**RECURSIVIDADE E RELAÇÕES DE RECORRÊNCIA**

## Definições Recorrentes

Uma definição onde o item sendo definido aparece como parte da definição é chamada de uma definição recorrente ou definição por recorrência ou ainda definição por indução. A princípio isso não parece fazer sentido - como podemos definir algo em termos de si mesmo? Isso funciona porque uma definição recorrente tem duas partes:

1. Uma base, ou condição básica, onde alguns casos simples do item sendo definido são dados explicitamente.
2. Um passo de indução ou recorrência, onde novos casos do item sendo definido são dados em função de casos anteriores.

A parte 1 nos dá um lugar para começar, fornecendo alguns casos simples e concretos; a parte 2 nos permite construir novos casos, a partir desses simples, e depois construir ainda outros casos a partir desses novos, e assim por diante. (O nome "definição por indução" é devido à analogia com demonstrações por indução matemática. Em uma demonstração por indução existe uma base da indução, a saber, mostrar que P(1) - ou P em algum outro valor inicial- é verdadeira, e existe um passo indutivo, onde a veracidade de P(k + 1) é estabelecida a partir da veracidade de P em valores anteriores.)

Recorrência é uma idéia importante que pode ser usada para definir seqüências de objetos, coleções mais gerais de objetos e operações com objetos. Até algoritmos podem ser recorrentes.

## Seqüências Definidas por Recorrência

Uma seqüência S é uma lista de objetos que são numerados em determinada ordem; existe um primeiro objeto, um segundo, e assim por diante. S(k) denota o k-ésimo objeto na seqüência. Uma seqüência é definida por recorrência nomeando-se, explicitamente, o primeiro valor (ou alguns poucos primeiros valores) na seqüência e depois definindo valores subseqüentes na seqüência em termos de valores anteriores.

**Exemplo 1:** A seqüência S é 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 seqüência S é

2,4,8,16,32, ...

Uma regra como a da proposição 2 no Exemplo 1, que define um valor de uma seqüência em termos de um ou mais valores anteriores, é chamada uma relação de recorrência.

**Exercício 1:** A seqüência 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 seqüência T.

**Exemplo 2:** A famosa seqüência de Fibonacci, introduzida no século XIII por um comerciante e matemático italiano, é uma seqüência de números definida por recorrência por

$F(1) = 1$
$F(2) = 1$
$F(n) = F(n - 2) + F(n - 1)$ para $n > 2$

Aqui são dados os dois primeiros valores da seqüência e a relação de recorrência define o n-ésimo valor 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.

**Exercício 2:** Escreva os oito primeiros valores da seqüência de Fibonacci.

**Exemplo 3:** A fórmula

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

pode ser provada diretamente, sem indução, usando apenas a relação de recorrência na definição dos números de Fibonacci. A relação de recorrência

$$F(n + 2) = F(n) + F(n + 1)$$

pode ser reescrita na forma

$$F(n + 1) = F(n + 2) - F(n)$$

Logo,

F(n + 4) = F(n + 3) + F(n + 2)
= F(n + 2) + F(n + 1) + F(n + 2) reescrevendo F(n + 3)
= F(n + 2) + [F(n + 2) - F(n)] + F(n + 2) reescrevendo F(n + 1)
= 3F(n + 2) - F(n) usando (1)

## Conjuntos Definidos por Recorrência

Os objetos em uma seqüência são ordenados - existe 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.

Cadeias de símbolos retiradas de um "alfabeto" finito são objetos encontrados com freqüência em ciência da computação. Computadores guardam os dados como cadeias binárias, cadeias do alfabeto que consiste apenas em 0 e 1; compiladores vêem proposições ou comandos em programas como cadeias de *tokens*, 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, pode ser definida de forma recorrente (veja o Exemplo 4). Muitos conjuntos de cadeias com propriedades particulares também têm definições recorrentes.

**Exemplo 4:** 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 $\lambda$ (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, $x\lambda = \lambda x = x$.

**Exercício 3:** Se x = 1011 e y = 001, escreva as cadeias xy, yx e yx $\lambda$ x.

**Exercício 4:** 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.

**Exemplo 5:** 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 é um identificador, a concatenação de A e qualquer letra ou dígito também o é.

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
Assim, o identificador *me2* pode ser obtido da definição por uma seqüência 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*

## Operações Definidas por Recorrência

Certas operações em objetos podem ser definidas de forma recorrente, como nos Exemplos 6 e 7.

**Exemplo 6:** Uma definição recorrente da operação de exponenciação $a^n$ de um número real não-nulo a, onde n é um inteiro não-negativo, é

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

**Exemplo 7:** Uma definição recorrente para a multiplicação de dois inteiros positivos *m* e *n* é

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

**Exercício 5:** Seja x uma cadeia de um determinado alfabeto. Dê uma definição recorrente para a operação $x^n$ (concatenação de x consigo mesmo n vezes) para $n \geq 1$.

## Algoritmos Definidos por Recorrência

O Exemplo 1 dá uma definição recorrente para uma seqüência S. Suponha que queremos escrever um programa de computador para calcular S(n) para algum inteiro positivo n. Podemos usar uma entre duas abordagens. Se queremos 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 em pseudocódigo S que usa esse algoritmo iterativo. A base, com n = 1, é obtida na primeira cláusula da proposição se; o valor 2 é retomado. A cláusula senão, para n > 1, tem uma atribuição inicial e entra em um laço enquanto que calcula valores maiores da seqüência 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.

```
ALGORITMO
S(inteiro n)
//função que calcula interativamente o valor S(n)
//para a seqüência S do Exemplo 1

Variáveis locais:
        inteiro i, ValorCorrente
se n = 1 então
        retorne 2
senão
        i = 2
        ValorCorrente = 2
        Enquanto i < = n faça
                ValorCorrente = 2*ValorCorrente
                i = i + 1
        fim do enquanto
         retorne ValorCorrente
fim do se
fim da função S
```

A segunda abordagem para calcular S(n) usa diretamente a definição recorrente de S. A versão a seguir é de um algoritmo recorrente ou recursivo, escrito novamente como uma função em pseudocódigo.

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

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

O corpo dessa função consiste em uma única proposição do tipo se-então-senão. Para compreender como essa função funciona, vamos seguir a execução para calcular o valor de S(3). Chamamos primeiro a função com um valor de entrada n = 3. Como n não é 1, a execução é direcionada para a cláusula senão. Nesse instante, a atividade de calcular S(3) tem que ser suspensa até se conhecer o valor de S(2). Qualquer informação conhecida, relevante para o cálculo de S(3), é armazenada na memória do computador em uma pilha, que será recuperada quando o cálculo for completado. (Uma pilha é uma coleção de dados onde qualquer novo item vai para o topo da pilha e, em qualquer instante, apenas o item no topo é acessível ou pode ser removido da pilha. Portanto, uma pilha é uma estrutura LIFO - do inglês *last in, first out*, isto é, o último a entrar é o primeiro a sair.) A função é chamada novamente com um valor de entrada n = 2. Mais uma vez, a cláusula senão é executada e o cálculo de S(2) é suspenso, com as informações relevantes armazenadas na pilha, enquanto a função é chamada novamente com n = 1.

Dessa vez é executada a primeira cláusula da proposição se e o valor da função, 2, pode ser calculado diretamente. Essa chamada final da função está completa e o valor 2 é usado na penúltima chamada da função, que remove agora da pilha qualquer informação relevante ao caso n = 2, calcula S(2) e usa esse valor na invocação prévia (inicial) da função. Finalmente, essa chamada original de S é capaz de esvaziar a pilha e completar seu cálculo, retomando o valor de S(3).

Quais são as vantagens relativas dos algoritmos iterativo e recorrente ao executar a mesma tarefa? Nesse exemplo, a versão recorrente é certamente mais curta, já que não precisa gerenciar um cálculo em laço. A descrição da execução do algoritmo recorrente parece soar mais complicada do que a do algoritmo iterativo, mas todos os passos são executados automaticamente. Não precisamos estar conscientes do que está acontecendo internamente, exceto para observar que uma série longa de chamadas recorrentes pode usar muita memória ao armazenar na pilha as informações relevantes para as invocações prévias. Se a utilização da memória for excessiva, pode acontecer um "transbordamento" (*overflow*) da pilha. Além de usar mais memória, algoritmos recorrentes podem necessitar de mais cálculos e executar mais lentamente do que os não-recorrentes.

Apesar disso, a recorrência (ou recursividade) fornece um modo natural de pensar em muitas situações, algumas das quais necessitariam soluções não-recorrentes muito complexas. Uma solução recorrente é bem adequada para o problema de calcular os valores de uma seqüência definida de maneira recorrente. Muitas linguagens de programação aceitam recorrência.

**Exercício 6:** Escreva o corpo de uma função recorrente para calcular T(n) para a seqüência T definida no Problema Prático 1.

**Exemplo 8:** No Exemplo 7, foi dada uma definição recorrente para a multiplicação de dois inteiros positivos m e n. A seguir temos uma função em pseudocódigo para a multiplicação baseada nessa definição.

```
ALGORITMO
Produto(inteiro m; inteiro 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
```

Um algoritmo recorrente chama a si mesmo com valores de entrada "menores". Suponha que um problema pode ser resolvido encontrando-se soluções para as versões menores do mesmo problema e que as versões menores acabam se tomando casos triviais, facilmente solucionados. Então um algoritmo recorrente pode ser útil, mesmo que o problema original não tenha sido enunciado de forma recorrente.

Para nos convencer de que um determinado algoritmo recorrente funciona, não precisamos começar com um dado particular de entrada, ir diminuindo, tratando de casos cada vez menores, e depois ir voltando, percorrendo todo o caminho inverso. Fizemos isso ao discutir o cálculo de S(3), mas foi só para ilustrar a mecânica de um cálculo recorrente. Ao invés disso, podemos verificar o caso trivial (como demonstrar a base em uma demonstração por indução) e verificar que, se o algoritmo funciona corretamente ao ser chamado com valores de entrada menores, então ele resolve, de fato, o problema para o valor de entrada original (o que é semelhante a provar P(k + 1) da hipótese P(k) em uma demonstração por indução).

**Exemplo 9:** Uma das tarefas mais comuns em processamento de dados é colocar uma lista L de n itens em ordem numérica ou alfabética, crescente ou decrescente. (Essa lista pode conter nomes de clientes, por exemplo, e, em ordem alfabética, "Vargas, Joana" deveria vir depois de "Teixeira, José".) O algoritmo de ordenação por seleção - um algoritmo simples mas particularmente eficaz - é descrito em pseudocódigo abaixo.

Essa função ordena os j primeiros itens em L em ordem crescente; quando a função é chamada pela primeira vez, j tem o valor n (de modo que a primeira chamada ordena toda a lista). A parte recorrente do algoritmo está dentro da cláusula senão; o algoritmo examina a seção da lista sob consideração e encontra o valor de i para o qual L[i] tem o valor máximo. Ele, então, permuta L[i] e L[j], depois do que o máximo ocorre na posição j, a última posição na parte da lista sendo

considerada. L[j] está correto agora e não deve mais ser modificado, de modo que esse processo é repetido na lista de L[1] a L[j - 1]. Se essa parte da lista for ordenada corretamente, então a lista inteira será ordenada corretamente. Quando j tem o valor 1, à parte da lista sendo considerada tem apenas um elemento, que tem que estar no lugar certo. Nesse instante a lista toda está ordenada.

```
ALGORITMO OrdenaçãoPorSeleção
OrdenaçãoPorSeleção(lista L; inteiro j)
//algoritmo recorrente para ordenar os itens de 1 a j em uma
//lista L em ordem crescente
        se j = 1 então
           a ordenação está completa, escreva a lista ordenada
        senão
           encontre o índice i do maior item em L entre 1 e j
           permute L[i] e L[j]
           OrdenaçãoPorSeleção(L, j – 1)
        fim do se
fim da função OrdenaçãoPorSeleção
```

**Exemplo 10:** Agora que ordenamos nossa lista, uma outra tarefa comum é procurar um item particular na lista. (Joana Vargas já é uma cliente?) Uma técnica eficiente de busca em uma lista ordenada é o algoritmo de busca binária, um algoritmo recorrente descrito em pseudocódigo a seguir:

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

        se i > j então
                escreva(“item não encontrado”)
        senão
        encontre o índice k do item do meio na lista L[i] – L[j]
        se x = item do meio então
        escreva("item encontrado")
        senão
        se x < item do meio então
        BuscaBinária(L, i, k - 1, x)
        senão

        BuscaBinária(L, k + l,j, x)
        fim do se
        fim do se
        fim do se
fim da função BuscaBinária
```

Esse algoritmo procura na seção da lista entre L[i] e L[j] por um item x; inicialmente, i e j têm os valores 1 e n, respectivamente. A primeira cláusula do se principal é o passo básico que diz que x não pode ser encontrado em uma lista vazia, uma na qual o primeiro índice é maior do que o último. Na cláusula senão principal, o item do meio em uma seção da lista tem que ser encontrado. (Se a seção tem um número ímpar de itens, existe, de fato, um item do meio; se a seção contém um número par de itens, basta escolher como "item do meio" o que fica no final da primeira metade da seção da lista.) Comparando x com o item do meio, localizamos a metade da lista onde devemos procurar a seguir.

**Exemplo 11:** Vamos aplicar o algoritmo de busca binária à lista

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

na qual 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, isto é, 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.

**Exercício 7:** Em uma busca binária da lista no Exemplo 11, nomeie os elementos que são comparados com x se x tem o valor 8.

Vimos uma série de definições recorrentes. A Tabela abaixo resume suas características.

| Definições Recorrentes | |
|---|---|
| O que está sendo definido | Características |
| Seqüência recorrente | O primeiro ou os dois primeiros valores da seqüência são conhecidos; os outros elementos na seqüência são definidos em termos dos anteriores. |
| Conjunto recorrente | Alguns elementos específicos do conjunto são conhecidos; outros elementos no conjunto são construídos a partir dos elementos que já sabemos que pertencem ao conjunto. |
| Operação recorrente | Um caso "pequeno" da operação fornece um valor específico; outros casos são definidos a partir de casos menores. |
| Algoritmo recorrente | Para o menor valor do(s) argumento(s), o comportamento do algoritmo é conhecido; para valores maiores, o algoritmo chama a si mesmo com valores menores do(s) argumento(s). |