# Tipos abstratos de dados Listas

Prof. Tiago Massoni
Prof. Fernando Buarque

Engenharia da Computação

Poli - UPE

## Noção de tipos abstratos de dados (TAD)

- TAD = Valores + Operações sobre estes valores
- TADs implicam em conceitos matemáticos (*i.e.* algebras) que são independentes das suas implementações



## Noção de tipos abstratos de dados (TAD)

- A implementação do algoritmo em uma linguagem de programação específica exige a representação do TAD em termos dos tipos de dados e dos operadores suportados
  - A representação do modelo matemático por trás do TAD é realizada mediante uma estrutura de dados



## TADs

- Exemplo: o conjunto dos inteiros (objetos) acompanhado das operações de adição, subtração e multiplicação
- Em Java
  - Classes: oferecem operações sobre um tipo
  - Encapsulamento: não sabemos como serão implementados



## Listas

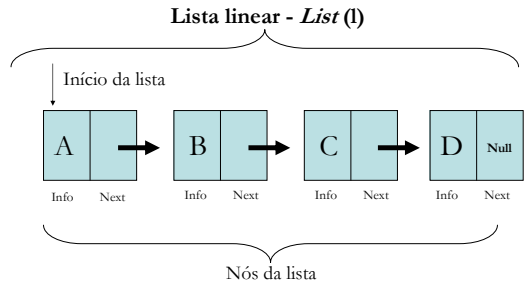
"**Listas** são conjuntos de dados (i.e. estruturas de dados) nos quais existe um encadeamento sequencial dos elementos. Novos elementos podem ser inseridos ou elementos pré-existentes podem ser removidos a depender da ordem especificada para o conjunto



## O TAD "Lista"

- A0,A1,A2, ..., A(n-1) – lista geral
  - N: tamanho da lista (n=0, lista vazia)
- Ai+1 sucede Ai, que sucede Ai-1
  - Posição = i
- Operações
  - imprimeLista
  - esvaziar
  - inserir, remover
  - procurarIEsimo

# Lista - intuição






# Listas

- Nó **node p**
  - Informação – conteúdo útil armazenado p.**info**
  - Next – refer. para o nó seguinte p.**next**

Observação: a referência nula (**null**) indica o final da lista

- Lista vazia
  list = null
- Se **p.next** != **null** então **p.next.info** é a porção informação do nó posterior ao nó apontado por **p**



# Listas com arrays

```
public class ListaArrays {
      private Object array;
      private int count=0;
      public ListaArrays(){
            this.array = new Object[MAX];
      }
      ...
      public void printList(){..}
      public Object find(Object x){..}
      public Object findKth(int pos){..}
      public void insert(Object x){..}
      public void remove(Object x){..}
}
```



# Listas com arrays

- findKth – tempo constante
- printList e find – sempre o mesmo tempo para o mesmo número de nós
- No entanto, inserir e remover são extremamente custosos
- Outras desvantagens
  - Tamanho não pode ser alterado durante execução
  - quantidade fixa de elementos
    - Ex.: impossível incluir um elemento a mais que MAX (sub-utilização)
  - Estrutura com menos elementos que MAX ocupam toda a memória pré-alocada



# Listas ligadas (encadeadas)

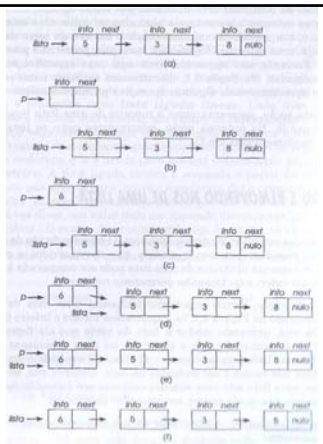
- Não são armazenadas de forma contígua
  - Nós não estão necessariamente em sequência na memória
- printList e find: começa a procura a partir do começo da lista
  - Uso do next de cada nó
- findKth mais ineficiente que array
- insert, remove: mais eficientes
  - Insert: uma criação de objeto e duas mudanças de referências
  - Remove: uma mudanças de referências



# Implementação - nós de uma lista ligada

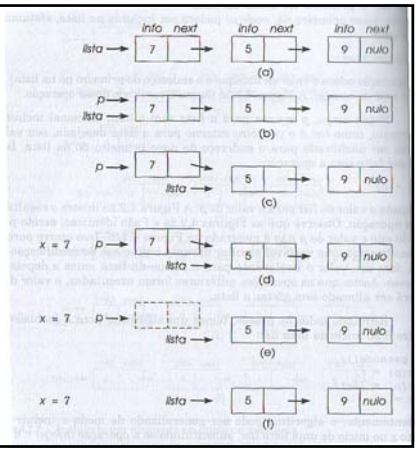
```
public class Node {
      private Object info;
      private Node next;

      public Node(Object el){
            this.info = el;
      }
      public Node(Object el, Node next){
            this.next = next;
      }
      //metodos get e set
}
```

## Inserção de nó no início





## Remoção de nó do início



## Detalhes de implementação

- Ao inserir ou remover do início temos que tratar da referência inicial da lista
  - Casos especiais
  - Solução – colocar um nó cabeçalho (header) na lista – não possui conteúdo
    - Evita casos especiais
- Ao remover um nó, temos que ter acesso ao seu nó anterior
  - Solução: findPrevious(Object x)



## Iterator de Lista

- Vamos usar um objeto para representar a noção de posição
  - Não vamos usar inteiros, para deixar a implementação mais flexível
- Também separaremos neste objeto a navegação na lista por posições
  - Separação entre a implementação da lista e as operações de navegação
- Este objeto – Iterator (Iterador)
  - Oferece navegação a partir de uma dada posição



## Implementação - Iterator

```
public class ListIterator {
    Node current;
    ListIterator( Node n){
      current= n;
    }

    public boolean isPastEnd( ) {
      return (current == null);
    }

    public Object retrieve( ) {
      return this.isPastEnd()?null:current.getElement();
    }

    public void advance( ){
      if( !isPastEnd( ) )
         current = current.next;
    }
}
```

## Implementação – Lista (com header)

```
public class LinkedList {
  private Node header;

  public LinkedList( ) {
    header = new Node(null);
  }

  public boolean isEmpty( ){..}

  public void makeEmpty( ){..}

  public ListIterator zeroth( ){..}

  public ListIterator first( ){..}
```

## Implementação – Lista (com header)

```
public void insert(Object x, ListIterator p){
 if (p!=null && p.getCurrent()!= null)
  p.getCurrent().setNext(new Node(x,p.getCurrent().getNext());
  ...
}
public ListIterator find(Object x){..}
public ListIterator findPrevious(Object x){..}
public void remove(Object x){..} //usa findPrevious
public void printList(){..}
}
```

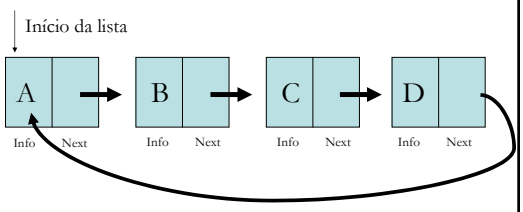


## Limitação das listas

- Dificuldade para do final da lista retornar ao começo
  - Listas circulares
- Dificuldade de acessar o elemento anterior
  - Listas duplamente ligadas (encadeadas)



## Listas ligadas circulares





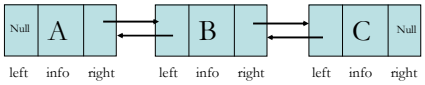

## Listas ligadas circulares

- Se existir nó cabeçalho, usá-lo com cuidado
  - Último nó da lista referencia header
- Operações primitivas – alguns testes são afetados
  - insert(x,p) idêntico às listas simples
  - remove(x)?

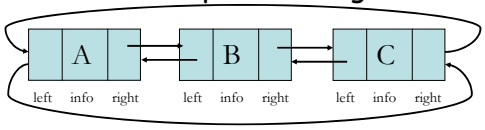


## Listas duplamente ligadas

- Lista linear duplamente ligada:




- Lista circular duplamente ligada:






## Listas duplamente ligadas

```
public class DoubleNode {
      private Object info;
      private Node prev;
      private Node next;
      ...
}
```

- Custo aumenta: atributo extra que armazena ref. para anterior
  - insert e remove também fazem mais mudanças
- Simplifica remoção
  - Não precisa mais usar findPrevious

## Exercício: "Problema de Josephus"

- Soldados cercados pelo inimigo decidem que um deles deve tentar escapar para buscar reforços usando o único cavalo disponível
- Forma-se um círculo (lista circular duplamente ligada), sorteia-se um número e se começa a contar os nós da lista até que se chegue ao número escolhido. Quando o soldado (nó) é retirado
- O processo reinicia e continuará se repetindo até que um último soldado seja o escolhido



## Exercício

- As classes ligadas à lista devem estar em um pacote separado (biblioteca)
- Escreva o método estático josephus(lista,n) que seleciona o nó vencedor de uma lista circular apontada por lista em sorteios dados por contagens de n. A contagem de n nós alternará à esquerda e à direita