Estrutura de Dados I

Profa. Suzana Matos França de Oliveira
suzana.matos@frn.uespi.br

Pilhas e filas com alocação encadeada

Pilhas e filas com alocação encadeada

- Algumas modificações podem ser feitas as listas lineares com alocação simplesmente encadeada para serem mais eficientes com pilhas e filas

Pilhas e filas com alocação encadeada

- Algoritmos vistos em sala:
  - Pilha
    - Empilhar
    - Desempilhar
  - Fila
    - Enfileirar
    - Desenfileirar

Lembrar que tem que implementar funções de criar, imprimir e de liberar o espaço

Serão vistas as opções sem nó cabeça

Aplicação: ordenação por distribuição

Aplicação: ordenação por distribuição

- Seja uma lista L composta de n chaves, cada qual representada por um número inteiro numa base b > 1
  - Problema: Ordenar L, utilizando b filas, F_k, onde $0 \leq k < b$

Aplicação: ordenação por distribuição

- Seja uma lista L composta de n chaves, cada qual representada por um número inteiro numa base b > 1
  - Problema: Ordenar L, utilizando b filas, F_k, onde $0 \leq k < b$
  - Resolução:
    - 1ª iteração: Destaca, em cada nó, o dígito menos significativo da representação b-ária de cada chave.
      - Se este for igual a k, a chave correspondente será inserida na fila F_k
      - Concatenar as filas F_0, F_1, … F_{b-1}

Tabela: 19 13 05 27 01 26 31 16 02 09 11 21 60 07

Iteração 1: 1ª distribuição (unidades simples)

$fila_0$:
$fila_1$:
$fila_2$:
$fila_3$:
$fila_4$:
$fila_5$:
$fila_6$:
$fila_7$:
$fila_8$:
$fila_9$:

Tabela:

Valores distribuídos em uma ordem diferente

Ex: b = 10

Aplicação: ordenação por distribuição

- Seja uma lista L composta de n chaves, cada qual representada por um número inteiro numa base b > 1
  - Problema: Ordenar L, utilizando b filas, F_k, onde $0 \leq k < b$
  - Resolução:
    - 1ª iteração: Destaca, em cada nó, o dígito menos significativo da representação b-ária de cada chave.
      - Se este for igual a k, a chave correspondente será inserida na fila F_k
      - Concatenar as filas F_0, F_1, … F_{b-1}
    - 2ª iteração: O processo deve ser repetido com o segundo dígito

Tabela: 60 01 31 11 21 02 13 05 26 16 07 27 19 09

Iteração 2: 2ª distribuição (dezenas simples)

fila$_0$:
fila$_1$:
fila$_2$:
fila$_3$:
fila$_4$:
fila$_5$:
fila$_6$:
fila$_7$:
fila$_8$:
fila$_9$:

Tabela:

Aplicação: ordenação por distribuição

- Seja uma lista L composta de n chaves, cada qual representada por um número inteiro numa base b > 1
  - Problema: Ordenar L, utilizando b filas, F_k, onde $0 \leq k < b$
  - Resolução:
    - 1ª iteração: Destaca, em cada nó, o dígito menos significativo da representação b-ária de cada chave.
      - Se este for igual a k, a chave correspondente será inserida na fila F_k
      - Concatenar as filas F_0, F_1, … F_{b-1}
    - 2ª iteração: O processo deve ser repetido com o segundo dígito

Tabela: 60 01 31 11 21 02 13 05 26 16 07 27 19 09

Iteração 2: 2ª distribuição (dezenas simples)

fila$_0$:
fila$_1$:
fila$_2$:
fila$_3$:
fila$_4$:
fila$_5$:
fila$_6$:
fila$_7$:
fila$_8$:
fila$_9$:

Tabela:

Se tiver d dígitos, é preciso de d iterações

Aplicação: ordenação por distribuição

- Seja uma lista L composta de n chaves, cada qual representada por um número inteiro numa base b > 1
  - Problema: Ordenar L, utilizando b filas, F_k, onde $0 \leq k < b$
  - Resolução:
    - 1ª iteração: Destaca, em cada nó, o dígito menos significativo da representação b-ária de cada chave.
      - Se este for igual a k, a chave correspondente será inserida na fila F_k
      - Concatenar as filas F_0, F_1, … F_{b-1}
    - 2ª iteração: O processo deve ser repetido com o segundo dígito
    - ...
    - d-ésima iteração, onde d é o comprimento máximo da representação das chaves na base b:
      - Repetir o processo para o d-ésimo dígito

No exemplo,
b = ?
d = ?

Aplicação: ordenação por distribuição

- Ideia do algoritmo
  - Receber os n itens a serem ordenados
  - Inicializar as b filas, F_0, F_1, … F_{b-1}
    - Dica: pode ter uma fila “principal”,
      que tem os dados iniciais,
      e os dados ao fim de cada iteração
  - Para cada dígito i (total d)
    - Para cada item j (total n)
      - Seja k o i-ésimo dígito menos significativo
      - O item L_j deve ser inserido na fila F_k
    - Ao final, concatenar todas as filas para analisar o próximo dígito

Aplicação: ordenação por distribuição

- Ideia do algoritmo
  - Receber os n itens a serem ordenados
  - Inicializar as b filas, F_0, F_1, … F_{b-1}
    - Dica: pode ter uma fila “principal”, que tem os dados iniciais, e os dados ao fim de cada iteração
  - Para cada dígito i (total d)
    - Para cada item j (total n)
      - Seja k o i-ésimo dígito menos significativo
      - O item L_j deve ser inserido na fila F_k
    - Ao final, concatenar todas as filas para analisar o próximo dígito

Ideia utilizada por classificadora de cartões que efetuava a ordenação física de cartões a partir de chaves representadas por perfurações localizadas em 12 alturas diferentes no cartão.

Aplicação: ordenação por distribuição

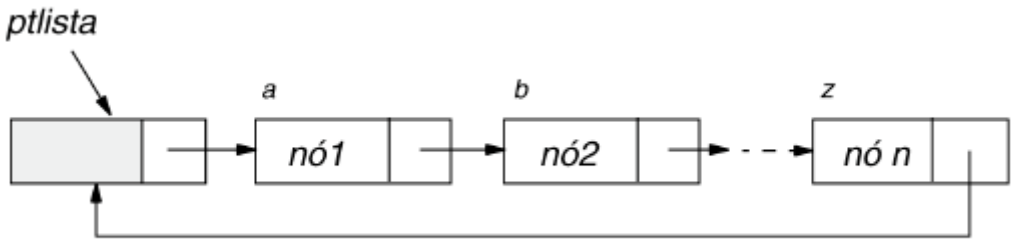
- Análise de complexidade
  - Ao utilizar uma fila encadeada, qualquer inclusão, remoção ou concatenação é O(1)
  - Depende de dois laços aninhados que dependem de d e n respectivamente

Listas circulares

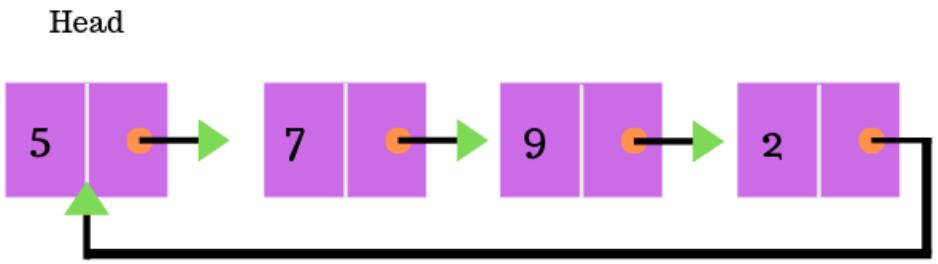
Listas circulares

- A lista circular encadeada tem que o último nó aponte para o início da lista

Com nó cabeça



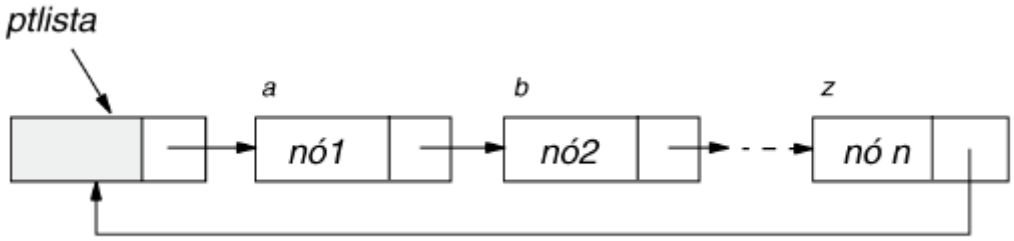
Sem nó cabeça



Listas circulares

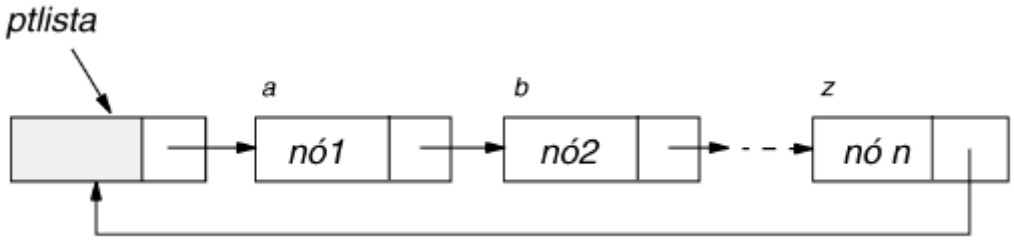
- O teste de final de lista nunca é satisfeito
  - Contudo a chave buscada pode ser colocada na cabeça e usada para sempre ter uma resposta positiva

Similar a busca alocação sequencial



Listas circulares

- O teste de final de lista nunca é satisfeito
  - Contudo a chave buscada pode ser colocada na cabeça e usada para sempre ter uma resposta positiva
- Alterações na inserção e remoção devem ser feitas



Listas circulares

- Implementação:
  - Criando nova lista

```
No *novaLista()
{
    No *ptLista = (No *)malloc(sizeof(No));
    if(ptLista == NULL) {
        printf("Erro ao alocar memória.\n");
        exit(EXIT_FAILURE);
    }

    // ptLista->info = 0; //o primeiro nó é ignorado
    ptLista->prox = ???
    return ptLista;
}
```

Retorna o nó cabeça

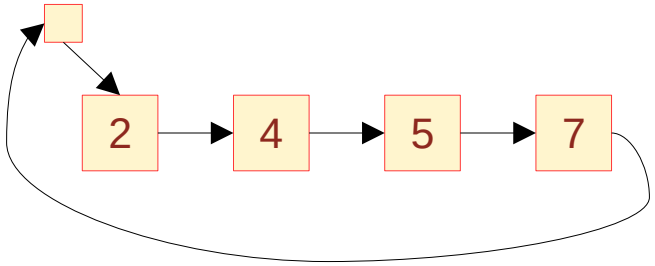
Listas circulares

- Implementação:
  - Criando nova lista

```
No *novaLista()
{
    No *ptLista = (No *)malloc(sizeof(No));
    if(ptLista == NULL) {
        printf("Erro ao alocar memória.\n");
        exit(EXIT_FAILURE);
    }

    // ptLista->info = 0; //o primeiro nó é ignorado
    ptLista->prox =     ???
    return ptLista;
}
```

Listas circulares

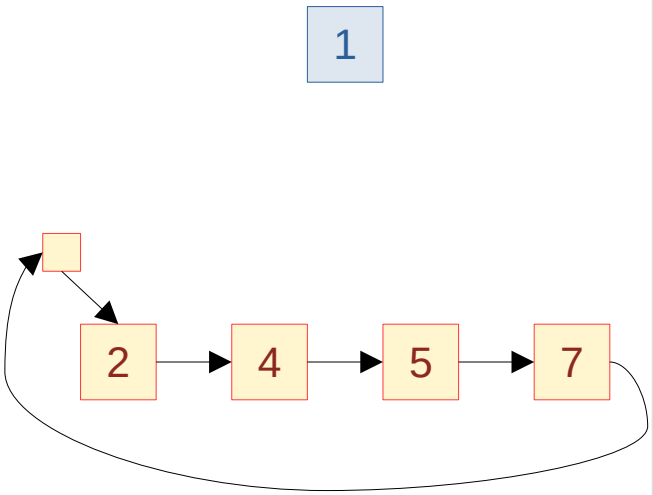
- Implementação:
  - Busca




```c
//retorna o nó anterior ao procurado, seja ele encontrado ou não
No *buscarOrdenado(Item x, No *ptLista)
{
    if(ptLista == NULL) {
        printf("Lista inexistente.\n");
        exit(EXIT_FAILURE);
    }

    ptLista->info = x;
    No *ant = ptLista;
    No *pt = ptLista->prox;

    while(pt->info < x)
    {
        ant = pt;
        pt = pt->prox;
    }

    return ant;
}
```

Listas circulares

- Implementação:
  - Inserção

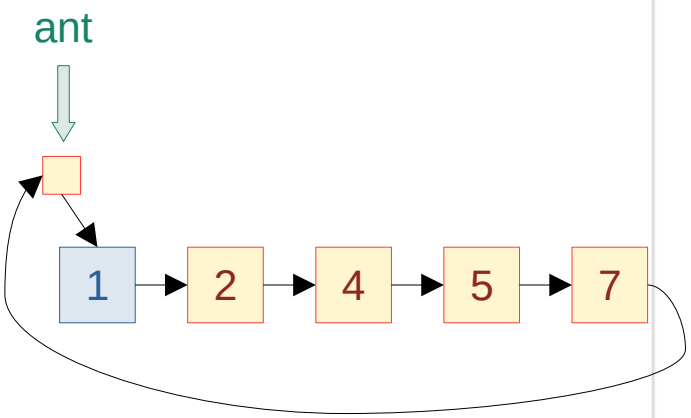



```
bool inserirOrdenado(Item x, No *ptLista)
{
    No *ant = buscarOrdenado(x, ptLista);

    //o próximo sempre existirá, o problema é se o próximo for a cabeça
    //ou se tiver já achado
    if(ant->prox != ptLista && ant->prox->info == x)
    {
        printf("Item %d já existe na lista\n", x);
        return false;
    }
    //caso contrário, inserir na próxima posição
    No *novo = novoNo(x);
    novo->prox = ant->prox; //para não perder o encadeamento
    ant->prox = novo;

    return true;
}
```

A função novoNo() é similar a novaLista()

Listas circulares

- Implementação:
  - Inserção

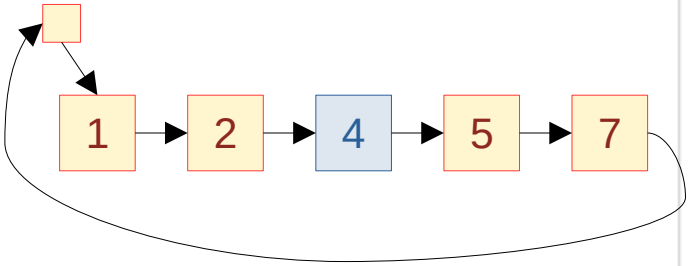


E se fosse 4?
e 8?

```
bool inserirOrdenado(Item x, No *ptLista)
{
    No *ant = buscarOrdenado(x, ptLista);

    //o próximo sempre existirá, o problema é se o próximo for a cabeça
    //ou se tiver já achado
    if(ant->prox != ptLista && ant->prox->info == x)
    {
        printf("Item %d já existe na lista\n", x);
        return false;
    }
    //caso contrário, inserir na próxima posição
    No *novo = novoNo(x);
    novo->prox = ant->prox; //para não perder o encadeamento
    ant->prox = novo;

    return true;
}
```

Listas circulares

- Implementação:
  - Remoção

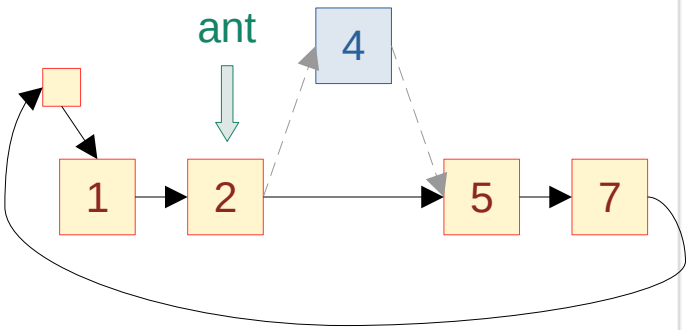



```
bool removerOrdenado(Item x, No *ptLista)
{
    No *ant = buscarOrdenado(x, ptLista);

    //se não for encontrado, não remove
    //se buscou até o final da lista achando só a cabeça, não remove
    if(ant->prox == ptLista || ant->prox->info != x)
    {
        printf("Item %d não existe na lista\n", x);
        return false;
    }
    //caso contrário, remover da próxima posição
    No *remover = ant->prox;
    ant->prox = remover->prox; //para não perder o encadeamento
    free(remover);

    return true;
}
```

Basta uma ser verdade

Listas circulares

- Implementação:
  - Remoção



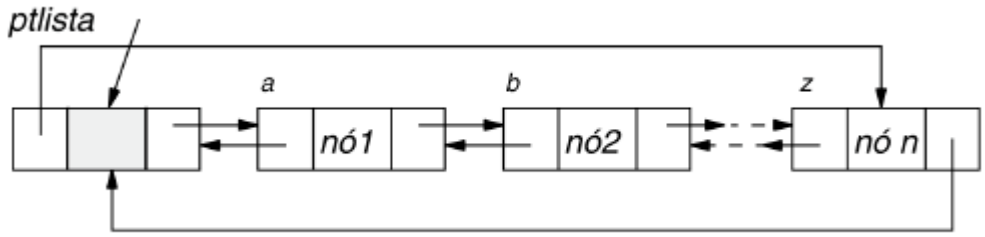

```
bool removerOrdenado(Item x, No *ptLista)
{
    No *ant = buscarOrdenado(x, ptLista);

    //se não for encontrado, não remove
    //se buscou até o final da lista achando só a cabeça, não remove
    if(ant->prox == ptLista || ant->prox->info != x)
    {
        printf("Item %d não existe na lista\n", x);
        return false;
    }
    //caso contrário, remover da próxima posição
    No *remover = ant->prox;
    ant->prox = remover->prox; //para não perder o encadeamento
    free(remover);

    return true;
}
```

Listas duplamente encadeadas circulares

Listas duplamente encadeadas circulares

- A lista duplamente encadeada tem ponteiros para o próximo e para o anterior

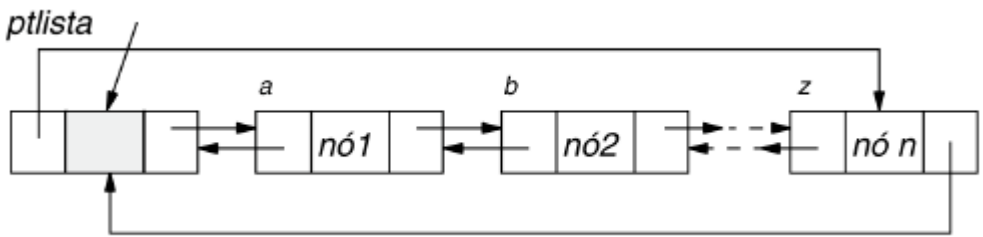
Pode ter ou não cabeça



Pode ser duplamente encadeada sem ser circular

Listas duplamente encadeadas circulares

- A lista duplamente encadeada tem ponteiros para o próximo e para o anterior
  - O gasto de memória com o novo campo de ponteiro pode ser justificado pela economia em não reprocessar praticamente a lista inteira na busca.

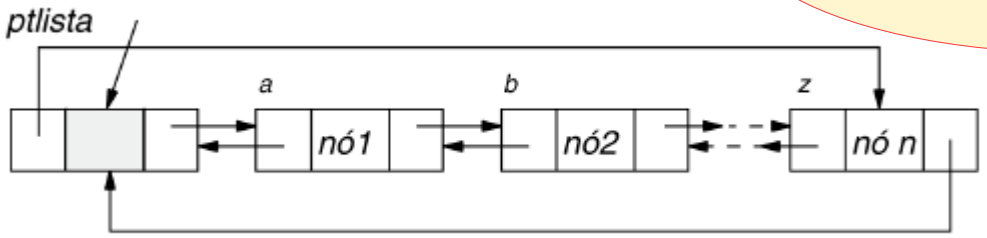


Ex: o ultimo elemento dá lista é 10, adianta procurar o 11?

Listas duplamente encadeadas circulares

- A lista duplamente encadeada tem ponteiros para o próximo e para o anterior
  - O gasto de memória com o novo campo de ponteiro pode ser justificado pela economia em não reprocessar praticamente a lista inteira na busca.

Qual a nova estrutura para nó?



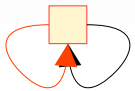
Listas duplamente encadeadas circulares

- Implementação:
  - Criando nova lista

```
No *novaLista()
{
    No *ptLista = (No *)malloc(sizeof(No));
    if(ptLista == NULL) {
        printf("Erro ao alocar memória.\n");
        exit(EXIT_FAILURE);
    }

    // ptLista->info = 0; //o primeiro nó é ignorado
    ???
    return ptLista;
}
```

Listas duplamente encadeadas circulares

- Implementação:
  - Criando nova lista

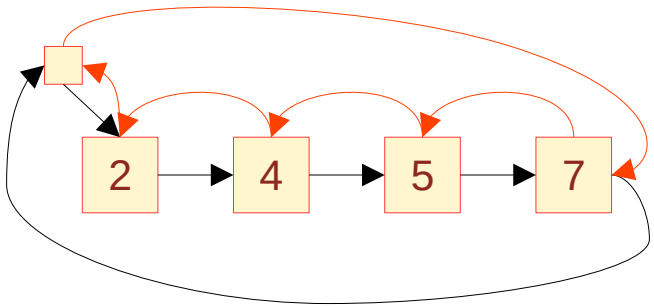


```
No *novaLista()
{
    No *ptLista = (No *)malloc(sizeof(No));
    if(ptLista == NULL) {
        printf("Erro ao alocar memória.\n");
        exit(EXIT_FAILURE);
    }

    // ptLista->info = 0; //o primeiro nó é ignorado
    ???
    return ptLista;
}
```

Listas duplamente encadeadas circulares

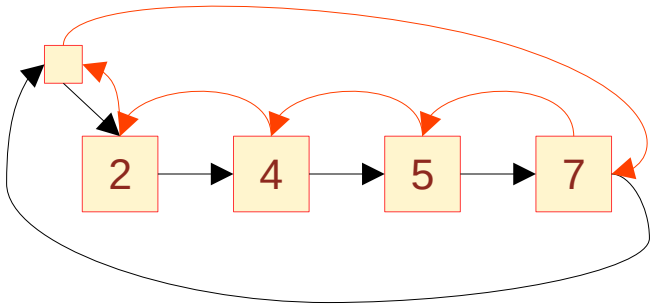
- Implementação:
  - Busca: Não precisa retornar o ponteiro anterior

Por quê?



Listas duplamente encadeadas circulares

- Implementação:
  - Busca




```
//retorna o nó procurado ou o posterior a ele (caso não exista)
No *buscarOrdenado(Item x, No *ptLista)
{
    if(ptLista == NULL) {
        printf("Lista inexistente.\n");
        exit(EXIT_FAILURE);
    }

    ???

    ptLista->info = x;
    No *pt = ptLista->prox;
    while(pt->info < x)
    {
        pt = pt->prox;
    }

    return pt;
}
```

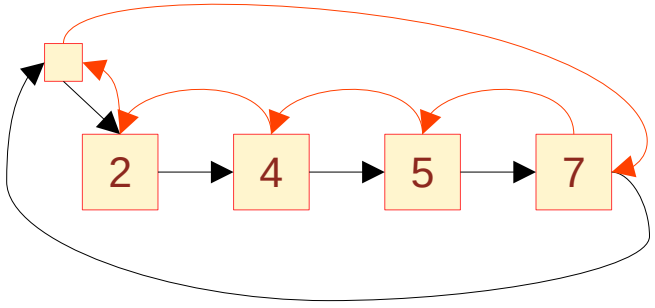
O que fazer para verificar se precisa mesmo fazer a busca?

Sem guardar o ant

Listas duplamente encadeadas circulares

- Implementação:
  - Inserção

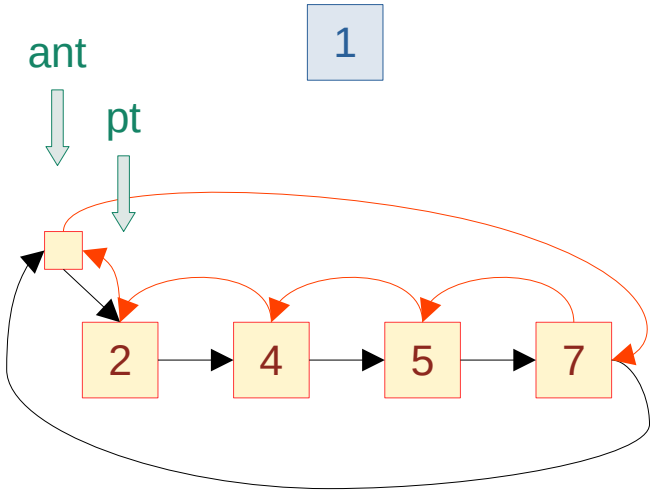
1




```
bool inserirOrdenado(Item x, No *ptLista)
{
    No *pt = buscarOrdenado(x, ptLista);

    //se o nó já existir na lista, não insere
    if(pt != ptLista && pt->info == x)
    {
        printf("Item %d já existe na lista\n", x);
        return false;
    }
    //caso contrário, inserir na próxima posição
    No *novo = novoNo(x);
    No *ant = pt->ant;

    //atualizando os ponteiros para não perder o encadeamento
    ???

    return true;
}
```

Listas duplamente encadeadas circulares

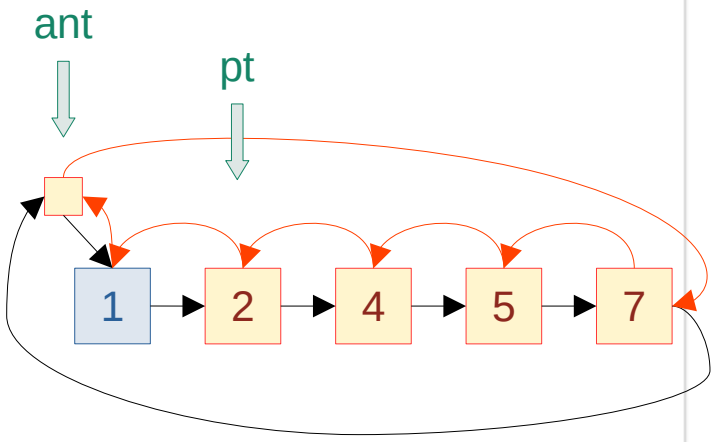
- Implementação:
  - Inserção




```
bool inserirOrdenado(Item x, No *ptLista)
{
    No *pt = buscarOrdenado(x, ptLista);

    //se o nó já existir na lista, não insere
    if(pt != ptLista && pt->info == x)
    {
        printf("Item %d já existe na lista\n", x);
        return false;
    }
    //caso contrário, inserir na próxima posição
    No *novo = novoNo(x);
    No *ant = pt->ant;

    //atualizando os ponteiros para não perder o encadeamento
    ???

    return true;
}
```

Listas duplamente encadeadas circulares

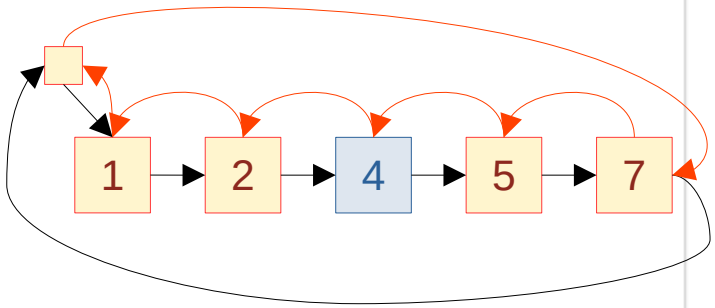
- Implementação:
  - Inserção




```
bool inserirOrdenado(Item x, No *ptLista)
{
    No *pt = buscarOrdenado(x, ptLista);

    //se o nó já existir na lista, não insere
    if(pt != ptLista && pt->info == x)
    {
        printf("Item %d já existe na lista\n", x);
        return false;
    }
    //caso contrário, inserir na próxima posição
    No *novo = novoNo(x);
    No *ant = pt->ant;

    //atualizando os ponteiros para não perder o encadeamento
    ???

    return true;
}
```

Listas duplamente encadeadas circulares

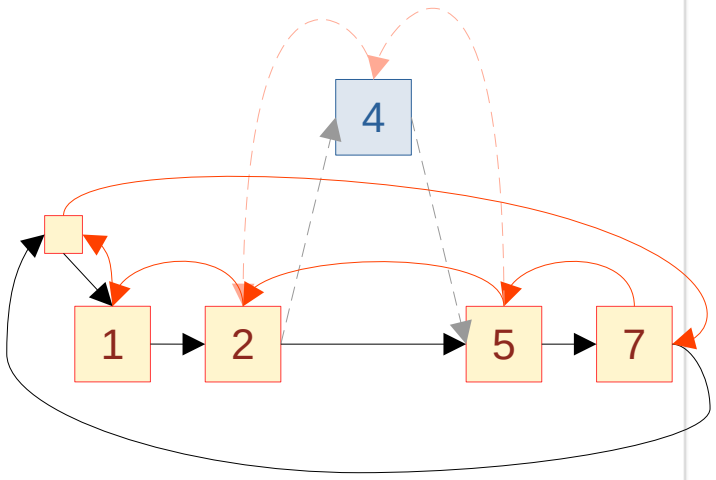
- Implementação:
  - Remoção




```
bool removerOrdenado(Item x, No *ptLista)
{
    No *pt = buscarOrdenado(x, ptLista);

    //se não for encontrado, não remove
    //se buscou até o final da lista achando só a cabeça, não remove
    if(pt == ptLista || pt->info != x)
    {
        printf("Item %d não existe na lista\n", x);
        return false;
    }
    //caso contrário, remover da próxima posição

    ???

    free(pt);

    return true;
}
```

Listas duplamente encadeadas circulares

- Implementação:
  - Remoção




```
bool removerOrdenado(Item x, No *ptLista)
{
    No *pt = buscarOrdenado(x, ptLista);

    //se não for encontrado, não remove
    //se buscou até o final da lista achando só a cabeça, não remove
    if(pt == ptLista || pt->info != x)
    {
        printf("Item %d não existe na lista\n", x);
        return false;
    }
    //caso contrário, remover da próxima posição

                    ???

    free(pt);

    return true;
}
```

Listas duplamente encadeadas circulares

- Exercício: concatenação de duas listas

Entrada?

Processo?

Podem assumir duas listas sem cabeça