Estrutura de Dados

Aula 08

Prof. Luiz Antonio Schalata Pacheco, Dr. Eng.

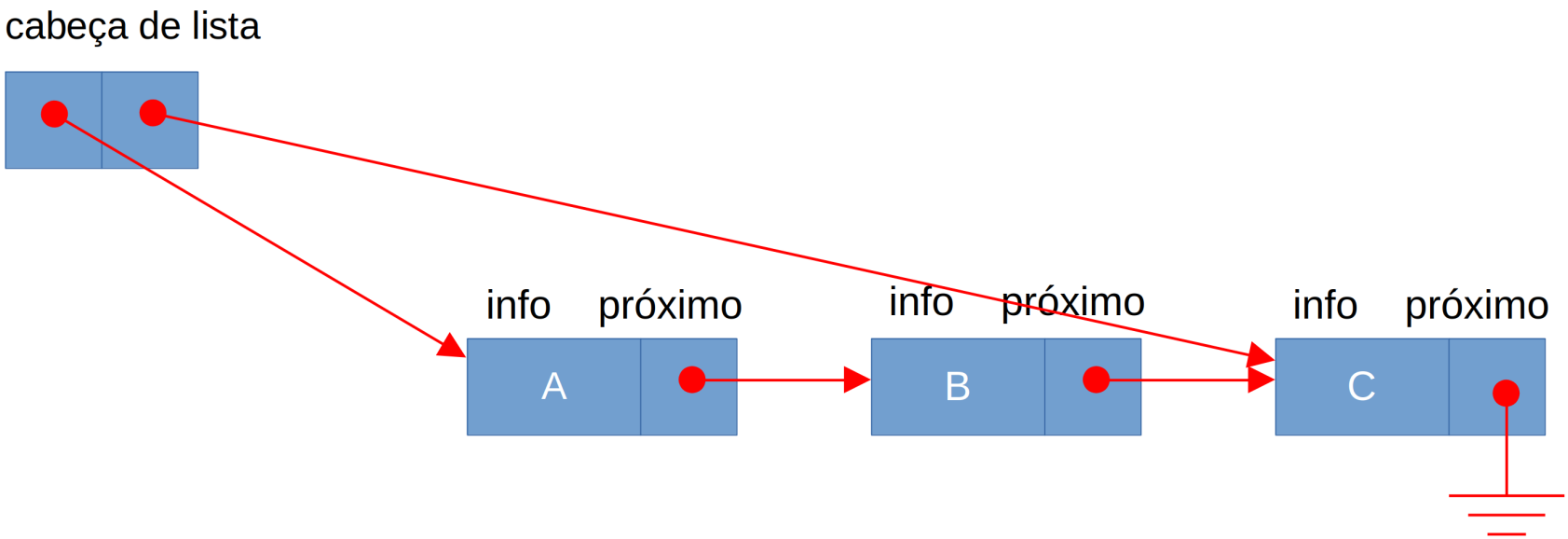
Instituto Federal de Santa Catarina
Câmpus Garopaba
Curso Superior de Tecnologia em Sistemas para Internet

schalata@ifsc.edu.br

11/05/2023

Listas Encadeadas com Extremidade Dupla

Representação



Conceito

- Possui referência para o primeiro e para o último nó
- A cabeça de lista tem mais um atributo
- Permite fazer a inserção no final da lista
- Nas listas encadeadas simples só é possível fazer inserção no início
- Inserções no fim, obrigavam a percorrer todos os elementos da lista
- Mantém-se as operações de inserção e exclusão do início
- É possível excluir diretamente do final?

Análises

- Inserção e exclusão no início e no final (para listas com extremidade duplas!) - O(1)
- Localizar, inserir ou eliminar item específico requer buscar, em média, metade dos itens - O(n)
  - Nesses casos, são semelhantes a vetores, porém não é necessário mover itens
- Otimiza o uso de memória, pois as listas podem ser alocadas de forma dinâmica

Implementação

- Implementar uma Lista Encadeada com Extremidade Dupla. Faça os métodos para inserir no início, inserir no final, excluir do início e mostrar a lista. Faça ainda um método privado para verificar se a lista está vazia.

Implementação: Inserção no início

```
1  class ListaEncadeadaExtremidadeDupla:
2
3    def __init__(self):
4      self.primeiro = None
5      self.ultimo = None
6
7    def __lista_vazia(self):
8      return self.primeiro == None
9
10   def insere_inicio(self, valor):
11     novo = No(valor)
12     if self.__lista_vazia(): # Na primeira insercao
13       self.ultimo = novo  # primeiro e igual ao ultimo
14     novo.proximo = self.primeiro
15     self.primeiro = novo
```


Implementação: Mostrar

```
17   def mostrar(self):
18     if self.__lista_vazia():
19       print('A lista esta vazia')
20       return
21     atual = self.primeiro
22     while atual != None:
23       atual.mostra_no()
24       atual = atual.proximo
```


Implementação: Inserção no final

```
26   def insere_final(self, valor):
27     novo = No(valor)
28     if self.__lista_vazia(): # Na primeira insercao
29       self.primeiro = novo # primeiro e igual ao ultimo
30     else:
31       # O proximo do ultimo elemento aponta para novo
32       self.ultimo.proximo = novo
33     # Atributo ultimo da cabeca de lista tambem aponta
       para o novo, pois ele e o ultimo elemento da lista
34     self.ultimo = novo
```


Implementação: Exclusão do início

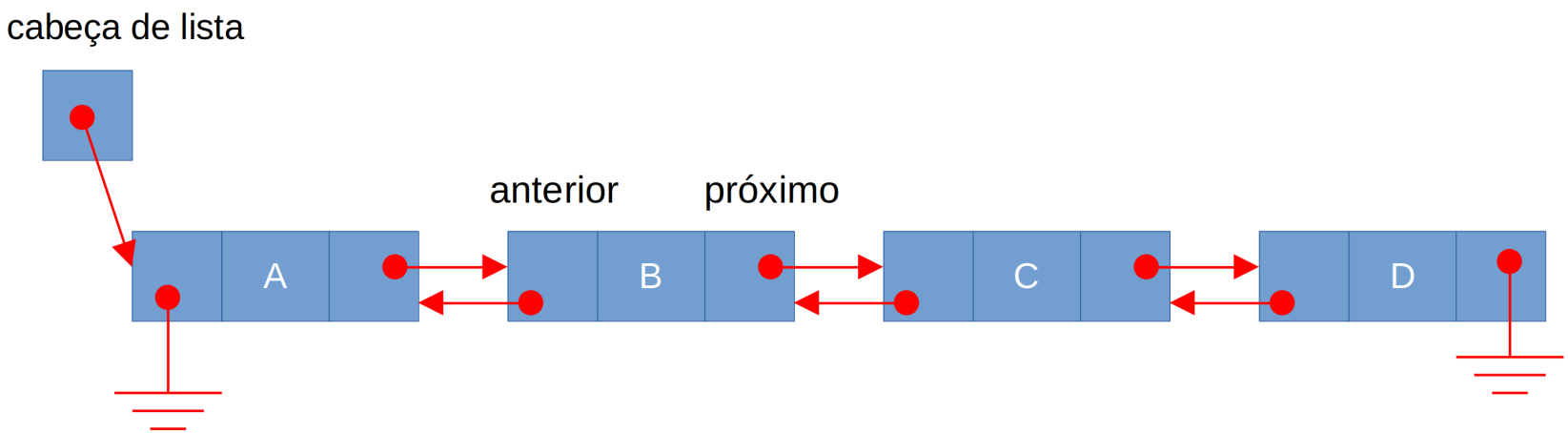
```
36   def excluir_inicio(self):
37     if self.__lista_vazia():
38       print('A lista esta vazia')
39       return
40
41     temp = self.primeiro # elemento a ser apagado
42     if self.primeiro.proximo == None: # Se existe somente
       um elemento na lista
43       self.ultimo = None # Passa a apontar para none,
      indicando que nao existe um ultimo elemento
44     self.primeiro = self.primeiro.proximo
45     return temp
```


Listas Duplamente Encadeadas

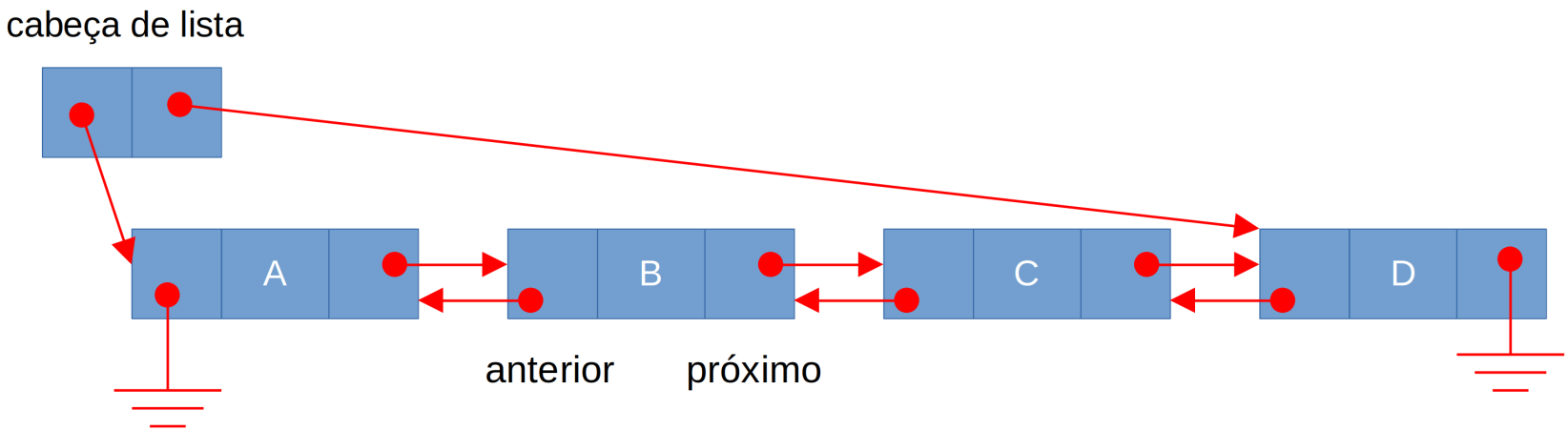
Conceito

- Nas listas encadeadas simples:
  - Não há como voltar ao nó anterior
  - As relações de referência entre os nós são apenas da esquerda para a direita
- As listas duplamente encadeadas:
  - Permitem percorrer a lista para trás ou para frente
  - Cada nó passa a ter duas referências: uma referência para o próximo nó e outra referência para o nó anterior
  - Pode ter extremidade simples ou dupla

Listas Duplamente Encadeadas



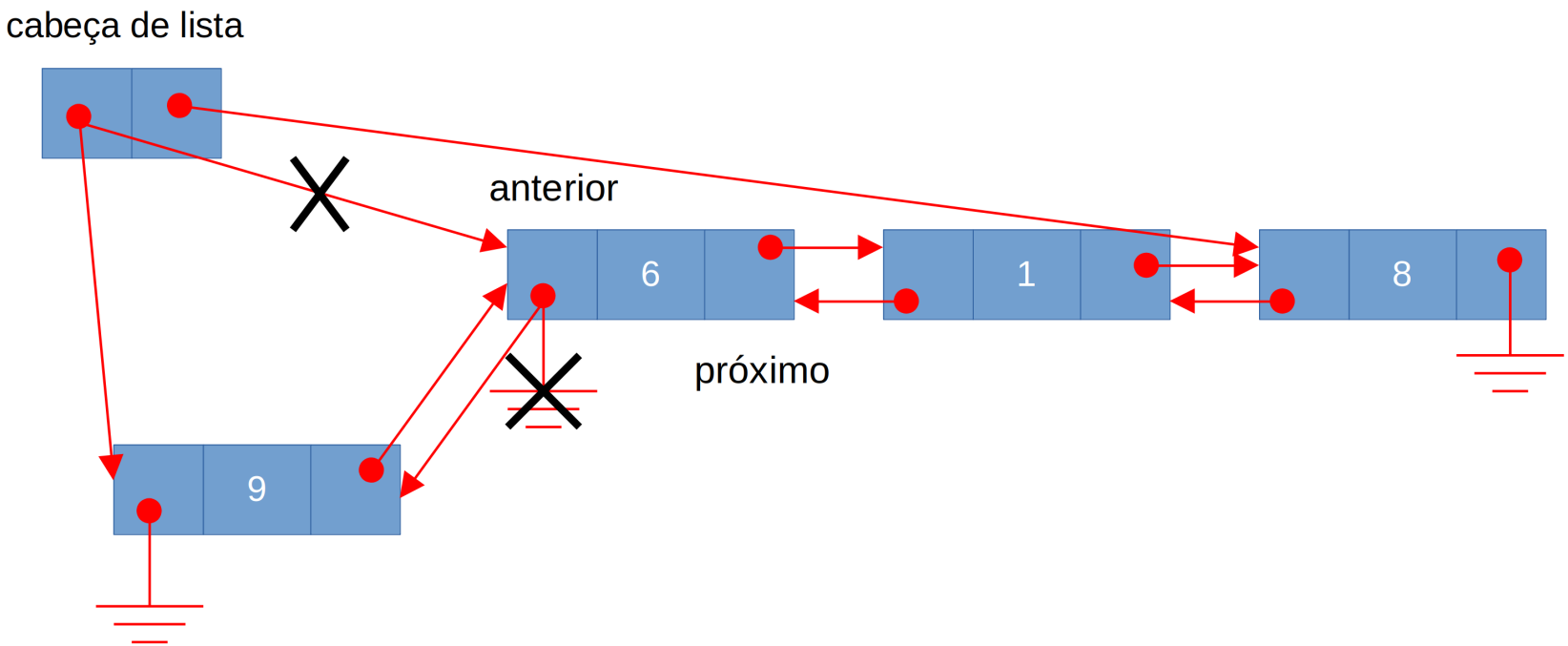
Listas Dup. Encadeadas com Extremidade Dupla



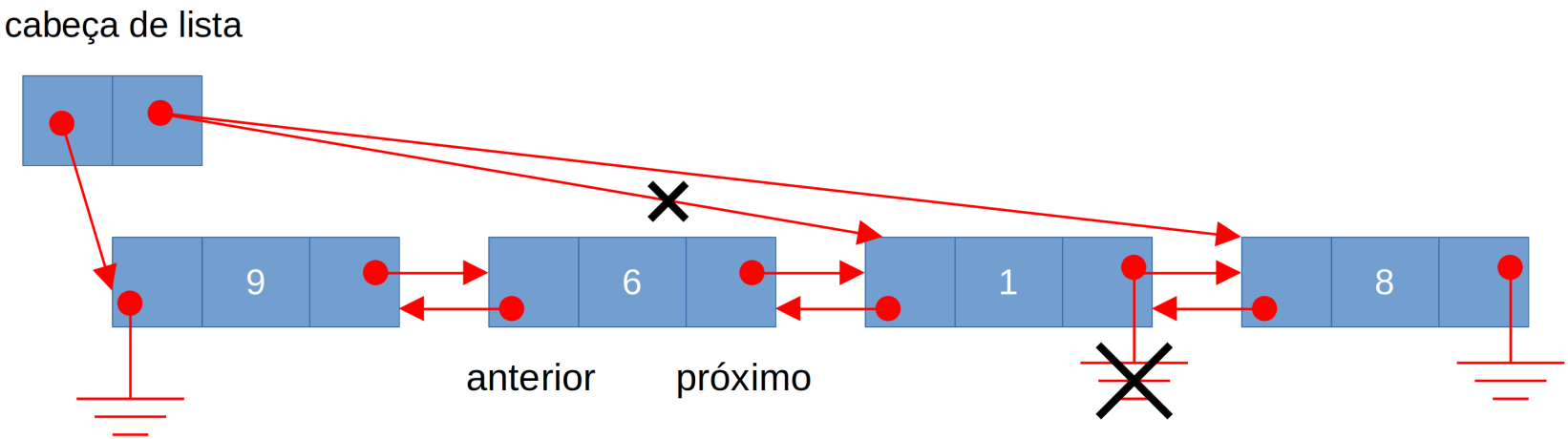
Operações

- Inserção no início
- Inserção no final
- Exclusão no início
- Exclusão do final
- Exclusão da posição

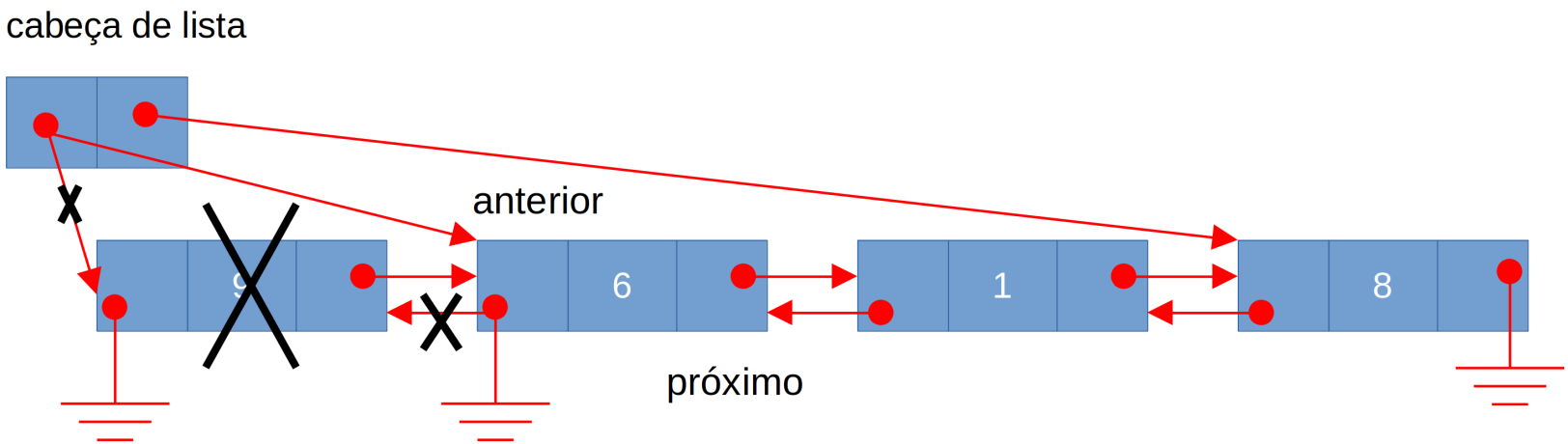
Listas Dup. Encadeadas: Inserção no início



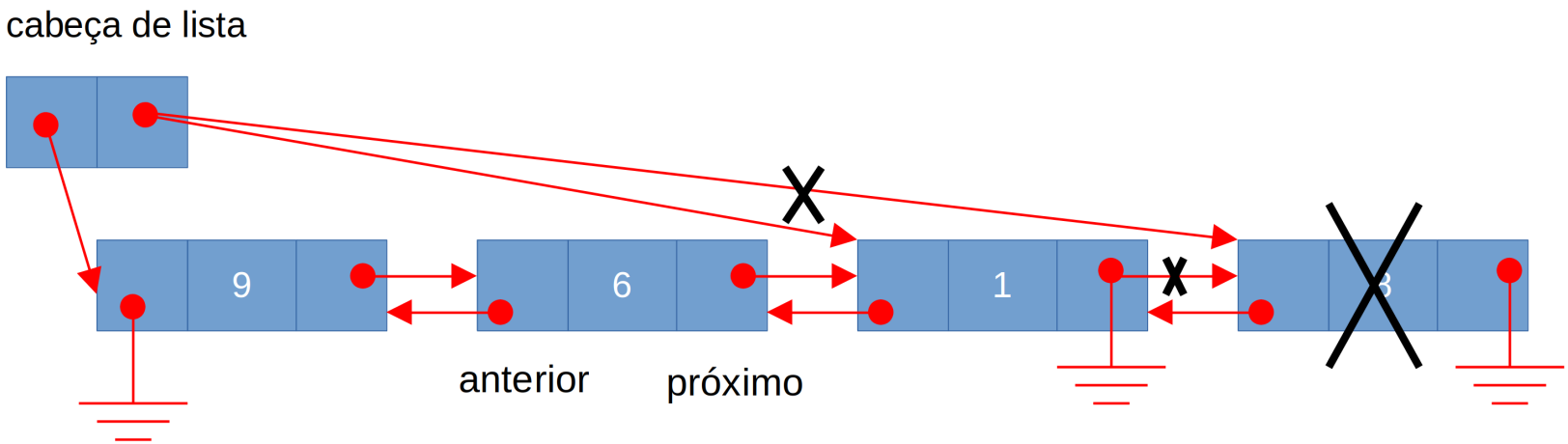
Listas Dup. Encadeadas: Inserção no final



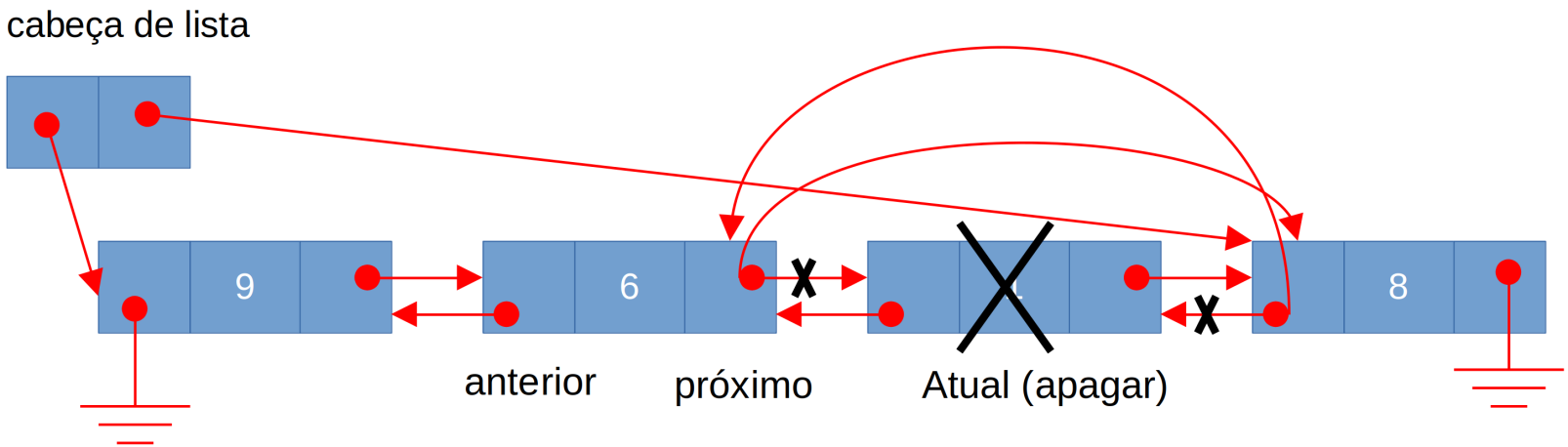
Listas Dup. Encadeadas: Exclusão no início



Listas Dup. Encadeadas: Exclusão do final



Listas Dup. Encadeadas: Exclusão da posição



Implementação

- Implemente uma Lista Duplamente Encadeada. Deve ser possível fazer inserções no início e no final, assim como fazer exclusões do início, do final e da posição. Faça ainda um método privado para verificar se a lista está vazia e outros dois métodos para mostrar a frente e a traseira da lista.

Implementação: Classe Nó

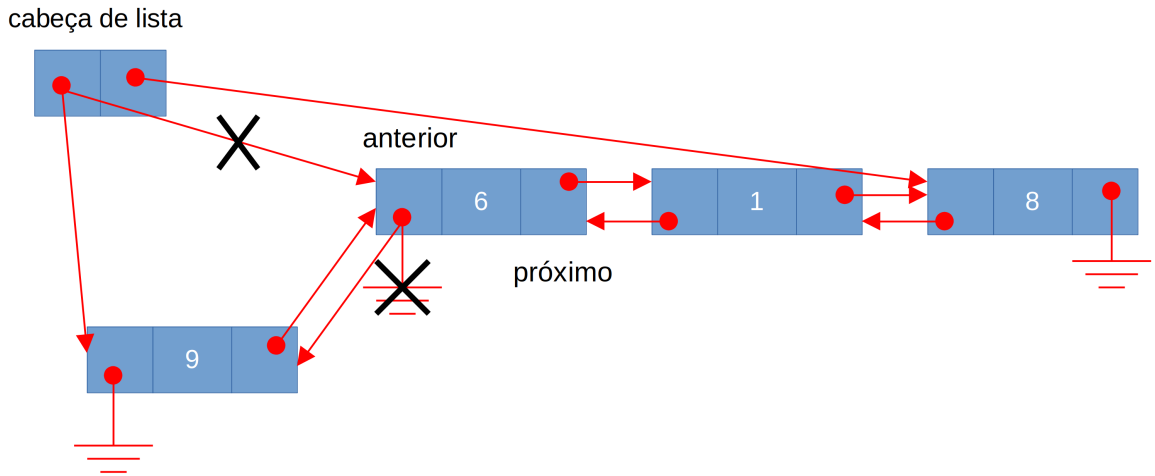
```
1 class No:
2
3   def __init__(self, valor):
4     self.valor = valor
5     self.proximo = None
6     self.anterior = None
7
8   def mostra_no(self):
9     print(self.valor)
```


Implementação: Construtor

```
11 class ListaDuplamenteEncadeada:
12
13   def __init__(self):
14     self.primeiro = None
15     self.ultimo = None
16
17   def __lista_vazia(self):
18     return self.primeiro == None
```

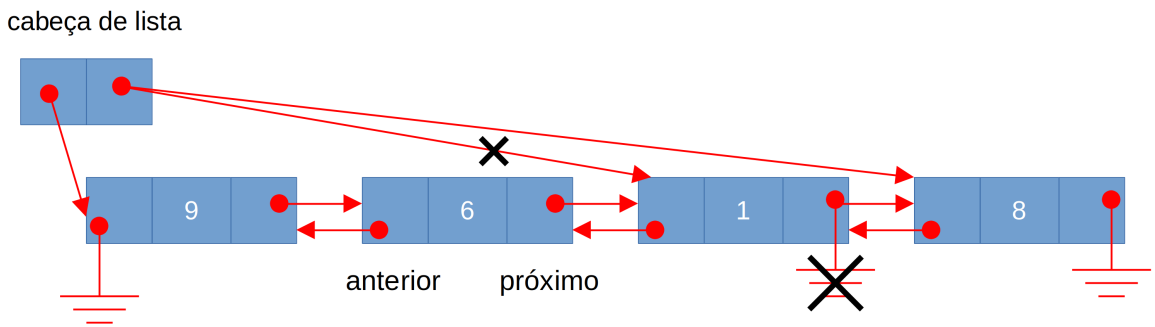

Implementação: Inserção no início

```
20  def insere_inicio(self, valor):
21      novo = No(valor)
22      if self.__lista_vazia():
23          self.ultimo = novo
24      else:
25          self.primeiro.anterior = novo
26      novo.proximo = self.primeiro
27      self.primeiro = novo
```



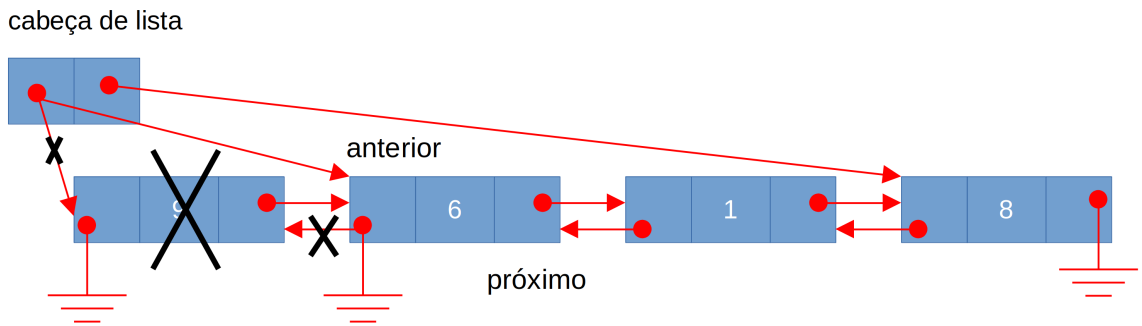
Implementação: Inserção no final

```
29   def insere_final(self, valor):
30     novo = No(valor)
31     if self.__lista_vazia():
32       self.primeiro = novo
33     else:
34       self.ultimo.proximo = novo
35       novo.anterior = self.ultimo
36     self.ultimo = novo
```



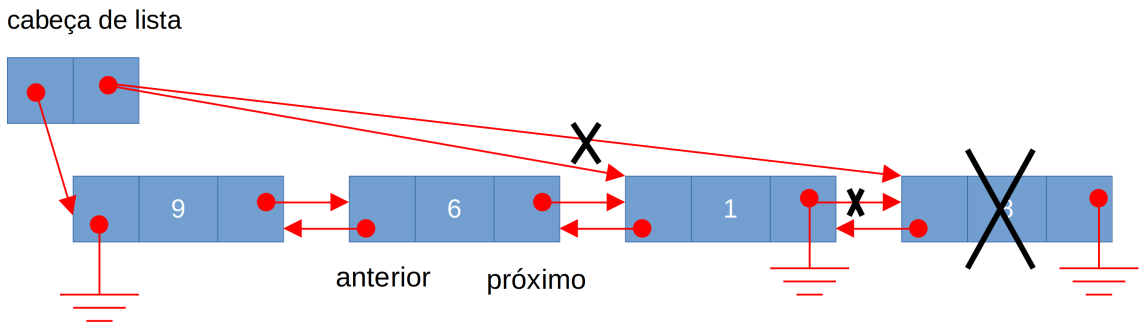
Implementação: Exclusão do início

```
38  def excluir_inicio(self):
39      temp = self.primeiro
40      if self.primeiro.proximo == None:
41          self.ultimo = None
42      else:
43          self.primeiro.proximo.anterior = None
44      self.primeiro = self.primeiro.proximo
45      return temp
```



Implementação: Exclusão do final

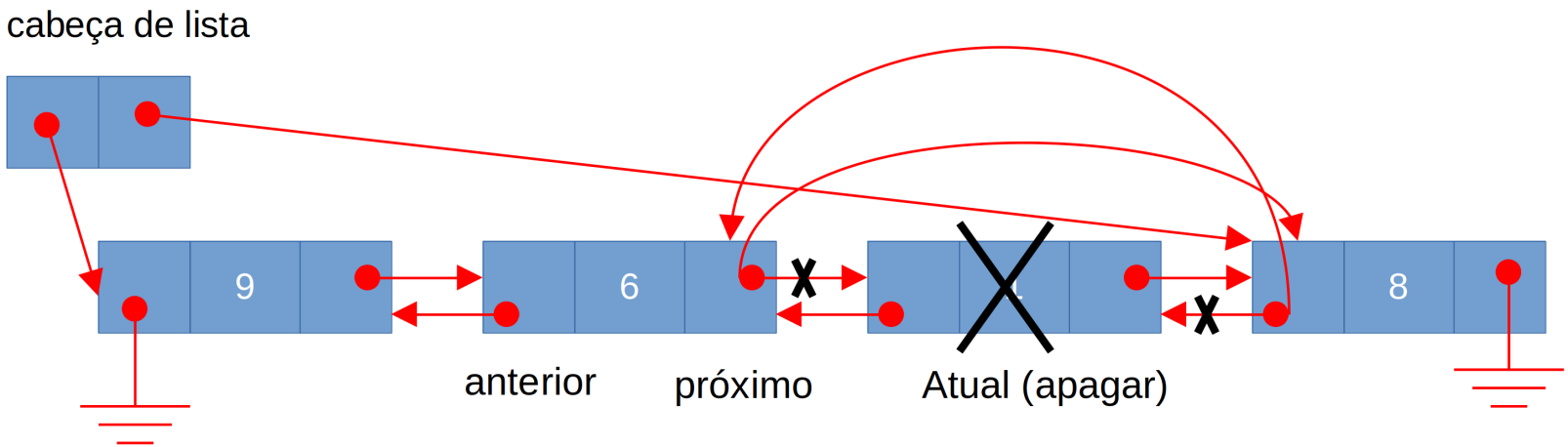
```
47    def excluir_final(self):
48        temp = self.ultimo
49        if self.primeiro.proximo == None:
50            self.primeiro = None
51        else:
52            self.ultimo.anterior.proximo = None
53        self.ultimo = self.ultimo.anterior
54        return temp
```



Implementação: Exclusão da posição

```
56    def excluir_posicao(self, valor):
57      atual = self.primeiro
58      while atual.valor != valor:
59        atual = atual.proximo
60        if atual == None:
61          return None
62      if atual == self.primeiro:
63        self.primeiro = atual.proximo
64      else:
65        atual.anterior.proximo = atual.proximo
66
67      if atual == self.ultimo:
68        self.ultimo = atual.anterior
69      else:
70        atual.proximo.anterior = atual.anterior
71      return atual
```


Listas Dup. Encadeadas: Exclusão da posição



Implementação: Mostrar frente e traseira

```
73   def mostrar_frente(self):
74     atual = self.primeiro
75     while atual != None:
76       atual.mostra_no()
77       atual = atual.proximo
78
79   def mostrar_tras(self):
80     atual = self.ultimo
81     while atual != None:
82       atual.mostra_no()
83       atual = atual.anterior
```