

ED62A-COM2A

ESTRUTURAS DE DADOS

Aula 05 - Listas ordenadas
(Estrutura dinâmica)

Prof. Rafael G. Mantovani

Roteiro



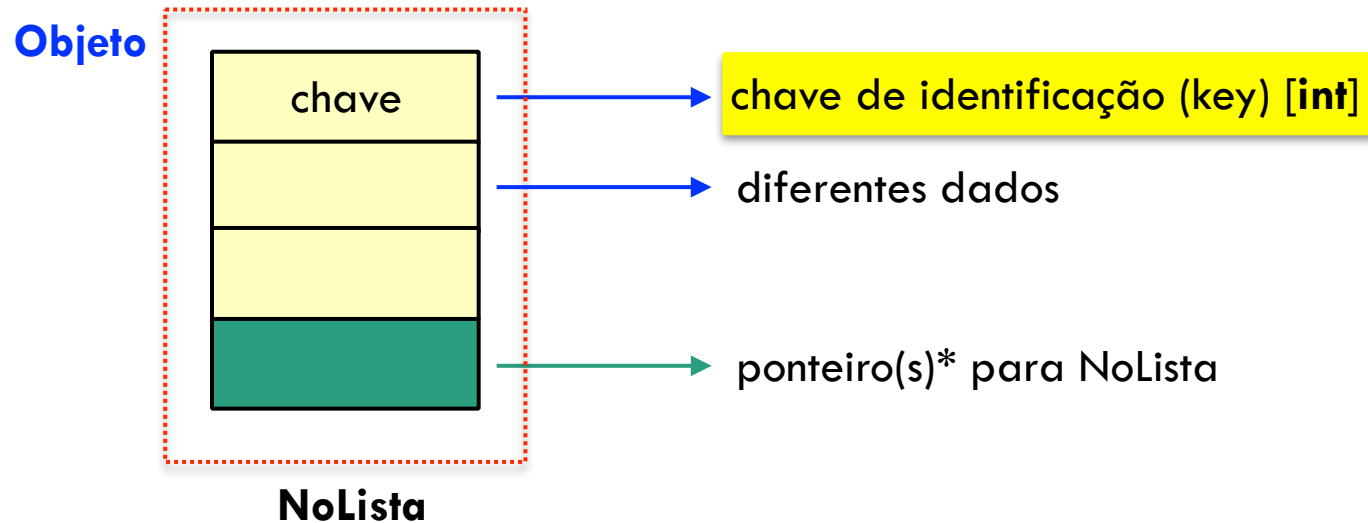
- 1 Listas Ordenadas**
- 2 Operações gerais**
- 3 Inserção de elementos**
- 4 Pesquisa de elementos**
- 5 Remoção de elementos**
- 6 Referências**

Roteiro

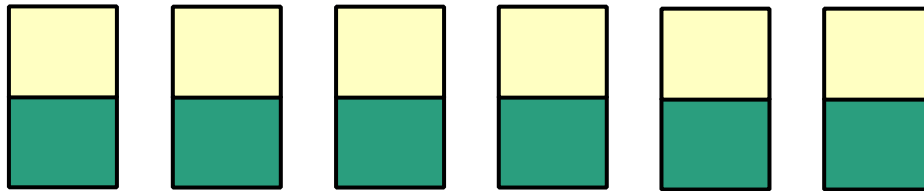
- 1 Listas Ordenadas**
- 2 Operações gerais**
- 3 Inserção de elementos**
- 4 Pesquisa de elementos**
- 5 Remoção de elementos**
- 6 Referências**

Lista dinâmica

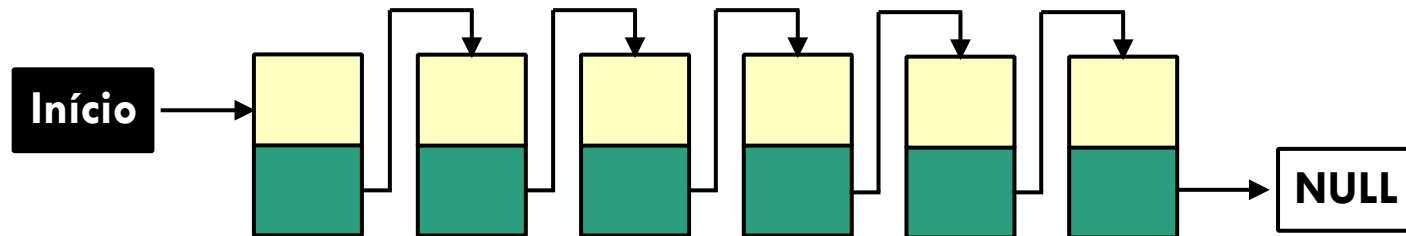
- Nós de Lista



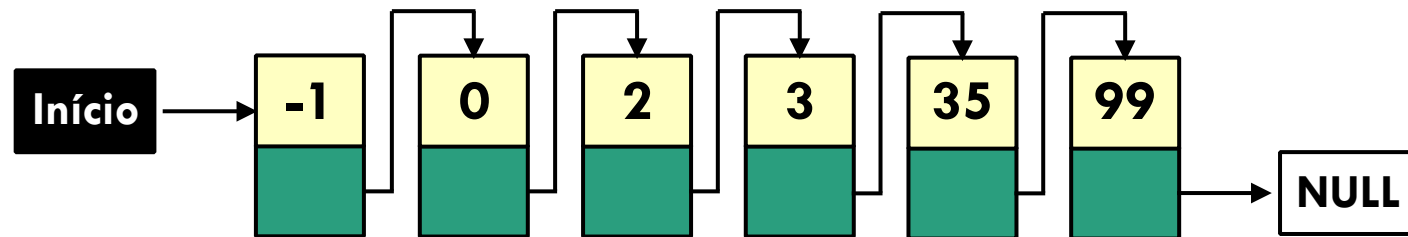
Lista dinâmica



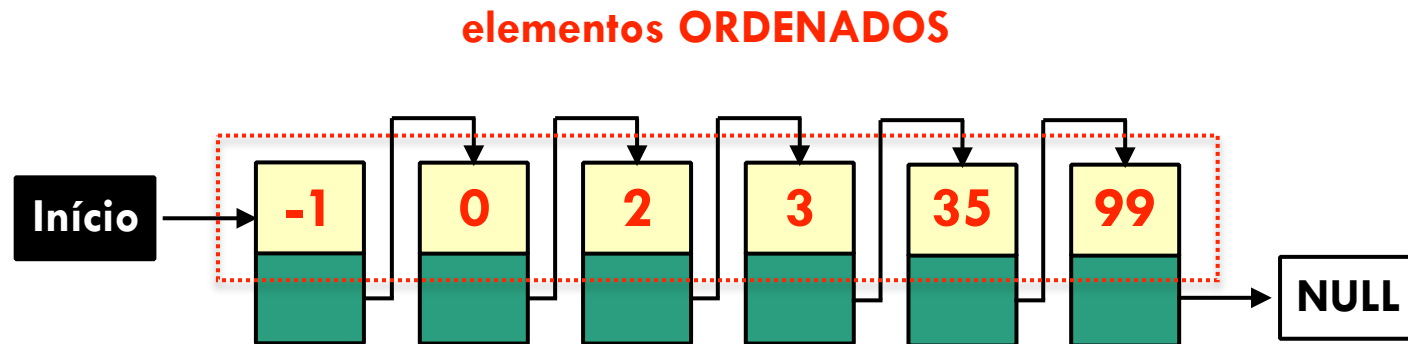
Lista dinâmica



Lista dinâmica



Lista dinâmica



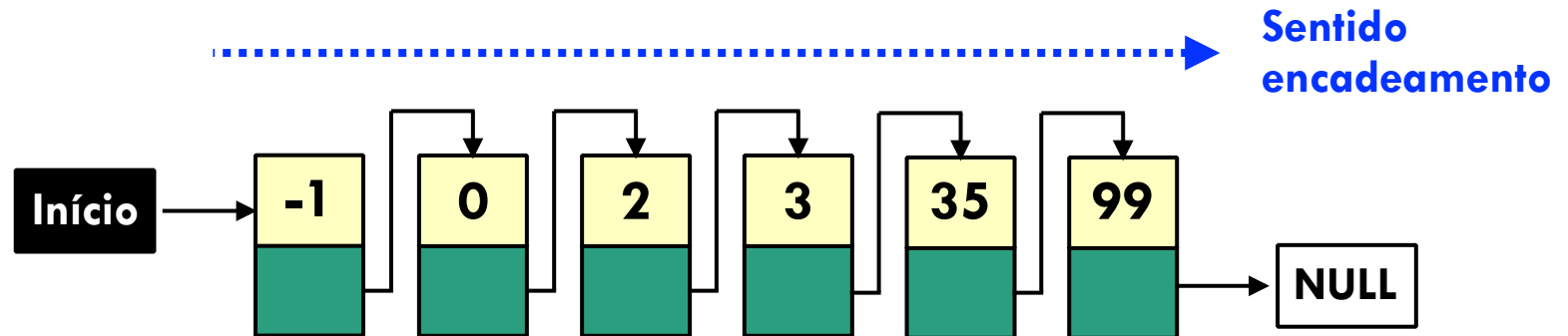
Tipos de lista

- Diferentes implementações de lista dinâmica:
 - A** Single-linkage: singularmente encadeada
 - B** Double-linkage: duplamente encadeada
 - C** Circulares: nó sentinela

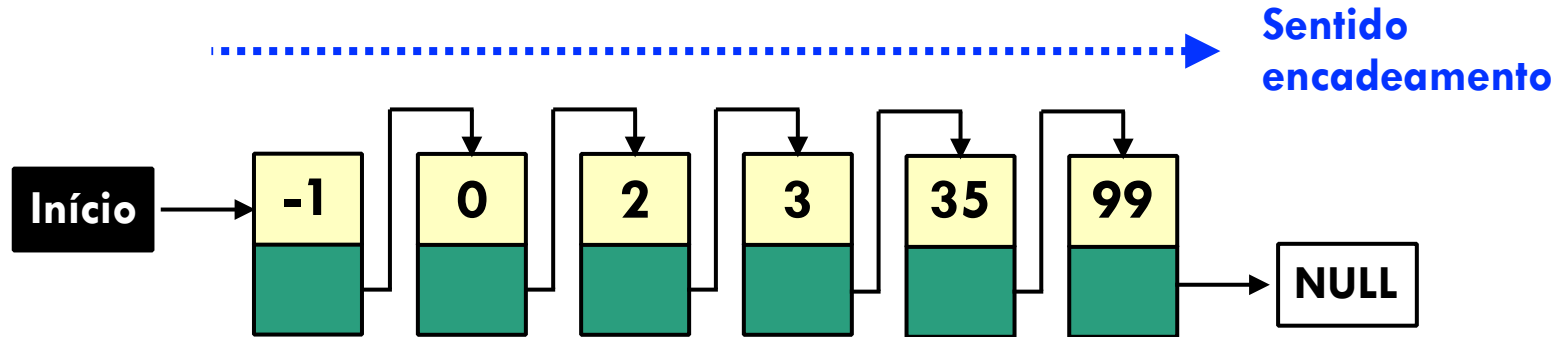
Tipos de lista

- Diferentes implementações de lista dinâmica:
 - A** Single-linkage: singularmente encadeada
 - B** Double-linkage: duplamente encadeada
 - C** Circulares: nó sentinela

Tipos de lista



Tipos de lista



tipo **Lista**

Início ***NoLista**

Número de elementos :

tipo **NoLista**

Obj

Item armazenado

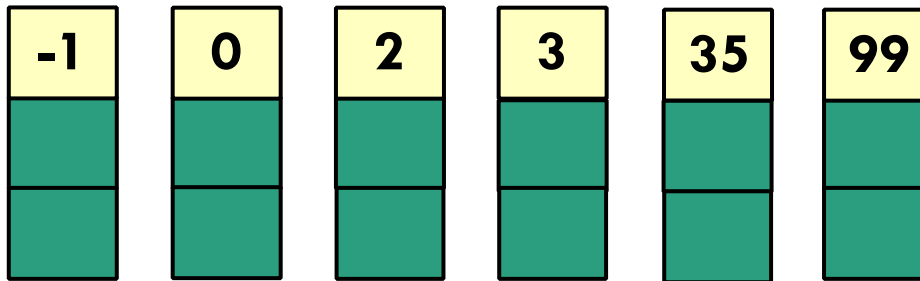
Prox

***NoLista**

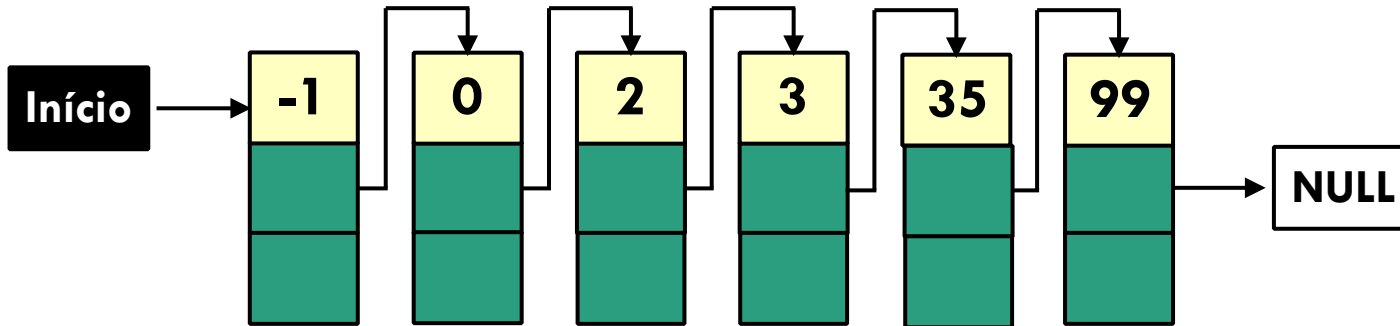
Tipos de lista

- Diferentes implementações de lista dinâmica:
 - A** Single-linkage: singularmente encadeada
 - B** Double-linkage: duplamente encadeada
 - C** Circulares: nó sentinela

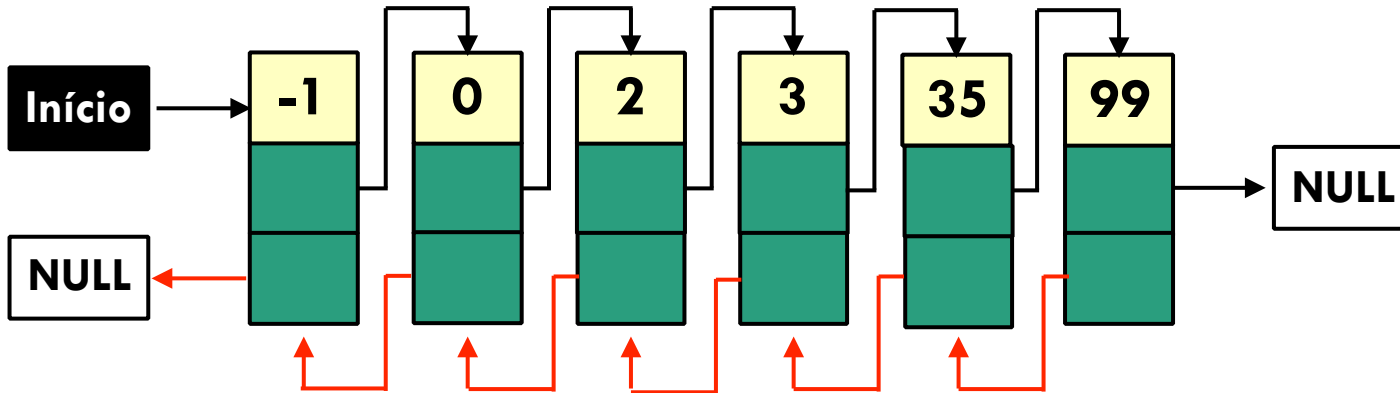
Tipos de lista



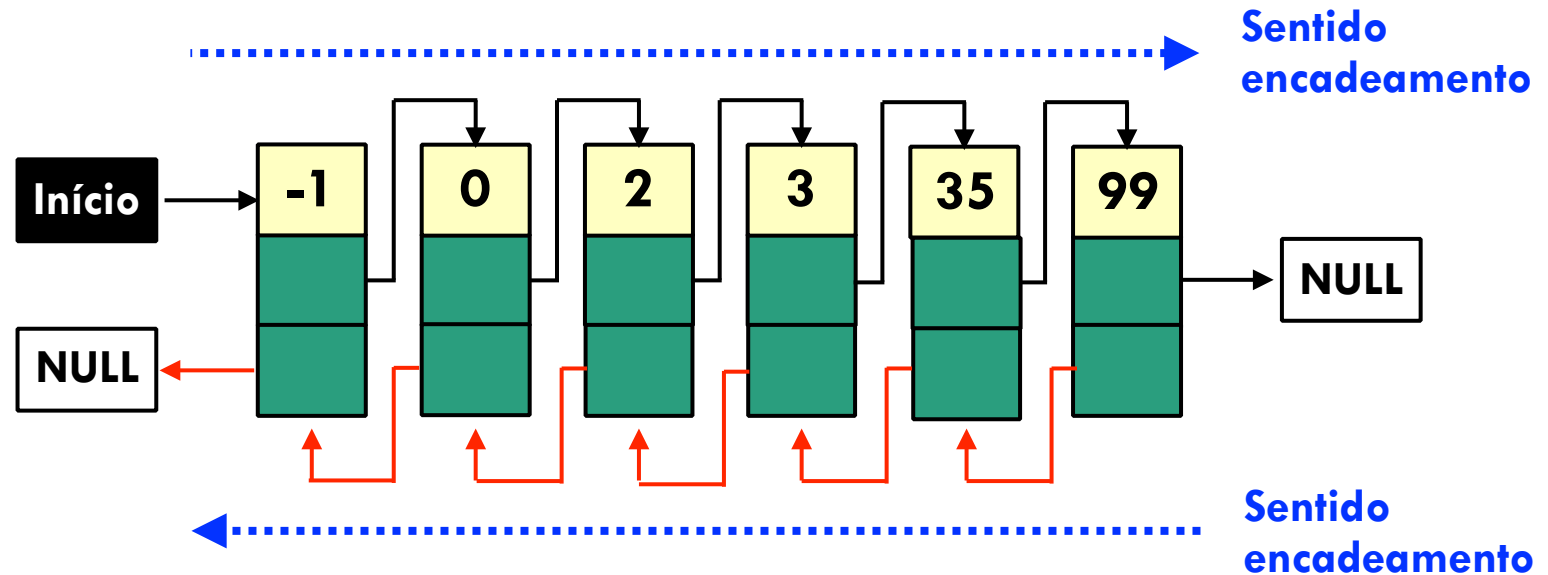
Tipos de lista



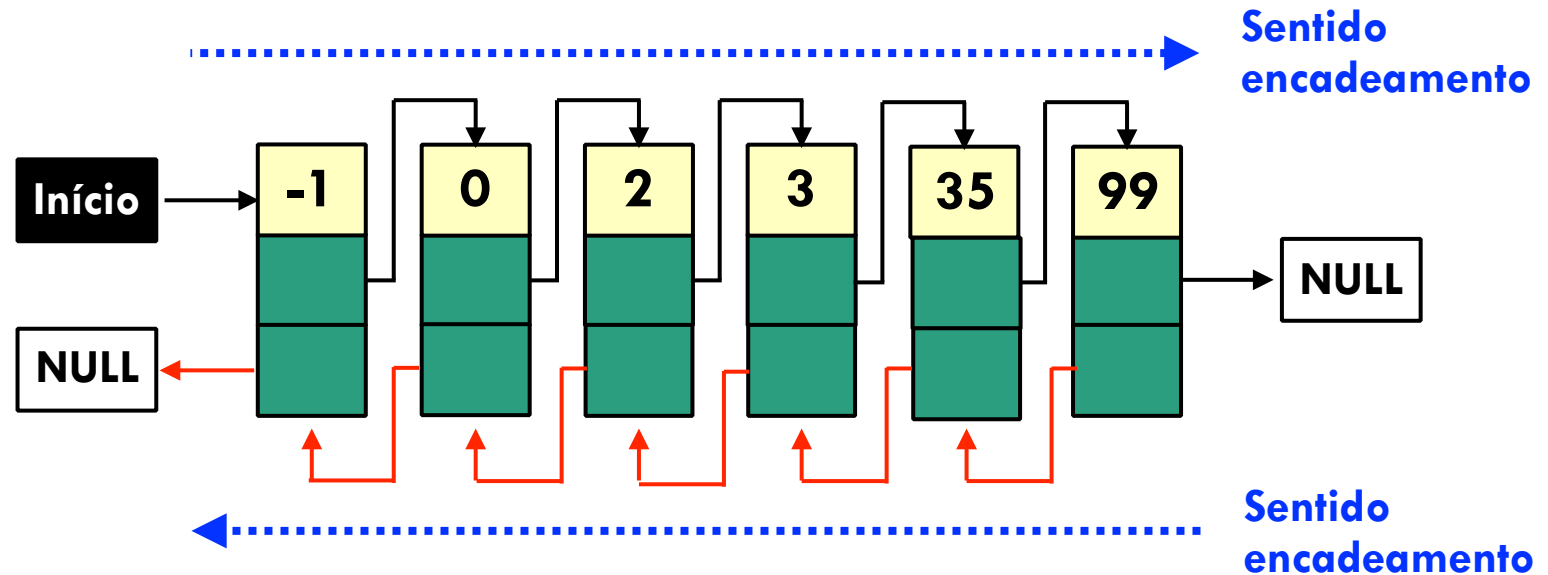
Tipos de lista



Tipos de lista



Tipos de lista



tipo **Lista**

Início ***NoLista**

Número de elementos :

tipo **NoLista**

Obj

Item armazenado

Prox

***NoLista** (proximo)

Ant

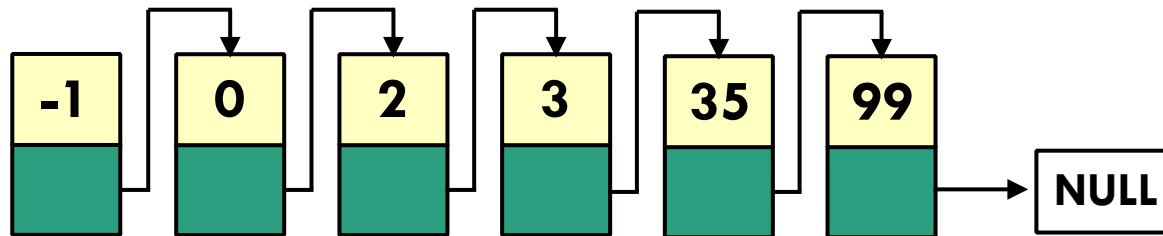
***NoLista** (anterior)

Tipos de lista

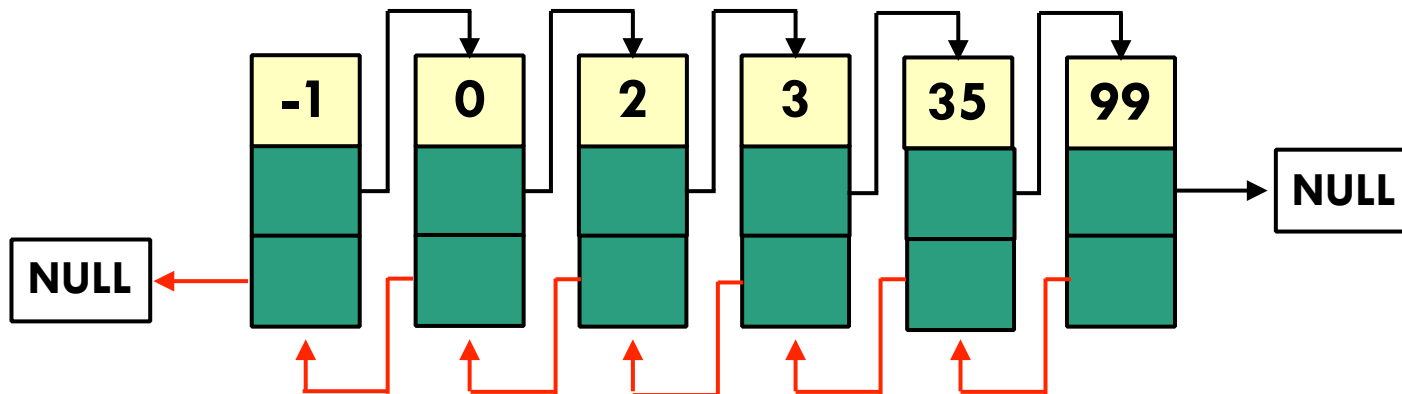
- Diferentes implementações de lista dinâmica:
 - A** Single-linkage: singularmente encadeada
 - B** Double-linkage: duplamente encadeada
 - C** Circulares: nó sentinela

Tipos de lista

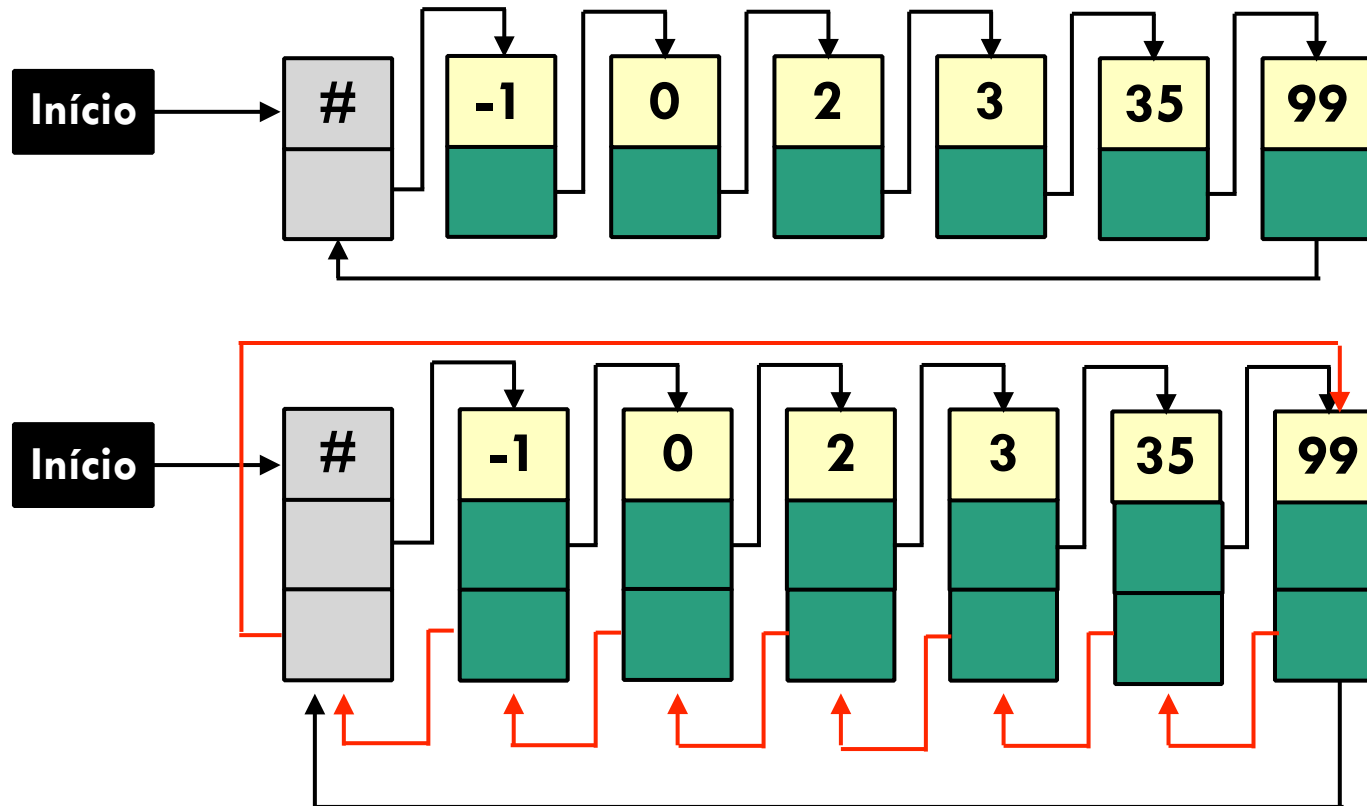
Início



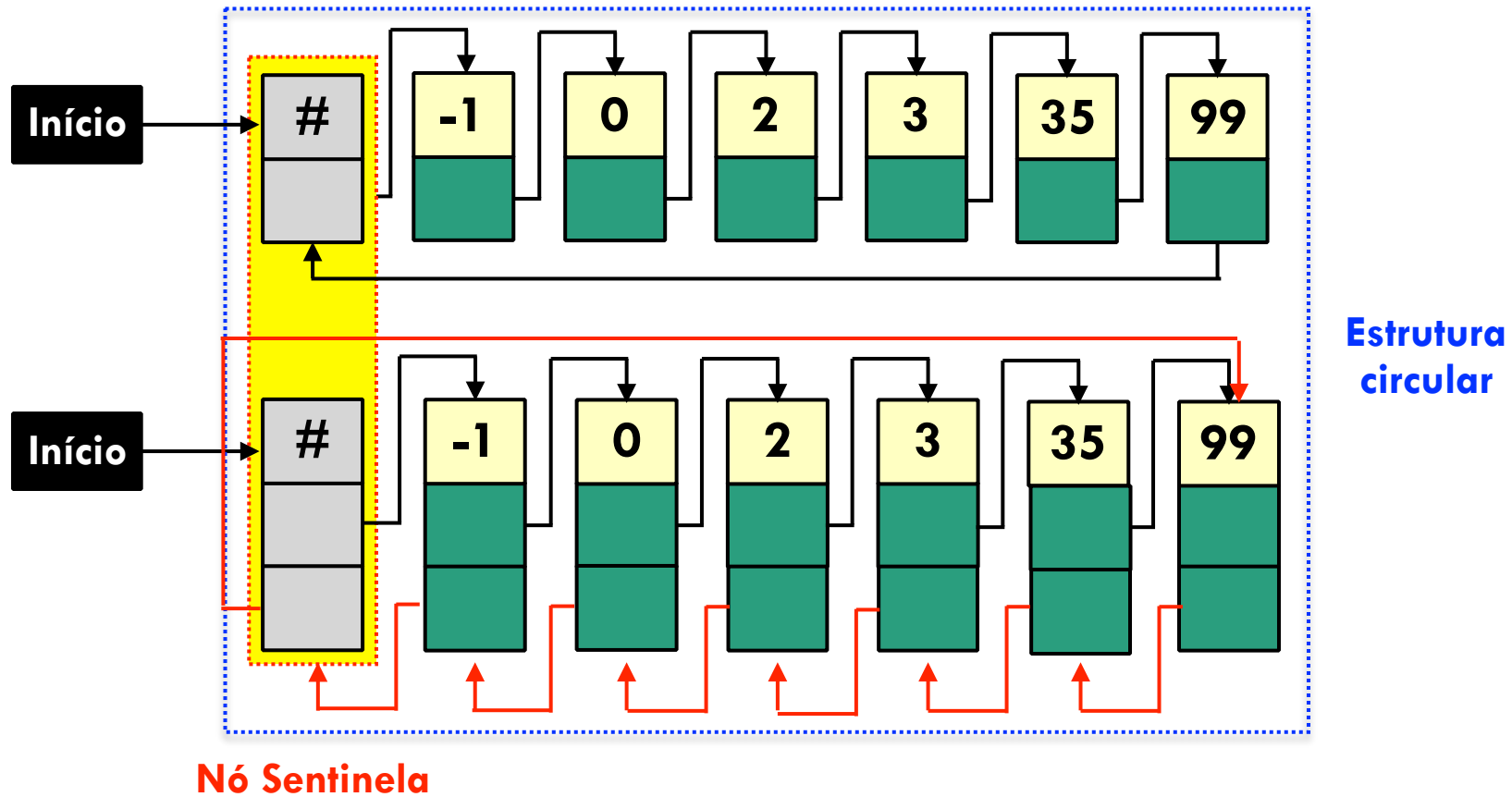
Início



Tipos de lista



Tipos de lista



Roteiro

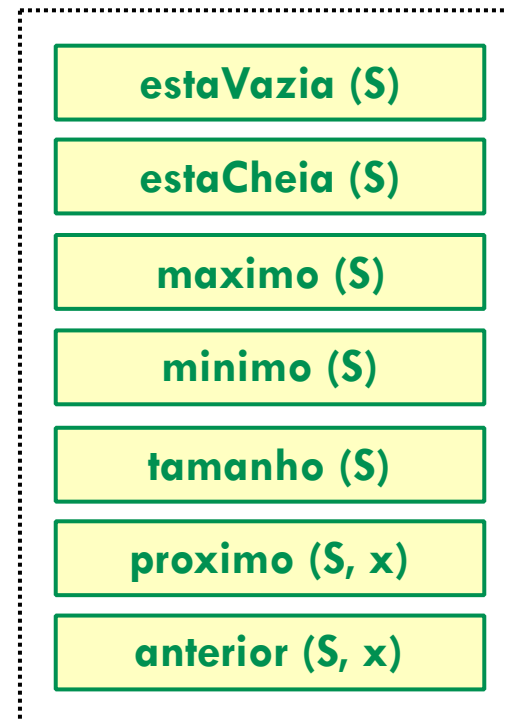
- 1 Listas Ordenadas
- 2 Operações gerais
- 3 Inserção de elementos
- 4 Pesquisa de elementos
- 5 Remoção de elementos
- 6 Referências

Operações em Listas Dinâmicas

Dada uma estrutura S , chave k , elemento x :



**Operações de
modificação**



**Operações adicionais
de consulta**

Operações em Listas Dinâmicas

Dada uma estrutura S , chave k , elemento x :



**Operações de
modificação**



**Operações adicionais
de consulta**

Lista dinâmica

- Single-linkage

tipo **Lista**

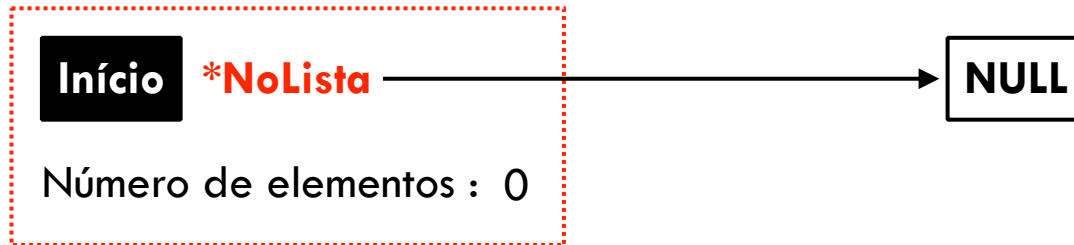
Início ***NoLista**

Número de elementos :

Inicialização

- Single-linkage

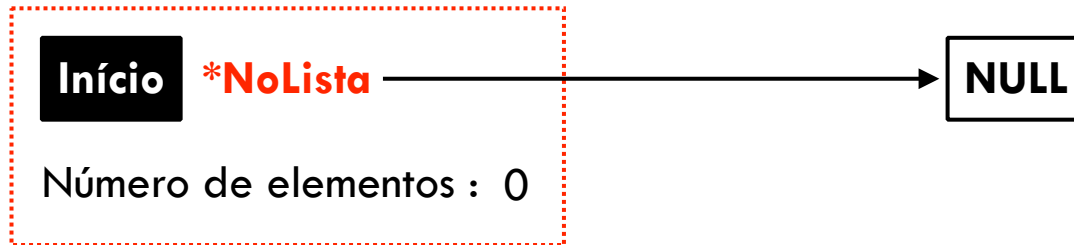
tipo **Lista**



Inicialização

- Single-linkage

tipo **Lista**

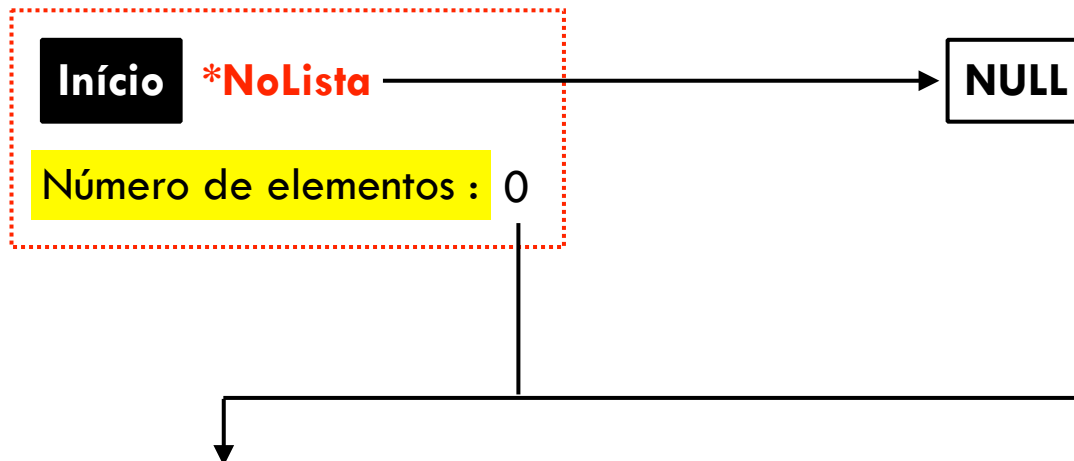


```
Inicializa (L)  
1. L.inicio = NULL;  
2. L.tamanho = 0;
```

Tamanho da Lista

- Single-linkage

tipo **Lista**



```
tamanhoLista (L)
1. return (L.tamanho);
```

```
estaVazia (L)
1. return (L.tamanho == 0);
// return(L.Inicio == NULL)
```

Exercício 01

- Mãos a obra: implemente um TDA para Lista com alocação dinâmica, e as funções de manipulação.
- Quais TDAs serão necessários?
- Implemente, em C, as funções para inicializar, verificar tamanho, se está vazia, e imprimir a lista.

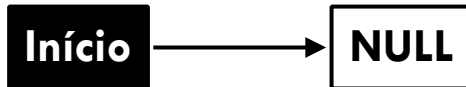
Roteiro

- 1 Listas Ordenadas
- 2 Operações gerais
- 3 Inserção de elementos
- 4 Pesquisa de elementos
- 5 Remoção de elementos
- 6 Referências

Inserção (Insert)

a) primeira inserção (elemento $x = 5$)

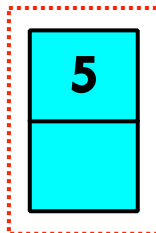
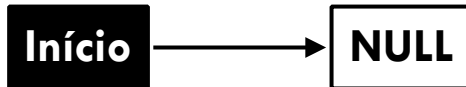
Número de elementos : 0



Inserção (Insert)

a) primeira inserção (elemento $x = 5$)

Número de elementos : 0

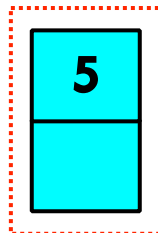
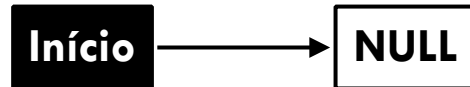


NoLista
(Novo)

Inserção (Insert)

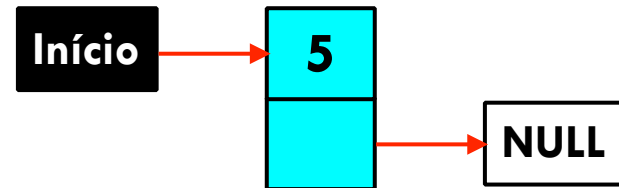
a) primeira inserção (elemento $x = 5$)

Número de elementos : 0



Novo
(Novo)

Número de elementos : 0



Novo

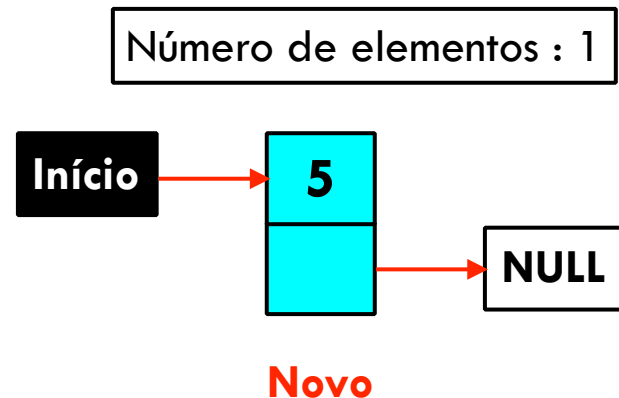
Inserção (Insert)

a) primeira inserção (elemento $x = 5$)

01

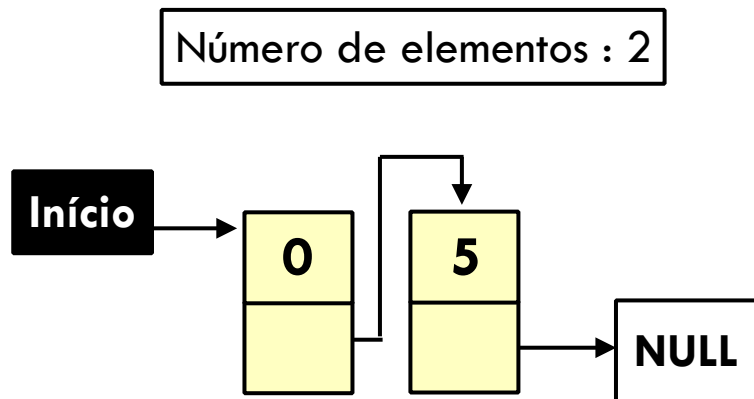
1. Criar ponteiro **Novo** e alocar memória para o nó
2. **Início** aponta para **Novo** (novo nó)
3. **Novo** aponta para NULL
4. Contador é incrementado

NoLista
(Novo)



Inserção (Insert)

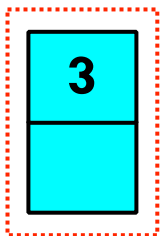
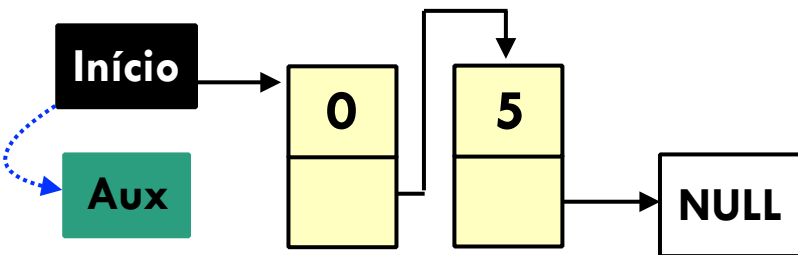
b) não é primeira inserção (elemento $x = 3$)



Inserção (Insert)

b) não é primeira inserção (elemento $x = 3$)

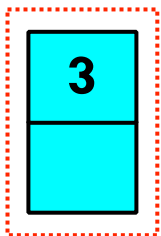
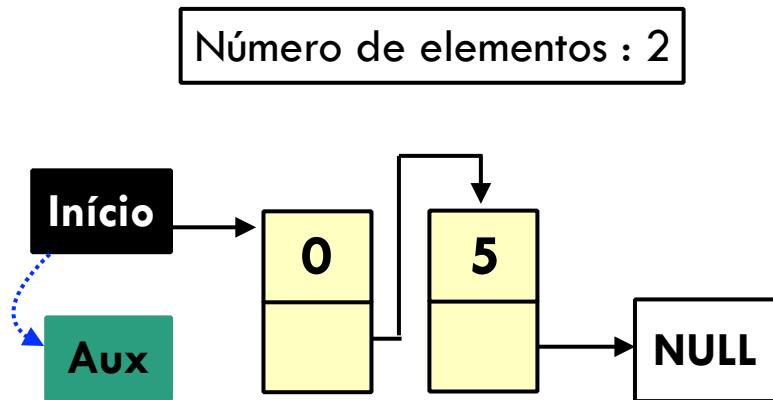
Número de elementos : 2



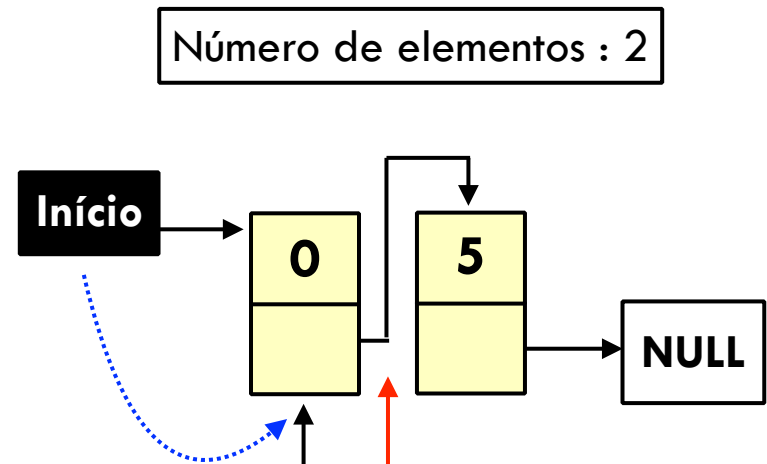
**NoLista
(Novo)**

Inserção (Insert)

b) não é primeira inserção (elemento $x = 3$)



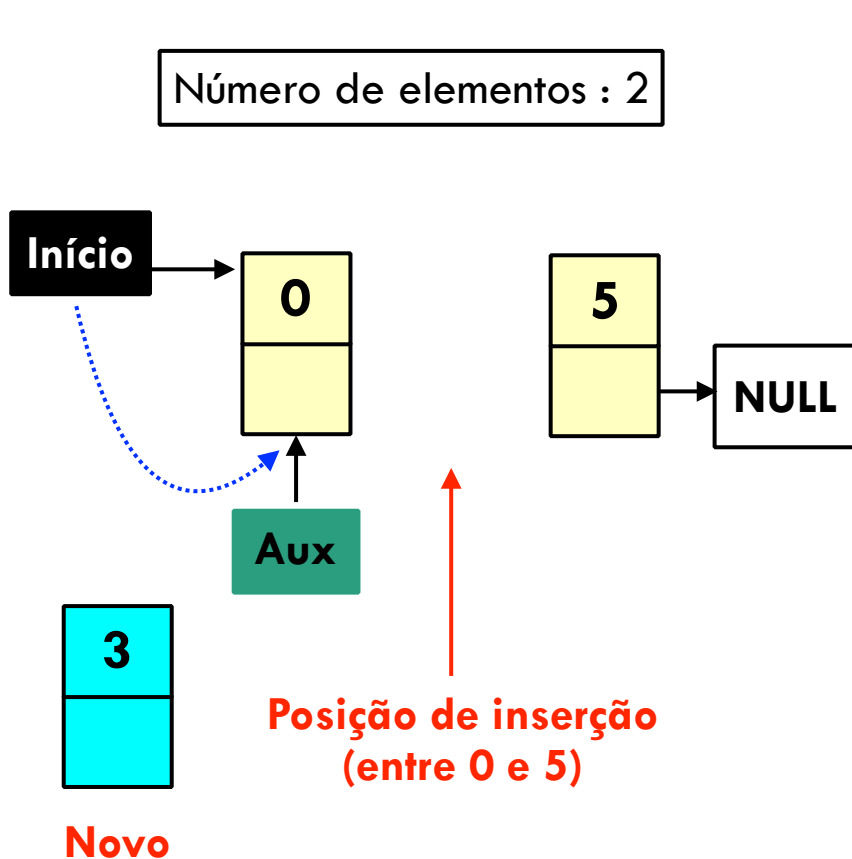
**NoLista
(Novo)**



**Posição de inserção
(entre 0 e 5)**

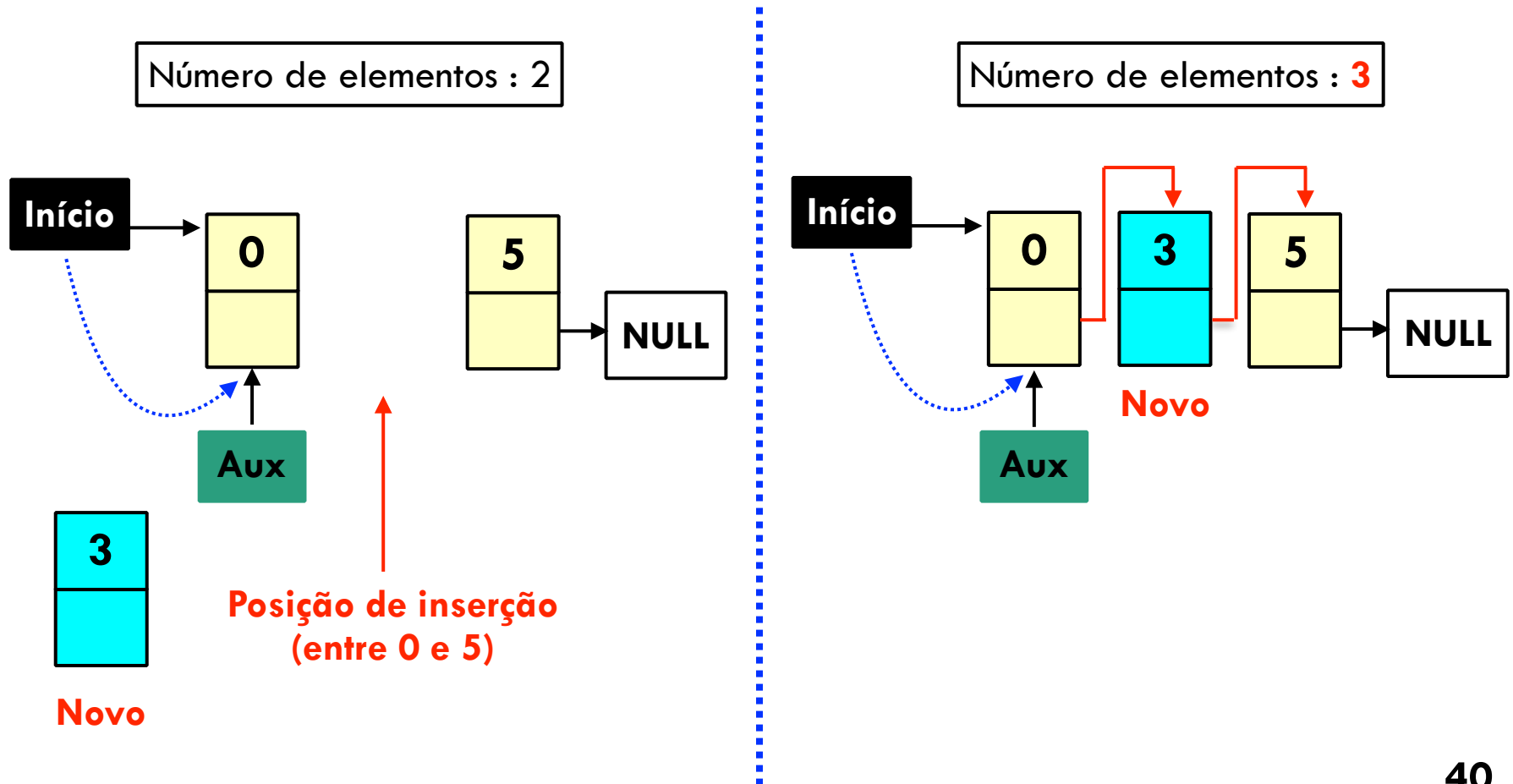
Inserção (Insert)

b) não é primeira inserção (elemento $x = 3$)



Inserção (Insert)

b) não é primeira inserção (elemento $x = 3$)



Inserção (Insert)

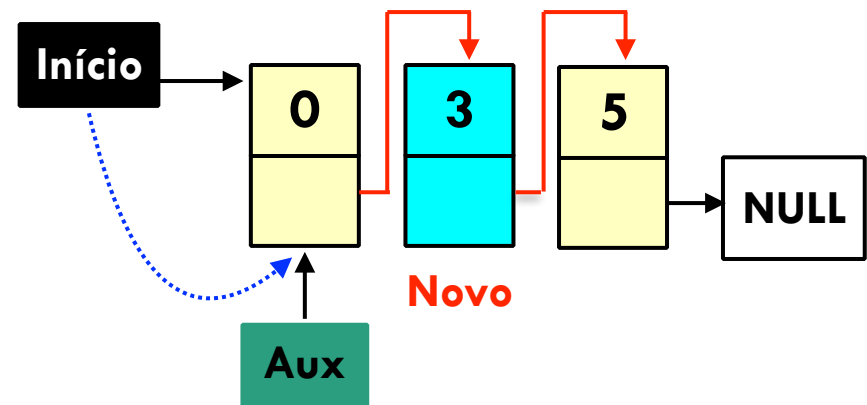
b) não é primeira inserção (elemento $x = 3$)

02

Número de elementos : 2

1. Percorrer a lista, usando **Aux** (Ponteiro)
Enquanto (**Aux** → proximo != NULL && $x > \text{Aux} \rightarrow \text{proximo.chave}$)
Aux = **Aux** → Proximo
2. Proximo do **Novo** recebe Proximo de **Aux**
3. Proximo de **Aux** recebe **Novo**
4. Contador é incrementado

Número de elementos : 3



Inserção (Insert)

Insert (L, x)

1. Criar novo nó **Novo** *//malloc*
2. **Novo**.chave = x
3. Se for a primeira inserção ou $x < \text{Inicio.chave}$:
4. **Novo**->proximo = L->primeiro *// Novo->proximo = NULL*
5. L->primeiro = **Novo**
6. Senão:
7. Criar ponteiro **Aux** = L->primeiro
8. *// percorrendo a lista ordenada*
9. Enquanto (**Aux**->proximo != NULL & $x > \text{Aux->proximo.chave}$)
10. **Aux** = **Aux**->proximo
11. **Novo**->proximo = **Aux**->proximo
12. **Aux**->proximo = **Novo**
13. incrementa contador de elementos

Inserção (Insert)

Insert (L, x)

1. Criar novo nó **Novo** *//malloc*
2. **Novo**.chave = x
3. Se for a primeira inserção ou $x < \text{Inicio.chave}$:
4. **Novo**->proximo = L->primeiro *// Novo->proximo = NULL*
5. L->primeiro = **Novo**
6. Senão:
7. Criar ponteiro **Aux** = L->primeiro
8. *// percorrendo a lista ordenada*
9. Enquanto (**Aux**->proximo != NULL & $x > \text{Aux->proximo.chave}$)
10. **Aux** = **Aux**->proximo
11. **Novo**->proximo = **Aux**->proximo
12. **Aux**->proximo = **Novo**
13. incrementa contador de elementos

Obs: precisaremos de dois ponteiros do tipo NoLista

- um para o novo elemento (**Novo**)
- um para percorrer a lista (**Aux**)

Exercício 02



- Implementar a função de inserção de uma lista ordenada

Roteiro

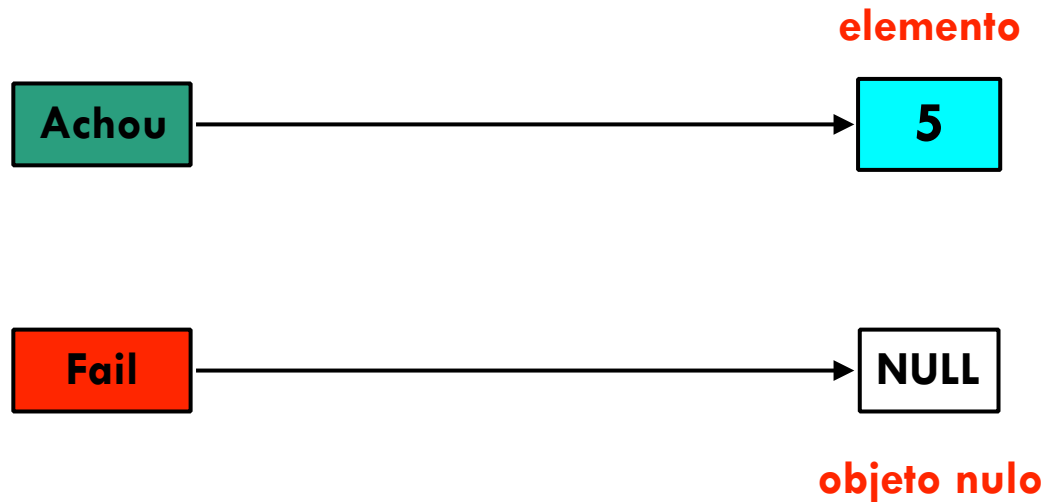
- 1 Listas Ordenadas
- 2 Operações gerais
- 3 Inserção de elementos
- 4 Pesquisa de elementos
- 5 Remoção de elementos
- 6 Referências

Pesquisa (Search)

- Procura a primeira ocorrência de um elemento
 - se achar: ?
 - se não achar?

Pesquisa (Search)

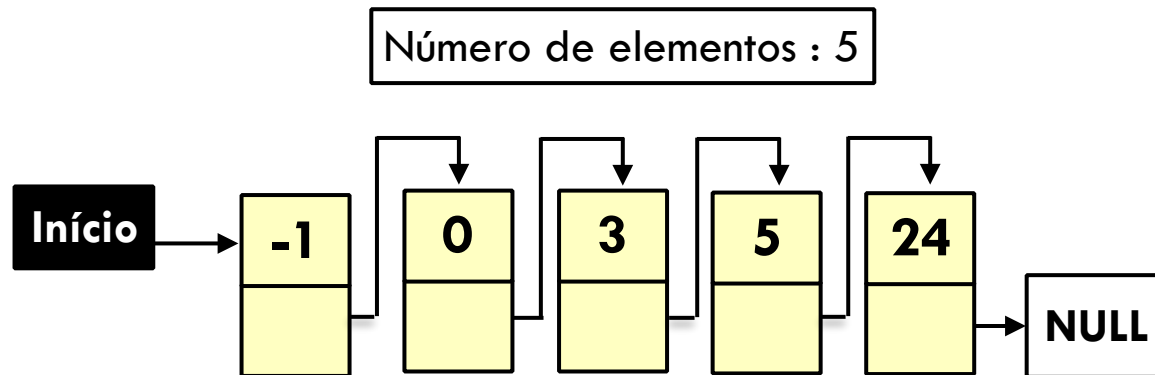
- Procura a primeira ocorrência de um elemento
 - se achar: ?
 - se não achar?



Pesquisa (Search)

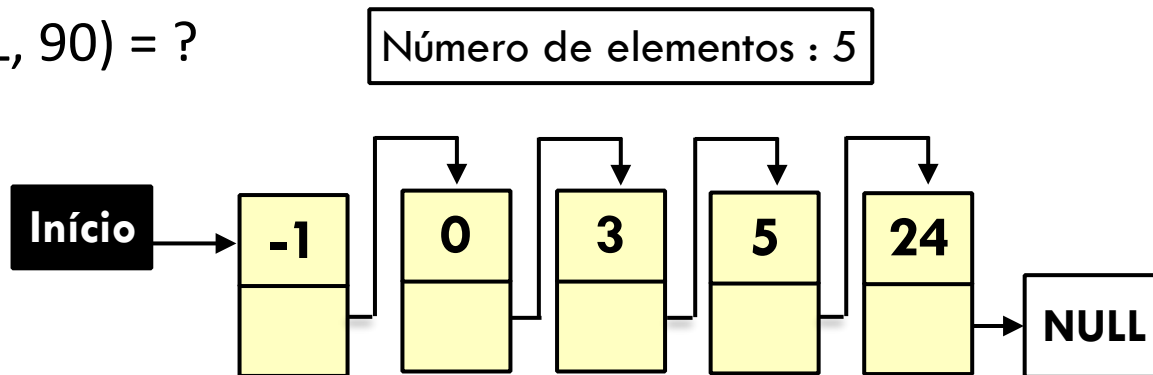
- Pergunta: como implementar se eu quiser fazer a função search do tipo **bool**?

Pesquisa (Search)



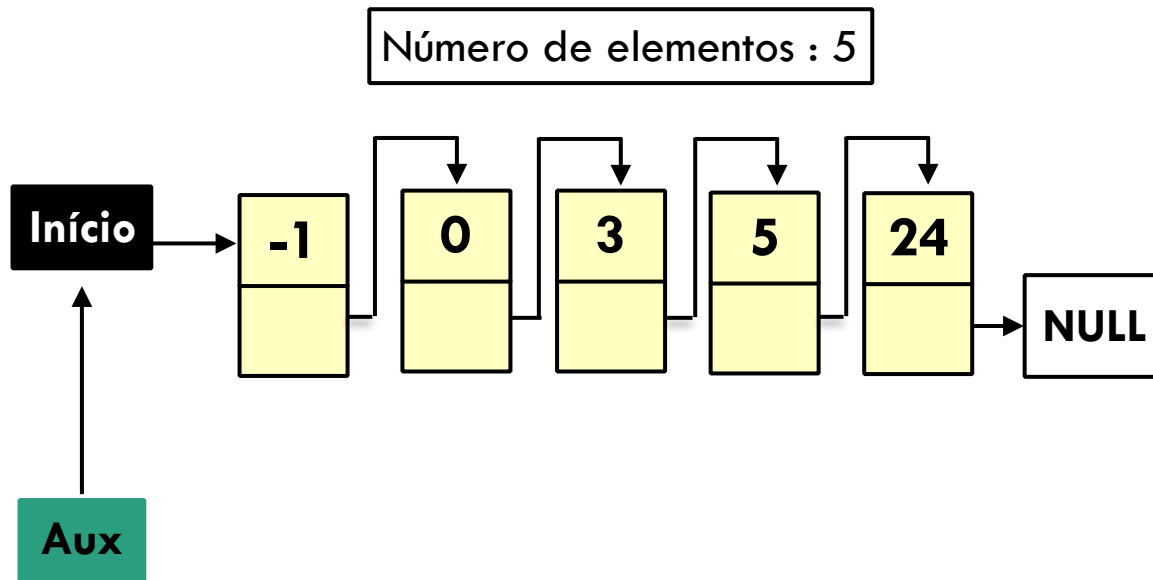
Pesquisa (Search)

- Search(L, 5) = ?
- Search(L, 4) = ?
- Search(L, -2) = ?
- Search(L, 90) = ?



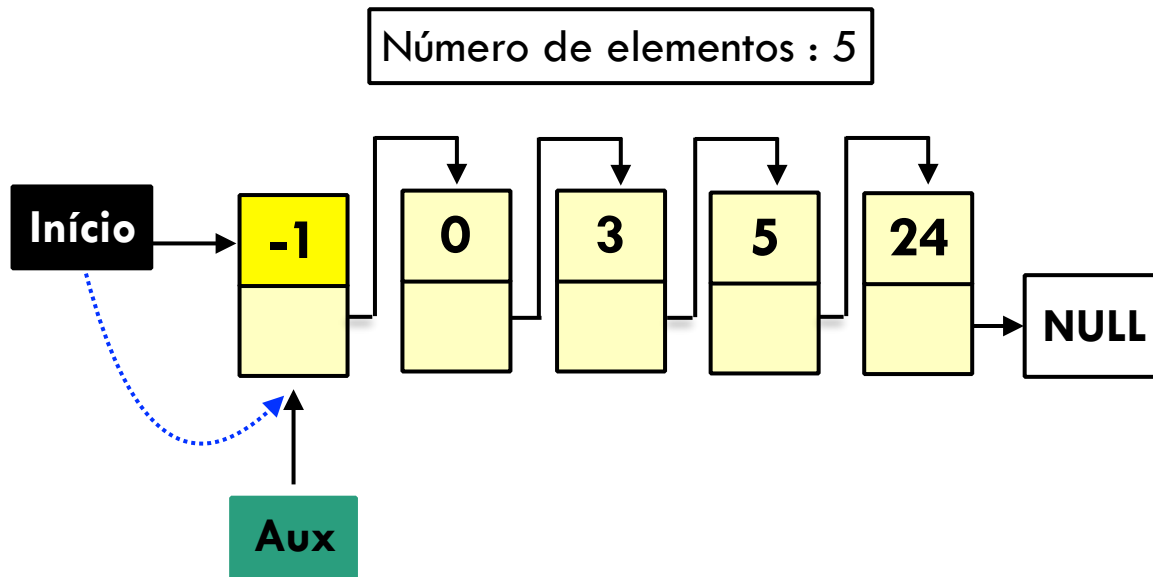
Pesquisa (Search)

□ Search(L, 5) = ?



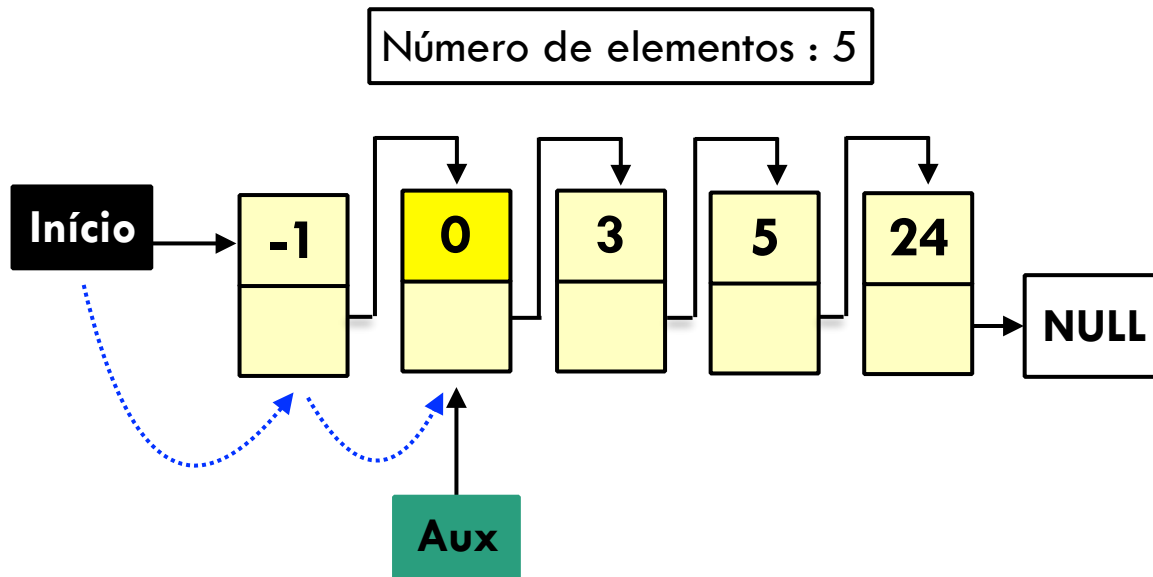
Pesquisa (Search)

□ Search(L, 5) = ?



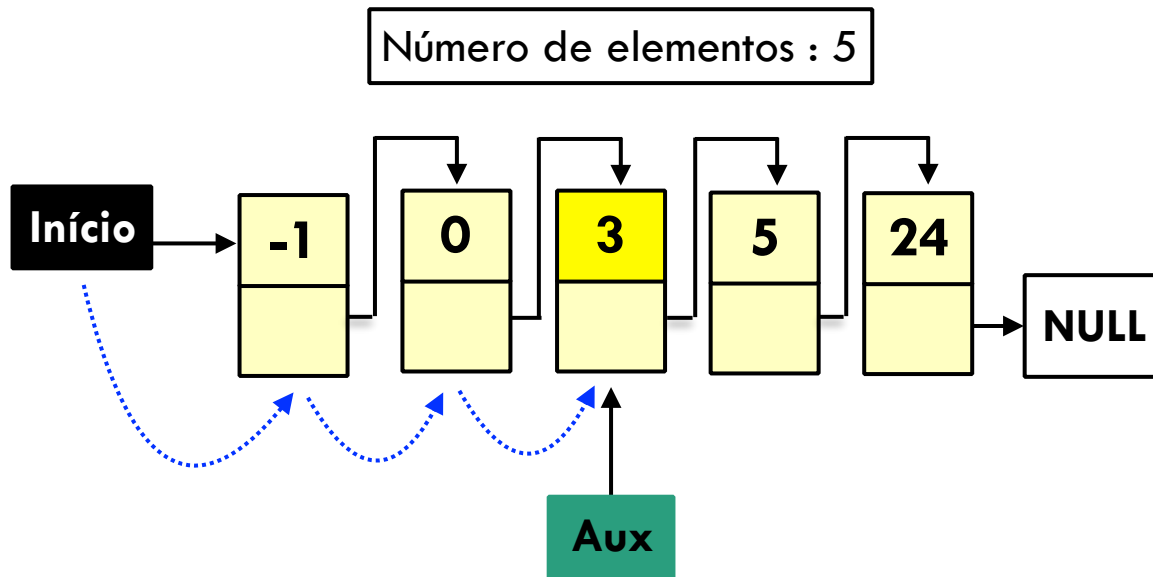
Pesquisa (Search)

□ Search(L, 5) = ?



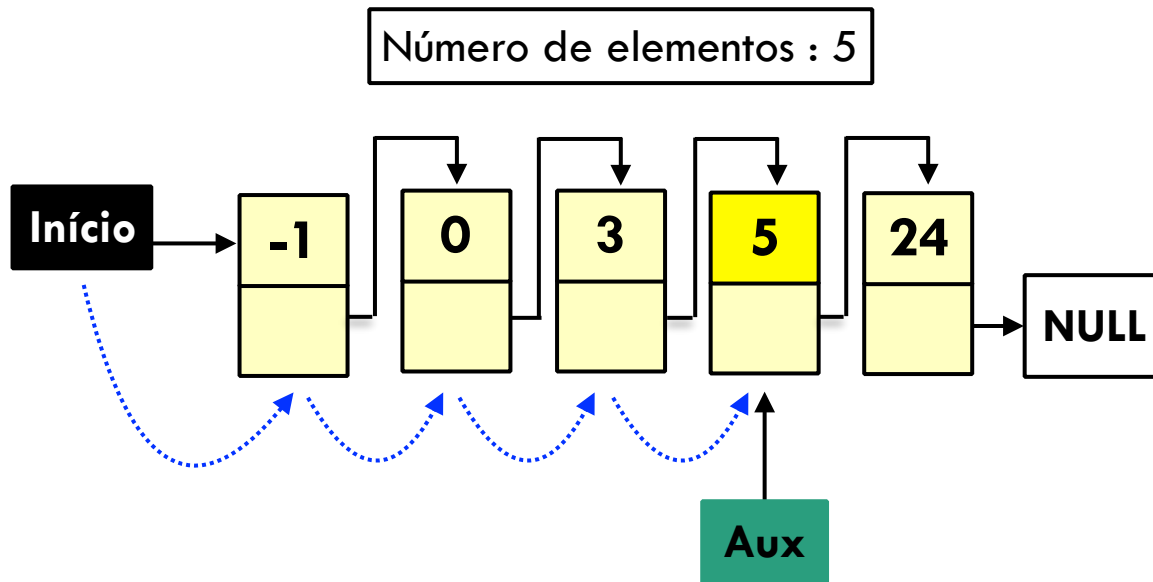
Pesquisa (Search)

□ Search(L, 5) = ?



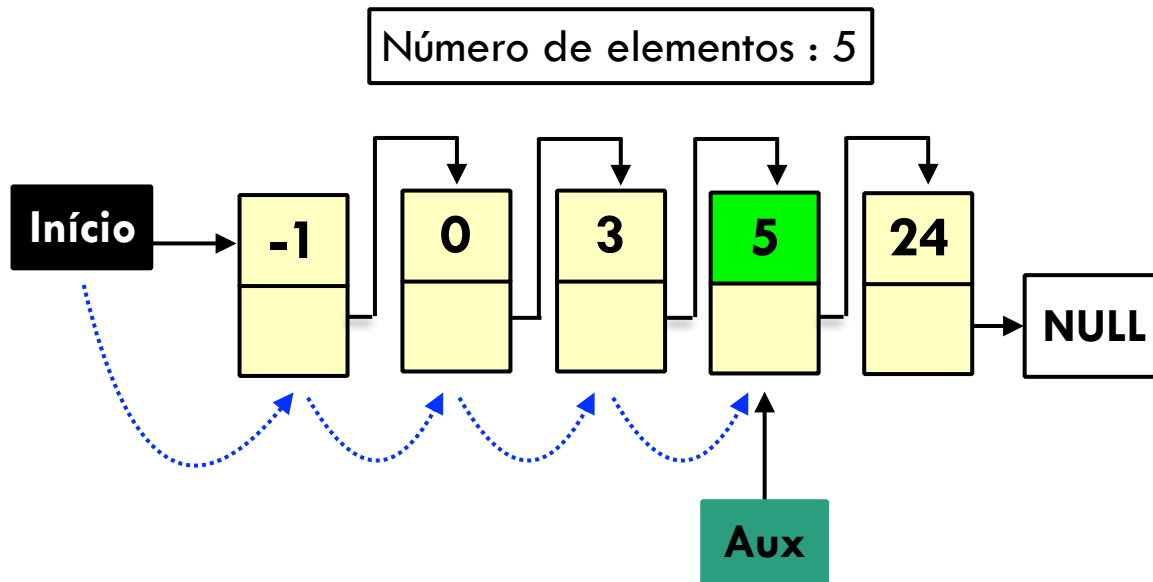
Pesquisa (Search)

□ Search(L, 5) = ?



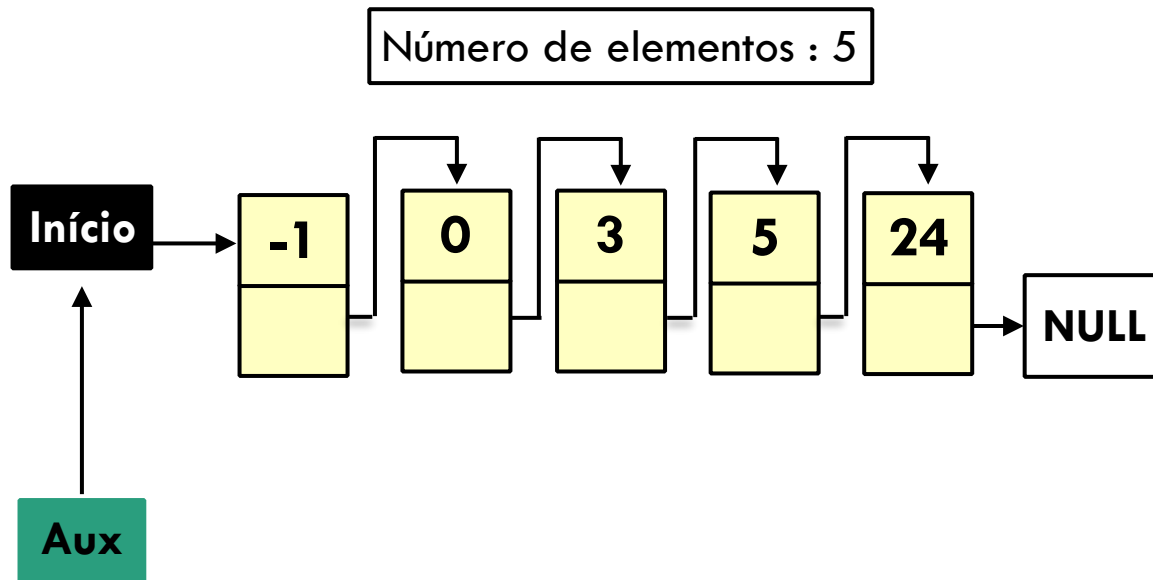
Pesquisa (Search)

- Search(L, 5) = **Sucesso** :)



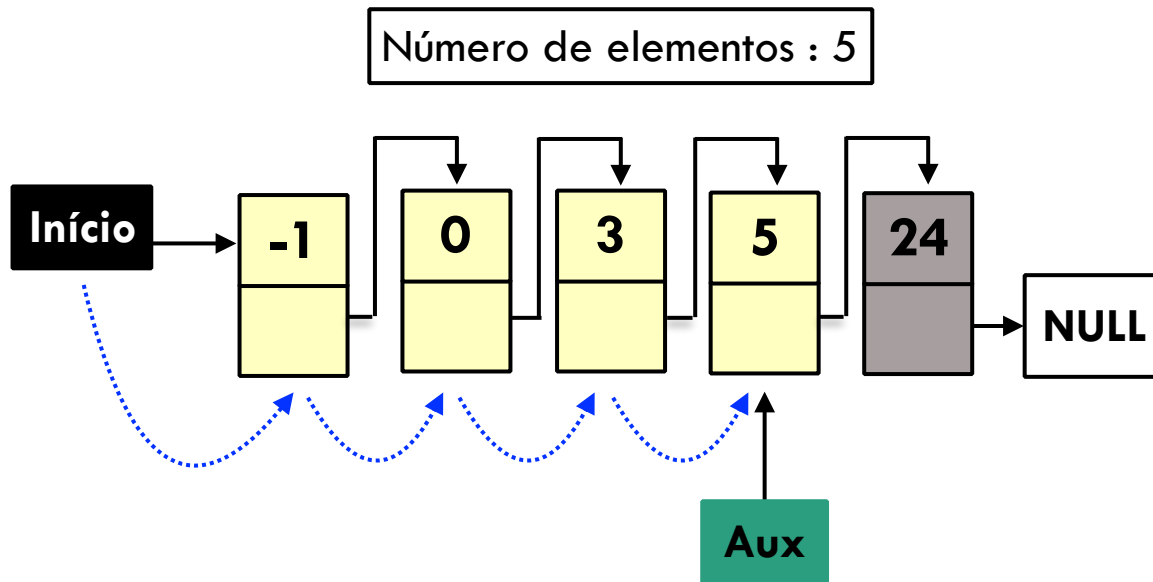
Pesquisa (Search)

□ Search(L, 4) = ?



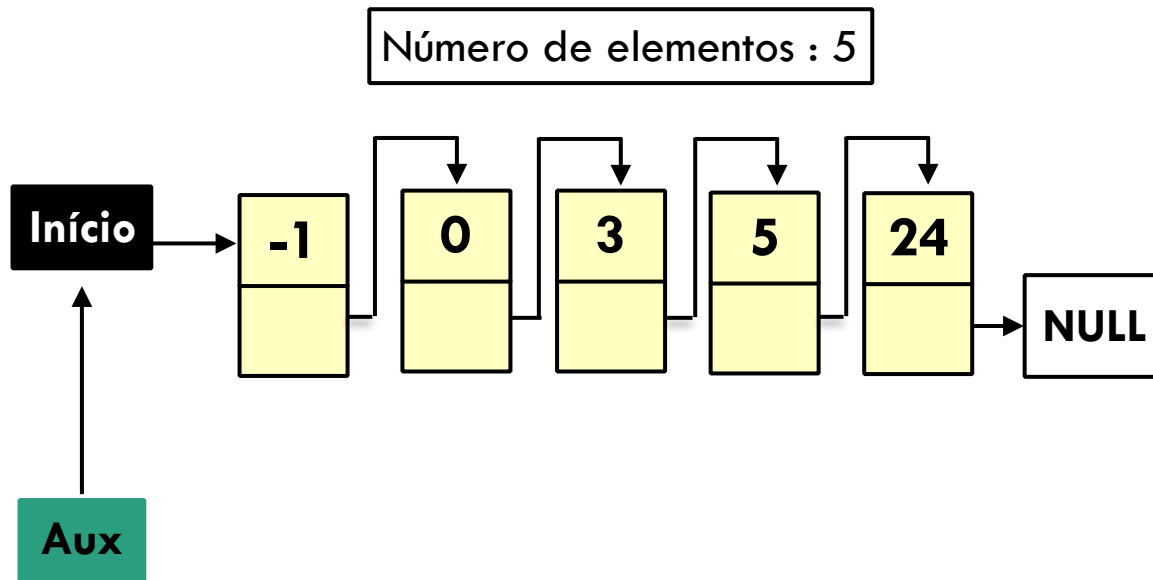
Pesquisa (Search)

□ Search(L, 4) = **Fail !**



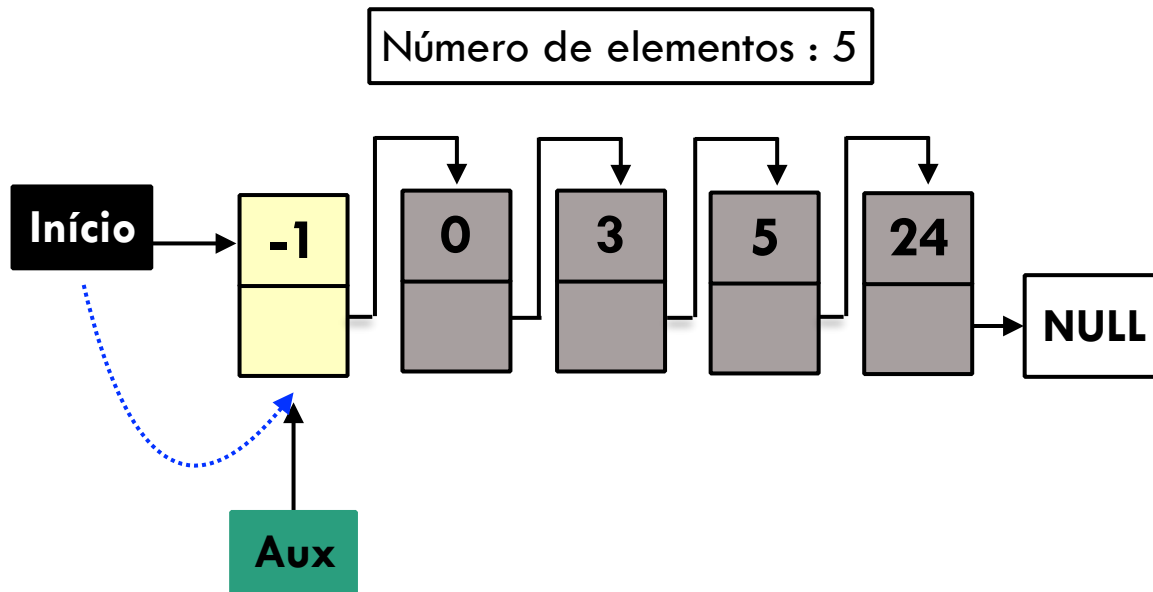
Pesquisa (Search)

□ Search(L, 2) = ?



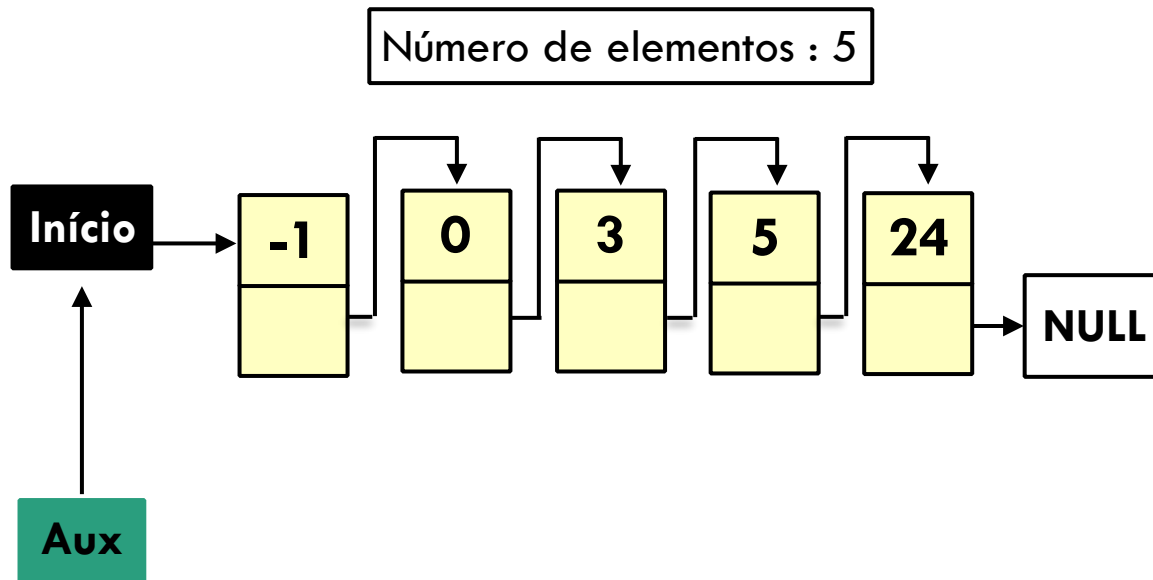
Pesquisa (Search)

- Search(L,-2) = **Fail !**



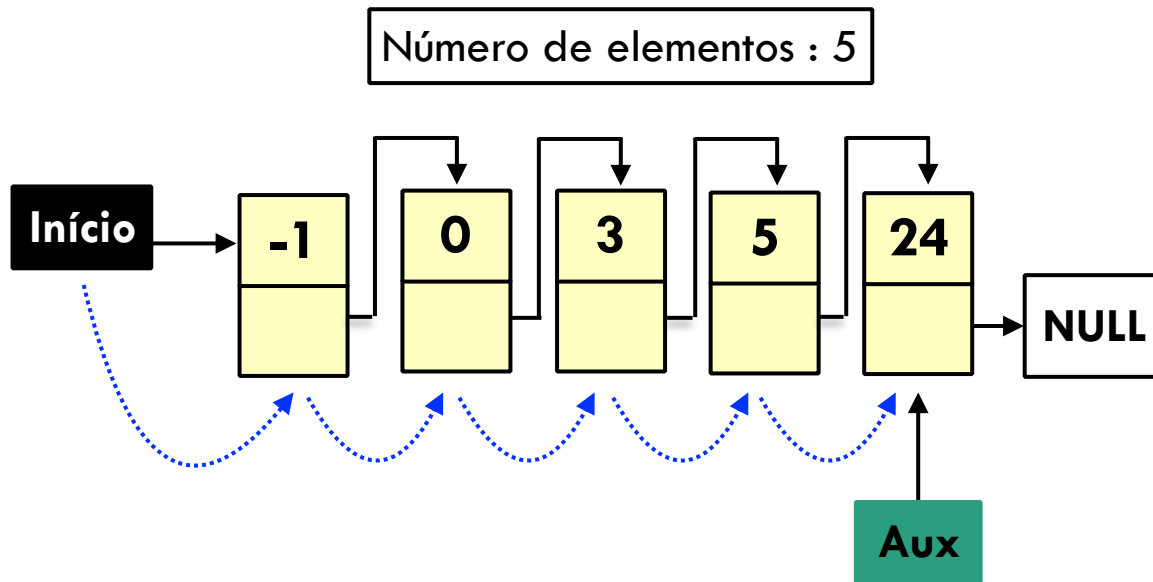
Pesquisa (Search)

□ Search(L, 90) = ?



Pesquisa (Search)

□ Search(L, 90) = **Fail !**



Pesquisa (Search)



- Podemos fazer de várias formas

Pesquisa (Search)

- Podemos fazer de várias formas

Pesquisa(L, x)

```
1. criar ponteiro Aux
2. Repetir (Aux = L->primeiro; Aux != NULL; Aux = Aux->proximo)
3.     se Aux->x == x
4.         return True;
5. return False;
```


Pesquisa (Search)

- Podemos fazer de várias formas

Pesquisa(L, x)

```
1. criar ponteiro Aux
2. Repetir (Aux = L->primeiro; Aux != NULL; Aux = Aux->proximo)
3.     se Aux->x == x
4.         return True;
5. return False;
```

Boa implementação?

Pesquisa (Search)

- Podemos fazer de várias formas

Pesquisa(L, x)

```
1. criar ponteiro Aux
2. Repetir (Aux = L->primeiro; Aux != NULL; Aux = Aux->proximo)
3.     se Aux->x == x
4.         return True;
5. return False;
```

Boa implementação? Não, pois percorre todos os elementos no pior caso.

Pesquisa (Search)

- Podemos fazer de várias formas

Pesquisa (L, x)

```
1. Se a Lista esta vazia
2.     return False;
3. criar ponteiro Aux = L->primeiro
4. Enquanto (Aux != NULL && x > Aux->x)
5.     Aux = Aux->next
6. Se Aux == NULL || Aux->x > x    // não existe elemento
7.     return False;
8. return True;
```

Melhor! Evita comparações desnecessárias.

Exercício 03



- Implementar a função de pesquisa de uma lista ordenada

Roteiro

- 1 Listas Ordenadas
- 2 Operações gerais
- 3 Inserção de elementos
- 4 Pesquisa de elementos
- 5 Remoção de elementos
- 6 Referências

Remover (remove)



- 5 casos diferentes para se checar

Remover (remove)

- 5 casos diferentes para se checar

A Lista vazia

Remover (remove)

- 5 casos diferentes para se checar

A Lista vazia

B elemento a ser removido é menor que o primeiro da lista

Remover (remove)

- 5 casos diferentes para se checar

A Lista vazia

B elemento a ser removido é menor que o primeiro da lista

C elemento a ser removido é o primeiro

Remove (remove)

- 5 casos diferentes para se checar

- A** Lista vazia
- B** elemento a ser removido é menor que o primeiro da lista
- C** elemento a ser removido é o primeiro
- D** elemento a ser removido não é o primeiro (percorrer a lista)

Remove (remove)

- 5 casos diferentes para se checar
 - A** Lista vazia
 - B** elemento a ser removido é menor que o primeiro da lista
 - C** elemento a ser removido é o primeiro
 - D** elemento a ser removido não é o primeiro (percorrer a lista)
 - D1** elemento não está na lista depois de percorrer

Remove (remove)

- 5 casos diferentes para se checar
 - A** Lista vazia
 - B** elemento a ser removido é menor que o primeiro da lista
 - C** elemento a ser removido é o primeiro
 - D** elemento a ser removido não é o primeiro (percorrer a lista)
 - D1** elemento não está na lista depois de percorrer
 - D2** elemento está na lista depois de percorrer

Remove (remove)

- 5 casos diferentes para se checar
 - A** Lista vazia
 - B** elemento a ser removido é menor que o primeiro da lista
 - C** elemento a ser removido é o primeiro
 - D** elemento a ser removido não é o primeiro (percorrer a lista)
 - D1** elemento não está na lista depois de percorrer
 - D2** elemento está na lista depois de percorrer

Remoção (Remove)

Remove (L, x)

//casos 1 e 2

1. Se a lista está vazia OU x é menor do que o primeiro elemento:

2. return **False**; *// NULL*

3. Se x == primeiro elemento: *//caso 3*

4. remove o primeiro *// dequeue*

5. decrementa o contador

6. return elemento;

7. *// casos 4 e 5*

8. Percorrer a lista:

9. Se achar o elemento:

10. remove o elemento;

11. decrementa o contador

12. return elemento;

13. Senão, chegou até o último elemento e ele o valor não existe

14. return **False**; *//NULL*

Exercício 04



- Implementar a função de remoção de uma lista ordenada

Complexidade das operações

- Custo (O)
 - pesquisa/busca =
 - inserção (ordenada) =
 - remoção do ultimo =
 - remoção do primeiro =
 - remoção de k =

Complexidade das operações

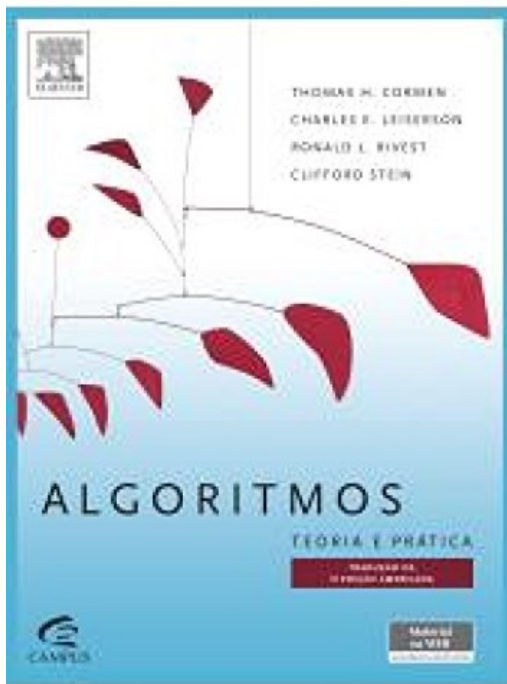
□ Custo (O)

- pesquisa/busca = $O(n)$ // percorrer lista
- inserção (ordenada) = $O(n)$ // percorrer lista
- remoção do ultimo = $O(n)$ // percorrer lista
- remoção do primeiro = $O(1)$ // como na fila
- remoção de k = $O(n)$ // percorrer lista

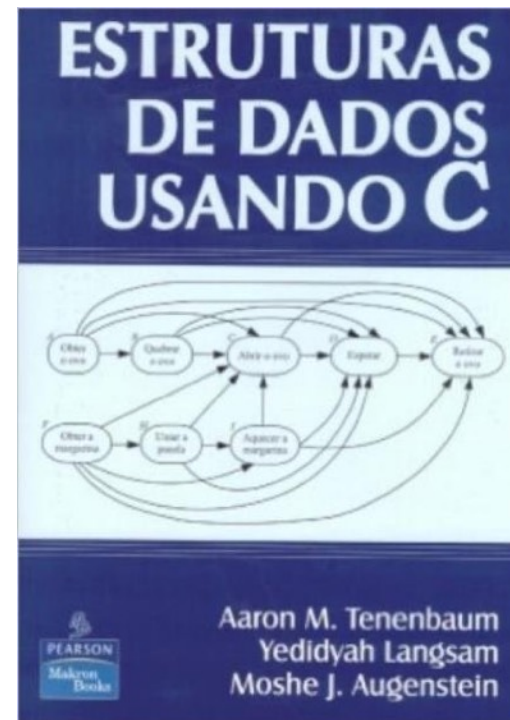
Roteiro

- 1 Introdução
- 2 Filas
- 3 Operações gerais
- 4 Inserção de elementos
- 5 Remoção de elementos
- 6 Referências

Referências sugeridas

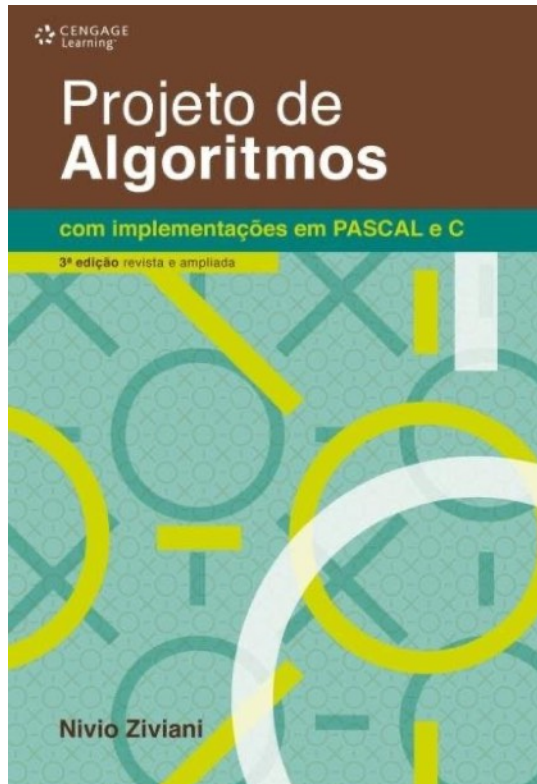


[Cormen et al, 2018]



[Tenenbaum et al, 1995]

Referências sugeridas



[Ziviani, 2010]



[Drozdek, 2017]

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br