Listas encadeadas

FURB
UNIVERSIDADE DE BLUMENAU

Tópicos

- Motivação
- Conceitos
- Listas circulares
- Listas duplamente encadeadas

Listas com implementação estática

- Benefícios
  - Acesso aleatório a qualquer elemento
- Deficiências:
  - Desperdício de espaço
  - Operações de remoção e inclusão no início da estrutura são trabalhosas
  - Realocação do vetor também é trabalhosa

Solução

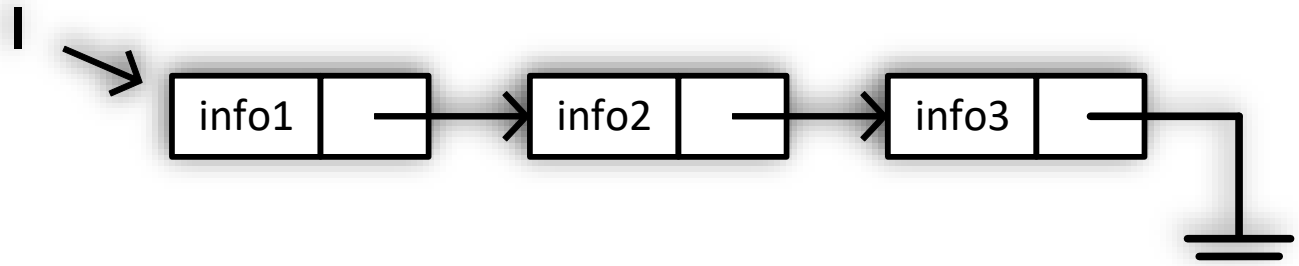
- Estruturas de dados dinâmicas
  - Crescem à medida que os elementos são inseridos
  - Decrescem à medida que os elementos são removidos
- Solução:
  - Listas encadeadas

Listas encadeadas

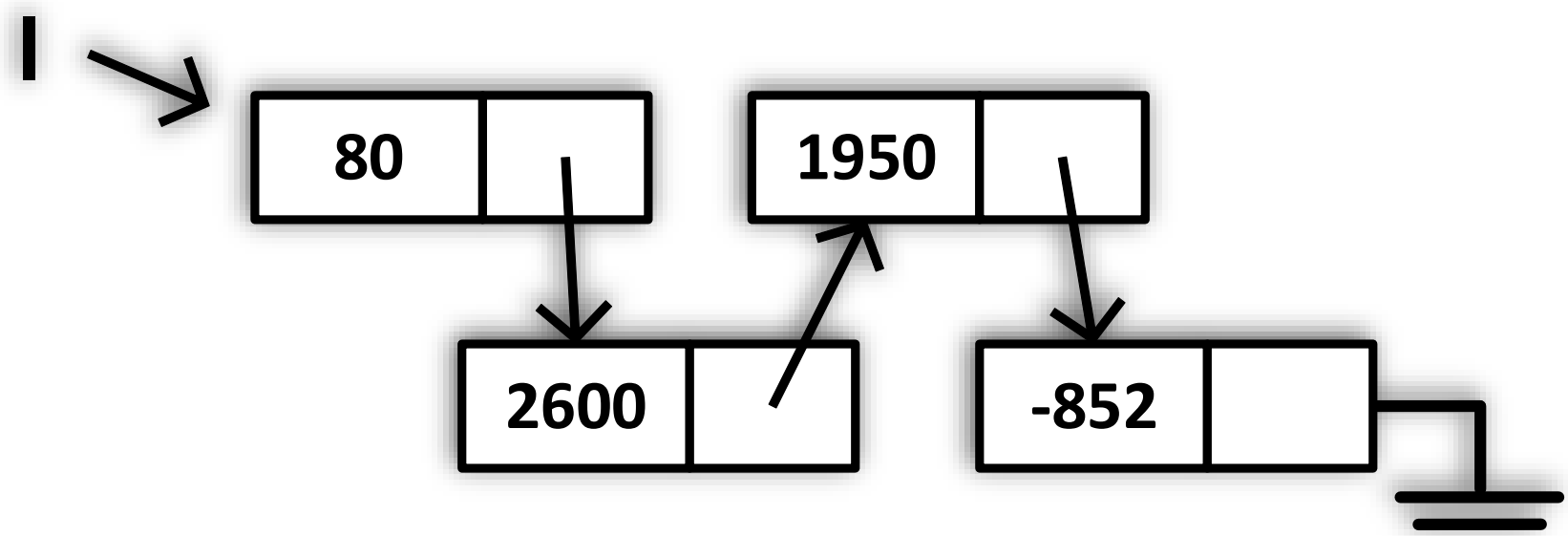
- Sequencia de elementos encadeados;
- Cada elemento é denominado de “nó da lista”, “nodo” ou “célula”;
- Cada nó da lista contém:
  a) O dado que se quer armazenar
  b) Uma referência para o próximo elemento da lista

Representação gráfica do nó:

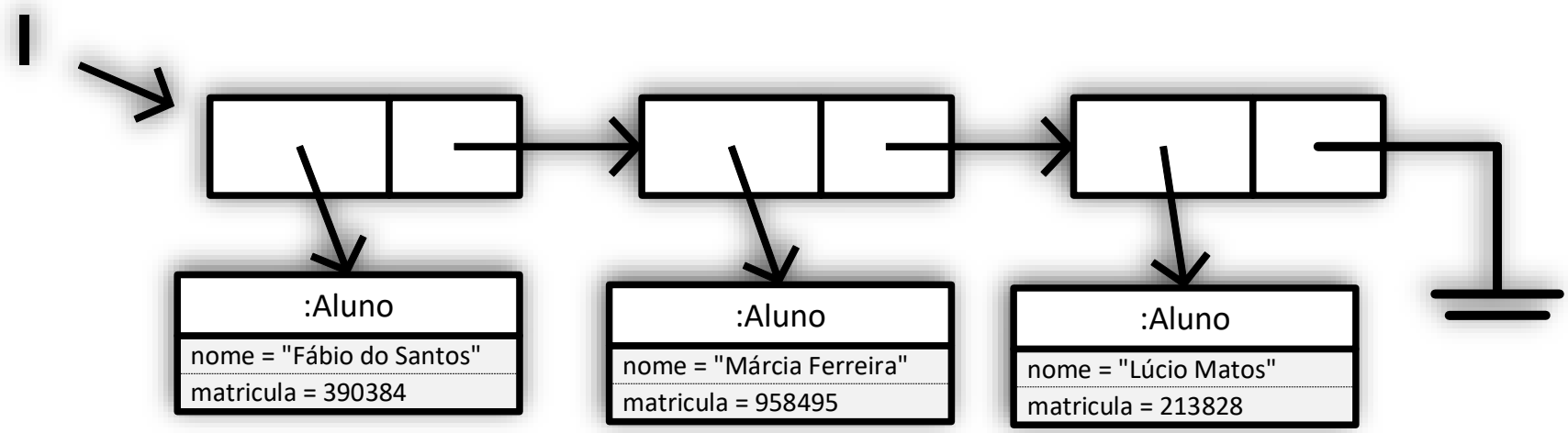
- A lista possui uma referência para o primeiro nó;
- A referência do último nó é **`null`**



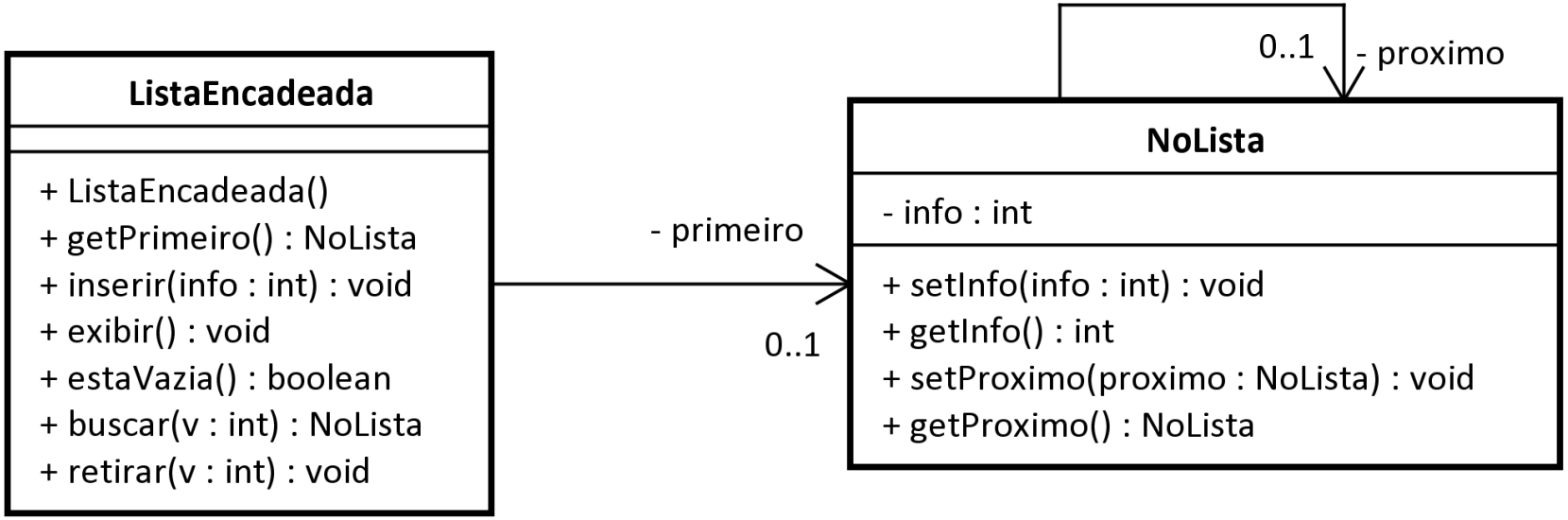
Exemplo de listas encadeada para armazenar dados primitivos



Exemplo de lista encadeada para armazenar objetos



Exemplo de projeto para construção de lista encadeada **para armazenar "inteiros"**



Classe `NoLista`

- Classe `NoLista` utilizada para representar os nós da lista encadeada

```
1 public class NoLista {
2
3     private int info;
4     private NoLista proximo;
5
6 }
```

- Possui uma associação reflexiva que é responsável por apontar para o próximo nó da lista.

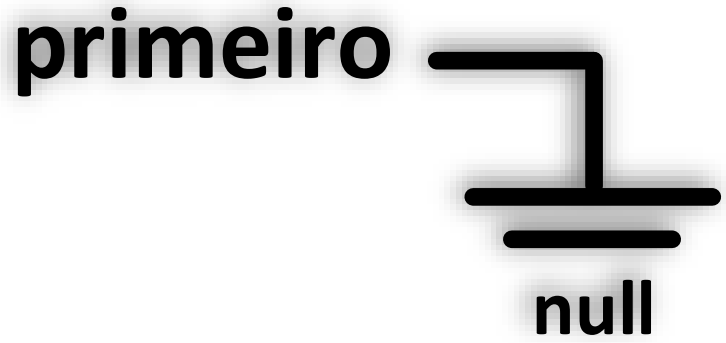
Classe `ListaEncadeada`

- Um objeto da classe `Lista` deve ser responsável por:
  - Referenciar o primeiro nó da estrutura de dados
  - Manipular os elementos da lista

```
public class ListaEncadeada {

    private NoLista primeiro;

}
```

Listas encadeada

- Método para construir uma lista encadeada
  - Deve:
    - Cria uma lista vazia



Algoritmo: **ListaEncadeada()**

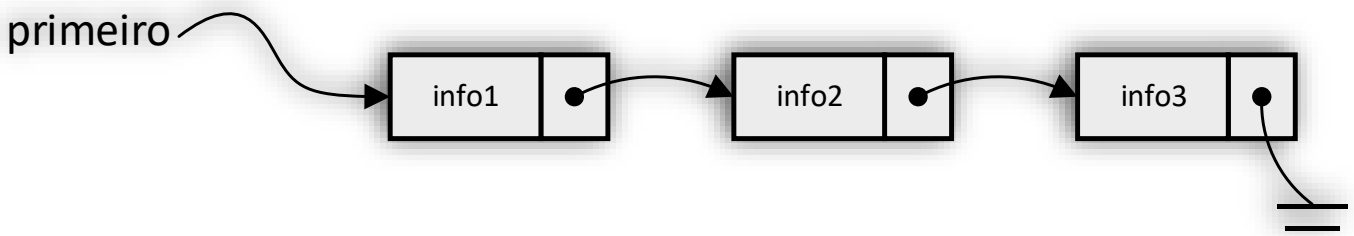
primeiro ← null**;**

Manipulação de listas encadeadas

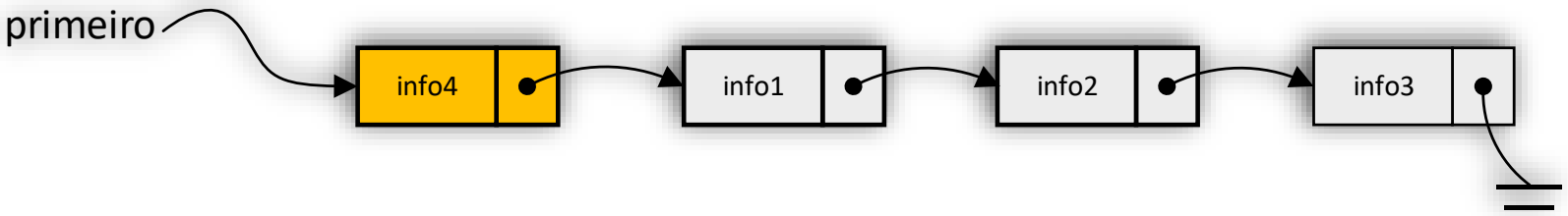
FURB
UNIVERSIDADE DE BLUMENAU

Inclusão de um novo elemento na lista

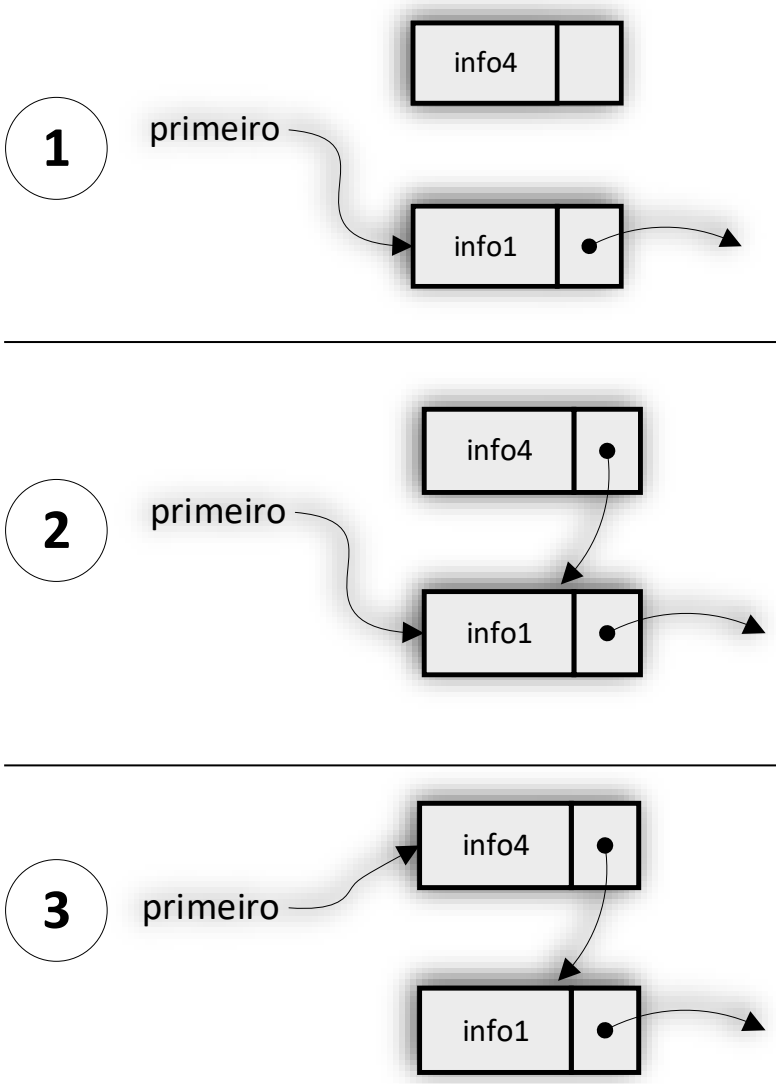
Dada a lista:



Ao executar: **inserir(info4)** deve causar:



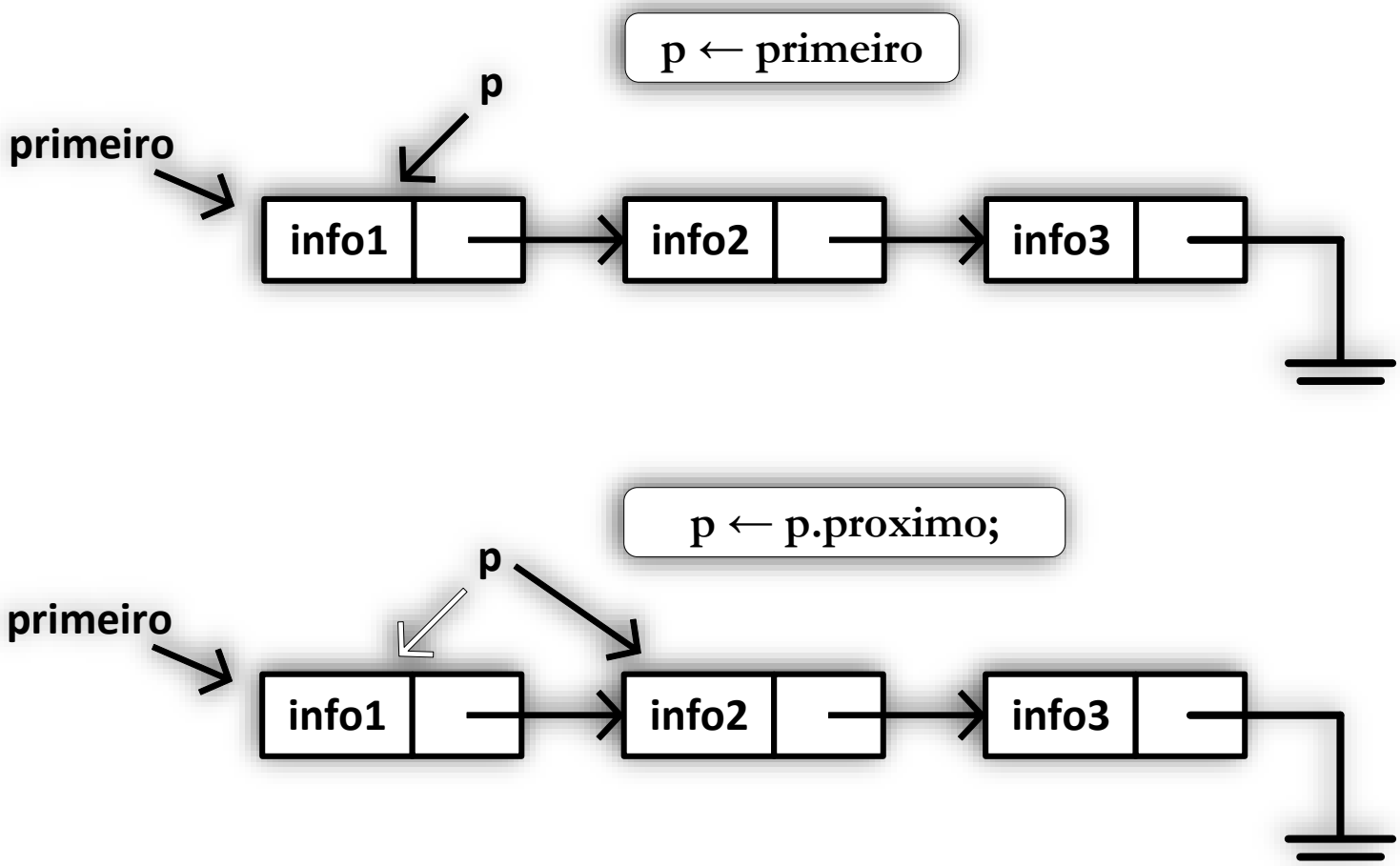
Inclusão de um novo elemento na lista



Algoritmo: inserir(int info)

NoLista novo ← new NoLista( );
novo.info ← info;
novo.proximo ← primeiro;
this.primeiro ← novo;

Exibir o conteúdo da lista encadeada



Exibir conteúdo da lista encadeada

- A variável **p** armazena a referência de um dos nós da lista.

```
Algoritmo: exibir()

NoLista p ← primeiro;
enquanto p ≠ null faça
   print(p.info);
   p ← p.proximo;
fim-enquanto;
```

- Operações realizadas na lista:
  - **Visitar um nó da lista**: acessar um nó da lista
  - **Percorrer a lista**: visitar todos os nós da lista
- O *loop* percorre a lista

Algoritmo para identificar se a lista está vazia

- Algoritmo que retorna **`verdadeiro`** se a lista está vazia, caso contrário, retorna **`falso`**.

```
Algoritmo: estaVazia()

se primeiro = null então
   retornar verdadeiro;
senão
   retornar falso;
fim-se;
```

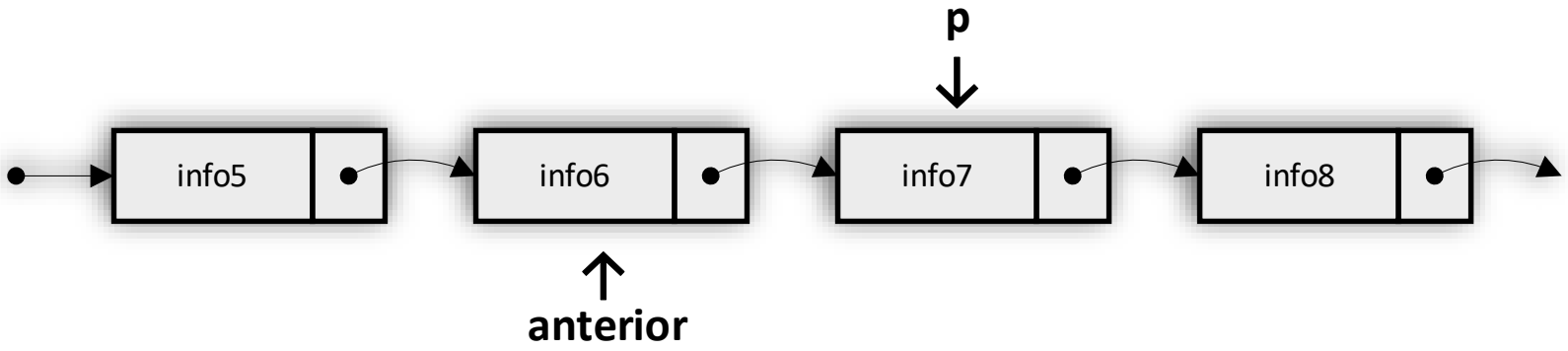
Buscar elemento na lista

- Algoritmo que percorre a lista em busca de um determinado valor armazenado na lista.
- Retorna o nó que contém o valor
- Caso não encontrar, retorna `null`

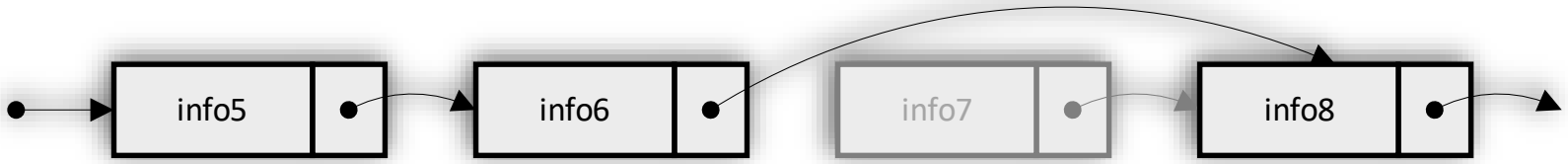
```
Algoritmo: buscar(int valor)

NoLista p ← primeiro;
enquanto (p ≠ null) faça
  se p.info = valor então
     retornar p;
  fim-se;
  p ← p.proximo;
fim-enquanto;
retornar null;
```

Retirar um elemento da lista

1 Dado um valor, procura um nó na lista que contém o valor:



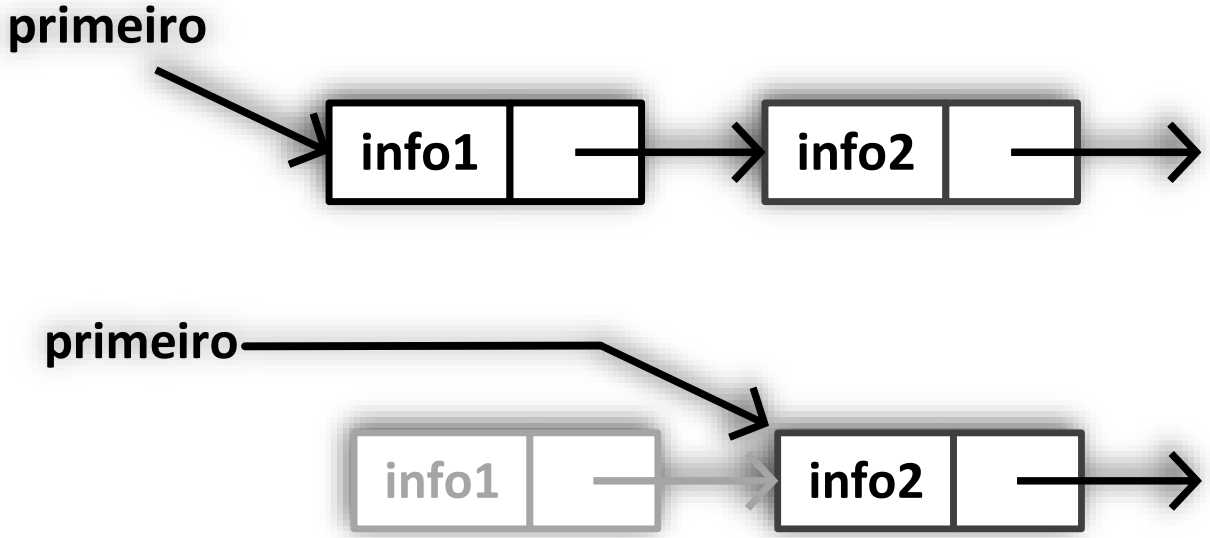
Fazer com que o *no anterior* referencie o *próximo do nó* a ser removido



anterior.proximo ← p.proximo

Caso especial: Retirar o primeiro nó da lista

Considerar que o nó a ser removido contém “info1”



Não pode executar: `anterior.proximo ← p.proximo`

Retirar um nó da lista

Algoritmo: retirar(int valor)

```
NoLista anterior ← null;
NoLista p ← primeiro;

// procura nó que contém dado a ser removido,
// guardando o anterior
enquanto (p ≠ null) e (p.info ≠ valor) faça
   anterior ← p;
   p ← p.proximo;
fim-enquanto;

// Se achou nó, retira-o da lista
se (p ≠ null) então
    se p = primeiro então
        this.primeiro ← p.proximo;
    senão
        anterior.proximo ← p.proximo;
    fim-se;
fim-se;
```