02 – Análise de Algoritmos (parte 1)

SCC5900 - Projeto de Algoritmos

Material gentilmente cedido pelo Prof. Moacir Ponti Jr.

Modificado por Joao Batista Neto

www.icmc.usp.br/~jbatista

Instituto de Ciências Matemáticas e de Computação – USP

2011/1

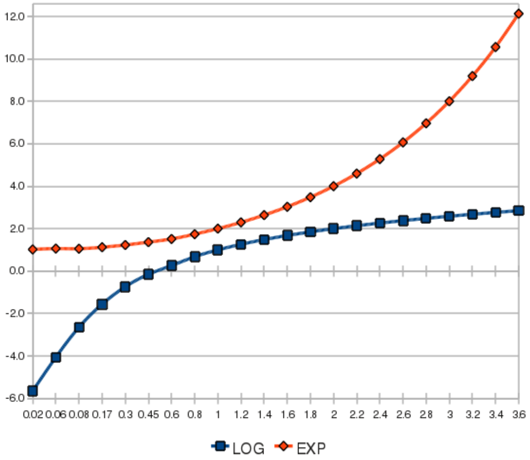
Sumário

1. Análise Assintótica: ordens de crescimento
   - Revisão de matemática
   - Abordagem: contagem de operações e tamanho da entrada

2. Bibliografia

- Expoentes
  - $x^a x^b = x^{a+b}$
  - $x^a / x^b = x^{a-b}$
  - $(x^a)^b = x^{ab}$
  - $x^n + x^n = 2x^n$ (e <u>não</u> x^{2n})
  - $2^n + 2^n = 2^{n+1}$
- Logaritmos (por padrão, base 2)
  - $(x^a = b) \Rightarrow (\log_x b = a)$
  - $\log_a b = \log_c b / \log_c a$ para $c > 0$
  - $\log ab = \log a + \log b$
  - $\log a/b = \log a - \log b$
  - $\log(a^b) = b \log a$
  - $\log x < x$ para todo $x > 0$

Função logaritmica X exponencial



Exponencial: $y = 2^x$, Logaritmo: $y = \log 2x$

Revisão de matemática

- Séries:

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}$$

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \approx \frac{n^2}{2}$$

Abordagem: contagem de operações e tamanho da entrada

Relembrando o objetivo

- ser capaz de, dado um problema, mapea-lo em uma **classe de algoritmos** e encontrar a **"melhor" escolha** entre os algoritmos, com base em sua eficiência.

Complexidade computacional e eficiência

- A **complexidade computacional** está ligada à eficiência. Algoritmos mais "caros" computacionalmente são menos eficientes.

Abordagem para análise assintótica da complexidade

- O número de **passos básicos** necessários em função do **tamanho da entrada** que o algoritmo recebe.
  - descorrelaciona a performance da máquina da performance do algoritmo.
  - reduz a análise ao número de operações realizadas em função do tamanho da entrada.

Análise Assintótica: passos básicos e tamanho da entrada

Tamanho da entrada?

- Depende do problema, mas geralmente é relativo ao número de elementos da entrada que são processados pelo algoritmo
  - o número de elementos em um arranjo, lista, árvore, etc.
  - o tamanho de um inteiro que é passado por parâmetro.

Passos básicos?

- Se referem às operações primitivas utilizadas pela máquina:
  - operações aritméticas,
  - comparações,
  - atribuições,
  - resolver um ponteiro ou referência,
  - indexação em um arranjo,
  - chamadas e retornos de funções.

Análise Assintótica: ordens de crescimento

Suponha que

- o tamanho da entrada é n
- cada operação leva aproximadamente o mesmo tempo (constante).
- a memória é infinita

Eficiência com base na ordem de crescimento

- a eficiência de um algoritmo representada por uma função
- **eficiência assintótica** descreve a eficiência de um algoritmo quando n torna-se grande.

Para comparar algoritmos

- determinamos suas ordens de crescimento (eficiência assintótica)
- o algoritmo com a **menor** ordem de crescimento deverá executar mais rápido para tamanhos de entradas maiores.

Análise Assintótica: ordens de crescimento

- A análise assintótica reduz o problema a uma resposta menos precisa, mas fácil de derivar e de interpretar.
- Algumas "consequências" desse tipo de análise são:
  - Ter de definir um **modelo de máquina** único com as operações básicas.
  - A eficiência de um algoritmo pode estar relacionada à detalhes dos dados de entrada além do seu tamanho, e portanto existem diversos cenários: **melhor caso**, **pior caso** e **caso esperado** (médio).

Análise Assintótica: ordens de crescimento

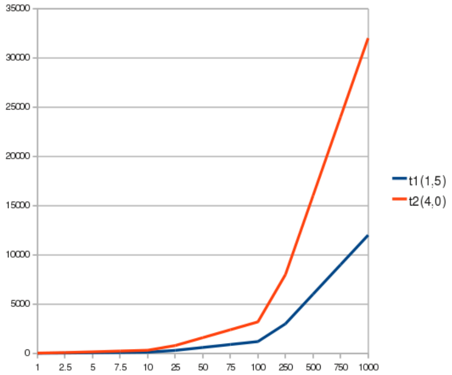
- **Melhor caso**: não é uma boa análise
  - Pode nunca ocorrer na prática!
- **Caso esperado** (médio): seria o ideal (intuitivamente), mas determiná-lo não é uma tarefa trivial.
  - Usado em algumas situações
  - É preciso conhecer a **distribuição de probabilidade** (*descreve a chance de uma variável aleatória assumir um valor ao longo de um espaço de valores*) típica da entrada, e utilizar teoria da probabilidade para determinar.
- **Pior caso**: recomendado
  - Fácil de identificar
  - Como se trata de um **limite superior** sobre o tempo de execução para cada entrada, não há surpresas!
  - Para diversos algoritmos o pior caso ocorre com frequência
  - Em muitos casos o caso esperado está próximo ao pior caso.

```
// entrada: arranjo A de n inteiros      ---
// saída: elemento máximo em A           Num. de operações
int maxArranjo(int *A, int n) {          ---
   int i, atualMax = *A;                 2
   for (i = 1; i < n; i++) {             2n
       if (*A > atualMax) {              2(n-1)
          atualMax = *A;                 2(n-1)
       }
       A++;                              2(n-1)
   }
   return atualMax;                      1
}
```

Estimando a eficiência de `maxArranjo`

- O algoritmo executa:
  - No pior caso: $2 + 1 + 2n + 6(n-1) + 1 = 8n - 2$ operações.
  - No melhor caso: $2 + 1 + 4(n-1) + 1 = 6n$ operações.
- Suponha que:
  - t_1 é o tempo gasto pela primitiva mais rápida
  - t_2 é o tempo gasto pela primitiva mais lenta
- Se $T(n)$ é o tempo de **pior caso** de `maxArranjo`, então:
  - $t_1(8n-2) \leq T(n) \leq t_2(8n-2)$
- $T(n)$ é limitado por duas funções **lineares**.

Estimando a eficiência de `maxArranjo`



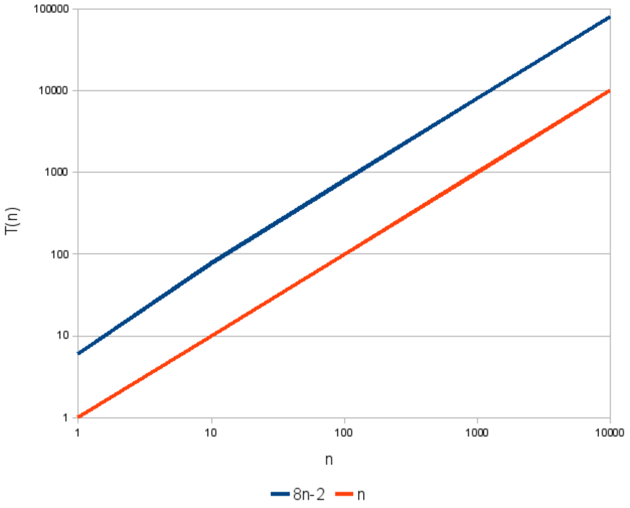
Exemplo: $t_1 = 1,5$, $t_2 = 4,0$

Estimando a eficiência de `maxArranjo`

- Após essa análise, q qual conclusão podemos chegar quanto à eficiência do algoritmo quando o tamanho da entrada aumenta?
  - **A função que caracteriza a complexidade computacional do algoritmo é de ordem linear.**
  - Podemos desprezar todos os fatores constantes e termos de menor ordem, pois esses não afetam a taxa de crescimento em si. $t(n) = 8n - 2 \approx n$

$T(n)$	1	10	100	1.000	10.000
$8n - 2$	6	78	798	7.998	79.998
$8n$	8	80	800	8.000	80.000
n	1	10	100	1.000	10.000

Estimando a eficiência de `maxArranjo`

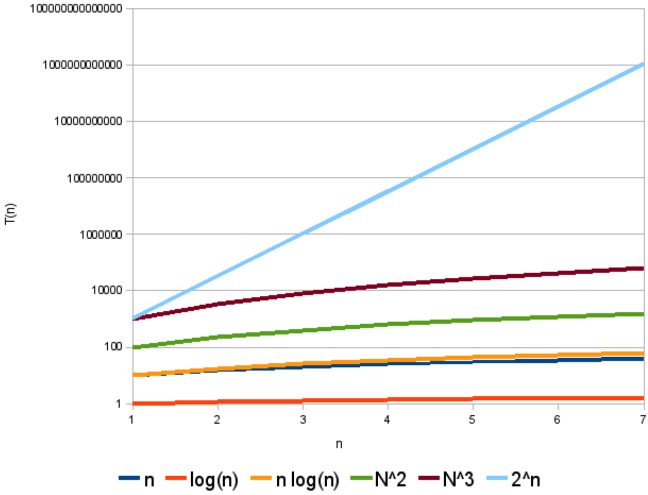


Exemplo: em escala logarítmica

Funções importantes

- As funções que aparecem na análise de algoritmos:
  - Constante: ≈ 1
  - Logarítmica: $\approx log_b n$
  - Linear: $\approx n$
  - Log linear (ou n-log-n): $\approx n \cdot log_b n$
  - Quadrática: $\approx n^2$
  - Cúbica: $\approx n^3$
  - Exponencial: $\approx a^n$
  - Fatorial: $\approx n!$

Funções importantes



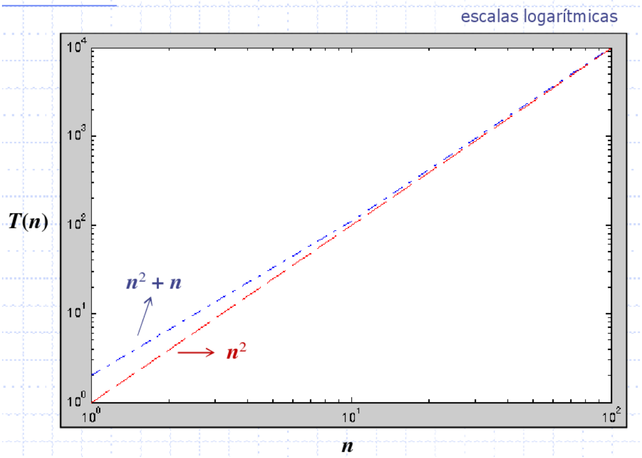
Exemplo: em escala logarítmica

Fatores constantes e de termos de menor ordem

- Mudar o hardware/software afeta a taxa de crescimento por um fator constante, mas não altera a ordem de complexidade dos algoritmos.
- Termos de menor ordem também não afetam o crescimento conforme o tamanho da entrada cresce

$T(n)$	1	10	100	1.000
n^2	1	100	10.000	1.000.000
$n^2 + n$	2	110	10.100	1.001.000
Δ	100%	10%	1%	0,1%

Fatores constantes e de termos de menor ordem



Fonte da figura: notas de aula do Prof. Ricardo Campello

Bibliografia

- CORMEN, T.H. et al. **Algoritmos: Teoria e Prática** (Caps. 1–3). Campus. 2002.
- ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C (Cap. 1). 2.ed. Thomson, 2004.
- FEOFILOFF, P. **Minicurso de Análise de Algoritmos**, 2010. Disponível em: `http://www.ime.usp.br/~pf/livrinho-AA/`.
- DOWNEY, A.B. **Analysis of algorithms** (Cap. 2), Em: Computational Modeling and Complexity Science. Disponível em: `http://www.greenteapress.com/compmod/html/book003.html`
- ROSA, J.L. **Notas de Aula de Introdução a Ciência de Computação II**. Universidade de São Paulo. Disponível em: `http://coteia.icmc.usp.br/mostra.php?ident=639`