Universidade Federal de Ouro Preto
Departamento de Computação

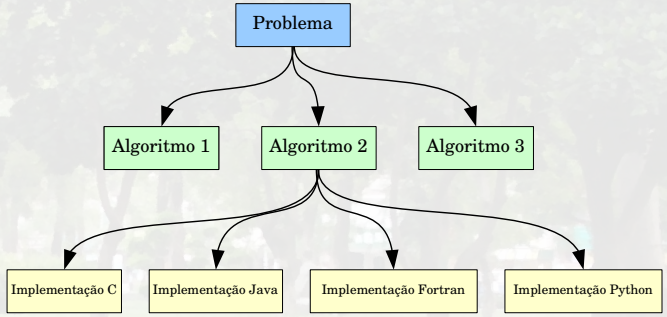
Projeto e Análise de Algoritmos - I

decom
departamento de computação

Prof. Haroldo Gambini Santos

Introdução

- Problema → Algoritmo → Implementação



Regra Geral

Nenhuma implementação excelente salva um Algoritmo inadequado.

Objetivo da Ciência da Computação

- Projetar algoritmos corretos (geram a solução esperada)
- Eficientes (executam em tempo e espaço aceitáveis) para problemas complexos

Curiosidade: Erros por Lin. de Código

- Média da Indústria:
  - 15-50 erros por mil linhas de código
  - Code Complete. Steve McConnel

Curso de Projeto e Análise de Algoritmos

- Calcular a eficiência de um algoritmo
- Conhecer paradigmas de projeto de algoritmo e suas aplicações
- Conhecer boas soluções para problemas clássicos e recorrentes na computação

Problema → Soluções

- Especificação (definição do problema)
- Projeto (“enciclopédia de soluções”)
- Prova de correção
- Análise de Complexidade – Eficiência Teórica
- Implementação (arquitetura/linguagem)
- Verificação/Testes

Tempo de Execução de um Algoritmo

- Depende:
  - Implementação computacional (programa, linguagem, compilador)
  - Ambiente de execução
    - Processador
    - Sistema Operacional
    - Memória (real e cache)
    - Barramento
    - ..
  - Organização e conteúdo dos dados de entrada

Difícil caracterizar precisamente !

Além da Análise Experimental

- Trabalharemos com uma **Metodologia Geral** para analisar o tempo de execução de um algoritmo, contrastando com a análise experimental, essa metodologia:
  - Usa uma **descrição de alto nível** do algoritmo
  - Considera **todas as entradas possíveis**
  - Analisa a eficiência do algoritmo de modo independente do ambiente de **hardware e software**

Pseudocódigo

- Explicação mais estruturada do que a linguagem natural, mas menos formal que no pseudocódigo
- Ex.: Buscar maior elemento em um vetor:

Algoritmo vMax(*A*, *n*):
Entrada: Vetor *A* com *n* inteiros
Saída: Maior elemento em *A*.
currentMax ← *A*[0]
for ***i*** ← **1 to *n* -1 do**
  if *currentMax* < A[i] **then** *currentMax* ← A[i]
return *currentMax*

Operações Primitivas

- Computações de baixo nível, independentes da linguagem de programação, que podem ser identificadas no pseudocódigo. Ex.:
  - Chamada e retorno de procedimentos
  - Operações aritméticas
  - Comparação de 2 números
  - Acessar conteúdo de uma posição de um vetor
  - ...

Contando Operações Primitivas

Algorithm *arrayMax*(*A*, *n*)
currentMax ← ***A***[0] — 2
for *i* ← 1 **to *n*** - 1 **do** — { 1 inicialização; *n* testes
  if *A*[*i*] > *currentMax* then — 2(*n*-1)
    currentMax ← ***A***[*i*] — 2(*n*-1)
  { increment counter ***i*** } — 2(*n*-1)
return *currentMax* — 1

Notação Assintótica

- Problemas realmente grandes
- Preocupação com o quão rápido os limites serão atingidos
  - Ordem de Crescimento de Funções

Notação Assintótica

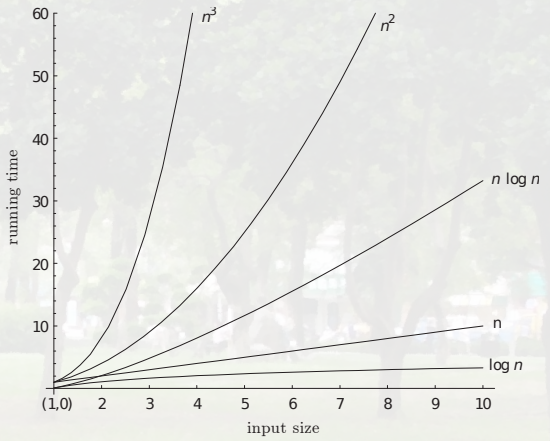
- Algoritmos A e B resolvem o mesmo problema
  - Complexidade de Tempo:
    - A: $5.000n$
    - B: 1.1^n
  - Considere n=10
    - A requer 50.000 passos
    - B requer 3
  - Considere n=1.000
    - A requer 5.000.000
    - B requer 2,5 x 10^{41}
  - Para **todos** os casos, exceto n<142 o tempo de A será menor que o tempo de B

Funções Típicas

- Tempo estimado em um computador com 4 GHz

n	$f=(n!)$	$f=(2^n)$	$f=(n^5)$	$f=(n^3)$	$f=(n^2)$
15	5 minutos	<1 s	<1 s	<1 s	<1 s
20	20 anos	<1 s	<1 s	<1 s	<1 s
50	d.f.v.t.	4 dias	<1 s	<1 s	<1 s
100	d.f.v.t.	d.f.v.t.	3 ss	<1 s	<1 s
500	d.f.v.t.	d.f.v.t.	3 horas	<1 s	<1 s
1.000	d.f.v.t.	d.f.v.t.	3 dias	<1 s	<1 s
10.000	d.f.v.t.	d.f.v.t.	8 séculos	4 min	<1 s
15.000	d.f.v.t.	d.f.v.t.	65 séculos	14 min	<1 s

Algumas Funções Típicas



Análise Assintótica

- Ignorar constantes dependentes da máquina
- Olhar a taxa de crescimento do tempo de execução, expressa através de uma função no tamanho da entrada
- Detalhes sem importância:
  - Constantes multiplicativas
  - Termos de mais baixa ordem

Dominação Assintótica

Uma função

$f(n)$

domina assintoticamente uma função

$g(n)$

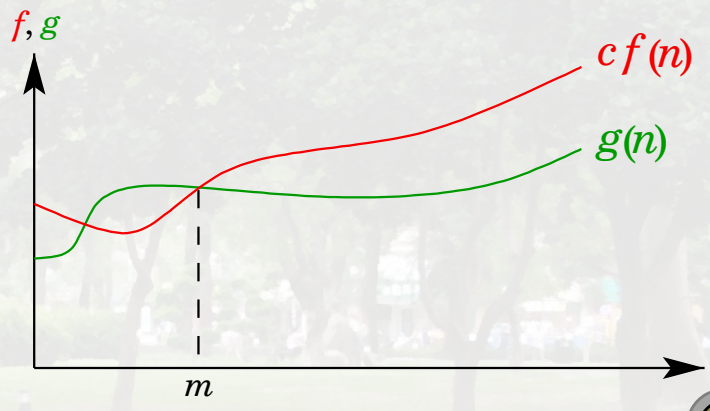
se existem duas constantes positivas,

c e m

tais que para $n \geq m$ temos que

$$|g(n)| \leq c \times |f(n)|$$

Dominação Assintótica



Notação big *O* – Knut, 1968

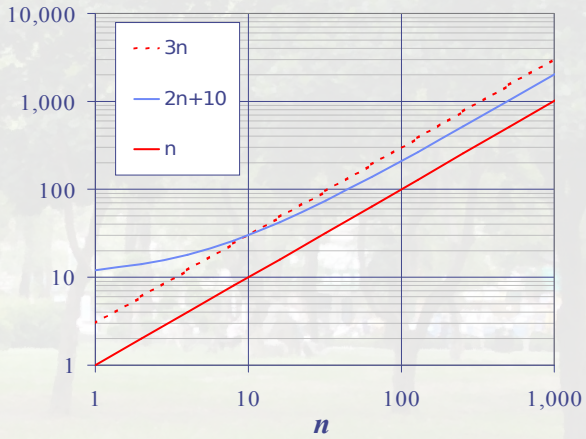
$g(n)$ é $O(f(n))$

se $f(n)$ **domina assintoticamente** $g(n)$

Notação big *O*, Exemplo

- Mostre que $2n + 10$ é $O(n)$
  - temos que encontrar m e c tais que
    $|2n+10| \leq c|n|$ para todo $n \geq c$
    - $2n+10 \leq cn$
    - $(c-2)n \geq 10$
    - $n \geq 10/(c-2)$
      - considere $c=3$ e $m=10$

Notação big *O*, Exemplo



Notação big *O*, Exemplo

- Mostre que n^2 não é $O(n)$
  - $n^2 \leq cn$
  - $n \leq c$

Desigualdade não pode ser satisfeita pois c é uma constante!

Big O, algumas regras gerais

- Se $f(n)$ é um polinômio de grau d, então $f(n)$ é $O(n^d)$, ou seja:
  - Descarte termos de menor ordem
  - Descarte fatores constantes
- Use a menor possível classe de funções
  - Diga: $2n$ é $O(n)$ ao invés de $2n$ é $O(n^2)$
- Use a expressão mais simples da classe
  - Diga $3n+5$ é $O(n)$ ao invés de $3n+5$ é $O(3n)$

Big O - Exemplo

- Mostre que $3x^3 + 5x^2 - 9 = O(x^3)$
  - considere $c=5$
  - $3x^3 + 5x^2 - 9 \leq 5x^3$
  - $5x^2 \leq 2x^3 + 9$
  - para qual m temos que $5x^2 \leq x^3$
    - 5

Big O - Exemplo

- Mostre que $2^{(n+1)}$ é $O(2^n)$
  - $2^{(n+1)} \leq 2.2^n \quad (c=2)$
  - $2^{(n+1)} \leq 2^{(n+1)}$
- Mostre que 2^{2n} não é $O(2^n)$
  - $2^n 2^n \leq c2^n$
  - 2^n não pode ser constante

Big O - Exemplo

- Mostre que $\log n + \log(\log n)$ é $O(\log n)$
  - $\log n + \log(\log n) \leq c \log n$
  - Considere $m = 2$
  - $\log 2 + \log(\log 2) \leq c \log 2$
    - $1 + \log 1 \leq c\ 1$
    - $1 + 0 \leq c\ 1$

Notação Θ

- Notação mais acurada: limita uma função tanto por baixo quanto por cima

***g*(*n*) é Θ*f*(*n*)** se existirem c_1 e c_2 e m tais que:

$$|g(n)| \geq c_1 \times |f(n)|$$

e

$$|g(n)| \leq c_2 \times |f(n)|$$

para todo $n \geq m$

Notação Ω

- Define um limite assintótico inferior, ou seja

$g(n)$ é $\Omega f(n)$ se existirem c e m tais que:
$|g(n)| \geq c \times |f(n)|$

para todo $n \geq m$

Exercícios

1. Preencha com Verdadeiro ou Falso

$f(n)$	$g(n)$	$f = O(g)$	$f = \Omega(g)$	$f = \Theta(g)$
$2n^3 + 3n$	$100n^2 + 2n + 100$			
$50n + \log n$	$10n + \log\log n$			
$50n \log n$	$10n \log\log n$			
$\log n$	$\log^2 n$			
$n!$	5^n			

Exercícios

2. Expresse as seguintes funções em termos da notação Θ:

(a) $7n^3 + 1000n\log n + 3n$
(b) $3n^{1,5} + (\sqrt{n})^3 \log n$
(c) $2^n + 100^n + n!$
(d) $18n^3 + \log n^8$
(e) $(n^3+n)/(n+5)$