Aula 14: Manipulação de Listas Simplesmente Encadeadas

- Algoritmo de inserção
- Algoritmo de remoção
- Complexidade dos métodos

Inserção de um nó em lista simplesmente encadeada

É realizada em 3 fases:

1) Solicitação à LED de um novo nó

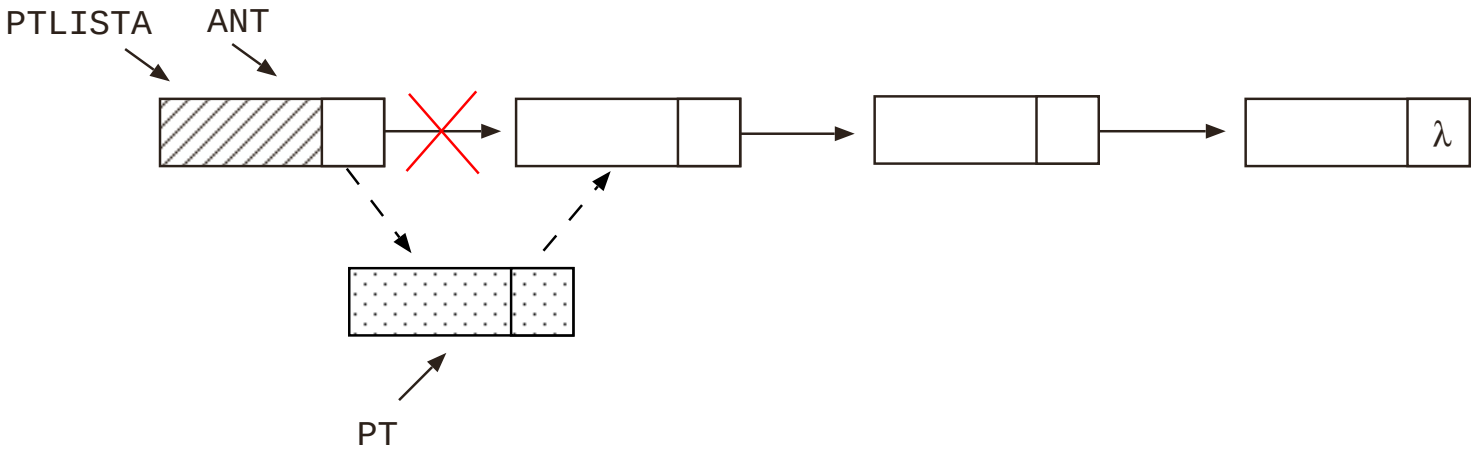
2) Inicialização do nó

3) Inserção do nó na lista, com o acerto dos ponteiros na estrutura

Inserção de um nó em lista simplesmente encadeada

A inserção do novo nó é feita após o nó apontado por ANT. São 3 situações:

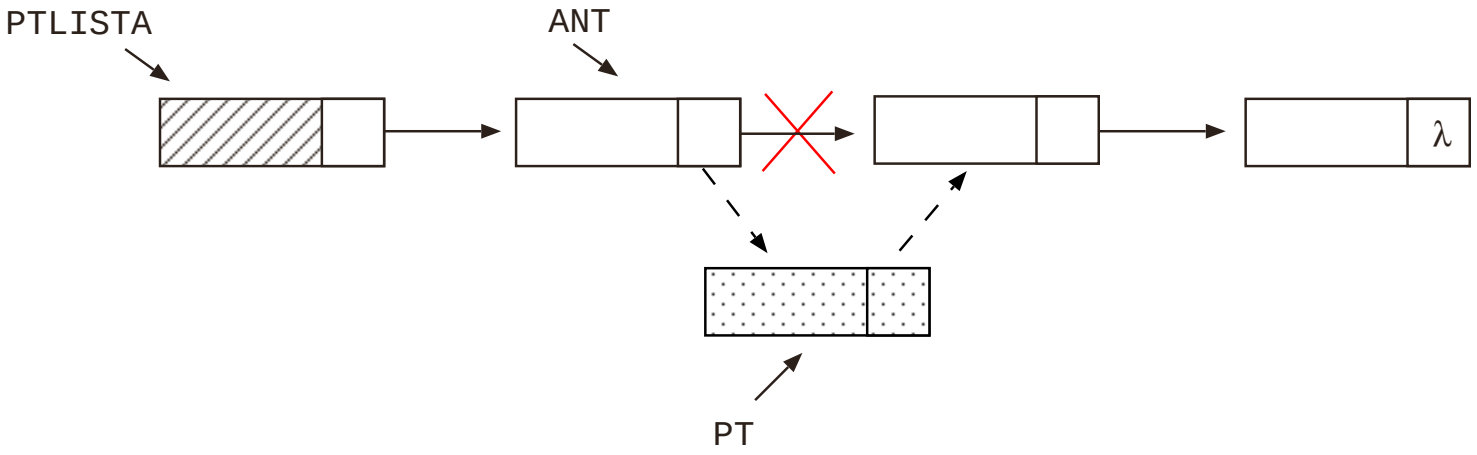
1) Inserção no início da lista



Inserir Voltar

Inserção de um nó em lista simplesmente encadeada

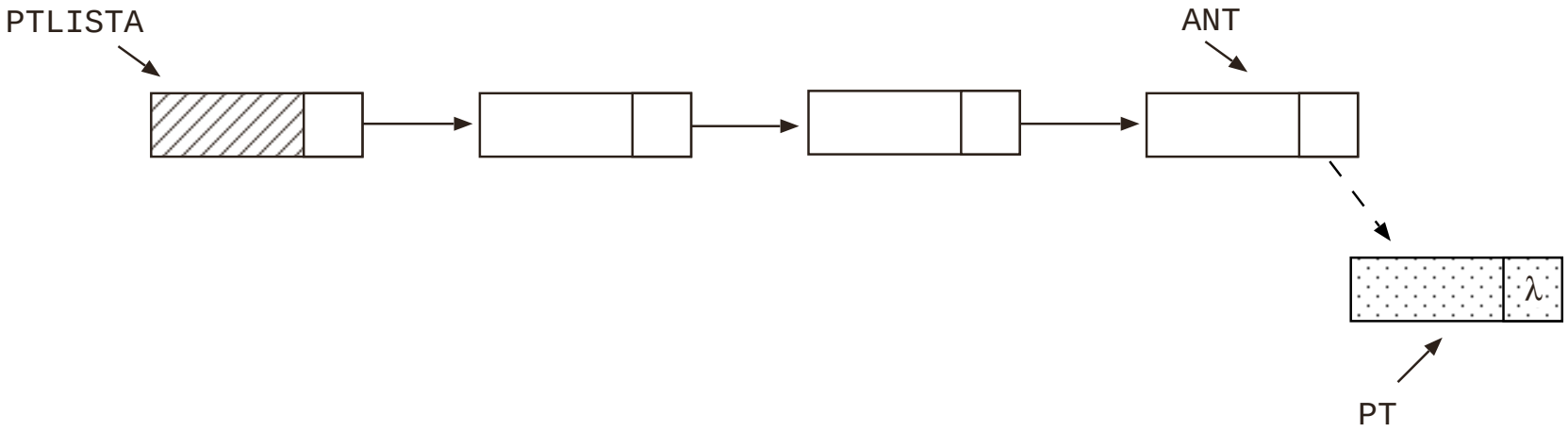
2) Inserção no meio da lista



Inserir Voltar

Inserção de um nó em lista simplesmente encadeada

3) Inserção no final da lista



Inserir Voltar

Descrição do algoritmo de inserção de um nó em lista simplesmente encadeada

Algoritmo: Inserção de um nó em lista simplesmente encadeada após o nó apontado por ANT.

```
(*) BUSCA_ENC_ORD( X, ANT, PONT )      % buscar o nó com chave X
    se PONT ≠ λ
    então "elemento já existe na lista"
    senão ocupar( PT )                  % solicitar novo nó à LED
       PT↑.info := NOVO_VALOR     %
       PT↑.chave := X                  %
       PT↑.prox := ANT.prox          %
       ANT↑.prox := PT                 % acertar a lista
```

(*) Obs: O algoritmo acima é para listas ordenadas. Caso a lista não seja ordenada, basta executar BUSCA_ENC_NÃO_ORD(X, ANT, PONT) na linha marcada com (*)
(veja aula anterior)

Complexidade da inserção em lista simplesmente encadeada

- A complexidade da inserção depende da complexidade da busca, já que as três fases da inserção podem ser executadas em tempo constante. Portanto, a complexidade é O(n), onde n é o número de nós da lista.

Remoção de um nó em lista simplesmente encadeada

É realizada em 3 fases:

1) Remoção do nó da lista, com o acerto dos ponteiros na estrutura

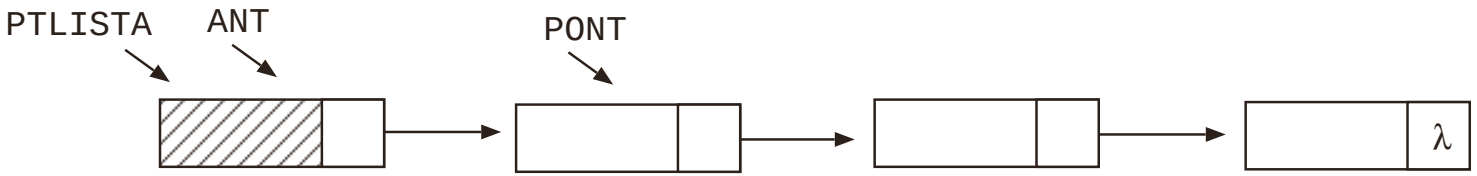
2) Utilização da informação contida no nó

3) Devolução do nó removido à LED

Remoção de um nó em lista simplesmente encadeada

Será removido o nó apontado por PONT.
São 3 situações:

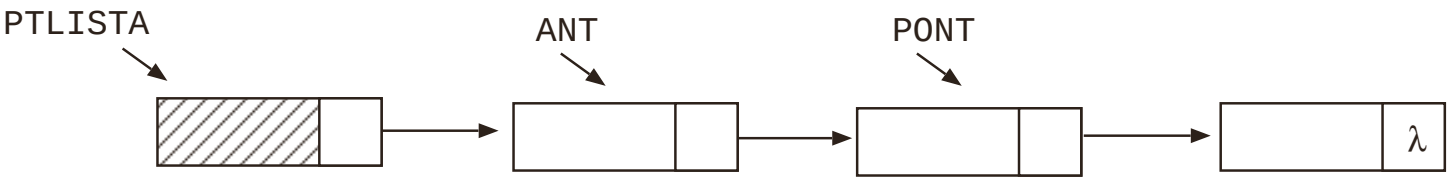
1) Remoção do início da lista



Remover Voltar

Remoção de um nó em lista simplesmente encadeada

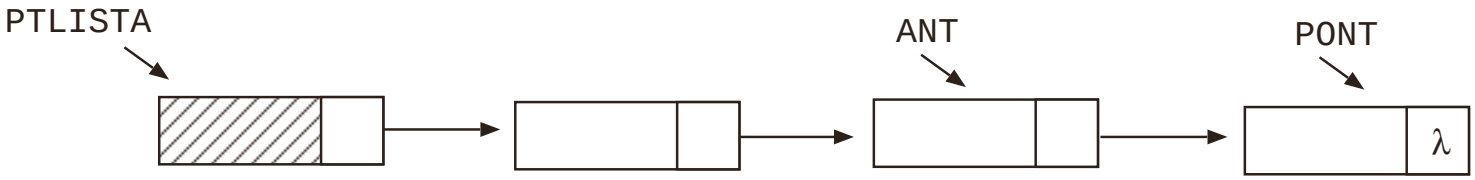
2) Remoção no meio da lista



Remover Voltar

Remoção de um nó em lista simplesmente encadeada

3) Remoção no final da lista



Remover Voltar

Descrição do algoritmo de remoção de um nó em lista simplesmente encadeada

Algoritmo: Remoção do nó apontado por PONT de uma lista simplesmente encadeada.

```
(*) BUSCA_ENC_ORD( X, ANT, PONT )          % buscar o nó com chave X
      se PONT = λ
      então "nó não se encontra na lista"
      senão
         ANT↑.prox := PONT↑.prox                    % acertar lista
         VALOR_RECUPERADO := PONT↑.info   % utilizar informação do nó
         DESOCUPAR( PONT )                          % devolver nó à LED
```

(*) Obs: O algoritmo acima é para listas ordenadas. Caso a lista não seja ordenada, basta executar BUSCA_ENC_NÃO_ORD(X, ANT, PONT) na linha marcada com (*)
(veja aula anterior)

Complexidade da remoção em lista simplesmente encadeada

Como no caso da inserção, a complexidade da remoção depende da complexidade da busca, já que as três fases da remoção podem ser executadas em tempo constante. Portanto, a complexidade é O(n), onde n é o número de nós da lista.

Exercício

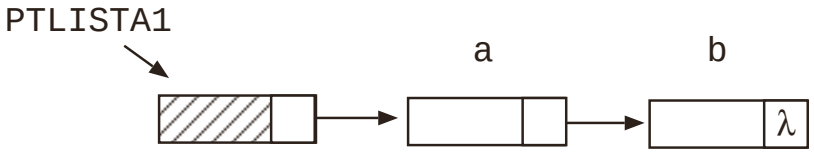
(tempo: 15 minutos)

- Sejam duas listas, não ordenadas, simplesmente encadeadas com nó cabeça. Apresentar um algoritmo que intercale as duas listas (intercalando os elementos de uma e outra enquanto possível, e incluindo no final os elementos que restaram da lista mais longa).

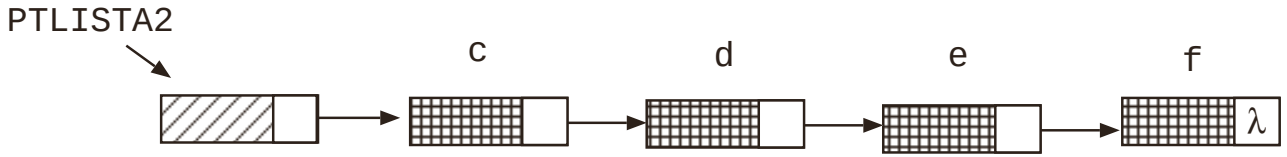
Exercício

Exemplo:

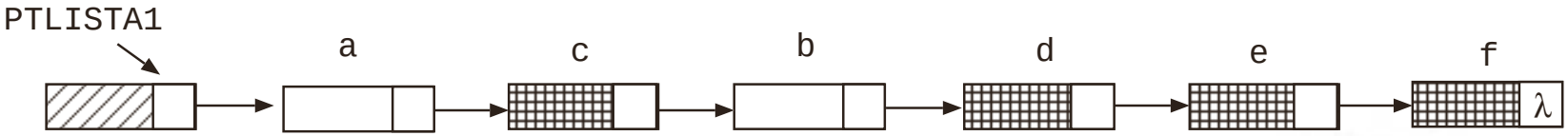
- Lista 1



- Lista 2



- Lista Intercalada



Solução

```
PONT1 := PTLISTA1↑.prox    % ponteiro para LISTA 1
PONT2 := PTLISTA2↑.prox    % ponteiro para LISTA 2
I := 1                              % variável para alternar entre as listas

repita
   se I é ímpar
      então                         % nó da LISTA 1 aponta para nó da LISTA 2
         se PONT1 ≠ λ
            então
               PTAUX := PONT1↑.prox
               PONT1↑.prox := PONT2
               PONT1 := PTAUX
      senão                         % nó da LISTA 2 aponta para nó da LISTA 1
         se PONT2 ≠ λ
            então
               PTAUX := PONT2↑.prox
               PONT2↑.prox := PONT1
               PONT2 := PTAUX
   I := I + 1
até que ( PONT1 = λ ) ou ( PONT2 = λ )
```

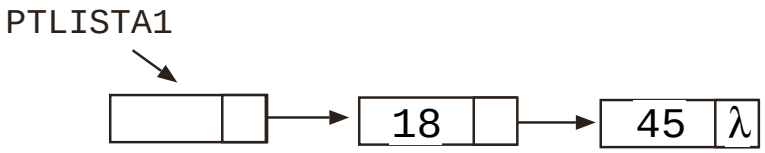
Exercício Final

- Sejam duas listas, ordenadas, simplesmente encadeadas com nó cabeça. Apresentar um algoritmo que intercale as duas listas de forma que a lista resultante esteja também ordenada.

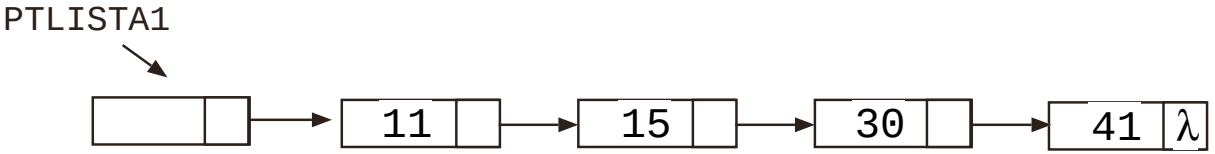
Exercício Final

Exemplo:

- LISTA1



- LISTA2



- LISTA INTERCALADA