

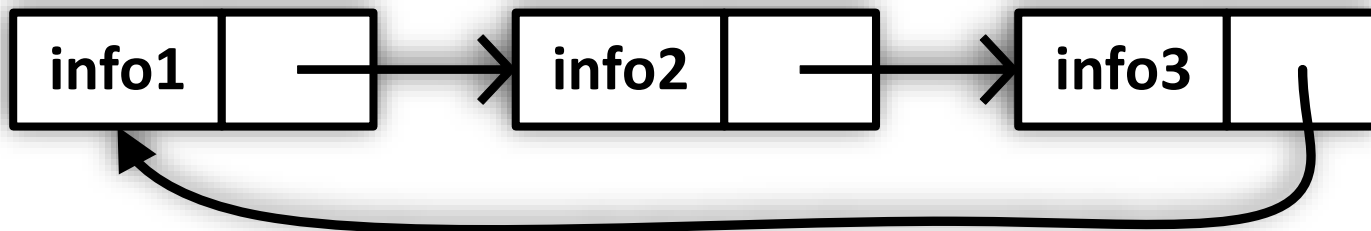
Outras variações de listas encadeadas

Outras variações de listas encadeadas

- Listas circulares
- Listas com acesso às duas extremidades
- Listas duplamente encadeadas

Listas circulares

- O último nó da lista tem como “seu próximo” o primeiro nó da lista, formando um ciclo



- Benefício:
 - Maior legibilidade em algoritmos que percorrem infinitamente uma estrutura de dados

Listas circulares

Impacto nos algoritmos

- Necessário escrever novos algoritmos para impedir execução infinita
- Exemplo: Exibir o conteúdo da lista

Algoritmo: `exibirCircular()`

NoLista $p \leftarrow \text{primeiro}$;

se ($p \neq \text{null}$) **então**

repita

`escrever(p.info);`

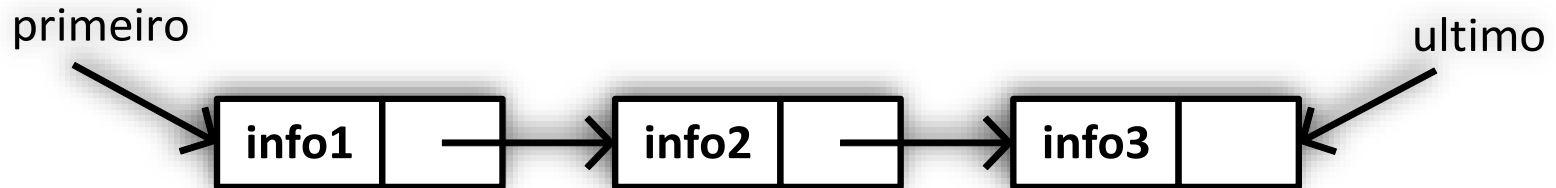
$p \leftarrow p.\text{proximo}$;

enquanto ($p \neq \text{primeiro}$);

fim-se;

- Percorre todos os nós a partir da referência do nó inicial até alcançar novamente este mesmo nó
- Se a lista é vazia, a referência para o nó inicial é nula

Listas com acesso as duas extremidades

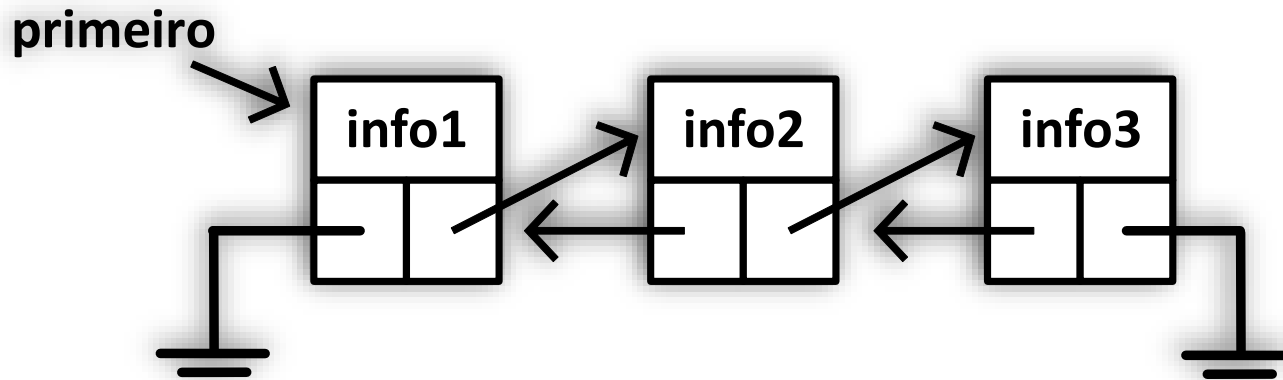


- Benefício:
 - Permite estabelecer outra ordem de inclusão de elementos

Listas duplamente encadeadas

Lista duplamente encadeada

- Cada nó possui uma referência para o próximo nó e para o nó anterior



- Dado um determinado nó, é possível ter acesso ao seu próximo nó e ao nó imediatamente anterior;
- Dada uma referência para o último nó da lista, é possível percorrer a lista em ordem inversa

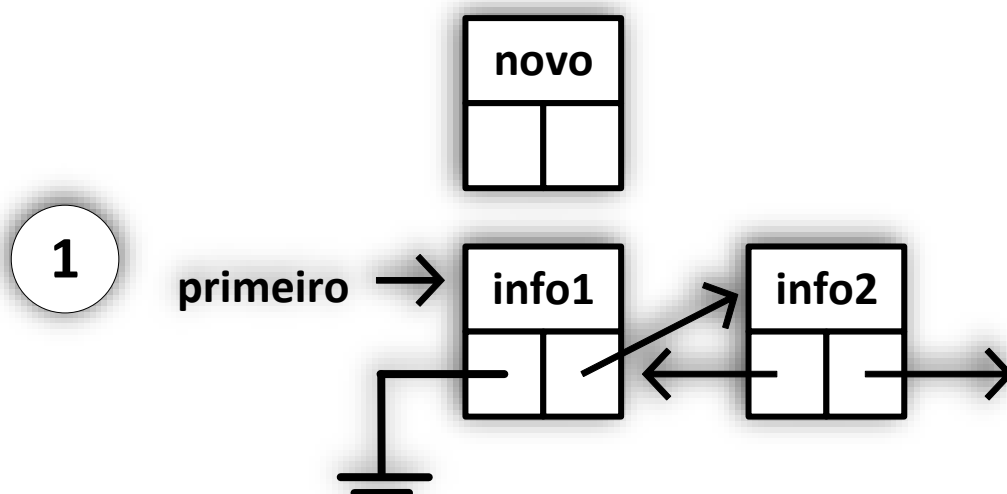
Lista duplamente encadeada

- Exemplo em Java para representar um nó da lista duplamente encadeada

```
public class NoListaDupla {  
  
    private int info;  
    private NoListaDupla proximo;  
    private NoListaDupla anterior;  
  
}
```


Incluir valor na lista duplamente encadeada

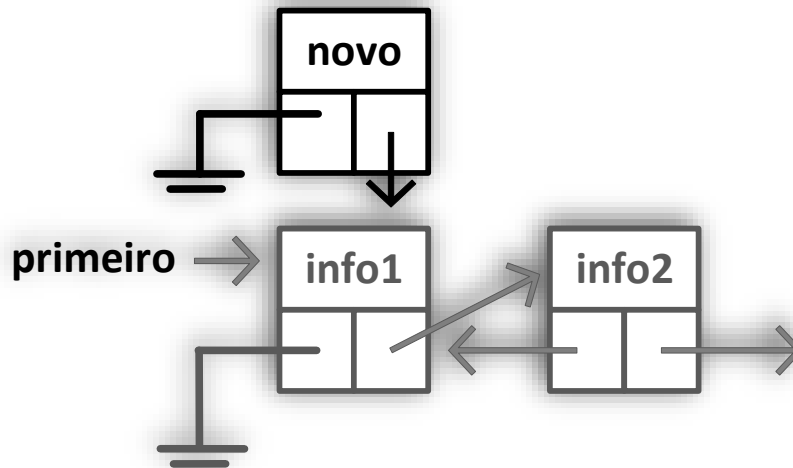
- Necessário:
 - Incluir novo objeto **NoListaDupla**
 - Encadear o novo nó no início da lista existente



Criar um novo nó para
armazenar o valor

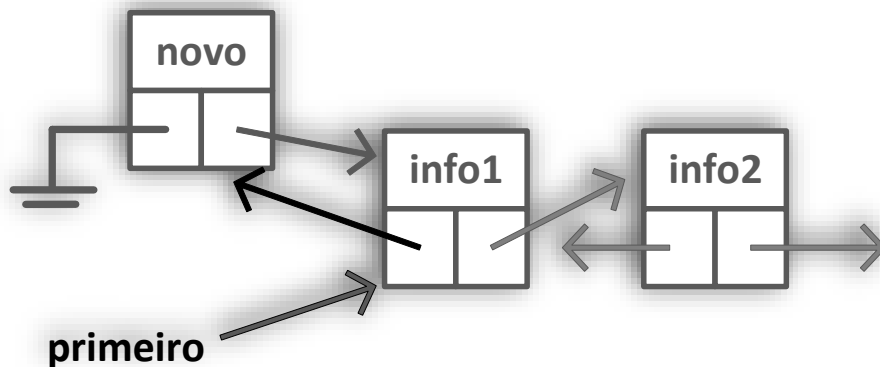
Incluir valor na lista duplamente encadeada

2



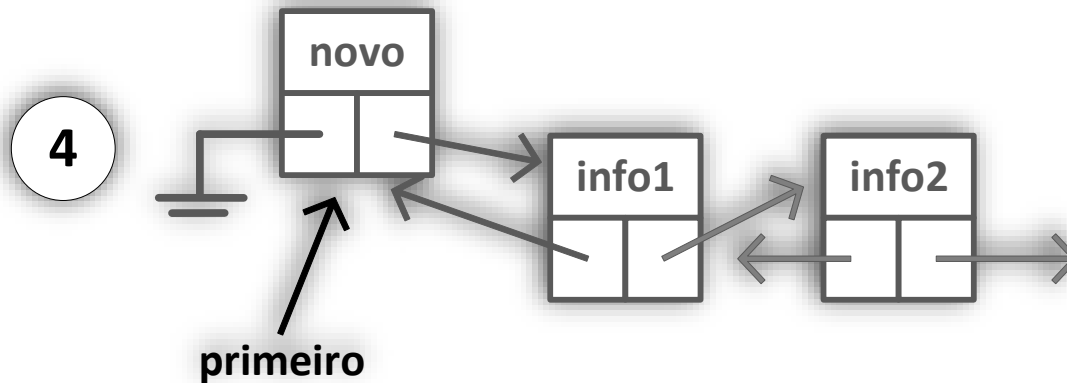
- Próximo do *novo nó* aponta para o primeiro
- Anterior do *novo nó* aponta para null

3



- *Primeiro nó* aponta para *novo nó*, como seu anterior

Incluir valor na lista duplamente encadeada



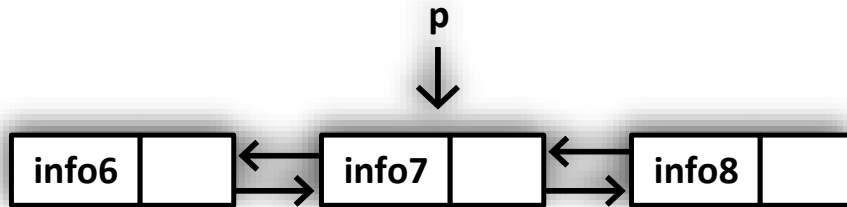
- *Primeiro* aponta para novo nó

Algoritmo: inserir(int valor)

```
NoListaDupla novo ← new NoListaDupla();  
novo.info ← valor;  
novo.proximo ← primeiro;  
novo.anterior ← null;  
se (primeiro ≠ null) então  
    primeiro.anterior ← novo;  
fim-se  
primeiro ← novo;
```

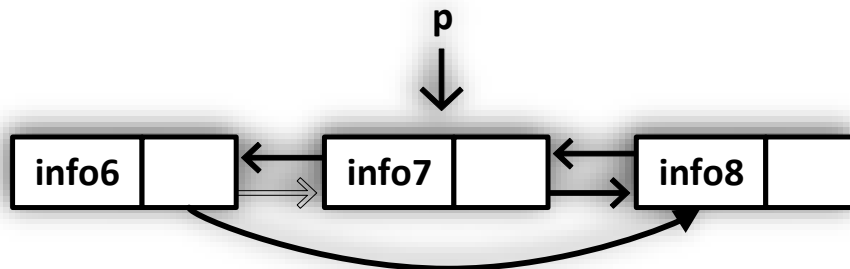
Retirar valor da lista duplamente encadeada

1



Localizar o nó a ser removido, apontado por “p”

2

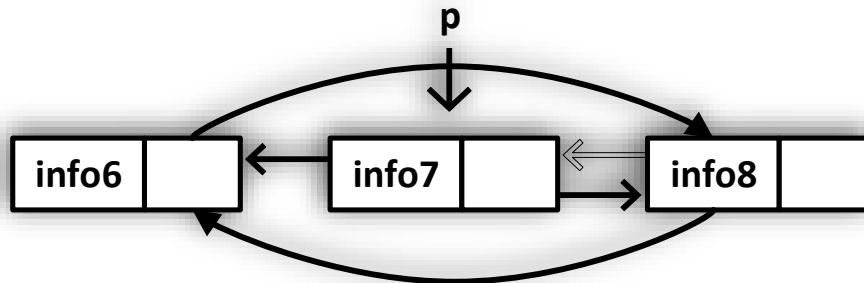


O nó anterior à p deve apontar (como próximo) para o próximo de p

```
p.anterior.proximo ← p.proximo;
```

Retirar valor da lista duplamente encadeada

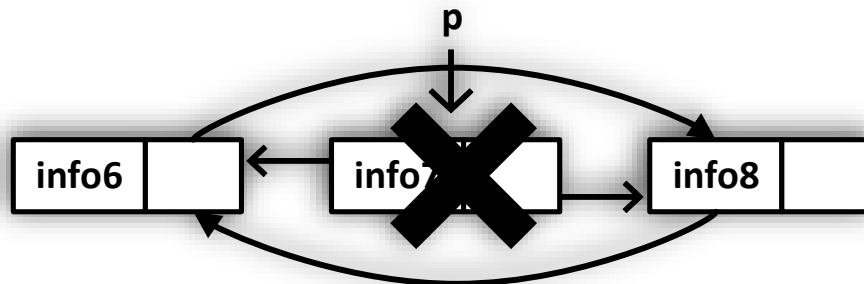
3



O nó posterior à p deve apontar como anterior para o anterior de p

$p.\text{proximo}.\text{anterior} \leftarrow p.\text{anterior}$

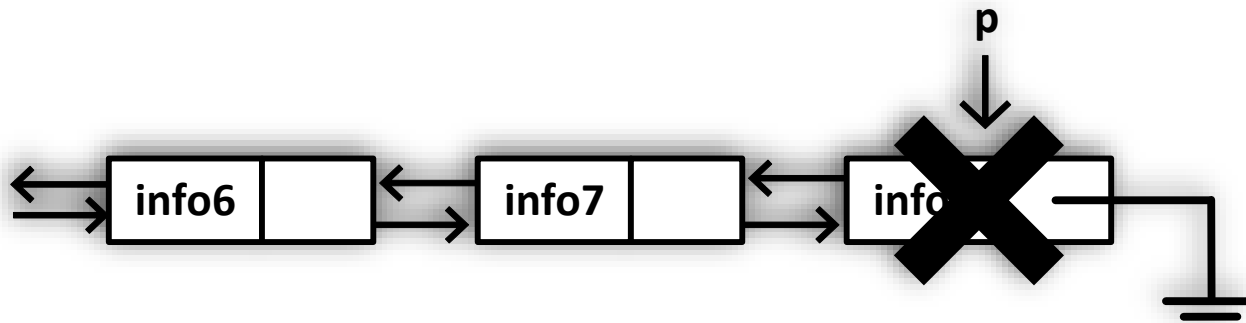
4



Nenhum outro nó da lista aponta para p

Retirar valor da lista duplamente encadeada

- Caso especial - Exclusão do último nó da lista



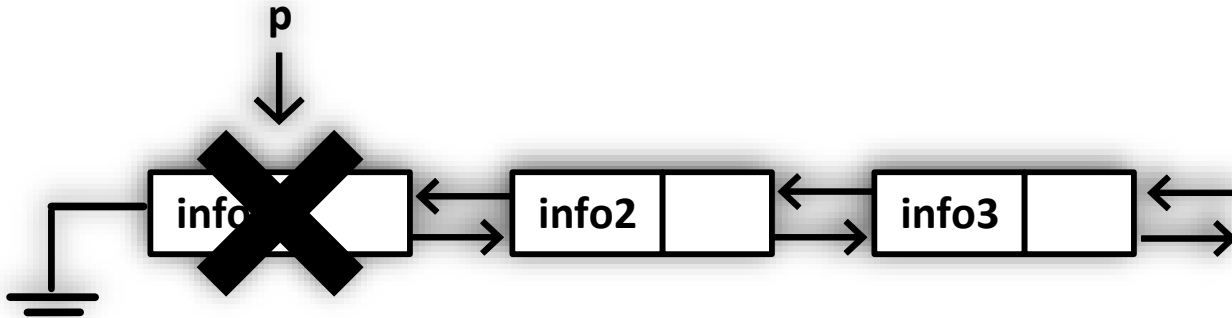
- Não é possível executar

`p.proximo.anterior ← p.anterior`

Pois `p.proximo` é `null`

Retirar valor da lista duplamente encadeada

- Caso especial – Exclusão do primeiro da lista



- Não é possível executar
`p.anterior.proximo ← p.proximo;`

Pois `p.anterior` é igual à `null`

Retirar valor da lista duplamente encadeada

Algoritmo: retirar(int valor)

NoListaDupla p \leftarrow buscar(valor);

se (p \neq null) **então** // achou

se (primeiro = p) **então** // primeiro elemento ?

 primeiro \leftarrow p.proximo;

senão

 p.anterior.proximo \leftarrow p.proximo;

fim-se

se (p.proximo \neq null) **então** // não é o último?

 p.proximo.anterior \leftarrow p.anterior;

fim-se

fim-se