

# **INE5403**

## **FUNDAMENTOS DE MATEMÁTICA DISCRETA PARA A COMPUTAÇÃO**

PROF. DANIEL S. FREITAS

UFSC - CTC - INE

# 3 - INDUÇÃO E RECURSÃO

3.1) Indução Matemática

3.2) Indução Forte

3.3) Definições Recursivas

3.4) Indução Estrutural

3.5) Algoritmos Recursivos

# DEFINIÇÕES RECURSIVAS

- Algumas vezes pode ser difícil definir um objeto explicitamente, mas pode ser fácil defini-lo recursivamente.
  - Incluindo o item que está sendo definido como parte da definição.
- A recursão pode ser usada para definir sequências, funções e conjuntos.
- Exemplo: uma sequência de potências de 2 é dada por:

$$a_n = 2^n, \text{ para } n = 0, 1, 2$$

- mas ela também pode ser definida a partir do 1o termo e de uma regra para encontrar um termo da sequência a partir do anterior, ou seja:

$$a_0 = 1$$

$$a_{n+1} = 2.a_n, \text{ para } n = 0, 1, 2, \dots$$

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- Quando definimos uma sequência recursivamente, podemos usar **indução** para provar resultados sobre a sequência.
- Quando definimos **um conjunto** recursivamente:
  - especificamos alguns elementos iniciais em um passo básico e
  - fornecemos uma **regra** para construir **novos elementos** a partir que já temos no passo recursivo.
- Para provar resultados sobre **conjuntos definidos recursivamente**, utilizamos um método chamado de **indução estrutural**.

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- A **definição recursiva de uma função** cujo domínio é o conjunto dos inteiros não-negativos consiste em duas etapas:
  - Passo básico: especificar o valor da função em zero.
  - Passo recursivo: fornecer uma regra para encontrar o valor da função em um inteiro a partir dos seus valores em inteiros menores.
- Esta definição é chamada de **recursiva** ou ainda de **indutiva**.

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

● **Exemplo:** Suponha que  $f$  é definida **recursivamente** por:

$$f(0) = 3$$

$$f(n + 1) = 2.f(n) + 3$$

Encontre  $f(1)$ ,  $f(2)$ ,  $f(3)$  e  $f(4)$ .

**Solução:**

$$f(1) = 2f(0) + 3 = 2.3 + 3 = 9$$

$$f(2) = 2f(1) + 3 = 2.9 + 3 = 21$$

$$f(3) = 2f(2) + 3 = 2.21 + 3 = 45$$

$$f(4) = 2f(3) + 3 = 2.45 + 3 = 93$$

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- Muitas funções podem ser estudadas recursivamente.
  - Um bom exemplo é a função fatorial.
- **Exemplo:** Forneça uma definição recursiva para a função fatorial  $F(n) = n!$  e use-a para avaliar  $5!$

## Solução:

- Definição recursiva:
  - $F(0) = 1$
  - $F(n + 1) = (n + 1)F(n)$
- Avaliando  $F(5) = 5!$ :
$$\begin{aligned} F(5) &= 5.F(4) = 5.4.F(3) = 5.4.3.F(2) = \\ &= 5.4.3.2.F(1) = \\ &= 5.4.3.2.1.F(0) = 5.4.3.2.1.1 = 120 \end{aligned}$$

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- O Princípio da **indução matemática** garante que funções definidas recursivamente ficam **bem definidas**:
  - para todo inteiro positivo, o valor da função neste inteiro é determinado de forma não ambígua
  - ou seja, obtemos o **mesmo valor** qualquer que seja o modo de aplicar as duas partes da definição
- Em algumas definições de funções, os valores da função **nos primeiros  $k$**  inteiros positivos são especificados.
  - E então é fornecida uma regra para determinar o valor da função em inteiros maiores **a partir dos seus valores** em alguns ou todos os inteiros que o precedem.
  - O princípio da indução forte garante que tais definições produzem funções bem definidas.



# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- Os **números de Fibonacci**,  $f_0, f_1, f_2, \dots$ , são definidos pelas equações:

$$f_0 = 0, \quad f_1 = 1$$

$$f_n = f_{n-1} + f_{n-2}, \quad \text{para } n = 2, 3, 4, \dots$$

- **Exemplo:** encontre os números de Fibonacci  $f_2, f_3, f_4, f_5$  e  $f_6$ .

**Solução:** segue da segunda parte da definição que:

$$f_2 = f_1 + f_0 = 1 + 0 = 1$$

$$f_3 = f_2 + f_1 = 1 + 1 = 2$$

$$f_4 = f_3 + f_2 = 2 + 1 = 3$$

$$f_5 = f_4 + f_3 = 3 + 2 = 5$$

$$f_6 = f_5 + f_4 = 5 + 3 = 8$$

# INDUÇÃO & RECURSÃO

- Existe uma conexão natural entre recursão e indução:
  - É comum ser usada uma **sequência natural** em definições recursivas de objetos.
  - É comum a indução ser o melhor (talvez o único) modo de provar resultados sobre objetos definidos recursivamente.

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- Pode-se usar a **definição recursiva** dos números de Fibonacci para **provar muitas propriedades** destes números.
- **Exemplo:** Mostre que, sempre que  $n \geq 3$ , temos que  $f_n > \alpha^{n-2}$ , onde  $\alpha = (1 + \sqrt{5})/2$ .

**Solução:** podemos provar esta desigualdade usando indução forte:

- Seja  $P(n)$ : “ $f_n > \alpha^{n-2}$ ”
- Queremos provar que  $P(n)$  é V sempre que  $n \geq 3$ .
- Passo básico:
  - $2 = f_3 > \alpha$
  - $3 = f_4 > \alpha^2 = (3 + \sqrt{5})/2$
- de modo que  $P(3)$  e  $P(4)$  são ambas V.

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

● **Exemplo (cont.):**  $f_n > \alpha^{n-2}$ , para  $n \geq 3$

**Solução:**

● Passo indutivo:

● vamos assumir que  $P(j)$  é V, ou seja:

$$f_j > \alpha^{j-2}, \quad \forall j, \text{ com } 3 \leq j \leq k, \text{ onde } k \geq 4$$

● (Temos que mostrar que  $P(k+1)$  é V, ou seja:  $f_{k+1} > \alpha^{k-1}$ )

● Como  $\alpha$  é uma solução de  $x^2 - x - 1 = 0$ , temos  $\alpha^2 = \alpha + 1$

● Portanto:

$$\begin{aligned} \alpha^{k-1} &= \alpha^2 \cdot \alpha^{k-3} \\ &= (\alpha + 1) \alpha^{k-3} \\ &= \alpha \cdot \alpha^{k-3} + 1 \cdot \alpha^{k-3} \\ &= \alpha^{k-2} + \alpha^{k-3} \end{aligned}$$

● Pela hipótese indutiva, se  $k \geq 4$ , segue que:

$$f_{k+1} = f_k + f_{k-1} > \alpha^{k-2} + \alpha^{k-3} = \alpha^{k-1}$$

□

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- **Exemplo:** Considere a seguinte definição recursiva da função fatorial:

$$1! = 1$$

$$n! = n(n - 1)!, \quad n > 1$$

Queremos provar que:  $\forall n \geq 1, n! \geq 2^{n-1}$

**Solução:** podemos provar esta desigualdade usando indução forte.

- Seja  $P(n)$ : “ $n! \geq 2^{n-1}$ ”
- Passo básico:
  - $P(1)$  é a proposição  $1! \geq 2^0$
  - o que é V, já que  $1! = 1$

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- **Exemplo (cont.):** Provar que:  $\forall n \geq 1, n! \geq 2^{n-1}$

**Solução:**

- Passo indutivo:

- Queremos provar que  $P(k) \Rightarrow P(k+1)$  é uma tautologia.
- Suponha que  $k! \geq 2^{k-1}$ , para algum  $k \geq 1$
- Daí, **pela definição recursiva**, o lado esquerdo de  $P(k+1)$  é:

$$\begin{aligned}(k+1)! &= (k+1)k! \\ &\geq (k+1)2^{k-1} && \text{usando } P(k) \\ &\geq 2 \times 2^{k-1} && k+1 \geq 2, \text{ pois } k \geq 1 \\ &= 2^k && \text{lado direito de } P(k+1)\end{aligned}$$

- Portanto,  $P(k+1)$  é V □

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- Vamos agora mostrar que o algoritmo de Euclides usa  $O(\log b)$  divisões para obter o mdc dos inteiros positivos  $a$  e  $b$  (onde  $a \geq b$ ).
  - Nota (princípio do algoritmo):  $mdc(a, b) = mdc(b, a \bmod b)$
- Para isto, vamos precisar do resultado a seguir.
- **Teorema de Lamé:** Se  $a$  e  $b$  são inteiros positivos com  $a \geq b$ , o número de divisões usado pelo algoritmo de Euclides para encontrar  $mdc(a, b)$  é  $\leq$  a 5 vezes o número de dígitos decimais em  $b$ .

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- **Teorema de Lamé:** “nro de divisões no algoritmo de Euclides para  $\text{mdc}(a, b)$  é  $\leq$  a 5 X nro de dígitos decimais em  $b$ ”.

- **Prova (1/3):**

- uma aplicação do algoritmo ( $a = r_0$  e  $b = r_1$ ):

$$r_0 = r_1 q_1 + r_2 \qquad 0 \leq r_2 < r_1$$

$$r_1 = r_2 q_2 + r_3 \qquad 0 \leq r_3 < r_2$$

...

$$r_{n-2} = r_{n-1} q_{n-1} + r_n \qquad 0 \leq r_n < r_{n-1}$$

$$r_{n-1} = r_n q_n$$

- note que:

- $n$  divisões foram usadas para chegar a  $r_n = \text{mdc}(a, b)$
- $q_1, q_2, \dots, q_{n-1}$  são todos  $\geq 1$
- $q_n \geq 2$ , pois  $r_n < r_{n-1}$



# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

● **Teorema de Lamé:** “**nro de divisões** no algoritmo de Euclides para  $\text{mdc}(a, b)$  é  $\leq$  **a 5 X nro de dígitos decimais em  $b$** ”.

● **Prova (2/3):**

● em uma aplicação do algoritmo

●  $n$  divisões usadas para chegar a  $r_n = \text{mdc}(a, b)$

● todos os  $q_i \geq 1$ , mas  $q_n \geq 2$  (pois  $r_n < r_{n-1}$ )

● o que permite escrever:

$$r_n \geq 1 = f_2,$$

$$r_{n-1} \geq 2r_n \geq 2f_2 = f_3,$$

$$r_{n-2} \geq r_{n-1} + r_n \geq f_3 + f_2 = f_4,$$

$$\vdots$$

$$r_2 \geq r_3 + r_4 \geq f_{n-1} + f_{n-2} = f_n,$$

$$b = r_1 \geq r_2 + r_3 \geq f_n + f_{n-1} = f_{n+1}.$$

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

● **Teorema de Lamé:** “**nro de divisões** no algoritmo de Euclides para  $\text{mdc}(a, b)$  é  **$\leq a$  5 X nro de dígitos decimais em  $b$** ”.

● **Prova (3/3):**

● logo, se  $n$  divisões são usadas pelo algoritmo:

● temos que:  $b \geq f_{n+1}$

● mas já sabemos que:  $f_{n+1} > \alpha^{n-1}$  (para  $n > 2$ )

● logo:  $b > \alpha^{n-1}$

$$\Rightarrow \log_{10} b > (n-1)/5$$

$$\Rightarrow n-1 < 5 \cdot \log_{10} b$$

● agora suponha que  $b$  tem  $k$  dígitos decimais:

● então:  $b < 10^k \Rightarrow k = \lfloor \log_{10} b \rfloor + 1$

● de modo que:  $\log_{10} b < k \leq \log_{10} b + 1$

● segue que:  $n-1 < 5k \Rightarrow n \leq 5k$

□

# DEFINIÇÕES RECURSIVAS DE FUNÇÕES

- Voltando à demonstração de que o algoritmo de Euclides utiliza  $O(\log b)$  divisões para encontrar o  $\text{mdc}(a, b)$ :
  - Pelo teorema de Lamé, sabemos que:
    - nro de divisões para obter  $\text{mdc}(a, b) \leq 5(\log_{10} b + 1)$
  - Ou seja:
    - $O(\log b)$  divisões são necessárias para encontrar  $\text{mdc}(a, b)$  pelo algoritmo de Euclides (com  $a > b$ ).  $\square$

# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

- Definições recursivas de conjuntos também têm duas partes:
  - **Passo básico:** uma coleção inicial de elementos é especificada.
  - **Passo recursivo:** regras para formar novos elementos a partir daqueles que já se sabe que estão no conjunto.
- Definições recursivas também podem incluir uma **regra de extensão:**
  - estipula que um conjunto definido recursivamente não contém nada mais do que:
    - os elementos especificados no passo básico
    - ou gerados por aplicações do passo indutivo
  - assumiremos que esta regra sempre vale.

# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

- **Exemplo:** Considere o subconjunto  $S$  dos inteiros definido por:
  - Passo básico:  $3 \in S$
  - Passo indutivo: se  $x \in S$  e  $y \in S$ , então  $x + y \in S$
- Elementos que **estão em  $S$** :
  - 3 (passo básico)
  - aplicando o passo indutivo:
    - $3 + 3 = 6$  (1ra aplicação)
    - $3 + 6 = 6 + 3 = 9$  e  $6 + 6 = 12$  (2da aplicação)
    - etc...
  - Mostraremos mais tarde que  $S$  é o conjunto de **todos os múltiplos positivos de 3**.

# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

- Definições recursivas são muito importantes no estudo de **strings**.
- **String** sobre um **alfabeto**  $\Sigma$ : sequência finita de **símbolos de  $\Sigma$** .
- O conjunto  $\Sigma^*$ , de **strings sobre o alfabeto  $\Sigma$**  pode ser definido por:
  - **Passo básico:**  $\lambda \in \Sigma^*$  (contém a string vazia)
  - **Passo recursivo:** se  $w \in \Sigma^*$  e  $x \in \Sigma$ , então  $wx \in \Sigma^*$
- O passo recursivo estabelece que:
  - novas strings são produzidas pela adição de **um símbolo de  $\Sigma$  ao final** das strings **já em  $\Sigma^*$**
  - a cada aplicação do passo recursivo, são geradas strings contendo **um símbolo a mais**.

# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

● **Exemplo:** se  $\Sigma = \{0, 1\}$ :

●  $\Sigma^*$  é o conjunto de todas as **strings de bits**

● Strings que **estão em  $\Sigma^*$** :

●  $\lambda$

● 0 e 1 ( $1^a$  aplicação do passo recursivo)

● 00, 01, 10, 11 (após  $2^a$  aplicação do passo recursivo)

● etc...

# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

- Definições recursivas podem ser usadas para definir operações ou funções sobre os elementos de **conjuntos definidos recursivamente**.
  - Exemplificado na **combinação de duas strings** mostrada a seguir.
- Sejam:
  - $\Sigma$  um conjunto de símbolos
  - $\Sigma^*$  o conjunto das strings formadas com símbolos de  $\Sigma$ .
  - A **concatenação** de duas strings ( $\cdot$ ) é definida como:
    - passo básico: se  $w \in \Sigma^*$ , então  $w \cdot \lambda = w$
    - passo recursivo: se  $w_1 \in \Sigma^*$  e  $w_2 \in \Sigma^*$  e  $x \in \Sigma$ , então:

$$w_1 \cdot (w_2 x) = (w_1 \cdot w_2) x$$



# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

- Uma aplicação repetida da definição recursiva mostra que:
  - a concatenação de duas strings  $w_1$  e  $w_2$  consiste dos **símbolos em  $w_1$  seguidos pelos símbolos em  $w_2$** .

• **Exemplo:** concatenação de  $ab$  e  $cde$ :

- $$\begin{aligned}(ab) \cdot (cde) &= (ab \cdot cd)e \\ &= (ab \cdot c)de \\ &= (ab \cdot \lambda)cde \\ &= abcde\end{aligned}$$

□

# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

- **Exemplo:** Forneça uma definição recursiva de  $l(w)$ , o comprimento de uma string  $w$

**Solução:**

- $l(\lambda) = 0;$

- se  $w \in \Sigma^*$  e  $x \in \Sigma$ :

- $l(wx) = l(w) + 1$

□

# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

- Um outro importante exemplo do uso de definições recursivas é na definição “fórmulas bem formadas” (FBFs) de vários tipos.
- Exemplo:** FBFs para **formatos de proposições compostas**:
  - envolvem **V**, **F** e:
    - variáveis proposicionais
    - operadores do conjunto:  $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ .
  - e são definidas como:
    - Passo básico:** **V**, **F**, e  $p$  (**uma** variável proposicional), **são fórmulas bem formadas**.
    - Passo recursivo:** se  $E$  e  $F$  já são fórmulas bem formadas, então também o serão:  
 $(\neg E)$ ,  $(E \wedge F)$ ,  $(E \vee F)$ ,  $(E \rightarrow F)$ , e  $(E \leftrightarrow F)$

# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

- Pelo passo básico, sabemos que:

$\mathbf{V}$ ,  $\mathbf{F}$ ,  $p$  e  $q$  são fórmulas bem formadas.

- Por uma aplicação inicial do passo recursivo:

$(p \vee q)$ ,  $(p \rightarrow \mathbf{F})$ ,  $(\mathbf{F} \rightarrow q)$  e  $q \wedge \mathbf{F}$  são fórmulas bem formadas.

- Uma 2ª aplicação do passo recursivo mostra que são FBFs:

$((p \vee q) \rightarrow q \wedge \mathbf{F}))$

$q \vee (p \vee q)$

$((p \rightarrow \mathbf{F}) \rightarrow \mathbf{V})$

- Note que **não são** fórmulas bem formadas:

$p \neg \wedge q$ ,  $pq \wedge$  e  $\neg \wedge pq$

# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

● **Exemplo:** FBFs para operadores e operandos:

● envolvem:

● variáveis, numerais

● operadores do conjunto  $\{+, -, *, /, \uparrow\}$

● e são definidas como:

● **Passo básico:**  $x$  é uma FBF se  $x$  é um número ou variável.

● **Passo recursivo:** se  $F$  e  $G$  já são fórmulas bem formadas, então também o serão:

$$(F + G), (F - G), (F * G), (F / G), \text{ e } (F \uparrow G)$$

# CONJS. E ESTRUTURAS DEFINIDOS RECURSIVAMENTE

- Pelo passo básico, sabemos que:

$x, y, 0$  e  $3$  são fórmulas bem formadas.

- FBFs geradas por uma aplicação do passo recursivo incluem:

$(x + 3), (3 + y), (x - y), (3 - 0), (x * 3), (3 * y)$

$(3/0), (x/y), (3 \uparrow x)$  e  $(0 \uparrow 3)$

- Uma 2ª aplicação do passo recursivo mostra que são FBFs:

$((x + 3) + 3)$  e  $(x - (3 * y))$

- Note que **não são** fórmulas bem formadas:

$x3+, y * +x$  e  $*x/y$

- (não podem ser obtidas usando: passo básico + aplicações do recursivo)