ESTRUTURA DE DADOS – EDAS2

Tecnologia em Análise e Desenvolvimento de Sistemas

2º. Semestre – 2019

Outras Listas e Lista Circular Encadeada

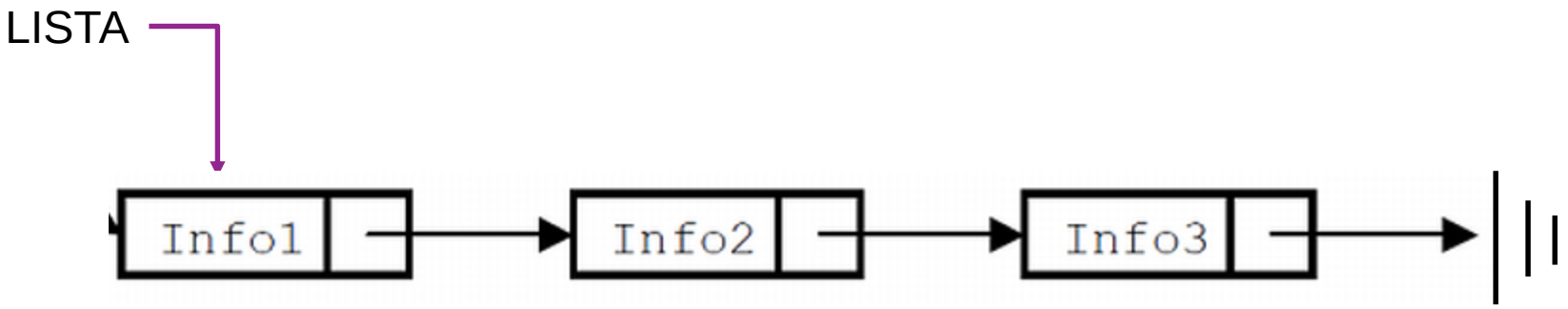
18/09/2019

Profa. Dra. Janaina Cintra Abib

Tipos de Listas (Ordenadas ou Não)

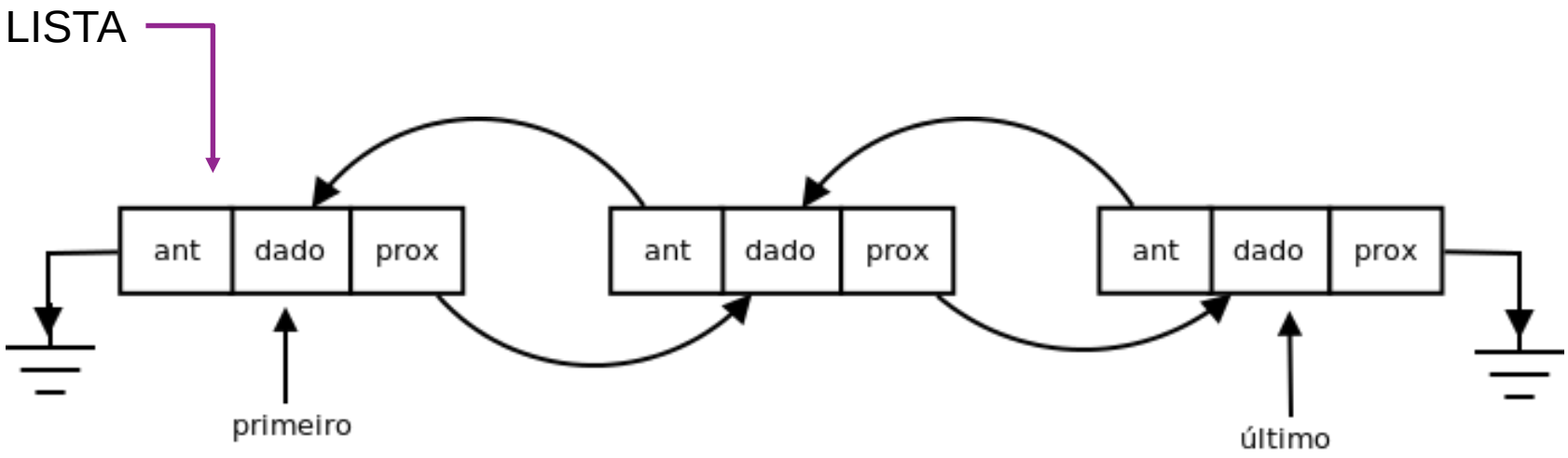
- **Simplesmente Encadeada**
  - Estamos em sequência.
- **Duplamente Encadeada**
  - Sabemos de onde viemos e para onde vamos.
- **Circular**
  - Ela nunca acaba, e seguimos na mesma direção.
- **Circular Duplamente Encadeada**
  - Podemos ir e voltar no anel lógico.
- **Com Descritor**
  - Quem sou, quantos nós tenho, etc.
- **Com Cabeça e Cauda**

Simplesmente Encadeada - Representação



ÚLTIMO NÓ
O próximo aponta para o NULL

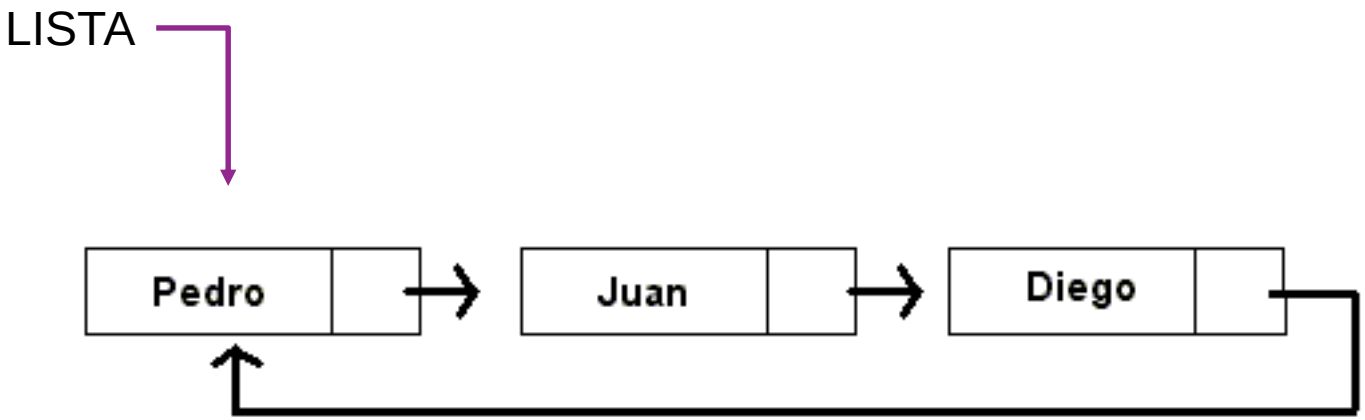
Duplamente Encadeada - Representação



PRIMEIRO NÓ
O anterior aponta para o NULL

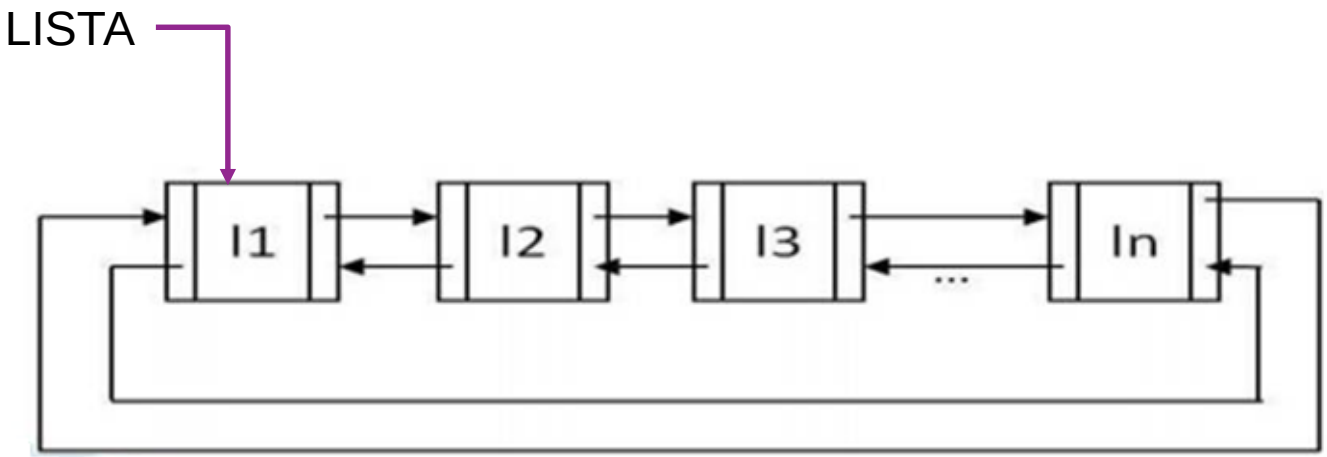
ÚLTIMO NÓ
O próximo aponta para o NULL

Circular Simplesmente Encadeada - Representação

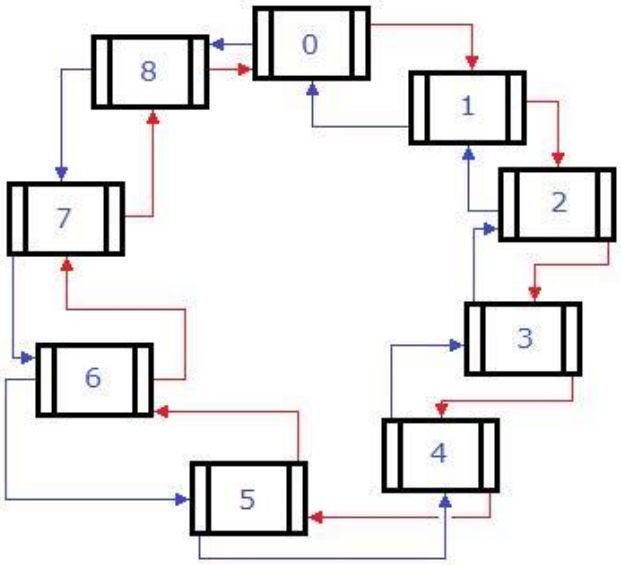


ÚLTIMO NÓ
O próximo aponta para o PRIMEIRO

Circula Duplamente Encadeada - Representação

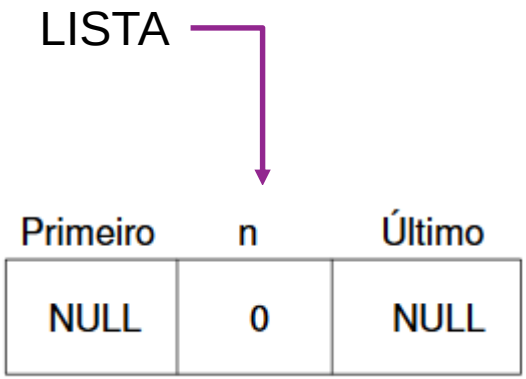


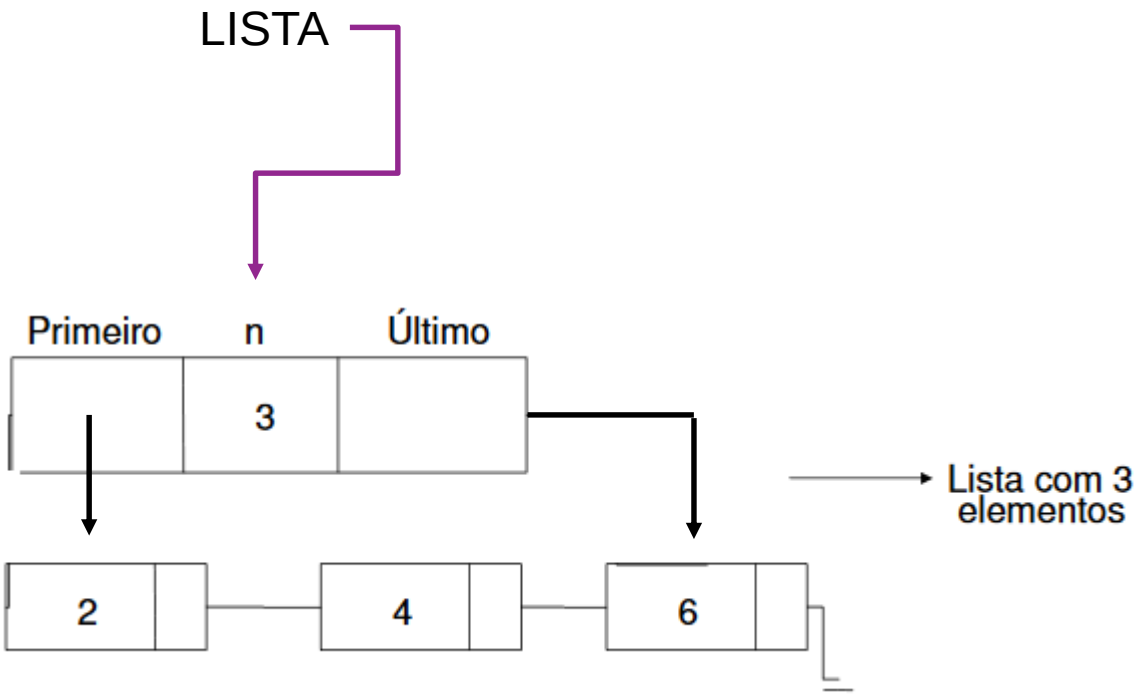
PRIMEIRO NÓ
O anterior aponta para o ÚLTIMO



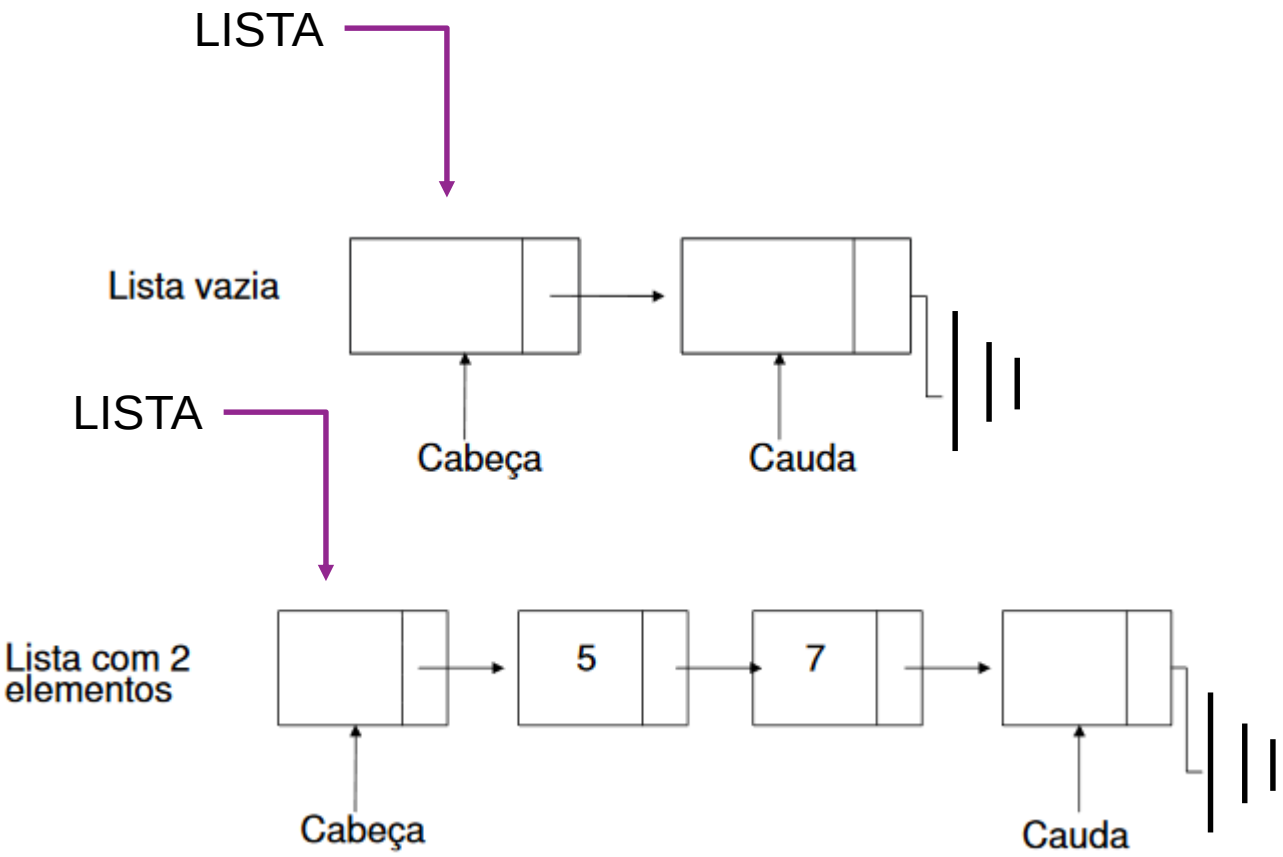
ÚLTIMO NÓ
O próximo aponta para o PRIMEIRO

Lista com Descritor - Representação





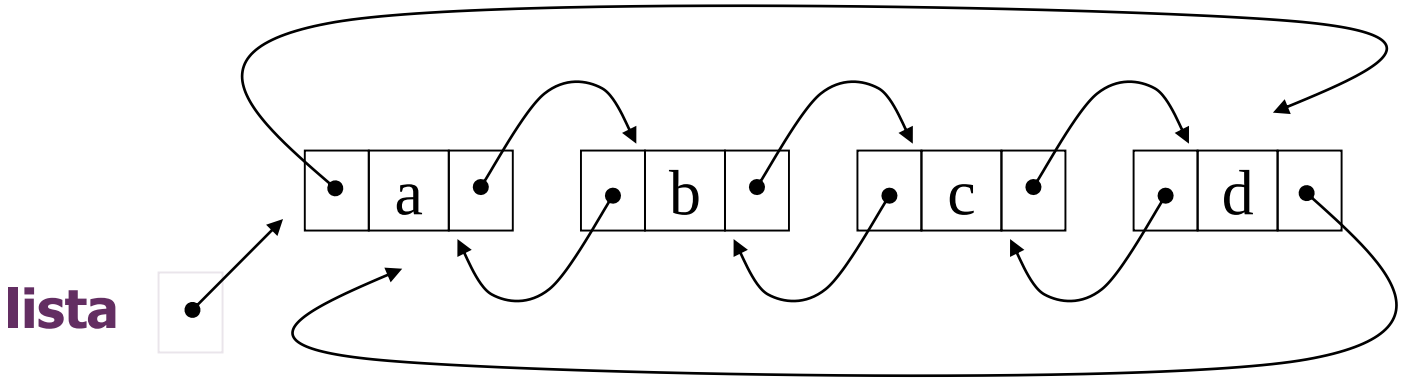
Com Cabeça e Cauda - Representação



Lista Circular Duplamente Encadeada

- Cada nó possui dois ponteiros: um para o elemento anterior e outro para o próximo elemento (ant e prox)

- O anterior do primeiro aponta para o último e o próximo do último aponta para o primeiro



Estrutura da Lista Circular DUPLA

```
struct no
{
    int info;
    struct no *ant;
    struct no *prox;
};

typedef struct no CIRC;
```

```
void criarLista (CIRC **l)
{
    *l = NULL;
}
```

```
int inserirInicio(CIRC **l, int valor)
{
    int resultado = FALSE;
    CIRC *aux;
    CIRC *temp;

    aux = (CIRC *) malloc(sizeof(CIRC));

    if(aux != NULL)
    {
        aux->info = valor;
        if(*l == NULL) // lista vazia
        {
            aux->prox = aux;
            aux->ant = aux;
            *l = aux;
        }
        else
        {
            temp = *l;
            aux->prox = temp;
            aux->ant = temp->ant;
            temp->ant->prox = aux;
            temp->ant = aux;
            *l = aux;
        }
        resultado = TRUE;
    }
    return(resultado);
}
```

Inserção INÍCIO

```
int inserirFim(CIRC **l, int valor)
{
     int resultado = FALSE;
     CIRC *aux;
     CIRC *temp;
     aux = (CIRC *) malloc(sizeof(CIRC));
     if(aux != NULL)
     {
          aux->info = valor;
          if(*l == NULL) // lista vazia
          {
               aux->prox = aux;
               aux->ant = aux;
               *l = aux;
          }
          else
          {
               temp = *l;
               while(temp->prox != *l)
               {
                    temp = temp->prox;
               }
               aux->prox = temp->prox;
               aux->ant = temp;
               temp->prox = aux;
               temp = *l;
               temp->ant = aux;
          }
          resultado = TRUE;
     }
     return(resultado);
}
```

Inserção FIM

```
void escreverLista(CIRC *l)
{
      CIRC* aux;
      int cont = 0;

      aux = l;

      if (aux == NULL)
         printf("\n\nA lista esta vazia !");
      else
      {
          do
          {          cont++;
                     printf("Elemento %d = %d ", cont, aux->info);
                     aux = aux->prox;
          }while(aux != l);
      }
}
```

Escrever Lista

Contar Elementos da Lista

```
int contarLista(CIRC *l)
{
      CIRC* aux;
      int cont = 0;

      aux = l;

      if (aux != NULL)
      {
          do
          {
              cont++;
              aux = aux->prox;
          }while(aux != l);
      }
      return(cont);
}
```

Apagar Lista

```
int apagarLista(CIRC **l)
{
    int resultado = FALSE;
    CIRC *p;

    p = *l;

    if(p == NULL)
    {
        resultado = TRUE;
    }
    else
    {
        while(verificarVazia(*l) == FALSE)
        {
            p = *l;
            *l = p->prox;
            free(p);
        }
        free(*l);
        *l = NULL;
        resultado = TRUE;
    }
    return(resultado);
}
```

```
int verificarVazia(CIRC *l)
{
    int resultado = FALSE;

    if (l == NULL)
    {
        resultado = TRUE;
    }
    return(resultado);
}
```

Agora é a sua vez!

- Considerando uma lista CIRCULAR duplamente encadeada, faça funções para:
  - Inserir uma informação no início da lista;
  - Inserir uma informação no final da lista;
  - Inserir ordenado na lista;
  - Mostrar os valores da lista;
  - Consultar um valor da lista;
  - Remover um valor da lista;
  - Contar quantos nós tem a lista.

Aplicação de Listas

$$C_{700x900} = \begin{bmatrix} 1 & 0 & 0 & 3 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & -1 & 4 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

700 x 900 = 630.000 elementos

Matriz esparsa com 6 elementos **não nulos**

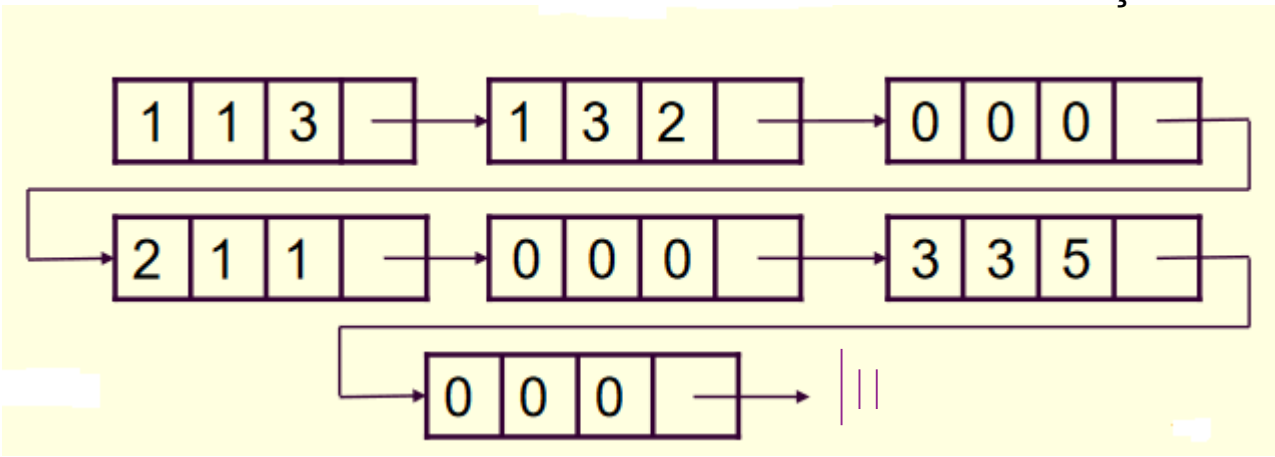
Aplicação Listas

- Representação de Matriz

$$A_{3x3} = \begin{bmatrix} 3 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

linha	coluna	valor	prox

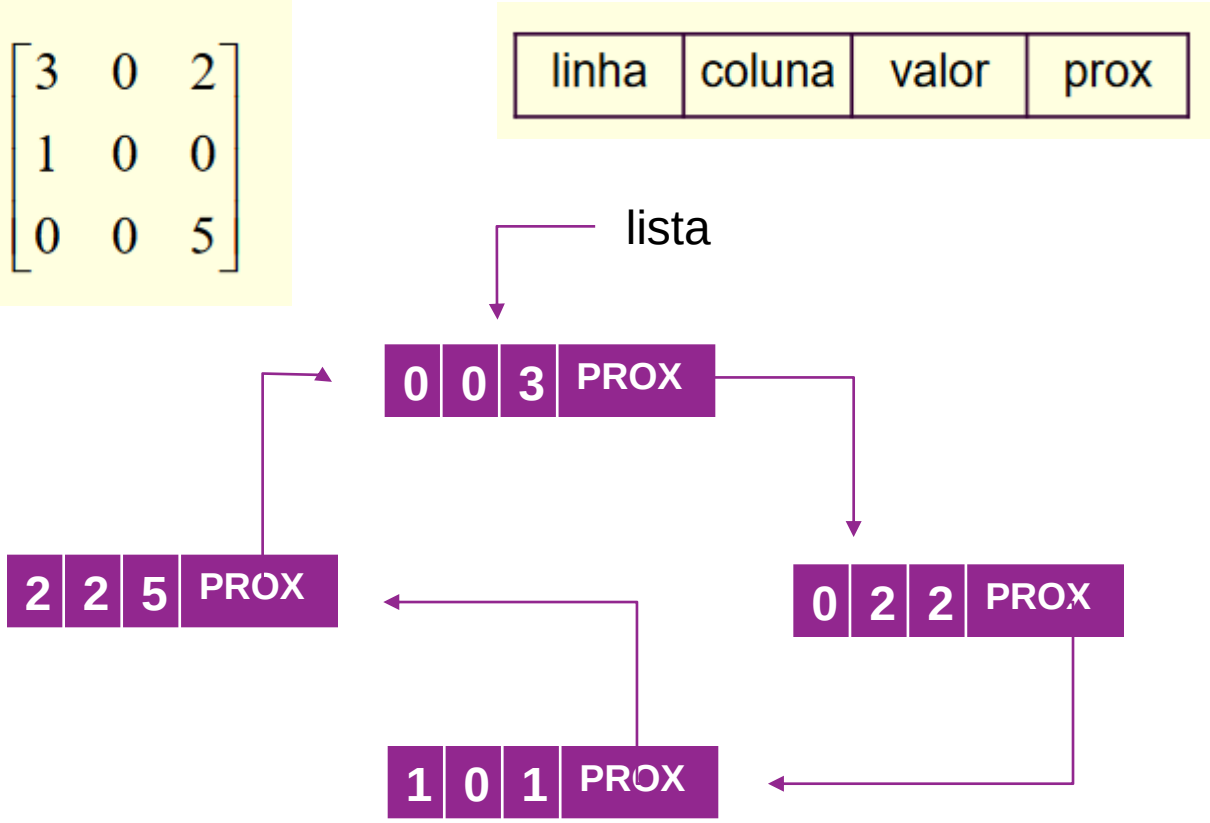
Representando a mudança de linha



Aplicação Listas

- Representação de Matriz - Otimização

$$A_{3x3} = \begin{bmatrix} 3 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$



Problema de Josephus – Lista Circular

- Imagine que N pessoas decidem eleger um líder usando um método de eliminações sucessivas, ficando como líder o último a ser eliminado. As N pessoas dispõe-se num círculo e elimina-se a M-ésima pessoa, cerrando fileiras com os restantes.

- A pessoa a ser eleita depende do N e do M. Para o exemplo seguinte, a pessoa na posição inicial 4 seria a eleita.