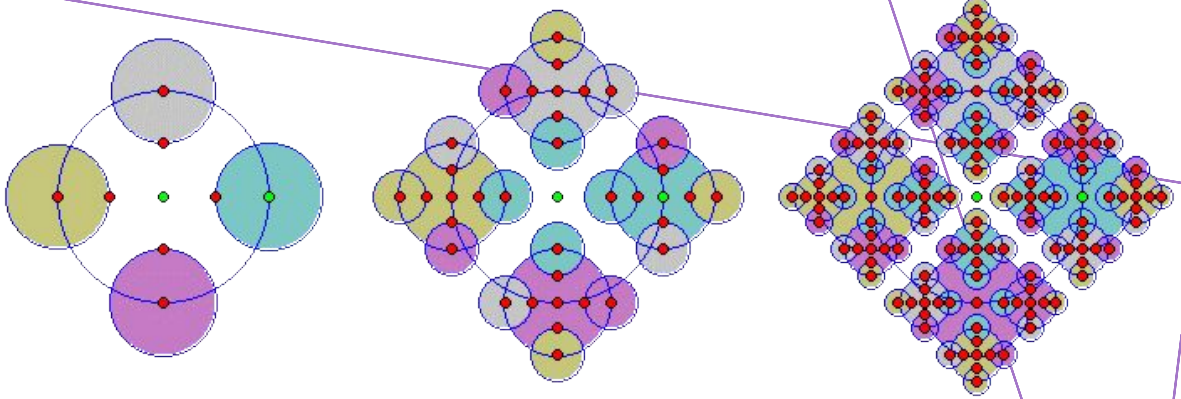
Fundamentos de Teoria da Computação Aula 11

PRISCILA MARQUES KAI

AGENDA DA AULA

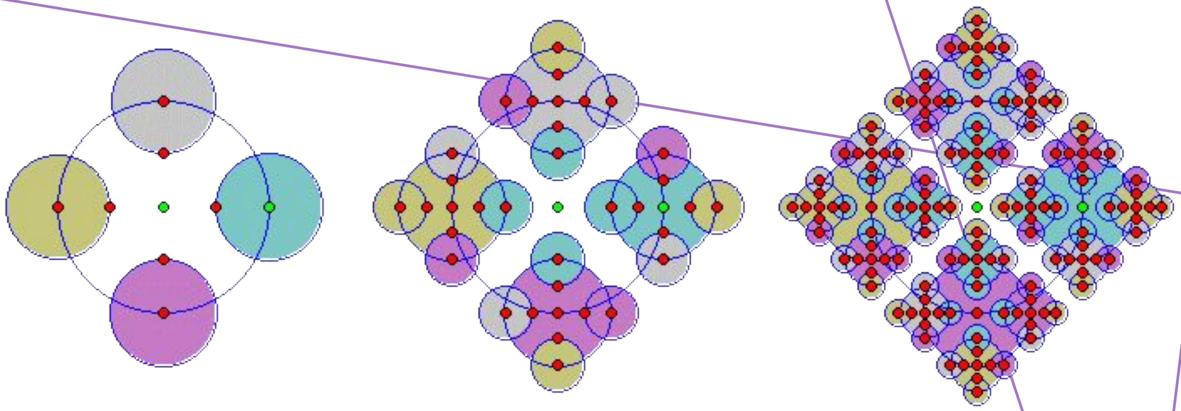


Recursividade e Recorrência

- Ao citarmos algoritmos, a recursividade se refere a um processo que se define a partir de si mesmo.

  Por exemplo, em algoritmos, a recursão existe quando um algoritmo invoca a si mesmo para resolver um problema, dividindo o problema em subproblemas menores.

RECURSIVIDADE E RECORRÊNCIA



Definição: O item que está sendo definido aparece como parte da própria definição

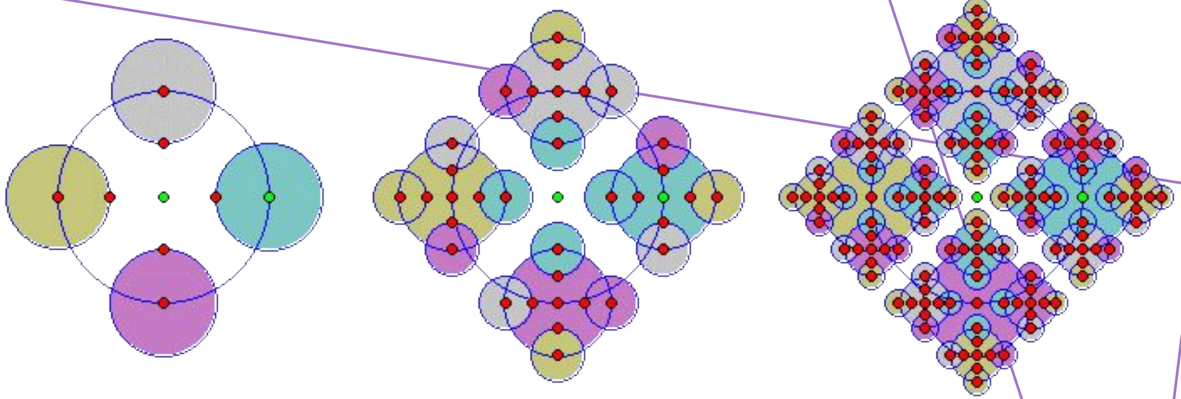
- **Definição recorrente** ou **definição por recorrência** ou ainda **definição recursiva**.

- Isso funciona porque uma definição recorrente tem duas partes:

  1) Uma base, ou condição básica, em que alguns casos simples do item que está sendo definido são dados explicitamente.

  2) Um passo indutivo ou de recorrência, em que novos casos do item que está sendo definido são dados em função de casos anteriores.

RECURSIVIDADE E RECORRÊNCIA



Definição: O item que está sendo definido aparece como parte da própria definição

- **Definição recorrente** ou **definição por recorrência** ou ainda **definição recursiva**.

- A recorrência está intimamente ligada a Indução Matemática
- **Recorrência** (ou **recursividade**) é uma ideia importante que pode ser usada para definir sequências de objetos, coleções mais gerais de objetos e operações com objetos.

RECURSIVIDADE E RECORRÊNCIA

SEQUÊNCIAS DEFINIDAS POR RECORRÊNCIA

Uma sequência S (uma sequência infinita) é uma lista de objetos que são numerados em determinada ordem; existem um primeiro objeto, um segundo e assim por diante. S(k) denota o k-ésimo objeto na sequência. A lista não termina, de modo que uma sequência consiste em

$$S(1),\ S(2),\ \ldots,\ S(k),\ \ldots$$

Muitas vezes são usados índices para denotar os elementos de uma sequência, como

$$S1,\ S2,\ \ldots,\ Sk,\ \ldots$$

A letra S é uma "variável muda", de modo que a sequência também poderia ser representada como

$$a1,\ a2,\ \ldots,\ ak,\ \ldots \text{ ou } w1,\ w2,\ \ldots,\ wk\ \ldots$$

e assim por diante.

Uma sequência é definida por recorrência nomeando-se, explicitamente, o primeiro valor (ou alguns poucos primeiros valores) na sequência e depois definindo valores subsequentes na sequência em termos de valores anteriores.

RECURSIVIDADE E RECORRÊNCIA

Considere a sequência S definida por recorrência por

1. $S(1) = 2$
2. $S(n) = 2S(n-1)$ para $n \geq 2$

Pela proposição 1, $S(1)$, o primeiro objeto em S, é 2. Depois, pela proposição 2, o segundo objeto em S é $S(2) = 2S(1) = 2(2) = 4$. Novamente pela proposição 2, $S(3) = 2S(2) = 2(4) = 8$. Continuando desse modo, vemos que a sequência S é

$$2, 4, 8, 16, 32, \ldots$$

Uma regra como a da proposição 2 no exemplo, que define um valor de uma sequência em termos de um ou mais valores anteriores, é chamada de uma **relação de recorrência**.

RECURSIVIDADE E RECORRÊNCIA

PROBLEMA PRÁTICO 1	A sequência T é definida por recorrência por:

1. $T(1) = 1$
2. $T(n) = T(n - 1) + 3$ para $n \geq 2$

Escreva os cinco primeiros valores da sequência T. ■

RECURSIVIDADE E RECORRÊNCIA

EXEMPLO 2

A **sequência de Fibonacci**, introduzida no século XIII por um comerciante e matemático italiano, é definida por recorrência por

$$F(1) = 1$$
$$F(2) = 1$$
$$F(n) = F(n-2) + F(n-1) \text{ para } n > 2$$

Aqui são dados os dois primeiros valores da sequência, e a relação de recorrência define o n-ésimo valor para $n > 2$ em termos dos dois valores precedentes. É melhor pensar na relação de recorrência em sua forma mais geral, que diz que F em qualquer valor — exceto em 1 e 2 — é a soma de F em seus dois valores anteriores. ●

PROBLEMA PRÁTICO 2	Escreva os oito primeiros valores da sequência de Fibonacci. ■

RECURSIVIDADE E RECORRÊNCIA

A sequência de Fibonacci é famosa por causa de suas propriedades interessantes. Eis uma lista curta (sem demonstrações):

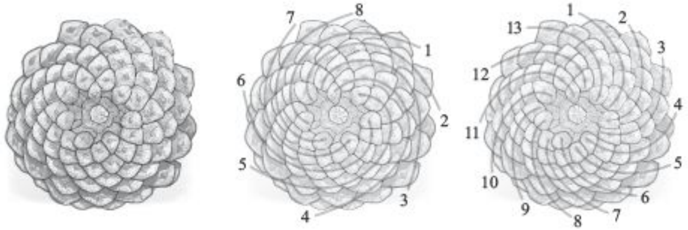
a. Todo inteiro positivo pode ser escrito de maneira única como a soma de um ou mais números de Fibonacci distintos não consecutivos. Por exemplo, $11 = 3 + 8$, em que $3 = F(4)$ e $8 = F(6)$.

b. $\text{mdc}(F(p), F(q)) = F(\text{mdc}(p, q))$. Por exemplo, se $p = 6$ e $q = 9$, então $F(6) = 8$, $F(9) = 34$ e $\text{mdc}(8, 34) = 2$. Por outro lado, $\text{mdc}(6, 9) = 3$ e $F(3) = 2$.

c. Dois números de Fibonacci consecutivos são primos entre si, ou seja, o máximo divisor comum entre eles é 1. Em consequência, o algoritmo de Euclides executa a quantidade máxima de operações para encontrar o $\text{mdc}(a, b)$ quando a e b são dois números de Fibonacci consecutivos.

RECURSIVIDADE E RECORRÊNCIA

Fibonacci. Os números de Fibonacci ocorrem com frequência na natureza. Muitas vezes, o número de pétalas em uma margarida é um número de Fibonacci. Olhando uma pinha, as sementes parecem estar arrumadas em espirais no sentido horário e no sentido anti-horário. Contando o número de cada tipo de espiral, frequentemente se obtêm dois números de Fibonacci consecutivos (8 e 13 aqui). O mesmo ocorre nas sementes de flores como o girassol ou nas espirais nos abacaxis.



E em arte e arquitetura, considera-se que a razão áurea cria proporções esteticamente agradáveis. A *razão áurea* é

$$\frac{1+\sqrt{5}}{2} \approx 1{,}6180339$$

e seu valor pode ser aproximado pela razão entre dois números de Fibonacci consecutivos $F(n+1)/F(n)$, com precisão melhor para valores cada vez maiores de n.

RECURSIVIDADE E RECORRÊNCIA

EXEMPLO 3

Prove que, na sequência de Fibonacci,

$$F(n+4) = 3F(n+2) - F(n) \text{ para todo } n \geq 1$$

Como queremos provar que alguma coisa é verdadeira para todo $n \geq 1$, é natural pensar em uma demonstração por indução. E como o valor de $F(n)$ depende de $F(n-1)$ e de $F(n-2)$, devemos usar o segundo princípio de indução. Para a base da indução, vamos provar dois casos, $n = 1$ e $n = 2$. Para $n = 1$, obtemos

$$F(5) = 3F(3) - F(1)$$

ou (usando os valores calculados no Problema Prático 2)

$$5 = 3(2) - 1$$

que é verdade. Para $n = 2$,

$$F(6) = 3F(4) - F(2)$$

ou

$$8 = 3(3) - 1$$

RECURSIVIDADE E RECORRÊNCIA

que também é verdade. Suponha que, para todo r, $1 \leq r \leq k$,

$$F(r+4) = 3F(r+2) - F(r).$$

Vamos mostrar o caso $k+1$, em que $k+1 \geq 3$. (Já provamos os casos $n=1$ e $n=2$.) Queremos mostrar, então, que

$$F(k+1+4) \stackrel{?}{=} 3F(k+1+2) - F(k+1)$$

ou

$$F(k+5) \stackrel{?}{=} 3F(k+3) - F(k+1)$$

Da relação de recorrência para a sequência de Fibonacci, temos

$F(k+5) = F(k+3) + F(k+4)$ (F em qualquer valor é a soma de F nos dois valores anteriores)

e, pela hipótese de indução, com $r = k-1$ e $r = k$, respectivamente, temos

$$F(k+3) = 3F(k+1) - F(k-1)$$

e

$$F(k+4) = 3F(k+2) - F(k)$$

RECURSIVIDADE E RECORRÊNCIA

Portanto,

$$
\begin{aligned}
F(k+5) &= F(k+3) + F(k+4) \\
&= [3F(k+1) - F(k) - 1)] + [3F(k+2) - F(k)] \\
&= 3[F(k+1) + F(k+2)] - [F(k-1) + F(k)] \\
&= 3F(k+3) - F(k+1) \qquad \text{(usando novamente a relação de recorrência)}
\end{aligned}
$$

Isso completa a demonstração por indução.

CONJUNTOS DEFINIDOS POR RECORRÊNCIA

Os objetos em uma sequência são ordenados — existem um primeiro objeto, um segundo e assim por diante.

Um conjunto de objetos é uma coleção na qual não há nenhuma ordem imposta.

Alguns conjuntos podem ser definidos por recorrência.

CONJUNTOS DEFINIDOS POR RECORRÊNCIA

Ao tratarmos de lógica proposicional, notamos que certas cadeias de letras de proposição, conectivos lógicos e parênteses, tais como $(A \wedge B)' \vee C$, são consideradas legítimas, enquanto outras, como $\wedge\wedge A'' B$, não o são. A sintaxe para arrumar tais símbolos constitui a definição do conjunto de fórmulas proposicionais bem formuladas e é uma definição por recorrência.

1. Qualquer letra de proposição é uma fbf.
2. Se P e Q são fbfs, então $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$, (P'), e $(P \leftrightarrow Q)$ também o são.

Usando as prioridades para os conectivos lógicos estabelecidas, podemos omitir os parênteses quando isso não causar confusão. Assim, podemos escrever $(P \vee Q)$ como $P \vee Q$, ou (P') como P'.

CONJUNTOS DEFINIDOS POR RECORRÊNCIA

1. Qualquer letra de proposição é uma fbf.
2. Se P e Q são fbfs, então $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$, (P'), e $(P \leftrightarrow Q)$ também o são.

Começando com letras de proposição e usando, repetidamente, a regra 2, podemos construir todas as fbfs proposicionais. Por exemplo, A, B e C são fbfs pela regra 1. Pela regra 2,

$$(A \wedge B) \text{ e } (C')$$

são, ambas, fbfs. Novamente pela regra 2,

$$((A \wedge B) \rightarrow (C'))$$

é uma fbf. Aplicando a regra 2 mais uma vez, obtemos a fbf

$$(((A \wedge B) \rightarrow (C))')$$

A negação de uma implicação é (p→q)' = p ^ q'

Eliminando alguns parênteses, podemos escrever essa fbf como

(A ^ B) ^ C'

CONJUNTOS DEFINIDOS POR RECORRÊNCIA

PROBLEMA PRÁTICO 4 | Mostre como construir a fbf $((A \vee (B')) \to C)$ da definição

- A, B e C são fbfs pela regra 1
- $(\boldsymbol{B}')$ é fbf pela regra 2
- $(\boldsymbol{A} \vee (\boldsymbol{B}'))$ é uma fbf pela regra 2
- $\left(\boldsymbol{A} \vee (\boldsymbol{B}')\right) \to \boldsymbol{C})$ é uma fbf pela regra 2

CADEIAS DE SÍMBOLOS

Cadeias de símbolos retiradas de um "alfabeto" finito são objetos encontrados com frequência em ciência da computação. Computadores guardam os dados como **cadeias binárias**, cadeias do alfabeto que consiste apenas em 0 e 1; compiladores veem proposições ou comandos em programas como cadeias de *marcas* ou *sinais*,† tais como palavras-chave e identificadores. A coleção de todas as cadeias de comprimento finito formada por símbolos de um alfabeto, chamadas de cadeias *de* um alfabeto, podem ser definidas de forma recorrente

CADEIAS DE SÍMBOLOS

O que são cadeias de símbolos?
Em computação, usamos muito as **cadeias de símbolos**. Por exemplo:

1. **Alfabeto Finito**:
   - Alfabeto usado na língua portuguesa possui 26 letras (A, B, C, ... Z).
   - Alfabeto binário: muito simples com somente dois símbolos: 0 e 1.

Presente em:

- **Computadores**: com armazenamento de dados como cadeias binárias (sequências de 0s e 1s).
- **Compiladores**: analisam o código (como programas de computador) e vêem esse código como cadeias de símbolos, que podem ser palavras-chave (como "if", "while") e identificadores (nomes de variáveis, como "x" ou "nome").

CADEIAS DE SÍMBOLOS

Cadeias de um alfabeto:

Cadeia representa uma sequência de símbolos de um alfabeto.

Exemplo: com o alfabeto {0, 1} podemos obter as cadeiras "0", "1", "01", "10", "1100", etc.

- **Coleção de todas as cadeias**: conjunto de todas as possíveis sequências finitas que podemos formar com os símbolos desse alfabeto.

Ex: O conjunto de todas as cadeias (de comprimento finito) de símbolos de um alfabeto A é denotado por A^*. A definição recorrente de A^* é

1. A **cadeia vazia** λ (a cadeia sem nenhum símbolo) pertence a A^*.
2. Um único elemento qualquer de A pertence a A^*.
3. Se x e y são cadeias em A^*, então a **concatenação** xy de x e y também pertence a A^*.

As partes 1 e 2 constituem a base, e a parte 3 é o passo indutivo dessa definição. Note que, para qualquer cadeia x, $x\lambda = \lambda x = x$.

Cadeia Vazia (λ)

CADEIAS DE SÍMBOLOS

PROBLEMA PRÁTICO 6

Se $x = 1011$ e $y = 001$, escreva as cadeias xy, yx e $yx\lambda x$. ■

CADEIAS DE SÍMBOLOS

PROBLEMA PRÁTICO 7	Dê uma definição recorrente para o conjunto de todas as cadeias binárias que são **palíndromos**, cadeias que são iguais se lidas normalmente ou de trás para a frente. ■

Casos base e a regra de formação para construir uma cadeia maior a partir de cadeias menores:

Caso base:

1. A cadeia vazia (λ) é um palíndromo.
2. Qualquer cadeia de um único símbolo ("0" ou "1") é um palíndromo.

Regra:

- Se y é um palíndromo, então 0y0 também é um palíndromo.
- Se y é um palíndromo, então 1y1 também é um palíndromo.

Ex:

Caso base: para x = λ, x = 0 e x = 1 ✅

Aplicando a regra recursiva usando λ, temos: 0λ0 = 00 e 1λ1 = 11, o que é verdade

Fundamentos de Teoria da Computação
Aula 12

PRISCILA MARQUES KAI

CADEIAS DE SÍMBOLOS

EXEMPLO 7

Suponha que, em determinada linguagem de programação, os identificadores podem ser cadeias alfanuméricas de comprimento arbitrário, mas têm que começar com uma letra. Uma definição recorrente para o conjunto dessas cadeias é

1. Uma única letra é um identificador.
2. Se *A* for um identificador, a concatenação de *A* e qualquer letra ou dígito também o será.

Uma notação mais simbólica para descrever conjuntos de cadeias definidas por recorrência é chamada de **forma de Backus Naur**, ou **FBN**, desenvolvida originalmente para definir a linguagem de programação ALGOL. Em notação FBN, os itens que são definidos em termos de outros itens são envolvidos pelos símbolos de menor e maior, enquanto itens específicos que não podem ser divididos não aparecem dessa forma. Um segmento vertical | denota uma escolha e tem o mesmo significado que a palavra *ou*. A definição em FBN de um identificador é

<identificador> ::= <letra> | <identificador> <letra> | <identificador> <dígito>
<letra> ::= *a* | *b* | *c* | ··· | *z*
<dígito> ::= 1 | 2 | ··· | 9

CADEIAS DE SÍMBOLOS

Assim, o identificador me2 pode ser obtido da definição por uma sequência de escolhas como

<identificador>	pode ser	<identificador> <dígito>
	que pode ser	<identificador>2
	que pode ser	<identificador> <letra>2
	que pode ser	<identificador>e2
	que pode ser	<letra>e2
	que pode ser	me2

CADEIAS DE SÍMBOLOS

Como outro exemplo de ligação entre recorrência e indução, existe uma forma de indução chamada de **indução estrutural** que pode ser aplicada a conjuntos definidos por recorrência. Suponha que S é um conjunto definido por recorrência e suponha que existe uma propriedade $P(x)$ que pode ser válida ou não para um elemento x de S. Se pudermos provar que:

1. A propriedade P é válida para todos os elementos descritos na base.
2. Se a propriedade P for válida para alguns elementos de S, então será válida para todos os elementos novos de S construídos desses elementos usando o passo indutivo.

então a propriedade P será válida para todos os elementos de S.

CADEIAS DE SÍMBOLOS

EXEMPLO 8

Um conjunto S de cadeias é definido recursivamente por

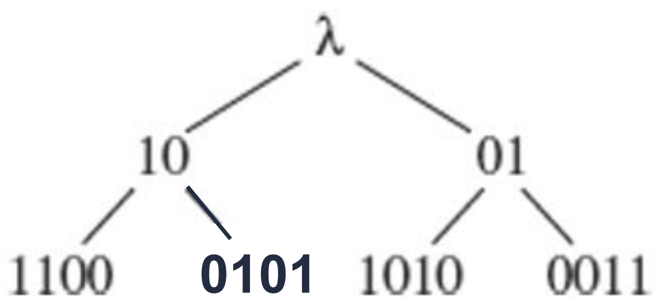
1. λ pertence a S.
2. Se x pertencer a S, então $1x0$ e $0x1$ também pertencerão a S.

Podemos usar a indução estrutural para provar que toda cadeia em S tem o mesmo número de zeros e de uns. A base, Regra 1, identifica uma única cadeia em S, a cadeia vazia **l**, que tem o mesmo número de zeros e de uns (0 zeros e 0 uns). Suponha que a cadeia x tem o mesmo número de zeros e de uns. Usando a Regra 2, as duas cadeias novas que podem ser construídas a partir de x adicionam um único 0 e um único 1 a x, de modo que o número de zeros e o número de uns foram aumentados de 1, que assim os números de zeros e de uns continuam iguais. Pela indução estrutural, toda cadeia em S tem o mesmo número de zeros e de uns.

Note que nem todas as cadeias contendo o mesmo número de zeros e de uns pertencem a S. Por exemplo, não é possível gerar a cadeia 1001 usando as regras dadas. ●

CADEIAS DE SÍMBOLOS

A indução matemática usual prova propriedades sobre valores inteiros positivos e os inteiros positivos são ordenados: 1, 2, …, k, $k + 1$, …. Um conjunto, no entanto, não precisa ser ordenado. Se considerarmos o conjunto S definido no Exemplo 8, ele parece com



e a indução estrutural nos ajuda a tratar essa “disseminação” de valores no conjunto.

OPERAÇÕES DEFINIDAS POR RECORRÊNCIA

Certas operações em objetos podem ser definidas de forma recorrente, como nos exemplos a seguir:

Uma definição recorrente da operação de potenciação a^n de um número real não nulo a, em que n é um inteiro não negativo, é

Para definir a potenciação de um número real a de forma recorrente:

1. $a^0 = 1$
2. $a^n = (a^{n-1}) \cdot a$ para $n \geq 1$

Ex:

1. **Caso Base**: $a^0 = 1$. Independentemente do valor de a, se elevarmos a a 0, o resultado sempre será 1. Por exemplo: $2^0 = 1$, $12^0 = 1$, $100^0 = 1$.
2. **Regra Recursiva**: para $n = 1$, temos que $a^1 = (a^{1-1})a = a^0 a = 1a = a$.

   para n = 2, temos que $a^2 = (a^{2-1})a = (a^1)a = ((a^{1-1})a)a = ((a^0 a)a = aa = a^2$.

   para n = 3, temos que $a^3 = (a^{3-1})a = (a^2)a = ((a^{2-1})a)a = ((a^1)a)a = aaa = a^3$.

OPERAÇÕES DEFINIDAS POR RECORRÊNCIA

Certas operações em objetos podem ser definidas de forma recorrente, como nos exemplos a seguir:

Uma definição recorrente para a multiplicação de dois inteiros positivos m e n.

1. $m(1) = m$
2. $m(n) = m(n - 1) + m$ para $n \geq 2$

Ex:

1. **Caso Base**: Ao multiplicamos m por 1, o resultado é $m(1) = m \cdot 1 = m$
2. **Regra Recursiva**: Para $n = 2$: $m(2) = m(1) + m = m + m$

   Para n = 5: $m(5) = m(4) + m = (m(3) + m) + m = ((m(2) + m) + m) + m = (((m(1) + m) + m) + m) + m$
   $m(5) = (((m + m) + m) + m) + m = m+m+m+m+m$

ALGORITMOS DEFINIDOS POR RECORRÊNCIA

O Exemplo 1 dá uma definição recorrente para uma sequência S. Suponha que queremos escrever um programa de computador para calcular $S(n)$ para algum inteiro positivo n. Temos duas abordagens possíveis. Se quisermos encontrar $S(12)$, por exemplo, podemos começar com $S(1) = 2$ e depois calcular $S(2)$, $S(3)$, e assim por diante, como fizemos no Exemplo 1, até chegar, finalmente, em $S(12)$. Sem dúvida, essa abordagem envolve iteração em alguma espécie de laço. A seguir, vamos dar uma função S em pseudocódigo que usa esse algoritmo iterativo. A base, com $n = 1$, é obtida na primeira cláusula do comando **se**; o valor 2 é retornado. A cláusula **senão**, para $n > 1$, tem uma atribuição inicial e entra em um laço **enquanto** que calcula valores maiores da sequência até atingir o limite superior correto. Você pode seguir a execução desse algoritmo para alguns valores de n para se convencer de que ele funciona.

Considere a sequência S definida por recorrência por

1. $S(1) = 2$
2. $S(n) = 2S(n - 1)$ para $n \geq 2$

ALGORITMO

```
S(inteiro positivo n)
//função que calcula iterativamente o valor S(n)
//para a sequência S do Exemplo 1
Variáveis locais:
inteiro i      //índice do laço
ValorAtual     //valor atual da função S

    se n = 1 então
      retorne 2
    senão
      i = 2
      ValorAtual = 2
      enquanto i <= n faça
      ValorAtual = 2 * ValorAtual
      i = i + 1
    fim do enquanto

  //agora ValorAtual tem o valor S(n)
  retorne ValorAtual
fim do se
fim da função S
```

ALGORITMO

```
S(inteiro positivo n)
//função que calcula o valor S(n) de forma recorrente
//para a sequência S do Exemplo 1

 se n = 1 então
   retorne 2
 senão
   retorne 2 * S(n – 1)
 fim do se
fim da função S
```

ALGORITMOS DEFINIDOS POR RECORRÊNCIA

PROBLEMA PRÁTICO 9	Escreva o corpo de uma função recursiva para calcular $T(n)$ para a sequência T definida no Problema Prático 1. ■

PROBLEMA PRÁTICO 1	A sequência T é definida por recorrência por:

1. $T(1) = 1$
2. $T(n) = T(n-1) + 3$ para $n \geq 2$

Escreva os cinco primeiros valores da sequência T. ■

ALGORITMOS DEFINIDOS POR RECORRÊNCIA

ALGORITMO

Produto(inteiro positivo m; inteiro positivo n)
//Função que calcula de forma recursiva o produto de m e n

se $n = 1$ **então**
retorne m;
senão
retorne *Produto*$(m, n - 1) + m$
fim do se
fim da função *Produto*

ALGORITMO *BUSCABINÁRIA*

```
BuscaBinária(lista L; inteiro positivo i; inteiro positivo j; tipo item x)
//procura na lista ordenada L, de L[i] a L[j], pelo item x

  se i > j então
    escreva("não encontrado")
  senão
    encontre o índice k do item do meio entre i e j
    se x = item do meio L[k] então
      escreva("encontrado")
    senão
      se x < item do meio L[k] então
        BuscaBinária(L, i, k – 1, x)
    senão
      BuscaBinária(L, k + 1, j, x)
    fim do se
   fim do se
  fim do se
fim da função BuscaBinária
```

ALGORITMOS DEFINIDOS POR RECORRÊNCIA

EXEMPLO 14

Vamos aplicar o algoritmo de busca binária à lista

3, 7, 8, 10, 14, 18, 22, 34

onde o item x a ser encontrado é o número 25. A lista inicial não é vazia, logo o item do meio é localizado e encontra-se o valor 10. Então x é comparado com o item do meio. Como $x > 10$, a busca é feita na segunda metade da lista, a saber, entre os itens

14, 18, 22, 34

Novamente, essa lista não é vazia, e o item do meio é 18. Como $x > 18$, procura-se na segunda metade da lista, ou seja, entre os itens

22, 34

Nessa lista não vazia, o item do meio é 22. Como $x > 22$, a busca continua na segunda metade da lista, a saber,

34

Essa é uma lista de apenas um elemento, com o item do meio sendo esse único elemento. Como $x < 34$, começa uma busca na "primeira metade" da lista; mas a primeira metade é vazia. O algoritmo termina aqui com a informação de que x não está na lista.

Essa execução necessita de quatro comparações ao todo; x é comparado com 10, 18, 22 e 34.

RELAÇÕES DE RECORRÊNCIA

Desenvolvemos dois algoritmos, um iterativo e o outro recorrente, para calcular um valor $S(n)$ para a sequência S do Exemplo 1. No entanto, existe um modo ainda mais fácil de calcular $S(n)$. Lembre que

$$S(1) = 2 \qquad (1)$$
$$S(n) = 2S(n-1) \text{ para } n \geq 2 \qquad (2)$$

Como

$$S(1) = 2 = 2^1$$
$$S(2) = 4 = 2^2$$
$$S(3) = 8 = 2^3$$
$$S(4) = 16 = 2^4$$

e assim por diante, vemos que

$$S(n) = 2^n \qquad (3)$$

RELAÇÕES DE RECORRÊNCIA

$$S(n) = 2^n \qquad (3)$$

Usando a Equação (3), podemos calcular diretamente S(n) sem ter que calcular antes todos os valores menores de *S*. Uma equação como (3), onde podemos substituir um valor e obter diretamente o que queremos, é chamada uma **solução em forma fechada** para a relação de recorrência (2) sujeita à condição básica (1). Quando encontramos uma solução em forma fechada, dizemos que **resolvemos** a relação de recorrência.

Uma técnica para resolver relações de recorrência é uma abordagem do tipo "**expandir**, **conjecturar** e **verificar**", que usa repetidamente a relação de recorrência para expandir a expressão a partir do *n*-ésimo termo até podermos ter uma ideia da forma geral. Finalmente essa conjectura é verificada por indução matemática.

RELAÇÕES DE RECORRÊNCIA

Considere novamente a condição básica e a relação de recorrência para a sequência S:

$S(1) = 2$

$S(n) = 2S(n - 1)$ para $n \geq 2$

Suponha que não sabemos a solução fechada. Vamos usar a abordagem de expandir, conjecturar e verificar para encontrá-la.

$$\begin{aligned}
S(n) &= 2S(n - 1) \\
&= 2(2S(n - 2) = 2^2S(n - 2) \\
&= 2^2(2S(n - 3)) = 2^3S(n - 3) \\
&= 2^3(2S(n - 4)) = 2^4S(n - 4) \\
&= 2^kS(n - k)
\end{aligned}$$

a expansão para quando $n - k = 1$ (primeiro elemento da sequência → S(1)), ou seja, $k = n - 1$. Substituindo:

$$\begin{aligned}
S(n) &= 2^kS(n - k) \\
&= 2^{n-1}S(n - (n - 1)) \\
&= 2^{n-1}S(1) \\
&= 2^{n-1} \cdot 2 \\
&= 2^n \qquad \text{que expressa a solução em forma fechada}
\end{aligned}$$

RELAÇÕES DE RECORRÊNCIA

Ainda precisamos confirmar nossa solução em forma fechada por indução no valor de n. A proposição que queremos provar, portanto, é que $S(n) = 2^n$ para $n \geq 1$.

Caso base:

$$S(1) = 2^1 = 2$$

Agora suponha para n = k, $S(k) = 2^k$. Então

$$S(k+1) = 2^{k+1}$$
$$2S(k) = 2^{k+1}$$
$$2.2^k = 2^{k+1}$$
$$2^{k+1} = 2^{k+1}$$

Isso prova que nossa solução em forma fechada está correta.

RELAÇÕES DE RECORRÊNCIAS LINEARES DE PRIMEIRA ORDEM

PROBLEMA PRÁTICO 11 Encontre uma solução em forma fechada para a relação de recorrência, sujeita à condição básica, para a sequência T:

1. $T(1) = 1$
2. $T(n) = T(n-1) + 3$ para $n \geq 2$

(*Sugestão:* Expanda, conjecture e verifique.) ■

$$
\begin{aligned}
T(n) &= T(n - 1) + 3 \\
&= (T(n - 2) + 3) + 3 \\
&= ((T(n - 3) + 3) + 3) + 3 \\
&= T(n - k) + 3k
\end{aligned}
$$

A expansão continua até que n - k = 1 (caso base), logo, $k = n - 1$. Substituindo, teremos:

$$
\begin{aligned}
T(n) &= T(n - (n - 1)) + 3(n - 1) \\
&= T(1) + 3n - 3 \\
&= 1 + 3n - 3 \\
&= 3n - 2
\end{aligned}
$$

Agora queremos provar que $T(n) = 3n - 2$, para $n \geq 1$

Caso base:

$T(1) = 3 \cdot 1 - 2 = 1$

Agora suponha para n = k, $T(k) = 3k - 2$. Então

$$
\begin{aligned}
T(k + 1) &= 3(k + 1) - 2 \\
T(k) + 3 &= 3(k + 1) - 2, \text{ substituindo pela H.I.} \\
(3k - 2) + 3 &= 3(k + 1) - 2
\end{aligned}
$$

manipulando os termos

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

Portanto, a solução em forma fechada está correta