



Aula 11 - Técnicas de Encadeamento (Nó Cabeçalho)

Prof. Me. Claudiney R. Tinoco
profclaudineytinoco@gmail.com

Faculdade de Computação (FACOM)
Bacharelado em Ciência da Computação (BCC)
Bacharelado em Sistemas de Informação (BSI)

Algoritmos e Estruturas de Dados 1 (AED1)
GBC024 - GSI006



Introdução

- Diferentes técnicas de encadeamento podem ser aplicadas a fim de **gerar algoritmos mais simples e/ou eficientes**

Introdução

- Diferentes técnicas de encadeamento podem ser aplicadas a fim de **gerar algoritmos mais simples e/ou eficientes**
- **Técnicas mais usuais:**
 - Uso do nó cabeçalho
 - Encadeamento circular
 - Encadeamento duplo

Introdução

- Diferentes técnicas de encadeamento podem ser aplicadas a fim de **gerar algoritmos mais simples e/ou eficientes**
- **Técnicas mais usuais:**
 - **Uso do nó cabeçalho**
 - Encadeamento circular
 - Encadeamento duplo

Uso do Nó Cabeçalho

- Envolve a **inclusão de um nó extra** no início da lista

Ex: $L = \{ 3, 6 \}$



Uso do Nó Cabeçalho

- Envolve a **inclusão de um nó extra** no início da lista

Ex: $L = \{ 3, 6 \}$



Ex: $L = \{ \}$ (**Lista Vazia**)





Uso do Nó Cabeçalho

- Simplifica as operações de inserção e remoção na **TAD lista ordenada**
 - Não precisa tratar o 1º nó da lista separado



Uso do Nó Cabeçalho

- Simplifica as operações de inserção e remoção na **TAD lista ordenada**
 - Não precisa tratar o 1º nó da lista separado
- O nó cabeçalho **não pertence a lista**
 - Campo **info** não guarda valor de elemento
 - Para evitar desperdício, pode ser usado para guardar alguma informação útil sobre a lista

Uso do Nó Cabeçalho

- Simplifica as operações de inserção e remoção na **TAD lista ordenada**
 - Não precisa tratar o 1º nó da lista separado
- O nó cabeçalho **não pertence a lista**
 - Campo **info** não guarda valor de elemento
 - Para evitar desperdício, pode ser usado para guardar alguma informação útil sobre a lista

Ex: Quantidade de elementos na lista





Estrutura de Representação em C

- **Mesma estrutura** da implementação SEM cabeçalho

- Declaração da estrutura nó inteiro no **lista.c**:

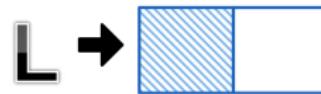
```
struct no {  
    int info;  
    struct no * prox;  
};
```

- Definição do tipo de dado lista no **lista.h**:

```
typedef struct no * Lista;
```

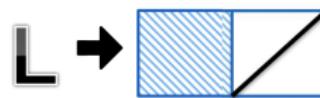
Operação **cria_list**

- A lista **sempre aponta para o nó cabeçalho**
 - O **nó cabeçalho deve ser alocado dinamicamente** na criação da lista



Operação **cria_list**

- A lista **sempre aponta para o nó cabeçalho**
 - O **nó cabeçalho deve ser alocado dinamicamente** na criação da lista
- **Colocar a lista no estado de vazia**
 - Lista vazia é representada pelo nó cabeçalho apontando para **NULL**





Operação **cria_list**a

- Implementação em C:

```
Lista cria_lista() {  
    // Aloca nó cabeçalho  
    Lista cab;  
    cab = (Lista) malloc(sizeof(struct no));  
  
    }  
}
```

Operação **cria_list**

- **Implementação em C:**

```
Lista cria_lista() {
    // Aloca nó cabeçalho
    List* cab;
    cab = (List*) malloc(sizeof(struct no));
    // Coloca lista no estado de vazia
    if (cab != NULL) { // Só se alocação NÃO falhar
        cab->prox = NULL;
        cab->info = 0; } // Opcional: guardar qtde
    }
```



Operação **cria_list**

- **Implementação em C:**

```
Lista cria_lista() {
    // Aloca nó cabeçalho
    List* cab;
    cab = (List*) malloc(sizeof(struct no));
    // Coloca lista no estado de vazia
    if (cab != NULL) { // Só se alocação NÃO falhar
        cab->prox = NULL;
        cab->info = 0; } // Opcional: guardar qtde
    return cab;
}
```



Operação **cria_lista**

- **Implementação em C:**

```
Lista cria_lista() {
    // Aloca nó cabeçalho
    Lista cab;
    cab = (Lista) malloc(sizeof(struct no));
    // Coloca lista no estado de vazia
    if (cab != NULL) { // Só se alocação NÃO falhar
        cab->prox = NULL;
        cab->info = 0; } // Opcional: guardar qtde
    return cab;
}
```

Se alocação falhar, retorna NULL



Operação **lista_vazia()**

- Verifica se a lista está na condição de vazia:
 - Se o campo **prox** do **nó cabeçalho** é **NULL**
 - **Nó cabeçalho** é apontado pela lista
- **Implementação em C:**

```
int lista_vazia(Lista lst) {  
    if (lst->prox == NULL)  
        return 1; // Lista vazia  
    else  
        return 0; // Lista NÃO vazia  
}
```



Operação lista_cheia()

- **Teoricamente** a lista NÃO fica cheia na alocação dinâmica (lista infinita)
- **Na prática**, tamanho da lista é limitado pelo espaço de memória
 - Função *malloc()* retorna **NULL** quando não é possível alocar um novo nó



Operação de Inserção

- Especialmente indicada para o **TAD lista ordenada**
- Implementação visa **simplificação** do código
 - Evita tratar separadamente o 1º nó da lista
 - Remove do algoritmo a parte que altera o conteúdo da variável que representa a **LISTA**



Operação de Inserção (Lista NÃO Ordenada)

- Existem **2 cenários** possíveis de inserção:
 - Lista sem elementos (**lista vazia**)
 - Lista com 1 ou mais elementos
- **Ambos são tratados da mesma forma:**
 - Alocação do novo nó
 - Preenchimento dos campos do novo nó
 - **info** recebe o valor do novo elemento
 - **prox** recebe o endereço armazenado pelo **nó cabeçalho**
 - **Nó cabeçalho** recebe o endereço do novo nó
 - **Lista é passada por valor** e não por referência

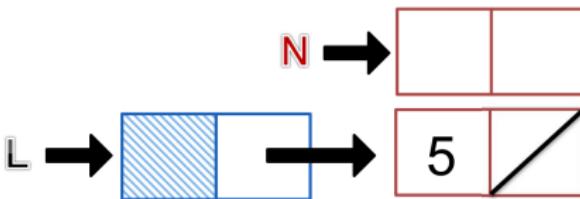
Operação de Inserção (Lista NÃO Ordenada)

- Ex: Inserir 3



Operação de Inserção (Lista NÃO Ordenada)

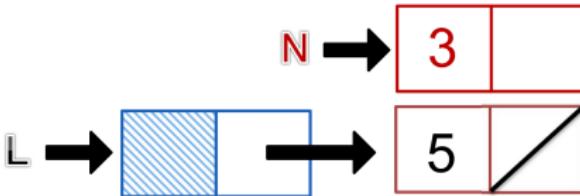
- Ex: Inserir 3
 - Alocar um novo nó



Operação de Inserção (Lista NÃO Ordenada)

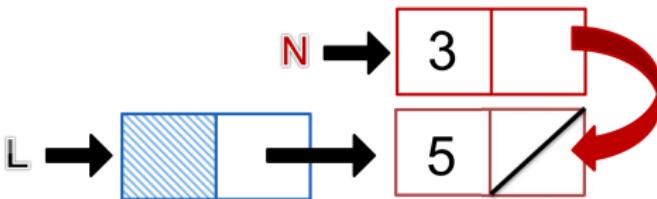
- Ex: Inserir 3

- Alocar um novo nó
- Preencher os campos do novo nó
 - Campo **info** = valor do elemento



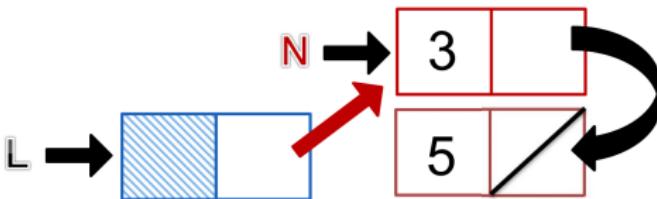
Operação de Inserção (Lista NÃO Ordenada)

- Ex: Inserir 3
 - Alocar um novo nó
 - Preencher os campos do novo nó
 - Campo **info** = valor do elemento
 - Campo **prox** = valor do campo **prox** do nó cabeçalho



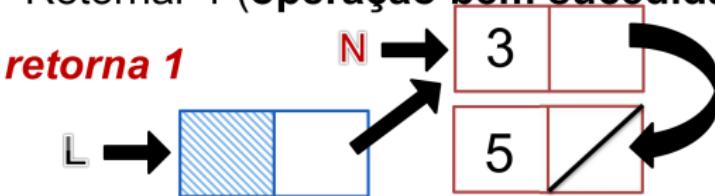
Operação de Inserção (Lista NÃO Ordenada)

- Ex: Inserir 3
 - Alocar um novo nó
 - Preencher os campos do novo nó
 - Campo **info** = valor do elemento
 - Campo **prox** = valor do campo **prox** do nó cabeçalho
 - Atribuir ao campo **prox** do nó cabeçalho o **endereço do novo nó**



Operação de Inserção (Lista NÃO Ordenada)

- Ex: Inserir 3
 - Alocar um novo nó
 - Preencher os campos do novo nó
 - Campo **info** = valor do elemento
 - Campo **prox** = valor do campo **prox** do nó cabeçalho
 - Atribuir ao campo **prox** do nó cabeçalho o **endereço do novo nó**
 - Retornar 1 (**operação bem sucedida**)





Operação de Inserção (Lista NÃO Ordenada)

- **Implementação em C:**

```
int insere_elem (Lista lst, int elem) {  
    // Aloca um novo nó  
    Lista N = (Lista) malloc(sizeof(struct no));  
    if (N == NULL) { return 0; } // Falha: nó não alocado  
    // Preenche os campos do novo nó  
    N->info = elem; // Insere o conteúdo (valor do elem)  
    N->prox = lst->prox; // Aponta para o 1º nó atual da lista  
    lst->prox = N; // Faz o nó cabeçalho apontar para o novo nó  
    lst->info++; // Opcional: Incrementa qtde de nós na lista  
    return 1;  
}
```



Operação de Inserção (Lista NÃO Ordenada)

- **Implementação em C:** *passagem por valor*

```
int insere_elem (Lista lst, int elem) {  
    // Aloca um novo nó  
    Lista N = (Lista) malloc(sizeof(struct no));  
    if (N == NULL) { return 0; } // Falha: nó não alocado  
    // Preenche os campos do novo nó  
    N->info = elem; // Insere o conteúdo (valor do elem)  
    N->prox = lst->prox; // Aponta para o 1º nó atual da lista  
    lst->prox = N; // Faz o nó cabeçalho apontar para o novo nó  
    lst->info++; // Opcional: Incrementa qtde de nós na lista  
    return 1;  
}
```

Operação de Inserção (Lista NÃO Ordenada)

- **Implementação em C:** passagem por referência
(mantém interface)

```
int insere_elem (Lista *lst, int elem) {  
    // Aloca um novo nó  
    Lista N = (Lista) malloc(sizeof(struct no));  
    if (N == NULL) { return 0; } // Falha: nó não alocado  
    // Preenche os campos do novo nó  
    N->info = elem; // Insere o conteúdo (valor do elem)  
    N->prox = (*lst)->prox; // Aponta para o 1º nó atual da lista  
    (*lst)->prox = N; // Faz o nó cabeçalho apontar para o novo nó  
    lst->info++; // Opcional: Incrementa qtde de nós na lista  
    return 1;  
}
```



Operação de Inserção (Lista Ordenada)

- Inserção na **posição correta**
 - Envolve **percorrimento**
- Existem **4 cenários** possíveis de inserção:
 - Lista está vazia
 - Novo elemento \leq 1º nó da lista
 - Novo elemento $>$ último nó da lista
 - Novo elemento entre o 1º e o último nó da lista



Operação de Inserção (Lista Ordenada)

- Inserção na **posição correta**
 - Envolve **percorrimento**
 - Existem **4 cenários** possíveis de inserção:
 - Lista está vazia
 - Novo elemento \leq 1º nó da lista
 - Novo elemento $>$ último nó da lista
 - Novo elemento entre o 1º e o último nó da lista
- Tratados da mesma forma (similar à Lista NÃO ordenada)

Operação de Inserção (Lista Ordenada)

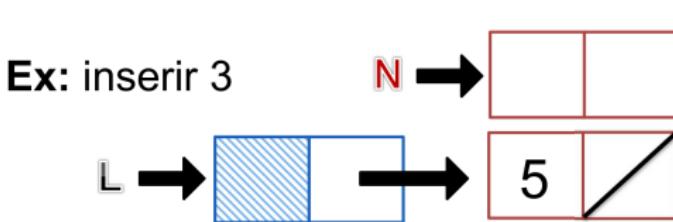
- **Lista vazia ou Elemento \leq 1º nó da lista:**

Ex: inserir 3



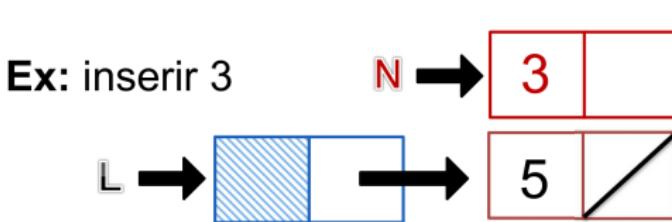
Operação de Inserção (Lista Ordenada)

- **Lista vazia ou Elemento \leq 1º nó da lista:**
 - Alocar um novo nó



Operação de Inserção (Lista Ordenada)

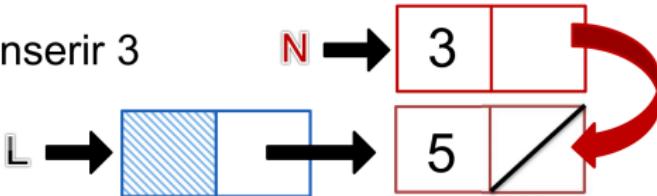
- **Lista vazia ou Elemento \leq 1º nó da lista:**
 - Alocar um novo nó
 - Preencher os campos do novo nó
 - Campo *info* = valor do elemento



Operação de Inserção (Lista Ordenada)

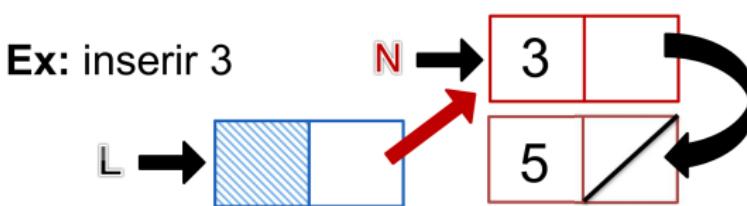
- **Lista vazia ou Elemento \leq 1º nó da lista:**
 - Alocar um novo nó
 - Preencher os campos do novo nó
 - Campo **info** = valor do elemento
 - Campo **prox** = valor do campo **prox** do **nó cabeçalho**

Ex: inserir 3



Operação de Inserção (Lista Ordenada)

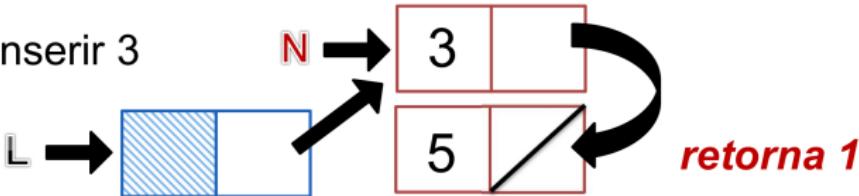
- **Lista vazia ou Elemento \leq 1º nó da lista:**
 - Alocar um novo nó
 - Preencher os campos do novo nó
 - Campo **info** = valor do elemento
 - Campo **prox** = valor do campo **prox** do **nó cabeçalho**
 - Atribuir ao campo **prox** do **nó cabeçalho** o **endereço do novo nó**



Operação de Inserção (Lista Ordenada)

- **Lista vazia ou Elemento \leq 1º nó da lista:**
 - Alocar um novo nó
 - Preencher os campos do novo nó
 - Campo **info** = valor do elemento
 - Campo **prox** = valor do campo **prox** do **nó cabeçalho**
 - Atribuir ao campo **prox** do **nó cabeçalho** o **endereço do novo nó**

Ex: inserir 3



Operação de Inserção (Lista Ordenada)

- Elemento > último nó da lista:

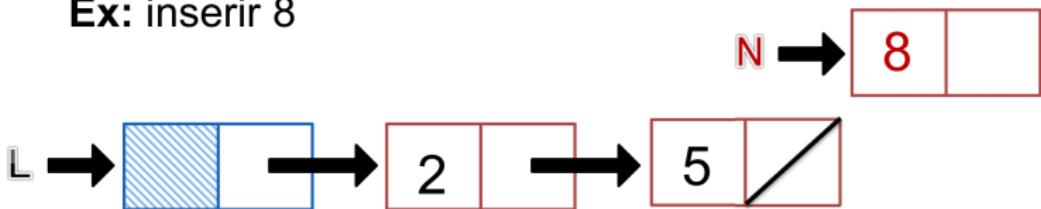
Ex: inserir 8



Operação de Inserção (Lista Ordenada)

- Elemento > último nó da lista:
 - Alocar um novo nó e preencher o campo *info*

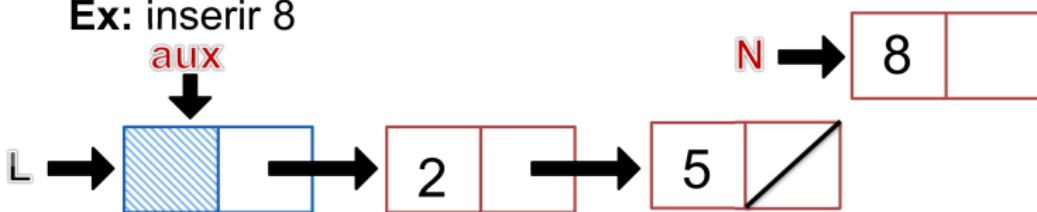
Ex: inserir 8



Operação de Inserção (Lista Ordenada)

- **Elemento > último nó da lista:**
 - Alocar um novo nó e preencher o campo *info*
 - Percorrer a lista até seu final (~~o~~ sucessor)
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**

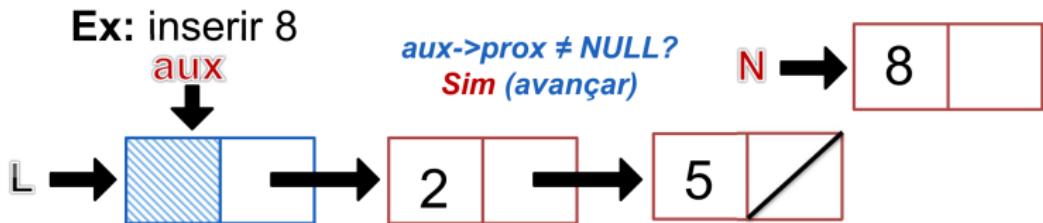
Ex: inserir 8





Operação de Inserção (Lista Ordenada)

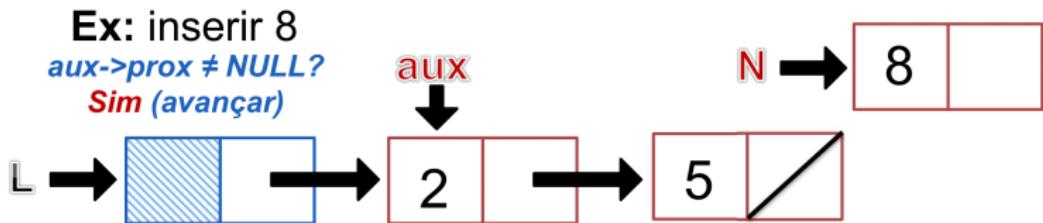
- **Elemento > último nó da lista:**
 - Alocar um novo nó e preencher o campo *info*
 - Percorrer a lista até seu final (\exists sucessor)
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Avançar enquanto \exists sucessor ($aux \rightarrow prox \neq NULL$)





Operação de Inserção (Lista Ordenada)

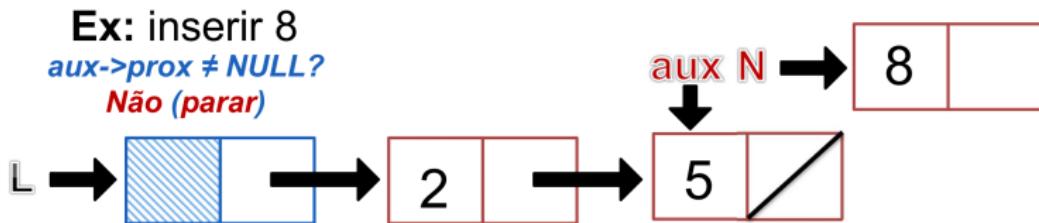
- **Elemento > último nó da lista:**
 - Alocar um novo nó e preencher o campo *info*
 - Percorrer a lista até seu final (\exists sucessor)
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Avançar enquanto \exists sucessor ($aux \rightarrow prox \neq NULL$)





Operação de Inserção (Lista Ordenada)

- **Elemento > último nó da lista:**
 - Alocar um novo nó e preencher o campo **info**
 - Percorrer a lista até seu final (\exists sucessor)
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Avançar enquanto \exists sucessor ($aux \rightarrow prox \neq NULL$)

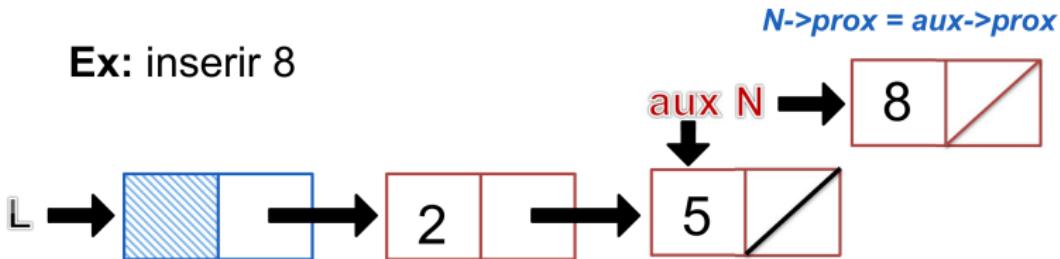




Operação de Inserção (Lista Ordenada)

- Elemento > último nó da lista:
 - Alocar um novo nó e preencher o campo **info**
 - Percorrer a lista até seu final (\exists sucessor)
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Avançar enquanto \exists sucessor ($aux \rightarrow prox \neq NULL$)
 - Preencher campo **prox** do novo nó

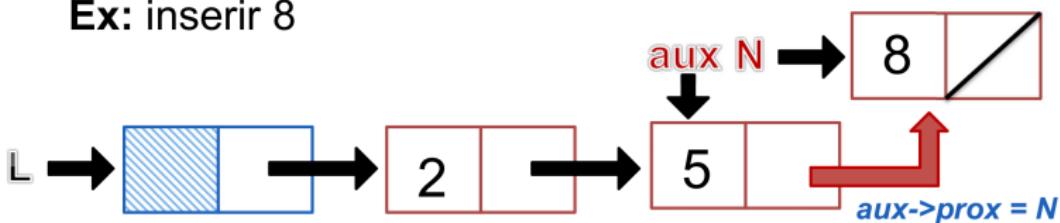
Ex: inserir 8



Operação de Inserção (Lista Ordenada)

- **Elemento > último nó da lista:**
 - Alocar um novo nó e preencher o campo **info**
 - Percorrer a lista até seu final (\exists sucessor)
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Avançar enquanto \exists sucessor ($aux \rightarrow prox \neq NULL$)
 - Preencher campo **prox** do novo nó
 - Fazer o último nó apontar para o novo nó

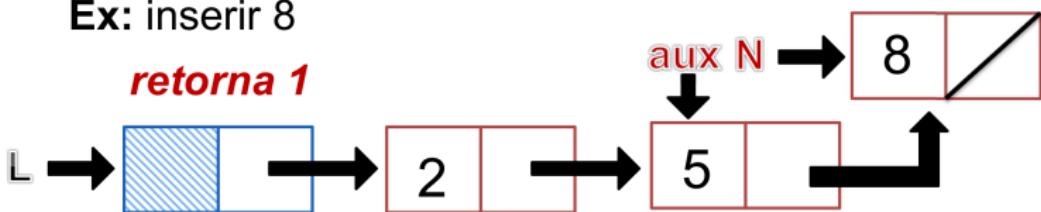
Ex: inserir 8



Operação de Inserção (Lista Ordenada)

- **Elemento > último nó da lista:**
 - Alocar um novo nó e preencher o campo **info**
 - Percorrer a lista até seu final (\exists sucessor)
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Avançar enquanto \exists sucessor ($aux \rightarrow prox \neq NULL$)
 - Preencher campo **prox** do novo nó
 - Fazer o último nó apontar para o novo nó

Ex: inserir 8



Operação de Inserção (Lista Ordenada)

- 1º nó < Elemento ≤ último nó da lista:

Ex: inserir 6

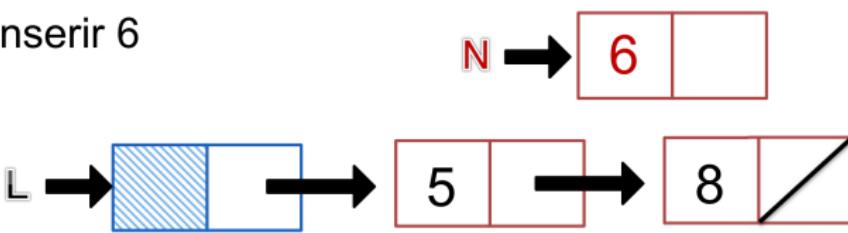




Operação de Inserção (Lista Ordenada)

- 1º nó < Elemento ≤ último nó da lista:
 - Alocar um novo nó e preencher o campo *info*

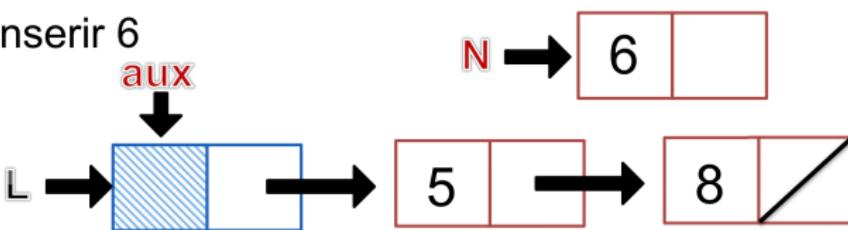
Ex: inserir 6



Operação de Inserção (Lista Ordenada)

- 1º $\text{nó} < \text{Elemento} \leq \text{último nó da lista:}$
 - Alocar um novo nó e preencher o campo *info*
 - Percorrer a lista até achar nó maior ou igual
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**

Ex: inserir 6



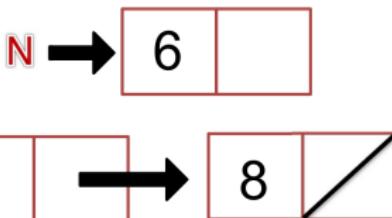
Operação de Inserção (Lista Ordenada)

- **1º nó < Elemento ≤ último nó da lista:**
 - Alocar um novo nó e preencher o campo **info**
 - Percorrer a lista até achar nó maior ou igual
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Verificar **info** do **sucessor de aux** (**aux->prox->info**)

Ex: inserir 6

$5 < 6?$ Sim
(avançar)

aux
↓



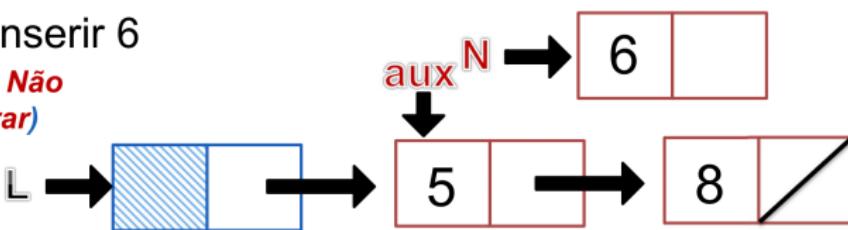


Operação de Inserção (Lista Ordenada)

- 1º $\text{nó} < \text{Elemento} \leq \text{último nó da lista:}$
 - Alocar um novo nó e preencher o campo *info*
 - Percorrer a lista até achar nó maior ou igual
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Verificar *info* do **sucessor de aux** (*aux->prox->info*)

Ex: inserir 6

$8 < 6?$ **Não**
(parar)

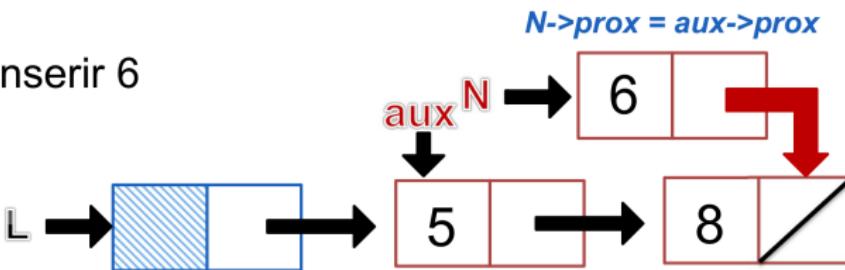




Operação de Inserção (Lista Ordenada)

- **1º nó < Elemento ≤ último nó da lista:**
 - Alocar um novo nó e preencher o campo **info**
 - Percorrer a lista até achar nó maior ou igual
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Verificar **info** do **sucessor de aux** (**aux->prox->info**)
 - Preencher campo **prox** do novo nó

Ex: inserir 6

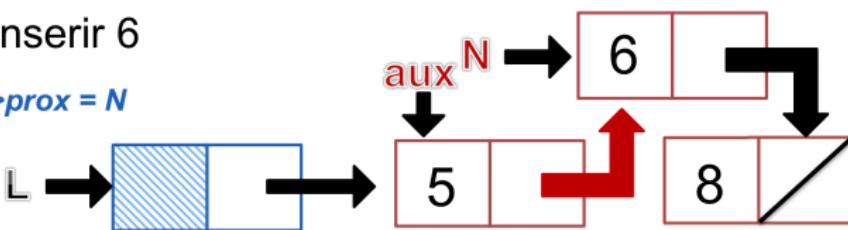


Operação de Inserção (Lista Ordenada)

- **1º nó < Elemento ≤ último nó da lista:**
 - Alocar um novo nó e preencher o campo **info**
 - Percorrer a lista até achar nó maior ou igual
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Verificar **info** do **sucessor de aux** (**aux->prox->info**)
 - Preencher campo **prox** do novo nó
 - **Nó apontado por aux** aponta para o novo nó

Ex: inserir 6

$aux \rightarrow prox = N$

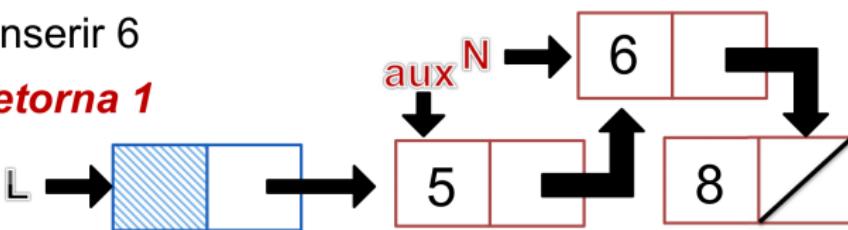


Operação de Inserção (Lista Ordenada)

- **1º nó < Elemento ≤ último nó da lista:**
 - Alocar um novo nó e preencher o campo **info**
 - Percorrer a lista até achar nó maior ou igual
 - Usar **ponteiro auxiliar** com ender. do **nó cabeçalho**
 - Verificar **info** do **sucessor de aux** (**aux->prox->info**)
 - Preencher campo **prox** do novo nó
 - **Nó apontado por aux** aponta para o novo nó

Ex: inserir 6

retorna 1



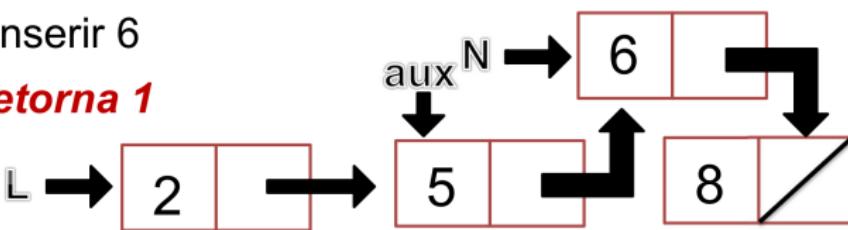


Operação de Inserção (Lista Ordenada)

- **1º nó < Elemento ≤ último nó da lista:**
 - Alocar um novo nó e preencher o campo **info**
 - Percorrer a lista até achar nó maior ou igual
 - Usar ponteiro auxiliar
 - Verificar **info** do **sucessor de aux** (**aux->prox->info**)
 - Preencher campo **prox** do novo nó
 - Nó apontado por **aux** aponta para o novo nó

Ex: inserir 6

retorna 1



Operação de Inserção (Lista Ordenada)

- **Implementação em C (COM cabeçalho):**

```
int insere_ord (Lista *lst, int elem) {  
    // Aloca um novo nó  
    Lista N = (Lista) malloc(sizeof(struct no));  
    if (N == NULL) { return 0; } // Falha: nó não alocado  
    N->info = elem; // Insere o conteúdo (valor do elem)  
if (lista_vazia(*lst) || elem <= (*lst)->info) {  
    N->prox = *lst; // Aponta para o 1º nó atual da lista  
*lst = N; // Faz a lista apontar para o novo nó  
return 1; }
```

...

REMOVER



Operação de Inserção (Lista Ordenada)

- **Implementação em C (COM cabeçalho):**

...

// Percorramento da lista

*Lista aux = *lst; // Faz aux apontar para nó cabeçalho*

while (aux->prox != NULL && aux->prox->info < elem)

aux = aux->prox; // Avança

// Insere o novo nó na lista

N->prox = aux->prox;

aux->prox = N;

lst->info++; // Opcional: Incrementa qtde de nós na lista

return 1; }



Operação de Remoção

- Necessita de percorrimento da lista
 - Busca pelo elemento a ser removido
- Remoção não afeta o ponteiro **LISTA**
 - 1º nó sempre é o nó cabeçalho
 - Envolve apenas **mudança no campo prox dos nós**
- **Critério de ordenação** afeta quando não existe o elemento na lista
 - **Lista não ordenada:** tem que percorrer **até o final**
 - **Lista ordenada:** percorrer **até achar nó maior**



Operação de Remoção (Lista NÃO Ordenada)

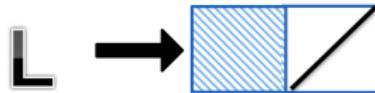
- Existem **3 casos** possíveis de remoção:
 - Lista vazia
 - Elemento existente na lista
 - Elemento não está na lista



Operação de Remoção (Lista NÃO Ordenada)

- **Lista vazia:**

Ex: remover 5

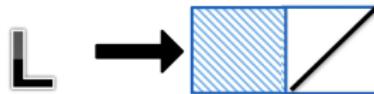


Operação de Remoção (Lista NÃO Ordenada)

- **Listas vazias:**
 - Não existe elemento a ser removido

Ex: remover 5

5





Operação de Remoção (Lista NÃO Ordenada)

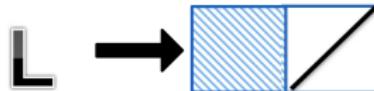
- **Listas vazias:**

- Não existe elemento a ser removido
- Retorna 0 (**operação falha**)

Ex: remover 5

5

retorna 0



Operação de Remoção (Lista NÃO Ordenada)

- Elemento existente na lista:

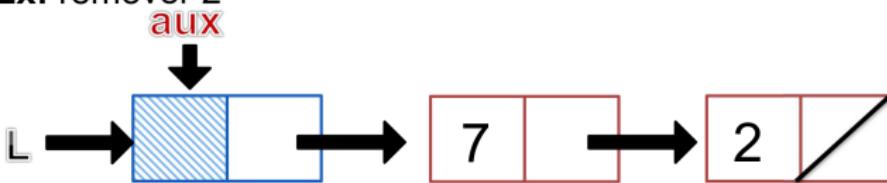
Ex: remover 2



Operação de Remoção (Lista NÃO Ordenada)

- **Elemento existente na lista:**
 - Colocar um ponteiro auxiliar no nó cabeçalho

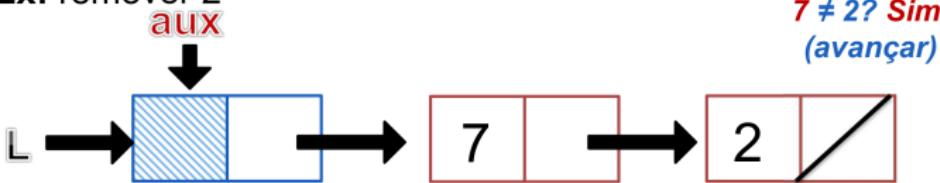
Ex: remover 2



Operação de Remoção (Lista NÃO Ordenada)

- Elemento existente na lista:
 - Colocar um ponteiro auxiliar no nó cabeçalho
 - Percorrer a lista até encontrar o elemento
 - Verificar o campo *info* do sucessor de *aux* (*aux->prox->info ≠ elem*)

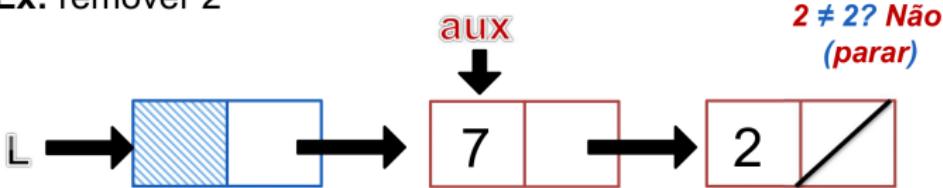
Ex: remover 2



Operação de Remoção (Lista NÃO Ordenada)

- Elemento existente na lista:
 - Colocar um ponteiro auxiliar no nó cabeçalho
 - Percorrer a lista até encontrar o elemento
 - Verificar o campo *info* do sucessor de *aux* (*aux->prox->info ≠ elem*)

Ex: remover 2

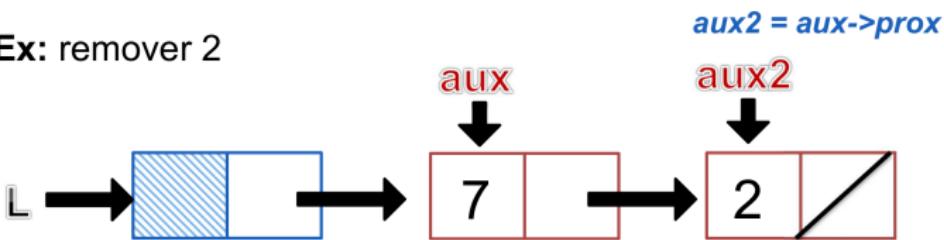




Operação de Remoção (Lista NÃO Ordenada)

- Elemento existente na lista:
 - Colocar um ponteiro auxiliar no **nó cabeçalho**
 - Percorrer a lista até encontrar o elemento
 - Verificar o campo **info do sucessor de aux** ($\text{aux} \rightarrow \text{prox} \rightarrow \text{info} \neq \text{elem}$)
 - Apontar o nó a ser removido (**sucessor de aux**)

Ex: remover 2

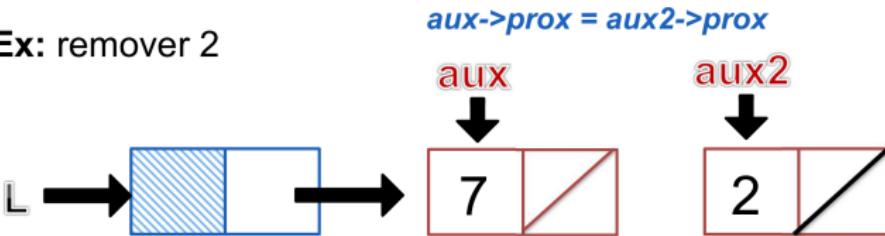




Operação de Remoção (Lista NÃO Ordenada)

- Elemento existente na lista:
 - Colocar um ponteiro auxiliar no **nó cabeçalho**
 - Percorrer a lista até encontrar o elemento
 - Verificar o campo **info do sucessor de aux** ($\text{aux} \rightarrow \text{prox} \rightarrow \text{info} \neq \text{elem}$)
 - Apontar o nó a ser removido (**sucessor de aux**)
 - Fazer o **nó apontado por aux** aponta para **o sucessor de seu sucessor** ($(\text{aux} \rightarrow \text{prox}) \rightarrow \text{prox}$)

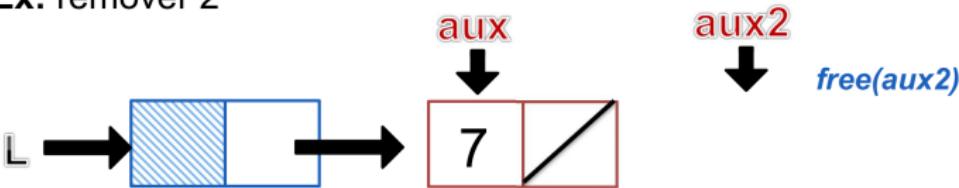
Ex: remover 2



Operação de Remoção (Lista NÃO Ordenada)

- **Elemento existente na lista:**
 - Colocar um ponteiro auxiliar no **nó cabeçalho**
 - Percorrer a lista até encontrar o elemento
 - Verificar o campo **info do sucessor de aux** ($\text{aux} \rightarrow \text{prox} \rightarrow \text{info} \neq \text{elem}$)
 - Apontar o nó a ser removido (**sucessor de aux**)
 - Fazer o **nó apontado por aux** aponta para **o sucessor de seu sucessor** ($(\text{aux} \rightarrow \text{prox}) \rightarrow \text{prox}$)
 - Liberar a memória alocada para o nó removido

Ex: remover 2

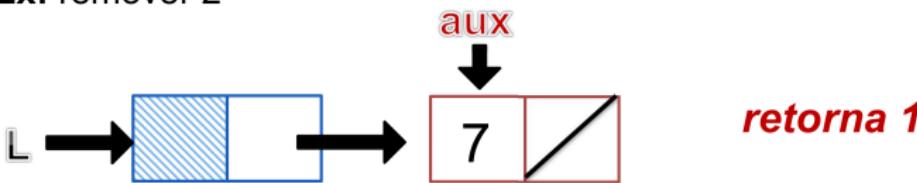




Operação de Remoção (Lista NÃO Ordenada)

- **Elemento existente na lista:**
 - Colocar um ponteiro auxiliar no **nó cabeçalho**
 - Percorrer a lista até encontrar o elemento
 - Verificar o campo **info do sucessor de aux** ($\text{aux} \rightarrow \text{prox} \rightarrow \text{info} \neq \text{elem}$)
 - Apontar o nó a ser removido (**sucessor de aux**)
 - Fazer o **nó apontado por aux** aponta para **o sucessor de seu sucessor** ($(\text{aux} \rightarrow \text{prox}) \rightarrow \text{prox}$)
 - Liberar a memória alocada para o nó removido

Ex: remover 2



Operação de Remoção (Lista NÃO Ordenada)

- Elemento não está na lista:

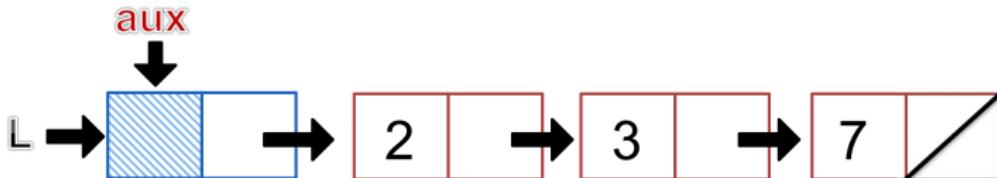
Ex: remover 1



Operação de Remoção (Lista NÃO Ordenada)

- **Elemento não está na lista:**
 - Colocar um ponteiro auxiliar no **nó cabeçalho**

Ex: remover 1

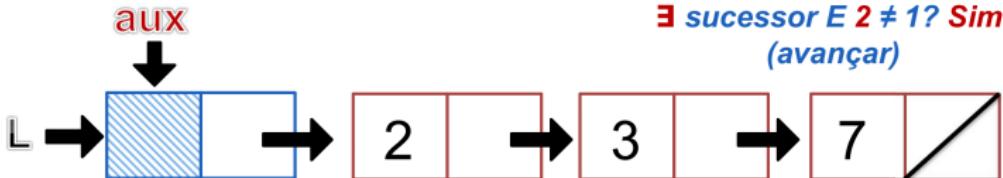




Operação de Remoção (Lista NÃO Ordenada)

- Elemento não está na lista:
 - Colocar um ponteiro auxiliar no nó cabeçalho
 - Percorrer a lista até o seu final
 - Avançar **aux** enquanto \exists sucessor de **aux**

Ex: remover 1

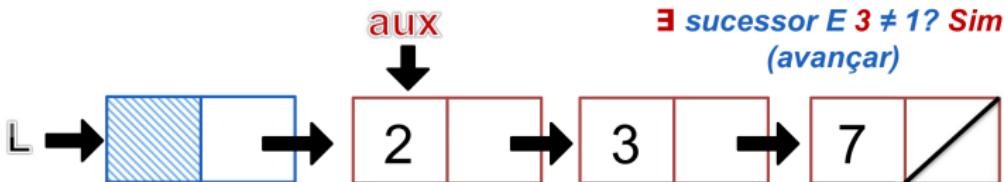




Operação de Remoção (Lista NÃO Ordenada)

- Elemento não está na lista:
 - Colocar um ponteiro auxiliar no nó cabeçalho
 - Percorrer a lista até o seu final
 - Avançar *aux* enquanto \exists sucessor de *aux*

Ex: remover 1



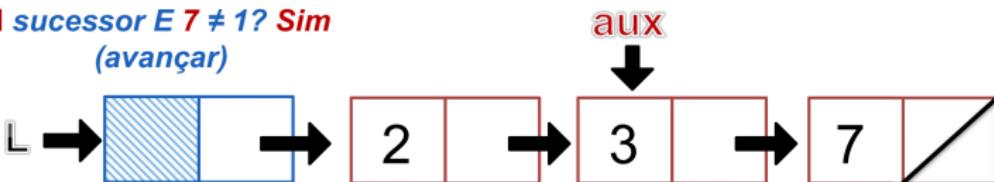


Operação de Remoção (Lista NÃO Ordenada)

- Elemento não está na lista:
 - Colocar um ponteiro auxiliar no nó cabeçalho
 - Percorrer a lista até o seu final
 - Avançar **aux** enquanto \exists sucessor de **aux**

Ex: remover 1

\exists sucessor E $7 \neq 1$? Sim
(avançar)

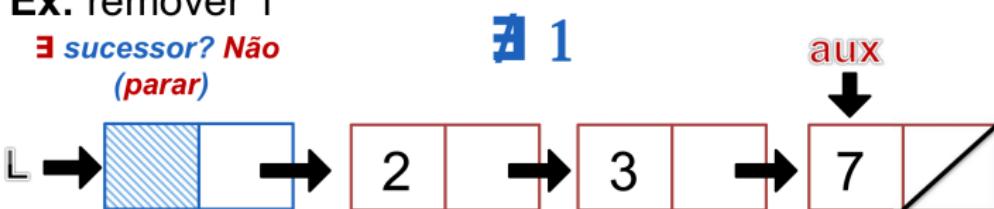


Operação de Remoção (Lista NÃO Ordenada)

- **Elemento não está na lista:**
 - Colocar um ponteiro auxiliar no **nó cabeçalho**
 - Percorrer a lista até o seu final
 - Avançar **aux** enquanto \exists sucessor de **aux**
 - Não existe o elemento desejado

Ex: remover 1

**\exists sucessor? Não
(parar)**

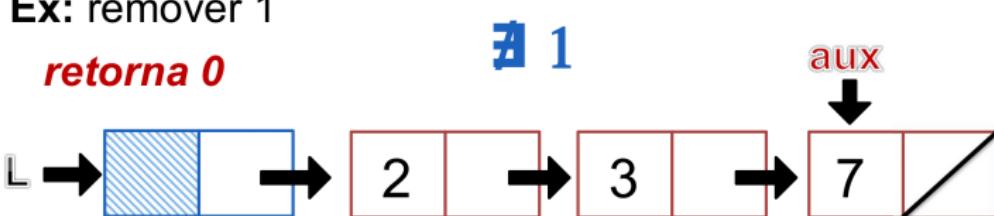




Operação de Remoção (Lista NÃO Ordenada)

- **Elemento não está na lista:**
 - Colocar um ponteiro auxiliar no **nó cabeçalho**
 - Percorrer a lista até o seu final
 - Avançar **aux** enquanto \exists sucessor de **aux**
 - Não existe o elemento desejado
 - Retornar 0 (**operação falha**)

Ex: remover 1





Operação de Remoção (Lista NÃO Ordenada)

- Implementação em C (COM cabeçalho):

```
int remove_elem (Lista *lst, int elem) {
    if (lista_vazia(lst) == 1)
        return 0; // Falha
    Lista aux = *lst; // Ponteiro auxiliar para o nó cabeçalho
    // Trata elemento = 1º nó da lista
    if (elem == (*lst)->info) {
        *lst = aux->prox; // Lista aponta para o 2º nó
        free(aux); // Libera memória alocada
        return 1;
    }
}
```

...

REMOVER



Operação de Remoção (Lista NÃO Ordenada)

- **Implementação em C (COM cabeçalho):**

```
...
    // Percorimento até achar o elem ou final de lista
    while (aux->prox != NULL && aux->prox->info != elem)
        aux = aux->prox;
    if (aux->prox == NULL) // Trata final de lista
        return 0; // Falha
    // Remove elemento ≠ 1º nó da lista
    Lista aux2 = aux->prox; // Aponta nó a ser removido
    aux->prox = aux2->prox; // Retira nó da lista
    free(aux2); // Libera memória alocada
    (*lst)->info--; // Opcional: Decrementa qtde de nós na lista
    return 1; }
```



Operação de Remoção (Lista Ordenada)

- Existem **4 cenários** possíveis de remoção:
 - Lista está vazia
 - Elemento existente na lista
 - Elemento não está na lista



Operação de Remoção (Lista Ordenada)

- Existem **6 cenários** possíveis de remoção:

- Lista está vazia
- Elemento existente na lista
- Elemento não está na lista

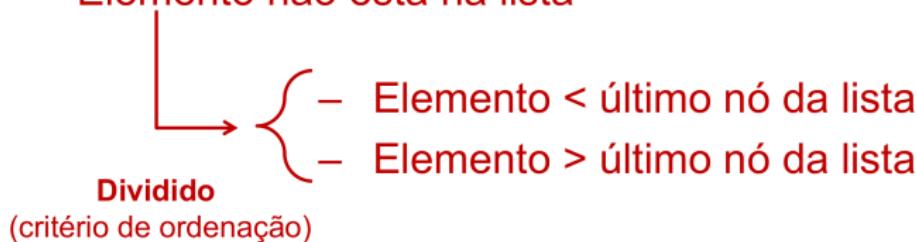
} similar ao TAD
Lista NÃO Ordenada



Operação de Remoção (Lista Ordenada)

- Existem **6 cenários** possíveis de remoção:

- Lista está vazia
- Elemento existente na lista
- ~~Elemento não está na lista~~



Operação de Remoção (Lista Ordenada)

- Elemento < último nó da lista:

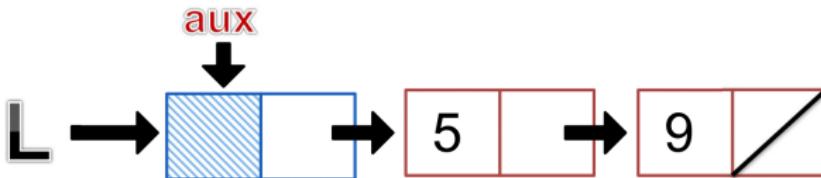
Ex: remover 7



Operação de Remoção (Lista Ordenada)

- Elemento < último nó da lista:
 - Percorre a lista até encontrar nó > elemento
 - Colocar um ponteiro auxiliar no nó cabeça

Ex: remover 7

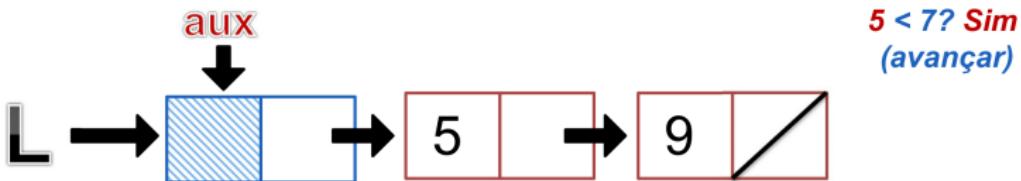




Operação de Remoção (Lista Ordenada)

- Elemento < último nó da lista:
 - Percorre a lista até encontrar nó > elemento
 - Colocar um ponteiro auxiliar no nó cabeçalho
 - Avançar ponteiro se campo *info* do **sucessor de aux** for menor que elemento (*aux->prox->info < elem*)

Ex: remover 7

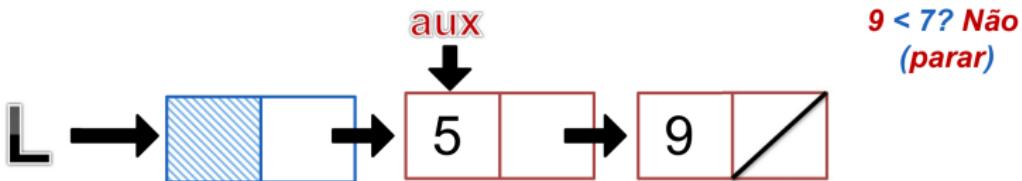




Operação de Remoção (Lista Ordenada)

- Elemento < último nó da lista:
 - Percorre a lista até encontrar nó > elemento
 - Colocar um ponteiro auxiliar no nó cabeçalho
 - Avançar ponteiro se campo *info* do **sucessor de aux** for menor que elemento (*aux->prox->info < elem*)

Ex: remover 7

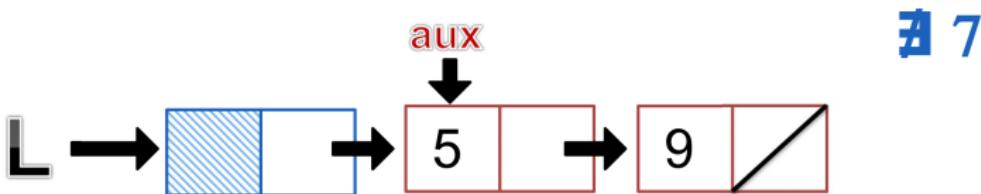




Operação de Remoção (Lista Ordenada)

- Elemento < último nó da lista:
 - Percorre a lista até encontrar nó > elemento
 - Colocar um ponteiro auxiliar no nó cabeçalho
 - Avançar ponteiro se campo *info* do **sucessor de aux** for menor que elemento (*aux->prox->info < elem*)
 - Elemento não está na lista

Ex: remover 7

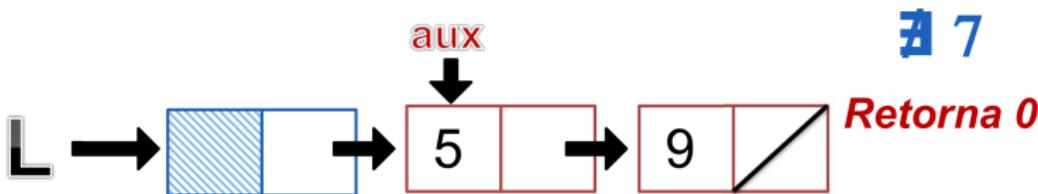




Operação de Remoção (Lista Ordenada)

- Elemento < último nó da lista:
 - Percorre a lista até encontrar nó > elemento
 - Colocar um ponteiro auxiliar no nó cabeçalho
 - Avançar ponteiro se campo *info* do **sucessor de aux** for menor que elemento (*aux->prox->info < elem*)
 - Elemento não está na lista
 - Retorna 0 (**operação falha**)

Ex: remover 7



Operação de Remoção (Lista Ordenada)

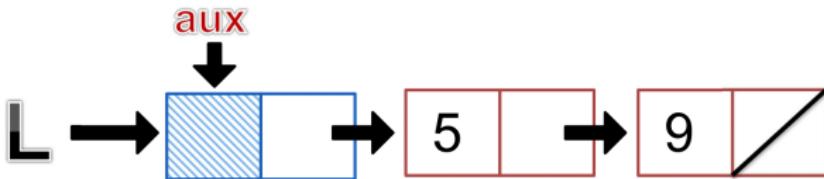
- Elemento > último nó da lista:

Ex: remover 11



Operação de Remoção (Lista Ordenada)

- Elemento > último nó da lista:
 - Percorre a lista até o seu final
 - Faz um ponteiro auxiliar apontar para o nó cabeçalho

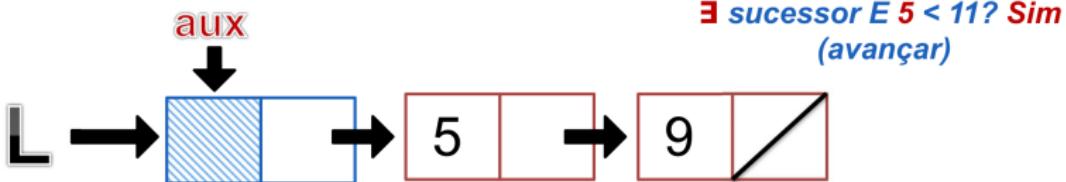




Operação de Remoção (Lista Ordenada)

- Elemento > último nó da lista:
 - Percorre a lista até o seu final
 - Faz um ponteiro auxiliar apontar para o nó cabeçalho
 - Avançar o ponteiro enquanto \exists sucessor
($aux->prox \neq NULL$)

Ex: remover 11

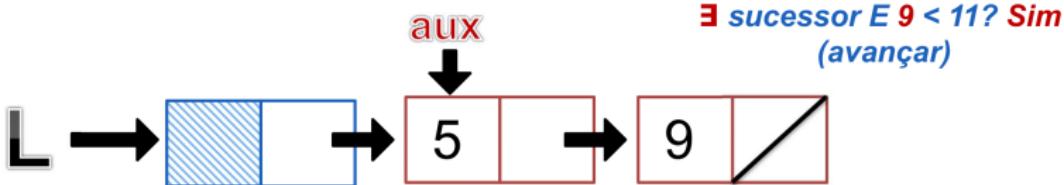




Operação de Remoção (Lista Ordenada)

- Elemento > último nó da lista:
 - Percorre a lista até o seu final
 - Faz um ponteiro auxiliar apontar para o nó cabeçalho
 - Avançar o ponteiro enquanto \exists sucessor
 - ($aux->prox \neq NULL$)

Ex: remover 11

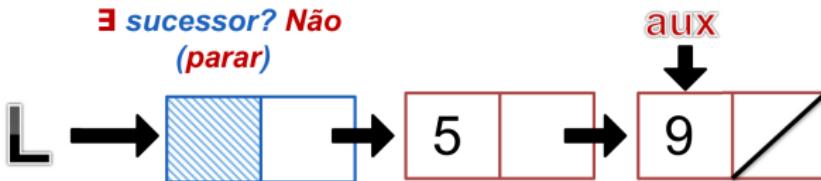


Operação de Remoção (Lista Ordenada)

- Elemento > último nó da lista:
 - Percorre a lista até o seu final
 - Faz um ponteiro auxiliar apontar para o nó cabeçalho
 - Avançar o ponteiro enquanto \exists sucessor
($aux->prox \neq NULL$)

Ex: remover 11

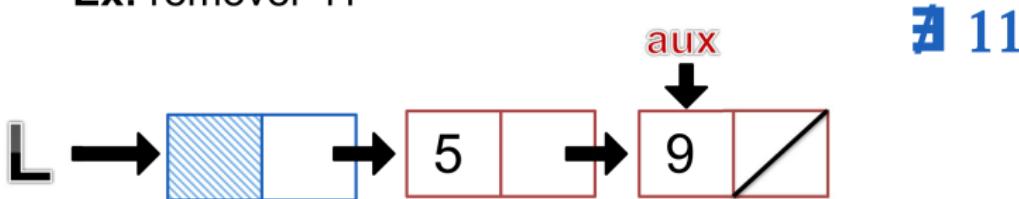
\exists sucessor? Não
(parar)



Operação de Remoção (Lista Ordenada)

- **Elemento > último nó da lista:**
 - Percorre a lista **até o seu final**
 - Faz um ponteiro auxiliar apontar para o nó cabeçalho
 - Avançar o ponteiro enquanto \exists sucessor
(aux->prox ≠ NULL)
 - Elemento não está na lista

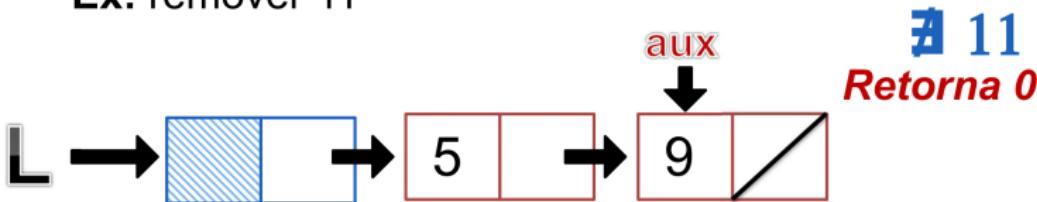
Ex: remover 11



Operação de Remoção (Lista Ordenada)

- Elemento > último nó da lista:
 - Percorre a lista até o seu final
 - Faz um ponteiro auxiliar apontar para o nó cabeçalho
 - Avançar o ponteiro enquanto \exists sucessor
($aux->prox \neq NULL$)
 - Elemento não está na lista
 - Retorna 0 (operação falha)

Ex: remover 11





Operação de Remoção (Lista Ordenada)

- Implementação em C:

```
int remove_ord (Lista *lst, int elem) {
    if (lista_vazia(lst) == 1 || elem < (*lst)->info)
        return 0; // Falha REMOVER
    Lista aux = *lst; // Ponteiro auxiliar para nó cabeçalho
    // Trata elemento = 1º nó da lista
    if (elem == (*lst)->info) {
        *lst = aux->prox; // Lista aponta para o 2º nó
        free(aux); // Libera memória alocada
        return 1;
    }
    ...
    REMOVER
```



Operação de Remoção (Lista Ordenada)

- **Implementação em C:**

```
...
    // Percorimento até final de lista, achar elem ou nó maior
    while (aux->prox != NULL && aux->prox->info < elem)
        aux = aux->prox;
    if (aux->prox == NULL || aux->prox->info > elem)
        return 0; // Falha
    // Remove elemento da lista
    Lista aux2 = aux->prox; // Aponta nó a ser removido
    aux->prox = aux2->prox; // Retira nó da lista
    free(aux2); // Libera memória alocada
    (*lst)->info--; // Opcional: Decrementa qtde de nós na lista
    return 1; }
```



Referências

✓ Básica

- CELES, W., CERQUEIRA, R. e RANGEL, J. L. "Introdução a estruturas de dados". Campus Elsevier, 2004.
- TENENBAUM, A. M., LANGSAM, Y. e AUGENSTEIN, M.J. "Estrutura de Dados Usando C". Makron Books.

✓ Extra

- BACKES, André. "Programação Descomplicada Linguagem C". Projeto de extensão que disponibiliza vídeo-aulas de C e Estruturas de Dados. Disponível em: <https://www.youtube.com/user/progdescomplicada>. Acessado em: 25/04/2022.

✓ Baseado nos materiais dos seguintes professores:

- Prof. André Backes (UFU)
- Prof. Bruno Travençolo (UFU)
- Prof. Luiz Gustavo de Almeida Martins (UFU)



Dúvidas?

Prof. Me. Claudiney R. Tinoco
profclaudineytinoco@gmail.com

Faculdade de Computação (FACOM)
Universidade Federal de Uberlândia (UFU)