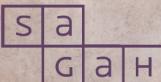


ESTRUTURA DE DADOS

Rafael Albuquerque



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS

Listas encadeadas duplas

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Identificar o uso de uma lista encadeada dupla.
- Reconhecer a navegação em uma lista encadeada dupla para inserir, remover, atualizar e consultar valores.
- Aplicar a implementação de listas encadeadas duplas.

Introdução

As listas duplamente encadeadas — também conhecidas como listas duplamente ligadas — são uma extensão das listas encadeadas simples. As listas duplamente encadeadas são estruturas de dados que vinculam objetos (nós) por referências tanto para um endereço posterior quanto para um anterior, diferentemente das listas encadeadas simples, que vinculam os nós apenas para elementos posteriores, ou seja, onde não existe a possibilidade de os elementos percorrerem na ordem inversa.

Em problemas computacionais, a aplicação de listas duplamente encadeadas é utilizada em vetores, em que uma coluna indica qual é o dado anterior ou posterior e a posição é definida como índice. Sempre que se pensa em listas com encadeamento duplo, tem-se a ideia de navegação em mão dupla, como a navegação em diretórios, navegadores de internet, reprodutores de música, redes sociais, etc.

Neste capítulo, você aprenderá a identificar o uso de listas encadeadas duplas e entenderá como funciona cada uma das ações de uma lista encadeada dupla, tais como inserir, remover, atualizar e consultar valores. Além disso, você estudará sobre como realizar a implementação de uma lista encadeada dupla.

1 Identificação do uso de listas encadeadas duplas

As listas são bem utilizadas pelo que tange à sensação de organização que elas nos trazem, seja de modo impresso em folhas de papel, como as famosas listas telefônicas, agendas, etc. O interessante em se utilizar listas é que elas podem conter qualquer critério de organização, mesmo que sua estrutura seja de modo sequencial, em que cada elemento, ou nó, tem referência ao próximo, até que não haja mais a qual item referenciar.

Logo, uma lista duplamente encadeada tem como estrutura duas referências, uma em cada ponta do elemento, o que permite que ela seja percorrida em dois sentidos, do início ao fim e do fim ao início (EDELWEISS; GALANTE, 2009). A ilustração do seu comportamento pode ser acompanhada pela Figura 1. Perceba que existem três itens em cada elemento da lista, em que dois deles correspondem aos ponteiros que se ligam aos elementos anteriores até a referência NULA e posteriores, igualmente até a referência NULA. O item Info corresponde à variável que contém os dados, que podem ser tanto do tipo primitivo como do tipo abstrato (também conhecido na literatura como tipo abstrato de dado [TAD]).

A forma como essa estrutura de dados é projetada, permite-nos aplicá-la em vários cenários em que sejam necessárias filas de esperas de processos computacionais ou apenas listar objetos para exibição, como apresentado em sites de compras, quando são listados os produtos de desejos ou que já estão no carrinho de compras, os próprios comandos de desfazer (CTRL + Z) e refazer (CTRL + SHIT + Z), sistemas de paginação web, os botões de voltar e avançar do navegador, etc. Perceba que existe uma ampla utilização das listas duplamente encadeadas em nosso cotidiano e que elas facilitam o uso de vários sistemas.

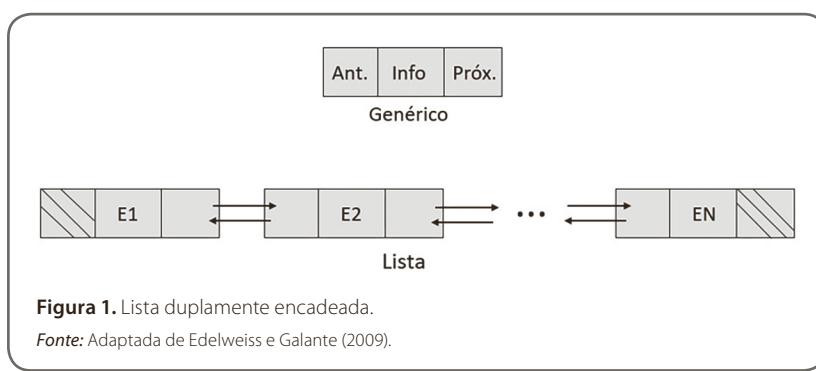


Figura 1. Lista duplamente encadeada.

Fonte: Adaptada de Edelweiss e Galante (2009).

Características das listas de encadeamento duplo

As listas de encadeamento duplo têm algumas características que precisam ser bem definidas para o seu funcionamento, conforme Celes, Cerqueira e Rangel (2004):

- Definir um ponteiro para o primeiro e outro para o último elemento da lista, para que esta seja percorrida tanto do primeiro elemento até o final como de forma contrária.
- Determinar os elementos da lista para que tenham dois atributos do tipo ponteiro, para que eles apontem para os elementos anteriores e posteriores.
- O ponteiro anterior, referente ao primeiro elemento da lista, aponta para um endereço NULO. Este indicativo é uma forma de sabermos que a lista inicia a partir desse elemento.
- De forma análoga, o ponteiro posterior ao último elemento aponta para um endereço NULO, indicando que esse elemento é de fato o último.
- Os elementos da lista podem ser variáveis de um único tipo de dado primitivo, ou ser compostos por estruturas de dados abstratos, também nomeados como registros.

Para fins de exemplificação de uma lista dinâmica com encadeamento duplo, criaremos um TAD para inserção de numeração, com seus respectivos ponteiros direcionais. As implementações serão apresentadas na linguagem de programação em Python 3.x, logo, o conceito de TAD aqui será substituído pelo uso de Classe, como forma de manter a ideia de representação de um elemento de lista. A Figura 2 ilustra a criação do elemento para uma lista.



Link

A ferramenta utilizada para as implementações realizadas aqui pode ser acessada por meio do *link* a seguir.

<https://qrgo.page.link/KHkUi>

```
class Elemento(object):
    def __init__(self, info):
        self.info = info
        self.ant = None
        self.prox = None
```

Figura 2. Registro em Python 3.x.

Como os exemplos estão descritos em Python, as variáveis têm o estilo de tipagem fraca, o que nos permite maior liberdade quanto ao tipo de dado passado ao objeto da classe Elemento em seu instanciamento. Logo, o atributo de classe `self.info` pode representar qualquer tipo de dado, podendo ser um tipo simples, como um valor em inteiro (p. ex., `0`, `1`, `10`, etc.), ponto flutuante (p. ex., `0.1`, `0.2`, `0.5`, etc.), *string* (p. ex., `“a”`, `“ab”`, `“abc”`, etc.) e *booleano* (p. ex., `True` e `False`), assim como um objeto com mais atributos.

Uma vez alocado o elemento na memória, a remoção dele fica por conta do módulo chamado de **Garbage Collector**. Este mecanismo é muito utilizado por linguagens modernas, como C#, Java, Python, etc. Em linguagens como C, a alocação dinâmica das variáveis na memória fica por conta de métodos como `calloc` e `malloc`, e a remoção, por conta do método `free`.



Fique atento

Por mais que as listas sejam comumente usadas em sistemas computacionais, sua utilização reflete em nossas vidas de maneira recorrente e natural, como a implementação de consultas em *sites* ou consultas em tabelas de usuários cadastrados em um banco de dados.

2 Ações de uma lista encadeada dupla

Em uma lista encadeada dupla, seja ela implementada sobre um vetor ou alocação dinâmica, é possível definir algumas ações para gerenciamento dos elementos contidos nela. As ações definidas por Piva Junior *et al.* (2014) são: listagem, inserção, edição, remoção e busca.

Percorrer elementos de uma lista

Uma lista duplamente encadeada nos permite percorrer os elementos nela contidos de duas formas. A primeira, e mais comumente usada, é a partir do primeiro elemento. A segunda forma, possibilitada pela característica das listas duplas, permite percorrer a lista do último ao primeiro elemento. Por exemplo, em um sistema de cadastro de pessoas físicas, com os atributos de nome, idade e CPF, a lista duplamente encadeada ficaria da seguinte forma, conforme apresentado na Figura 3.

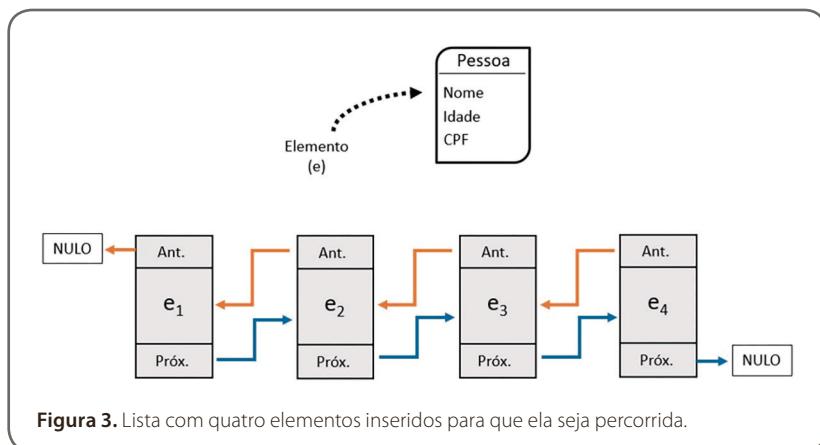


Figura 3. Lista com quatro elementos inseridos para que ela seja percorrida.

Perceba que, como a lista tem duas direções, será necessário apenas guardar o primeiro e o último elemento em uma variável, para que seja possível percorrê-la em ambas as direções.

Inserir elemento na lista dupla

A inserção de um elemento em uma lista dupla pode ser realizada de três maneiras: em seu início, em alguma posição intermediária e no final da lista. Sempre que algum elemento for adicionado à lista, é importante que sejam mantidas as ligações de referências dos apontadores para os elementos anteriores e posteriores. Além dessas formas, caso a lista esteja vazia, o novo elemento adicionado será o único item da lista, naturalmente. A Figura 4 ilustra uma inserção intermediária em uma lista encadeada dupla.

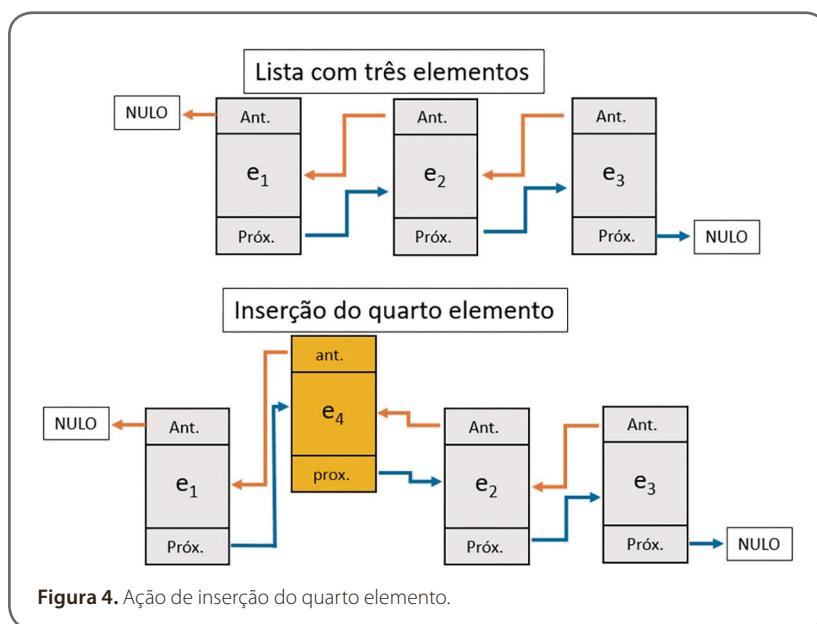


Figura 4. Ação de inserção do quarto elemento.

De forma análoga, a mesma operação pode ser realizada para inserção no início e no final da lista, sempre com os ponteiros ativos, na qual, em caso de inserção no início da lista, o seu ponteiro de **Próx.** precisará se ligar ao que passou a ser o segundo elemento da lista. Em seguida, é preciso atualizar o ponteiro **Ant.** do segundo elemento, para que ele se atualize de **NULO** para o novo primeiro elemento. Tenha sempre em mente a organização e a estrutura de uma lista com encadeamento duplo para não deixar nenhuma referência indevida.

Remover elemento na lista dupla

Assim como descrito na subseção de inserção sobre manter as referências dos ponteiros dos elementos, em se tratando de remoção, não se faz diferente, como apresentado na Figura 5. Após a remoção do último elemento, o ponteiro de próximo é apenas redirecionado para uma referência nula.

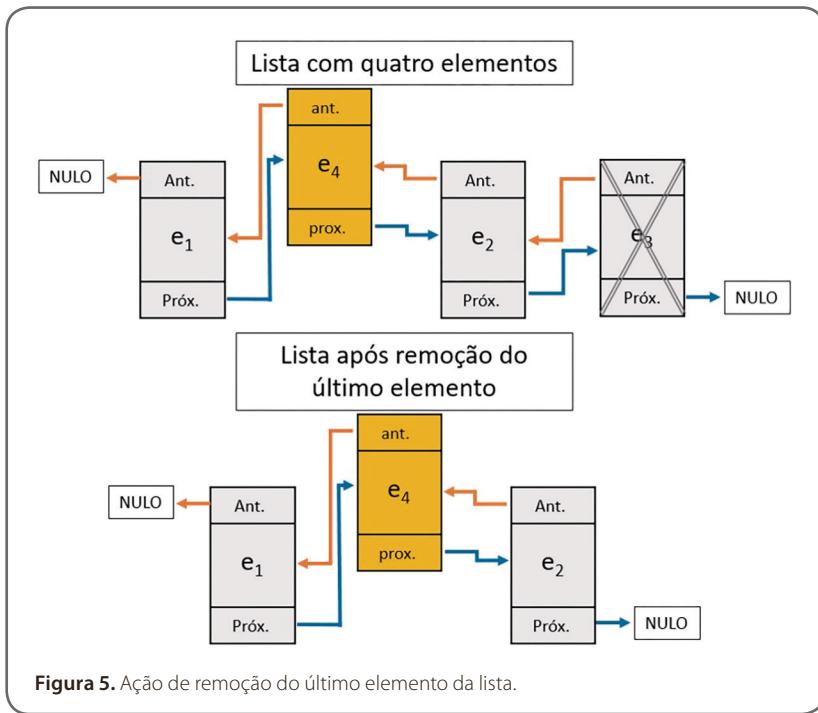


Figura 5. Ação de remoção do último elemento da lista.

Sempre que se remove um elemento, é interessante utilizar uma manipulação de memória, como o comando `free`, comumente utilizado na linguagem de programação C, em que a exclusão se efetiva após a sua liberação da memória. Em Python, isso ocorre automaticamente, de tempos em tempos, usando um mecanismo chamado Garbage Collector, que percorre a lista que contém todos os objetos registrados e verifica a quantidade de referências que cada um tem. Quando não existem referências para um objeto, o Garbage Collector desaloca a memória que havia sido previamente alocada para esse objeto.

Buscar elemento na lista dupla

A busca de qualquer elemento é feita de forma análoga ao percorrido ou à iteração da lista. De modo geral, para se buscar algum elemento na lista, seja ela simples ou dupla, precisa-se apenas fazer uma checagem simples para saber se o item a ser procurado corresponde ao elemento atual da iteração. Caso não seja o elemento atual o item a ser procurado, verifica-se o próximo elemento, no sentido da iteração (se é da esquerda para direita ou semelhantemente em caso contrário), até que seja analisado o último elemento.



Saiba mais

A eficiência é um ponto que devemos levar em consideração, principalmente quando trabalhamos com massivas cargas de dados. Em se tratando de listas duplas, é possível que um programa não suporte o adicional de dois ponteiros para cada elemento de uma lista. Para contornar esse problema, existem técnicas como a adição de um ponteiro em cada elemento que contenha a soma e a subtração (operação está inválida para ponteiros declarados em C) dos ponteiros para seus vizinhos da esquerda e da direita (TENENBAUM; LANGSAM; AUGENSTEIN, 1995).

3 Implementação de listas encadeadas duplas

Uma lista pode conter vários métodos associados a ela e não existe, segundo Celes, Cerqueira e Rangel (2004), uma definição exata de quais métodos de acesso uma lista dupla pode conter, pois cada projeto carrega sua particularidade, podendo ter alguns métodos mais conhecidos, como:

- criar uma lista vazia;
- verificar se a lista está vazia;
- verificar se a lista está cheia — para o caso de uma lista estática;
- inserir um elemento à lista;
- editar um elemento da lista;
- excluir um elemento da lista;
- acessar uma posição da lista;
- identificar o número de itens da lista;
- imprimir os elementos da lista;

- concatenar duas listas;
- inverter a ordem da lista.

Criar lista vazia

Para checarmos que a lista está vazia, precisamos, antes de tudo, declarar o objeto da lista que estamos fazendo. Para isso, precisamos criar a classe que representará a lista e, em seguida, declarar a variável com a mesma classe. A Figura 6 apresenta bem esse processo, no qual pode-se perceber a existência de duas variáveis de controle chamadas de `início` e `fim`.

```
class ListaDupla(object):
    def __init__(self):
        self.inicio = None
        self.fim = None
        print("Lista criada.")
    :
```

Figura 6. Criar e instanciar lista.

A inserção de elementos pode ser realizada de três formas: **início**, **inter-médio** e **final**. Além disso, deve atender a alguns critérios:

- verificar se existe espaço para inserir um novo elemento;
- acomodar um espaço para inserir um novo item e assegurar que os outros elementos não ficaram com uma referência vazia, a menos que os elementos sejam incluídos no final da lista;
- adicionar o contador de itens.

O método da classe `ListaDupla` para inserção de elementos pode ser visto na Figura 7.

```
def lista_vazia(self):
    """ Checa se a lista está vazia ou não. """
    return self.inicio is None

def adicionar(self, info):
    """ Adiciona um novo Elemento no fim da lista. """
    novo_e = Elemento(info)

    if self.lista_vazia():
        self.inicio = novo_e
        self.fim = novo_e
    else:
        # O anterior 'aponta' para o último Elemento adicionado (fim)
        novo_e.ant = self.fim
        # O próximo sempre aponta para None
        novo_e.prox = None
        # O próximo do fim sempre aponta para o novo elemento
        self.fim.prox = novo_e
        # ... e o atributo fim passa a ser o novo Elemento
        self.fim = novo_e
    print("Elemento adicionado: {}".format(novo_e.info))
```

Figura 7. Método de inserção do novo elemento à lista.

A forma de inserção apresentada na Figura 7 ilustra a inserção no fim da lista. Para o caso de não existir nenhum elemento na lista, as variáveis de controle das pontas da lista apontarão para o mesmo elemento, uma vez que tendo apenas um elemento, o início também marcará o final da lista. No entanto, para o caso em que haja mais de um elemento contido na lista, é realizada uma nova estratégia.

Logo, para cada elemento adicionado a uma lista não vazia, o novo elemento precisará se ligar ao último elemento já inserido nela. Essa operação é possível graças à natureza das listas duplamente encadeadas, as quais permitem que o novo elemento se ligue ao anterior pelo ponteiro nomeado na classe `Elemento` como `ant`, em alusão ao elemento localizado à esquerda do novo elemento da lista.

Remoção do elemento

A remoção do elemento na lista é realizada de forma sistemática, em que se verifica primeiro se o elemento que está sendo procurado se encontra na lista (Figura 8). Caso seja encontrado e seja o primeiro elemento da lista, basta verificar se o elemento atual tem referência nula no apontamento do elemento anterior. Se essa verificação não for verdadeira, é um indicativo de que o elemento não é o primeiro elemento da lista, bastando fazer um movimento de pulo em relação ao elemento atual.

```
def remover(self, info):
    """ Remove um elemento da lista. """
    # O Elemento atual é o primeiro item da lista
    e_atual = self.inicio

    # É feito uma busca no Elemento que será removido
    # enquanto o Elemento atual for válido
    while e_atual is not None:
        # Verifica se é a info da busca
        if e_atual.info == info:
            # Se a info da busca for o primeiro
            # da lista, o anterior não existe
            if e_atual.ant is None:
                # O início passa a ser o próximo Elemento da lista
                self.inicio = e_atual.prox
                # E o anterior do próximo Elemento aponta para None
                e_atual.prox.ant = None
            else:
                # Caso esteja no meio, redireciona os ponteiros direcionais
                e_atual.ant.prox = e_atual.prox
                e_atual.prox.ant = e_atual.ant
            # Imprime e finaliza o laço de repetição.
            print("\nElemento {0} removido. \n".format(e_atual.info))
            break
        # Se não é o Elemento que estamos buscando vá para o próximo
        e_atual = e_atual.prox
```

Figura 8. Remover um elemento da lista.

Percorrer os elementos da lista

Os elementos de uma lista dupla podem ser lidos tanto de forma crescente quanto de maneira decrescente, sem dificuldades ou arranjo em programação, graças à sua estrutura de ponteiro duplo. A Figura 9 apresenta a forma de amostragem da lista de forma crescente à ordem de inserção.

```
def imprimir_crescente(self):
    """ Apresenta os dados da lista em ordem crescente. """
    # O Elemento atual é o primeiro elemento da lista
    e_atual = self.inicio
    elemento = ""
    # Para cada elemento válido da lista
    while e_atual is not None:
        if e_atual.ant is None:
            elemento += "None "
        elemento += "<-> | " + str(e_atual.info) + " | "
        if e_atual.prox is None:
            elemento += "<-> None"
        # próximo elemento da lista
        e_atual = e_atual.prox
    print("Em ordem Crescente: \n")
    print (elemento)
    print ("="*60)
```

Figura 9. Modo crescente de impressão da lista.

No entanto, a Figura 10 apresenta a forma de impressão de forma decrescente à ordem de inserção. A mudança feita em relação à forma crescente é que, ao invés da lista começar da variável denominada como `inicio`, ela começa com a variável `fim`, e vai sendo iterada pelos elementos apontados pela referência `ant` de cada elemento até atingir a referência nula.

```

def imprimir_decrecente(self):
    """ Apresenta os dados da lista em ordem decrescente. """
    # O Elemento atual é o ultimo elemento da lista
    e_atual = self.fim
    elemento = ""
    # Para cada elemento válido da lista
    while e_atual is not None:
        if e_atual.prox is None:
            | | elemento += "None"
            elemento += "<-> | " + str(e_atual.info) + " | "
        if e_atual.ant is None:
            | | elemento += "<-> None"
        # anterior elemento da lista
        e_atual = e_atual.ant
    print("Em ordem Decrescente: \n")
    print (elemento)
    print ("="*60)

```

Figura 10. Modo decrescente de impressão da lista.

Uma vez que terminamos as principais funções, podemos efetuar os seguintes testes para avaliar o seu funcionamento, conforme apresentado na Figura 11. Na lateral esquerda da figura, é apresentado o código implementado, seguido dos resultados à direita.

```

# instancia a lista dupla
lista = ListaDupla()
print("Lista vazia?: "+str(lista.lista_vazia()))
}

# adiciona elementos à lista
lista.adicionar(5)
lista.adicionar(15)
lista.adicionar(25)
lista.adicionar(35)

}

# Imprime os elementos nas duas direções
lista.imprimir_crescente()
lista.imprimir_decrecente()

}

# Remove o elemento de valor 15
lista.remover(15)

}

# Imprime os elementos nas duas direções
lista.imprimir_crescente()
lista.imprimir_decrecente()

```

Lista criada.
Lista vazia?: True

Elemento adicionado: 5.
Elemento adicionado: 15.
Elemento adicionado: 25.
Elemento adicionado: 35.

Em ordem Crescente:
None <-> | 5 | <-> | 15 | <-> | 25 | <-> | 35 | <-> None

Em ordem Decrescente:
None <-> | 35 | <-> | 25 | <-> | 15 | <-> | 5 | <-> None

Elemento 15 removido.

Em ordem Crescente:
None <-> | 5 | <-> | 25 | <-> | 35 | <-> None

Em ordem Decrescente:
None <-> | 35 | <-> | 25 | <-> | 5 | <-> None

Figura 11. Teste das funções implementadas com os respectivos resultados.



Link

O *link* a seguir contém informações sobre listas duplas e serve como um bom referencial teórico sobre o tema.

<https://qrgo.page.link/9pwP7>



Referências

CELES, W.; CERQUEIRA, R.; RANGEL, J. L. *Introdução a estrutura de dados: com técnicas de programação em C*. Rio de Janeiro: Campus, 2004.

EDELWEISS, N.; GALANTE, R. *Estrutura de dados*. Porto Alegre: Bookman, 2009.

PIVA JUNIOR, D. et al. *Estrutura de dados e técnicas de programação*. Rio de Janeiro: Elsevier, 2014.

TENENBAUM, A. M; LANGSAM, Y.; AUGENSTEIN, M. J. *Estruturas de dados usando C*. São Paulo: MAKRON Books, 1995.



Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS