ESTRUTURAS DE DADOS

prof. Alexandre César Muniz de Oliveira

Sugestão bibliográfica:

- **ESTRUTURAS DE DADOS USANDO C**

Aaron M. Tenenbaum, et alli

- **DATA STRUCTURES, AN ADVANCED APPROACH USING C**

Jeffrey Esakov & Tom Weiss

- **ESTRUTURAS DE DADOS E ALGORITMOS EM JAVA (2ED)**

Michael Godrich & Roberto Tamassia

LISTAS ENCADEADAS

1. Motivação

Vetores, em geral, utilizam memória pré-alocada, estática e pouco flexível para inclusões e remoções de novos elementos. Vetores dinâmicos também dificultam a manutenção de elementos. Uma alternativa é alocação de pequenas porções de memória ligadas entre si.

2. Definição

Listas encadeadas *(linked lists)* são coleções de objetos ordenados de forma que cada objeto guarda, além de dados, informação sobre o endereço do próximo objeto da lista, formando uma cadeia de objetos.

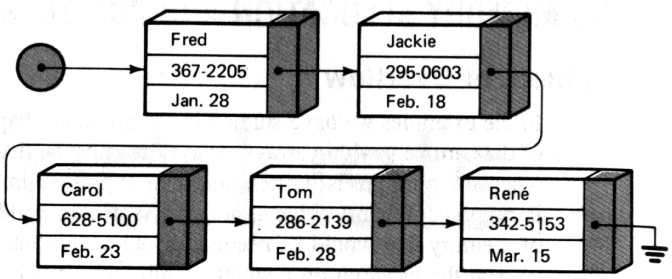


Figure 4.2. A linked list

3. Vantagens e desvantagens

↑ Alocação de memória sob demanda
↑ Facilidade de manutenção
↓ Memória extra
↓ Acesso seqüencial

4. Propriedades

- a lista pode ter 0 ou infinitos itens
- um novo item pode ser incluído em qualquer ponto
- qq item pode ser removido
- qq item pode ser acessado
- permite muitas operações

5. Notação de listas

- *info (p)* - informação (INFO) apontada por *p;*
- *prox (p)* - próximo nó (PROX) apontado por *p* (endereço);
- *aloca( )* - aloca memória para um nó (endereço);
- *libera(p)* - libera a memória apontada por (*p*);
- *vazia(p)* - teste se lista apontada por *p* está vazia (booleano);

LISTAS ENCADEADAS

5.1 Nó da lista

Elemento = Nó = INFO + Elo para PROX

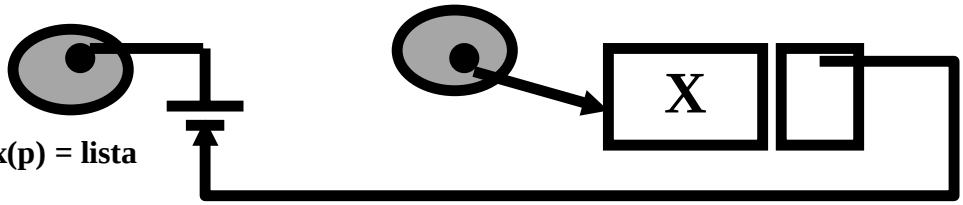
NÓ

Lista Vazia

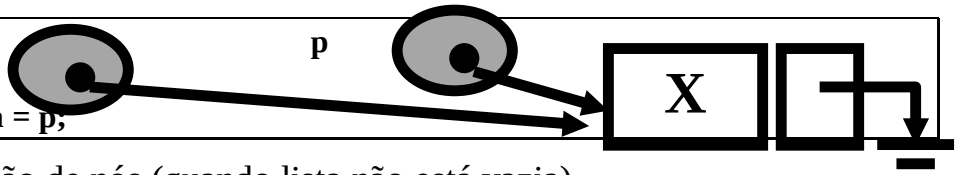
6. Operações com listas encadeadas

6.1 Inserção de nós (inicialmente vazia)

a) p=aloca();
b) info(p) = x;



c) prox(p) = lista

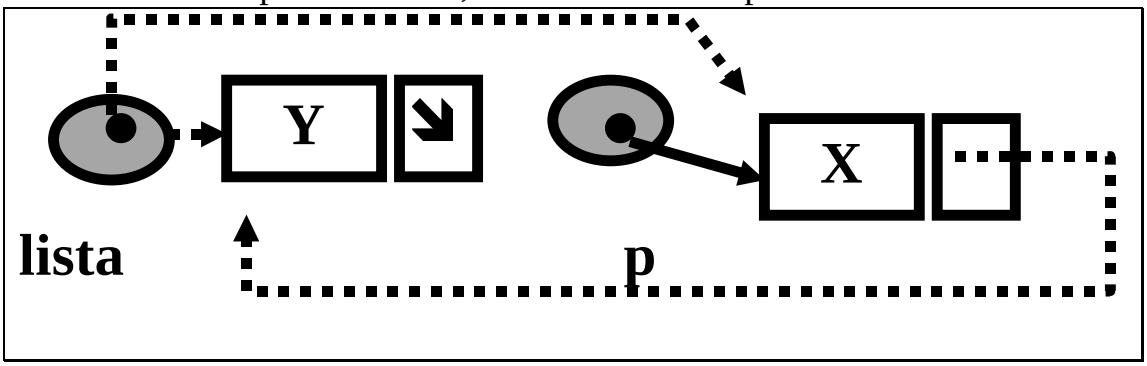


d) lista = p;

6.2 Inserção de nós (quando lista não está vazia)

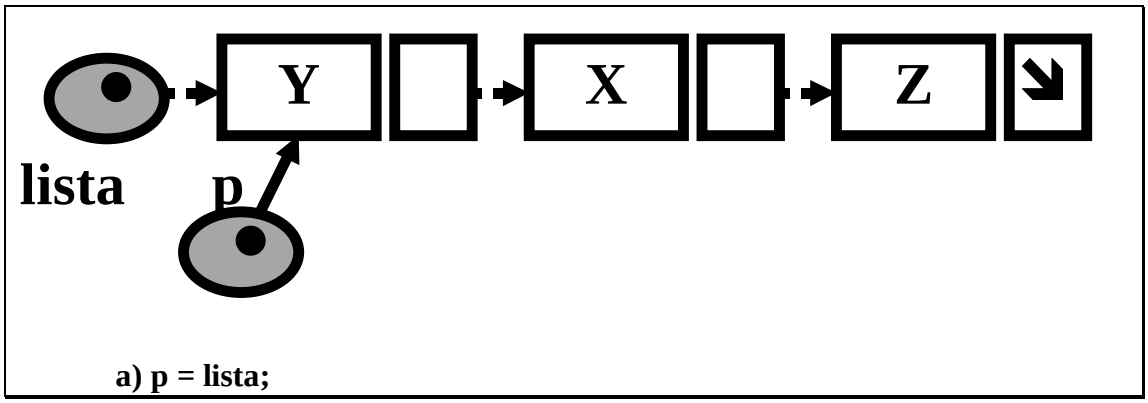
a) p=aloca();
b) info(p) = x;
c) prox(p) = lista
d) lista = p;

Funciona perfeitamente, inserindo nós sempre no início

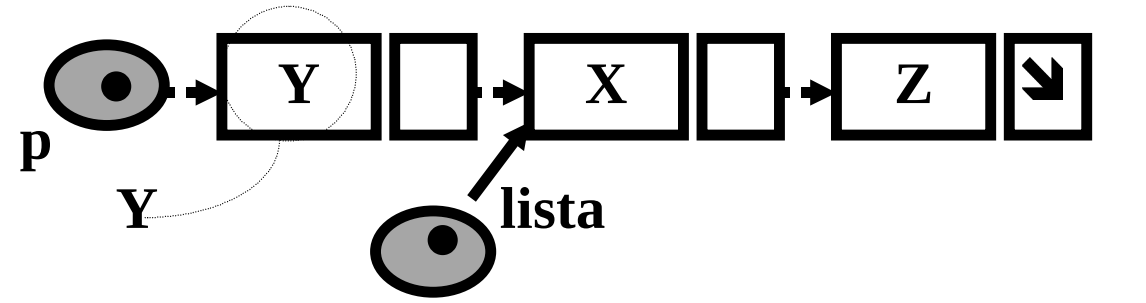


LISTAS ENCADEADAS

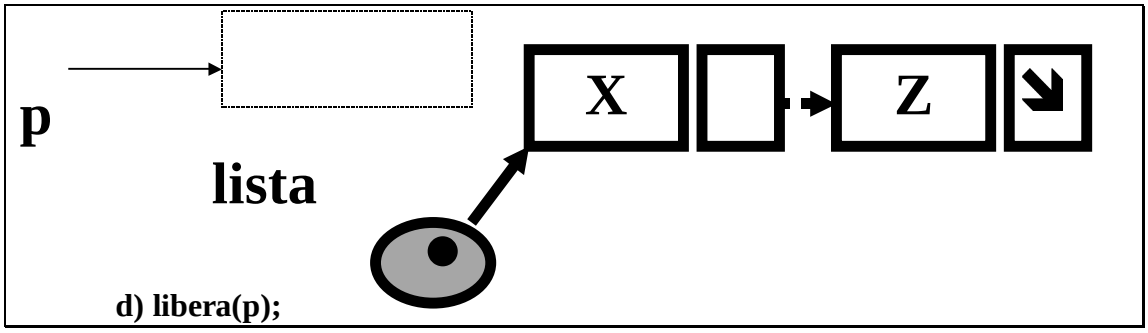
6.2 Remoção de nós (início da lista)



a) p = lista;



b) lista = prox(p);
c) aux = info (p);



d) libera(p);

LISTAS ENCADEADAS

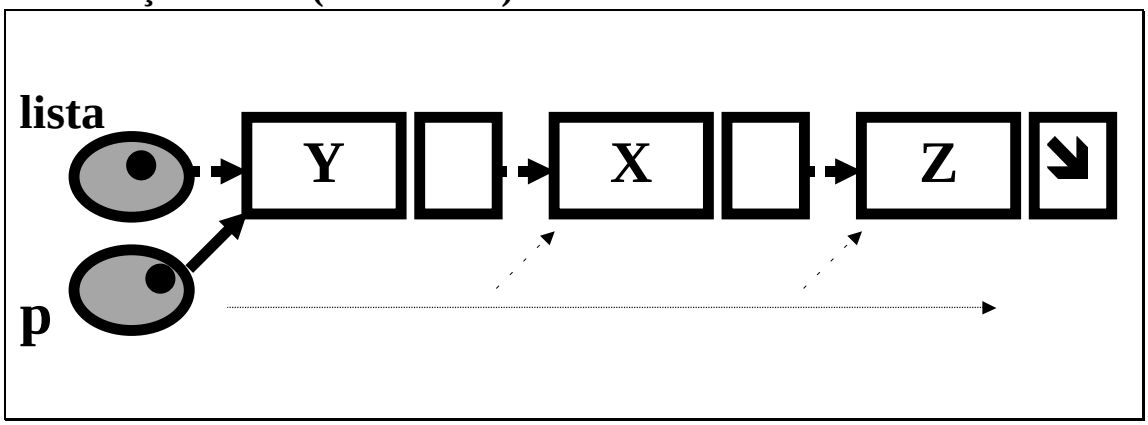
7. Implementação de Pilhas usando listas encadeadas

- Push = Insere_Inicio_lista √
- Pop = Remove_Início_lista √
- Empty = Vazia_lista √

8. Implementação de Filas usando listas encadeadas

- Insert = Insere_Fim_lista ?
- Remove = Remove_Início_lista √
- Empty = Vazia_lista √

- **Inserção de nós (fim da lista)**

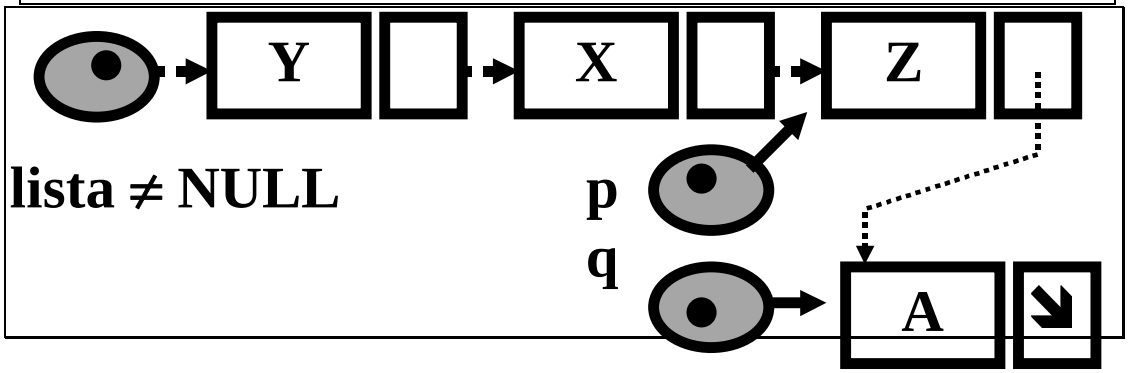


- **Para percorrer uma lista até o último nó**

```
p=lista;
if (p <> NULL) {
        while ( prox(p) <> NULL ) p = prox (p);
}
return (p);
```

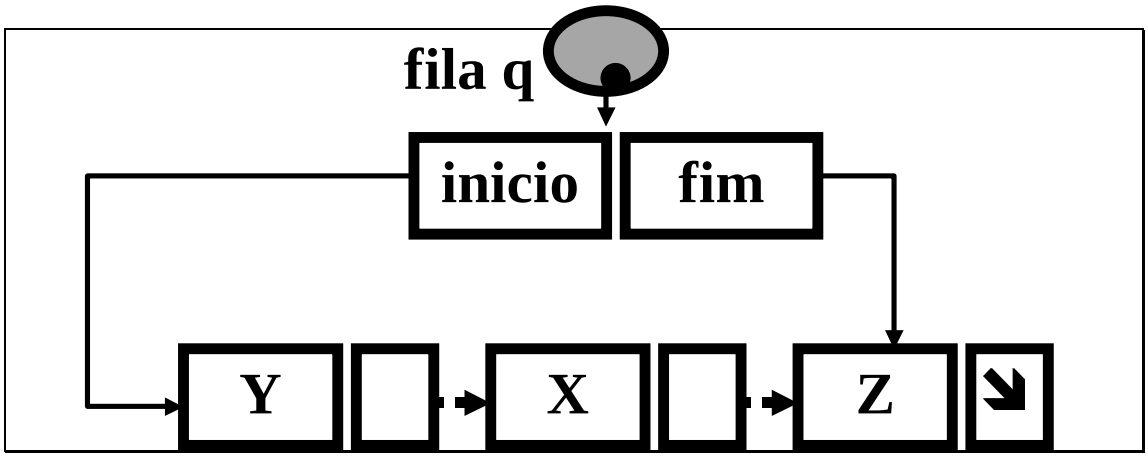
- **Para inserir depois do último nó**

```
p=Percorre(lista);
q=aloca();
info(q)=a;
prox(q)=NULL  /*q = aloca(a)*/
if ( p = NULL )        lista = q;
else                   prox(p) = q;
```



LISTAS ENCADEADAS

- **Para inserir depois do último nó em listas longas**
- **Lista encadeada com dois ponteiros**



- **Notação de listas**
  - **inicio(q)** — **elo para o início da fila q**
  - **fim(q)** — **elo para o fim da fila q**
- **Inicializa (fila q)**

  inicio (q) = fim (q) = NULL;
- **Vazia (fila q)**

  inicio (q) = NULL ? TRUE : FALSE

- **Insere (fila q, x)**

```
p=aloca(x);
if (p = NULL) return (FALSE);
if (Vazia(q)) inicio (q) = fim(q) = p;
else { prox(fim(q)) = p;
        fim(q) = p;
}
return (TRUE);
```

- **Remove (fila q)**

```
if (Vazia(q)) Erro ("Underflow");
else {
        p = inicio (q); x = info (p);
        inicio (q) = prox (p);
        if (Vazia(q)) fim (q) = NULL;
        libera(p);
        return(x);
}
```

LISTAS ENCADEADAS

9 Implementação de Listas Encadeadas

- **Usando vetores**

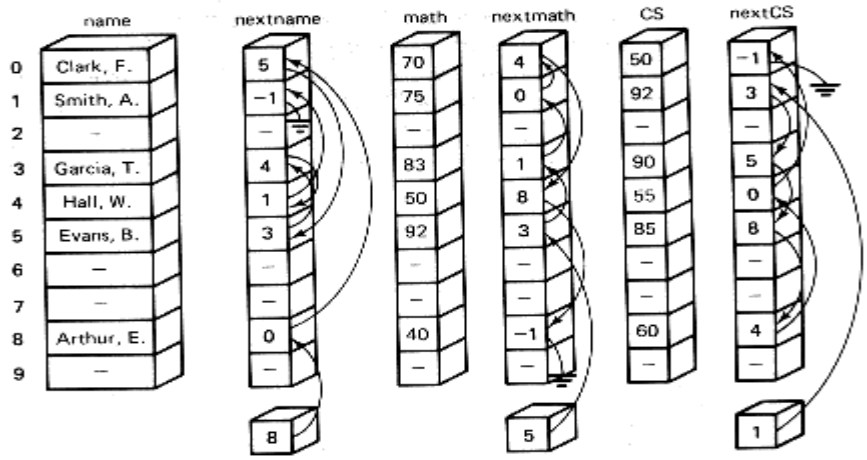


Figure 4.16. Linked lists in arrays

Lista de espaços vazios que, inicialmente, teria MAX_LISTA

- **Usando ponteiros genéricos**

```
#define Info(p)                 (p) -> Info
#define Prox(p)                 (p) -> Prox
#define TAM_ITEM                sizeof ( struct Item )
typedef void * TipoDado;
struct Item {
        TipoDado    Info;
        struct Item * Prox;
};
typedef struct Item*  PtrLista;
typedef Boolean       unsigned char
PtrLista              lista;
```

- ```
  Boolean Insere_Inicio ( PtrLista * lista, TipoDado x)
  {     PtrLista p;
        p = Aloca ( x );
        if ( ! p ) return ( FALSE );
        Prox(p) = *lista;
        *lista = p;
        return (TRUE);
  }
  ```
  ```

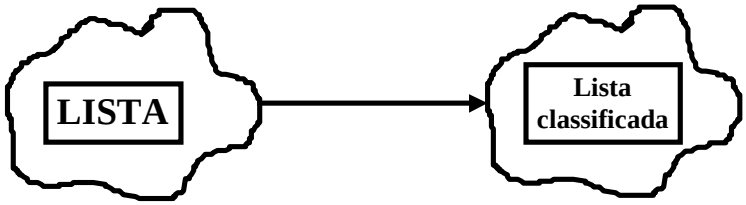
LISTAS ENCADEADAS

- **Usando ponteiros genéricos**

- ```
  Boolean Remove_Fim ( PtrLista * lista , TipoDado * x )
  { PtrLista p,q;
      if (Vazia(lista)) return FALSE;
      q = NULL;
      for ( p = *lista; Prox(p) < > NULL; p = Prox(p))
            q = p;
      if ( q = NULL ) Remove_Inicio (lista , x);
      else {
            Prox(q) = NULL;
            *x = Info(p);
            Libera(p);
      }
      return (TRUE);
  ```

- ```
  Boolean Procura_Item(PtrLista *lista, TipoDado x)
  { ptr_lista p;
    for (p= *lista ;p < > NULL && !Igual (Info(p), x); p=Prox(p));
    return (p);
  }
  ```

10. Listas Classificadas



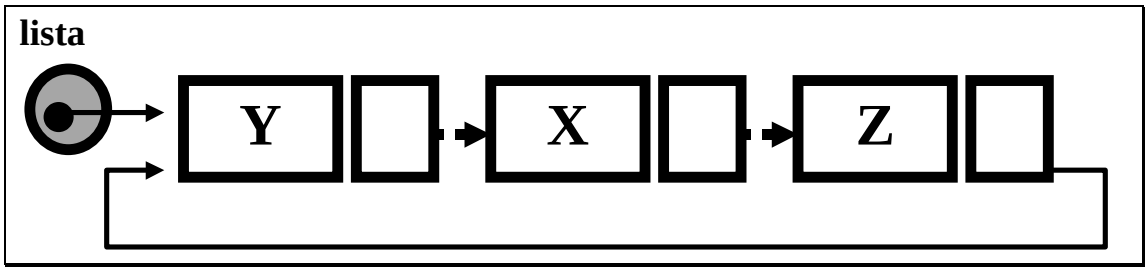

- ```
  Insere_Ordenado (PtrLista *lista, TipoDado x)
  { ptr_lista p;
    for (p= *lista ;p < > NULL && Maior(x, Info(p)); p=Prox(p))
            q = p;
    if ( ! q )      Insere_Inicio (lista, x );
    else            Insere_Depois (lista, x );
  }
  ```

- ```
  Boolean Pesquisa_Ordenado (PtrLista *lista, TipoDado x)
  { ptr_lista p;
    for (p= *lista ;p < > NULL; p=Prox(p)) {
            if ( Igual (Info(p), x) ) return (TRUE);
            if ( Maior (Info(p), x) return (FALSE);
    }
    return (FALSE);
  }
  ```

LISTAS ENCADEADAS

11. Listas Circulares

Ultimo elemento da lista aponta para o primeiro. Assim, todos os nós são acessíveis a partir de qualquer nó, mas sempre de forma seqüencial.



- **Operações com Listas Circulares**
- *Boolean IniciaListCirc (PtrLista *lista)*
  *{ *lista = NULL; return TRUE; }*
- *Boolean VaziaListCirc (PtrLista *lista)*
  *{ return (*lista = = NULL ? TRUE : FALSE); }*
- *Boolean InserListCirc (PtrLista *lista, TipoDado x)*

```
{       list p;
        if ( ( p=aloca(x) ) = = ERRO ) return FALSE;
        if ( VaziaListCirc( lista ) ) {
                Prox(p) = p;
                *lista = p;
        }
        else {
                Prox(p) = Prox (*lista);
                Prox(*lista) = p;
                *lista=p;
        }
        return TRUE;
}
```

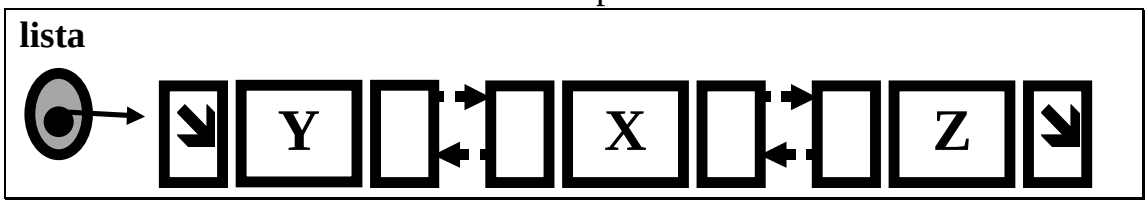
- *Boolean RemListCirc (PtrLista *lista, TipoDado *x)*

```
{ list p;
        if ( VaziaListCirc( lista ) ) return FALSE;
        p = Prox(*lista);
        Prox(*lista) = Prox(p);
        *x = Info(p);
        if (p = = *lista) *lista =NULL;
        libera(p);
        return(TRUE);
}
```

LISTAS ENCADEADAS

12. Listas Duplamente Encadeadas

Limitações das implementações anteriores de listas encadeadas que só podem ser percorridas em uma direção e, por isso, dificulta a remoção de nós arbitrários, sendo necessário para isso percorrer a lista inteira. Listas duplamente encadeadas permitem percurso seqüencial em ambos os sentidos através de o acréscimo de um ponteiro no nó da lista.



Estrutura do Nó

```
typedef struct ItemDup*          PtrListDup;
struct ItemDup {
        TipoDado        info;
        PtrListDup      * prox;
PtrListDup      *ante;
};
#define Ante(p) ((p) -> ante)
PtrListDup              AlocaDup(TipoDado x)
Boolean                 IniListDup (PtrListDup *lista)
Boolean                 VaziaListDup (PtrListDup *lista)
```

- `Boolean  InsListDup (PtrListDup *pno, TipoDado x)`

```
{       PtrListDup p;
        p = AlocaDup(x);
        if (! p ) return (FALSE);
        if (VaziaListDup (pno) ) Ante(p) = Prox(p) = NULL;
        else {
                Prox(p) = *pno;
                Ante(p) = Ante (*pno);
                Ante(*pno) = p;
                if ( Ante(p) != NULL ) Prox (Ante (p) ) = p;
        }
        *pno = p;
        return TRUE;
}
```

- `Boolean  RemListDup (PtrListDup *lista, PtrListDup pno)`

```
{
        if (VaziaListDup(*lista) ) return FALSE
        if (Ante(pno) !=NULL) Prox(Ante(pno)) = Prox(pno);
        if (Prox(pno) !=NULL) Ante(Prox(pno)) = Ante(pno);
        if (pno = = *lista) {
                if ( Prox(pno) != NULL) *lista = Prox (pno);
                else *lista = Ante(pno);
        }
        LiberaDup(pno);
        return TRUE;
```