Algoritmos e Estruturas de Dados

listas ligadas

2010-2011

Carlos Lisboa Bento

Listas Ligadas

roadmap

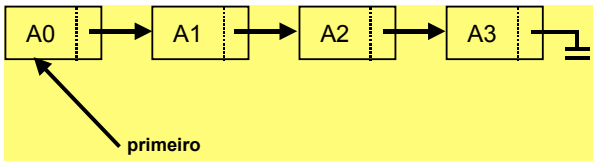
- **listas simplesmente ligadas**
  - **conceitos**
  - **vantagens e desvantagens**
  - ***implementação***
- **listas duplamente ligadas**
  - **conceitos**
  - **vantagens e desvantagens**
  - ***implementação***
- **listas circulares**
  - **conceitos**
  - **vantagens e desvantagens**
  - ***implementação***
- **listas auto-organizadas**
  - **conceitos**
  - **exemplo**
  - **vantagens e desvantagens**
- **listas ligadas e tabelas esparsas**
  - **exemplo de uma aplicação das listas ligadas**
  - **vantagens e desvantagens**
- **listas de saltos**
  - **conceitos**
  - **vantagens e desvantagens**
  - ***implementação***

Listas Ligadas
conceitos

Estrutura em que cada elemento contém o endereço do elemento seguinte (a sequência é estabelecida explicitamente).

Cada elemento de uma lista contém dois campos:

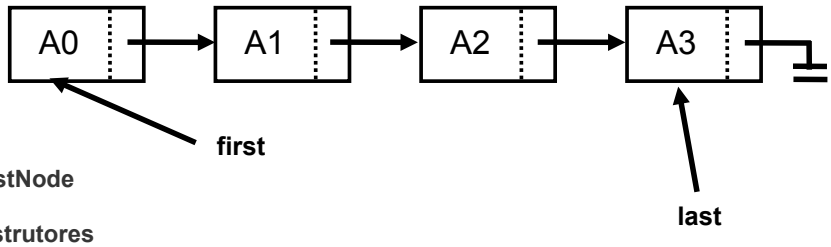
- Campo da informação;
- Campo do endereço do elemento seguinte.



A lista é acedida através de um ponteiro externo que aponta para o primeiro nó.

O fim da lista é indicado pelo ponteiro para elemento seguinte a NULL

Listas Ligadas
conceitos

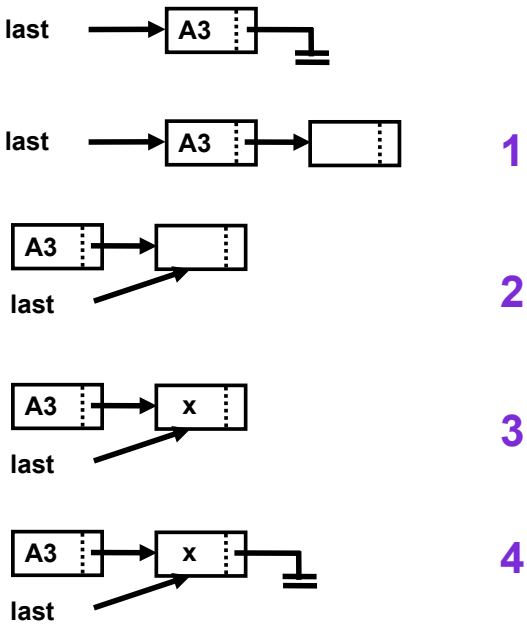



```
Class ListNode
{
   // construtores

  ListNode()
     { this( null, null ); }

   ListNode( Object theElement )
     { this( theElement, null ); }

   ListNode( Object theElement, ListNode n )
    { element = theElement; next = n;}

    Object element;
    ListNode next;
}
```

Inserção a seguir ao último elemento !!

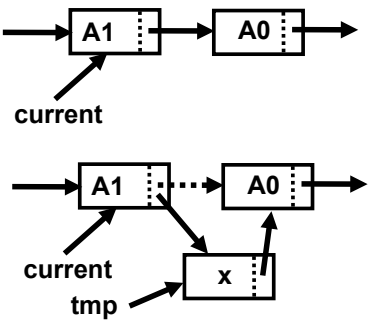
Listas Ligadas
conceitos

```
/* inserção de um elemento x a
seguir a ultimo */

last.next = new ListNode();  // 1

last = last.next;            // 2

last.data = x;               // 3

last.next = null;            // 4
```



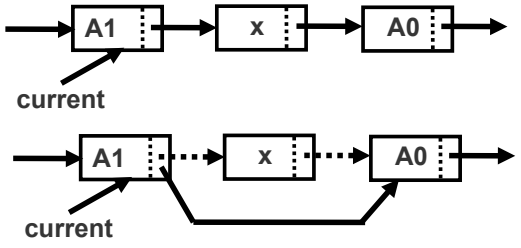
Listas Ligadas
implementação

Duas operações básicas numa lista - inserção a partir de current




```
tmp = new ListNode();
tmp.element = x;
tmp.next = current.next;
current.next = tmp;

// MELHOR

tmp = new ListNode( x, current.next );
current.next = tmp;

// MELHOR AINDA

current.next = new ListNode( x, current.next );
```

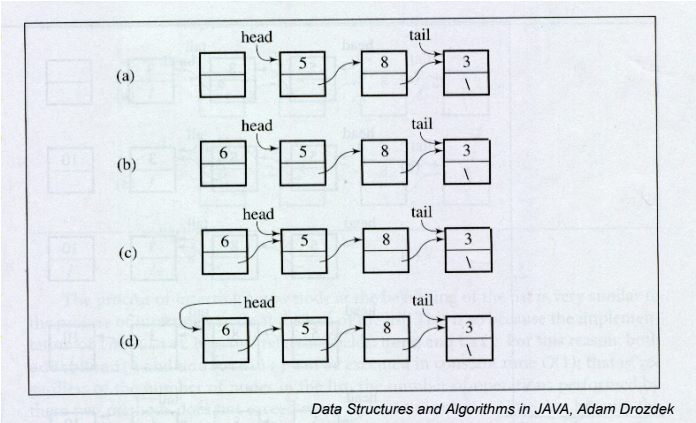
eliminação a partir de current




```
current.next = current.next.next;
```

Listas Ligadas
vantagens e desvantagens

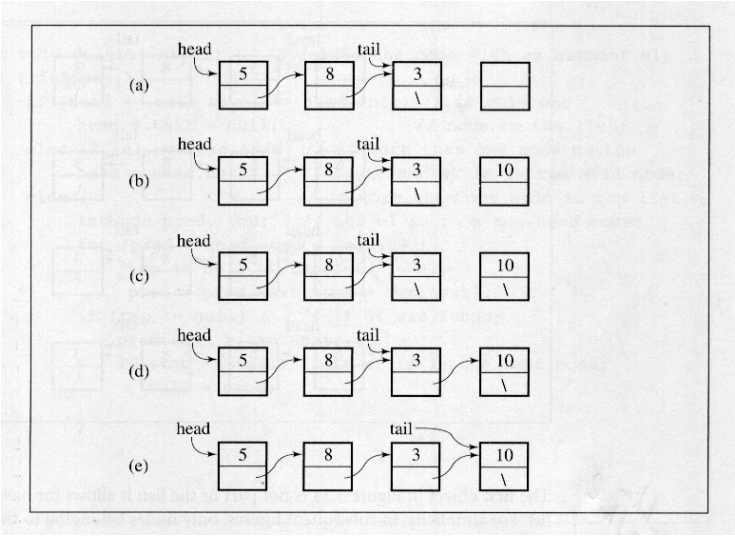
Inserção na cabeça de uma lista ligada ☺



Data Structures and Algorithms in JAVA, Adam Drozdek

Listas Ligadas
vantagens e desvantagens

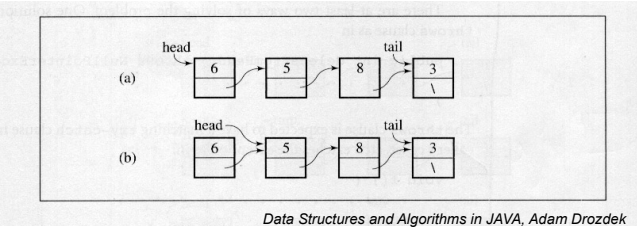
Inserção na cauda de uma lista ligada ☺



Data Structures and Algorithms in JAVA, Adam Drozdek

Listas Ligadas
vantagens e desvantagens

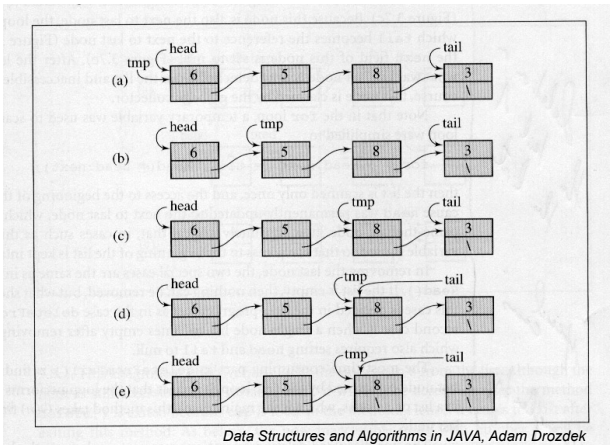
Eliminação na cabeça de uma lista ligada ☺



Data Structures and Algorithms in JAVA, Adam Drozdek

Listas Ligadas
vantagens e desvantagens

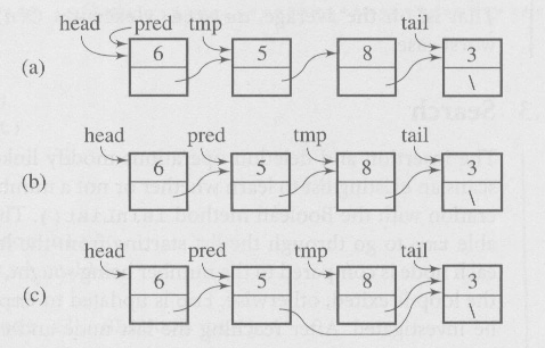
Eliminação na cauda de uma lista ligada ☹



Data Structures and Algorithms in JAVA, Adam Drozdek

Listas Ligadas
vantagens e desvantagens

Eliminação num ponto intermédio de uma lista ligada ☹



Data Structures and Algorithms in JAVA, Adam Drozdek

Listas Ligadas
vantagens e desvantagens

Vantagens

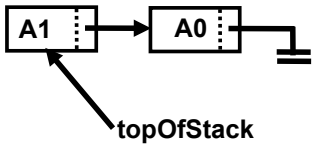
- A inserção de um elemento no meio de uma lista ligada não implica mover todos os elementos que se seguem.
- É ocupada apenas a memória necessária em cada momento, não havendo limitações prévias às dimensões destas estruturas (excepto a capacidade de memória do computador utilizado). *Esta vantagem não é substancial em JAVA, embora o seja noutras linguagens.*

Desvantagens

- Acesso sequencial.
  - Acesso ou eliminação de um nó intermédio.
  - Eliminação de um nó no fim da lista.
- Na prática a movimentação de elementos numa lista ligada é dispendiosa.

Listas Ligadas
implementação

Implementação de uma PILHA sobre uma lista ligada




```
package DataStructures;
import Exceptions.*;

// ******************PUBLIC OPERATIONS*****************
// void push( x )         --> Insert x
// Object pop( )          --> Return and
//                            Remove most recently
//                            inserted item
// boolean isEmpty( )     --> Return true if empty;
//                            else false
// void makeEmpty( )      --> Remove all items
// ******************ERRORS*********************************
// pop on empty stack
```

```
/**
 * List-based implementation of the stack.
 * @author Mark Allen Weiss
 * @changes Francisco Pereira
 */
public class StackLi implements Stack
{
    public StackLi( )
    { topOfStack = null; }

    public boolean isEmpty( )
    { return topOfStack == null; }

    public void makeEmpty( )
    { topOfStack = null; }
```


Listas Ligadas
implementação

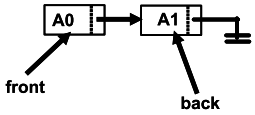
Implementação de uma PILHA sobre uma lista ligada (cont.)

```
public Object pop( ) throws Underflow
{
    if( isEmpty( ) )
        throw new Underflow( "Stack topAndPop" );

    Object topItem = topOfStack.element;
    topOfStack = topOfStack.next;
    return topItem;
}
```

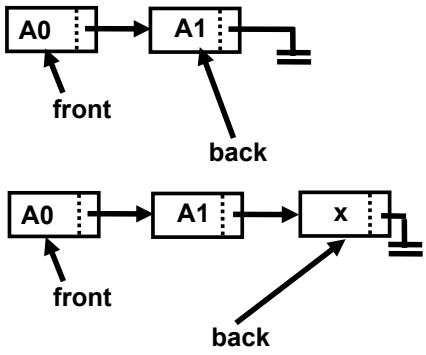
```
public void push( Object x )
    {
        topOfStack = new ListNode( x, topOfStack );
    }

    private ListNode topOfStack;
}
```

Listas Ligadas
implementação

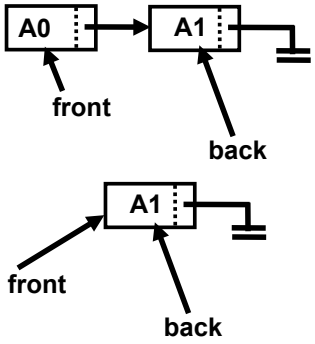
Implementação de uma FILA DE ESPERA sobre uma lista ligada



Enqueue (x)

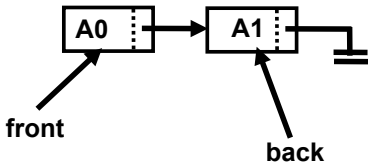


Dequeue (x)

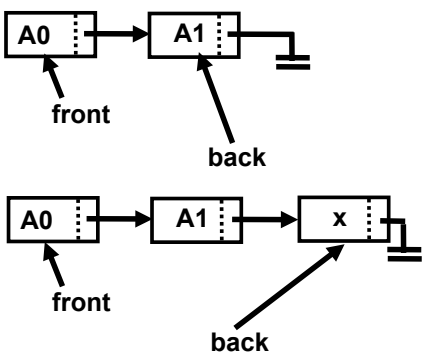


Listas Ligadas
implementação

Implementação de uma FILA DE ESPERA sobre uma lista ligada



Enqueue (x)




```
package DataStructures;
import Exceptions.*;

// QueueLi class
//
// ******************PUBLIC OPERATIONS*********************
// void enqueue( x )      --> Insert x
// Object dequeue( )      --> Return and remove least recent item
// boolean isEmpty( )     --> Return true if empty; else false
// void makeEmpty( )      --> Remove all items
// ******************ERRORS********************************
// getFront or dequeue on empty queue

/**
 * List-based implementation of the queue.
 * @author Mark Allen Weiss
 */
public class QueueLi implements Queue
{
   public QueueLi( )
   {
              makeEmpty( );
   }
```

© DEI Carlos Bento ALGORITMOS E ESTRUTURAS DE DADOS 04 -

Listas Ligadas
implementação

Implementação de uma FILA DE ESPERA sobre uma lista ligada (cont.)

```
public boolean isEmpty( )
 {
    return front == null;
 }

 public void makeEmpty( )
 {
    front = null;
    back = null;
 }
```

```
public Object dequeue( ) throws Underflow
   {
      if( isEmpty( ) )
         throw new Underflow( "QueueLi dequeue" );

      Object returnValue = front.element;
      front = front.next;
      return returnValue;
   }

   public void enqueue( Object x )
   {
      if( isEmpty( ) )    // Make queue of one element
         back = front = new ListNode( x );
      else                // Regular case
         back = back.next = new ListNode( x );
   }

   private ListNode front;
   private ListNode back;
}
```


Listas Ligadas
implementação

```
//************************  IntSLList.java  ***************************
//          singly-linked list class to store integers

public class IntSLList {
   private IntNode head, tail;
   public IntSLList() {
      head = tail = null;
   }
   public boolean isEmpty() {
      return head == null;
   }
   public void addToHead(int el) {
      head = new IntNode(el,head);
      if (tail == null)
         tail = head;
   }
   public void addToTail(int el) {
      if (!isEmpty()) {
         tail.next = new IntNode(el);
         tail = tail.next;
      }
      else head = tail = new IntNode(el);
   }
```

Listas Ligadas
implementação

```
public int deleteFromHead() { // delete the head and return its info;
    int el = head.info;
    if (head == tail)    // if only one node on the list;
        head = tail = null;
    else head = head.next;
    return el;
}
```

```
public int deleteFromTail() { // delete the tail and return its info;
    int el = tail.info;
    if (head == tail)    // if only one node in the list;
        head = tail = null;
    else {              // if more than one node in the list,
        IntNode tmp;    // find the predecessor of tail;
        for (tmp = head; tmp.next != tail; tmp = tmp.next);
        tail = tmp;     // the predecessor of tail becomes tail;
        tail.next = null;
    }
    return el;
}
```

?

Listas Ligadas
implementação

```
public void printAll() {
    for (IntNode tmp = head; tmp != null; tmp = tmp.next)
        System.out.print(tmp.info + " ");
}
```

```
public boolean isInList(int el) {
    IntNode tmp;
    for (tmp = head; tmp != null && tmp.info != el; tmp = tmp.next);
    return tmp != null;
}
```

?

Listas Ligadas
implementação

```
public void delete(int el) {  // delete the node with an element el;
    if (!isEmpty())
        if (head == tail && el == head.info) // if only one
            head = tail = null;              // node on the list;
        else if (el == head.info) // if more than one node on the list;
            head = head.next;    // and el is in the head node;
        else {                   // if more than one node in the list
            IntNode pred, tmp;   // and el is in a non-head node;
            for (pred = head, tmp = head.next;
                                tmp != null && tmp.info != el;
                                        pred = pred.next, tmp = tmp.next);
                if (tmp != null) {   // if el was found;
                    pred.next = tmp.next;
                    if (tmp == tail) // if el is in the last node;
                            tail = pred;
            }
        }
    }
}
```

?

Listas Ligadas
complexidade

Inserção	CABEÇA	O(1)
	CAUDA	O(1)
	PONTO INTERMÉDIO (1)	O(n)
Eliminação	CABEÇA	O(1)
	CAUDA	O(n)
	PONTO INTERMÉDIO (1)	O(n)

(1)

$$\frac{0+1+\ldots+(n-1)}{n} = \frac{\frac{(n-1)n}{2}}{n} = \frac{n-1}{2}$$

Listas Duplamente Ligadas
conceitos

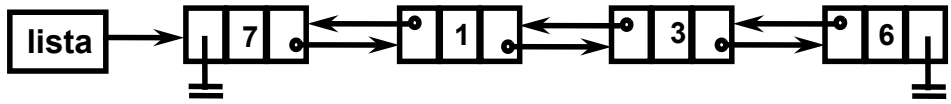
Problemas da listas lineares

- **Dado um ponteiro para um nó, não se pode ter acesso aos nós que o precedem;**
- **Quando se atravessa a lista é necessário preservar sempre o ponteiro para o início da lista**
  Uma possível solução (que também tem inconvenientes) é ter uma lista circular

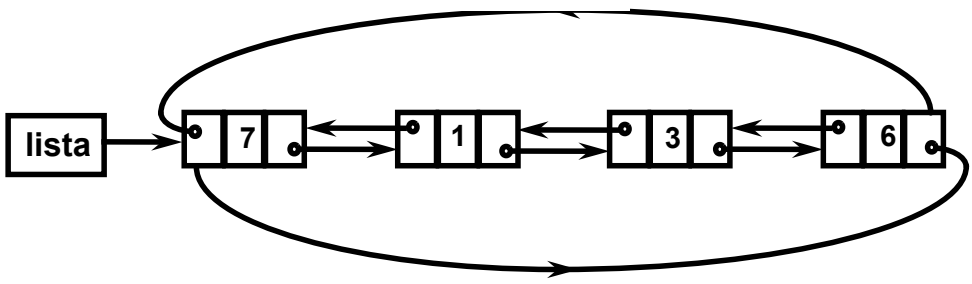
As listas duplamente ligadas permitem percursos em ambos os sentidos

Listas Duplamente Ligadas
conceitos

Simples



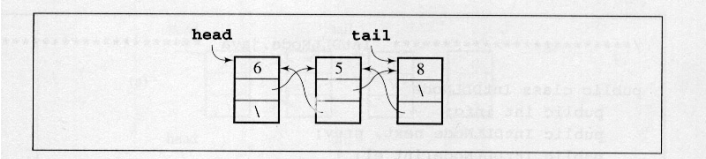
Circular



Cada nó destas listas tem, pelo menos, três campos:
- **campo (ou campos) para a informação;**
- **ponteiro para o elemento da esquerda;**
- **ponteiro para elemento da direita.**

Listas Duplamente Ligadas
vantagens e desvantagens

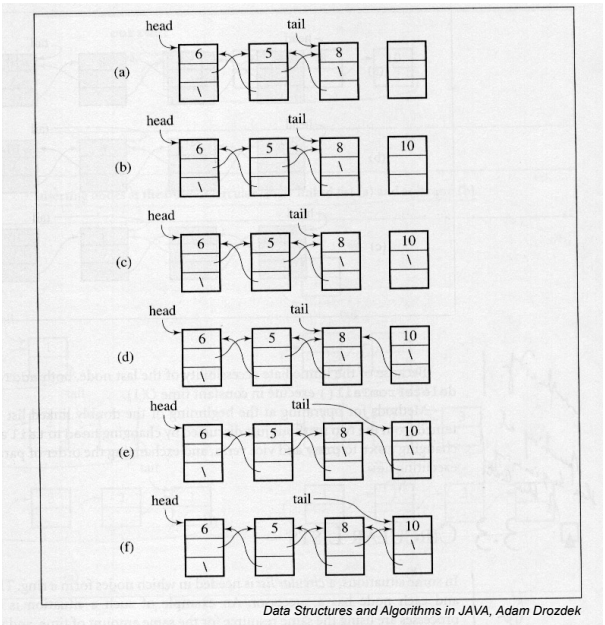
Uma lista duplamente ligada de inteiros



Data Structures and Algorithms in JAVA, Adam Drozdek

Listas Duplamente Ligadas
vantagens e desvantagens

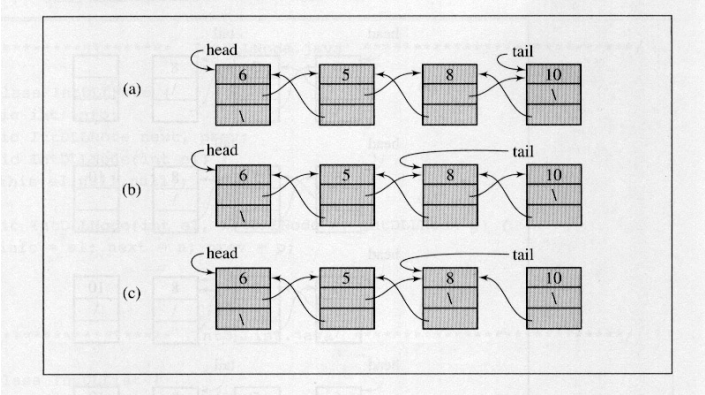
Inserção na cauda de uma lista duplamente ligada ☺



Data Structures and Algorithms in JAVA, Adam Drozdek

Listas Duplamente Ligadas
vantagens e desvantagens

Eliminação na cauda de uma lista duplamente ligada ☺



Data Structures and Algorithms in JAVA, Adam Drozdek

Listas Duplamente Ligadas
implementação LDLs ordenadas

```
//*********************  IntDLLNode.java  *********************

class IntDLLNode {

    int info;
    IntDLLNode next = null, prev = null;

    IntDLLNode() {
    }

    IntDLLNode(int el) {
        info =  el;
    }
    IntDLLNode(int el, IntDLLNode n, IntDLLNode p) {
        info = el; next = n; prev = p;
    }
}
```

Listas Duplamente Ligadas
implementação LDLs ordenadas

```
//************************  IntDLList.java  ***************************/

public class IntDLList {

   private IntDLLNode head, tail;
   public IntDLList() {
      head = tail = null;
   }
   public boolean isEmpty() {
      return head == null;
   }
   public void setToNull() {
      head = tail = null;
   }
   public int firstEl() {
      if (!isEmpty())
          return head.info;
      else return 0;
   }
```


Listas Duplamente Ligadas
implementação LDLs ordenadas

```
public void addToDLListHead(int el) {
      if (!isEmpty()) {
          head = new IntDLLNode(el,head,null);
          head.next.prev = head;
      }
      else head = tail = new IntDLLNode(el);
  }
  public void addToDLListTail(int el) {
      if (!isEmpty()) {
          tail = new IntDLLNode(el,null,tail);
          tail.prev.next = tail;
      }
      else head = tail = new IntDLLNode(el);
  }
```

Listas Duplamente Ligadas
implementação LDLs ordenadas

```
public int deleteFromDLListHead() {
    if (!isEmpty()) {        // if at least one node in the list;
        int el = head.info;
        if (head == tail)   // if only one node in the list;
            head = tail = null;
        else {              // if more than one node in the list;
            head = head.next;
            head.prev = null;
        }
        return el;
    }
    else return 0;
}
```

```
public int deleteFromDLListTail() {
    if (!isEmpty()) {
        int el = tail.info;
        if (head == tail)   // if only one node on the list;
            head = tail = null;
        else {              // if more than one node in the list;
            tail = tail.prev;
            tail.next = null;
        }
        return el;
    }
    else return 0;
}
```


Listas Duplamente Ligadas
implementação LDLs ordenadas

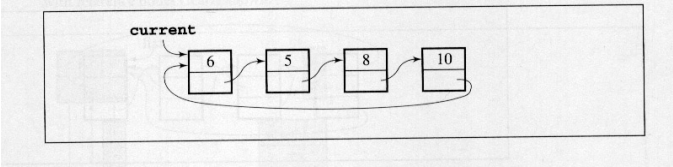
```
public void printAll() { //OutputStream Out) {
    for (IntDLLNode tmp = head; tmp != null; tmp = tmp.next)
        System.out.print(tmp.info + " ");
}
```

```
public int find(int el) {
    IntDLLNode tmp;
    for (tmp = head; tmp != null && tmp.info != el; tmp = tmp.next);
    if (tmp == null)
        return 0;
    else return tmp.info;
}
}
```

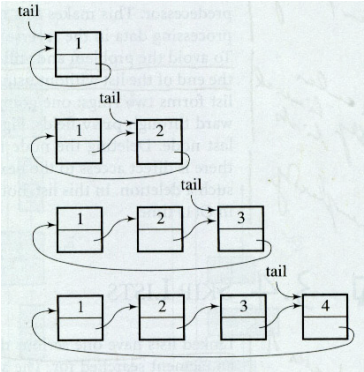
Inserção ou eliminação no ponto intermédio de uma lista duplamente ligada???

Listas Circulares
conceitos

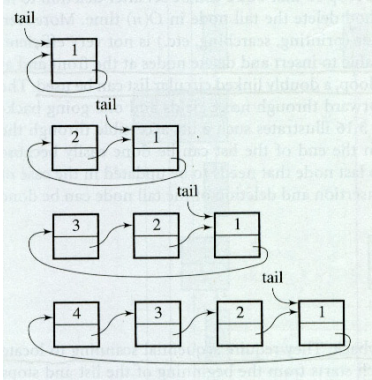
Uma lista circular ligada de inteiros



Inserção na cauda de uma lista circular

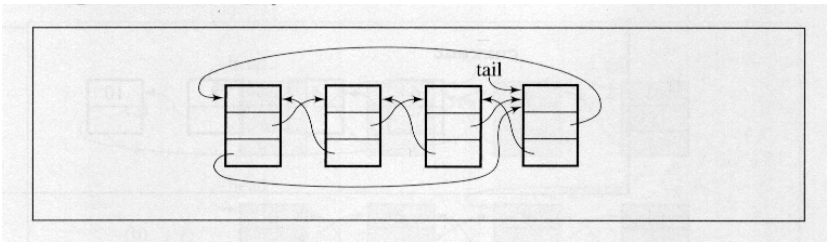


Inserção na cabeça de uma lista circular



Listas Circulares
conceitos

Uma lista circular duplamente ligada



Data Structures and Algorithms in JAVA, Adam Drozdek

Listas Circulares
vantagens e desvantagens

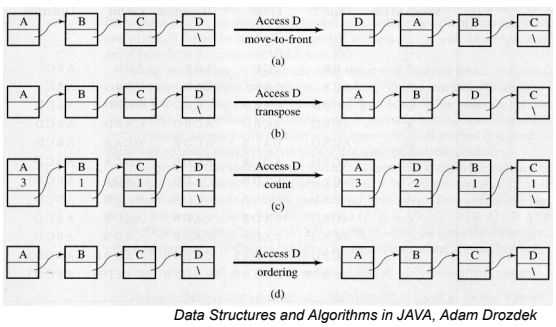
??

Listas Auto-Organizadas
conceitos

Quatro formas de organizar os elementos que se procuram numa lista:

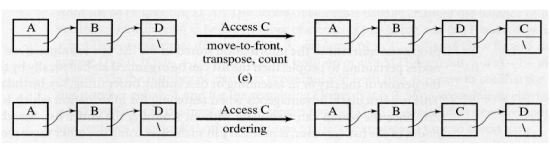
- **À CABEÇA – quando o elemento é encontrado colocá-lo à cabeça.**
- **TRANSPOSIÇÃO - quando o elemento é encontrado troca-o com o seu predecessor a menos que já esteja à cabeça.**
- **CONTAGEM – ordena a lista pelo número de vezes que cada elemento é acedido.**
- **POR CRITÉRIO – recorre a um critério adequado à informação sob escrutínio.**

CONSULTA



Data Structures and Algorithms in JAVA, Adam Drozdek

INSERÇÃO



Data Structures and Algorithms in JAVA, Adam Drozdek

Listas Auto-Organizadas
exemplo

Element Searched for	Plain	Move-to-Front	Transpose	Count	Ordering
A:	A	A	A	A	A
C:	AC	AC	AC	AC	AC
B:	ACB	ACB	ACB	ACB	ABC
C:	ACB	CAB	CAB	CAB	ABC
D:	ACBD	CABD	CABD	CABD	ABCD
A:	ACBD	ACBD	ACBD	CABD	ABCD
D:	ACBD	DACB	ACDB	DCAB	ABCD
A:	ACBD	ADCB	ACDB	ADCB	ABCD
C:	ACBD	CADB	CADB	CADB	ABCD
A:	ACBD	ACDB	ACDB	ACDB	ABCD
C:	ACBD	CADB	CADB	ACDB	ABCD
C:	ACBD	CADB	CADB	CADB	ABCD
E:	ACBDE	CADBE	CADBE	CADBE	ABCDE
E:	ACBDE	ECADB	CADEB	CAEDB	ABCDE

Listas Auto-Organizadas
vantagens e desvantagens

Quatro formas de organizar os elementos que chegam a uma lista:

- **À CABEÇA – quando o elemento é encontrado colocá-lo à cabeça.**
- **TRANSPOSIÇÃO - quando o elemento é encontrado troca-o com o seu predecessor a menos que já esteja à cabeça.**
- **CONTAGEM – ordena a lista pelo número de vezes que cada elemento é acedido.**

??

Análise de Complexidade
(leituras)

Sedgewick, Cap.4, pp. 127-153

Sedgewick, Cap.3, pp. 69-126

- Comprender o conceito de Tipo Abstracto de Dados (TAD)
- Conhecer e compreender as seguintes estruturas de dados:
  - matrizes;
  - pilhas; filas
  - listas ligadas
  - cadeias de caracteres
  - estruturas de dados compostas
- Saber como desenhar um programa simples de simulação de filas de espera

Análise de Complexidade

… bom trabalho, FIM!