01 – Análise de Algoritmos (parte 1)

SCC201/501 - Introdução à Ciência de Computação II

Prof. Moacir Ponti Jr.

www.icmc.usp.br/~moacir

Instituto de Ciências Matemáticas e de Computação – USP

2010/2

Sumário

Algoritmos

- Um algoritmo corresponde a uma descrição de um padrão de comportamento, expresso em termos de um conjunto finito de ações (Dijkstra, 1971 citado por Ziviani, 2004).
- Ao executarmos a operação $a + b$ percebemos um padrão de comportamento, mesmo para valores diferentes de a e b.

- Segundo Cormen et al. (2002): informalmente, um algoritmo é um procedimento computacional bem definido que toma um conjunto de valores como **entrada** e produz algum conjunto de valores como **saída**.

O que é importante?

- Quais características importantes em um algoritmo/software?

Performance

- ponto chave de qualquer software.

Simplicidade

- um algoritmo simples é mais fácil de ser implementado corretamente;
- por consequência há menor probabilidade de obter erros.

Clareza

- deve ser escrito de forma clara e documentada para facilitar a manutenção.

O que é importante?

Segurança

- deve ser seguro.

Funcionalidade

- deve possuir diversas funcionalidades.

Modularidade

- permite melhor manutenção, reuso, etc.

Interface amigável

- fundamental para a maior parte dos usuários.

Corretude

- ...

Algoritmos: corretude

- Segundo Cormen et al. (2002): um algoritmo é dito **correto** se, para cada instância de entrada, ele pára com a saída correta (ou informa que não há solução para aquela entrada.
- "deseja-se que um algoritmo termine e seja correto"

- Um algoritmo correto sempre termina?
- E se eu oferecer um algoritmo correto que permite obter uma solução em 3 anos?

- Problema **intratável**: não existe um algoritmo que solucione com demanda de recursos e tempo razoável.
  - Problemas para os quais não se conhece solução eficiente: $\mathcal{NP}$-difícil, $\mathcal{NP}$-completo

Sumário

Eficiência

- Computadores são muito rápidos atualmente
- No entanto, problemas crescem mais rapido do que a velocidade do computador

- É muito importante levar em consideração a eficiência de um algoritmo ao desenvolver software:
  - alguns programas executam instantaneamente
  - alguns programas executam de um dia para o outro,
  - alguns programas podem executar por séculos.

Um problemas e muitas estratégias e soluções

- Temos um mapa rodovíario e nossa meta é determinar a menor rota de um local a outro.
  - O numero de rotas pode ser enorme.
  - Diversas estratégias podem ser utilizadas para obter a menor rota.

Como garantir uma boa rota?

- Garantir a corretude do algoritmo!

Eficiência

- Uma das perguntas comuns em entrevista de emprego do Google: **"Qual a maneira mais eficiente de ordenar um milhão de inteiros de 32 bits?"**

- **É importante conseguir relacionar classes de problemas e algoritmos com a eficiência com base no tempo de execução.**

- Para conhecer a resposta é preciso:
  - definir "eficiente",
  - definir uma metodologia padronizada para medir **eficiência**, e
  - **comparar** a eficiência entre os algoritmos.

Eficiência

- A eficiência pode ser associada aos recursos computacionais:
  - a quantidade de **espaço de armazenamento** que utiliza,
  - a quantidade de **tráfego** que gera em uma rede de computadores,
  - a quantidade de dados que precisam ser **movidos do disco ou para o disco**.
- No entanto para a maior parte dos problemas a eficiência está relacionada ao **tempo de execução** em função do tamanho da entrada a ser processada.

Objetivo desse tópico da disciplina

- 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.

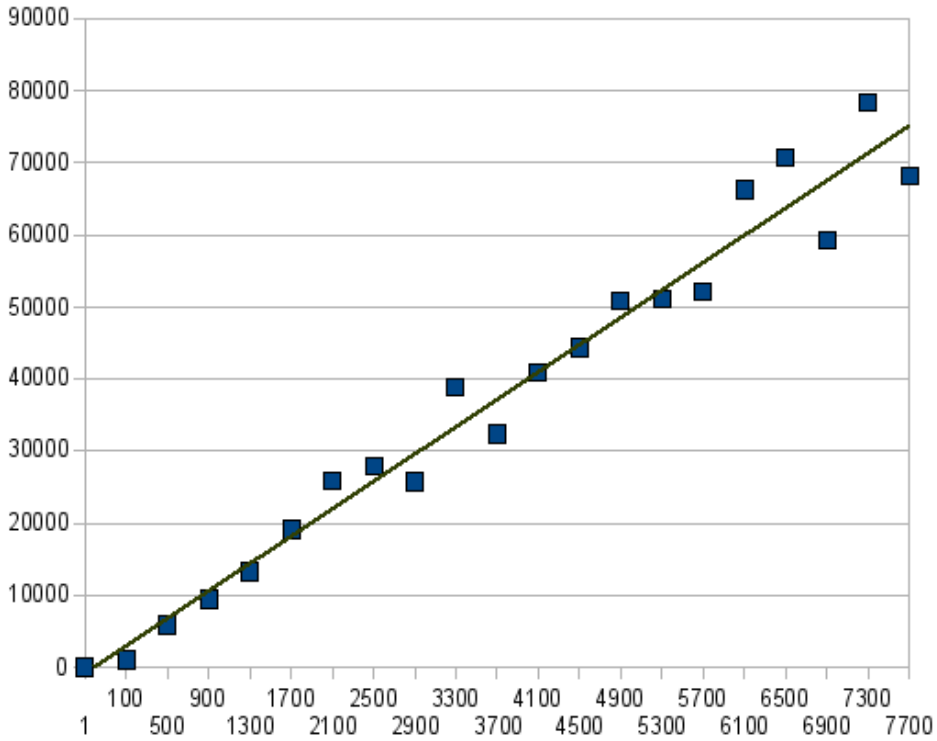
Eficiência: análise experimental

- Computar o tempo total de execução de um programa

```
#include <stdio.h>
#include <time.h>

int main (void) {
  time_t t1, t2, total;

  t1 = time(NULL); // retorna hora atual do sistema

  /* algoritmo */

  t2 = time(NULL);
  total = difftime(t2,t1); // retorna a diferenca t2-t1
  printf("\n\nTotal: %ld seg.\n", total);
  return 0;
 }
```

Gráfico da análise experimental

- Se realizarmos diversos experimentos e computarmos o tempo para entradas diferentes, poderemos traçar um gráfico como abaixo
  - no eixo x está o tamanho da entrada
  - no eixo y está o tempo em milisegundos que o algoritmo demora para processar a entrada



Eficiência: análise experimental

- Desvantagens e limitações:
  - é preciso realmente implementar e testar o algoritmo com diversas entradas diferentes
  - a análise será feita por um conjunto limitado de dados
  - o tempo dependerá de diversos fatores: hardware, sistema operacional, linguagem de programação, compilador, etc.

Analise a afirmação abaixo

"Desenvolvi um novo algoritmo chamado TripleX que leva 14,2 segundos para processar 1.000 números, enquanto o método SimpleX leva 42,1 segundos."

- você trocaria o SimpleX que já roda na sua empresa pelo TripleX?
- há vários fatores envolvidos (acima) e mais:
  - o TripleX também é mais rápido para processar quantidades maiores que 1.000 números?

Sumário

Eficiência: passos básicos e tamanho da entrada

Abordagem

- 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 desempenho em função do 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.

Eficiência: passos básicos e tamanho da entrada

Passos básicos?

- Se referem às operações primitivas utilizadas pela máquina:
  - operações aritméticas,
  - comparações,
  - chamadas à funções,
  - retornos de funções, etc.

Assumiremos

- o tamanho da entrada como sendo n
- cada operação leva aproximadamente o mesmo tempo constante.

Eficiência: TripleX vs. SimpleX

- *TripleX*: para uma entrada de tamanho n o algoritmo realiza $n^2 + n$ operações.
  - em termos de função, $t(n) = n^2 + n$.
- *SimpleX*: para uma entrada de tamanho n o algoritmo realiza $2000n$ operações.
  - em termos de função, $s(n) = 2000 \cdot n$.

Eficiência: TripleX vs. SimpleX

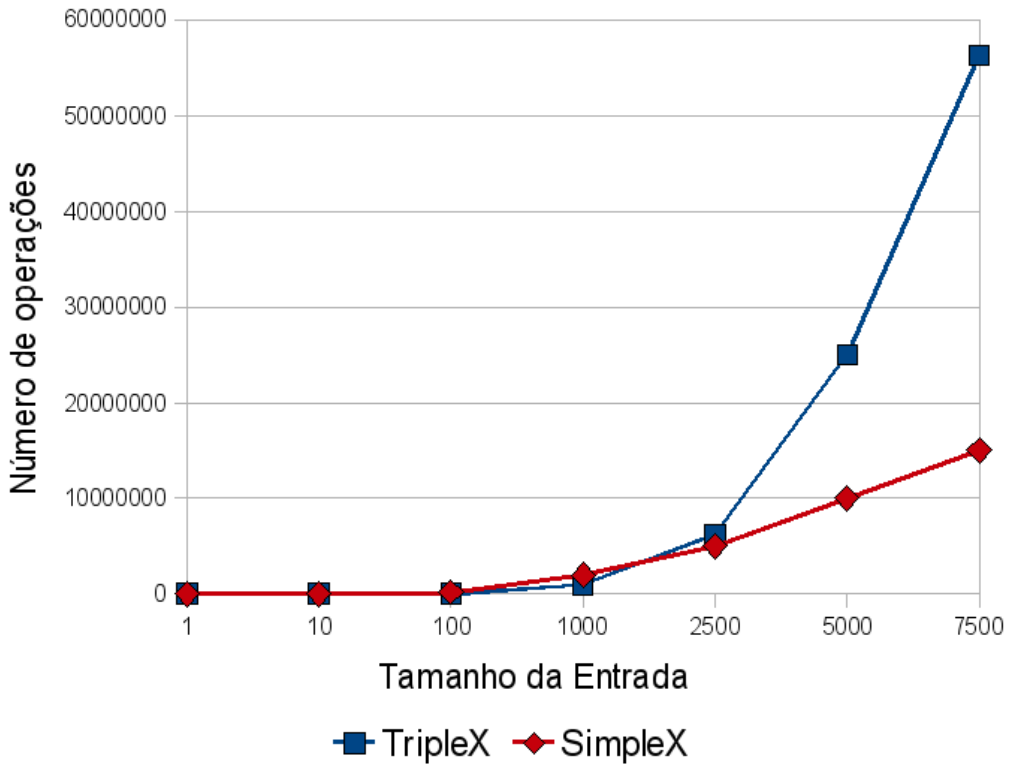
- Calculando o número de operações em função da entrada:

n	1	10	100	1.000	10.000
$t(n) = n^2 + n$					
$s(n) = 2000 \cdot n$					

Eficiência: TripleX vs. SimpleX

- Calculando o número de operações em função da entrada:

n	1	10	100	1.000	10.000
$t(n) = n^2 + n$	2	110	10.100	1.001.000	**100.010.000**
$s(n) = 2000 \cdot n$	2.000	20.000	20.000	2.000.000	20.000.000



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`