# ED62A-COM2A
# ESTRUTURAS DE DADOS

## Aula 05 - Listas ordenadas
## (Estrutura dinâmica)

Prof. Rafael G. Mantovani

12/04/2019

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Apucarana - PR, Brasil
Abril, 2019

Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia de Computação

# Roteiro

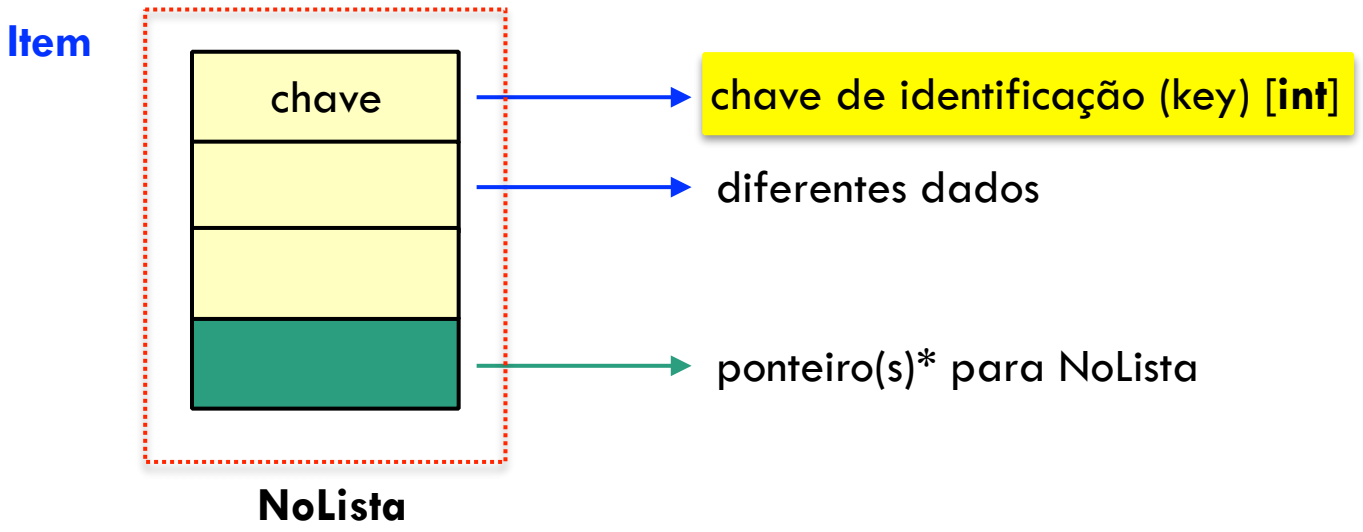
1. Listas Ordenadas
2. Operações gerais
3. Inserção de elementos
4. Pesquisa de elementos
5. Remoção de elementos
6. Referências

# Roteiro

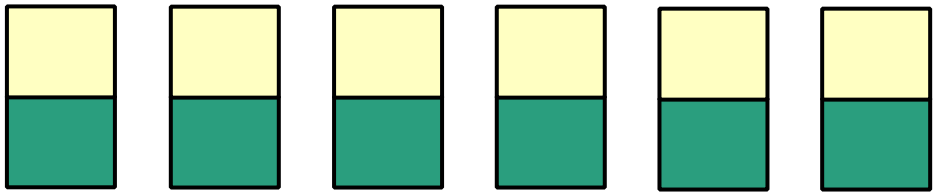
1. Listas Ordenadas
2. Operações gerais
3. Inserção de elementos
4. Pesquisa de elementos
5. Remoção de elementos
6. Referências

# Lista dinâmica

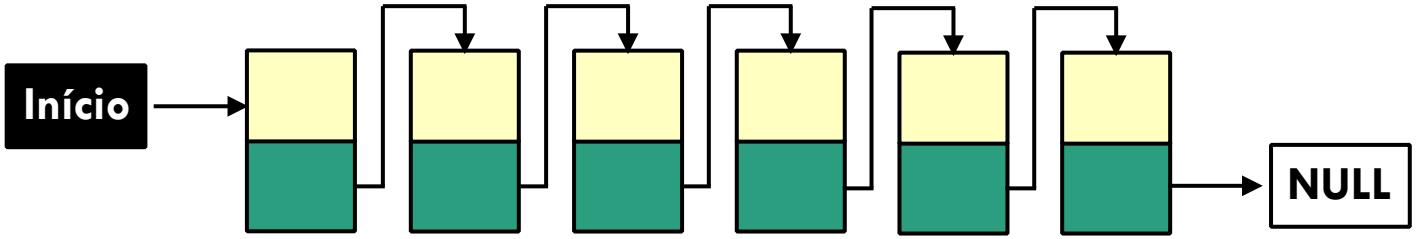
- Nós de Lista



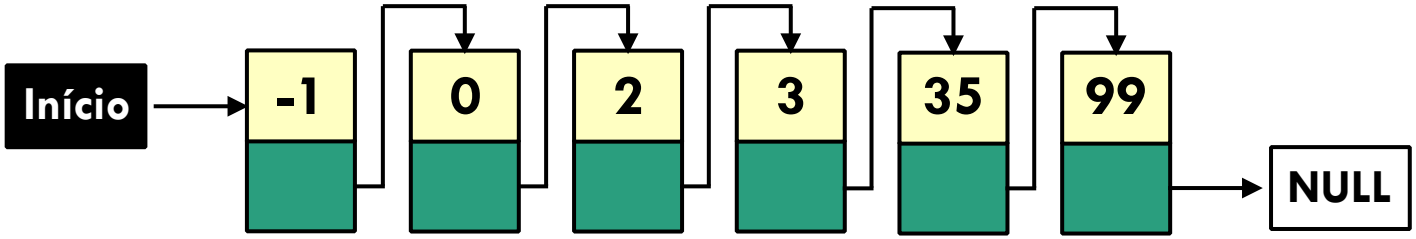
# Lista dinâmica



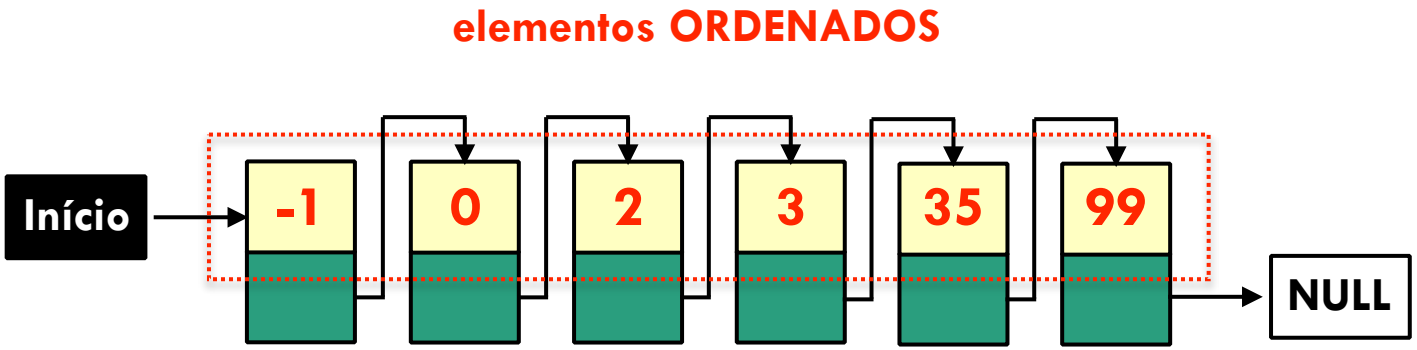
# Lista dinâmica



# 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

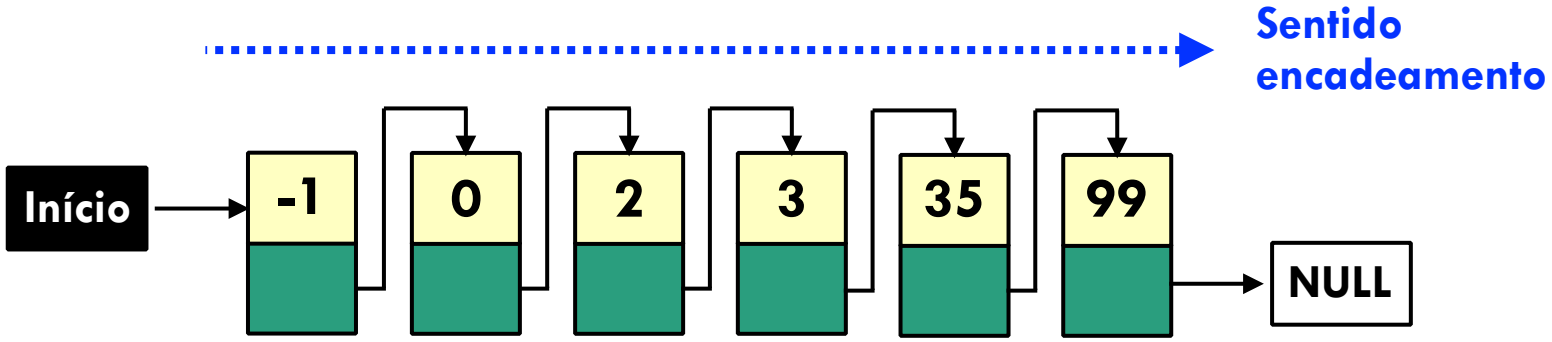
- Diferentes implementações de lista dinâmica:

A **Single-linkage: singularmente encadeada**

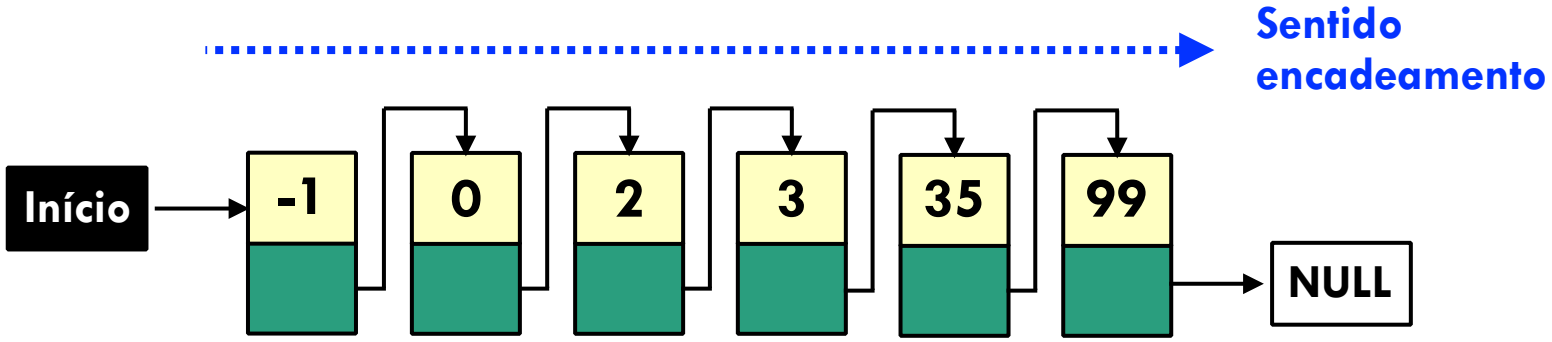
B **Double-linkage: duplamente encadeada**

C **Circulares: nó sentinela**

# Tipos de lista



# Tipos de lista




tipo **Lista**

**Início** *NoList

Número de elementos :

tipo **NoLista**

**Item** Item armazenado

**Prox** *NoList

# Tipos de lista

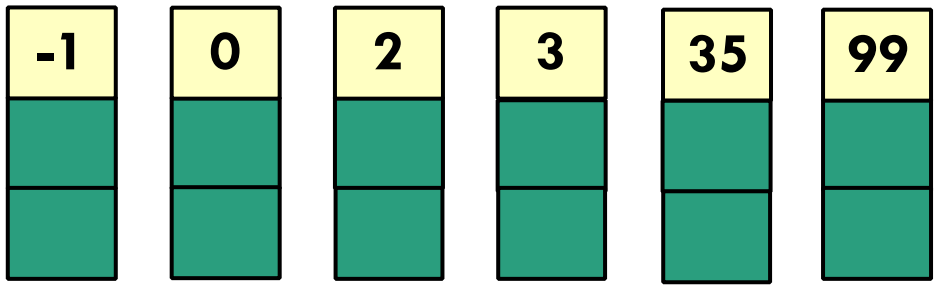
- Diferentes implementações de lista dinâmica:

A **Single-linkage: singularmente encadeada**

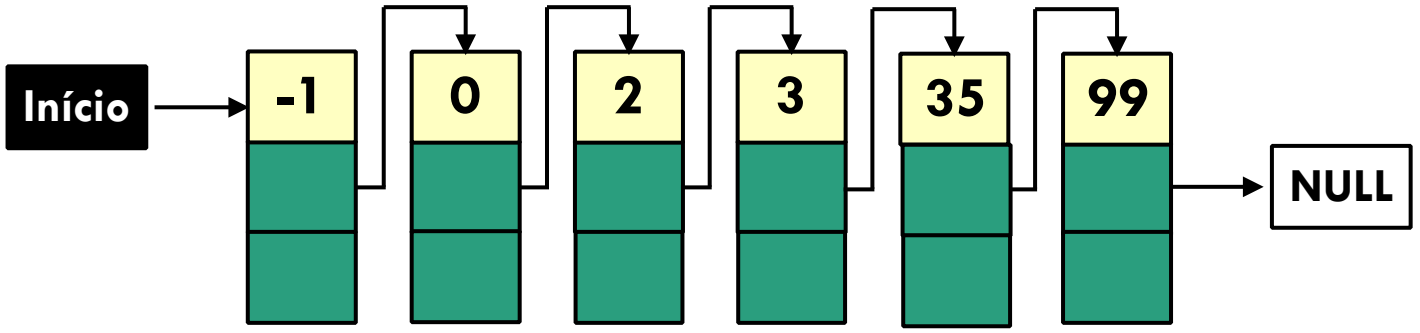
B **Double-linkage: duplamente encadeada**

C **Circulares: nó sentinela**

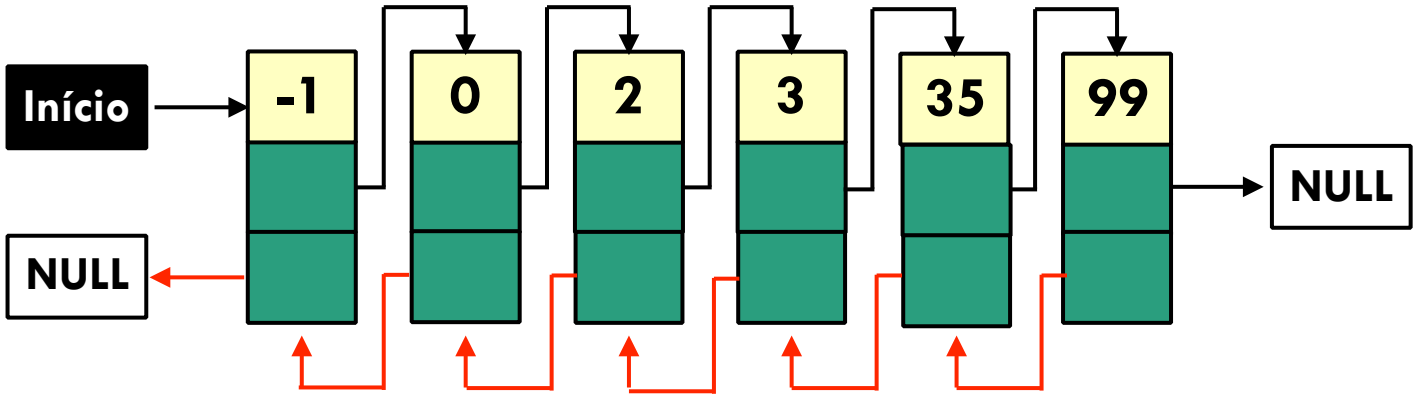
# Tipos de lista



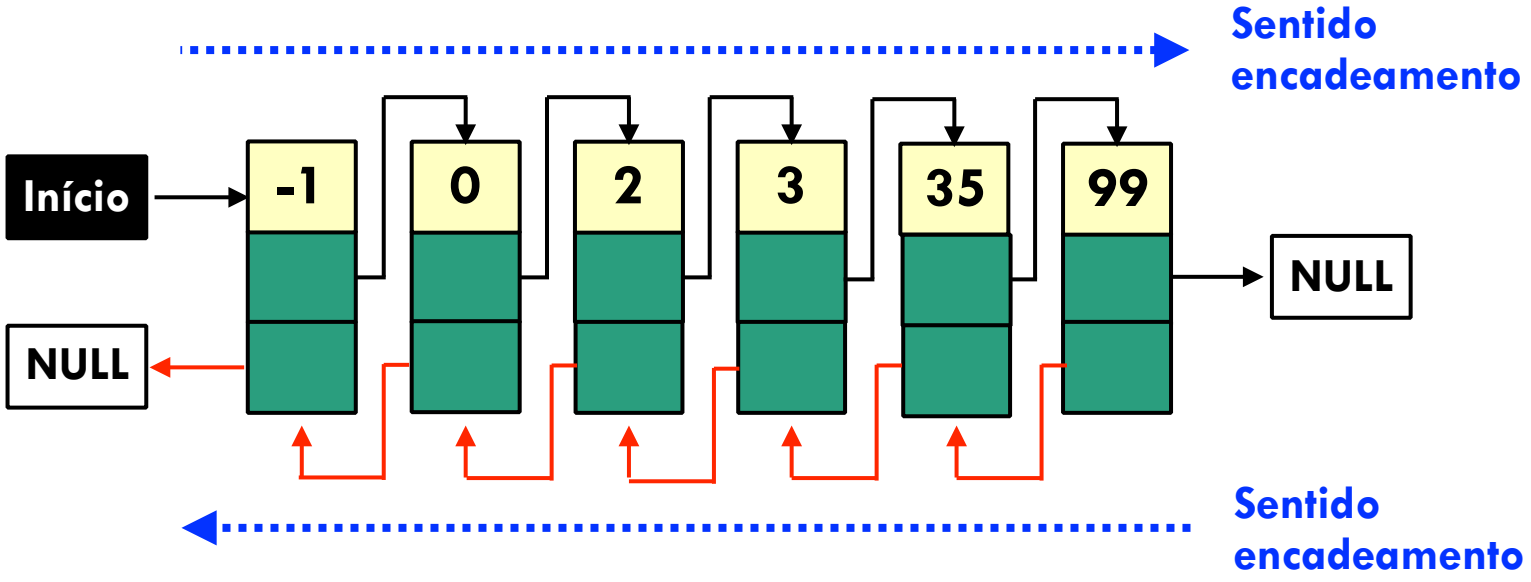
# Tipos de lista



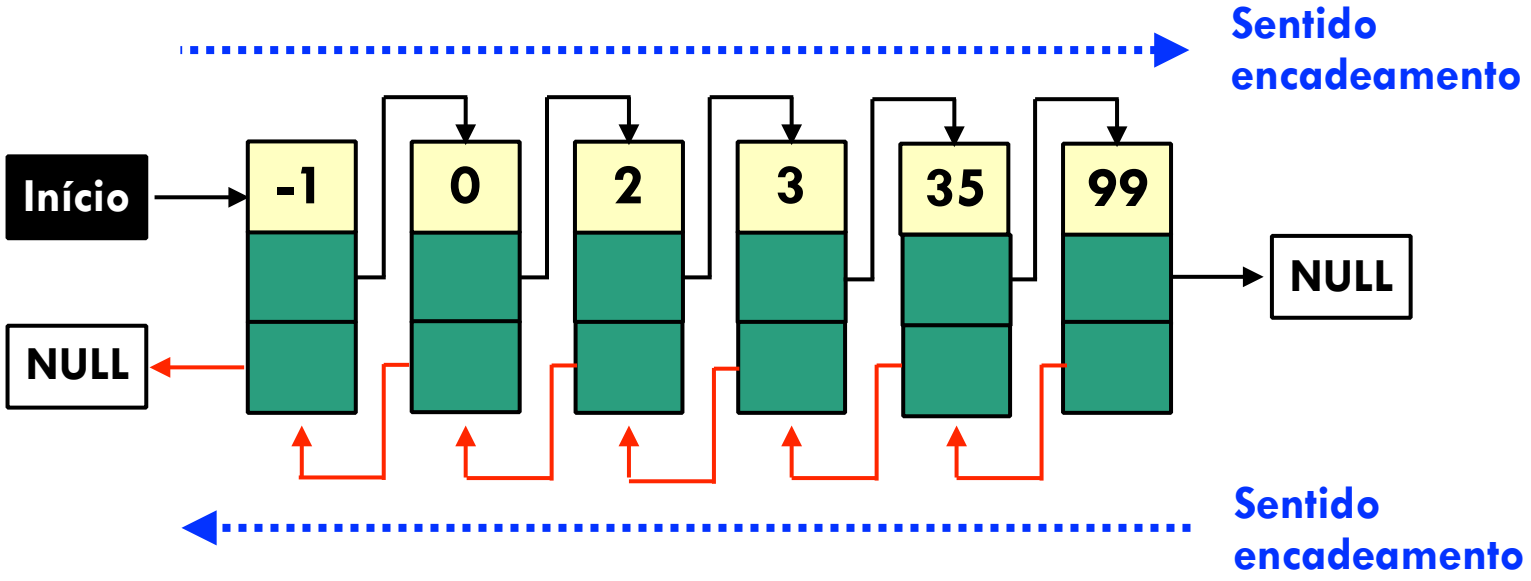
# Tipos de lista



# Tipos de lista



# Tipos de lista




tipo **Lista**

**Início** *NoList

Número de elementos :

tipo **NoLista**

**Item** Item armazenado

**Prox** *NoList (proximo)

**Ant** *NoList (anterior)

# Tipos de lista

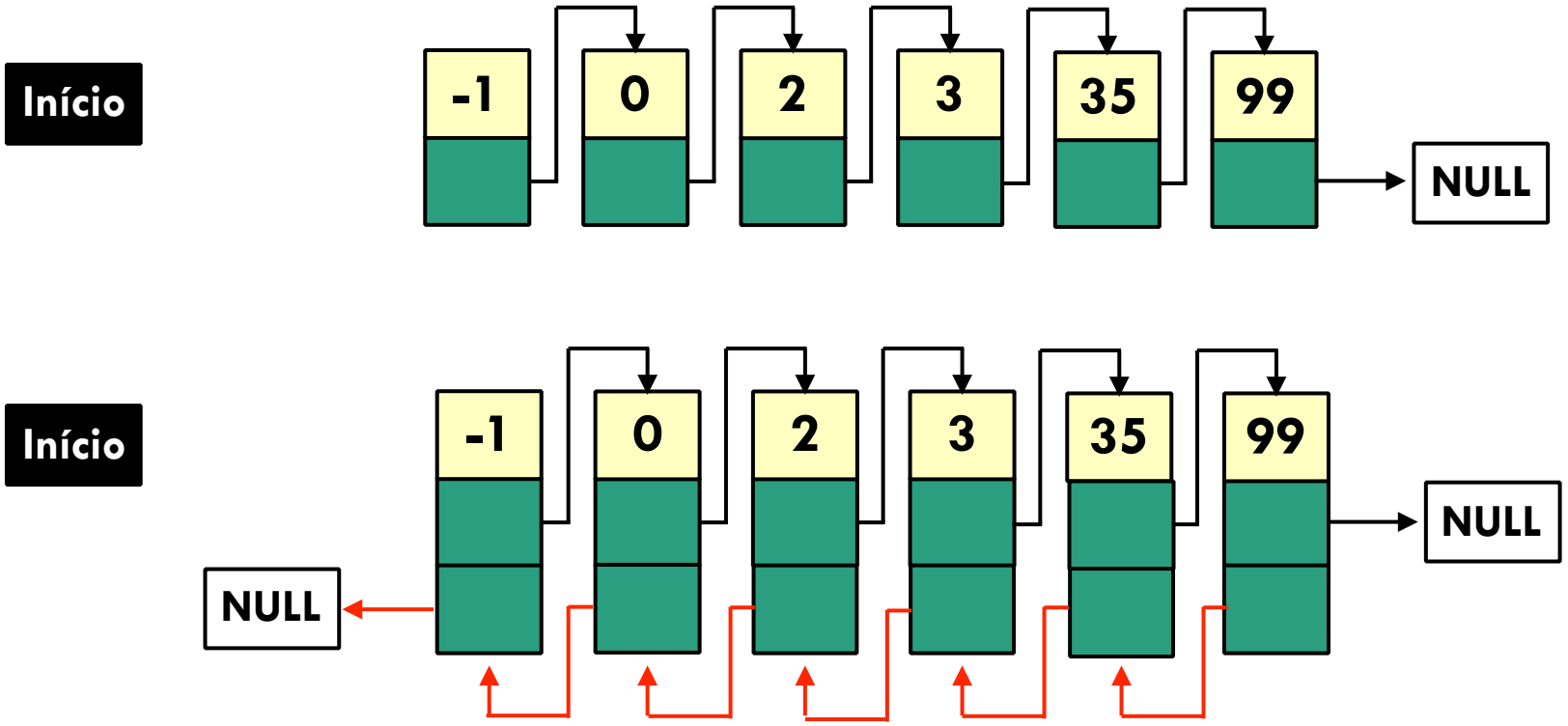
- Diferentes implementações de lista dinâmica:

A **Single-linkage: singularmente encadeada**

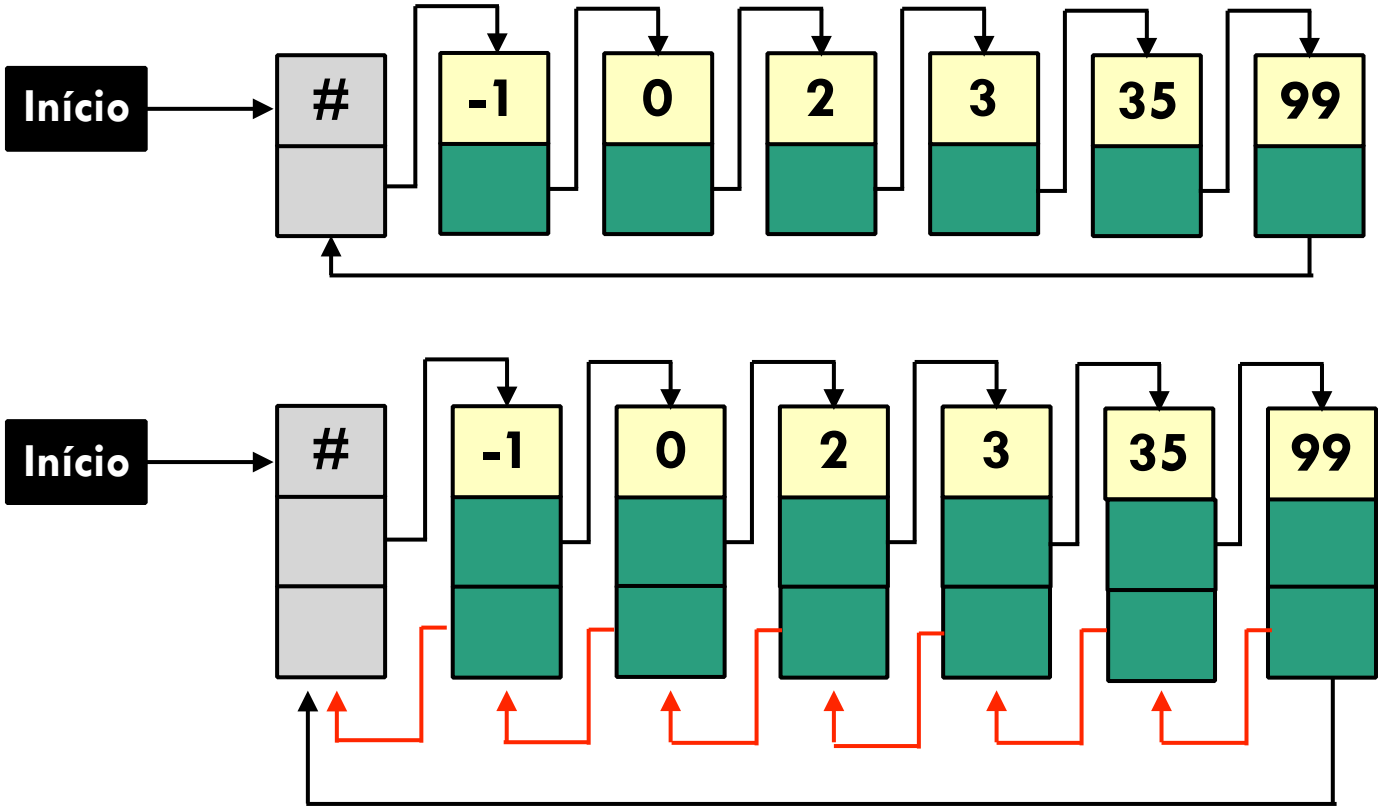
B **Double-linkage: duplamente encadeada**

C **Circulares: nó sentinela**

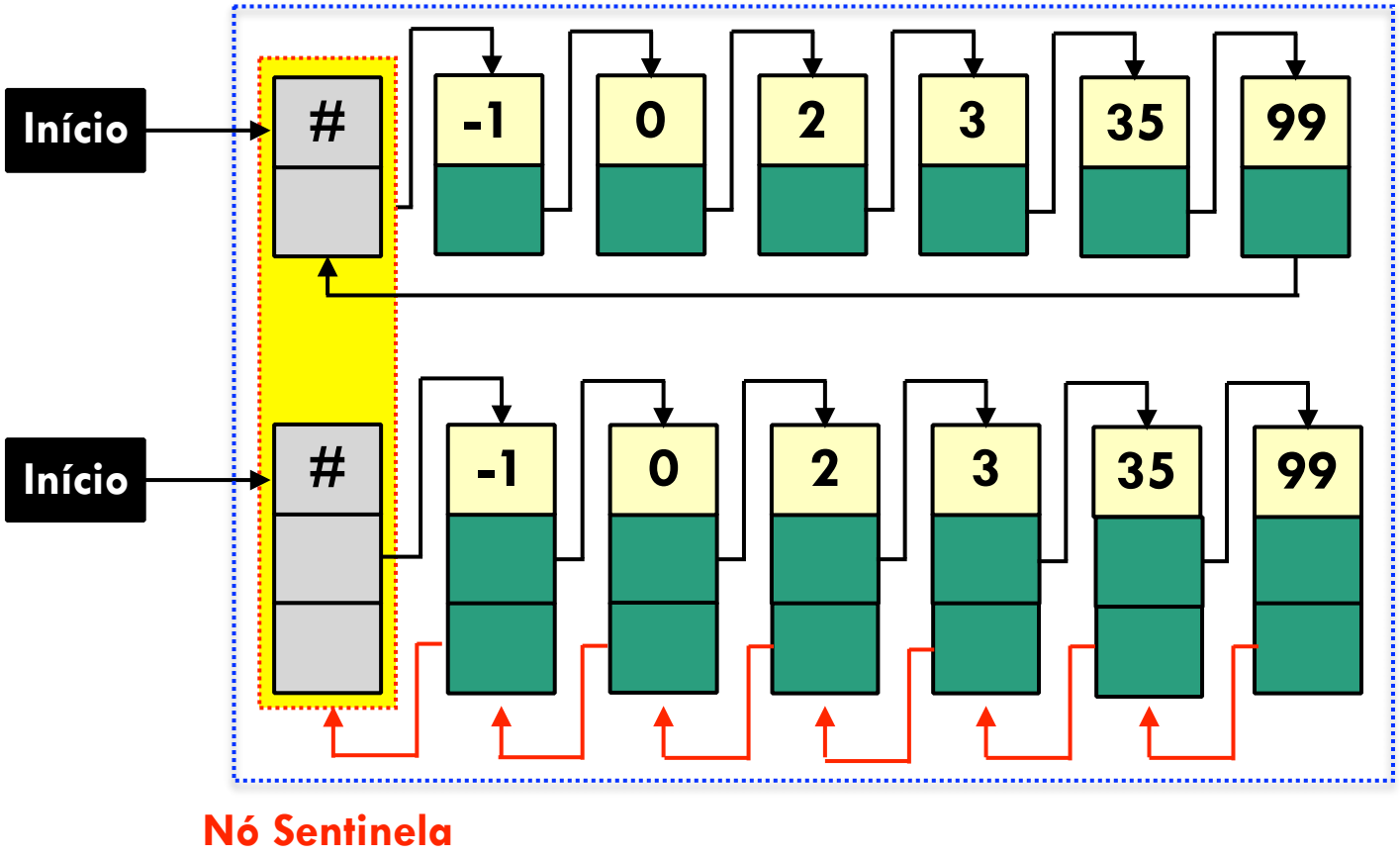
# Tipos de lista



# 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

Dada uma pilha **S**, chave **k**, elemento **x**:

- **iniciar/destruir → iniciar e destruir a fila**
- **pesquisar(S, k) → procurar k em S [TRUE/FALSE]**
- **inserir(S, k) → inserir k em S**
- **remover(S, k) → remover k em S**
- ~~minimo(S) → menor valor armazenado em S~~
- ~~maximo(X) → maior valor armazenado em S~~
- **proximo(S, x) → elemento sucessor a x**
- **anterior(S, x) → elemento antecessor a x**
- **tamanho(S) → tamanho de S**
- **vazia(S) → S está vazia? [TRUE/FALSE]**
- ~~cheia(S) → S está cheia? [TRUE/FALSE]~~

# Lista dinâmica

- Single-linkage

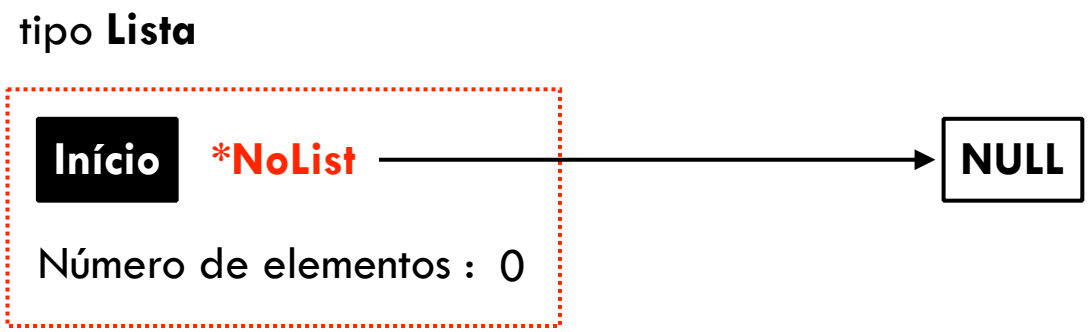
tipo **Lista**

**Início** ***NoList**

Número de elementos :

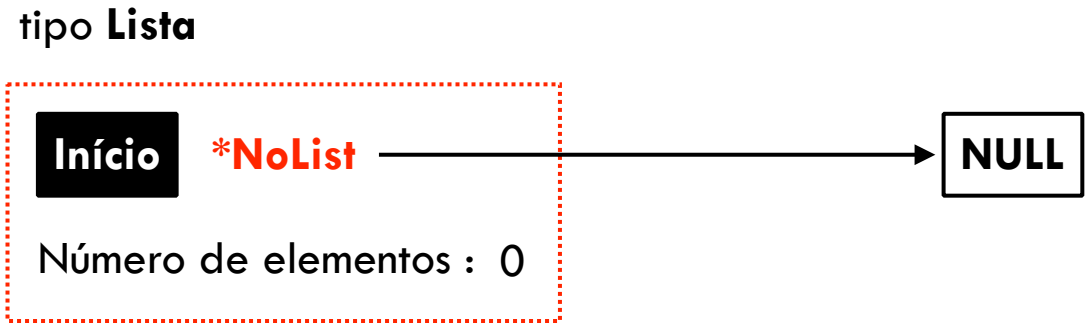
# Inicialização

- Single-linkage



# Inicialização

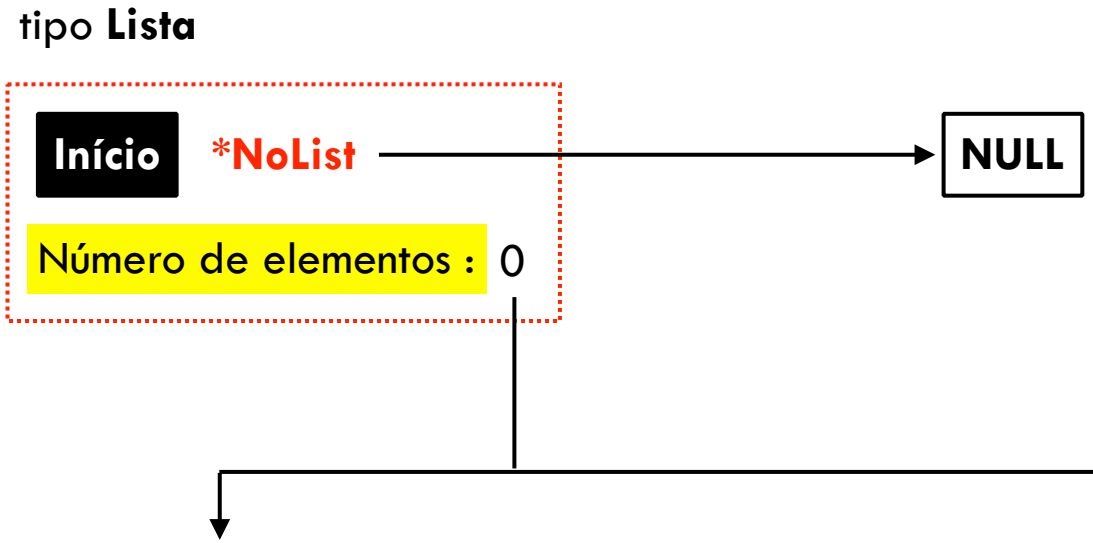
- Single-linkage




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

# Tamanho da Lista

- Single-linkage




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

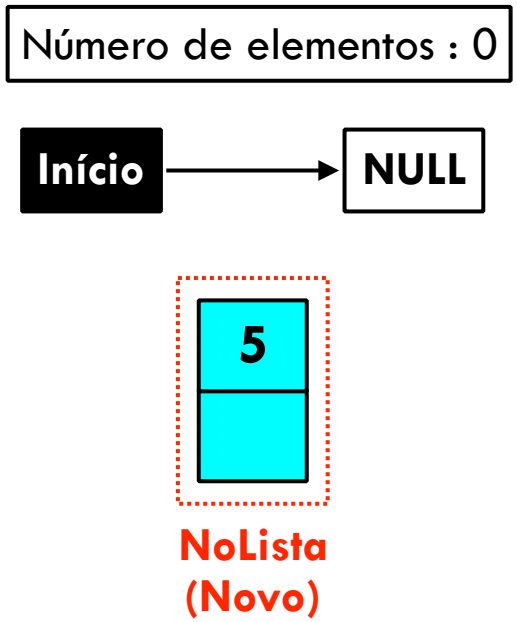
```
estaVazia (L)
1. return (L.tamanho == 0);
// return(L.Inicio == NULL)
```

# Roteiro

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

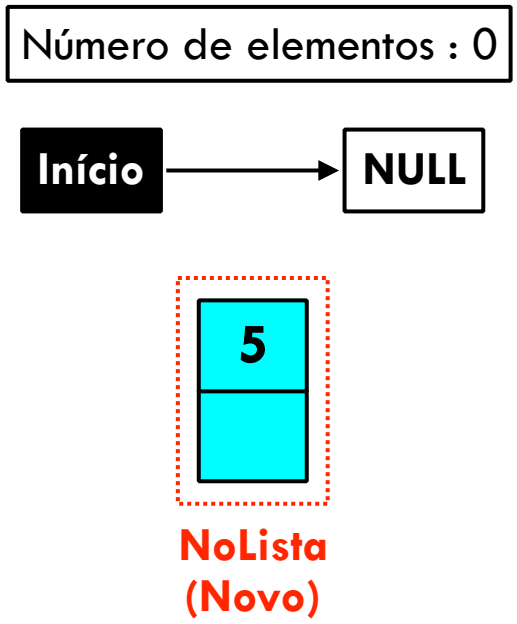
# Inserção (Insert)

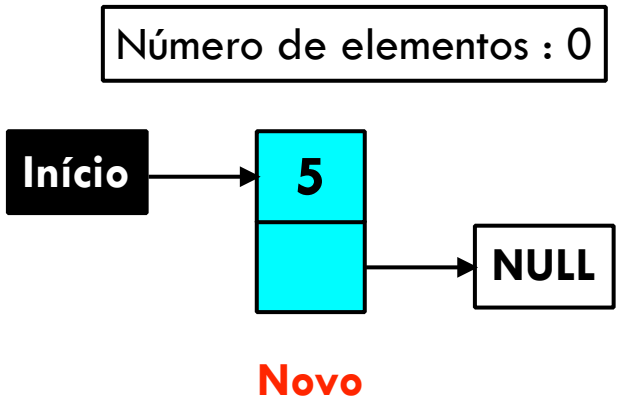
a) primeira inserção (elemento x = 5)



# Inserção (Insert)

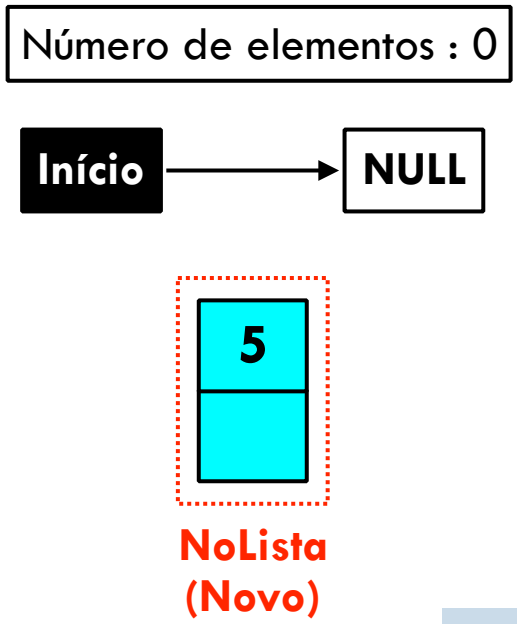
a) primeira inserção (elemento x = 5)

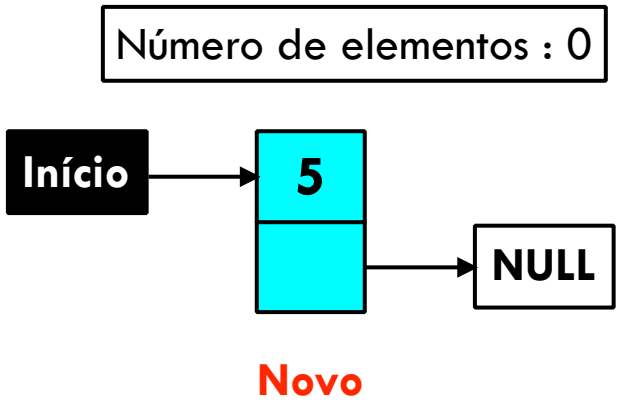


# Inserção (Insert)

a) primeira inserção (elemento x = 5)



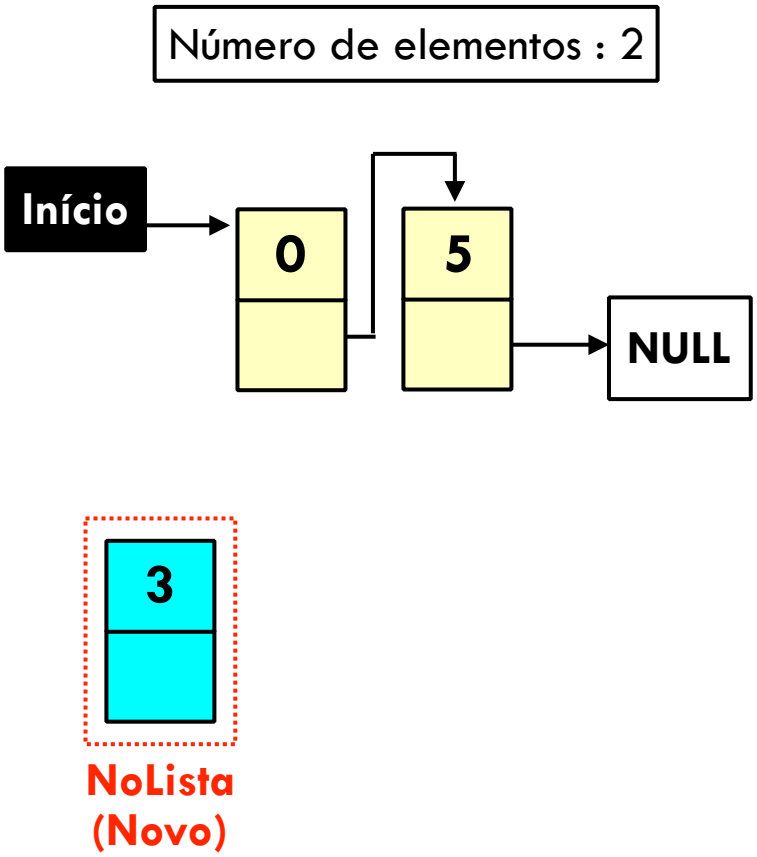



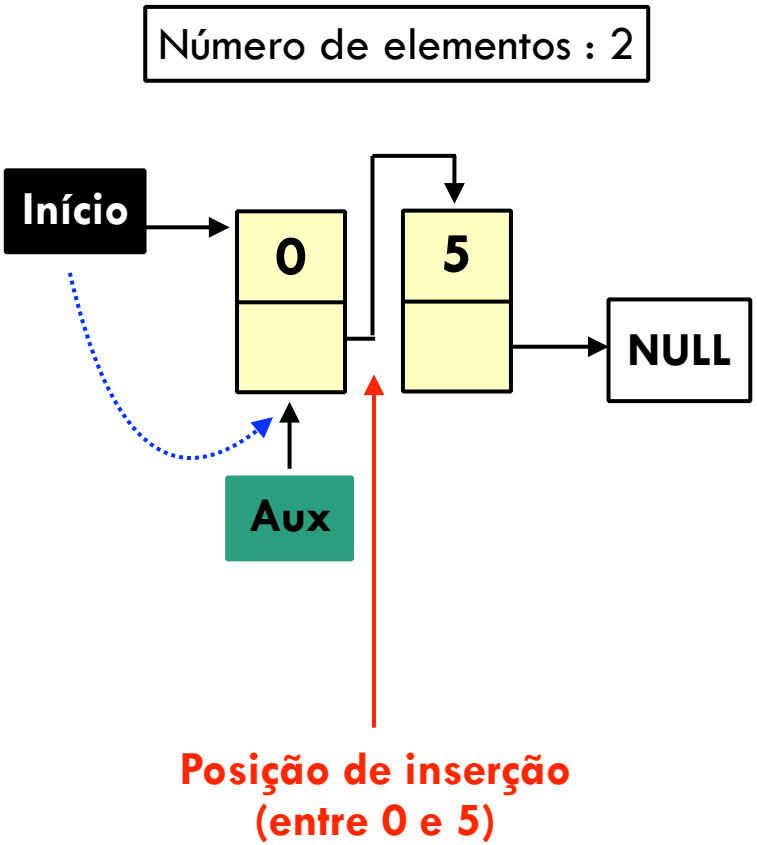

01

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

# Inserção (Insert)

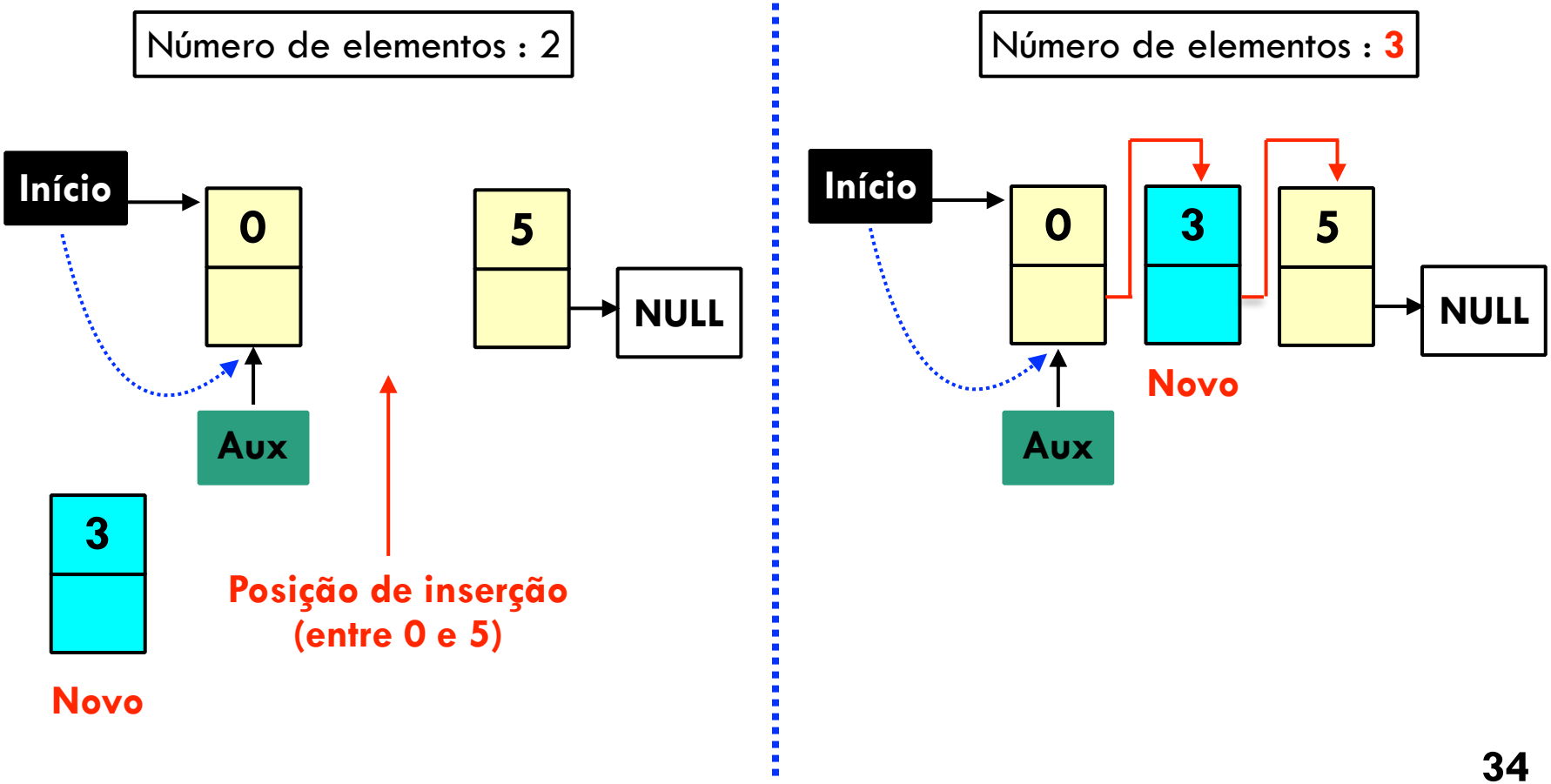
b) não é primeira inserção (elemento x = 3)



# Inserção (Insert)

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



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

Número de elementos : 2

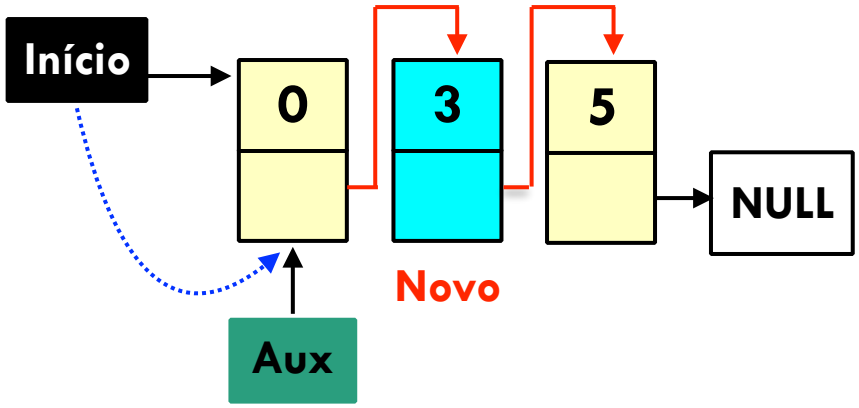
**02**

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

Posição de inserção (entre 0 e 5)

Novo

Número de elementos : **3**



# Inserção (Insert)

```
Insert (L, x)
1. Criar novo nó Novo
2. Novo.chave = x
3. Se for a primeira inserção ou x < 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 > 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
2. Novo.chave = x
3. Se for a primeira inserção ou x < 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 > Aux->proximo.chave)
10.       Aux = Aux->proximo
11.   Novo->proximo = Aux->proximo
12.   Aux->proximo = Novo
13. incrementa contador de elementos
```

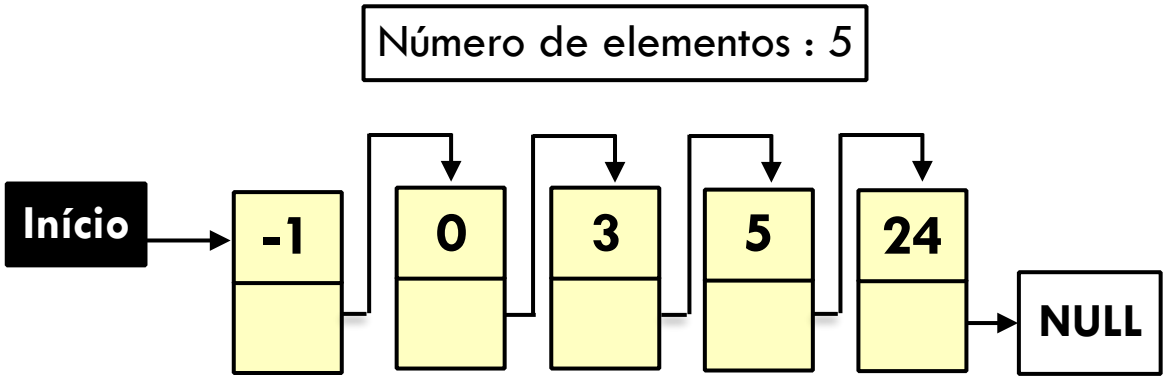
**Obs:** precisaremos de dois ponteiros do tipo NoLista
- um para o novo elemento (**Novo**)
- um para percorrer a lista (**Aux**)

# Roteiro

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

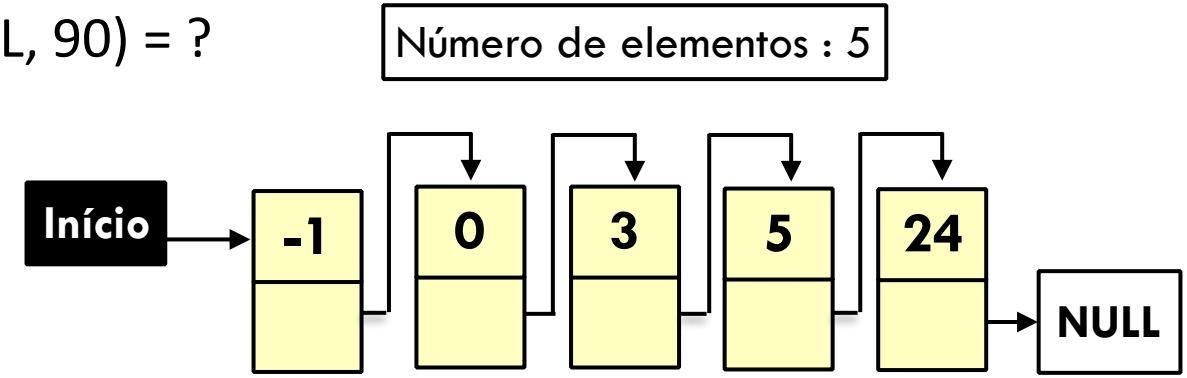
# Pesquisa (Search)

- procura a primeira ocorrência de um elemento
  - se achar retorna
  - senão retorna **NULL** ou nada



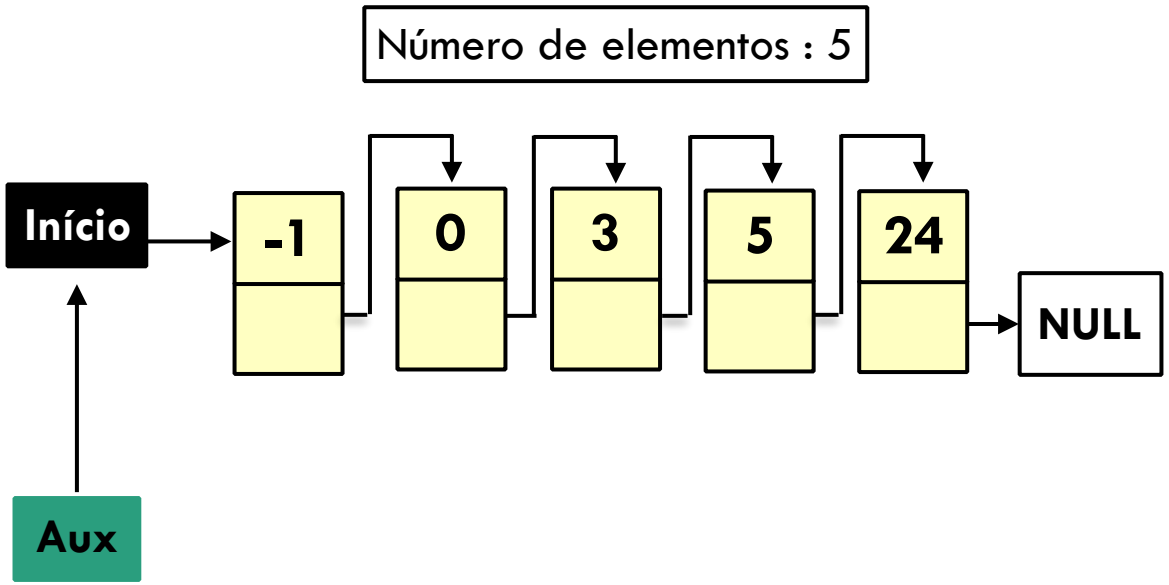
# Pesquisa (Search)

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



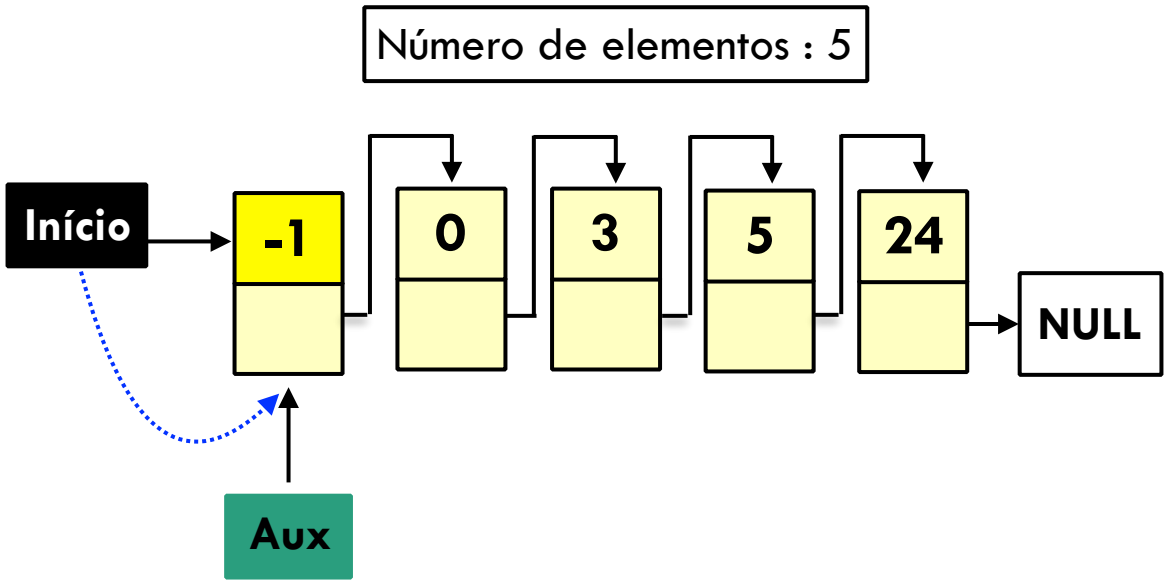
# Pesquisa (Search)

- Search(L, 5) = ?



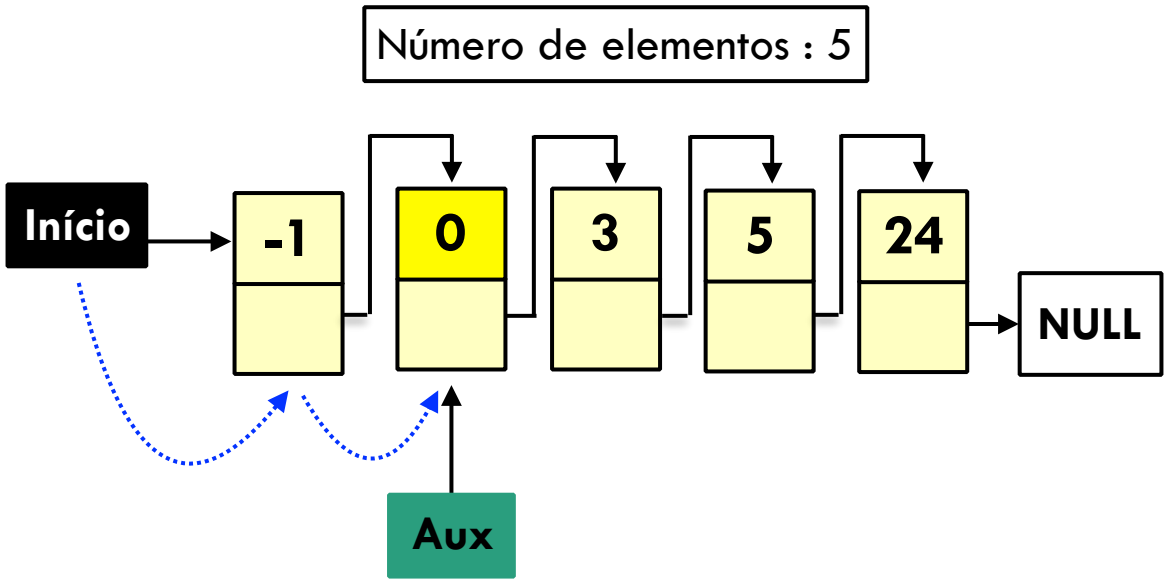
# Pesquisa (Search)

- Search(L, 5) = ?



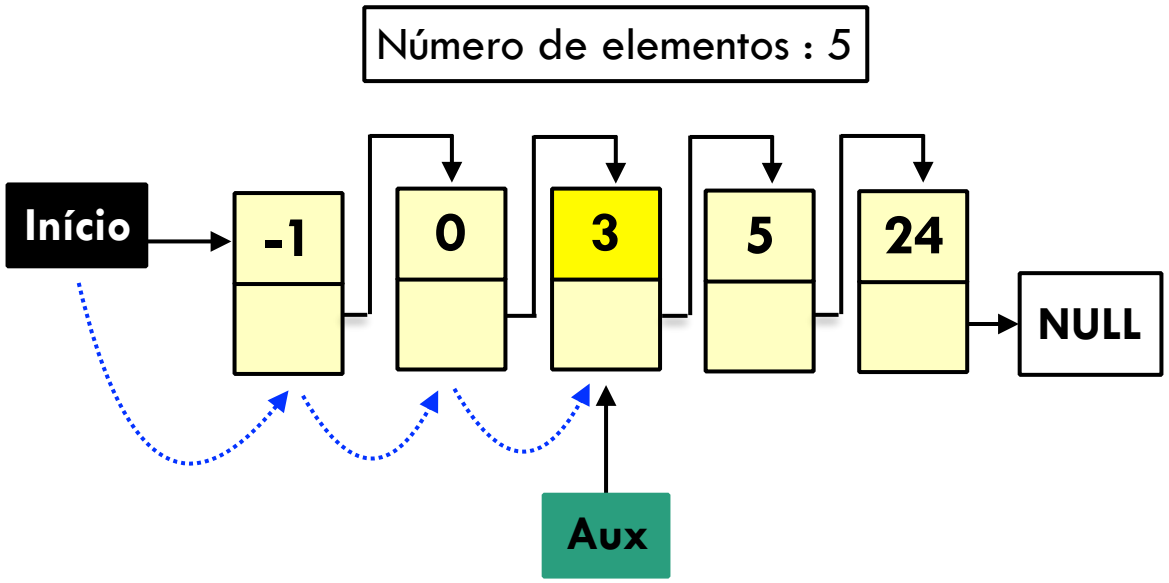
# Pesquisa (Search)

- Search(L, 5) = ?



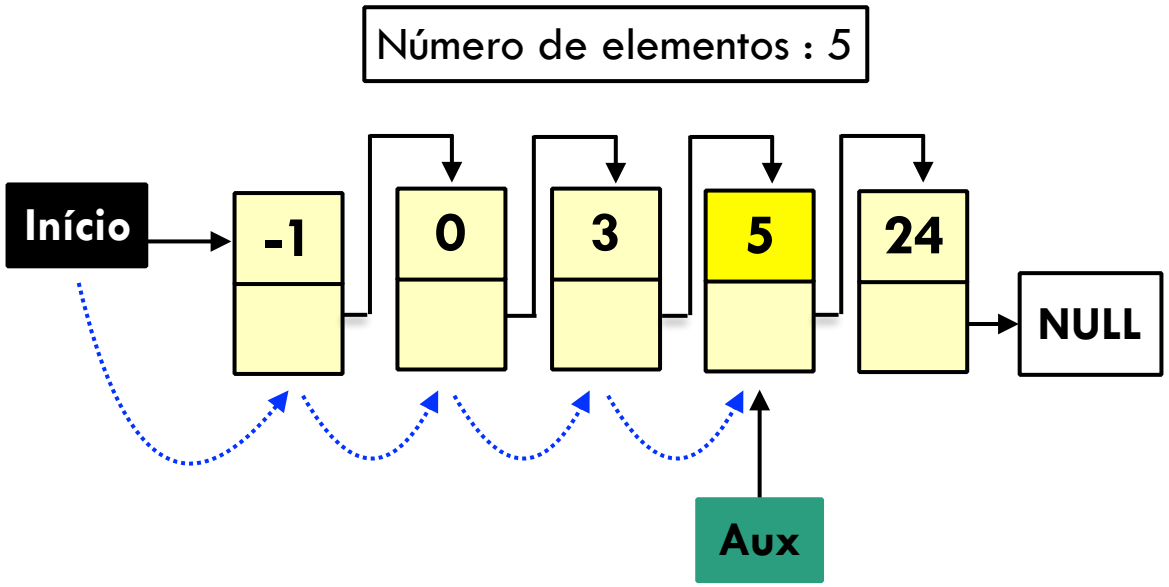
# Pesquisa (Search)

- Search(L, 5) = ?



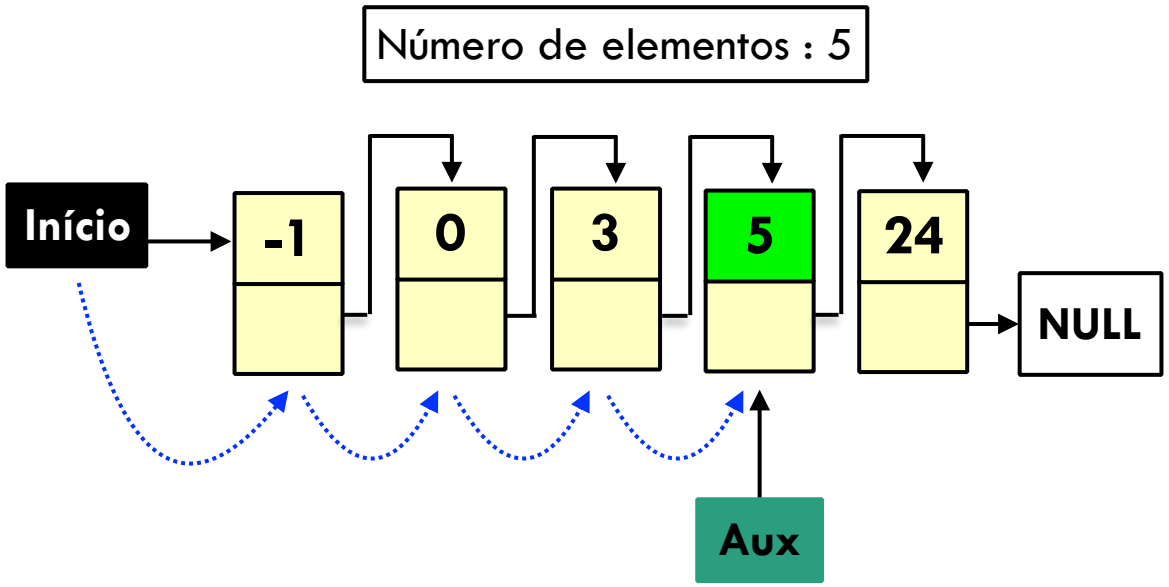
# Pesquisa (Search)

- Search(L, 5) = ?



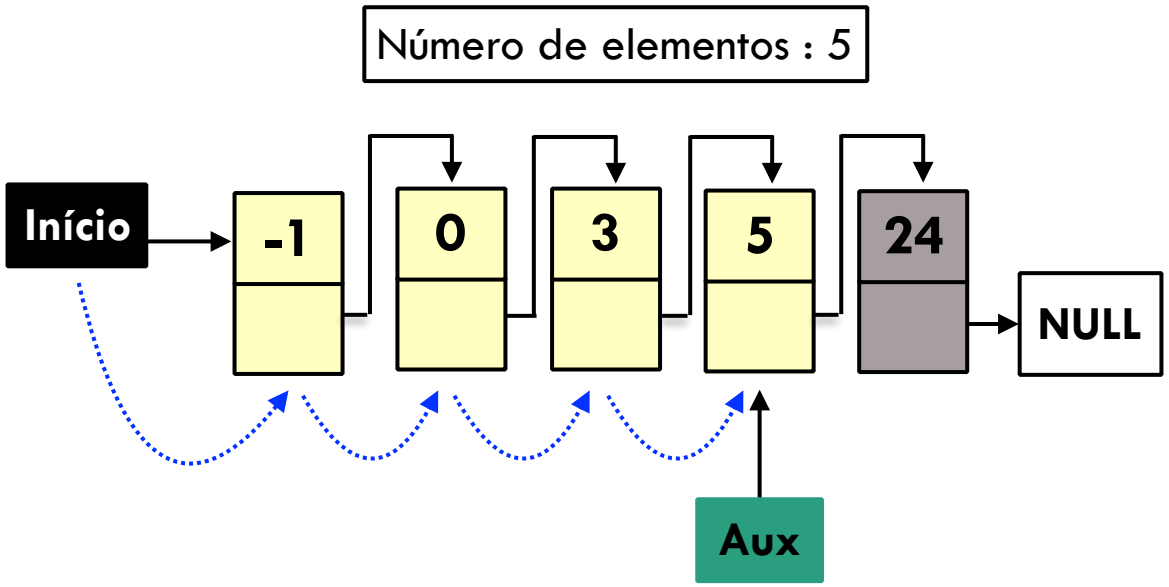
# Pesquisa (Search)

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



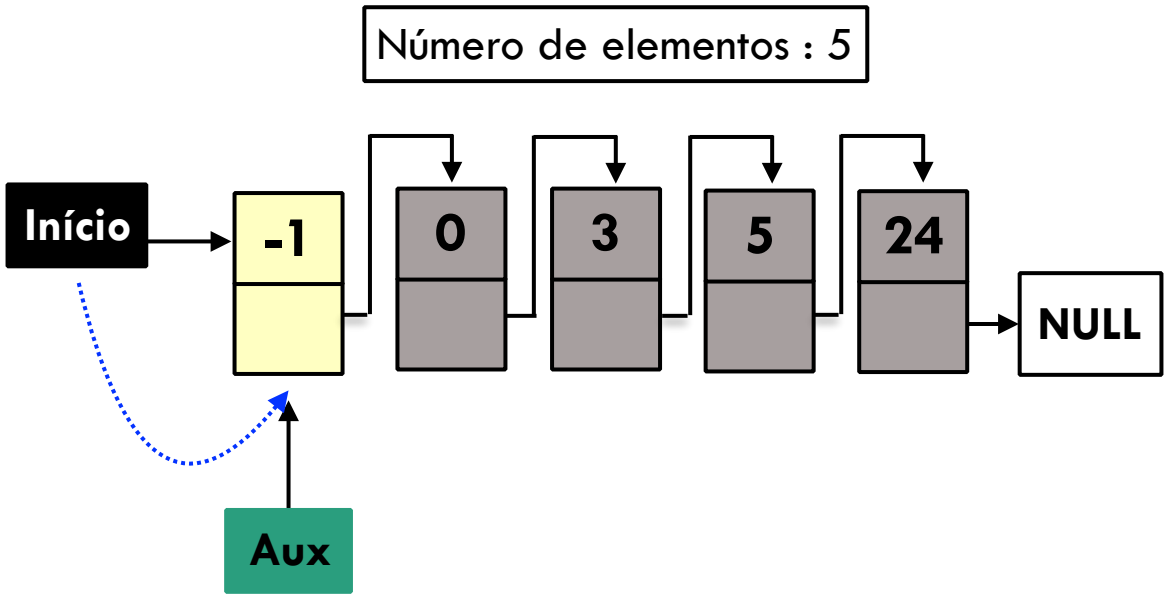
# Pesquisa (Search)

- Search(L, 4) = **Fail !**



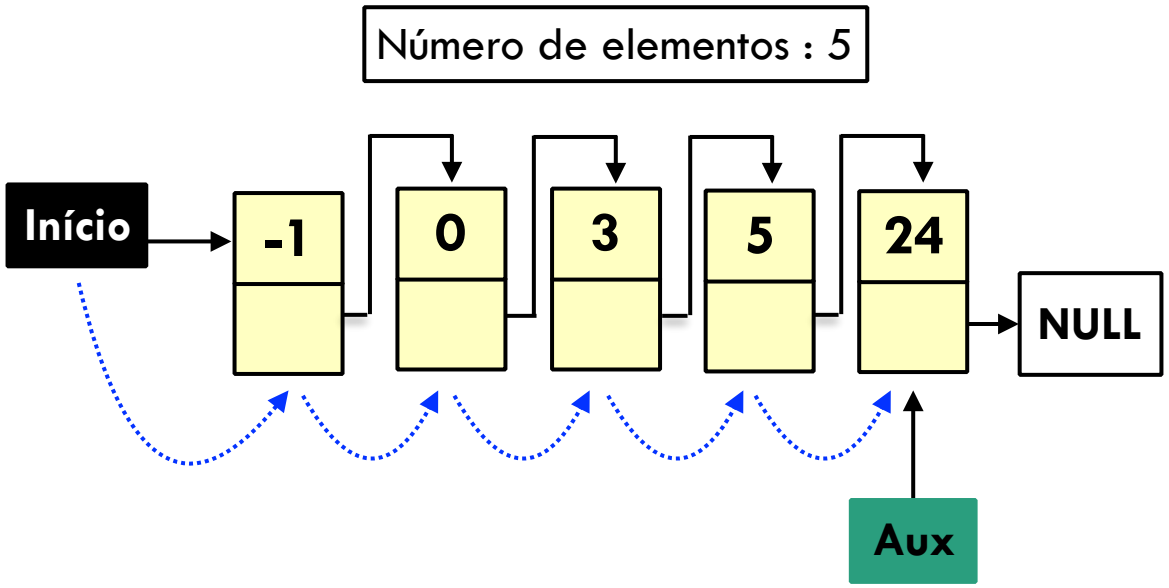
# Pesquisa (Search)

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



# Pesquisa (Search)

- Search(L, 90) = **Fail !**



# Pesquisa (Search)

```
Search.1 (L, x)
1. criar ponteiro Aux
2. Repetir (Aux = L->primeiro; Aux != NULL; Aux = Aux->proximo)
3.      se Aux->x == x
4.          return 1;
5. return 0;
```

# Pesquisa (Search)

```
Search.1 (L, x)
1. criar ponteiro Aux
2. Repetir (Aux = L->primeiro; Aux != NULL; Aux = Aux->proximo)
3.      se Aux->x == x
4.          return 1;
5. return 0;
```

**Obs:** Não muito vantajoso, pois percorre todos os elementos caso a chave não exista.

# Pesquisa (Search)

```
Search.1 (L, x)
1. Se a lista está vazia, return 0
2. criar ponteiro Aux
3. Repetir (Aux = L->primeiro; Aux != NULL; Aux = Aux->proximo)
4.      se Aux->x == x
5.          return 1;
6. return 0;
```

```
Search.2 (L, x)
1. Se a Lista esta vazia
2.     return 0;
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 0
8. return 1;
```

# Roteiro

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

# Remover (remove)

- 5 diferentes casos

**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**

# Exercício 01

- Mãos a obra: implemente um TDA para Lista com alocação dinâmica, e as funções de manipulação.

- Quais TDAs serão necessários?

# Exercício 02

- Implementar a função de remoção de uma lista ordenada

# Complexidade das operações

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

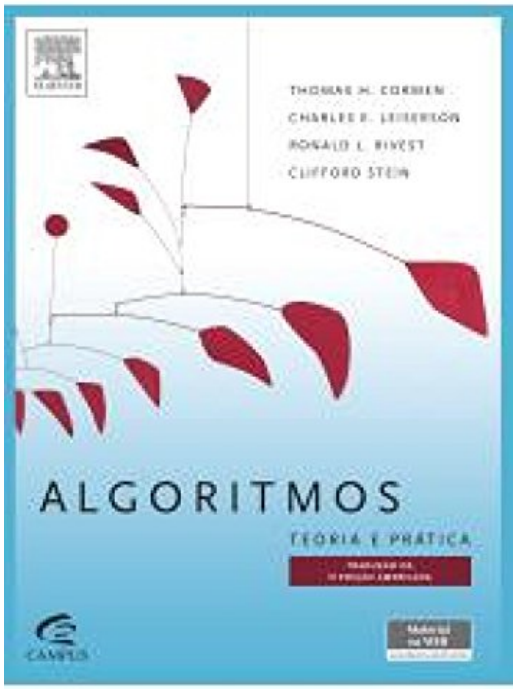
# Complexidade das operações

- Custo (O)
  - 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

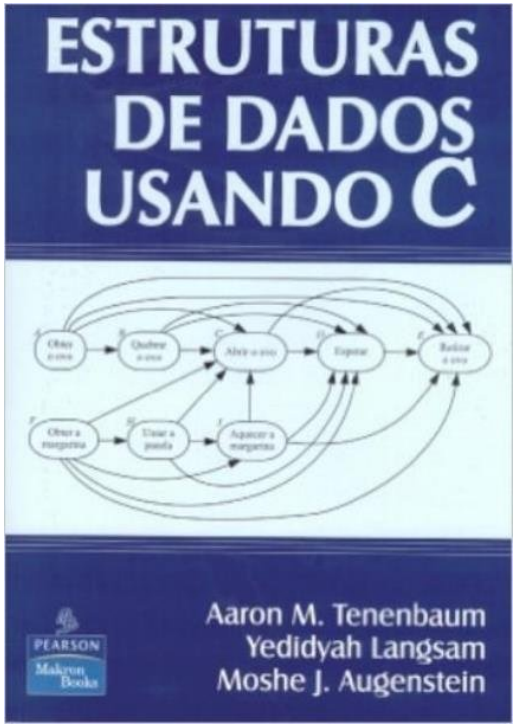
# Roteiro

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

# Referências sugeridas

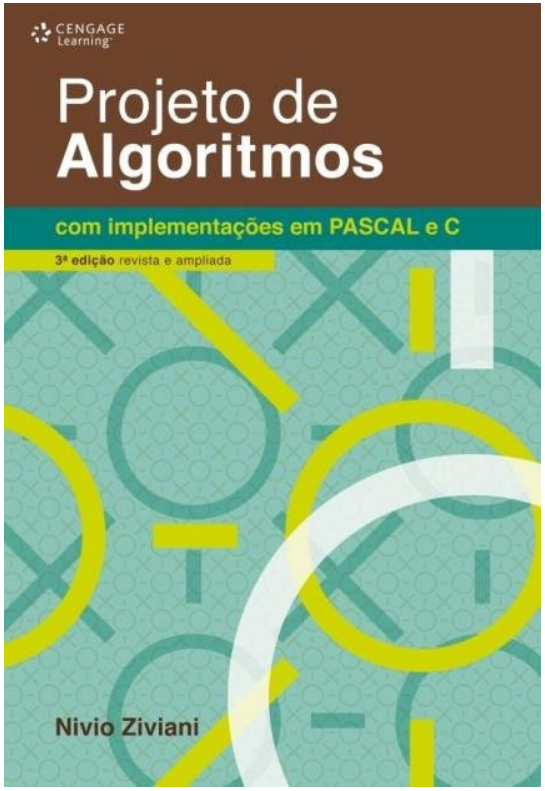



[Cormen et al, 2018]




[Tenenbaum et al, 1995]

# Referências sugeridas




[Ziviani, 2010]


[Drozdek, 2017]

# Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br