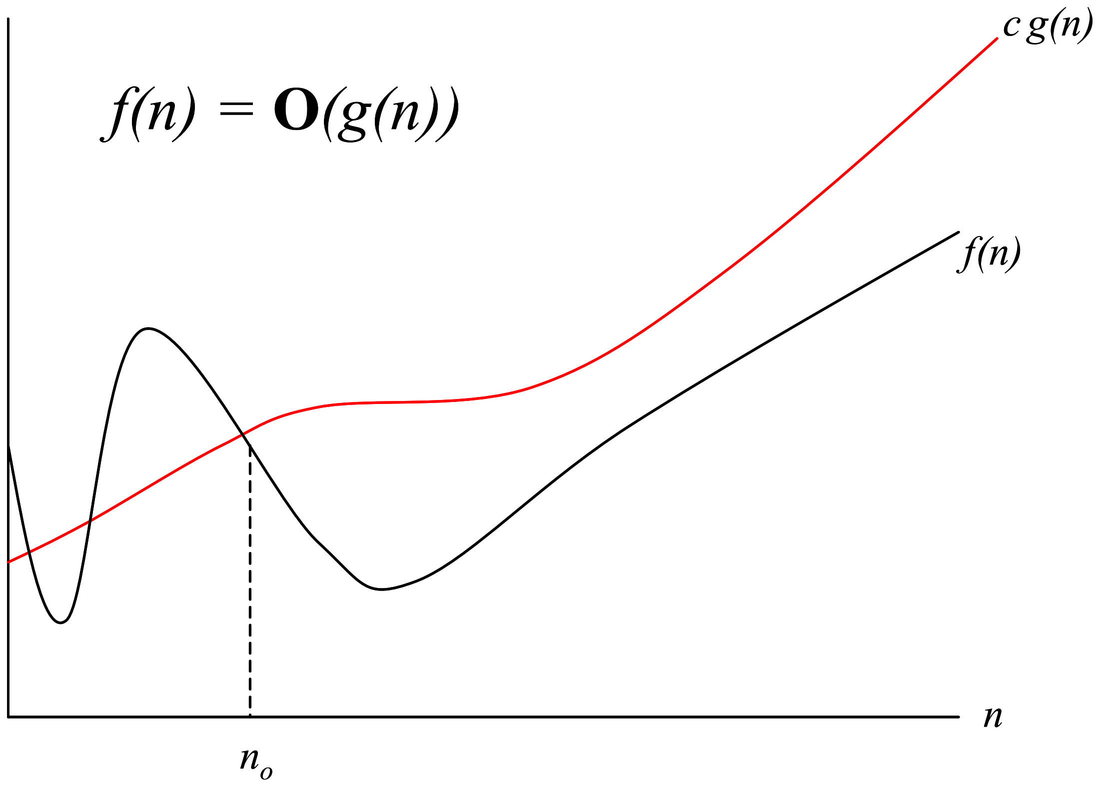
Definição

- Uma função g(n) domina assintoticamente outra função, f(n), se existem constantes c e n_0 (positivas) tais que:

$$\boxed{\forall n \geq n_0 \quad |f(n)| \leq c\,|g(n)|}$$

- Neste caso dizemos que f(n) é O(g(n)) e escrevemos f(n) = O(g(n))

Definição



Dizemos que g(n) domina assintoticamente f(n), ou que g(n) é o limite assintótico superior para f(n)

Exemplo

- Suponha que $f(n) = (3/2)n^2 + (7/2)n – 4$ e que $g(n)= n^2$. A tabela abaixo sugere que $f(n) \leq 2\ g(n)$ para $n \geq 6$ e, portanto, $f(n) = O(g(n))$

n	f(n)	g(n)
0	-4	0
1	1	1
2	9	4
3	20	9
4	34	16
5	51	25
6	71	36
7	94	49
8	120	64

Exemplo

- Mostre, usando a definição, que $3n^2+n$ é $O(n^2)$

Análise de Algoritmos

- A análise de um algoritmo conta apenas algumas operações “elementares” (por exemplo, comparações, atribuições, multiplicações, acessos ao disco, etc).
- A medida de custo ou medida de complexidade relata o crescimento assintótico da operação considerada.
- Se f é uma função de complexidade para um algoritmo F, então O(f) é considerada a complexidade assintótica ou o comportamento assintótico do algoritmo F.
- A relação de dominação assintótica permite comparar funções de complexidade.

Teorema

- Se $A(n) = a_m n^m + ... + a_1 n + a_0$ é um polinômio de grau m, então $A(n) = O(n^m)$.

Exemplo:

- A função de complexidade de um algoritmo de ordenação A é dada por

$$f(n) = \frac{n^3 + 2n^2 + n + 1}{5}$$

- Logo, dizemos que este algoritmo tem complexidade $O(n^3)$.
- Isto quer dizer que para valores muito grandes de n poderíamos desconsiderar a outra parte do polinômio ($2n^2 + n + 1$) pois ela não influenciaria tanto no crescimento de f(n).

Importante:

- Um algoritmo é $O(g(n))$ significa que se ele está rodando em um computador com valores crescentes de n, os tempos de execução resultantes serão sempre menores que $|g(n)|$.
- Quando tentamos determinar a ordem de complexidade de $f(n)$, estamos procurando pela menor $g(n)$ tal que $f(n) = O(g(n))$.

Propriedades:

- $f(n) = O(f(n))$
- $c.O(f(n)) = O(f(n))$ c=constante
- $O(f(n) + O(f(n)) = O(f(n))$
- $O(O(f(n)) = O(f(n))$
- $O(f(n) + g(n)) = O(Max(f(n), g(n)))$
- $O(f(n)) . O(g(n)) = O(f(n).g(n))$
- $O(f(n).g(n)) = f(n). O(g(n)) = g(n).O(f(n))$

Outras notações assintóticas

- Notação Omega (Ω):
  Limite assintótico inferior $f(n) = \Omega(g(n))$ se existirem constantes positivas c e n_0 tais que:

$$|f(n)| \geq c\,|g(n)|, \ \forall n \geq n_0$$

- Notação Teta (θ):
  $f(n) = \theta(g(n))$ se existirem constantes positivas c_1, c_2, n_0 tais que:

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|, \ \forall n \geq n_0$$

Propriedades das funções O, θ, Ω

- Considere # = O, θ, Ω
  - f(n) = #(g(n)) e g(n) = #(h(n)) ⇒ f(n) = #(h(n))
  - f(n) = #(f(n))
  - f(n) = θ(g(n)) se e somente se f(n) = O(g(n)) e f(n) = Ω(g(n))
- Resultados Importantes:
  - f(n) é O(g(n)) ⇒ $0 \leqslant \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$
  - f(n) é Ω(g(n)) ⇒ $0 < \lim_{n \to \infty} \frac{f(n)}{g(n)} \leqslant \infty$
  - f(n) é θ(g(n)) ⇒ $0 < \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

Exercícios

1. Mostre, por definição, que :
   a) $\sqrt{n}$ é $O(n^2)$
   b) n^2 é $O((n+1)^2)$

2. João desenvolveu um algoritmo cuja função de complexidade é $f(n) = 2n^2+3$. Mostre, por definição, que $f(n)$ é $O(n^2)$.

Exercícios

- José, na análise de quatro algoritmos para criptografia de dados, A_1, A_2, A_3 e A_4 encontrou as seguintes expressões para o número de operações matemáticas realizadas:

A_1: $f(n) = \log n^{10}$

A_2: $f(n) = 2n$

A_3: $f(n) = 10n$

A_4: $f(n) = \sqrt[4]{n}$

Ajude-o a demonstrar o seguinte

- A_1 é $\theta(\log n)$
- A_2 é $\Omega(n^m)$ para $m > 1$
- A_3 é $O(n \log n)$
- A_4 é $\Omega(\ln n)$

Classes de comportamento assintótico

- $f = O(1)$ complexidade constante
  - O uso do algoritmo independe de n.
  - $O(c)$ para $c \in R$ é $O(1)$
  - Comandos de atribuição, leitura, escrita.
- $f = O(\log(n))$ complexidade logarítmica
  - O custo de aplicar o algoritmo a problemas com n suficientemente grande pode ser limitado por uma função do tipo $K.\log n$ $K \in R$.
- $f = O(n)$ complexidade linear
  - O custo do algoritmo cresce linearmente com o tamanho da entrada.

Classes de comportamento assintótico

- $f = O(n \log n)$
- $f = O(n^2)$ complexidade quadrática
- $f = O(n^3)$ complexidade cúbica
- $f = O(cn)^c$, constante, complexidade exponencial
- $f = O(n!)$ complexidade fatorial

$O(1) < O(\log \log n) < O(\log n) < O(n^{1/10}) < O(\sqrt{n}) < O(n) <\ldots < O(n \log n) < O(n^2) < O(n^5) < O(2^n) < O(n!)$

Análise de algoritmos: Alguns princípios

1. O tempo de execução de um comando de atribuição de leitura e ou de escrita pode ser considerado O(1).
2. O tempo de execução de uma SEQUÊNCIA de comandos é determinado pelo maior tempo de execução de qualquer comando da sequência
3. O tempo de execução de um comando de DECISÃO é composto pelo tempo dos comandos executados dentro da condição, mais o tempo para avaliar a condição.
4. O tempo para executar um laço é a soma do tempo de execução do corpo do laço mais o tempo de avaliar a condição de parada, multiplicado pelo número de iterações do laço.

Análise de algoritmos: Alguns princípios

5. PROGRAMAS COM PROCEDIMENTOS NÃO RECURSIVOS
   - Cada procedimento é tratado separadamente, iniciando pelos procedimentos que não chamam outros procedimentos.
   - A seguir, computar o tempo dos procedimentos que chamam os procedimentos que não chamam outros procedimentos, utilizando os tempos já avaliados.
   - Repetir o processo até chegar no procedimento principal.

Exemplo 1: Encontrar o maior e o menor elemento de um vetor

```
void maxmin(A:vetor;)
{
    int i;
    max = A[0];
    min = A[0];
    for (i = 1; i < n;i++)
        {
            if (A[i] > max) max = A[i];
            if (A[I] < min)  min = A[i];
        }
}
```

Exemplo 2: Ordenar os elementos de um vetor

```
void Ordena (vetor A)
{
    int i, j, min, x;
    for (i = 1; i <= n; i++)
      {
        min = i;
        for ( j = (i+1); j <= n; j++ )
            if (A[j-1] < A[min-1] )
                min := j;
        x = A[min-1];
        A[min-1] = A[i-1];
        A[i-1] = x;
      }
}
```

Análise de algoritmos: Alguns princípios

6. PROGRAMAS COM PROCEDIMENTOS RECURSIVOS
   - Para cada procedimento recursivo, associar uma função de custo f(n) desconhecida (n mede o tamanho dos argumentos)
   - Obter uma relação de recorrência.
   - Calcular a complexidade da relação de recorrência.

Recursão

- Uma definição na qual o item que está sendo definido aparece como parte da definição é chamada definição indutiva ou definição recursiva.
- Definições recursivas são compostas de duas partes:
  - Uma base, onde alguns casos simples do item que está sendo definido são dados explicitamente.
  - Um passo indutivo ou recursivo, onde outros casos do item que está sendo definido são dados em termos dos casos anteriores.

Sequências Recursivas

- Uma sequência S é uma lista de objetos que são enumerados segundo alguma ordem; existe o 1º objeto, um segundo e assim por diante.
- S(k) denota o K-ésimo objeto da sequência
- Uma sequência é definida recursivamente, explicitando-se seu primeiro valor (ou seus primeiros valores) e, a partir daí, definindo-se outros valores na sequência em termos dos valores iniciais.

Exemplo

- Seja a sequência S definida recursivamente por:

$$\begin{cases} S(1) = 2 \\ S(n) = 2S(n-1) \quad \text{para } n \geq 2 \end{cases}$$

- Qual o valor de S(1)? E de S(2)? E de S(3)? E de S(5)?

Relações de Recorrência

- As relações de recorrência são utilizadas para descrever algoritmos recursivos.
- Uma recorrência é uma equação ou desigualdade que descreve uma função em termos de entradas menores.
- Existem três métodos principais para resolver relações de recorrência.
  - Método da Substituição
  - Método da Iteração
  - Método Master ou Mestre

Método da Substituição

- "Chute Inicial"
- Indução matemática para provar que o chute estava correto.

$$
\begin{cases}
T(n) = 3T(n-1) + 1 \\
T(1) = 1
\end{cases}
$$

Método da Iteração

- A ideia é expandir (iterar) a recorrência e expressá-la como uma soma de termos dependentes apenas de n e das condições iniciais. Técnicas para avaliar somas podem então ser usadas para encontrar a solução.

Exemplos:

$$\begin{cases} T(n) = T(n-1) + 1 \\ T(1) = 1 \end{cases}$$

$$\begin{cases} T(n) = T(n/2)+1 \\ T(1) = 0 \end{cases}$$

$$\begin{cases} T(n) = T(n-1) + n \\ T(1) = 1 \end{cases}$$

Método Master (ou Mestre)

- O método Máster provê uma “receita” para resolver recorrências do tipo **$T(n) = a\,T(n/b) + f(n)$** com $a \geq 1$, $b>1$ constantes e $f(n)$ função assintoticamente positiva.
- A ideia da recorrência é a divisão de um problema de tamanho **n** em **a** subproblemas, cada um de tamanho **n/b**.
- Os a subproblemas são resolvidos de forma recursiva, cada um com tempo $T(n/b)$.
- O custo de dividir o problema e combinar os resultados dos subproblemas é descrito pela função $f(n)$.

O Teorema Máster:

- Seja $T(n) = a\,T(n/b) + f(n)$ $a \geq 1$, $b > 1$, $f(n) \geq 0$ Então $T(n)$ pode ser limitada assintoticamente como segue:
  i. Se $f(n) = O(n^{(\log_b a) - \varepsilon})$ para alguma constante $\varepsilon > 0$ então $T(n) = \theta(n^{\log_b a})$
  ii. Se $f(n) = \theta(n^{\log_b a})$ então $T(n) = \theta(n^{\log_b a}.\ \log n)$
  iii. Se $f(n) = \Omega(n^{(\log_b a) + \varepsilon})$ para alguma constante $\varepsilon > 0$ e $a.f(n/b) \leq c.f(n)$ para alguma constante $c < 1$ então $T(n) = \theta(f(n))$

Exemplos:

$$\begin{cases} T(n) = 9T(n/3) + n \\ T(1) = 1 \end{cases} \qquad \begin{cases} T(n) = T(2n/3) + 1 \\ T(1) = 1 \end{cases}$$