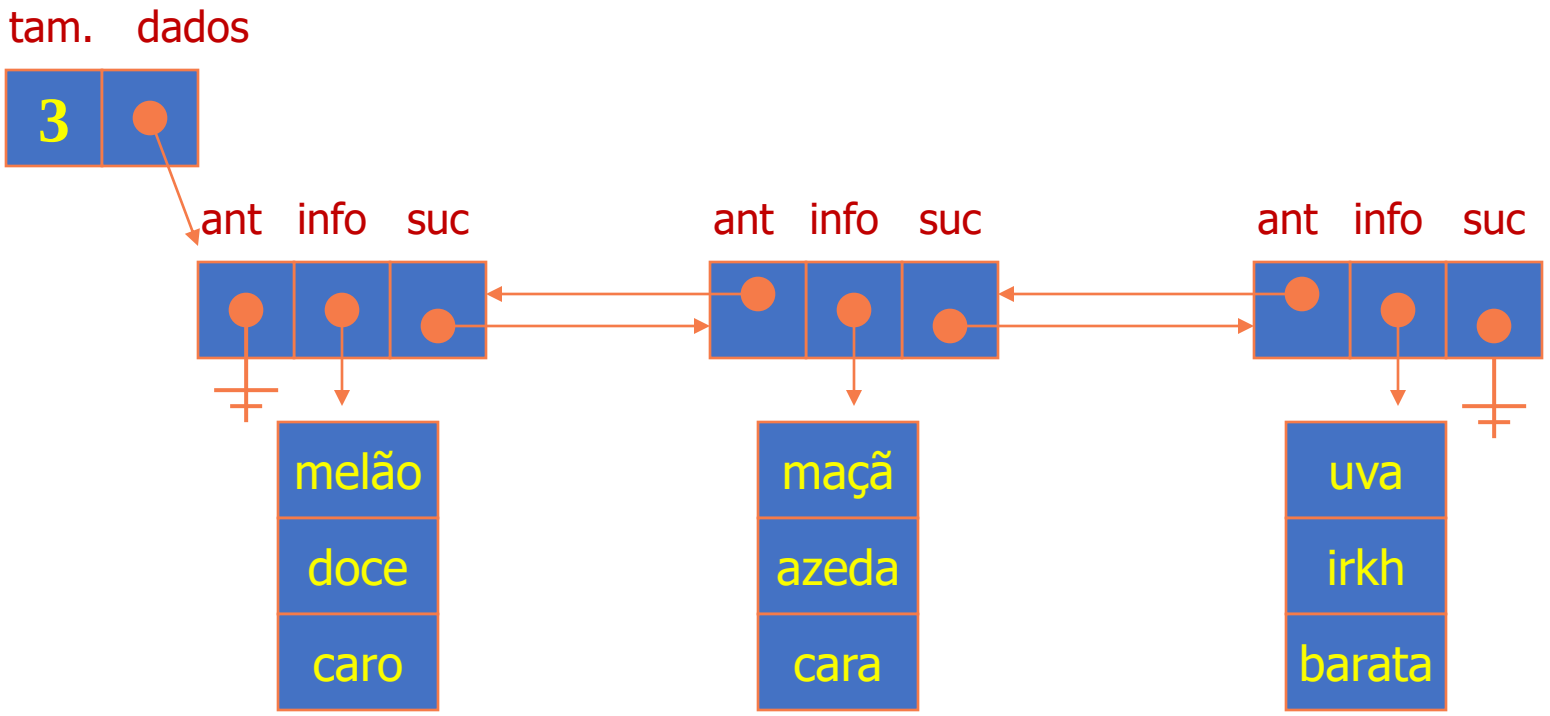
LISTAS ENCADEADAS

LISTAS DUPLAMENTE ENCADEADAS

4.2.3 LISTAS DUPLAMENTE ENCADEADAS

- A Lista Simplesmente Encadeada possui a desvantagem de somente podermos caminhar em uma direção.
  - Para acessar um elemento que “acabamos de passar” precisamos de uma variável auxiliar “anterior”.
  - Para acessar outros elementos ainda anteriores não temos nenhum meio, a não ser começar de novo.
- A Lista Duplamente Encadeada é uma estrutura de lista que permite deslocamento em ambos os sentidos:
  - Útil para representar conjuntos de eventos ou objetos a serem percorridos em dois sentidos.
  - Ex.: Itinerários de ônibus, trem ou avião.
  - Útil também quando realizamos uma busca aproximada e nos movemos para e frente e trás.

4.2.3 LISTAS DUPLAMENTE ENCADEADAS: MODELAGEM



4.2.3 MODELAGEM: CABEÇA DE LISTADUPLA

- Necessitamos:
  - Um ponteiro para o primeiro elemento da lista.
  - Um inteiro para indicar quantos elementos a lista possue.

```
tipo ListaDupla {
  ElementoDuplo  *dados;
  inteiro        tamanho;
};
```

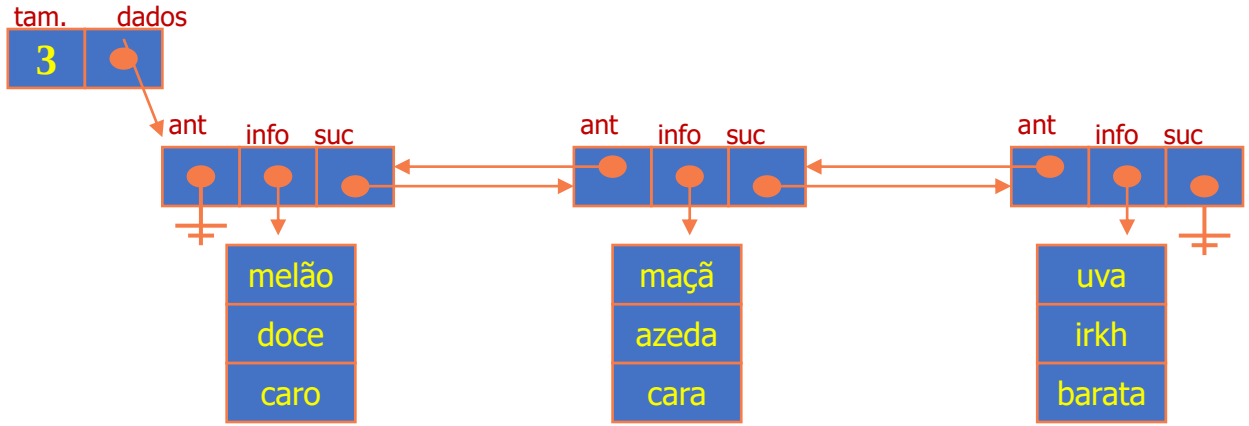
4.2.3 MODELAGEM: ELEMENTO DE LISTA DUPLA

- Necessitamos:
  - Um ponteiro para o elemento anterior na lista.
  - Um ponteiro para o elemento sucessor na lista.
  - Um ponteiro para a informação que vamos armazenar.
- Pseudo-código:

```
tipo ElementoDuplo {
  ElementoDuplo  *anterior;
  ElementoDuplo  *sucessor;
  TipoInfo       *info;
};
```

4.2.4 MODELAGEM DA LISTA DUPLAMENTE ENCADEADA

- Aspecto Funcional:
  - Colocar e retirar dados da lista.
  - Testar se a lista está vazia e outros testes.
  - Inicializa-la e garantir a ordem dos elementos.



MODELAGEM DA LISTA DUPLAMENTE ENCADEADA

- Operações: Colocar e retirar dados da lista:
  - AdicionaDuplo(listaDupla, dado)
  - AdicionaNoInícioDuplo(listaDupla, dado)
  - AdicionaNaPosiçãoDuplo(listaDupla, dado, posição)
  - AdicionaEmOrdemDuplo(listaDupla, dado)

  - RetiraDuplo(listaDupla)
  - RetiraDoInícioDuplo(listaDupla)
  - RetiraDaPosiçãoDuplo(listaDupla, posição)
  - RetiraEspecíficoDuplo(listaDupla, dado)

MODELAGEM DA LISTA DUPLAMENTE ENCADEADA

- Operações: Testar a lista e outros testes:
  - ListaVaziaDuplo(listaDupla)
  - PosicaoDuplo(listaDupla, dado)
  - ContemDuplo(listaDupla, dado)
- Operações: Inicializar ou limpar:
  - CriaListaDupla()
  - DestroiListaDupla(listaDupla)

ALGORITMO CRIALISTADUPLA

```
ListaDupla* FUNÇÃO criaListaDupla()
   "Retorna ponteiro para uma nova cabeça de lista ou NULO"
   variáveis
       ListaDupla  *aLista;
   início
       aLista <- aloque(ListaDupla);
       SE (aLista ~= NULO) ENTÃO
             "So posso inicializar se consegui alocar"
             aLista->tamanho <- 0;
             aLista->dados <- nulo;
       FIM SE
       RETORNE(aLista);
   fim;
```

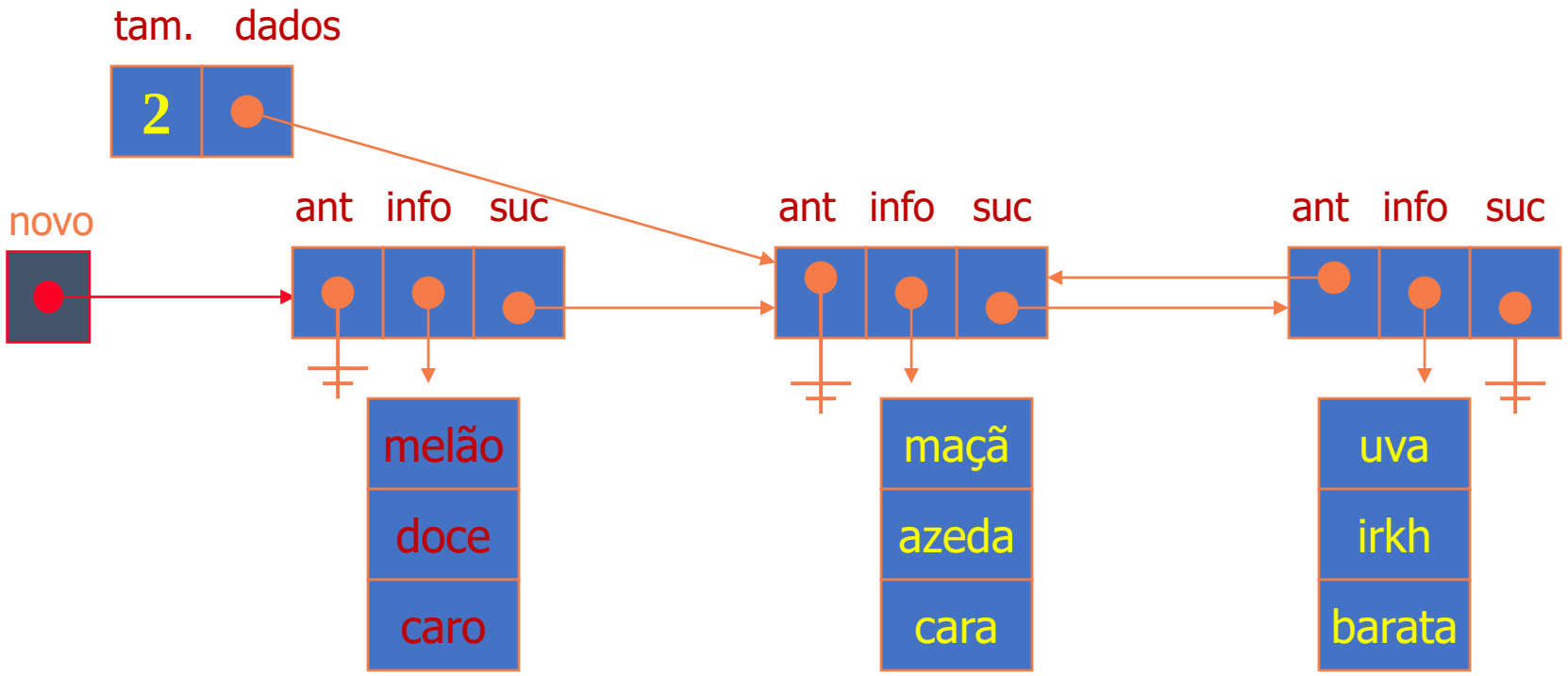
ALGORITMO LISTAVAZIADUPLO

```
Booleano FUNÇÃO listaVaziaDuplo(ListaDupla *aLista)
   início
       SE (aLista->tamanho = 0) ENTÃO
             RETORNE(Verdade)
       SENÃO
             RETORNE(Falso);
   fim;
```

ALGORITMO ADICIONANOINÍCIODUPLO

- Procedimento:
  - Testamos se é possível alocar um elemento.
  - Fazemos o sucessor deste novo elemento ser o primeiro da lista.
  - Fazemos o seu antecessor ser NULO.
  - Fazemos a cabeça de lista apontar para o novo.
- Parâmetros:
  - O tipo info (dado) a ser inserido.
  - ListaDupla.

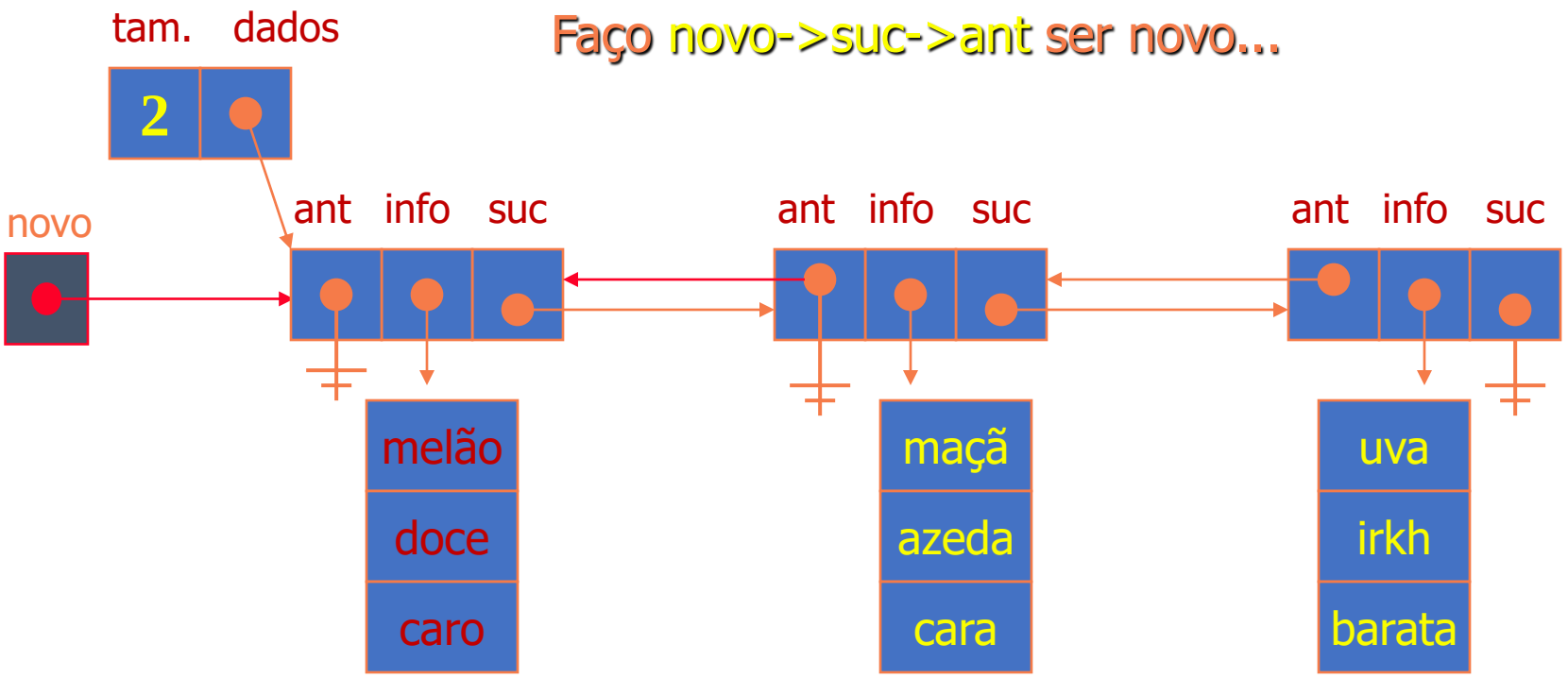
ALGORITMO ADICIONANOINÍCIODUPLO



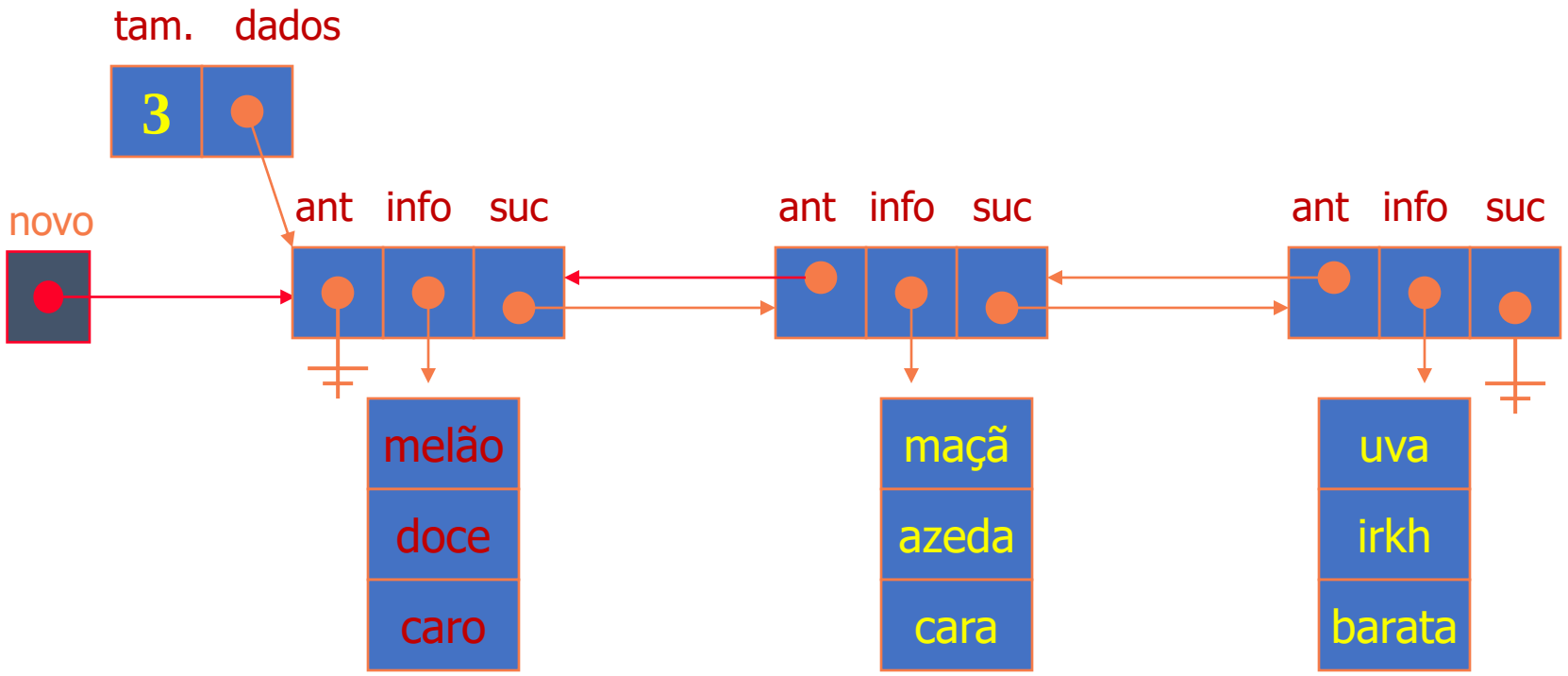
ALGORITMO ADICIONANOINÍCIODUPLO

Caso novo->suc não seja nulo...

Faço novo->suc->ant ser novo...



ALGORITMO ADICIONANOINÍCIODUPLO



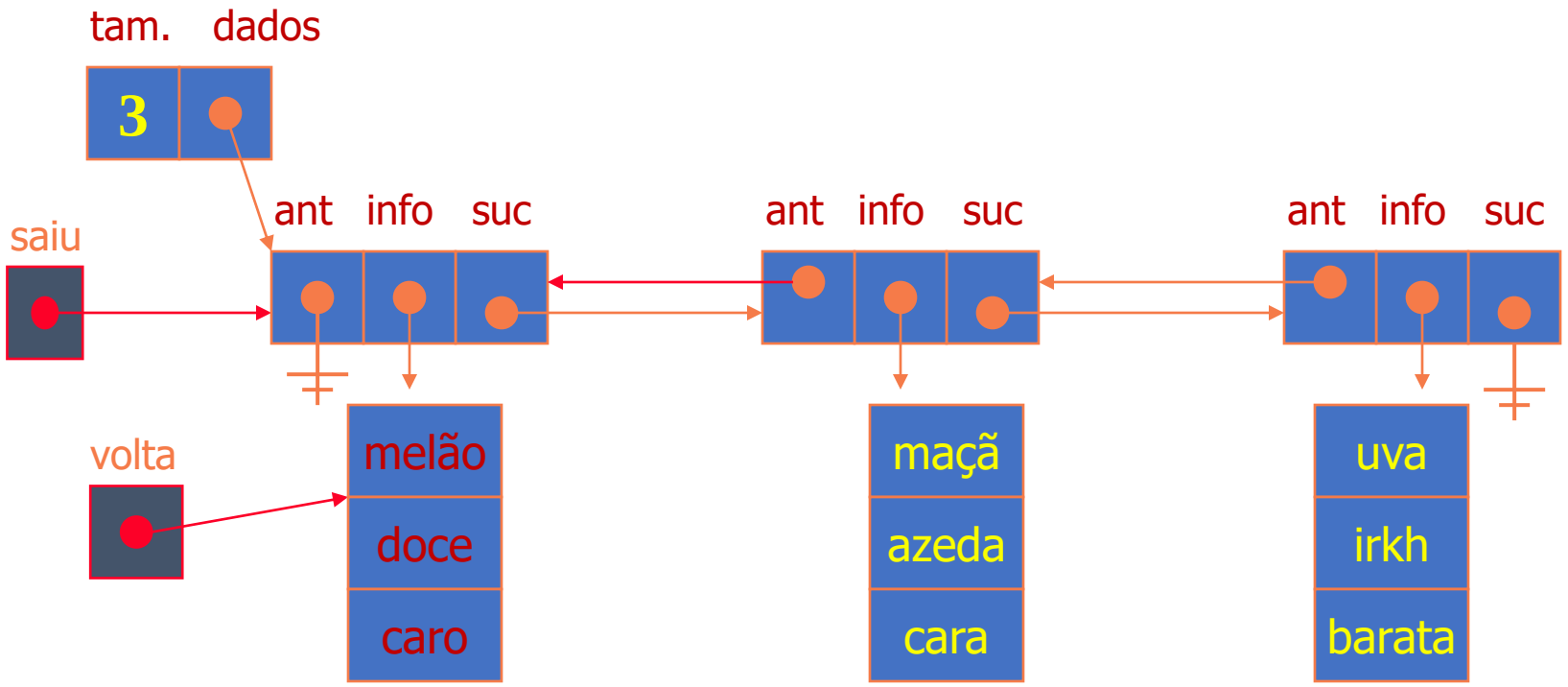
ALGORITMO ADICIONANOINÍCIODUPLO

```
Inteiro FUNÇÃO adicionaNoInícioDuplo(    ListaDupla *aLista,
                                         TipoInfo *dado )
   variáveis
        ElementoDuplo *novo; "Var. auxiliar para o novo elemento"
   início
        novo <- aloque(ElementoDuplo);
        SE (novo = NULO) ENTÃO
          RETORNE( ErroListaCheia );
        SENÃO
          novo->suc <- aLista->dados;
          novo->ant <- NULO;
          novo->info <- dado;
          aLista->dados <- novo;
          SE (novo->suc ~= NULO) ENTÃO
               novo->suc->ant <- novo;
          FIM SE;
          aLista->tamanho <- aLista->tamanho + 1;
          RETORNE(1);
        FIM SE
   fim;
```

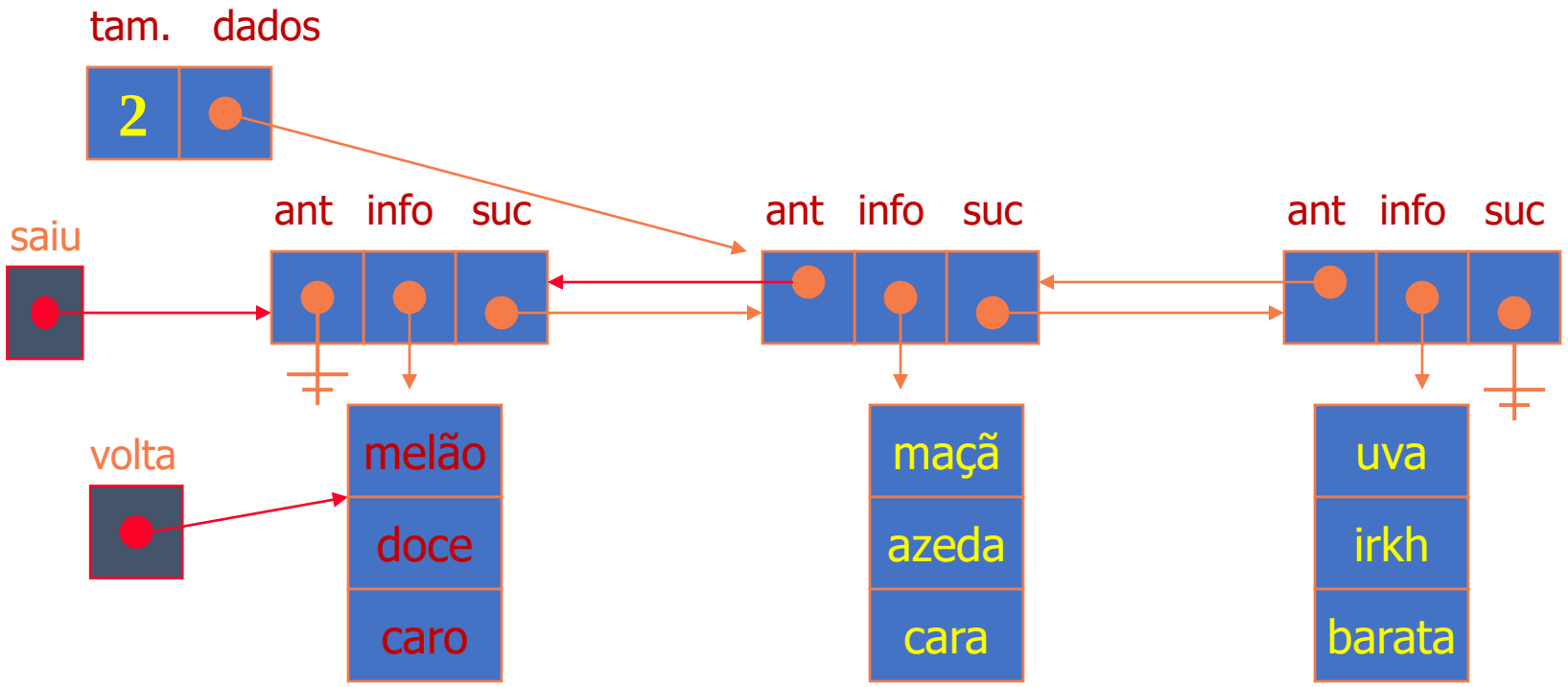
ALGORITMO RETIRADOINÍCIODUPLO

- Procedimento:
  - Testamos se há elementos.
  - Decrementamos o tamanho.
  - Se o elemento possuir sucessor, o antecessor do sucessor será NULO.
  - Liberamos a memória do elemento.
  - Devolvemos a Informação.
- Parâmetros:
  - ListaDupla.

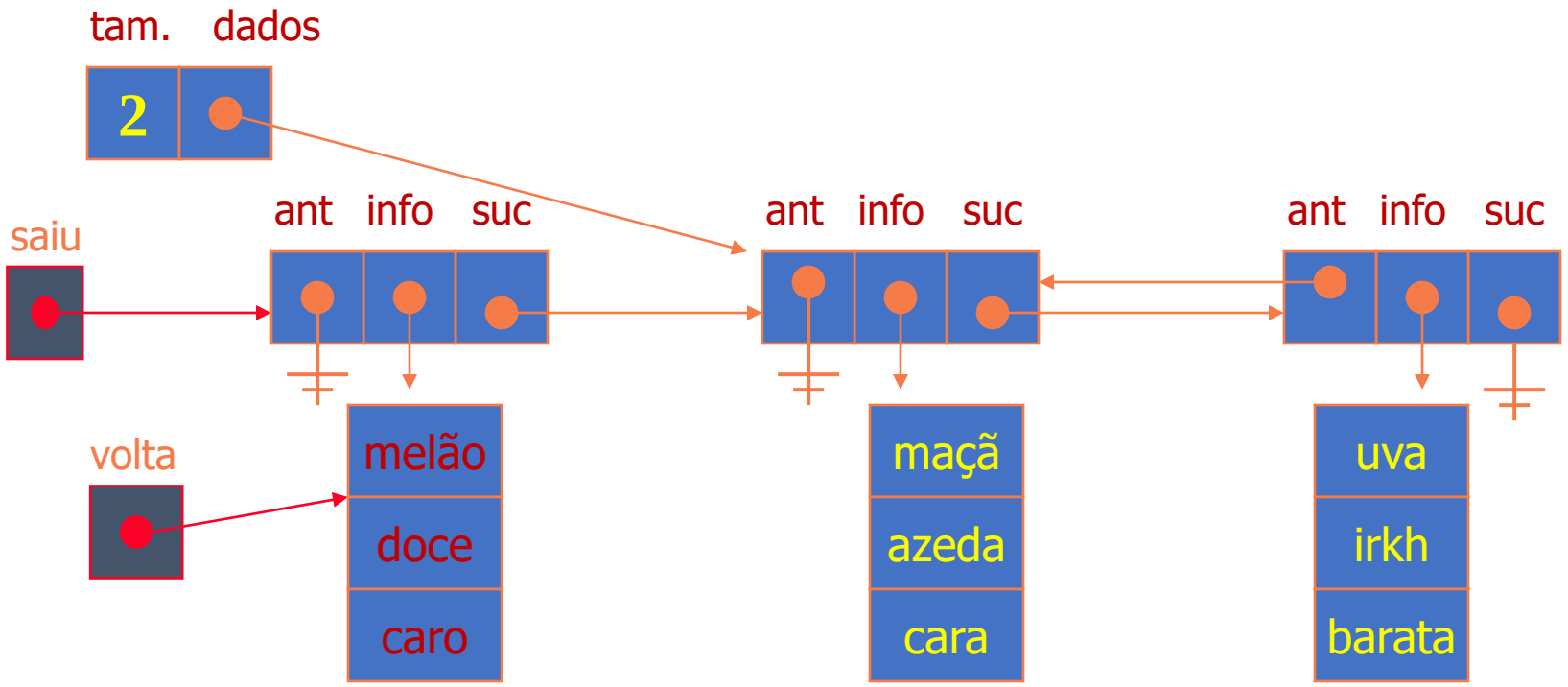
ALGORITMO RETIRADOINÍCIODUPLO



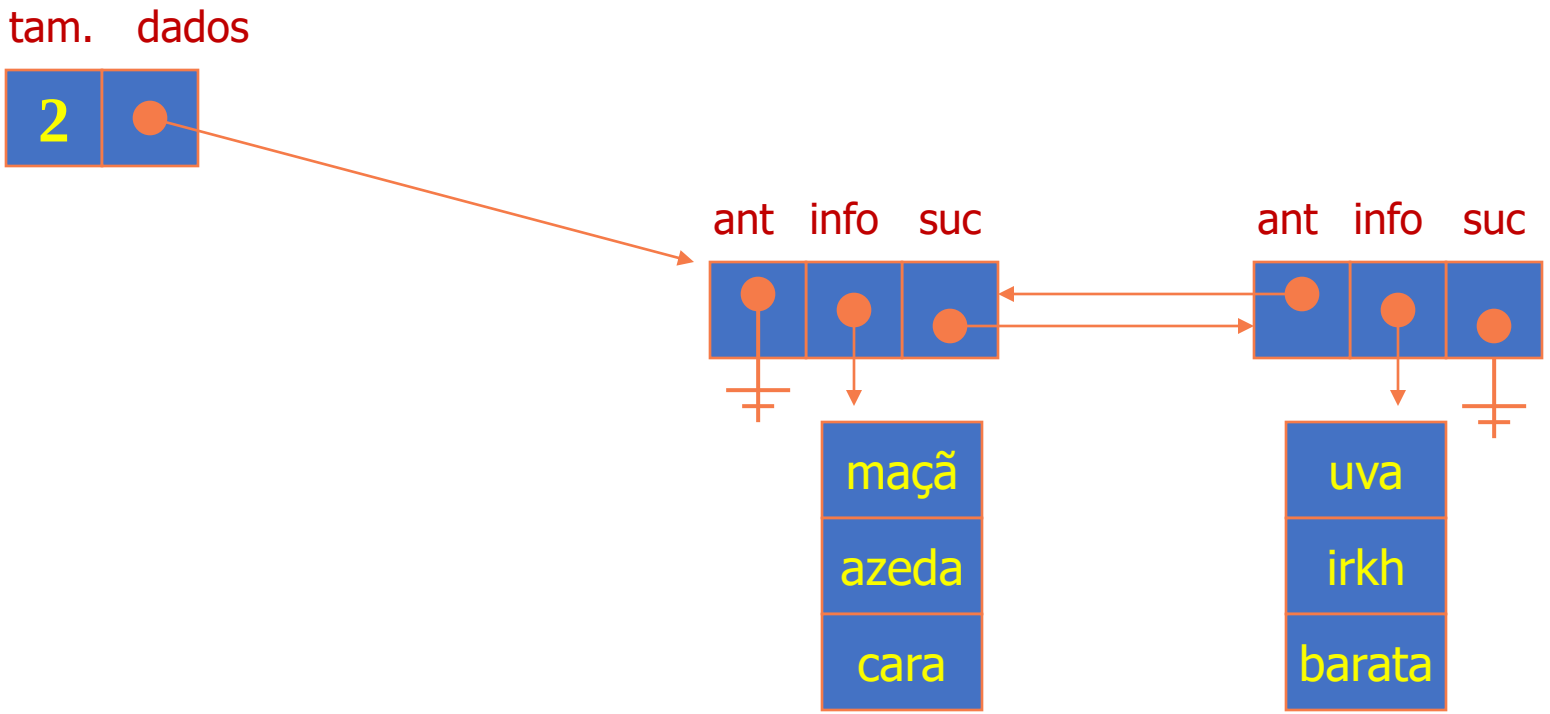
ALGORITMO RETIRADOINÍCIODUPLO



ALGORITMO RETIRADOINÍCIODUPLO



ALGORITMO RETIRADOINÍCIODUPLO



ALGORITMO RETIRADOINÍCIODUPLO

```
TipoInfo* FUNÇÃO retiraDoInício( ListaDupla *aLista )
   "Elimina o 1. Elemento de uma lista duplamente encadeada.
    Retorna a informação do elemento eliminado ou NULO."
   variáveis
        ElementoDuplo *saiu; "Var. auxiliar para o 1. elemento"
        TipoInfo  *volta; "Var. auxiliar para o dado retornado"
   início
        SE (listaVaziaDuplo(aLista)) ENTÃO
          RETORNE( NULO );
        SENÃO
          saiu <- aLista->dados;
          volta <- saiu->info;
          aLista->dados <- saiu->suc;
          SE (aLista->dados ~= NULO) ENTÃO
                aLista->dados->ant <- NULO;
          FIM SE
          aLista->tamanho <- aLista->tamanho - 1;
          LIBERE(saiu);
          RETORNE(volta);
        FIM SE
   fim;
```

ALGORITMO ADICIONANAPOSIÇÃODUPLO

- Praticamente idêntico à lista encadeada.
- Procedimento:
  - Testamos se a posição existe e se é possível alocar elemento.
  - Caminhamos até a posição.
  - Adicionamos o novo dado na posição.
  - Incrementamos o tamanho.
- Parâmetros:
  - O dado a ser inserido.
  - A posição onde inserir.
  - Lista.

ALGORITMO ADICIONANAPOSIÇÃO

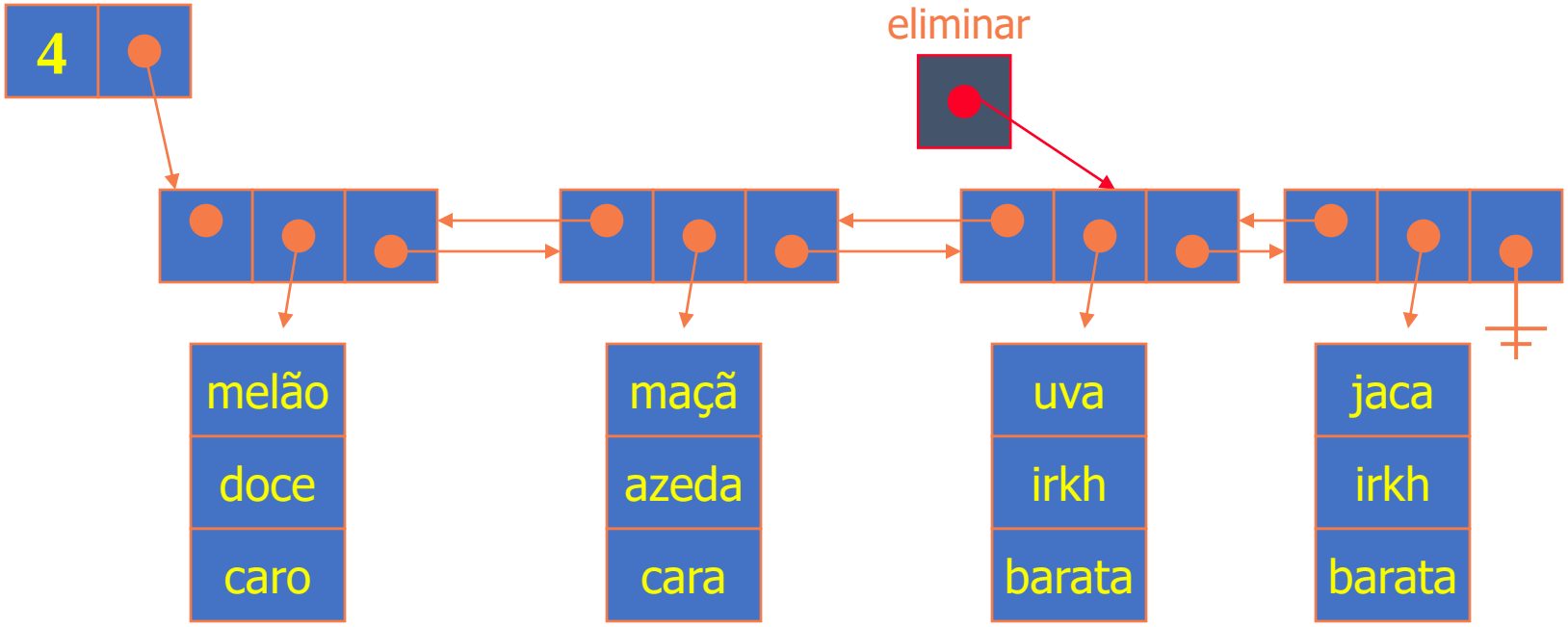
```
Inteiro FUNÇÃO adicionaNaPosiçãoDuplo( ListaDupla *aLista, TipoInfo *info,
                                       inteiro posição )
  "Adiciona novo elemento na posição dada.
   Retorna a novo numero de elementos da lista ou erro."
  variáveis
    ElementoDuplo *novo, *anterior;  "Ponteiros auxiliares"
  início
    SE (posição > aLista->tamanho + 1) ENTÃO
       RETORNE( ErroPosição )
    SENÃO
       SE (posição = 1) ENTÃO
         RETORNE(adicionaNoInícioDuplo(aLista, info);
       SENÃO
         novo <- aloque(Elemento);
         SE (novo = NULO) ENTÃO
           RETORNE( ErroListaCheia );
         SENÃO
           anterior <- aLista->dados;
           REPITA (posição - 2) VEZES
                anterior <- anterior->suc;
           novo->suc <- anterior->suc;
           SE (novo->suc ~= NULO ENTÃO "SE o novo não é o último da lista"
               novo->suc->ant <- novo; "Seto o antecessor do sucessor do novo"
           FIM SE
           novo->info <- info;
           anterior->suc <- novo;
           novo->ant <- anterior;
           aLista->tamanho <- aLista->tamanho + 1;
           RETORNE(aLista->tamanho);
         FIM SE
      FIM SE
    FIM SE
  fim;
```

ALGORITMO RETIRADAPOSIÇÃODUPLO

- Mais simples que a Lista Encadeada.
- Procedimento:
  - Testamos se a posição existe.
  - Caminhamos até a posição.
  - Retiramos o dado da posição.
  - Decrementamos o tamanho.
- Parâmetros:
  - A posição de onde retirar.
  - Lista.

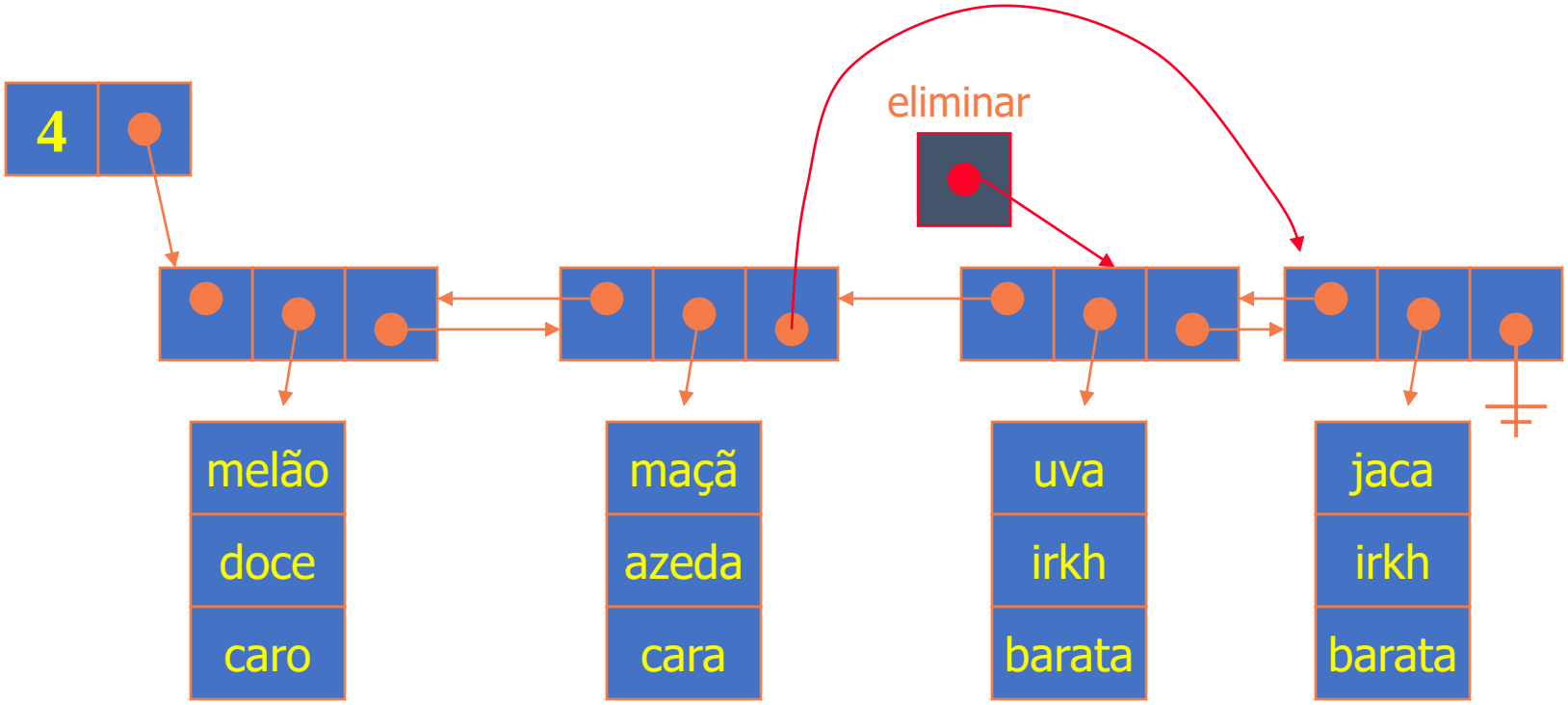
ALGORITMO RETIRADAPOSIÇÃODUPLO

Posições > 1



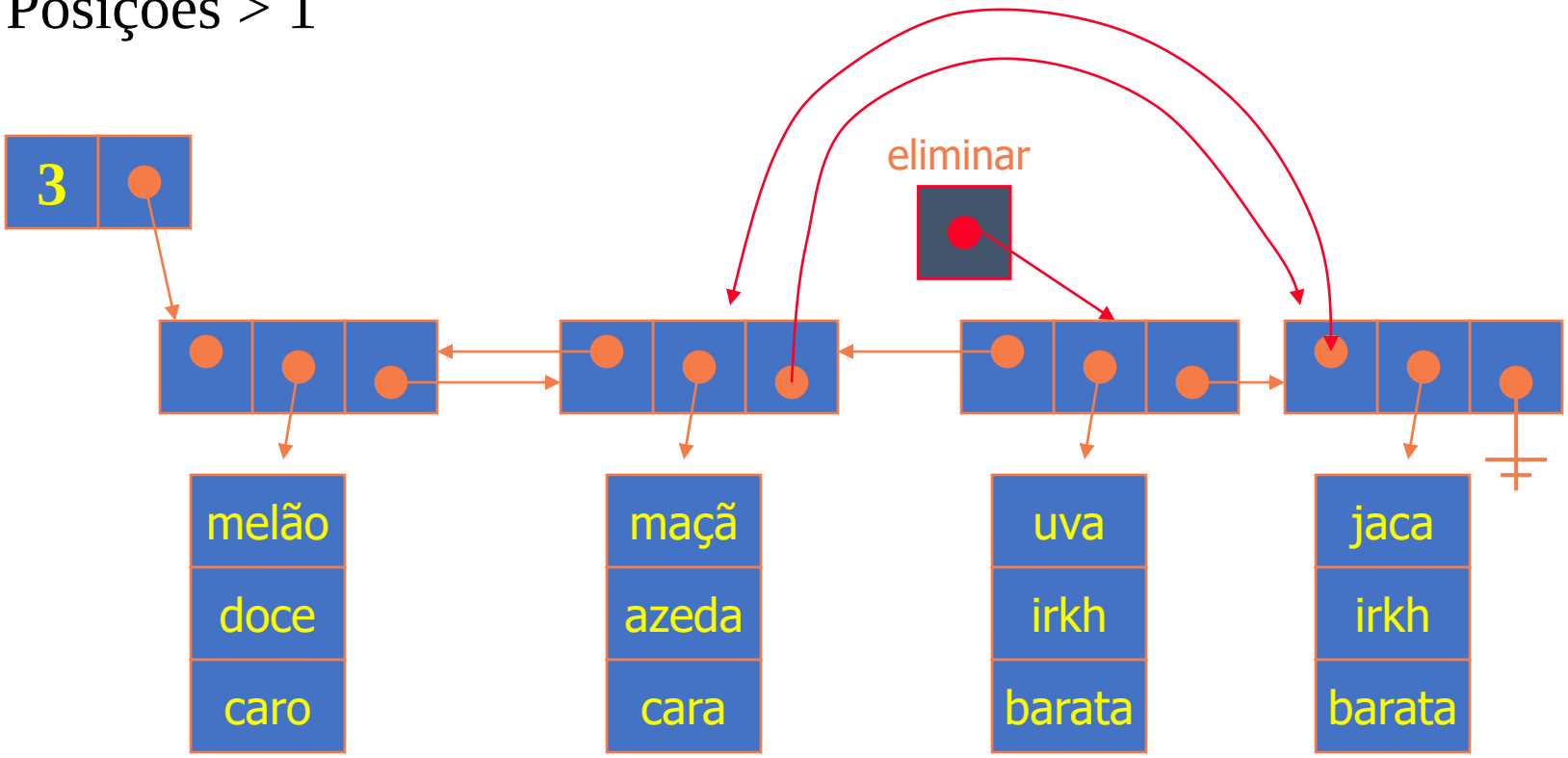
ALGORITMO RETIRADAPOSIÇÃO

Posições > 1



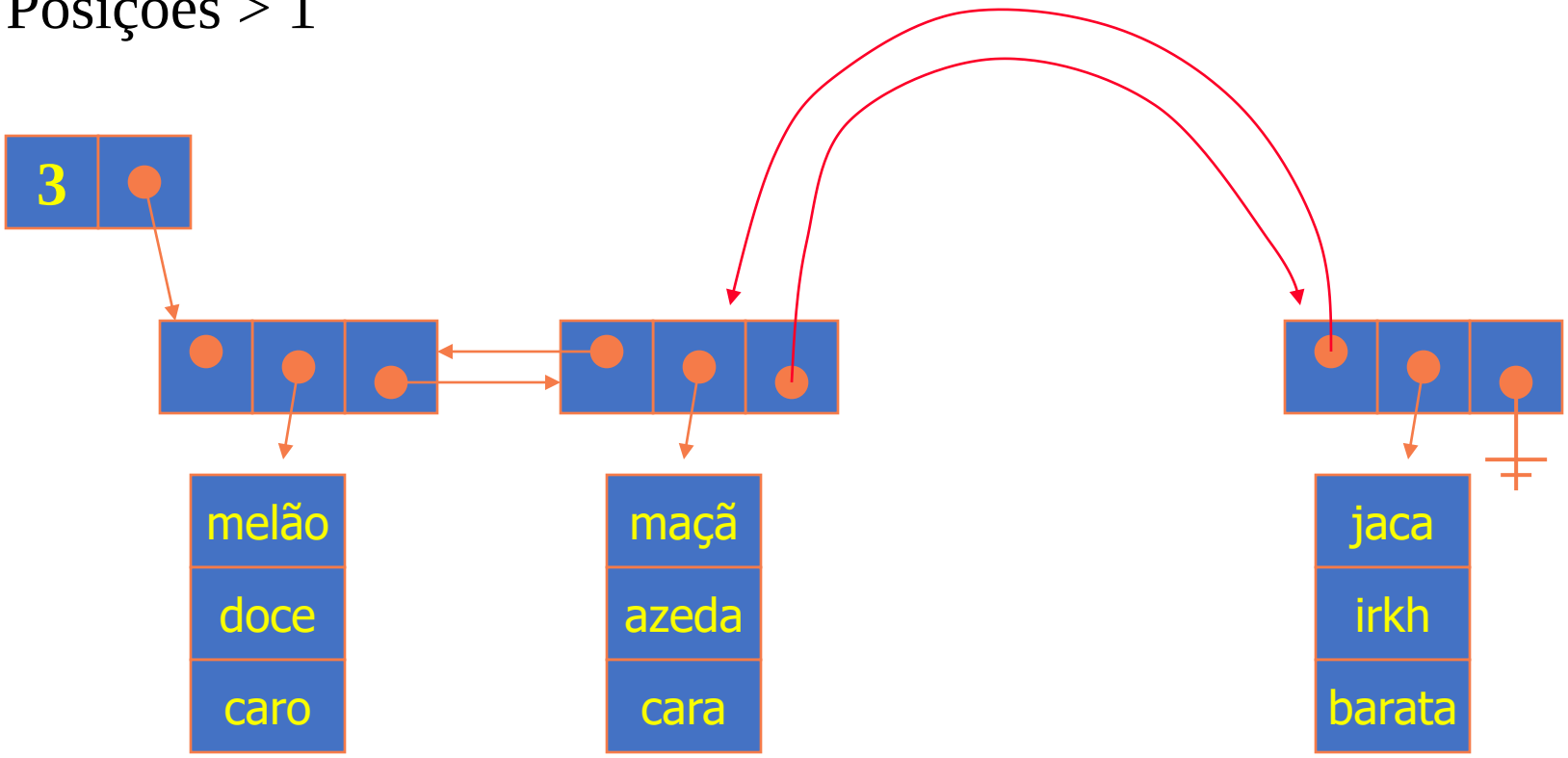
ALGORITMO RETIRADAPOSIÇÃO

Posições > 1



ALGORITMO RETIRADAPOSIÇÃO

Posições > 1



ALGORITMO RETIRADAPOSIÇÃO

```
TipoInfo* FUNÇÃO retiraDaPosição( Lista *aLista, inteiro posição )
   "Elimina o Elemento na posição posição de uma lista.
    Retorna a informação do elemento eliminado ou NULO."
   variáveis
        Elemento *anterior, *eliminar;  "Var. auxiliar para elemento"
        TipoInfo *volta;          "Var. auxiliar para o dado retornado"
   início
     SE (posição > aLista->tamanho) ENTÃO
        RETORNE( NULO );
     SENÃO
        SE (posição = 1) ENTÃO
          RETORNE( retiraDoInício(aLista) );
        SENÃO
          anterior <- aLista->dados;
          REPITA (posição - 2) VEZES
                anterior <- anterior->proximo;
          eliminar <- anterior->proximo;
          volta <- eliminar->info;
          anterior->proximo <- eliminar->proximo;
          aLista->tamanho <- aLista->tamanho - 1;
          LIBERE(eliminar);
          RETORNE(volta);
        FIM SE
     FIM SE
   fim;
```

ALGORITMO ADICIONAEMORDEMDUPLO

- Idêntico à lista encadeada.
- Procedimento:
  - Necessitamos de uma função para comparar os dados (maior)
  - Procuramos pela posição onde inserir comparando dados.
  - Chamamos **`adicionaNaPosiçãoDuplo`**.
- Parâmetros:
  - O dado a ser inserido.
  - ListaDupla.

ALGORITMO ADICIONAEMORDEMDUPLO

```
Inteiro FUNÇÃO adicionaEmOrdem(ListaDupla *aLista, TipoInfo dado)
   variáveis
     ElementoDuplo *atual;     "Variável auxiliar para caminhar"
     inteiro       posição;
   início
     SE (listaVaziaDupla(aLista)) ENTÃO
       RETORNE(adicionaNoInícioDuplo(aLista, dado));
     SENÃO
       atual <- aLista->dados;
       posição <- 1;
       ENQUANTO (atual->sucessor ~= NULO E
                 maior(dado, atual->info)) FAÇA
          "Encontrar posição para inserir"
          atual <- atual->sucessor;
          posição <- posição + 1;
        FIM ENQUANTO
        SE maior(dado, atual->info) ENTÃO "Parou porque acabou lista"
          RETORNE(adicionaNaPosiçãoDuplo(aLista,dado,posição + 1));
        SENÃO
          RETORNE(adicionaNaPosiçãoDuplo(aLista, dado, posição));
        FIM SE
     FIM SE
   fim;
```

ALGORITMOS RESTANTES: LISTA DUPLAMENTE ENCADEADA

- Operações de inclusão e exclusão:
  - AdicionaDuplo(listaDupla, dado)
  - RetiraDuplo(listaDupla)
  - RetiraEspecíficoDuplo(listaDupla, dado)
- Operações: Inicializar ou limpar:
  - DestroiListaDupla(listaDupla

ALGORITMO DESTROILISTADUPLA

```
FUNÇÃO destroiListaDupla(ListaDupla *aLista)
   variáveis
     ElementoDuplo *atual, *anterior; "Variável auxiliar para caminhar"
   início
     SE (listaVaziaDupla(aLista)) ENTÃO
        LIBERE(aLista);
     SENÃO
        atual <- aLista->dados;
        ENQUANTO (atual ~= NULO) FAÇA
          "Eliminar até o fim"
          anterior <- atual;
           "Vou para o próximo mesmo que seja nulo."
          atual <- atual->sucessor;
          "Liberar primeiro a Info"
          LIBERE(anterior->info);
          "Liberar o elemento que acabei de visitar."
          LIBERE(anterior);
        FIM ENQUANTO
        LIBERE(aLista);
     FIM SE
   fim;
```