Listas Circulares

Com base no que foi visto implemente a operação destruir() que compõem o TAD LISTA_CIRCULAR.

```
1  typedef struct nodo
2  {
3     int inf;
4     struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_CIRCULAR;
7  void cria_lista (LISTA_CIRCULAR *);
8  int eh_vazia (LISTA_CIRCULAR);
9  int tam (LISTA_CIRCULAR);
10 void ins (LISTA_CIRCULAR *, int, int);
11 int recup (LISTA_CIRCULAR, int);
12 void ret (LISTA_CIRCULAR *, int);
13 void destruir (LISTA_CIRCULAR) ;
```

UNIVASF

```c
void destruir (LISTA_CIRCULAR l)
{
    if (l)
    {
        LISTA_CIRCULAR aux;
        for (aux=l->next; aux!=l ; aux = aux->next)
            free (aux);
        free (aux);
    }
}
```

```
1  void destruir (LISTA_CIRCULAR *pl)
2  {
3      if (*pl)
4      {
5          LISTA_CIRCULAR aux;
6          for (aux=(*pl)->next; aux!=*pl ; aux = aux->next)
7              free (aux);
8          free (aux);
9          *pl=NULL;
10     }
11 }
```

Listas Circulares – Nó de Cabeçalho

O conceito de ***nó de cabeçalho*** também pode ser empregado nas listas circulares.

A implementação de um TAD LISTA_CIRCULAR_COM_NC é sugerida como uma exercício de fixação.

Listas Duplamente Encadeadas

UNIVASF

Listas Duplamente Encadeadas

Como vimos, uma lista circular possui vantagem sobre uma lista linear. Contudo, esta ainda possui limitações.

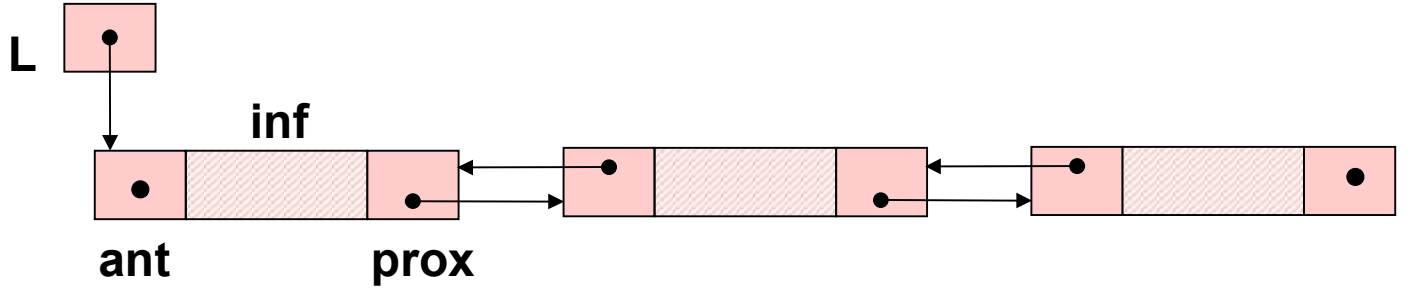
Por exemplo, não podemos percorrê-la no sentido contrário o que impõe, por exemplo, a necessidade de se ter um ponteiro para o antecessor para inserirmos ou retirarmos um k-ésimo elemento.

Com o objetivo de sanar esta limitação surgiram as ***listas duplamente encadeadas***.

UNIVASF

Listas Duplamente Encadeadas

Em uma ***lista duplamente encadeada*** os elementos possuem três campos: o campo ***inf*** o qual contém a informação, o campo ***ant*** que possui um ponteiro para o elemento antecessor e o campo ***prox*** que é uma referência para o próximo elemento.



```c
typedef struct nodo
{
    int inf;
    struct nodo * ant;
    struct nodo * prox;
}NODO;
typedef NODO * LISTA_DUP_ENC;
void cria_lista (LISTA_DUP_ENC *);
int eh_vazia (LISTA_DUP_ENC);
int tam (LISTA_DUP_ENC);
void ins (LISTA_DUP_ENC *, int, int);
int recup (LISTA_DUP_ENC, int);
void ret (LISTA_DUP_ENC *, int);
void destruir (LISTA_DUP_ENC);
```

Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação cria_lista() que compõem o TAD LISTA_DUP_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * ant;
5      struct nodo * prox;
6  }NODO;
7  typedef NODO * LISTA_DUP_ENC;
8  void cria_lista (LISTA_DUP_ENC *);
9  int eh_vazia (LISTA_DUP_ENC);
10 int tam (LISTA_DUP_ENC);
11 void ins (LISTA_DUP_ENC *, int, int);
12 int recup (LISTA_DUP_ENC, int);
13 void ret (LISTA_DUP_ENC *, int);
14 void destruir (LISTA_DUP_ENC);
```

```
1  void cria_lista (LISTA_DUP_ENC *pl)
2  {
3      *pl=NULL;
4  }
```

Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação eh_vazia que compõem o TAD LISTA_DUP_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * ant;
5      struct nodo * prox;
6  }NODO;
7  typedef NODO * LISTA_DUP_ENC;
8  void cria_lista (LISTA_DUP_ENC *);
9  int eh_vazia (LISTA_DUP_ENC);
10 int tam (LISTA_DUP_ENC);
11 void ins (LISTA_DUP_ENC *, int, int);
12 int recup (LISTA_DUP_ENC, int);
13 void ret (LISTA_DUP_ENC *, int);
14 void destruir (LISTA_DUP_ENC);
```

```c
int eh_vazia (LISTA_DUP_ENC l)
{
    return (!l);
}
```

Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação tam() que compõem o TAD LISTA_DUP_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * ant;
5      struct nodo * prox;
6  }NODO;
7  typedef NODO * LISTA_DUP_ENC;
8  void cria_lista (LISTA_DUP_ENC *);
9  int eh_vazia (LISTA_DUP_ENC);
10 int tam (LISTA_DUP_ENC);
11 void ins (LISTA_DUP_ENC *, int, int);
12 int recup (LISTA_DUP_ENC, int);
13 void ret (LISTA_DUP_ENC *, int);
14 void destruir (LISTA_DUP_ENC);
```

```c
int tam (LISTA_DUP_ENC l)
{
    int cont;
    for (cont=0; l; cont++, l = l->prox);
    return (cont);
}
```

Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação ins() que compõem o TAD LISTA_DUP_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * ant;
5      struct nodo * prox;
6  }NODO;
7  typedef NODO * LISTA_DUP_ENC;
8  void cria_lista (LISTA_DUP_ENC *);
9  int eh_vazia (LISTA_DUP_ENC);
10 int tam (LISTA_DUP_ENC);
11 void ins (LISTA_DUP_ENC *, int, int);
12 int recup (LISTA_DUP_ENC, int);
13 void ret (LISTA_DUP_ENC *, int);
14 void destruir (LISTA_DUP_ENC);
```

Listas Duplamente Encadeadas

Dicas:

A posição é válida?

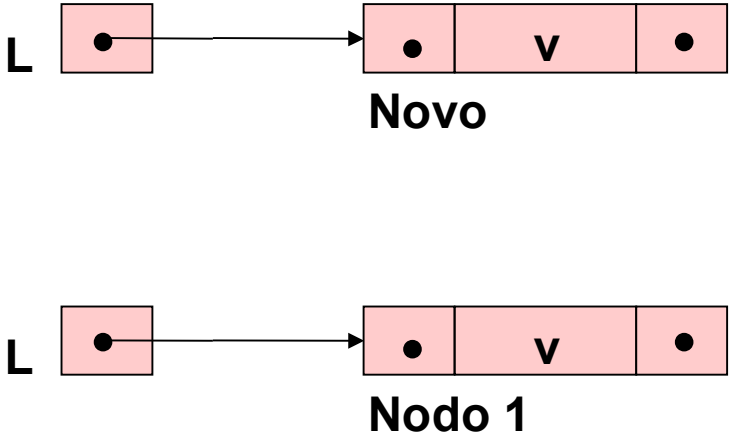
Tem espaço na memória para armazenar mais um elemento?

Todas as situações de inserção são tratadas da mesma forma?

UNIVASF

Listas Duplamente Encadeadas

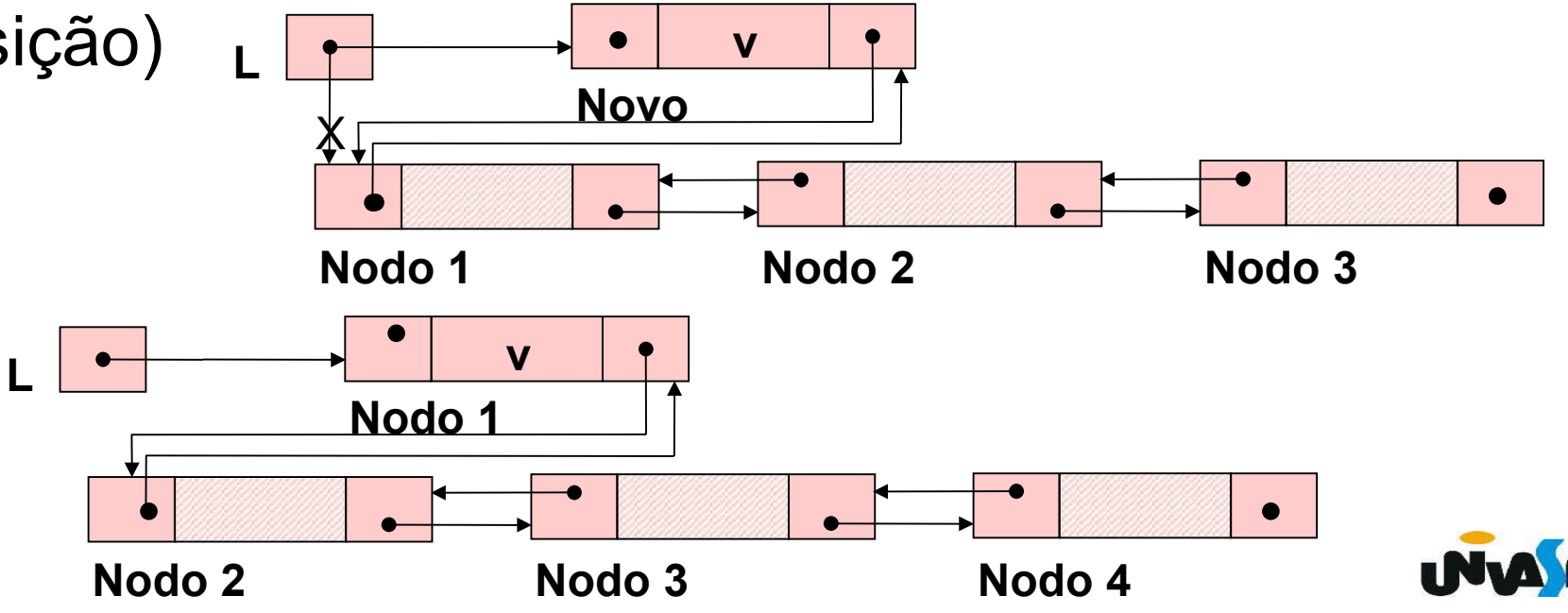
Esquema do processo da inserção de um nó da lista duplamente encadeada. (situação um inserção de nó em lista vazia)



Listas Duplamente Encadeadas

Esquema do processo da inserção de um nó da lista duplamente encadeada.

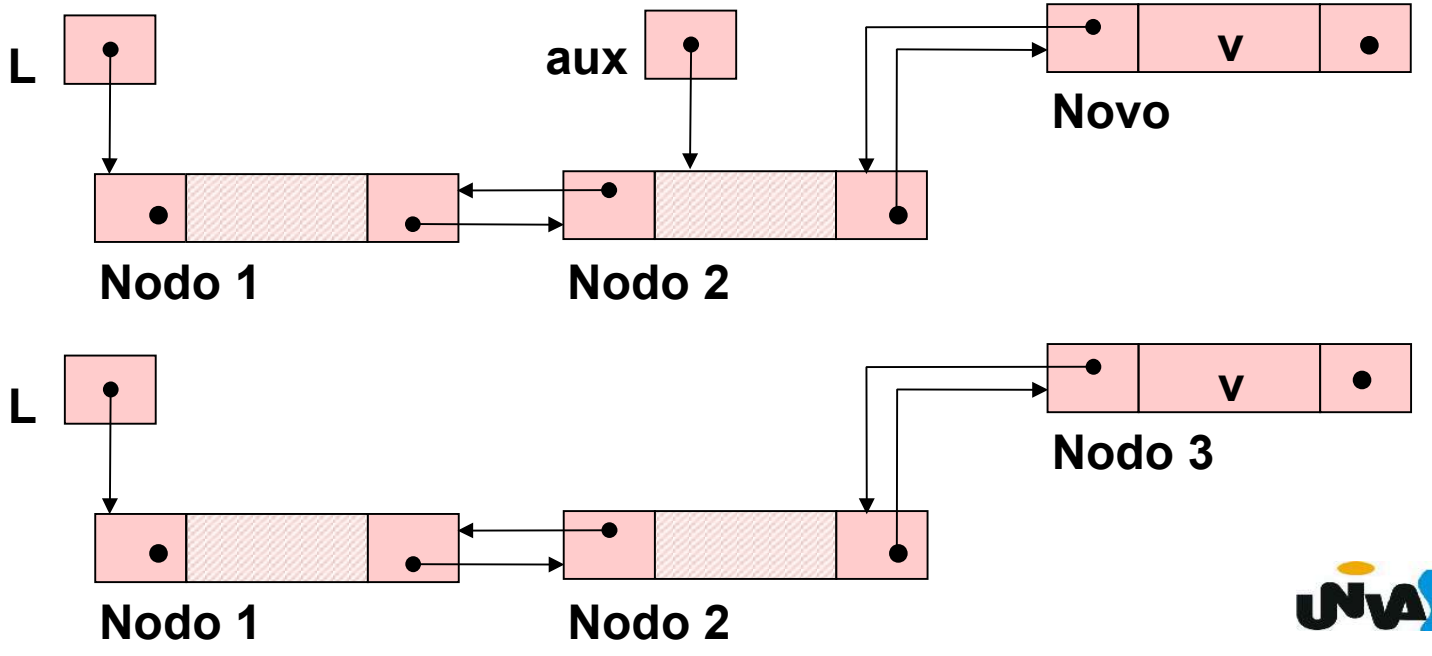
(situação dois, inserção de um elemento na primeira posição)



Listas Duplamente Encadeadas

Esquema do processo da inserção de um nó da lista duplamente encadeada.

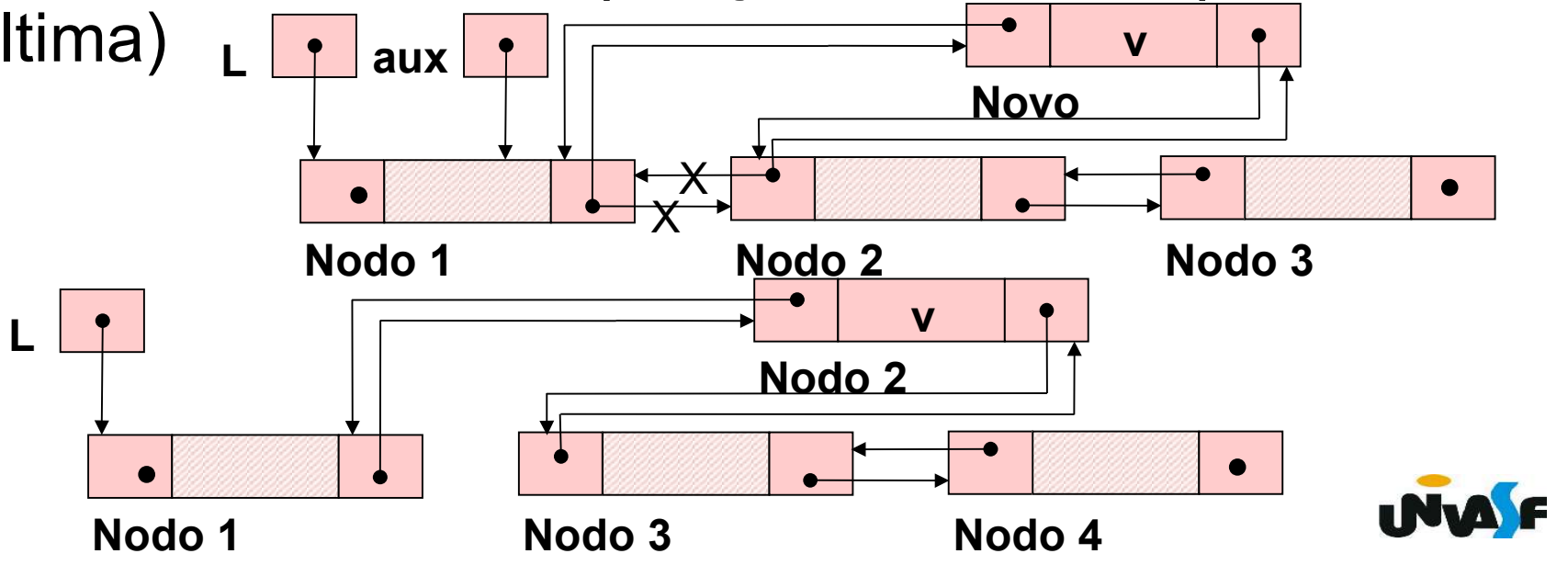
(situação três, inserção de um elemento na última posição)



Listas Duplamente Encadeadas

Esquema do processo da inserção de um nó da lista duplamente encadeada.

(situação quatro, inserção de um elemento em uma lista não vazia em uma posição distinta da primeira e da última)



```c
void ins (LISTA_DUP_ENC *pl, int v, int k) {
    LISTA_DUP_ENC novo;
    if (k < 1 || k > tam(*pl)+1) {
       printf ("\nERRO! Posição invalida para insercao.\n");
       exit (1);
    }
    novo = (LISTA_DUP_ENC) malloc (sizeof(NODO));
    if (!novo) {
       printf ("\nERRO! Memoria insuficiente!\n");
       exit (2);
    }
    novo->inf = v;
```

```
13      if (k==1) { /*situações um e dois*/
14          novo->ant = NULL;
15          novo->prox = *pl;
16          *pl = novo;
17          if ((*pl)->prox) /*situação dois*/
18              (*pl)->prox->ant=novo;
19      } else { /*situações três e quatro*/
20          LISTA_DUP_ENC aux;
21          for (aux=*pl; k>2; aux=aux->prox, k--);
22          novo->prox = aux->prox;
23          aux->prox = novo;
24          novo->ant=aux;
25          if (novo->prox) /*situação quatro*/
26              novo->prox->ant=novo;
27      }
28  }
```

Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação recup() que compõem o TAD LISTA_DUP_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * ant;
5      struct nodo * prox;
6  }NODO;
7  typedef NODO * LISTA_DUP_ENC;
8  void cria_lista (LISTA_DUP_ENC *);
9  int eh_vazia (LISTA_DUP_ENC);
10 int tam (LISTA_DUP_ENC);
11 void ins (LISTA_DUP_ENC *, int, int);
12 int recup (LISTA_DUP_ENC, int);
13 void ret (LISTA_DUP_ENC *, int);
14 void destruir (LISTA_DUP_ENC);
```

UNIVASF

```c
int recup (LISTA_DUP_ENC l, int k)
{
    if (k < 1 || k > tam(l))
    {
        printf ("\nERRO! Consulta invalida.\n");
        exit (3);
    }
    for (;k>1;l=l->prox,k--) ;
    return (l->inf);
}
```

Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação ret() que compõem o T A D LISTA_DUP_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * ant;
5      struct nodo * prox;
6  }NODO;
7  typedef NODO * LISTA_DUP_ENC;
8  void cria_lista (LISTA_DUP_ENC *);
9  int eh_vazia (LISTA_DUP_ENC);
10 int tam (LISTA_DUP_ENC);
11 void ins (LISTA_DUP_ENC *, int, int);
12 int recup (LISTA_DUP_ENC, int);
13 void ret (LISTA_DUP_ENC *, int);
14 void destruir (LISTA_DUP_ENC);
```

Listas Duplamente Encadeadas

Dicas:

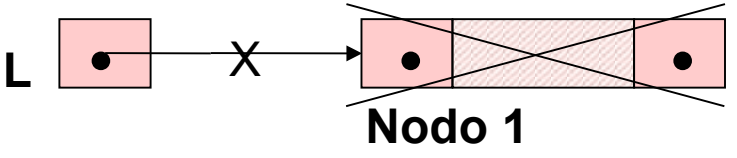
A posição é válida?

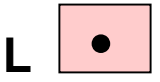
Todas as situações de remoção são tratadas da mesma forma?

UNIVASF

Listas Duplamente Encadeadas

Esquema do processo da retirada de um nó da lista duplamente encadeada. (situação um, remoção do único elemento)

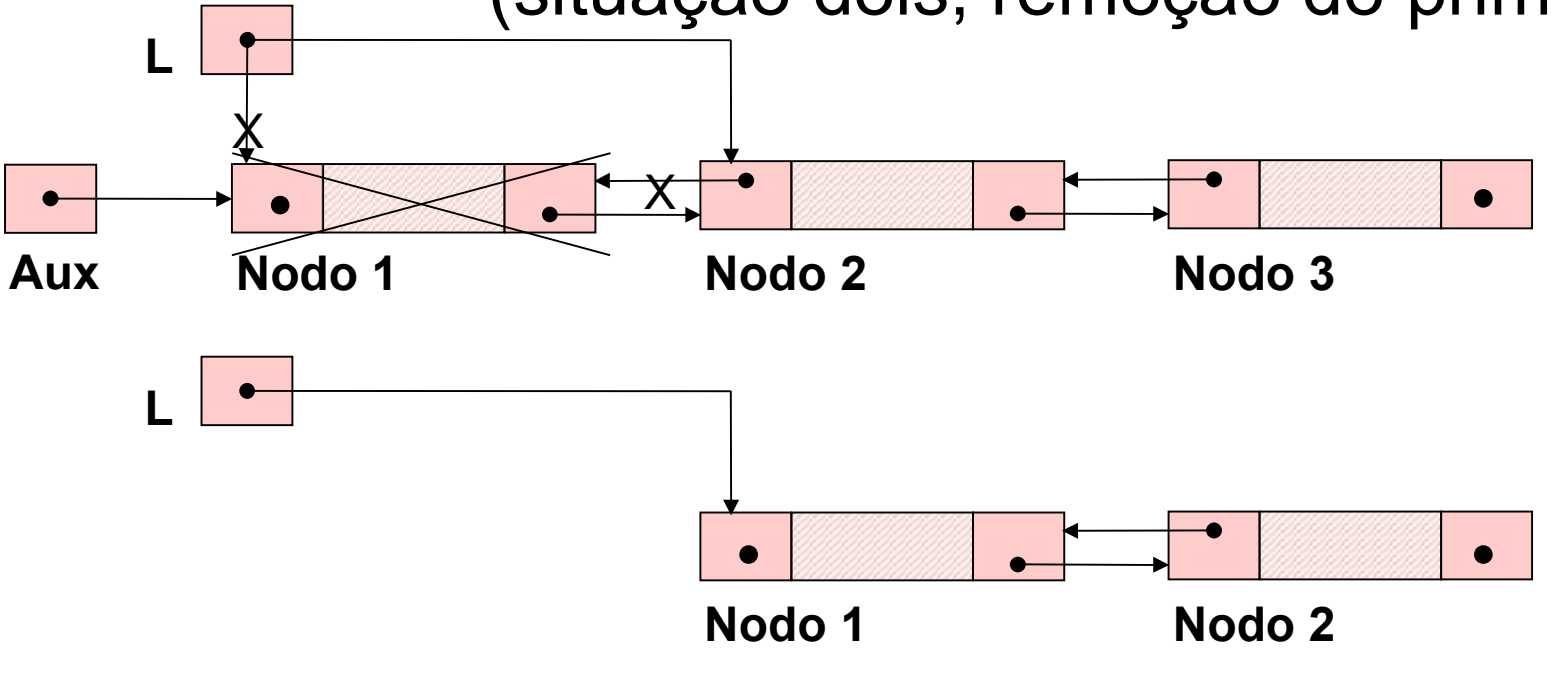




Listas Duplamente Encadeadas

Esquema do processo da retirada de um nó da lista duplamente encadeada.

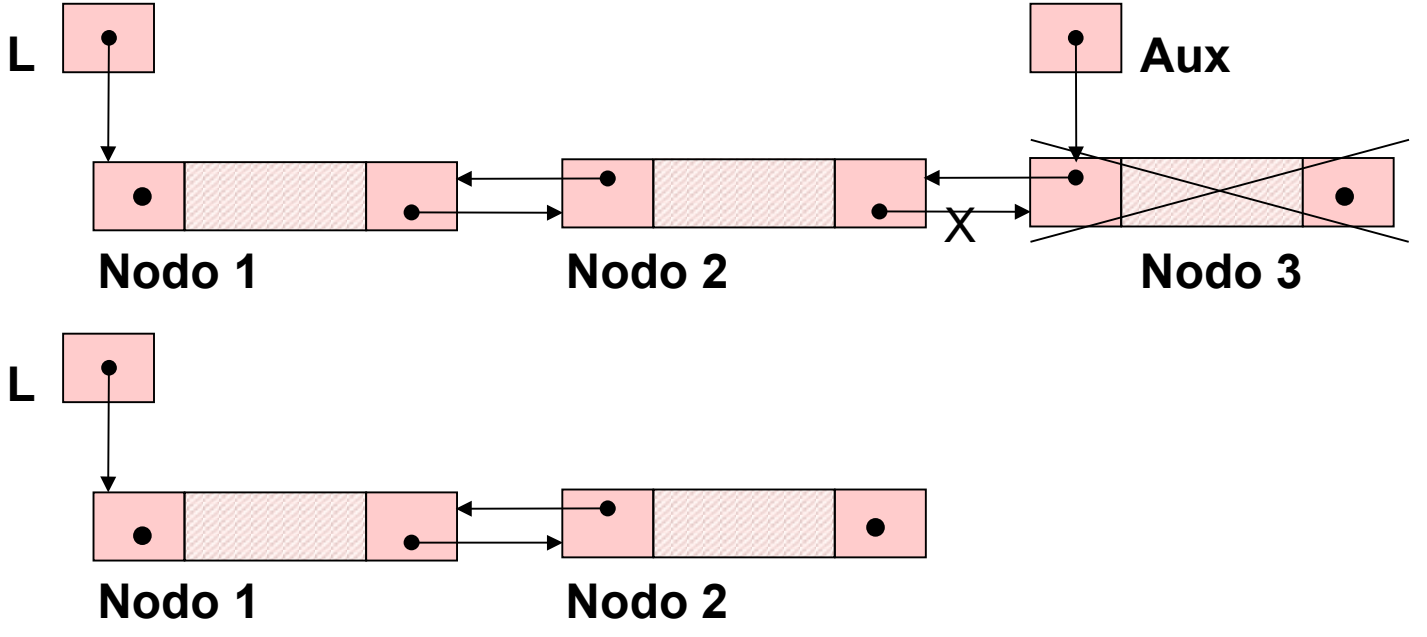
(situação dois, remoção do primeiro)



Listas Duplamente Encadeadas

Esquema do processo da retirada de um nó da lista duplamente encadeada.

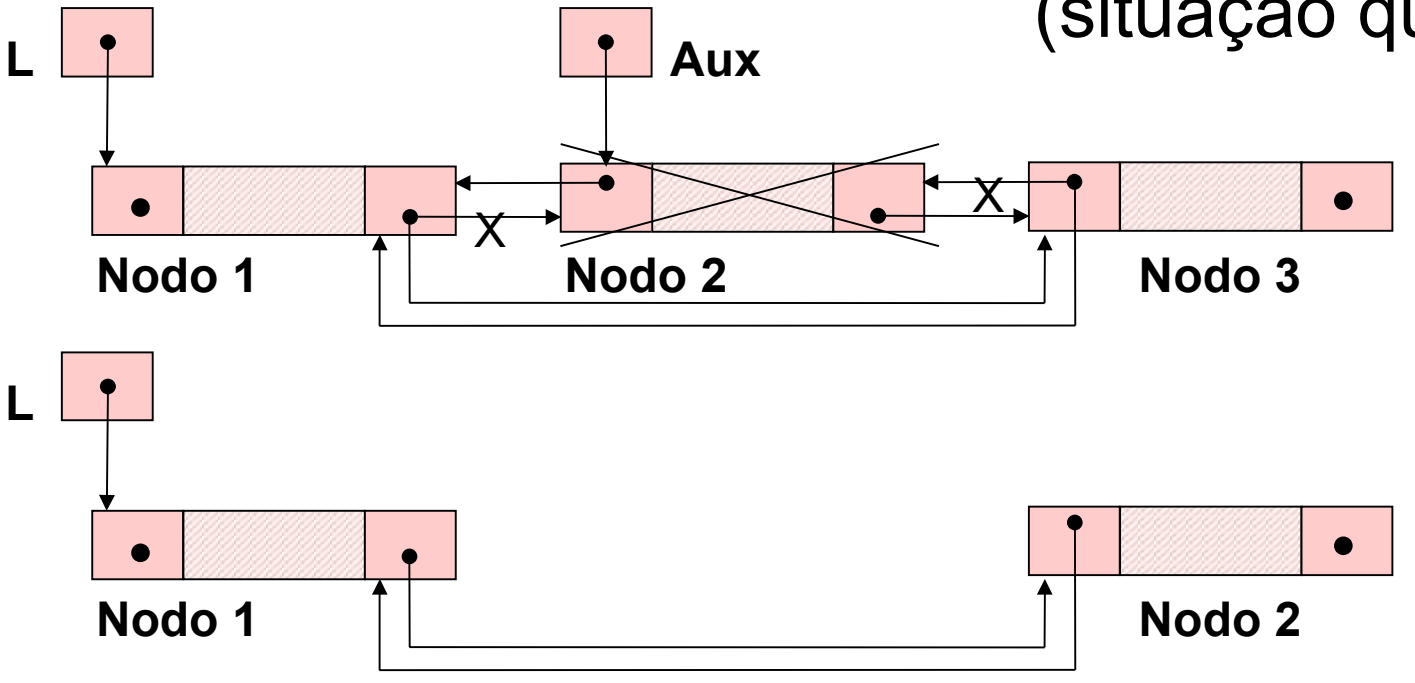
(situação três, remoção do último)



Listas Duplamente Encadeadas

Esquema do processo da retirada de um nó da lista duplamente encadeada.

(situação quatro)



```c
void ret (LISTA_DUP_ENC *pl, int k) {
   LISTA_DUP_ENC aux;
   if (k < 1 || k > tam(*pl)) {
      printf ("\nERRO! Posição invalida para retirada.\n");
      exit (4);
   }
   if (k==1) { /*situações um e dois*/
      aux = *pl;
      *pl = aux->prox;
      if (*pl) /*situação dois*/
         (*pl)->ant=NULL;
      free (aux);
   } else { /*situações três e quatro*/
      for (aux=(*pl)->prox; k>2; k--, aux=aux->prox);
      aux->ant->prox = aux->prox;
      if (aux->prox) /*situação quatro*/
         aux->prox->ant = aux->ant;
      free (aux);
   }
}
```

Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação destruir() que compõem o TAD LISTA_DUP_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * ant;
5      struct nodo * prox;
6  }NODO;
7  typedef NODO * LISTA_DUP_ENC;
8  void cria_lista (LISTA_DUP_ENC *);
9  int eh_vazia (LISTA_DUP_ENC);
10 int tam (LISTA_DUP_ENC);
11 void ins (LISTA_DUP_ENC *, int, int);
12 int recup (LISTA_DUP_ENC, int);
13 void ret (LISTA_DUP_ENC *, int);
14 void destruir (LISTA_DUP_ENC);
```

Listas Duplamente Encadeadas

Implemente, no TAD LISTA_DUP_ENC, a seguinte operação:

void inverter_lista (LISTA_DUP_ENC *pl);

a qual recebe uma referência para uma lista duplamente encadeada e inverte a ordem de seus elementos.