ESTRUTURA DE DADOS

LISTA

Prof. Ademar Schmitz, M.Sc.

REFERÊNCIAS

- GOODRICH, Michael. T., TAMASSIA, Roberto. **Estruturas de Dados e Algoritmos em Java.** São Paulo: Bookman, 2002.
  - Capítulo 4: Pilhas, Filas e Deques.
- WEISS, Mark A. **Data Structures & Algorithm Analisis in Java**. Addison Wesley Longman, 1999.
  - Chapter 3: Lists, Stacks, and Queues

DEFINIÇÃO

- Uma **lista** é uma estrutura de dados básica para manter elementos e o relacionamento entre eles.
- Uma lista apresenta um **seqüência finita** de elementos, sendo que existe uma determinada **ordem** na seqüência destes elementos.

DEFINIÇÃO

- O número de elementos em uma lista vai determinar o **comprimento** da lista.
- Se o comprimento da lista for zero, isto quer dizer que a lista é vazia, ou seja, não existe elemento na lista.
- Uma lista pode ser **ordenada** ou **não-ordenada**.

EXEMPLOS: LISTAS

- Mundo real:
  - Lista de compras
  - Lista telefônica
  - Lista de pagamentos
  - Lista de alunos
- Computação
  - Lista de compras
  - Lista telefônica
  - Lista de pagamentos
  - Lista de alunos

TAD LISTA

- *size():* retorna o número de elementos em S.
  - entrada: nenhuma
  - saída: inteiro
- *isEmpty():* retorna um valor booleano indicando se a lista S esta vazia ou não.
  - entrada: nenhuma
  - saída: booleano

TAD LISTA

- *insertFirst(o):* insere o novo elemento *o* em S como primeiro elemento.
  - entrada: objeto
  - saída: nenhuma
- *insertLast(o):* insere o novo elemento *o* em S como último elemento.
  - entrada: objeto
  - saída: nenhuma
- *insertAt(o, i):* insere o novo elemento *o* em S na posição *i*; ocorre um erro se a posição for inválida.
  - entrada: objeto, inteiro
  - saída: nenhuma

TAD LISTA

- *first():* retorna o primeiro elemento de S; ocorre um erro se S estiver vazio.
  - entrada: nenhuma
  - saída: objeto
- *last():* retorna o último elemento de S; um erro ocorre se S está vazio.
  - entrada: nenhuma
  - saída: objeto

TAD LISTA

- *elementAt(i):* retorna o elemento na posição *i* de S; ocorre um erro se a posição *i* for inválida ou se a lista estiver vazia.
  - entrada: inteiro
  - saída: objeto
- *findElement(o):* retorna a posição do elemento *o* em S.
  - entrada: objeto
  - saída: inteiro

TAD LISTA

- *removeFirst():* remove o primeiro elemento da lista; ocorre um erro se a lista estiver vazia.
  - entrada: nenhuma
  - saída: nenhuma
- *removeLast():* remove último elemento da lista; ocorre um erro se a lista estiver vazia.
  - entrada: nenhuma
  - saída: nenhuma

TAD LISTA

- *remove(o):* remove o elemento *o* da lista.
  - entrada: objeto
  - saída: nenhuma
- *removeAt(i):* remove o elemento que está na posição *i* da lista; ocorre um erro se a posição *i* for inválida.
  - entrada: inteiro
  - saída: nenhuma

TAD LISTA: LIST

- Implementar um TAD envolve dois passos:
  - Definir uma interface que descreve os nomes dos métodos que o TAD suporta e como eles são declarados e usados.
  - Fornecer uma classe concreta que implemente os métodos descritos na interface associada com o TAD.
  - Esta classe concreta passa a ser uma Estrutura de Dados do tipo Lista.

INTERFACE LIST

```
public interface List {

    public int size();

    public boolean isEmpty();

    public void insertFirst(Object element);

    public void insertLast(Object element);

    public void insertAt(Object element, int position)
        throws InvalidPositionException;
}
```


INTERFACE LIST

```
public interface List {

    public Object first() throws EmptyListException;

    public Object last() throws EmptyListException;

    public Object elementAt(int position)
        throws InvalidPositionException;

    public int findElement(Object element);
}
```


INTERFACE LIST

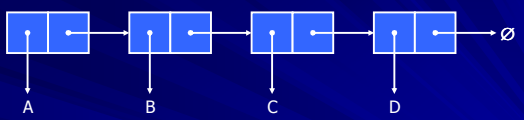
```
public interface List {

    public void removeFirst()
        throws EmptyListException;

    public void removeLast()
        throws EmptyListException;

    public void remove(Object element);

    public void removeAt(int position)
        throws InvalidPositionException;
}
```


IMPLEMENTAÇÃO COM ARRAYS

Primeiro Elemento
Segundo Elemento
...
...
...
...
...
...
...
Ultimo Elemento

- Elementos dispostos em uma organização física linear.

IMPLEMENTAÇÃO COM LISTAS ENCADEADAS

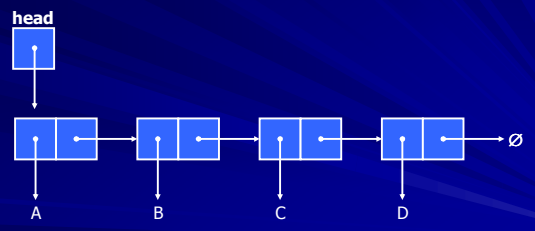


- Elementos dispostos em organização física não linear.
- Cada elemento do conjunto possui informações sobre o elemento seguinte, e possivelmente sobre o anterior.

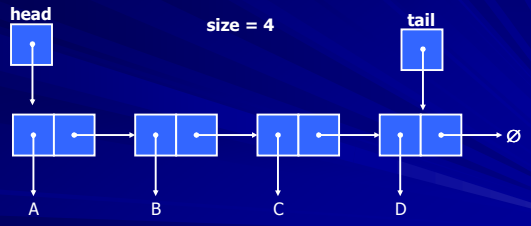
TIPOS DE LISTAS ENCADEADAS

- **Encadeamento simples:** os elementos da lista possuem apenas um ponteiro que aponta pra o elemento sucessor.
- **Duplamente encadeadas:** cada elemento possui um campo que aposta para o seu predecessor (anterior) e outro pra o seu sucessor (próximo).
- **Circulares:** o ponteiro próximo do último elemento aponta para o primeiro, e o ponteiro anterior do primeiro elemento aponta para o último.

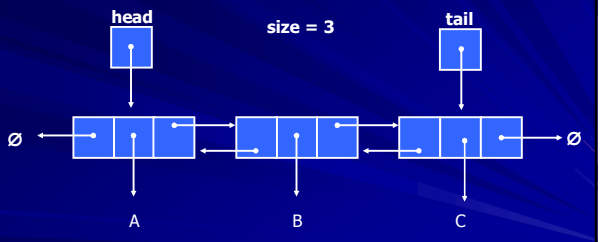
LISTA SIMPLESMENTE ENCADEADA: HEAD



LISTA SIMPLESMENTE ENCADEADA: HEAD,TAIL, SIZE



LISTA DUPLAMENTE ENCADEADA



ANÁLISE ASSINTÔTICA DAS OPERAÇÕES

OPERAÇÃO	ARRAY	SIMPLES (head)	SIMPLES (head, tail, size)	DUPLA (head, tail, size)
size()	O(1)	O(n)	O(1)	O(1)
isEmpty()	O(1)	O(1)	O(1)	O(1)
insertFirst(o)	O(n)	O(1)	O(1)	O(1)
insertLast(o)	O(1)	O(n)	O(1)	O(1)
insertAt(o, i)	O(n)	O(n)	O(n)	O(n)
first()	O(1)	O(1)	O(1)	O(1)
last()	O(1)	O(n)	O(1)	O(1)
elementAt(i)	O(1)	O(n)	O(n)	O(n)
findElement(o)	O(n)	O(n)	O(n)	O(n)
removeFirst()	O(n)	O(1)	O(1)	O(1)
removeLast()	O(1)	O(n)	O(n)	O(1)
remove(o)	O(n)	O(n)	O(n)	O(n)
removeAt(i)	O(n)	O(n)	O(n)	O(n)