: Complexidade e Projeto de Algoritmos **versão: 5/6/24**

Tópico 1 — Introdução à análise de algoritmos

Professor: Vinícius Dias *Autor: Vinícius Dias*

Sumário

1.1 Introdução à análise de algoritmos

Um algoritmo pode ser definido como uma sequência de passos usada para resolver algum problema. Naturalmente, em computação, estamos mais interessados em problemas que possam ser *abstraídos* usando matemática e algoritmos para esses problemas que sejam *precisos* na definição de cada passo e *passíveis de reprodução* por um computador (ou modelo de computação).

Sendo assim, podemos estabelecer a seguinte relação entre o que é prático e o que é abstrato no estudo da resolução de problemas:

Programa	$\approx$	Algoritmo
Linguagem de programação	$\approx$	Pseudocódigo
Computador físico	$\approx$	Computador abstrato (ou modelo de computação)

Nosso objetivo é estudar algoritmos do ponto de vista abstrato (segunda coluna acima). A primeira etapa, portanto, é formalizar bem um problema para que não se tenha dúvidas com relação ao que é dado como entrada e o que se espera como resultado (saída). Por exemplo, o problema de ordenar um conjunto de números inteiros em um arranjo pode ser formalizado da seguinte forma:

Entrada:	sequência de números inteiros organizados em um arranjo $a = [a_1, a_2, \cdots, a_n]$
Saída:	permutação Θ $[a_1^\Theta, \cdots, a_n^\Theta]$ tal que $a_1^\Theta \leq a_2^\Theta \leq \cdots \leq a_n^\Theta$

1.2 Modelo de computação

O primeiro requisito ao se trabalhar com algoritmos é ser capaz de analisá-los, isto é, precisamos ser capazes de determinar certas características do algoritmo antes mesmo de implementá-lo e executá-lo na prática. Por exemplo, é muito comum que estejamos interessados em algoritmos *eficientes*. Precisamos, portanto, estabelecer como será o computador abstrato ao qual o algoritmo estudado se refere. No estudo de algoritmos vamos adotar um modelo de computação abstrata que reflete características muito comuns de arquiteturas computacionais contemporâneas: o modelo RAM – *Random Access Machine*. As seguintes características estão presentes em algoritmos projetados para o modelo RAM:

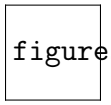
- As instruções dos algoritmos são executadas sempre sequencialmente, sem nenhum tipo de concorrência ou paralelismo;
- Vamos assumir que os dados manipulados pelos algoritmos são organizados em uma memória de acesso aleatório e organizada em palavras;

- Tipos de dados: inteiros, ponto flutuante, arranjos de palavras (sequência de inteiros ou ponto flutuante), qualquer outra estrutura de dados derivada;
- Operações aritméticas em geral: $+, -, \div, \times,$ mod , etc.;
- Controle: while, for, do .. while, return, subrotinas/funções, condicionais (if, else), operadores de comparação ($=, <, >,$, etc.);
- Atribuições de variáveis são representadas como $\leftarrow$.

1 Exemplo de pseudocódigo no modelo RAM. Vamos ver um exemplo de algoritmo para o problema de contar quantas vezes um número inteiro n ocorre em um arranjo de inteiros v. Formalmente, entrada: inteiro n e arranjo de inteiros v; saída: inteiro c representando o número de ocorrências de n em v. O tamanho de v é representado nesse exemplo como $|v|$:

```
CONTA-OCORRENCIAS(n, v)
1  c ← 0
2  for i ← 1 to |v|
3      if v[i] = n
4          c ← c + 1
5  return c
```

2 Dimensões de análise. Diferentes instâncias podem demandar diferentes níveis de dificuldade para a execução do algoritmo. Esperamos que, seja em uma máquina abstrata ou não, cada problema é passível de encontrar instâncias fáceis (entradas fáceis) ou instâncias mais difíceis (entradas mais difíceis). Por exemplo, para o problema de contar as ocorrências de um número no arranjo não é razoável assumir que o mesmo algoritmo irá se comportar da mesma forma para um arranjo de 10 elementos em comparação a um arranjo de 10^7 elementos. Por esse motivo, vamos sempre estudar algoritmos em função do *tamanho da entrada* que lhe é apresentada. Com isso, temos nosso principal objetivo em análise de algoritmos: para um algoritmo, estimar ou representar sua dificuldade frente a instâncias cada vez maiores na entrada:



As duas noções de dificuldade para um computador abstrato que estamos interessados são:

1. Tempo: quantas *operações básicas* um algoritmo executa para uma dada instância do problema. Nesse caso, ao invés de medir tempo de relógio, nós vamos contar quantas operações um algoritmo executa.
2. Espaço: quanto de espaço (palavras do RAM) além da entrada o algoritmo utiliza em função também do tamanho da instância de entrada.

Chamamos portanto de *operação básica* aquilo que pretendemos medir em um algoritmo. Essa escolha é feita por quem analisa o algoritmo, mas por exemplo: (a) para um problema de ordenar números em um arranjo pode ser natural pensar que as operações que são mais importantes para determinar a ordem entre os elementos são as operações que *comparam* dois elementos do arranjo: $a[i] \leq a[j]$, por exemplo; (b) para um problema matemático de encontrar o máximo divisor comum entre dois números pode ser uma boa prática escolher operações aritméticas (possivelmente mod) como operação básica. Portanto, não existe regra mas existem boas práticas.

3 Contexto de análise pior caso, melhor caso, caso médio

1.3 Corretude de algoritmos

usar o exemplo do insertion sort para mostrar a ideia do invariante de loop: inicialização, antes da execução de cada iteração do loop, terminação

1.4 Notações assintóticas

1.5 Analisando algoritmos iterativos

1.6 Analisando algoritmos recursivos