Lista Duplamente Encadeada

Profa. Divani Barbosa Gavinier
Aula 11 – Estrutura de Dados

Lista Duplamente Encadeada

Cada elemento possui dois elos que apontam para elemento anterior e próximo, respectivamente.

Em java o elemento é representado através de uma classe chamada Nó.

Classe Nó:

```
class No {
   public double item;
   public No prox;
   public No ant;
}
```

espaço para armazenamento da informação

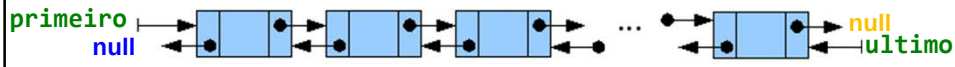
espaço para armazenar uma referência da localização na memória (elo) onde o próximo Nó se encontra.

elo para o item anterior

2

Lista Duplamente Encadeada

O inicio é identificado com um Nó (**primeiro**) apontando para o Nó inicial.
O inicio também pode ser identificado como o Nó cujo o elo ant aponta para **null**

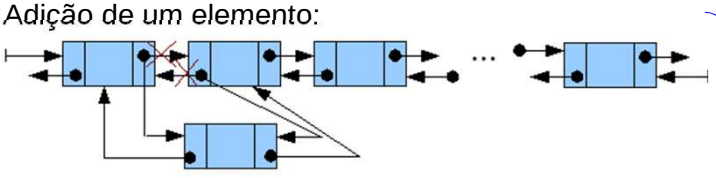


O fim é identificado com um Nó (ultimo) apontando para o Nó final.
O fim também pode ser identificado como o Nó cujo o elo prox aponta para **null**

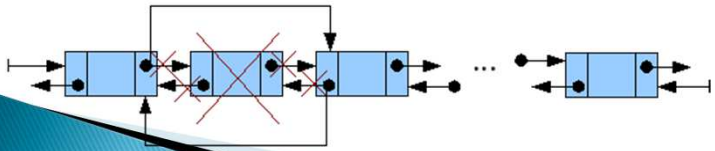
3

Lista Duplamente Encadeada

Adição de um elemento:



Remoção de um elemento:



Somente dois elos serão alterados. O restante permanece intacto.

Classe ListaDE:

Campos

```
private No primeiro;
private No ultimo;
```

Construtor

```
public ListaDE() {
    primeiro = null;
    ultimo = null;
}
```

Métodos

vazio: Retorna verdadeiro se lista vazia.
insere: Insere itens na lista.
insere_fim: Insere itens no final da lista.
pesquisa: Retorna verdadeiro se item encontrado.
imprime_inicio: Imprime lista toda do inicio ao fim.
imprime_fim: Imprime lista toda do fim ao inicio.
apaga: Remove itens.
item_frente: Retorna o primeiro item
item_fim: Retorna o ultimo item

VAZIO

Entrada: Nenhuma
Saída: verdadeiro ou falso

```
public boolean vazio() { return (primeiro==null); }
```

ITEM_FRENTE

Entrada: Nenhuma
Saída: item da frente

```
public double item_frente() { return primeiro.item; }
```

ITEM_FIM

Entrada: Nenhuma
Saída: item do fim

```
public double item_fim() { return ultimo.item; }
```

INSERE_FIM

Entrada: Valor a ser inserido
Saída: não há

```
public void insere_fim(double valor) {
  No novo = new No();
  novo.item = valor;
  novo.prox = null;
  if (vazio()) {
    novo.ant = null;
    primeiro = novo;
  }
  else {
    novo.ant = ultimo;
    ultimo.prox = novo;
  }
  ultimo = novo;
}
```

Alocando memória para o novo Nó que se chama novo

Se lista vazia, primeiro recebe o Nó novo

Se lista não vazia, o ultimo elo recebe o Nó novo

Ultimo recebe novo Nó

PESQUISA

Entrada: não há
Saída: verdadeiro ou falso

```
public boolean pesquisa(double chave) {

 No atual = primeiro; // Posicionando no inicio da
                      // lista
 while (atual != null) { // Enquanto não chegou no
                         // fim da lista
    if (atual.item == chave)
         return true; // se encontrou sai com true
    atual = atual.prox; // Caminha um Nó na lista
 }

 return false; // se caminhou até o fim e não
               // encontrou sai com false
}
```

IMPRIME_INICIO

Entrada: não há
Saída: não há

```
public void imprime_inicio() {
   No atual = primeiro;
   while (atual != null) { // Caminha inicio ao fim
     System.out.print("{" + atual.item + "} ");
     atual = atual.prox;
   }
   System.out.println("");
 }
```

IMPRIME_FIM

Entrada: não há
Saída: não há

```
public void imprime_fim() {
   No atual = ultimo;
   while (atual != null) {// Caminha do fim ao inicio
     System.out.print("{" + atual.item + "} ");
     atual = atual.ant;
   }
   System.out.println("");
 }
```

APAGA

Entrada: item a ser removido
Saída: não há

```
public void apaga(double valor) {
  No atual, anterior, proximo;
  anterior = null; atual = primeiro; proximo = atual.prox;
  while (atual != null) {
      if (atual.item == valor) break;
      anterior = atual;
      atual = atual.prox;
      if (atual != null) proximo = atual.prox;
  }
  if (atual == null) return;
  if (atual == primeiro && atual == ultimo) primeiro = ultimo = null;
  else if (atual == primeiro) { primeiro = primeiro.prox;
                                primeiro.ant = null;
       }
       else if (atual.prox == null) { ultimo =  anterior;
                                      ultimo.prox = null;
            }
            else { anterior.prox = proximo;
                   proximo.ant = anterior;
            }
}
```

Posiciona anterior, atual e próximo sobre a posição que deseja apagar

APAGA

Entrada: item a ser removido
Saída: não há

```
public void apaga(double valor) {
  No atual, anterior, proximo;
  anterior = null; atual = primeiro; proximo = atual.prox;
  while (atual != null) {
      if (atual.item == valor) break;
      anterior = atual;
      atual = atual.prox;
      if (atual != null) proximo = atual.prox;
  }
  if (atual == null) return; // se não encontrou item a apagar sai
  if (atual == primeiro && atual == ultimo) primeiro = ultimo = null;
  else if (atual == primeiro) { primeiro = primeiro.prox;
                                primeiro.ant = null;
       }
       else if (atual.prox == null) { ultimo =  anterior;
                                      ultimo.prox = null;
            }
            else { anterior.prox = proximo;
                   proximo.ant = anterior;
            }
}
```

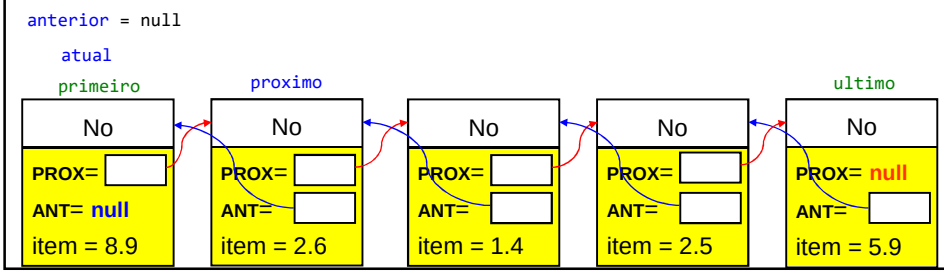
Posiciona anterior, atual e próximo sobre a posição que deseja apagar

Trecho de código responsável por remover item da lista

Trecho de código responsável por posicionar anterior, atual e próximo entre a posição que se deseja remover item

```
No atual, anterior, proximo;
anterior = null; atual = primeiro; proximo = atual.prox;
while (atual != null) {
     if (atual.item == valor) break;
     anterior = atual;
     atual = atual.prox;
     if (atual != null) proximo = atual.prox;
}
```

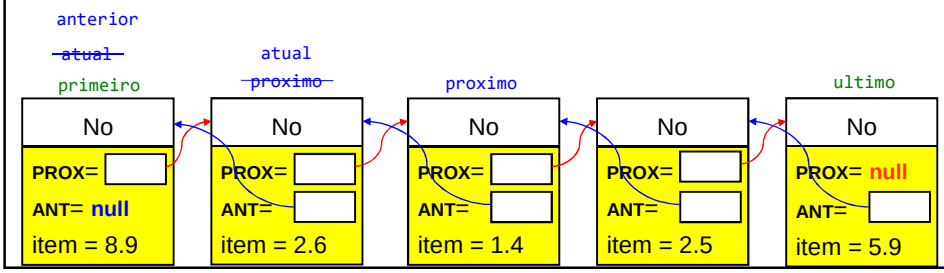
Exemplo: Removendo item 1.4 → `apaga(1.4)`



Trecho de código responsável por posicionar anterior, atual e próximo entre a posição que se deseja remover item

```
No atual, anterior, proximo;
anterior = null; atual = primeiro; proximo = atual.prox;
while (atual != null) {
     if (atual.item == valor) break;
     anterior = atual;
     atual = atual.prox;
     if (atual != null) proximo = atual.prox;
}
```

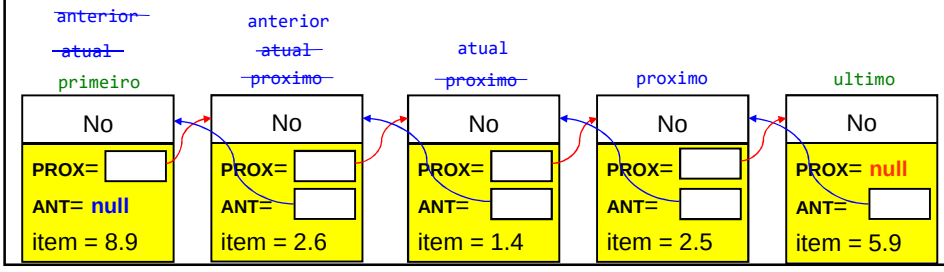
Exemplo: Removendo item 1.4 → `apaga(1.4)`



Trecho de código responsável por posicionar anterior, atual e próximo entre a posição que se deseja remover item

```
No atual, anterior, proximo;
anterior = null; atual = primeiro; proximo = atual.prox;
while (atual != null) {
     if (atual.item == valor) break;
     anterior = atual;
     atual = atual.prox;
     if (atual != null) proximo = atual.prox;
}
```

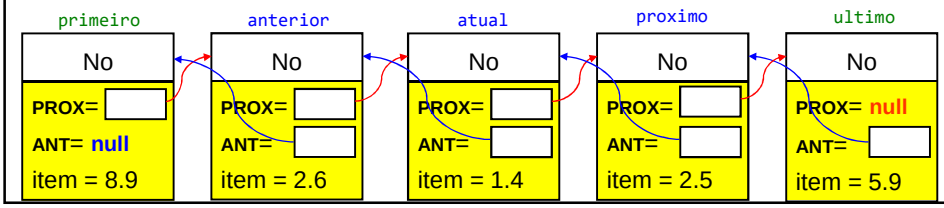
Exemplo: Removendo item 1.4 → `apaga(1.4)`



Trecho de código responsável por posicionar anterior, atual e próximo entre a posição que se deseja remover item

```
No atual, anterior, proximo;
anterior = null; atual = primeiro; proximo = atual.prox;
while (atual != null) {
     if (atual.item == valor) break;
     anterior = atual;
     atual = atual.prox;
     if (atual != null) proximo = atual.prox;
}
```

Exemplo: Removendo item 1.4 → apaga(1.4)



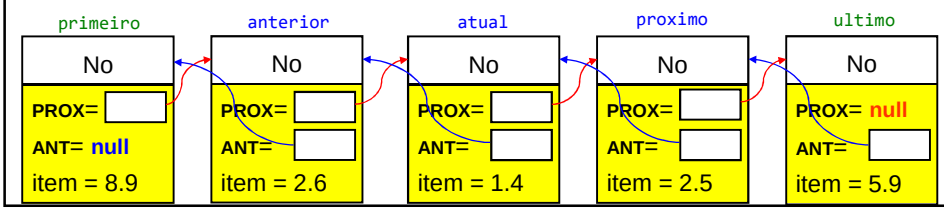
Trecho de código responsável por remover item

```
  if (atual == primeiro && atual == ultimo) primeiro = ultimo = null;
  else if (atual == primeiro) { primeiro = primeiro.prox;
                                primeiro.ant = null;
       }
       else if (atual.prox == null) { ultimo =  anterior;
                                      ultimo.prox = null;
            }
            else { anterior.prox = proximo;
                   proximo.ant = anterior;
            }
}
```

Primeiro if:
Se lista contem um único item e esse é o item a ser removido

Exemplo: Removendo item 1.4 → apaga(1.4)

(atual == primeiro && atual == ultimo) → **Falso**



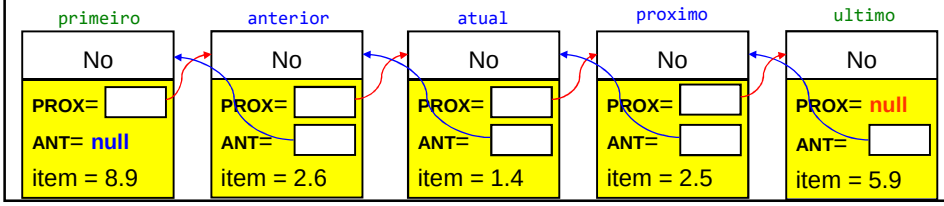
Trecho de código responsável por remover item

```
if (atual == primeiro && atual == ultimo) primeiro = ultimo = null;
else if (atual == primeiro) { primeiro = primeiro.prox;
                              primeiro.ant = null;
     }
     else if (atual.prox == null) { ultimo =  anterior;
                                    ultimo.prox = null;
          }
          else { anterior.prox = proximo;
                 proximo.ant = anterior;
          }
}
```

Segundo if:
Se o item a ser removido esta na primeira posição

Exemplo: Removendo item 1.4 → apaga(1.4)

(atual == primeiro) → **Falso**



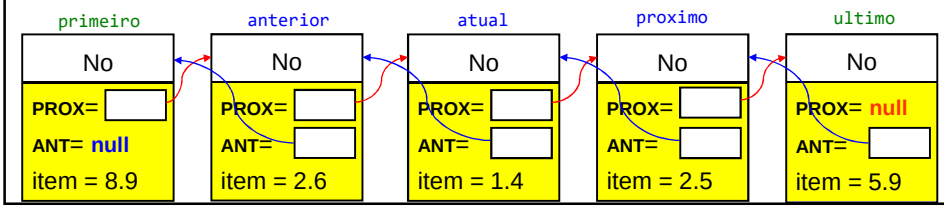
Trecho de código responsável por remover item

```
if (atual == primeiro && atual == ultimo) primeiro = ultimo = null;
else if (atual == primeiro) { primeiro = primeiro.prox;
                              primeiro.ant = null;
     }
     else if (atual.prox == null) { ultimo =  anterior;
                                    ultimo.prox = null;
          }
          else { anterior.prox = proximo;
                 proximo.ant = anterior;
          }
}
```

Terceiro if:
Se o item a ser removido esta na ultima posição

Exemplo: Removendo item 1.4 → apaga(1.4)

(atual.prox == null) → **Falso**



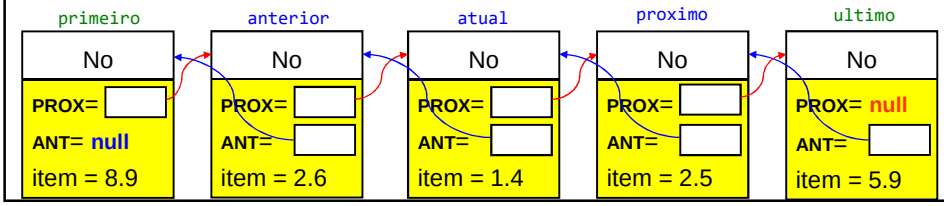
Trecho de código responsável por remover item

```
if (atual == primeiro && atual == ultimo) primeiro = ultimo = null;
else if (atual == primeiro) { primeiro = primeiro.prox;
                              primeiro.ant = null;
     }
     else if (atual.prox == null) { ultimo =  anterior;
                                    ultimo.prox = null;
          }
          else { anterior.prox = proximo;
                 proximo.ant = anterior;
          }
}
```

Se o item a ser removido esta no meio da lista

Exemplo: Removendo item 1.4 → apaga(1.4)

```
anterior.prox = proximo;
proximo.ant = anterior;
```



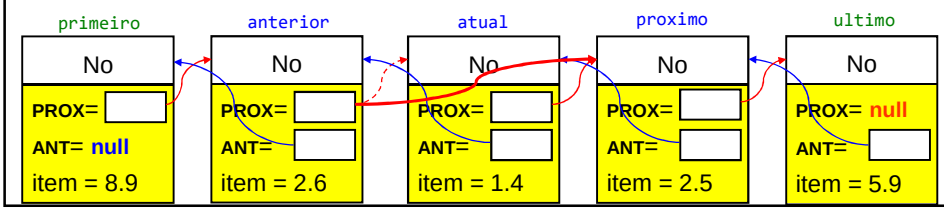
Trecho de código responsável por remover item

```
if (atual == primeiro && atual == ultimo) primeiro = ultimo = null;
else if (atual == primeiro) { primeiro = primeiro.prox;
                              primeiro.ant = null;
     }
     else if (atual.prox == null) { ultimo =  anterior;
                                    ultimo.prox = null;
          }
          else { anterior.prox = proximo;
                 proximo.ant = anterior;
          }
}
```

Se o item a ser removido esta no meio da lista

Exemplo: Removendo item 1.4 → apaga(1.4)

```
anterior.prox = proximo;
proximo.ant = anterior;
```



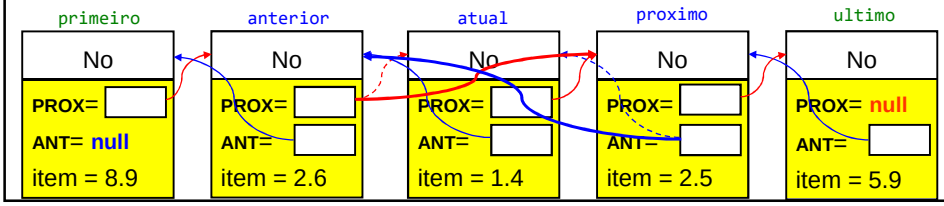
Trecho de código responsável por remover item

```
if (atual == primeiro && atual == ultimo) primeiro = ultimo = null;
else if (atual == primeiro) { primeiro = primeiro.prox;
                              primeiro.ant = null;
     }
     else if (atual.prox == null) { ultimo =  anterior;
                                    ultimo.prox = null;
          }
          else { anterior.prox = proximo;
                 proximo.ant = anterior;
          }
}
```

Se o item a ser removido esta no meio da lista

Exemplo: Removendo item 1.4 → apaga(1.4)

```
anterior.prox = proximo;
proximo.ant = anterior;
```



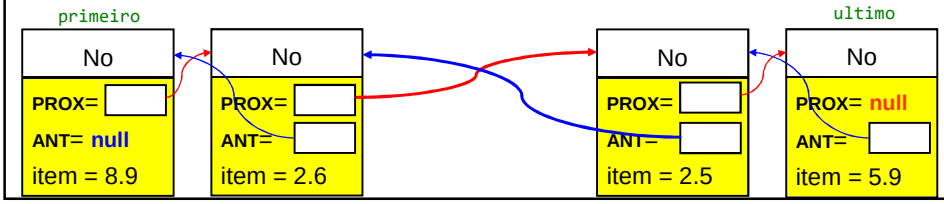
Trecho de código responsável por remover item

```
if (atual == primeiro && atual == ultimo) primeiro = ultimo = null;
else if (atual == primeiro) { primeiro = primeiro.prox;
                              primeiro.ant = null;
     }
     else if (atual.prox == null) { ultimo =  anterior;
                                    ultimo.prox = null;
          }
          else { anterior.prox = proximo;
                 proximo.ant = anterior;
          }
}
```

Se o item a ser removido esta no meio da lista

Exemplo: Removendo item 1.4 → apaga(1.4)

```
anterior.prox = proximo;
proximo.ant = anterior;
```



INSERE

Entrada: Posição de inserção e item a ser inserido
Saída: não há

```
public void insere(int pos, double valor) {
  if (pos < 0) return;
  No novo = new No();
  novo.item = valor;

  if (pos == 0) { // se primeira posição
    novo.prox = primeiro;
    novo.ant = null;
    primeiro.ant = novo;
    primeiro = novo;
    return; // saída imediata do método
  }

// Continua ...
```

Se posição passada como parâmetro for menor que zero, sai do método

Aloca memória para o novo Nó

Acertando elos do novo Nó e do Nó inicial

```
// Continuação ...

 No atual, proximo;
 atual = primeiro;
 proximo = atual.prox;
 for(int i=1; i < pos && atual.prox != null; i++) {
    atual = atual.prox;
    proximo = atual.prox;
 }

 novo.prox = atual.prox;
 novo.ant = atual;
 atual.prox = novo;

 if (novo.prox == null) ultimo = novo;
 else proximo.ant = novo;

} // fim método insere
```

Posiciona atual e próximo entre a posição que se deseja inserir item

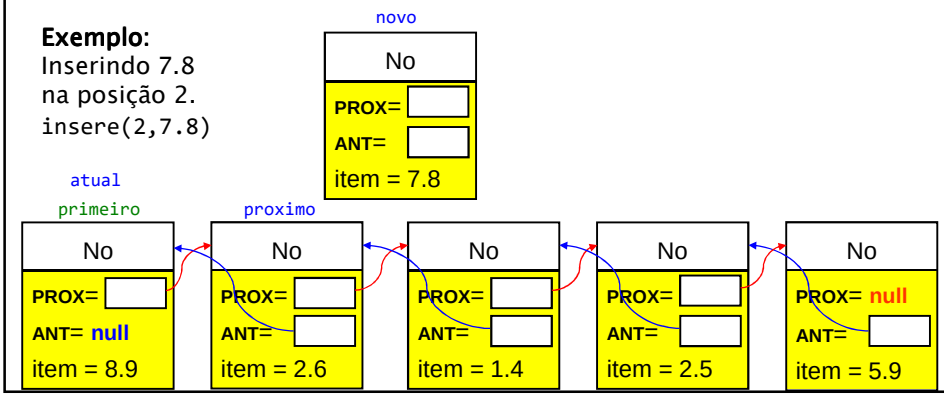
Ajusta os elos do novo Nó

Caso inserção no final: Define o novo Nó final
Caso inserção no meio: Ajusta o elo anterior do próximo Nó.

Trecho de código responsável por posicionar atual e próximo entre a posição que se deseja inserir item

```
No atual, proximo;
atual = primeiro;
proximo = atual.prox;
for(int i=1; i < pos && atual.prox != null; i++) {
    atual = atual.prox;
    proximo = atual.prox;
}
```

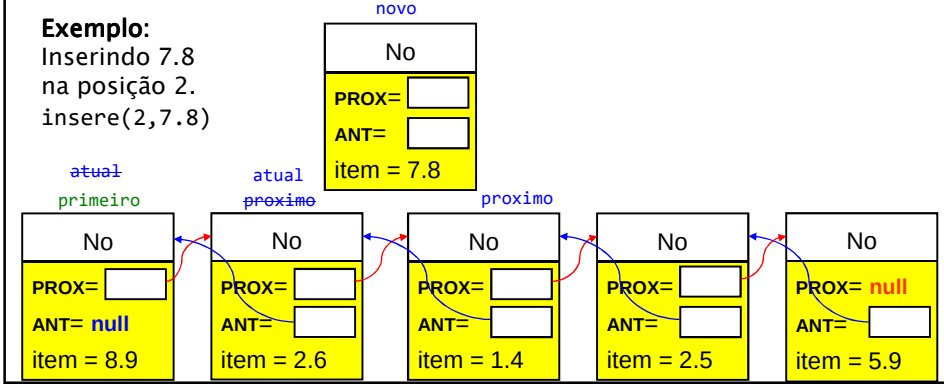
Exemplo:
Inserindo 7.8
na posição 2.
`insere(2,7.8)`



Trecho de código responsável por posicionar atual e próximo entre a posição que se deseja inserir item

```
No atual, proximo;
atual = primeiro;
proximo = atual.prox;
for(int i=1; i < pos && atual.prox != null; i++) {
    atual = atual.prox;
    proximo = atual.prox;
}
```

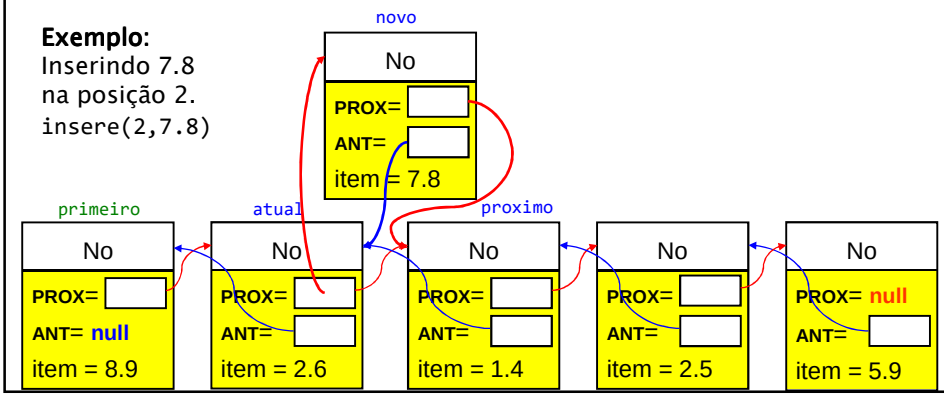
Exemplo:
Inserindo 7.8
na posição 2.
`insere(2,7.8)`



Trecho de código responsável por ajustar os elos do novo Nó

```
novo.prox = atual.prox;
novo.ant = atual;
atual.prox = novo;
```

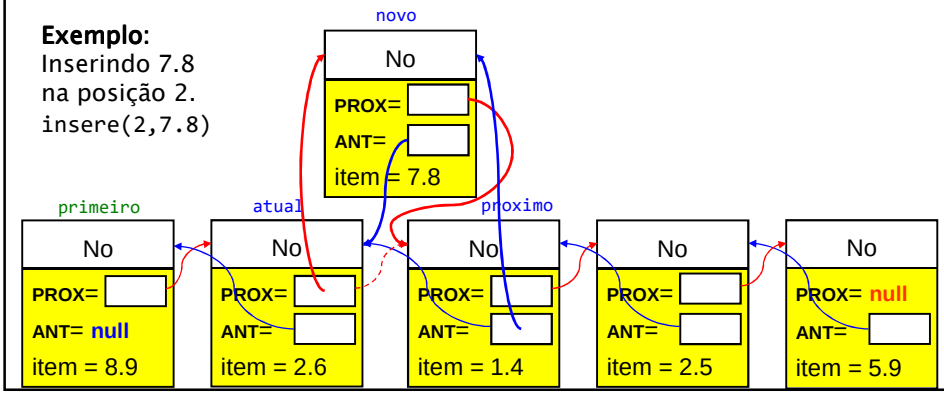
Exemplo:
Inserindo 7.8
na posição 2.
`insere(2,7.8)`



Trecho de código responsável por:
Caso inserção no final: Definir o novo Nó final
Caso inserção no meio: Ajustar o elo anterior do próximo Nó.

```
if (novo.prox == null) ultimo = novo;
else proximo.ant = novo;
```

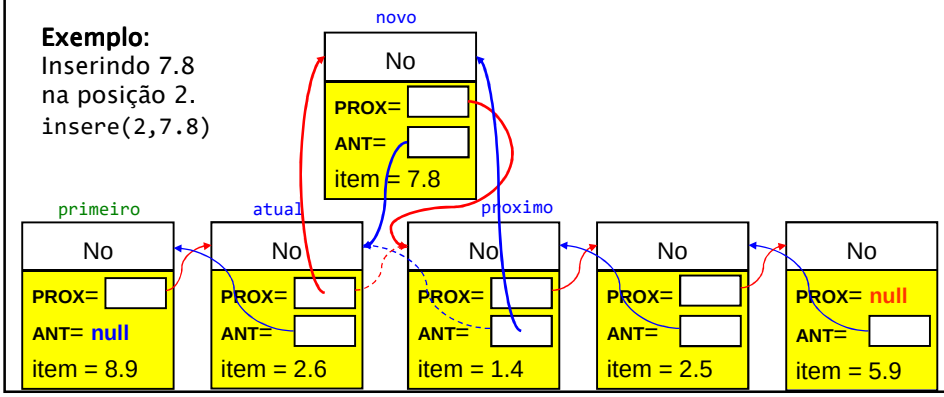
Exemplo:
Inserindo 7.8
na posição 2.
`insere(2,7.8)`



Trecho de código responsável por:
Caso inserção no final: Definir o novo Nó final
Caso inserção no meio: Ajustar o elo anterior do próximo Nó.

```
if (novo.prox == null) ultimo = novo;
else proximo.ant = novo;
```

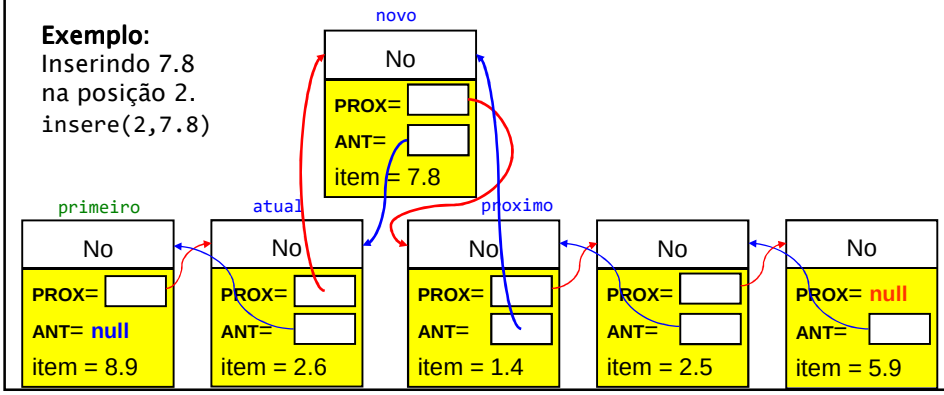
Exemplo:
Inserindo 7.8
na posição 2.
`insere(2,7.8)`



Trecho de código responsável por:
Caso inserção no final: Definir o novo Nó final
Caso inserção no meio: Ajustar o elo anterior do próximo Nó.

```
if (novo.prox == null) ultimo = novo;
else proximo.ant = novo;
```

Exemplo:
Inserindo 7.8
na posição 2.
`insere(2,7.8)`



Classe para Lista Duplamente Encadeada

```
class ListaDE {

  private No primeiro, ultimo; // referencia ao primeiro e ultimo Nó da lista
  public ListaDE() { primeiro = ultimo = null; } // Construtor

  public boolean vazio()      { return (primeiro==null); }
  public double item_frente() { return primeiro.item;    }
  public double item_fim()    { return ultimo.item;      }

  public void insere_fim(double valor) {
     No novo = new No(); // Alocando memoria para o Nó
     novo.item = valor;
     novo.prox = null;
     if (vazio()) {
         novo.ant = null;
         primeiro = novo;
     }
     else {
         novo.ant = ultimo;
         ultimo.prox = novo;
     }
     ultimo = novo;
  }

// Continua ...
```

```
// Continuação ...

 public void insere(int pos, double valor) {
    if (pos < 0) return; // pos = POSIÇÃO DE INSERÇÃO
    No novo = new No(); // Alocando memoria para o Nó
    novo.item = valor;
    if (pos == 0) { // se na primeira posição
      novo.prox = primeiro;
      novo.ant = null;
      primeiro.ant = novo;
      primeiro = novo;
      return;
    }
    No atual, proximo;
    atual = primeiro;
    proximo = atual.prox;
    for(int i=1; i < pos && atual.prox != null; i++) {
         atual = atual.prox; // posicionando no Nó atual
         proximo = atual.prox;
    }
    novo.prox = atual.prox;
    novo.ant = atual;
    atual.prox = novo;
    if (novo.prox == null) ultimo = novo; // se ultima posição
    else proximo.ant = novo;
  }

// Continua ...
```

```
// Continuação ...

  public boolean pesquisa(double chave) {
    No atual = primeiro;
    while (atual != null) {  // caminhando para o fim da lista
      if (atual.item == chave) return true;
      atual = atual.prox;
    }
    return false;
  }

  public void imprime_inicio() {
    No atual = primeiro;
    while (atual != null) {   // caminhando para o fim da lista
      System.out.print("{" + atual.item + "} ");
      atual = atual.prox;
    }
    System.out.println("");
  }

  public void imprime_fim() {
    No atual = ultimo;
    while (atual != null) {   // caminhando para o fim da lista
      System.out.print("{" + atual.item + "} ");
      atual = atual.ant;
    }
    System.out.println("");
  }                                                          // Continua...
```

```
// Continuação ...

  public void apaga(double valor) {
    No atual, anterior, proximo;
    anterior = null; atual = primeiro; proximo = atual.prox;
    while (atual != null) {
      if (atual.item == valor) break;
      anterior = atual;
      atual = atual.prox;
      if (atual != null) proximo = atual.prox;
    }
    if (atual == null) return;
    if (atual == primeiro && atual == ultimo) primeiro = ultimo = null;
    else if (atual == primeiro) { primeiro = primeiro.prox;
                                  primeiro.ant = null;
         }
         else if (atual.prox == null) { ultimo =  anterior;
                                        ultimo.prox = null;
              }
              else { anterior.prox = proximo;
                     proximo.ant = anterior;
              }
  }

} // fim classe ListaDE
```

Exemplo de uso da classe ListaDE

```
class ListaDEApp {
  public static void main (String[] args) {

    ListaDE l = new ListaDE();

    System.out.println(" >>> Adicionando 8.99, 3.43, 8.56 e 5.5 no fim");
    l.insere_fim(8.99); l.insere_fim(3.43); l.insere_fim(8.56); l.insere_fim(5.5);

    System.out.print(" Imprimindo Lista inicio ao fim: "); l.imprime_inicio();
    System.out.print(" Imprimindo Lista fim ao inicio: "); l.imprime_fim();

    System.out.println(" >>> Adicionando 0 na posicao 2"); l.insere(2,0);

    System.out.println(" >>> Adicionando 15 no fim");      l.insere_fim(15);

    System.out.print(" Imprimindo Lista inicio ao fim: "); l.imprime_inicio();
    System.out.print(" Imprimindo Lista fim ao inicio: "); l.imprime_fim();

    System.out.println(" >>> Removendo todos os itens da lista do inicio ao fim:");
    while (!l.vazio()) {
       System.out.println(" Apagado {" + l.item_frente() + "}");
       l.apaga(l.item_frente());
    }

  }
}
```

Classe ListaOrd:

Campos

```
private No primeiro;
private No ultimo;
```

Construtor

```
public ListaOrd() {
    primeiro = null;
    ultimo = null;
}
```

Métodos

vazio: Retorna verdadeiro se lista vazia.
insere: Insere itens na lista de maneira ordenada.
pesquisa: Retorna verdadeiro se item encontrado.
imprime_inicio: Imprime lista toda do inicio ao fim.
imprime_fim: Imprime lista toda do fim ao inicio.
apaga_menor: Remove o menor item
apaga_maior: Remove o maior item
item_menor: Retorna o menor item
item_maior: Retorna o maior item

APAGA_MENOR

Entrada: não há
Saída: não há

```
public void apaga_menor() {
 if( vazio() ) return;
 if( primeiro == ultimo ) { ultimo = primeiro = null;
                            return;
 }
 primeiro = primeiro.prox;
 primeiro.ant = null;
}
```

APAGA_MAIOR

Entrada: não há
Saída: não há

```
public void apaga_maior() {
 if( vazio() ) return;
 if( primeiro == ultimo ) { ultimo = primeiro = null;
                            return;
 }
 ultimo = ultimo.ant;
 ultimo.prox = null;
}
```

ITEM_MENOR

Entrada: não há
Saída: menor item

```
public double item_menor() {
   return primeiro.item;
}
```

ITEM_MAIOR

Entrada: não há
Saída: maior item

```
public double item_maior() {
   return ultimo.item;
}
```

INSERE

Entrada: valor entrada
Saída: não há

```
public void insere(double valor) {
  No novo = new No(); // Alocando memoria para o Nó
  novo.item = valor; // Atribuindo valor para o novo Nó
  if (vazio()) { // Se lista vazia
    novo.ant = novo.prox = null;
    primeiro = ultimo = novo;
    return; // se lista vazia insere o novo Nó e sai do método
  }
  No atual = primeiro;
  No proximo = atual.prox;
  No anterior = null;
  while (atual != null) {
     if (atual.item > valor) break;
     anterior = atual;
     atual = atual.prox;
     if (atual == null) proximo = null;
     else proximo = atual.prox;
  }
                                        // Continua ...
```

Trecho de código responsável por posicionar atual, proximo e anterior entre a posição que se deseja inserir o novo Nó

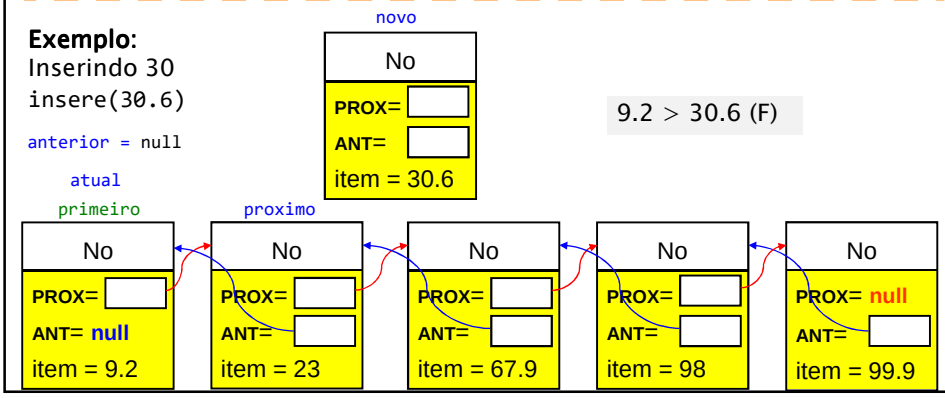
```
// Continuação

  if (anterior == null) { // Se o No será inserido no inicio
     novo.prox = primeiro;
     novo.ant = null;
     primeiro.ant = novo;
     primeiro = novo;
  }
  else if (atual == null) { // Se o No será inserido no fim
           novo.prox = null;
           anterior.prox = novo;
           novo.ant = anterior;
           ultimo = novo;
       }
       else { // Se No será inserido no meio
            novo.prox = atual;
            novo.ant = anterior;
            atual.ant = novo;
            anterior.prox = novo;
       }
} // fim método insere
```

Trecho de código responsável por posicionar atual, proximo e anterior entre a posição que se deseja inserir o novo Nó

```
No atual = primeiro, proximo = atual.prox, anterior = null;
while (atual != null) { // caminha para a posição de inserir
     if (atual.item > valor) break;
     anterior = atual;
     atual = atual.prox;
     if (atual == null) proximo = null; // se chegou no final
     else proximo = atual.prox;
} // fim laço while
```

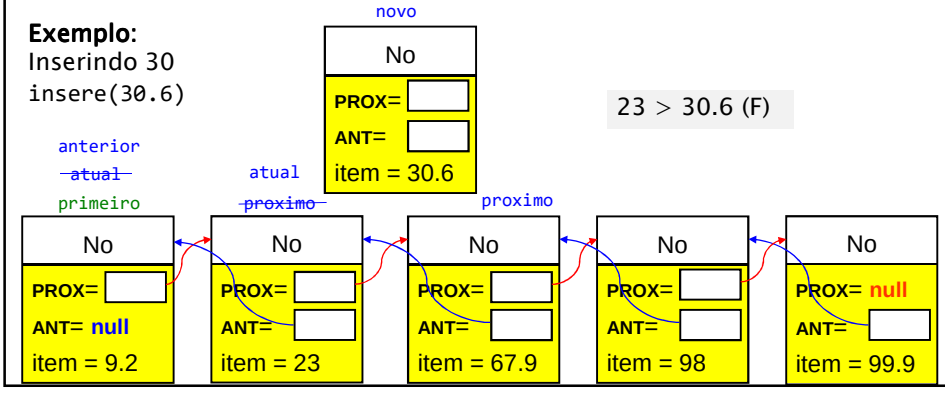
Exemplo:
Inserindo 30
insere(30.6)



Trecho de código responsável por posicionar atual, proximo e anterior entre a posição que se deseja inserir o novo Nó

```
No atual = primeiro, proximo = atual.prox, anterior = null;
while (atual != null) { // caminha para a posição de inserir
     if (atual.item > valor) break;
     anterior = atual;
     atual = atual.prox;
     if (atual == null) proximo = null; // se chegou no final
     else proximo = atual.prox;
} // fim laço while
```

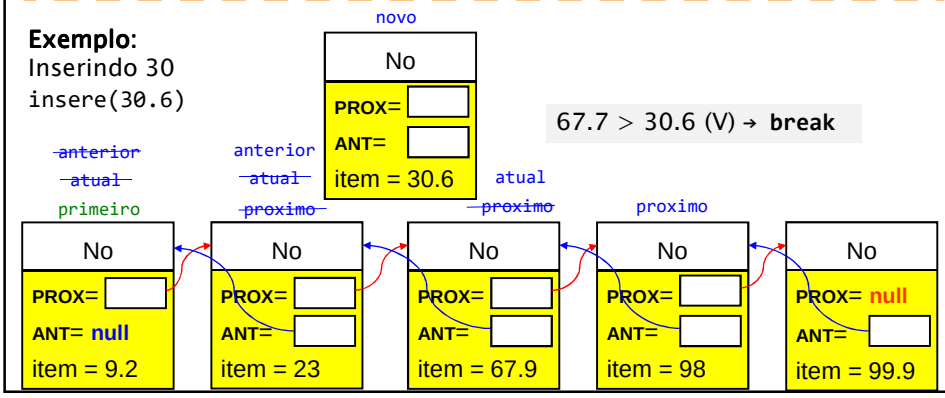
Exemplo:
Inserindo 30
insere(30.6)



Trecho de código responsável por posicionar atual, proximo e anterior entre a posição que se deseja inserir o novo Nó

```
No atual = primeiro, proximo = atual.prox, anterior = null;
while (atual != null) { // caminha para a posição de inserir
     if (atual.item > valor) break;
     anterior = atual;
     atual = atual.prox;
     if (atual == null) proximo = null; // se chegou no final
     else proximo = atual.prox;
} // fim laço while
```

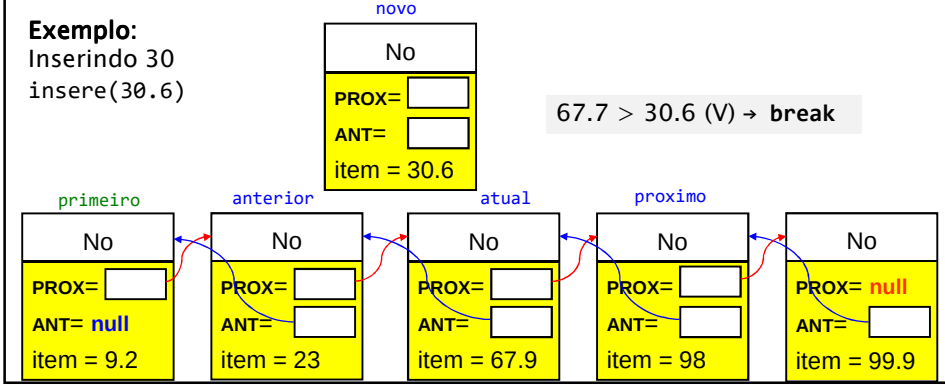
Exemplo:
Inserindo 30
`insere(30.6)`



Trecho de código responsável por posicionar atual, proximo e anterior entre a posição que se deseja inserir o novo Nó

```
No atual = primeiro, proximo = atual.prox, anterior = null;
while (atual != null) { // caminha para a posição de inserir
     if (atual.item > valor) break;
     anterior = atual;
     atual = atual.prox;
     if (atual == null) proximo = null; // se chegou no final
     else proximo = atual.prox;
} // fim laço while
```

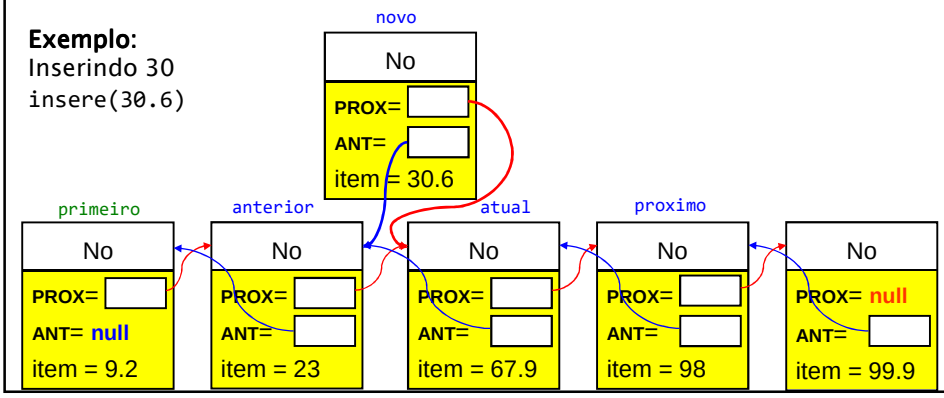
Exemplo:
Inserindo 30
`insere(30.6)`



Trecho de código responsável por inserção do Nó no meio

```
novo.prox = atual;
novo.ant = anterior;
atual.ant = novo;
anterior.prox = novo;
```

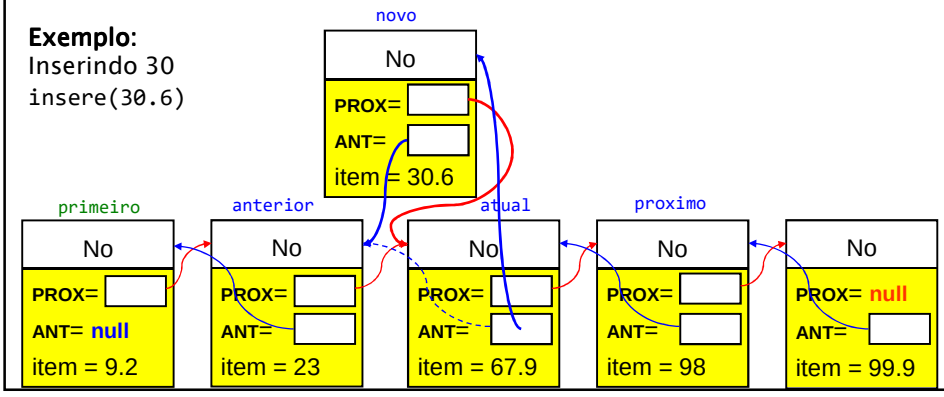
Exemplo:
Inserindo 30
`insere(30.6)`



Trecho de código responsável por inserção do Nó no meio

```
novo.prox = atual;
novo.ant = anterior;
atual.ant = novo;
anterior.prox = novo;
```

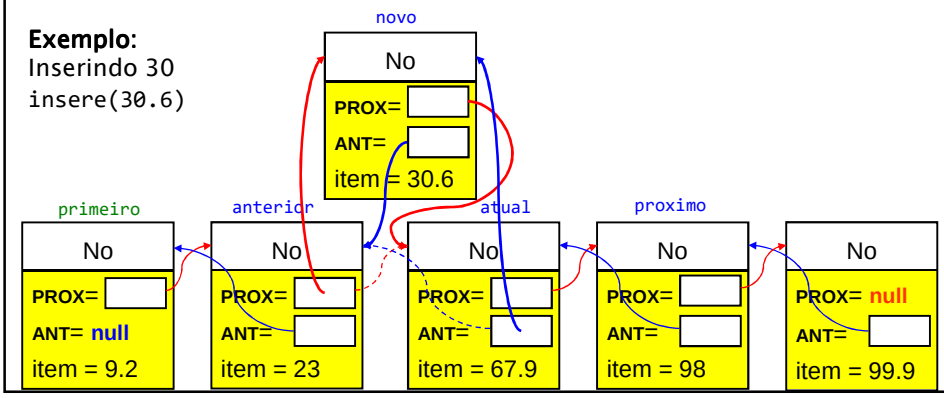
Exemplo:
Inserindo 30
`insere(30.6)`



Trecho de código responsável por inserção do Nó no meio

```
novo.prox = atual;
novo.ant = anterior;
atual.ant = novo;
anterior.prox = novo;
```

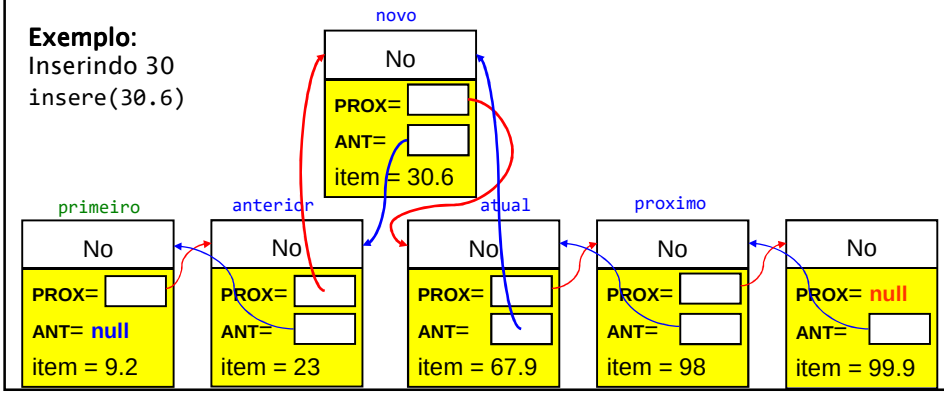
Exemplo:
Inserindo 30
`insere(30.6)`



Trecho de código responsável por inserção do Nó no meio

```
novo.prox = atual;
novo.ant = anterior;
atual.ant = novo;
anterior.prox = novo;
```

Exemplo:
Inserindo 30
`insere(30.6)`



Exemplo de uso da classe ListaOrd

```
class ListaDEOrdApp {
  public static void main (String[] args) {
    ListaOrd l = new ListaOrd();

    System.out.println(" >>> Adicionando 8.99, 3.43, 8.56 e 5.5 de
maneira ordenada");
    l.insere(8.99); l.insere(3.43); l.insere(8.56); l.insere(5.5);

    System.out.print(" Imprimindo Lista inicio: ");
    l.imprime_inicio();
    System.out.print(" Imprimindo Lista fim:    ");
    l.imprime_fim();

    System.out.println(" >>> Removendo todos os itens da lista:");
    while (!l.vazio()) {
       System.out.println(" Apagado {" + l.item_menor() + "}");
       l.apaga_menor();
    }

  } // fim programa principal
} // fim classe ListaDEOrdApp
```

Comparação Vetores e Listas Encadeadas

Em muitas situações o vetor será o primeiro tipo de estrutura que você deverá considerar ao armazenar e manipular dados.
Eles são úteis quando:
- A quantidade de dados é razoavelmente pequena
- A quantidade de dados é previsível de antemão

Se a velocidade de inserção for importante, use vetor não ordenado.
Se a velocidade de pesquisa for importante use vetor ordenado.
A eliminação é sempre lenta em vetores.

Considere uma lista sempre quando:
- A quantidade de dados a ser armazenada não puder ser prevista de antemão ou
- Quando dados forem inseridos e removidos com frequência.

A lista encadeada pode se expandir para preencher toda a memória disponível, pois, os dados podem ser inseridos em qualquer espaço.

Uma lista encadeada é mais complicada de programar que vetor, mas, é mais simples em comparação que uma arvore e tabela hash.

Tabela de Estruturas de Armazenamento

de Propósito Geral

Resumo das velocidades das várias estruturas de Armazenamento de Propósito Geral usando a notação Big O

	Inserção	Eliminação	Pesquisa
Vetor	O(1)	O(n)	O(n)
Vetor Ordenado	O(n)	O(n)	O(log n)
Lista Simplesmente Encadeada	O(1) fim ou inicio O(n) meio	O(n)	O(n)
Lista Duplamente Encadeada	O(1) fim ou inicio O(n) meio	O(n)	O(n)
Lista Duplamente Encadeada Ordenada	O(n)	O(1) fim ou inicio O(n) meio	O(n)

Atividades

1. Faça o seguinte menu iterativo com o usuário dos métodos da classe Lista Encadeada Ordenada. Implemente uma Lista de itens do tipo long.

```
PROGRAMA LISTA ENCADEADA ORDENADA DO TIPO LONG
0: Sair
1: Inserir itens
2: Apaga item menor
3: Apaga item maior
4: Imprimir maior item
5: Imprimir menor item
6: Imprimir todo conteúdo
Entre com a opção desejada:
```

2. Reescreva os métodos de inserção de itens da classe Lista Duplamente Encadeada de double para que os mesmos não permitam a inserção de itens já existentes na lista.