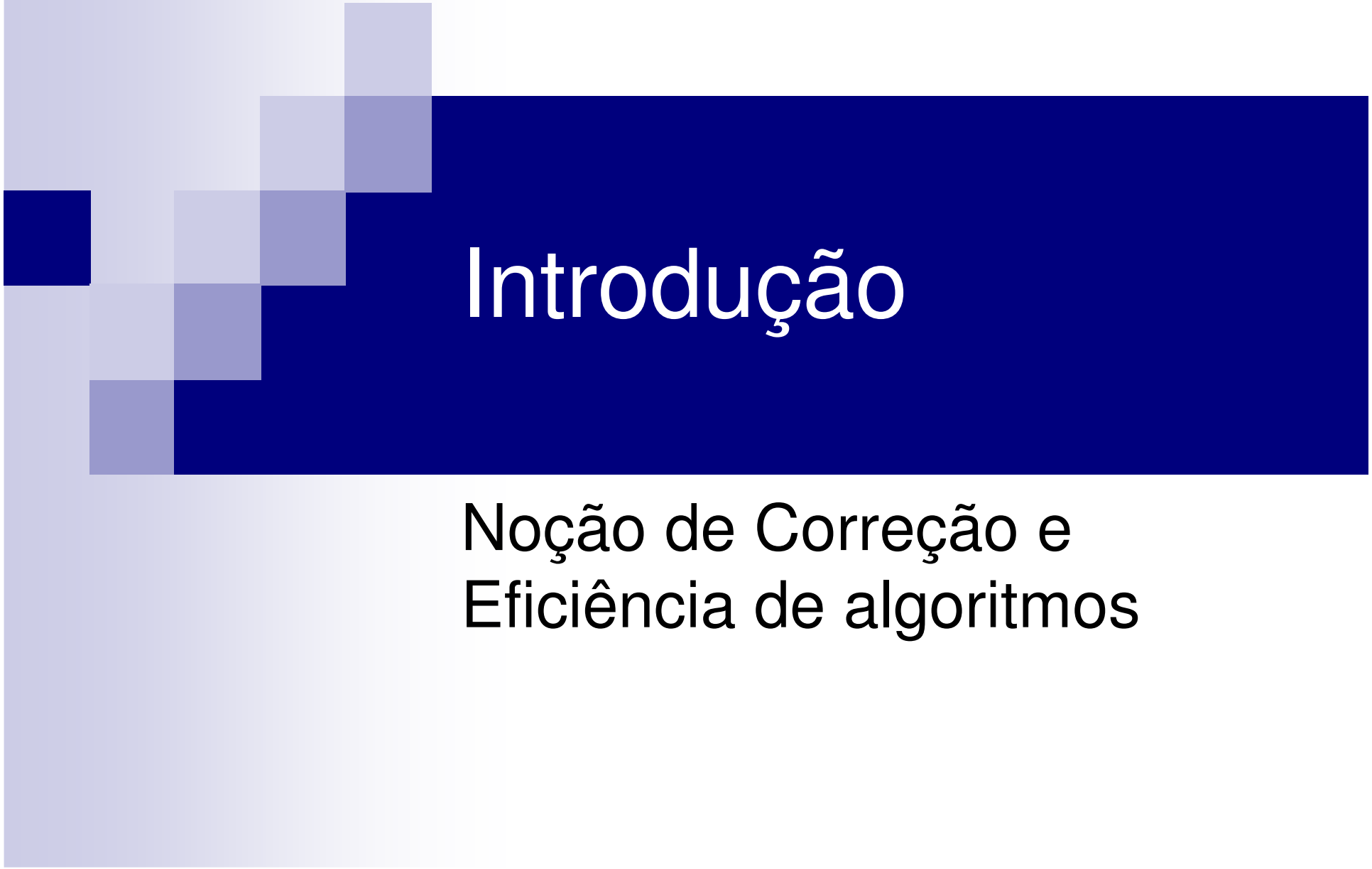

Introdução
Noção de Correção e
Eficiência de algoritmos

Lógica de Programação [2]

- Lógica é a arte de pensar corretamente. A lógica estuda a correção do raciocínio.

- Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo

- Estes pensamentos, podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa, isto é, uma Sequência lógica

Algoritmo

- Um algoritmo é formalmente uma **sequência finita de passos** que levam a execução de uma tarefa.

- Um algoritmo é semelhante a uma **receita**, isto é, uma sequência de instruções que permitem atingir uma meta específica.

- Esta sequência de instruções não podem ser ambíguas, redundantes nem subjetivas na sua definição, devem ser **claras** e **precisas**.

- Como exemplos de algoritmos podemos citar as **operações básicas** (multiplicação, divisão, adição e subtração) de números reais decimais.

- Em **ciência da computação** algoritmos são as instruções que podem ser utilizadas pelo computador para solucionar um problema [1]

Exemplo: Trocar uma lâmpada queimada [2]

- Desligue o interruptor;
- Pegue uma escada;
- Posicione-a embaixo da lâmpada;
- Busque uma lâmpada nova;
- Suba na escada;
- Retire a lâmpada velha;
- Coloque a lâmpada nova;
- Desça da escada;
- Descarte a lâmpada queimada;
- Guarde a escada.

Algoritmo

- Há várias maneiras diferentes de escrever o mesmo algoritmo.

- Cada pessoa tem seu jeito particular de pensar.
  - Probabilidade muito baixa de duas pessoas escreverem um algoritmo igual ou muito parecido para o mesmo problema.

Descrever Algoritmo [6]

- Podemos descrever um algoritmo de várias maneiras:
  - usando uma linguagem de programação de alto nível: C, Pascal, Java, etc.
  - implementando-o em linguagem de máquina diretamente executável em hardware.
  - em português.
  - em um pseudo-código de alto nível.

Exemplo de pseudo-código [6]

- Algoritmo ORDENA-POR-INSERÇÃO: rearranja um vetor A[1 . . . n] de modo que fique crescente.

```
ORDENA-POR-INSERÇÃO(A, n)
1   para j ← 2 até n faça
2       chave ← A[j]
3       ▷ Insere A[j] no subvetor ordenado A[1 . . . j−1]
4       i ← j − 1
5       enquanto i ≥ 1  e A[i] > chave faça
6           A[i + 1] ← A[i]
7           i ← i − 1
8       A[i + 1] ← chave
```

Analisar um algoritmo [1]

- Um algoritmo deve possuir diversas qualidades:
  - Correção
  - Eficiência
  - Simplicidade

- Existem dois tipos de eficiência: eficiência temporal e eficiência espacial.
  - **Eficiência temporal**: indica quão rápido um algoritmo é executado.
  - Eficiência espacial: está relacionado ao espaço extra que o algoritmo necessita.

Analisar um algoritmo [3]

- Métodos:
  - Análise empírica
  - Análise teórica
- Critérios:
  - Correção
  - Eficiência
  - Simplicidade

Métodos para Analisar um algoritmo [3]

- Abordagem experimental:
  - Implementar o algoritmo e executar o programa para um conjunto de dados testes.
  - **Problema**:
    - Não podem ser testadas todas as possíveis entradas;
    - Podem ser esquecidos alguns casos em que o algoritmo falha
    - Os resultados dependem dos aspectos de implementação

Métodos para Analisar um algoritmo [3]

- Abordagem teórica:
  - Estudar o algoritmo em termos gerais e tentar prever aspectos gerais do seu comportamento.
  - **Correção**: ele fornece uma solução válida para o problema?
  - **Eficiência**: quanto tempo ele gasta? Quanto de memória ele usa?

Corretude de um algoritmo

- A prova da corretude de um algoritmo consiste em mostrar que ele executa corretamente o processo desejado
- A corretude do algoritmo pode ser provada matematicamente.
  - Ex: utilizando Indução Matemática
- Corretude: Um algoritmo está correto se para toda entrada (legal) ele produz a saída correta.

Como provar a corretude de um algoritmo? [4]

- Os **testes de escritório** não são suficientes: servem apenas para mostrar que um algoritmo tem erros.

- Técnicas utilizadas:
  - Prova por indução
  - Invariantes de laço
  - Função recursiva (e o paradigma de divisão e conquista)

Indução Matemática [5]

- Técnica de prova poderosa e comumente usada em Ciência da Computação.

- Axioma da Indução:
  - Suponha que:
    - **P(0)** é verdadeiro
    - $\forall m \in N, P(m) \rightarrow P(m+1)$
  - Então:
    - $\forall n \in N$, P(n) é verdadeiro

Prova por Indução [5]

- O texto de uma prova por indução consiste de três partes:
  - Especificação da hipótese da indução:
    - P(n)
  - O caso base:
    - P(0)
  - O passo indutivo:
    - $\forall m \in N, P(m) \rightarrow P(m+1)$

Exemplo: Identidade de Gauss [5]

- Queremos provar que: $1 + 2 + \ldots + n = n.(n + 1)/2$
- Hipótese da Indução (H.I.):
  - A própria identidade
  - $I(n) = n.(n + 1)/2$
- Caso base:
  - $n = 1$
  - $I(1) = 1 = 1.(1 + 1)/2 = 1$
- Passo indutivo:
  - Provar que $I(n + 1) = (n + 1).(n + 2)/2$
  - Utilizar a H.I.

Invariantes de laço [7]

- **Definição**: uma invariante de um laço é uma propriedade que relaciona as variáveis do algoritmo a cada execução completa do laço [6].
- Método para provar a corretude de blocos de laço.
- A estratégia “típica” para mostrar a corretude de um algoritmo iterativo através de invariantes segue os seguintes passos:

  1. Mostre que o invariante vale no início da primeira iteração (trivial, em geral)
  2. Suponha que o invariante vale no início de uma iteração qualquer e prove que ele vale no início da próxima iteração
  3. Conclua que se o algoritmo pára e o invariante vale no início da última iteração, então o algoritmo é correto.

- Note que (1) e (2) implicam que o invariante vale no início de qualquer iteração do algoritmo. Isto é similar ao método de indução matemática ou indução finita.

Recursão e o paradigma de divisão-e-conquista [7]

- Um algoritmo recursivo encontra a saída para uma instância de entrada de um problema chamando a si mesmo para resolver instâncias menores deste mesmo problema.

- Algoritmos de divisão-e-conquista possuem três etapas em cada nível de recursão:
  - Divisão: o problema é dividido em subproblemas semelhantes ao problema original, porém tendo como entrada instâncias de tamanho menor.
  - Conquista: cada subproblema é resolvido recursivamente a menos que o tamanho de sua entrada seja suficientemente “pequeno”, quando este é resolvido diretamente.
  - Combinação: as soluções dos subproblemas são combinadas para obter uma solução do problema original.

Eficiência de Algoritmos

- Dado que os recursos de um computador são limitados é importante considerar o tempo e o espaço.

- Algoritmos desenvolvidos para solucionar um mesmo problema diferem em forma drástica quanto a sua eficiência [1]
  - Ex. Quicksort vs Bubblesort

Eficiência de Algoritmos – Ex [1]

- Dado 2 algoritmos de ordenação:
  - **Ordenação por inserção**: assumimos que o tempo de execução é $t=c_1 n^2$
  - **Ordenação por intercalação**: assumimos que o tempo de execução é $t = c_2 n \log_2 n$
  - Normalmente as constantes $c_1 < c_2$
  - Normalmente as constantes c_1 *e* c_2 são menos significativos que ***n***

    Note que não há uma relação direta entre o “tempo de execução” e o “tempo físico”

Eficiência de Algoritmos – Ex [1]

- Comparar um computador rápido “A” que executa a ordenação por inserção com um computador mais lento “B” que executa a ordenação por intercalação.

- Cada computador deve ordenar um arranjo de um milhão de números.

Computador “A” 1 bilhão op/sec

Computador “B” 10 milhões op/sec

Eficiência de Algoritmos – Ex [1]

- Utilizando um algoritmo cujo tempo de execução cresce mais lentamente, mesmo com um computador mais fraco, B funciona 20x mais rápido que A.

- O desempenho total de um sistema computacional depende da combinação de: *Algoritmos eficientes + Hardware Rápido*

- É importante lembrar que um algoritmo correto não é aquele que funciona na maioria dos casos, mas aquele que funciona corretamente para todas as entradas legítimas.

Computador “A” 2000 segundos

Computador “B” 100 segundos

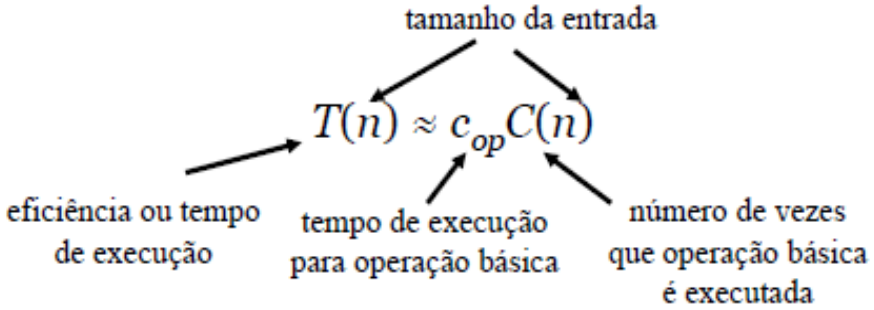
Eficiência de Algoritmos [3]

- **Tamanho da entrada**: a maioria dos algoritmos levam mais tempo para ser executados sobre entradas maiores.

- A **eficiência do algoritmo** é analisada em função do parâmetro de entrada ***n***, o qual indica o tamanho da entrada do algoritmo.

- A **eficiência temporal** de um algoritmo é analisada determinando o número de repetições de operações básicas com uma função do tamanho de entrada.

- A **operação básica** é aquela que contribui mais para o tempo de execução do algoritmo.



Exemplo tamanho de entrada e operação básica [3]

Problema	Medida do Tamanho da Entrada	Operação Básica
Busca por uma chave em uma lista de n itens	Número de itens na lista	Comparação de chaves
Multiplicação de duas matrizes de números de pontos flutuantes	Dimensões das matrizes	Multiplicação de Ponto Flutuante
Calcular a^n	n	Multiplicação de Ponto Flutuante
Grafos	Número de vértices e/ou arestas	Visitar um vértice ou atravessar uma aresta

Simplicidade de um algoritmo

- Um algoritmo é simples se puder ser facilmente: entendido, implementado e mantido.
  - Ex: há dificuldade de implementar o algoritmo?; tem problema em encontrar erros em uma implementação feita por um terceiro?

Resumo

- Um algoritmo é formalmente uma sequência finita de passos que levam a execução de uma tarefa.
- Podemos descrever algoritmos utilizando: linguagem natural (português), linguagens de programação, pseudo-código.
- Analisamos algoritmos em base a três critérios: corretude, eficiência e simplicidade.

Referências

- [1] Notas de aula: http://www.ppgia.pucpr.br/~alekoe/PAA/20061/1-NocoesFundamProblemas-PAA2006.pdf
- [2] MC102 Algoritmos e Programação de Computadores. L. G. Turatti and F. B. Valio and J. C. F. Cornacchia. 2009
- [3] Notas de aula: http://www.ppgia.pucpr.br/~alekoe/PAA/20061/2-NotacaoAssintotica-PAA2006.pdf
- [4] Notas de aula: http://www.dsc.ufcg.edu.br/~abrantes/CursosAnteriores/ATAL051/IntroducaoInformal.pdf
- [5] Notas de aula: http://www.dsc.ufcg.edu.br/~abrantes/CursosAnteriores/ATAL051/Corretude.pdf
- [6] Notas de aula: http://www.ic.unicamp.br/~rezende/ensino/mo417/2010s2/Slides/Aula1.pdf
- [7] Notas de aula: http://www.ic.unicamp.br/~rezende/ensino/mo417/2010s2/Slides/Aula2.pdf