Listas encadeadas

Prof. Gilvan Justino
Prof. Marcel Hugo

FURB
UNIVERSIDADE DE BLUMENAU

1

Tópicos

- Motivação
- Conceitos
- Listas circulares
- Listas duplamente encadeadas

FURB
UNIVERSIDADE DE BLUMENAU

2

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

3

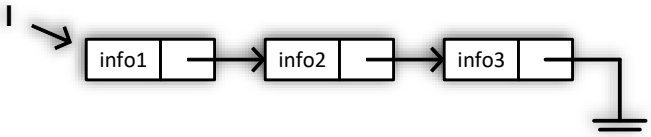
Solução

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

4

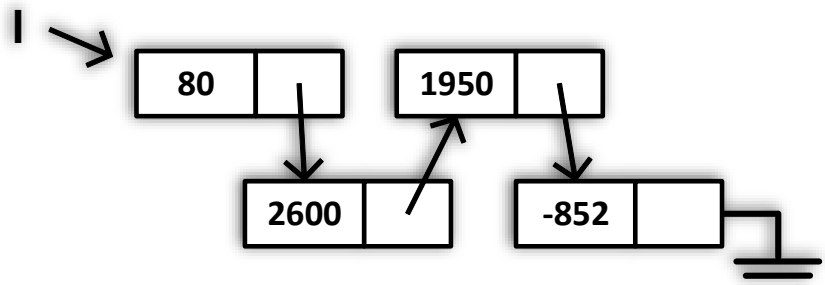
Listas encadeadas

- Sequencia de elementos encadeados;
- Cada elemento é denominado de “nó da lista”, “nodo” ou “célula”;
- Cada nó da lista contém:
  - A informação que se quer armazenar
  - Uma referência para o próximo elemento da lista
- A lista possui uma referência para o primeiro nó;
- A referência do último nó é **`null`**



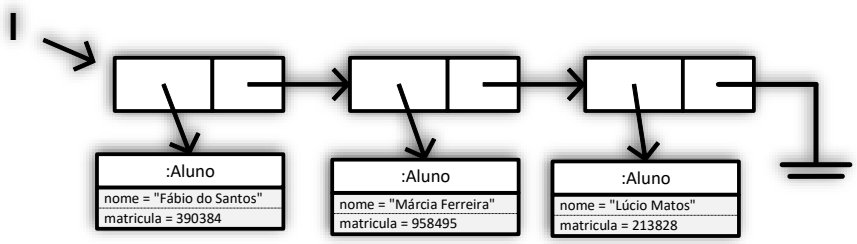
5

Exemplo de listas encadeada para armazenar dados primitivos



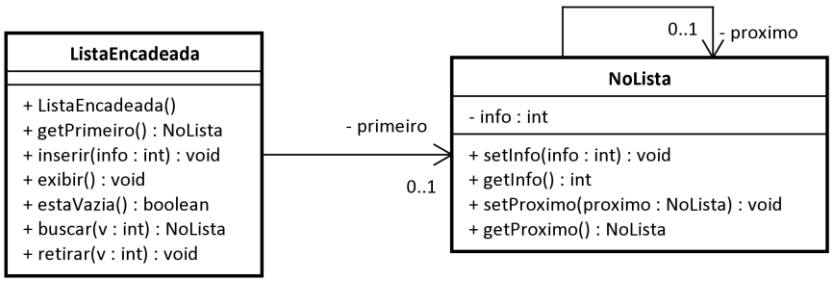
6

Exemplo de lista encadeada para armazenar objetos



7

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



8

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.

9

Classe `ListaEncadeada`

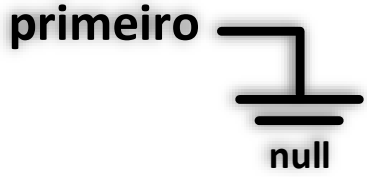
- Um objeto da classe `Lista` deve ser responsável por:
  - Referenciar o primeiro/último nó da estrutura de dados
  - Manipular os elementos da lista

```
public class ListaEncadeada {

    private NoLista primeiro;
    private NoLista ultimo;
    private int qtdeElem;

}
```

10

Listas encadeada

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



Algoritmo: **ListaEncadeada()**

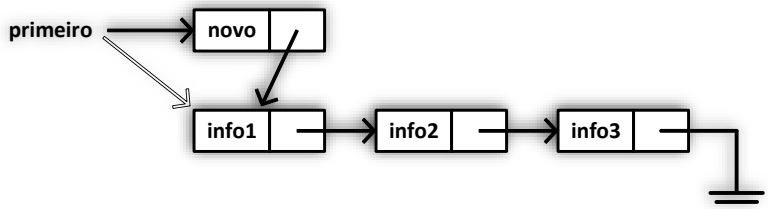
primeiro ← null**;**

11

Manipulação de listas encadeadas

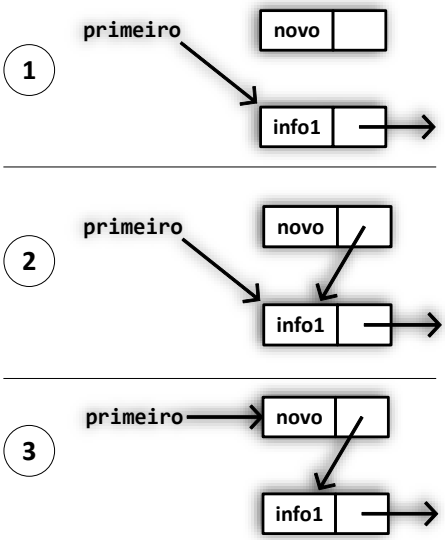
12

Inclusão de um novo elemento na lista



13

Inclusão de um novo elemento na lista

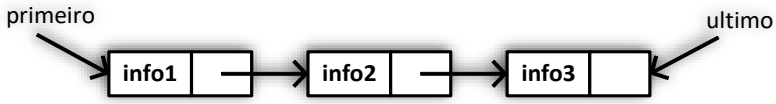


Algoritmo: inserir(int info)

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

14

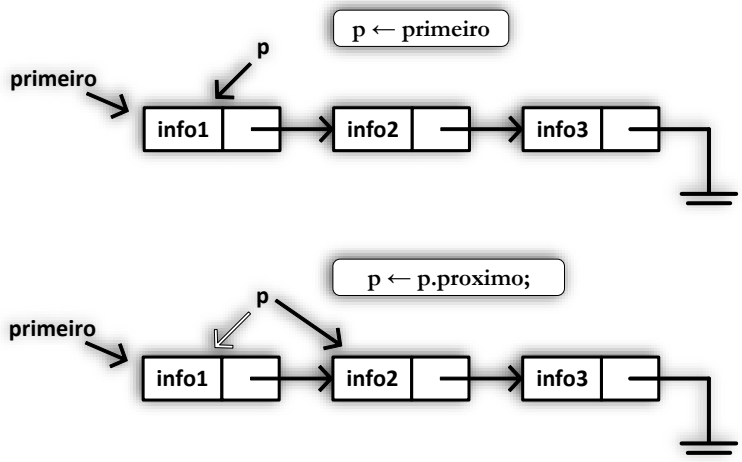
Listas com acesso as duas extremidades



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

15

Exibir o conteúdo da lista encadeada



16

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

17

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;
```

18

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`

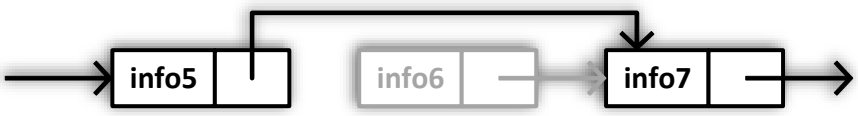
```
Algoritmo: buscar(int v)

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

19

Retirar um elemento da lista

Dado um valor, procura o nó na lista para removê-lo

Faz com que o *no anterior* aponte para o *próximo do nó* a ser removido

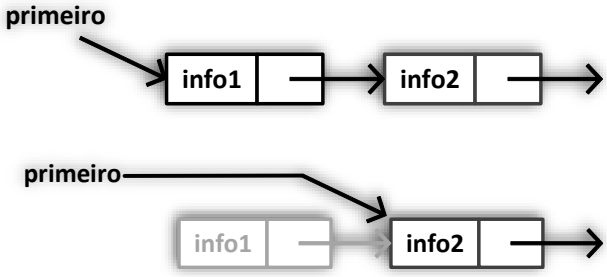


anterior.proximo ← p.proximo

20

Caso especial: Retirar o primeiro nó da lista

Considerar que o nó a ser removido contém "info1"



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

21

Retirar um nó da lista

```
Algoritmo: retirar(int v)

NoLista anterior ← null;
NoLista p ← primeiro;

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

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

22