# Análise e Técnicas de Algoritmos
# Período 2003.1

Corretude de Algoritmos

# Corretude de Algoritmos

## Introdução

- Como podemos saber se um algoritmo funciona?
  - Testes.
  - Prova de corretude.
- Testes: executar o algoritmo para algumas amostras de entradas.
- Provas de corretude: provar matematicamente.
- Testes podem não revelar erros não triviais.

## Indução Matemática

- É uma técnica bastante versátil de provas.
- Pode ser aplicada em muitos tipos de problemas.
- Para provar que uma propriedade é válida para qualquer $I\!N$:
  1. Caso Base: Provar que a propriedade é válida para 1.
  2. Provar que $\forall n \geq 1$, se a propriedade é válida para $n$, então ela é válida para $n + 1$.
- Variação 1:
  1. Caso base: Provar que a propriedade é válida para 1.
  2. Provar que $\forall n \geq 2$, se a propriedade é válida para $n - 1$, então ela é válida para $n$.
- Variação 2:
  1. Vários casos base: Provar que a propriedade é válida para 1, 2 e 3.
  2. Provar que $\forall n \geq 3$, se a propriedade é válida para $n$, então ela é válida para $n + 1$.
- Variação 3 (indução forte):
  1. Caso base: Provar que a propriedade é válida para 1.
  2. Provar que $\forall n \geq 1$, se a propriedade é válida para $\forall 1 \leq m \leq n$, então ela é válida para $n + 1$.

**Exemplos de Indução**

**Identidade de Gauss**

$\forall n \in I\!N$, $1 + 2 + ... + n = n(n+1)/2$.

**Parte 1**: Provar que a propriedade é válida para $n = 1$.

$1 = 1(1+1)/2$

**Parte 2**: Provar que se a propriedade é válida para $n$, então a propriedade é válida para $n+1$

Seja $S(n) = 1 + 2 + ... + n$.

Vamos assumir que $S(n) = n(n+1)/2$ (hipótese da indução).

Temos que provar que: $S(n+1) = (n+1)(n+2)/2$.

$S(n+1)$
$= S(n) + (n+1)$
$= n(n+1)/2 + (n+1)$ (pela hipótese da indução)
$= n^2/2 + n/2 + n + 1$
$= (n^2 + 3n + 2)/2$
$= (n+1)(n+2)/2$

**Árvore Binária Completa**

Uma árvore binária completa com $k$ níveis tem exatamente $2^k - 1$ nós.

**Parte 1**: Para $k = 1$, a propriedade é válida pois a árvore binária completa com um único nível possui um único nó.

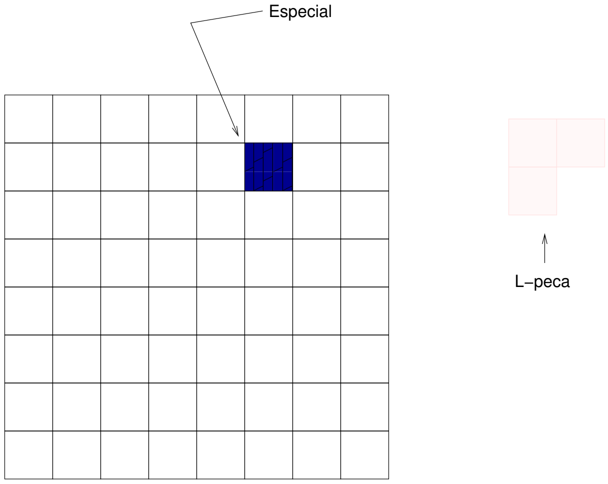
**Parte 2**: Assumindo que uma árvore binária completa com $k$ níveis possui $2^k - 1$ nós (hipótese da indução), temos que provar que uma árvore binária completa com $k+1$ níveis possui $2^{k+1} - 1$ nós.

Uma árvore binária completa com $k+1$ níveis é formada por um nó raiz e duas sub-árvores com $k$ níveis. Logo, pela hipótese da indução, o número total de nós é

$1 + 2(2^k - 1) = 2^{k+1} - 1$.

**Triomino**

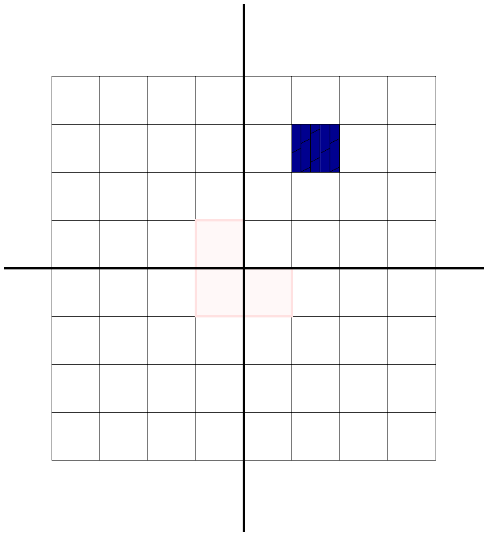
Considere um tabuleiro formado por quadrados de um mesmo tamanho (como no tabuleiro de xadrez). O tabuleiro tem $m$ quadrados em cada linha e $m$ quadrados em cada coluna. $m$ é potência de 2. Um dos quadrados é considerado especial, sendo diferenciado dos demais. Uma *L-peça* é semelhante a um tabuleiro $2 \times 2$, removendo-se um dos quadrados. É possível cobrir o tabuleiro usando *L-peças*, considerando que cada quadrado só é coberto uma única vez, com exceção do quadrado especial que não é coberto por nenhuma *L-peça*?




O problema do quebra-cabeça do tabuleiro é sempre resolvível. A prova utiliza indução matemática sobre o inteiro $n$, onde $m = 2^n$.

**Parte 1:** Caso base quando $n = 0$ (tabuleiro $1 \times 1$) ou $n = 1$ (tabuleiro $2 \times 2$).

**Parte 2:** Vamos considerar qualquer $n \geq 1$. A hipótese da indução é que o problema é resolvível para $n-1$, ou seja, o tabuleiro $2n - 1 \times 2n - 1$. Dividindo-se o tabuleiro $m \times m$ em 4 partes iguais, o quadrado especial vai ficar em um dos 4 sub-tabuleiros. Colocar uma *L-peça* no meio do tabuleiro original, de modo a ficar um quadrado especial em cada um dos sub-tabuleiros. O problema se reduziu a 4 sub-tabuleiros $2n - 1 \times 2n - 1$, cada um com um quadrado especial. Pela nossa hipótese da indução, cada um dos sub-tabuleiros pode ser resolvido. Logo, o tabuleiro original pode ser resolvido considerando uma *L-peça* no meio.



## Corretude de Algoritmos Recursivos

- Para provar a corretude de algoritmos recursivos:
  - Prová-lo por indução, considerando o tamanho do problema aser resolvido.
  - O caso base da recursão é o caso base da indução.
  - É necessário provar que as chamadas recursivas são subproblemas sem recursão infinita.
  - Assumir que as chamadas recursivas executam corretamente, e usar esse argumento para provar que a execução corrente é correta (passo indutivo).

**Exemplos**

**Números de Fibonacci (solução recursiva)**

$F_0 = 0$, $F_1 = 1$, e $\forall n \geq 2$, $F_n = F_{n-2} + F_{n-1}$.

**Algoritmo**:

**fib(n)**
**if** $n \leq 1$ **then**
  return $n$
**else**
  return $fib(n-1) + fib(n-2)$

Provar que $\forall n \geq 0$, $fib(n)$ retorna $F_n$.

**Caso base**: Para $n = 0$, $fib(n)$ retorna 0 ou $F_0$, como pretendido. Para $n = 1$, $fib(n)$ retorna 1 ou $F_1$, como pretendido.

**Hipótese da indução**: $\forall n \geq 2$ e $\forall 0 \leq m \leq n$, $fib(m)$ retorna $F_m$.

O que temos que provar: $fib(n)$ retorna $F_n$.

$fib(n)$ retorna $fib(n-1) + fib(n-2)$.

$fib(n-1) + fib(n-2)$

$= F_{n-1} + F_{n-2}$ (pela hipótese da indução)

$= F_n$.

**Maximum (solução recursiva)**

**Algoritmo**:

**maximum(n)**
**if** $n \leq 1$ **then**
  return $A[1]$
**else**
  return $max(maximum(n-1), A[n])$

Provar que $\forall n \geq 1$, $maximum(n)$ retorna $max\{A[1], A[2], ..., A[n]\}$.

**Caso base**: Para $n = 1$, $maximum(n)$ retorna $A[1]$, como pretendido.

**Hipótese da indução**: $n \geq 1$ e $maximum(n)$ retorna $max\{A[1], A[2], ..., A[n]\}$.

O que temos que provar: $maximum(n+1)$ retorna $max\{A[1], A[2], ..., A[n+1]\}$.

$maximum(n+1)$ retorna $max(maximum(n), A[n+1])$.

$max(maximum(n), A[n+1])$

$= max(max\{A[1], A[2], ..., A[n]\}, A[n+1])$ (pela hipótese da indução)

$= max\{A[1], A[2], ..., A[n+1]\}$.

## Corretude de Algoritmos Não-Recursivos

- Para provar a corretude de um algoritmo interativo:
  - Analisar um laço por vez no algoritmo, começando pelo laço mais interno no caso de aninhamento de laços.
  - Para cada laço determinar um *invariante de laço* que permanece verdadeiro em todas as interações do laço, e que captura o progresso do laço.
  - Provar que o invariante de laço é válido.
  - Usar os invariantes de laço para provar que o algoritmo termina.
  - Usar o invariante de laço para provar que o algoritmo computa o resultado correto

Considerar algoritmos com um laço.

O valor do identificador $x$ imediatamente após a $i$-ésima interação em um laço é denotada por $x_i$ ($i = 0$ indica o valor imediatamente antes de iniciar a primeira interação). Por exemplo, $x_6$ denota o valor do identificador $x$ após a sexta ocorrência do laço.

### Exemplos

### Números de Fibonacci (solução não-recursiva)

$F_0 = 0$, $F_1 = 1$, e $\forall n \geq 2$, $F_n = F_{n-2} + F_{n-1}$.

**Algoritmo**:

**fib(n)**
**if** $n = 0$ **then**
  return 0
**else**
  a = 0
  b = 1
  i = 2
  **while** $i \leq n$ **do**

c = a + b
a = b
b = c
i = i + 1
return $b$

Provar que $\forall n \geq 0$, $fib(n)$ retorna $F_n$.

**Fatos sobre o algoritmo**:

$i_0 = 2$
$i_{j+1} = i_j + 1$
$a_0 = 0$
$a_{j+1} = b_j$
$b_0 = 1$
$b_{j+1} = c_{j+1}$
$c_{j+1} = a_j + b_j$

**Invariante de Laço**:

Para todos os números naturais, $j \geq 0$, $i_j = j + 2$, $a_j = F_j$, e $b_j = F_{j+1}$.

A prova é por indução sobre $j$. O caso base, $j = 0$, é trivial, uma vez que $i_0 = 2$, $a_o = 0 = F_0$, e $b_0 = 1 = F_1$.

**Hipótese da indução**: $j \geq 0$, $i_j = j + 2$, $a_j = F_j$ e $b_j = F_{j+1}$.

O que temos que provar: $i_{j+1} = j + 3$, $a_{j+1} = F_{j+1}$ e $b_{j+1} = F_{j+2}$.

$i_{j+1} = i_j + 1$
$= (j + 2) + 1$ (pela hipótese da indução)
$= j + 3$.
$a_{j+1} = b_j$
$= F_{j+1}$ (pela hipótese da indução)
$b_{j+1} = c_{j+1}$
$= a_j + b_j$
$= F_j + F_{j+1}$ (pela hipótese da indução)
$= F_{j+2}$.

**Prova de Corretude**:

Provar que o algoritmo termina com $b$ contendo $F_n$.

Para $n = 0$ está OK. Se $n > 0$, então nós entramos no laço.

Uma vez que $i_{j+1} = i_j + 1$, eventualmente $i$ será igual a $n + 1$ e o laço vai terminar. Vamos supor que isto acontece depois de $t$ interações. Uma vez que $i_t = n + 1$ e $i_t = t + 2$, podemos concluir que $t = n - 1$.

Pelo invariante de laço, $b_t = F_{t+1} = F_n$.