Estruturas de Dados e Algoritmos

Lista Ligada

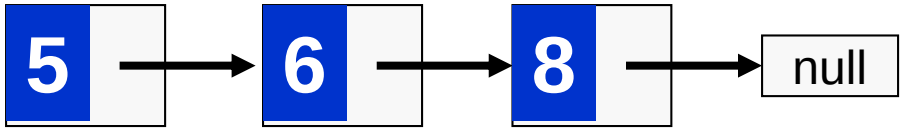
Adalberto Cajueiro

Departamento de Sistemas e Computação

Universidade Federal de Campina Grande

Lista Ligada (Linked List)

- É uma estrutura de dados em que objetos sao arranjados em ordem linear.
- Ordem estabelecida por ligacao (links) entre elementos da lista.
- Cada nó da lista armazena um elemento e uma referencia para o próximo nó da lista.



Lista Ligada (Linked List)

- Características
  - Um dos tipos abstratos de dados fundamentais
  - Seqüência de tamanho arbitrário e dinâmico de elementos de algum tipo base
  - Utiliza o espaço necessário (alocação dinâmica)
  - A ordem é determinada por um ponteiro e não pelo índice como no array
  - Acesso sequencial

Lista

- Operações
  - Inserir
  - Remover
  - Pesquisar
  - Tamanho
  - Vazio

Lista

- Operações
  - Inserir
  - Remover
  - Pesquisar
  - Tamanho
  - Vazio

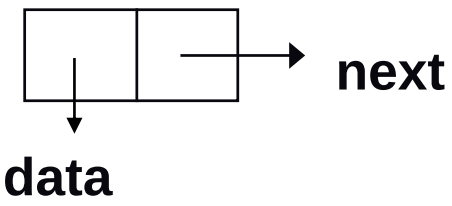
Como definir em Java tal estrutura?

LISTA (IMPLEMENTACAO)

```
public interface LinkedList<T> {
 public boolean isEmpty();
 public int size();
 public T search(T element);
 public void insert(T element);
 public void remove(T element);
 public T[] toArray();
}
```

Questões de Implementação

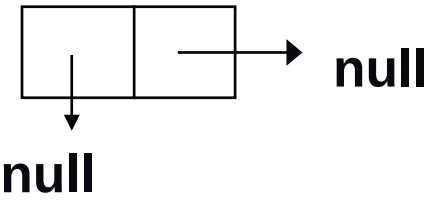
- Implementacao estática – implementada com arrays
- Implementacao dinâmica – através de duas abordagens:
  - Estrutura recursiva com métodos iterativos
    - Cada elemento (nó) da lista contém um dado armazenado e um apontador para o próximo elemento da lista
  - Estrutura recursiva com métodos recursivos
    - Cada elemento (nó) da lista contém um dado armazenado e um apontador para o próximo elemento da lista



Questões de Implementação

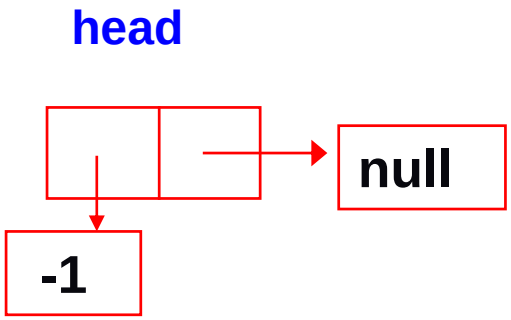
- Lista vazia
  - Representada por um nó especial (NIL) sem dado
  - Pode também ser representado por **null** (não recomendado)
- Nós sentinela
  - Representam a lista vazia e são acrescentados nas extremidades das listas
  - Simplificam e aceleram alguns algoritmos sobre lista e dão garantia de que uma lista sempre contém algum nó (mesmo que seja vazio)

Nó sentilena NIL

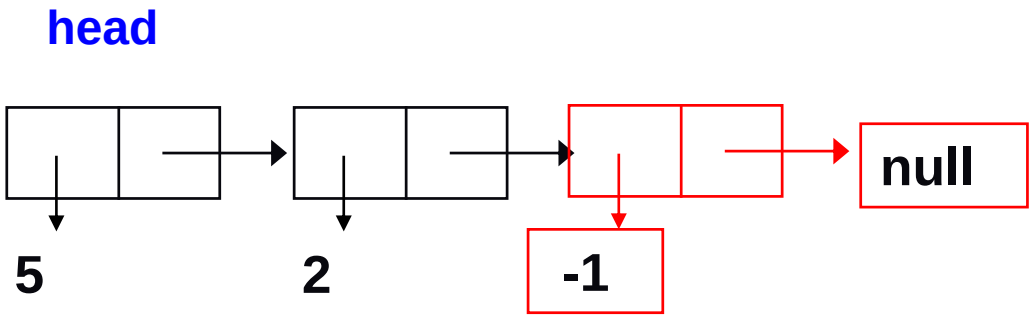


IMPLEMENTAÇÃO

- Exemplo: lista de inteiros não negativos



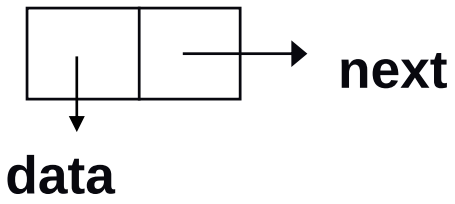
Lista vazia=NIL
(caso base)



Lista não vazia
(caso indutivo)

Como implementar em Java?

LISTA (IMPLEMENTACAO)




```
public class SingleLinkedListNode<T> {
 protected T data;
 protected SingleLinkedListNode<T> next;

 public SingleLinkedListNode () {
 }
 public SingleLinkedListNode(T data,Node<T> next){
  this.data = data;
  this.next = next;
 }
 public boolean isNIL(){
  return (this.data == null);
 }
}
```

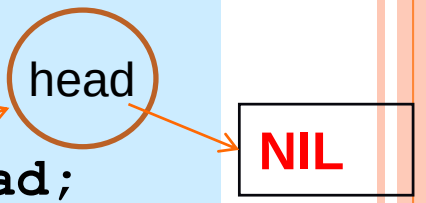
NIL

next

data

LISTA (IMPLEMENTACAO)

```
public class SingleLinkedListImpl<T>
        implements LinkedList<T>{

  protected SingleLinkedListListNode<T> head;

  public SingleLinkedListImpl() {
    head = new SingleLinkedListNode<T>();
  }
  ...
}
```



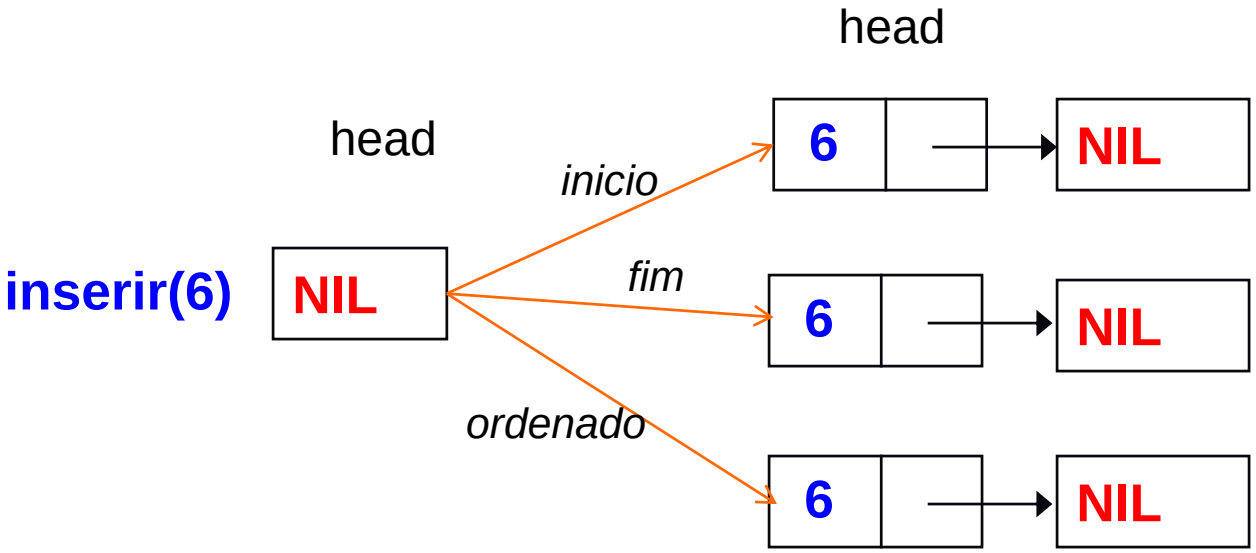
LISTA (ISEMPTY)

- Apenas a lista vazia deve retornar true.

```
isEmpty(){
  if head == NIL
      return true
  else
     return false
}
```

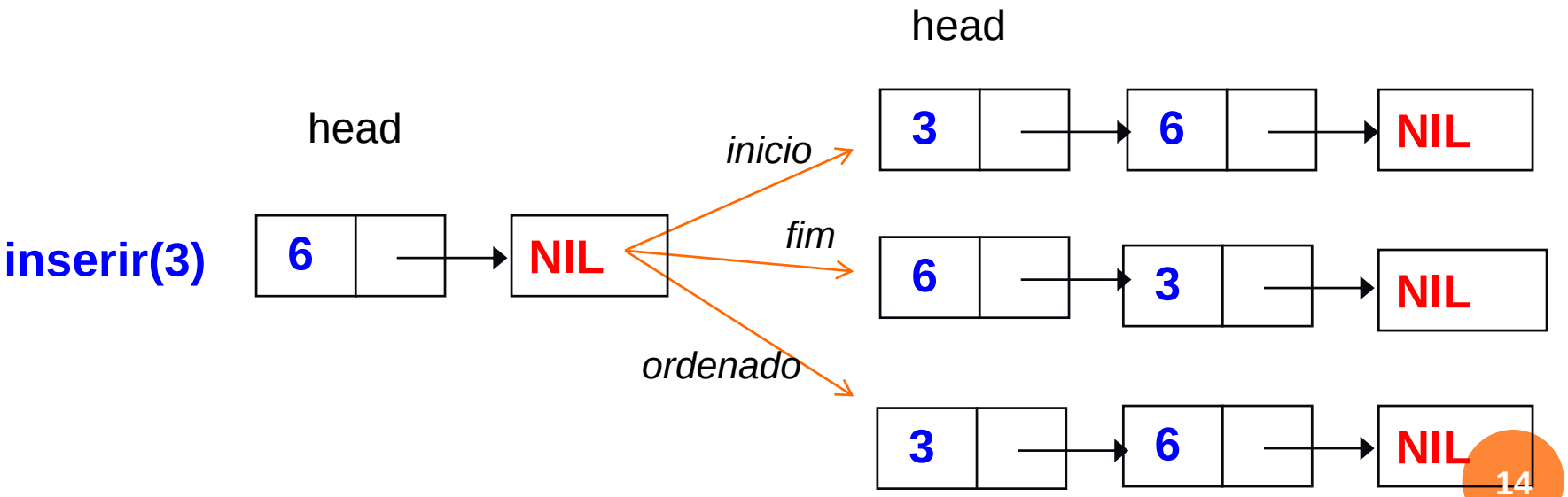
LISTA (INSERIR)

- Listas não ordenadas
  - Insercoes no início ou no fim (default)
- Listas ordenadas
  - Insercoes na posição correta (preservar ordem)



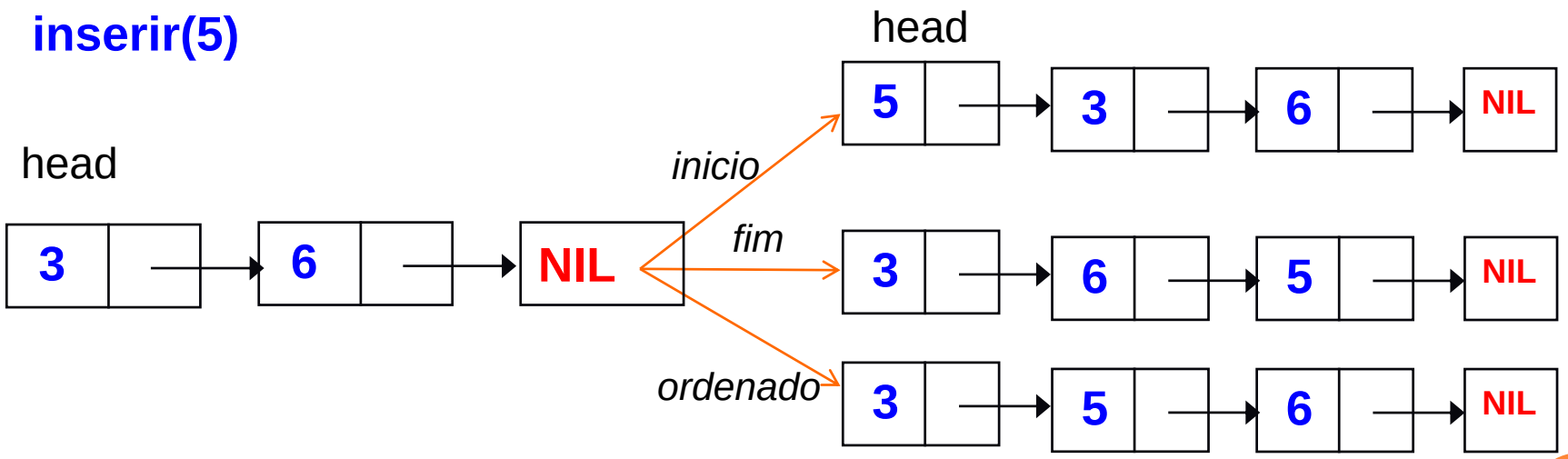
LISTA (INSERIR)

- Listas não ordenadas
  - Insercoes no início ou no fim (default)
- Listas ordenadas
  - Insercoes na posição correta (preservar ordem)



LISTA (INSERIR)

- Listas não ordenadas
  - Insercoes no início ou no fim (default)
- Listas ordenadas
  - Insercoes na posição correta (preservar ordem)



Lista (Inserir)

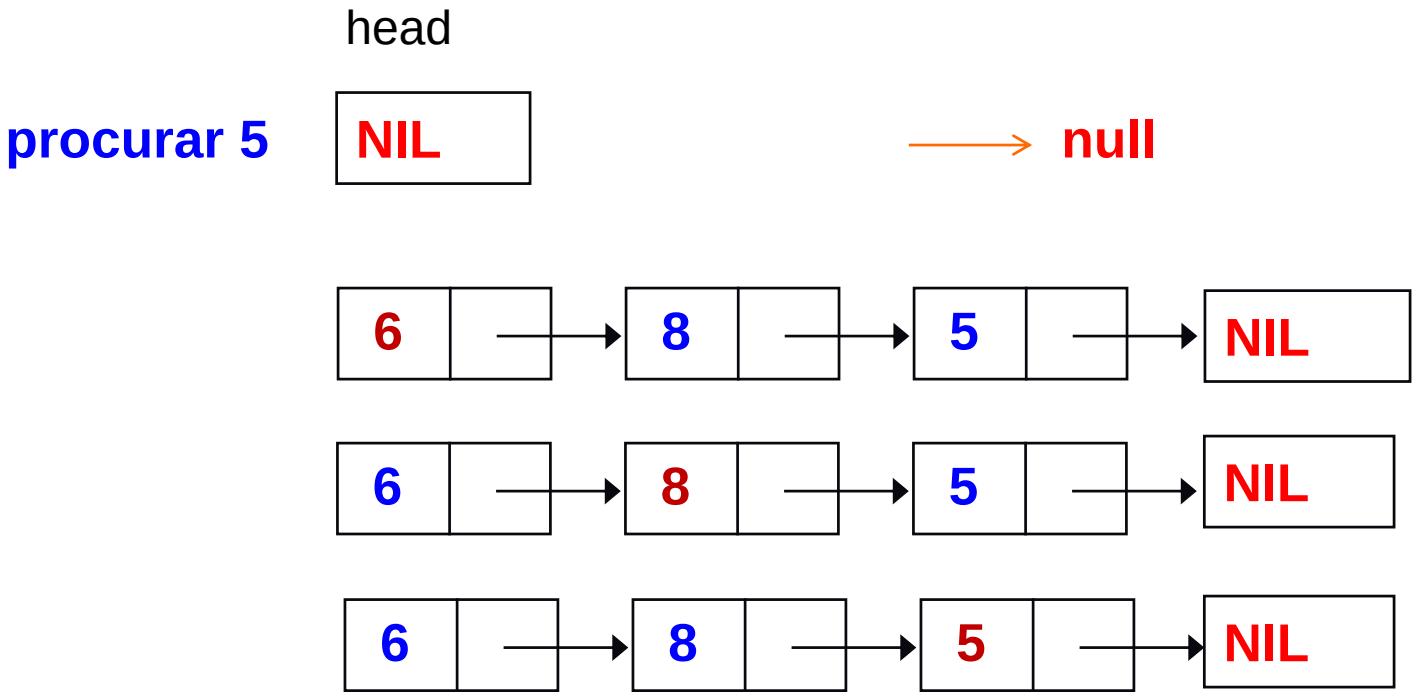
- Adotar listas não ordenadas e inserção ao final

```
list-insert(item){
  auxHead = head
  if(head = NIL){
    newHead = new SingleLinkedListNode(item)
    newHead.next= head
    head = newHead
  } else{
    while(auxHead.next != NIL){
       auxHead = auxHead.next
    }
   newNode = new SingleLinkedListNode(item)
   newNode.next = auxHead.next
   auxHead.next = newNode
  }
}
```

Qual o tempo de execução?

LISTA (PROCURAR)

- A procura é sequencial e se dá pelo valor armazenado



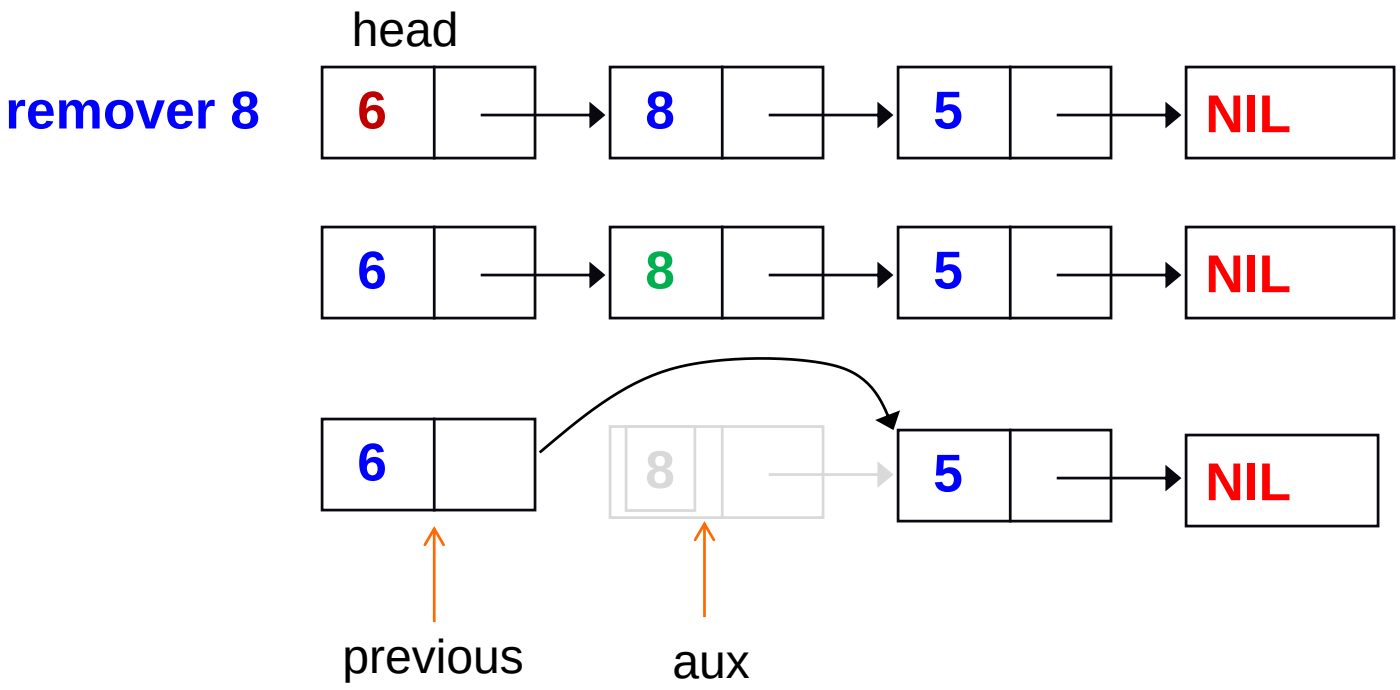
LISTA (PROCURAR)

```
LIST-SEARCH(L, k)
1  x ← head[L]
2  while x ≠ NIL and key[x] ≠ k
3        do x ← next[x]
4  return x
```

Qual o tempo de execução?

LISTA (REMOVER)

- As remocoes se dão pelo valor armazenado



Como seria a remoção na lista?

Lista (Remover)

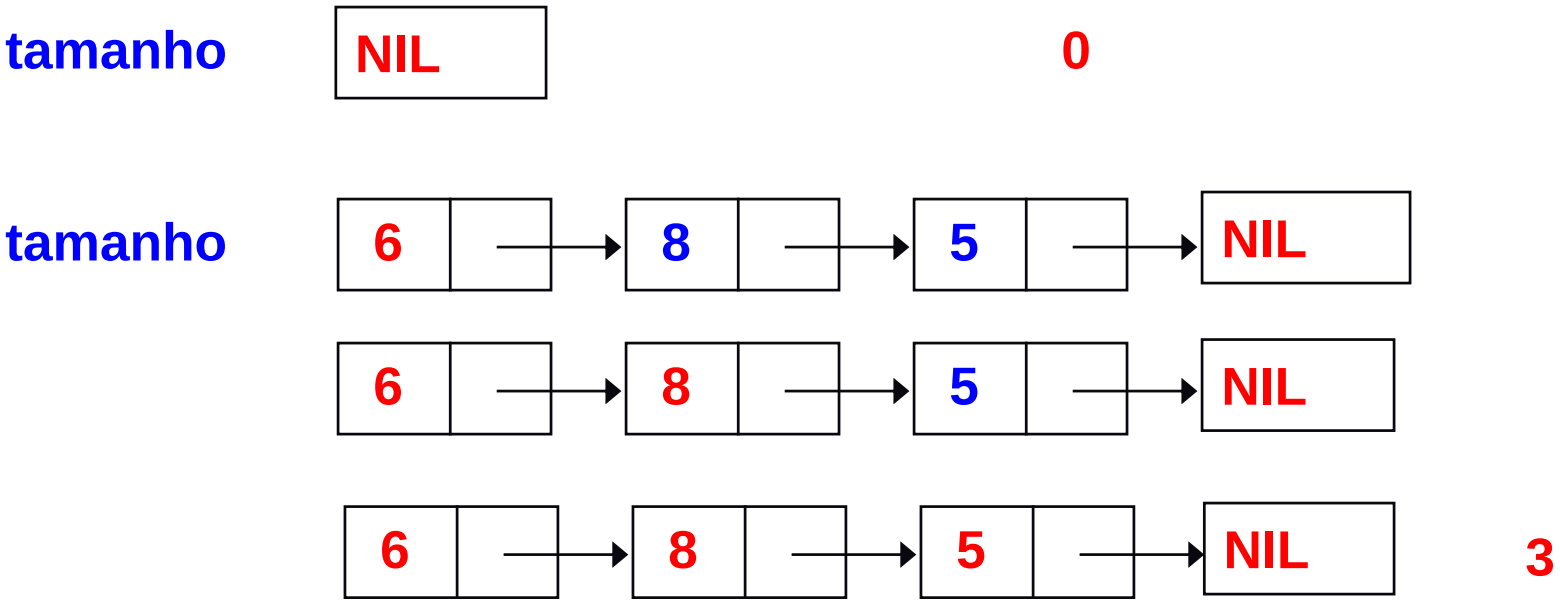
- As remocoes se dão pelo valor armazenado

```
list-remove(item){
  if head.data == item
     head = head.next
  else{
    aux = head
    while aux != NIL and aux.data != item {
      previous = aux
      aux = aux.next
    }
    if aux != NIL{
     previous.next = aux.next
  }
```

Qual o tempo de execução?

LISTA (TAMANHO)

- Como calcular o tamanho de uma lista?



Como seria o algoritmo do tamanho?

Lista (Tamanho)

```
list-size(){
  size = 0
  aux = head
  while aux != NIL {
      size = size + 1
      aux = aux.next
  }
  return size
}
```

Qual o tempo de execução?

LISTA (TOARRAY)

- Deve retornar um array contendo os elementos da lista na ordem do head para o final.

```
T[] list-toArray(){
  T[] result = new T[]
  aux = head
  int count = 0
  while aux != NIL {
      result[count] = aux.data
      aux = aux.next
      count++
  }
  return result
}
```

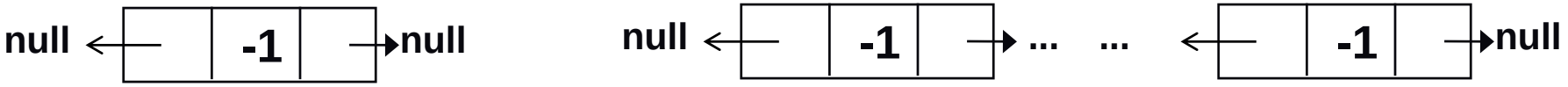
Exercício

- Implemente um método que encontre o maior elemento de uma lista ligada.
- Implemente um método que encontre o menor elemento de uma lista ligada.

Lista Duplamente Ligada

Lista Duplamente Ligada

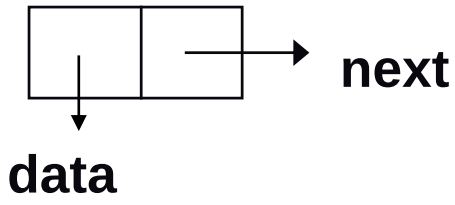
- Cada nó tem referências para os nós sucessor e predecessor
- Listas duplamente encadeadas têm a vantagem de poderem ser percorridas em ambas as direções

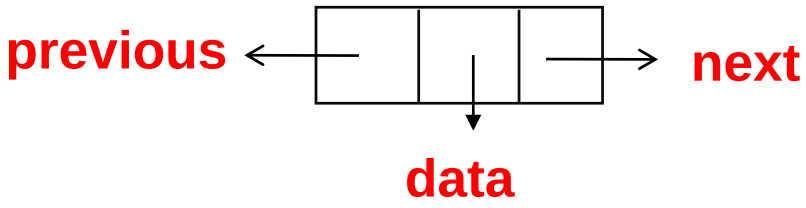


- Isto permite reduzir para a metade o tempo máximo para acessar um elemento com base no seu índice
  - Head e last ficam acessíveis na lista dupla

LISTA (IMPLEMENTACAO)

- Como seria a implementação de uma lista duplamente ligada?
  - A estrutura do nó muda?
  - A estrutura da lista em si muda?

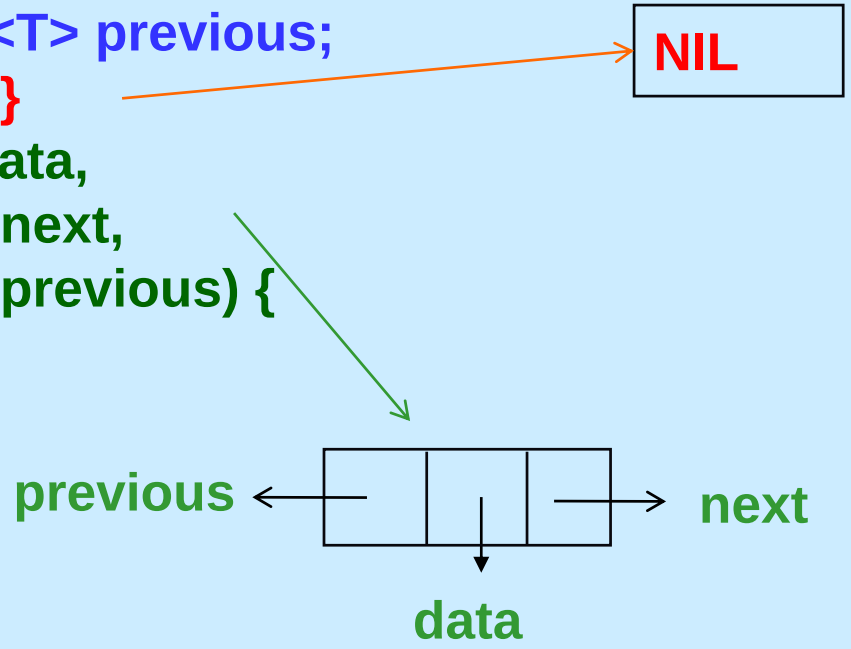




LISTA (IMPLEMENTACAO)

- Como seria a implementação de uma lista duplamente ligada?

```
public class DoubleLinkedListNode<T>
   extends SingleLinkedListNode<T> {

  protected DoubleLinkedListNode<T> previous;
  public DoubleLinkedListNode() {  }
  public DoubleLinkedListNode(T data,
            DoubleLinkedListNode<T> next,
            DoubleLinkedListNode<T> previous) {

     super(data, next);
     this.previous = previous;
  }
}
```



LISTA (IMPLEMENTACAO)

- Como seria a implementação de uma lista duplamente ligada?

```
public interface DoubleLinkedList<T> {
 public boolean isEmpty();
 public int size();
 public T search(T element);
 public void insert(T element);
 public void remove(T element);
 public T[] toArray();
 public void insertFirst(T element);
 public void removeFirst();
 public void removeLast();
}
```

LISTA (IMPLEMENTACAO)

- Como seria a implementação de uma lista duplamente ligada?

```
public interface DoubleLinkedList<T> {
 public boolean isEmpty();
 public int size();
 public T search(T element);
 public void insert(T element);
 public void remove(T element);
 public T[] toArray();
 public void insertFirst(T element);
 public void removeFirst();
 public void removeLast();
}
```

Esses métodos já existem em algum lugar?

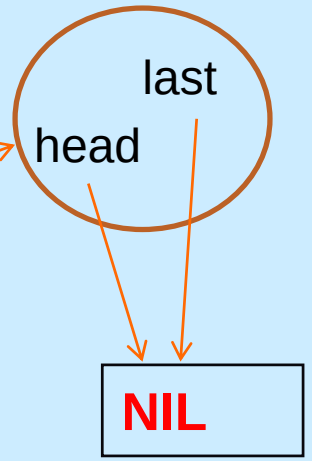
LISTA (IMPLEMENTACAO)

- Como seria a implementação de uma lista duplamente ligada?

```
public interface DoubleLinkedList<T> extends LinkedList<T> {

  public void insertFirst(T element);
  public void removeFirst();
  public void removeLast();
}
```

LISTA (IMPLEMENTACAO)

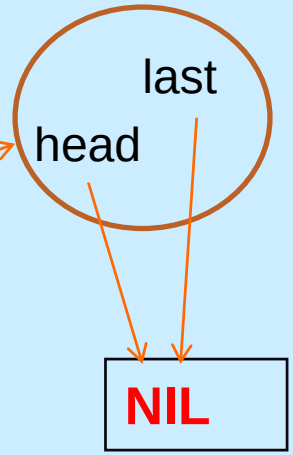
```
public class DoubleLinkedListImpl<T>
  extends SingleLinkedListImpl<T>
    implements DoubleLinkedList<T> {

  DoubleLinkedListNode<T> last;

  public DoubleLinkedListImpl() {
    head = new DoubleLinkedListNode();
    last = head;
  }
}
```



Algum método herdado precisa ser sobrescrito?

LISTA (IMPLEMENTACAO)

```
public class DoubleLinkedListImpl<T>
  extends SingleLinkedListImpl<T>
    implements DoubleLinkedList<T> {

  DoubleLinkedListNode<T> last;

  public DoubleLinkedListImpl() {
    head = new DoubleLinkedListNode();
    last = head;
  }
}
```

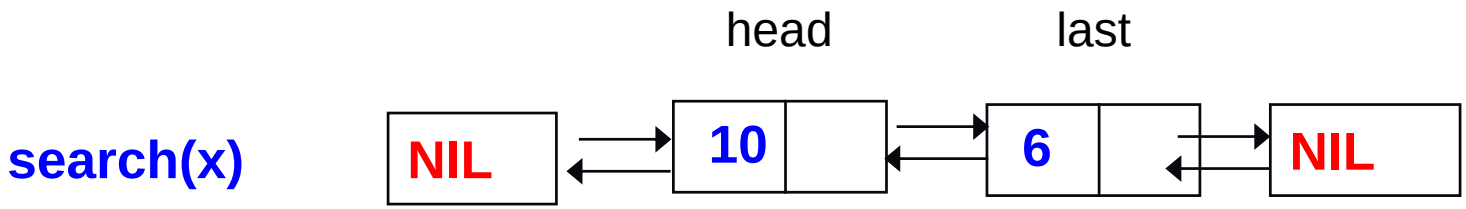


Algum método herdado precisa ser sobrescrito?

search (por otimização),insert,remove

LISTA DUPLA(SEARCH)

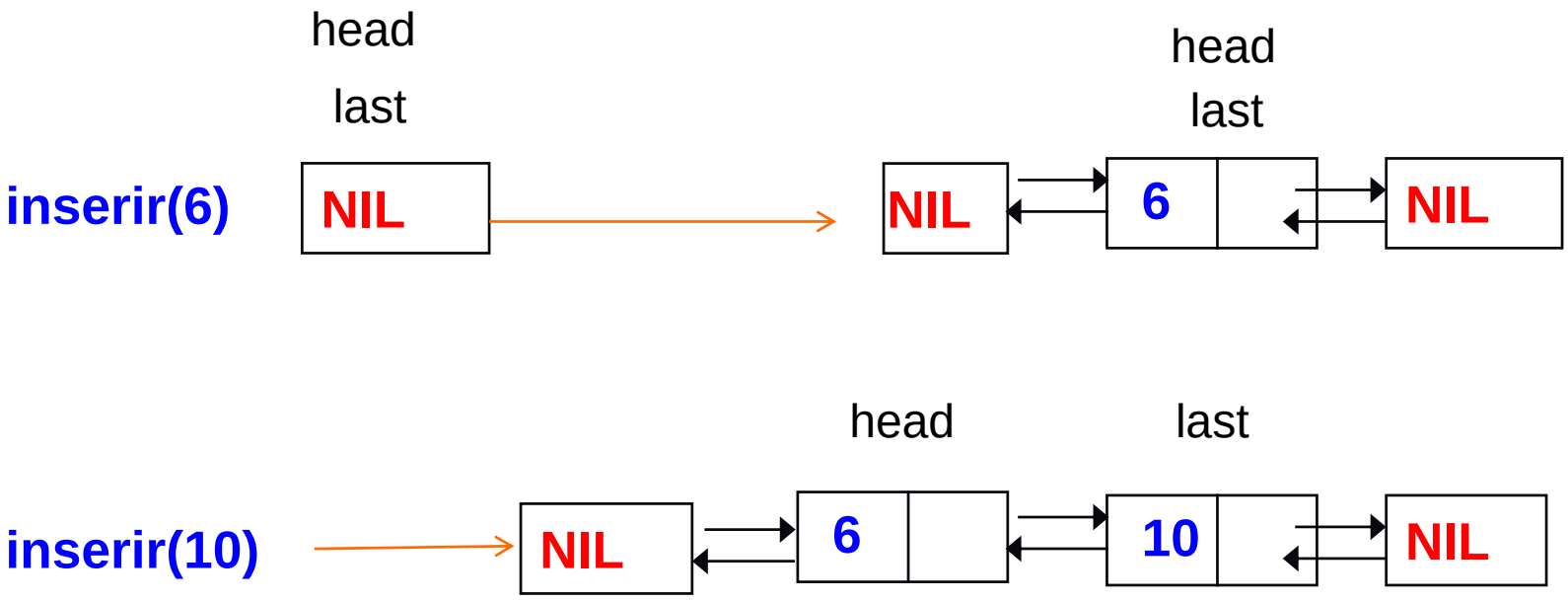
- Pelos dois sentidos: head e last



Lista Dupla(Implementação)

```
//busca (com otimização)
List-search(item){
  auxHead = head
  auxLast = last
  while(auxHead != auxLast and auxHead.next != auxLast
          and auxHead.data != item and auxLast.data != item){
    auxHead = auxHead.next
    auxLast = auxLast.previous
  }
  if auxHead.data == item
     return auxHead.data
  if auxLast.data == item
     return auxLast.data
}
```

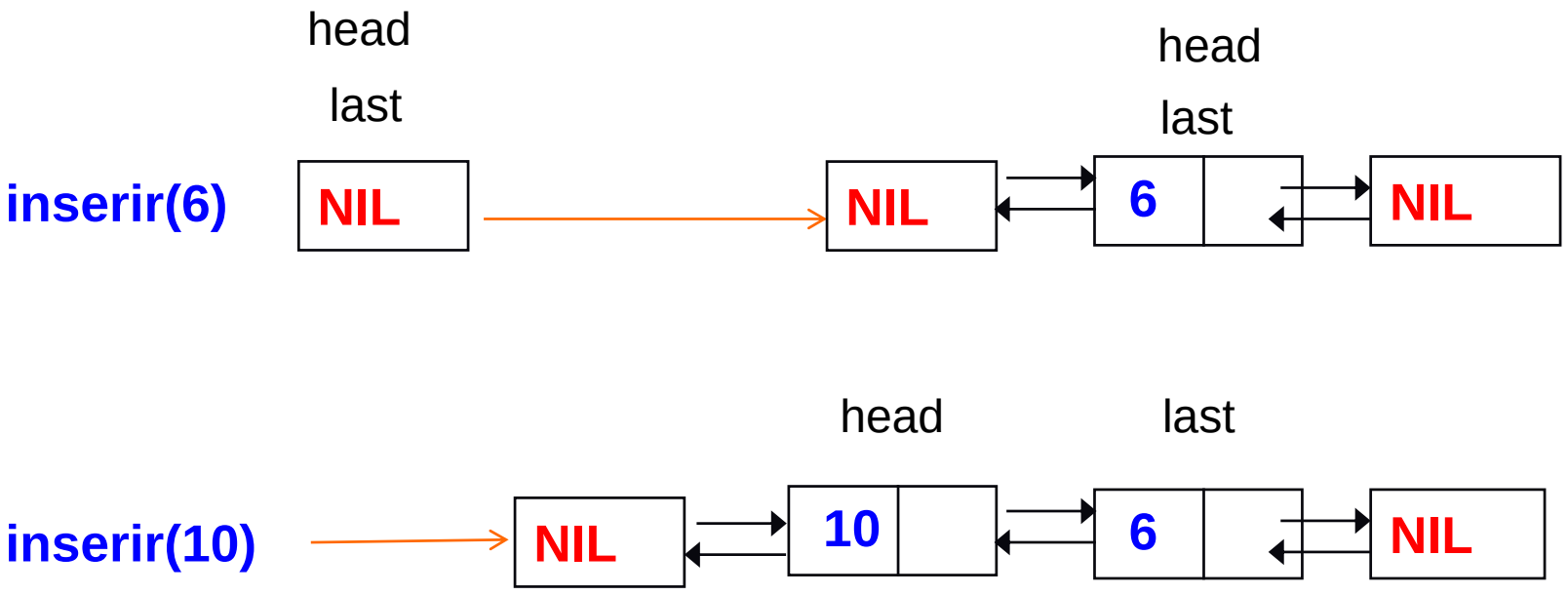
Lista Dupla(Inserir no final= default)



Lista Dupla(Inserir no final = default)

```
//insere no final e ajusta os links next e previous
List-insert-last(item){
  newLast = new DoubleLinkedListNode(item)
  newLast.previous = last
  newLast.next = NIL
  last.next = newLast
  if last == NIL{ //para o caso da lista vazia
    head = newLast
  }
  last = newLast
}
```

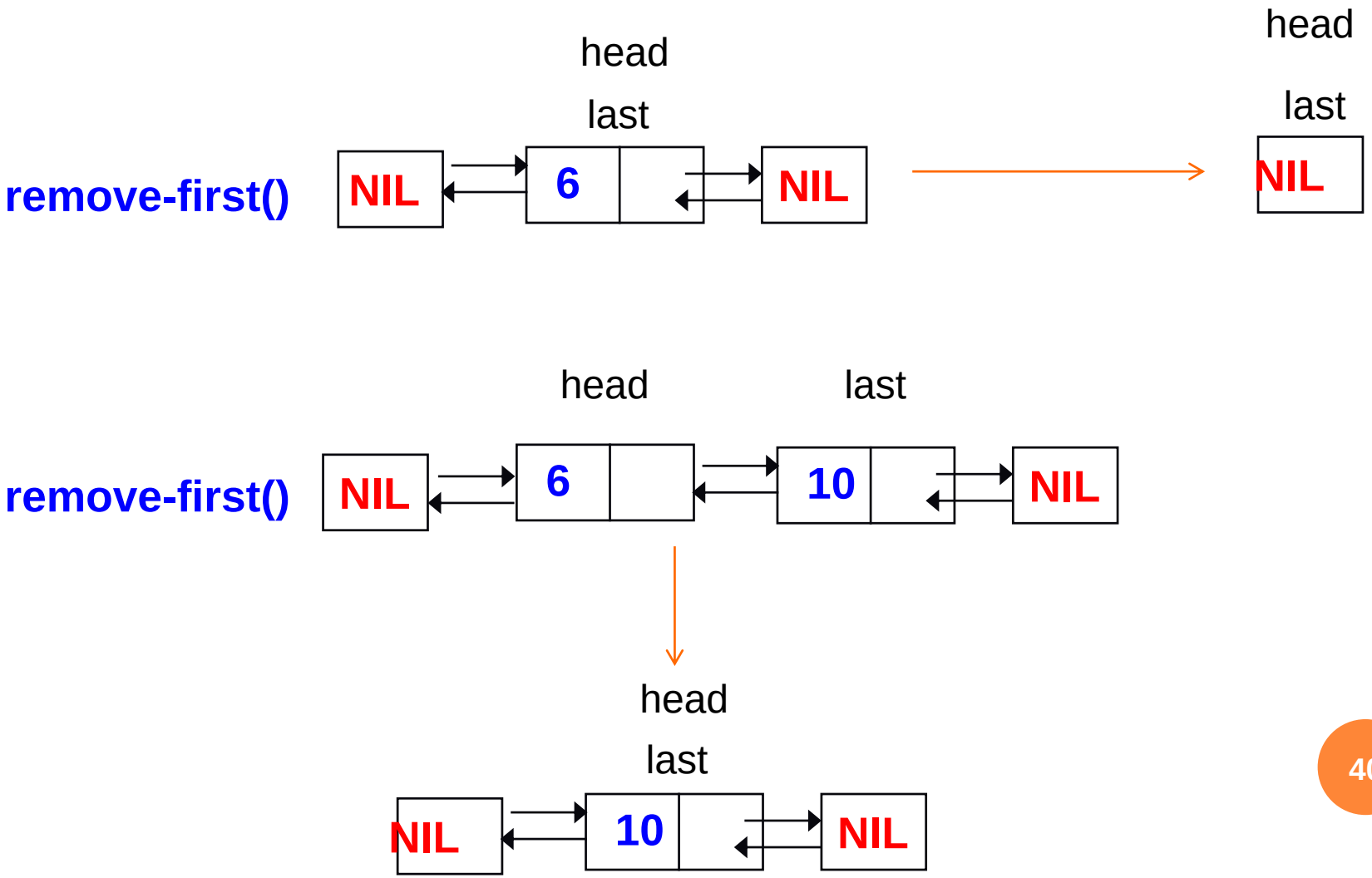
Lista Dupla(Inserir no início)



Lista Dupla(Inserir no início)

```
//insere no começo e ajusta os links next e previous
List-insert(item){
  newHead = new DoubleLinkedListNode(item)
  newHead.next = head
  newHead.previous = NIL
  head.previous = newHead
  if head == NIL{  //para o caso da lista vazia
    last = newHead
  }
 head = newHead
}

List-insert-first(item){
  List-insert(item)
}
```

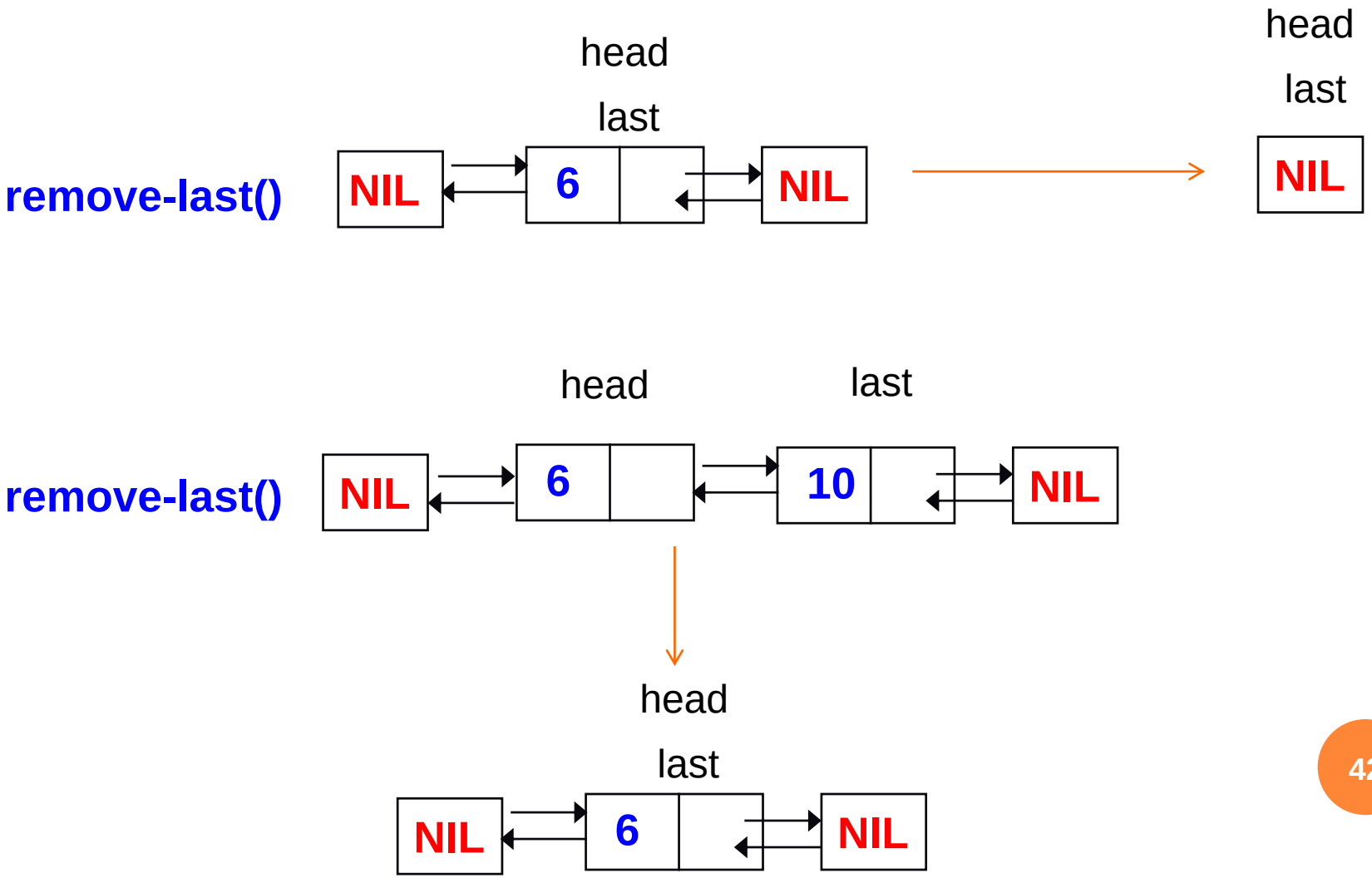
Lista Dupla(Remover no inicio)



Lista Dupla(Remover no inicio)

```
//remover do inicio
List-remove-first(){
  if head != NIL {
    head = head.next
    if head == NIL{
      last = head
    }
    head.previous = NIL
  }
}
```

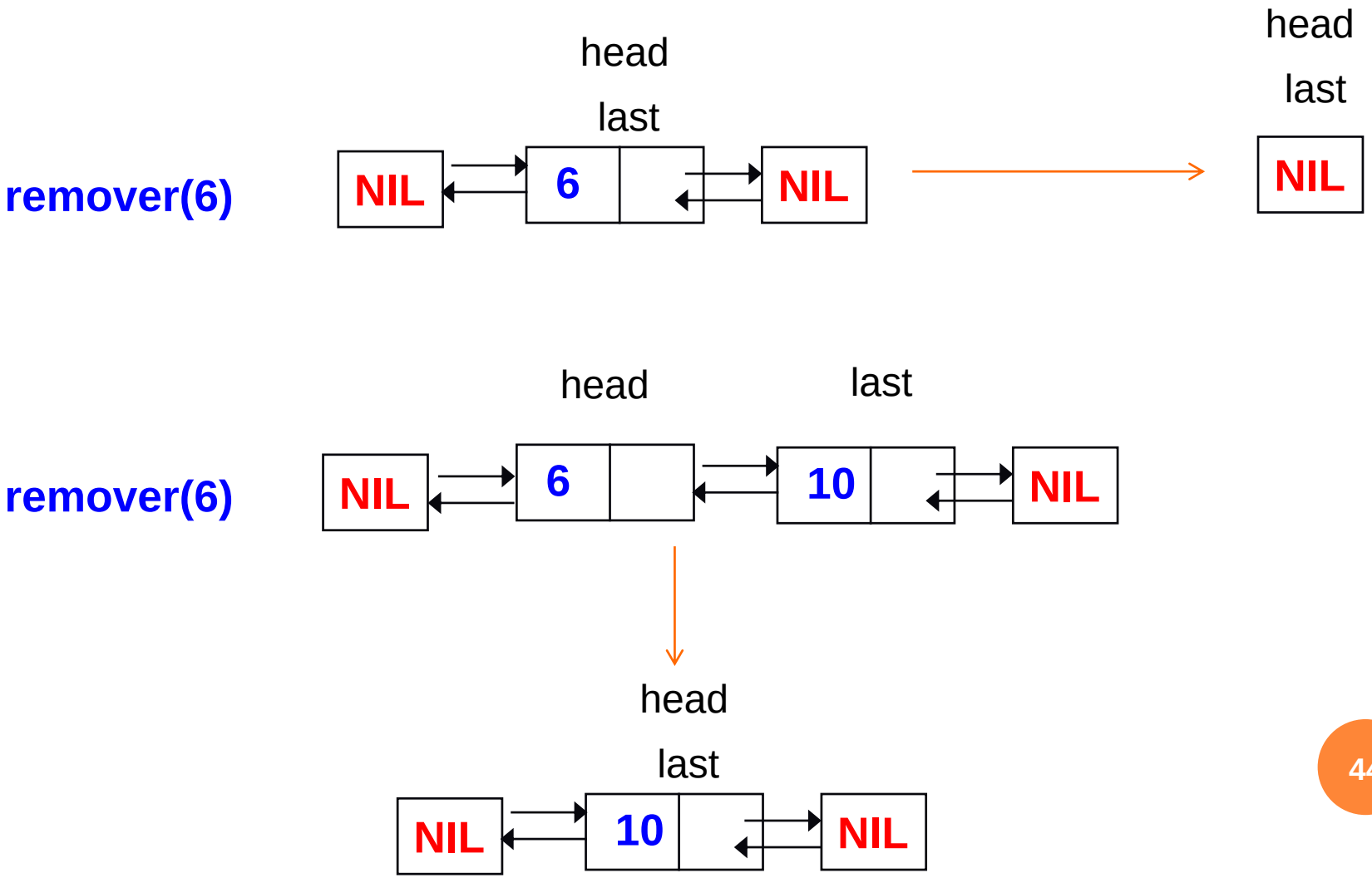
Lista Dupla(Remover no final)



Lista Dupla(Remover no final)

```
//remover do final
List-remove-last(){
  if last != NIL {
    last = last.previous
    if last == NIL{
      head = last
    }
    last.next = NIL
  }
}
```

Lista Dupla(Remover pela Chave = default)



Lista Dupla(Remover pela Chave = default)

```
list-remove(item){
  if head.data == item
     remove-first()
  else{
    aux = head
    while aux != NIL and aux.data != item {
      aux = aux.next
    }
    if aux != NIL{
     aux.previous.next = aux.next
     aux.next.previous = aux.previous
  }
```

Referências

- Capítulo 11