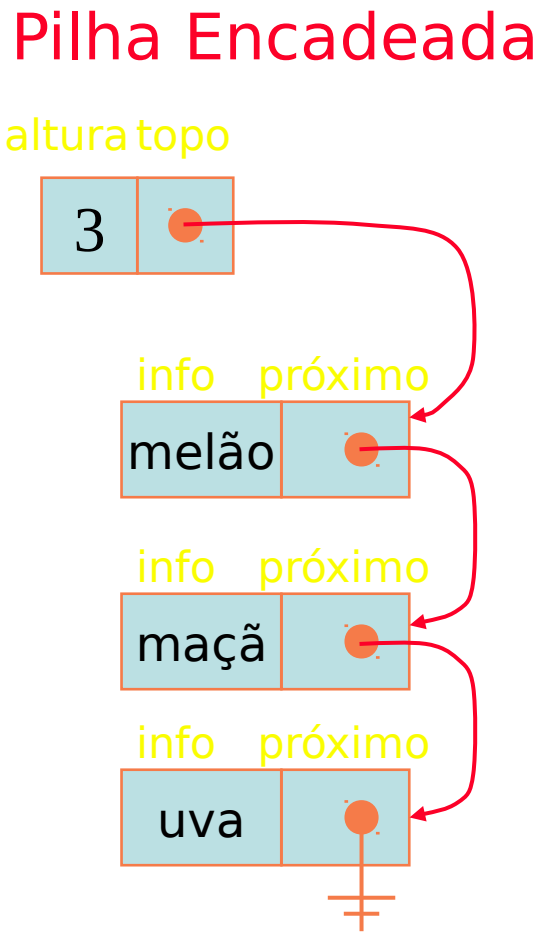
Estruturas de Dados

Listas Encadeadas

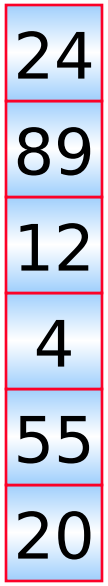
- Pilhas encadeadas
- Filas encadeadas
- Listas duplamente encadeadas

Extensões do conceito de Lista Encadeada

- A idéia da Lista Encadeada vista até agora é o modelo mais geral e simples;
- pode ser especializada e extendida das mais variadas formas:
  - Especializada:
    - **Pilhas encadeadas**
    - **Filas**
  - Extendida:
    - **Listas Duplamente Encadeadas**
    - **Listas Circulares Simples e Duplas**

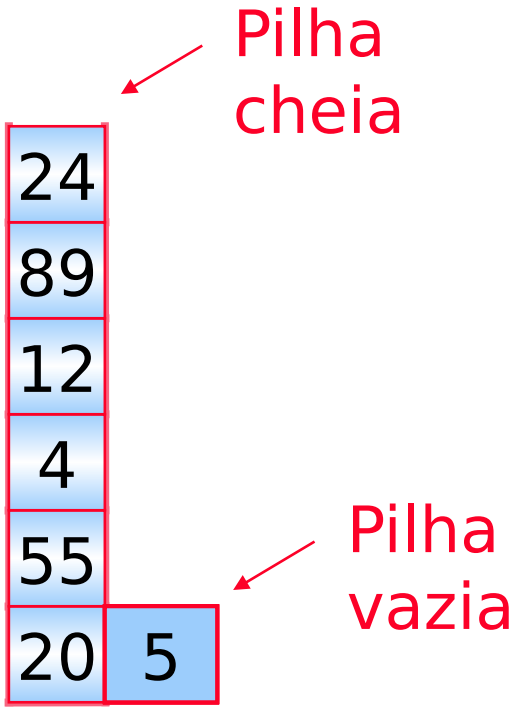


Pilhas



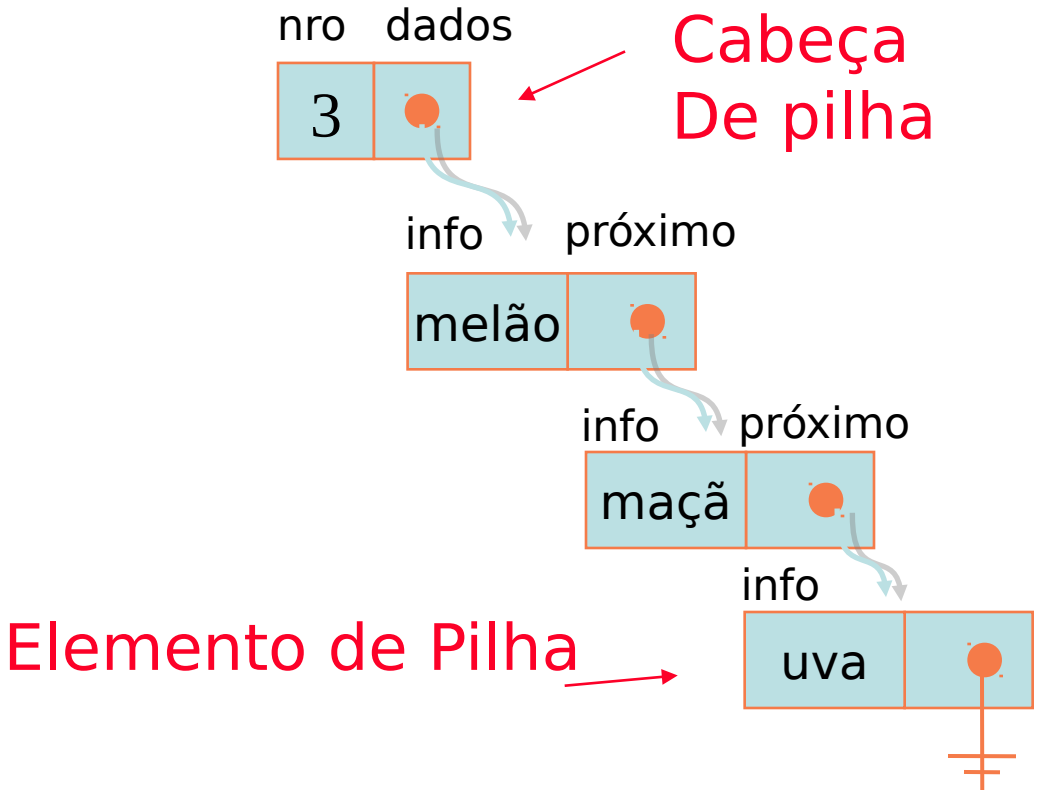
A Pilha é uma estrutura de dados cujo funcionamento é inspirado no de uma pilha “natural”.

Pilhas usando Vetores



- Vetores possuem um espaço limitado para armazenar dados;
- necessitamos definir um espaço grande o suficiente para a nossa pilha;

Pilhas Encadeadas



- A estruturas é limitada pela memória disponível;
- Não é necessário definir um valor fixo para o tamanho da Pilha;

Modelagem: Cabeça de Pilha

- Necessitamos:
  - um ponteiro para o primeiro elemento da pilha;
  - um inteiro para indicar quantos elementos a pilha possui.

- Pseudo-código:

```
classe tPilha {
   tElemento *dados;
   inteiro tamanho;
};
```

Modelagem: Elemento de Pilha

- Necessitamos:
  - um ponteiro para o próximo elemento da pilha;
  - um campo do tipo da informação que vamos armazenar.

- Pseudo-código:

```
class tElemento {
   tElemento *próximo;
   T* info;
   };
```

Modelagem da Pilha

- Aspecto Funcional:
  - colocar e retirar dados da pilha;
  - testar se a pilha está vazia;
- Colocar e retirar dados da pilha:
  - Empilha(dado)
  - Desempilha()
- Testar se a pilha está vazia:
  - PilhaVazia
- Inicializar ou limpar:
  - criaPilha

Algoritmo criaPilha

```
MÉTODO criaPilha()
//Inicializa a cabeça e o tamanho da pilha
início
 dados <- NULO;
 tamanho <- 0;
fim;
```

Algoritmo PilhaVazia

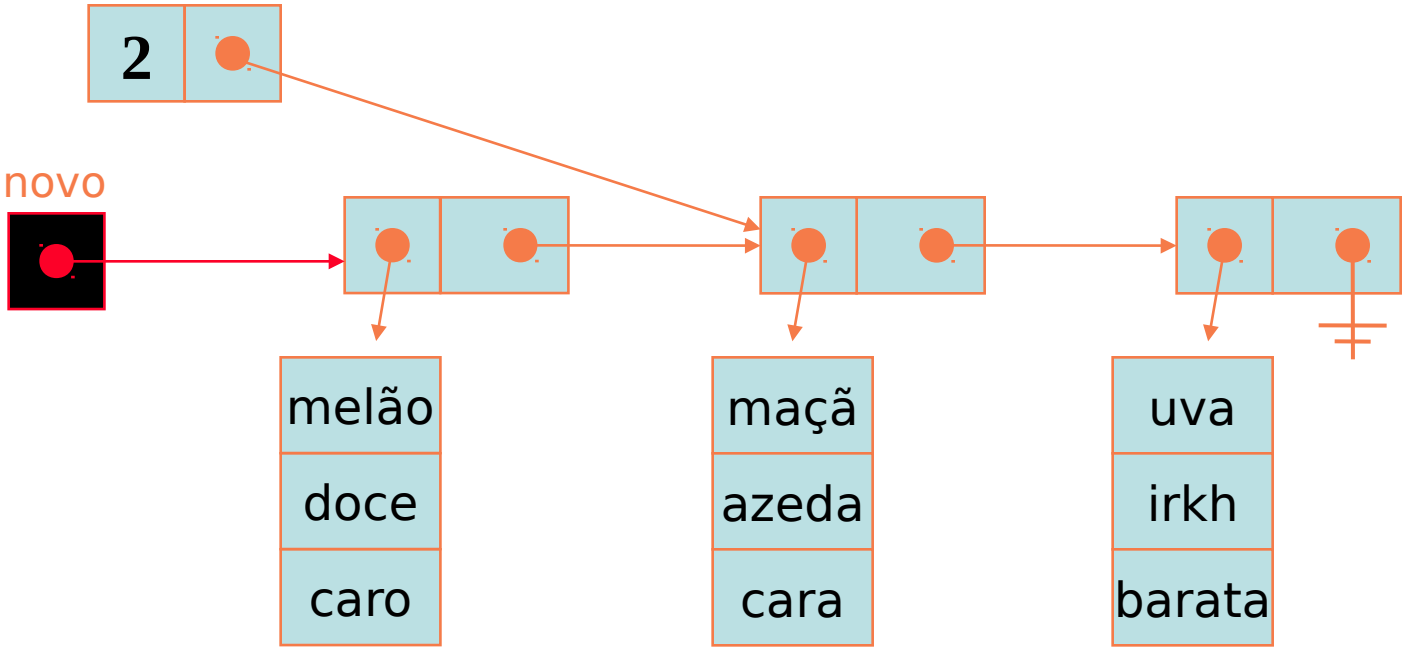
```
Booleano MÉTODO pilhaVazia()
início
 SE (tamanho = 0) ENTÃO
  RETORNE(Verdadeiro)
 SENÃO
  RETORNE(Falso);
fim;
```

- Um algoritmo PilhaCheia não existe aqui;
- Verificar se ha espaço na memória para um novo elemento será responsabilidade de cada operação de adição.

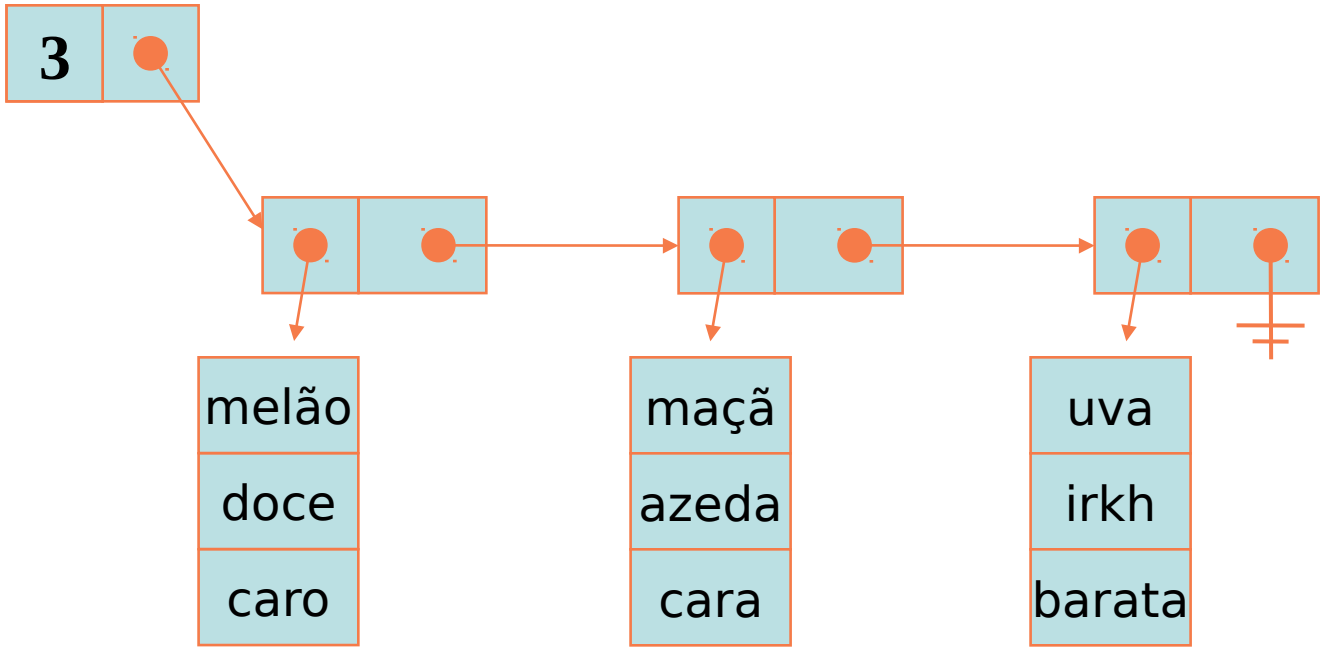
Algoritmo Empilha

- Procedimento:
  - Alocamos um elemento;
  - fazemos o próximo deste novo elemento ser o primeiro da Pilha;
  - fazemos a cabeça de Pilha apontar para o novo elemento.
- Parâmetros:
  - O tipo info (dado) a ser inserido;
- Semelhanças????

Algoritmo Empilha = AdicionaNoInício



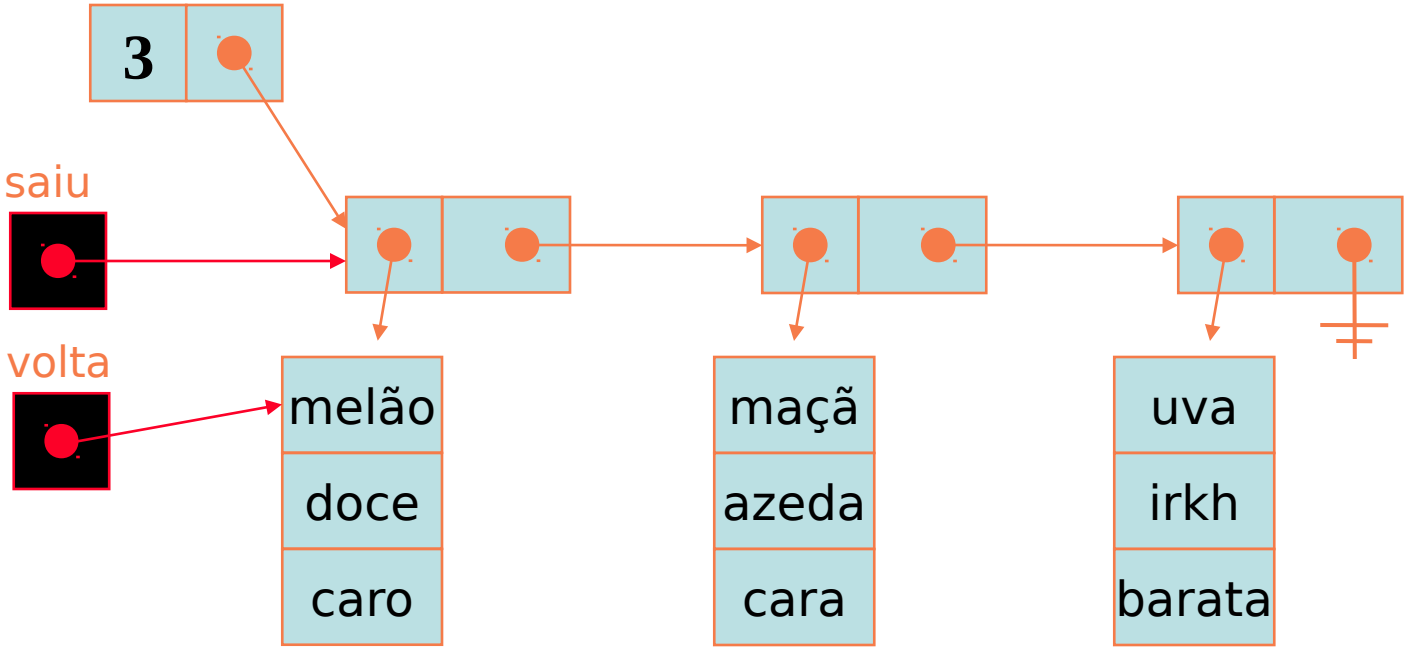
Algoritmo Empilha



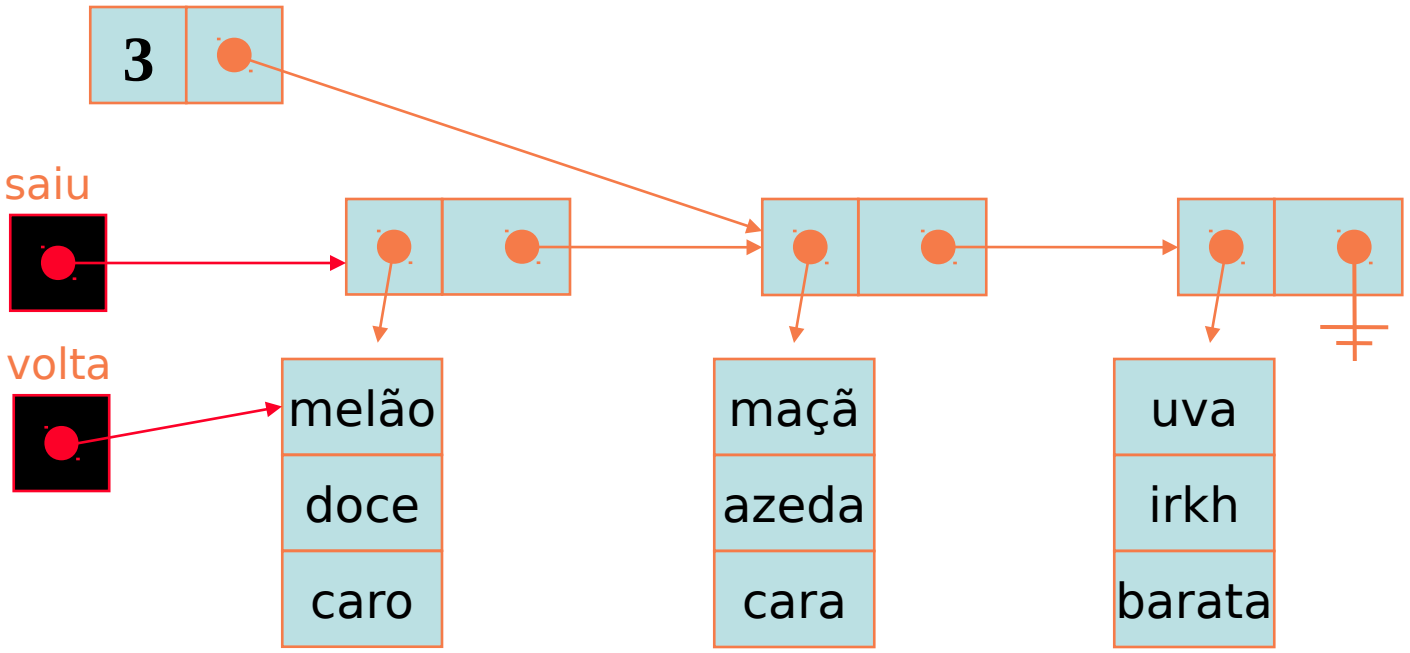
Algoritmo Desempilha

- Procedimento:
  - testamos se há elementos;
  - decrementamos o tamanho;
  - liberamos a memória do elemento;
  - devolvemos a informação.
- Semelhanças??

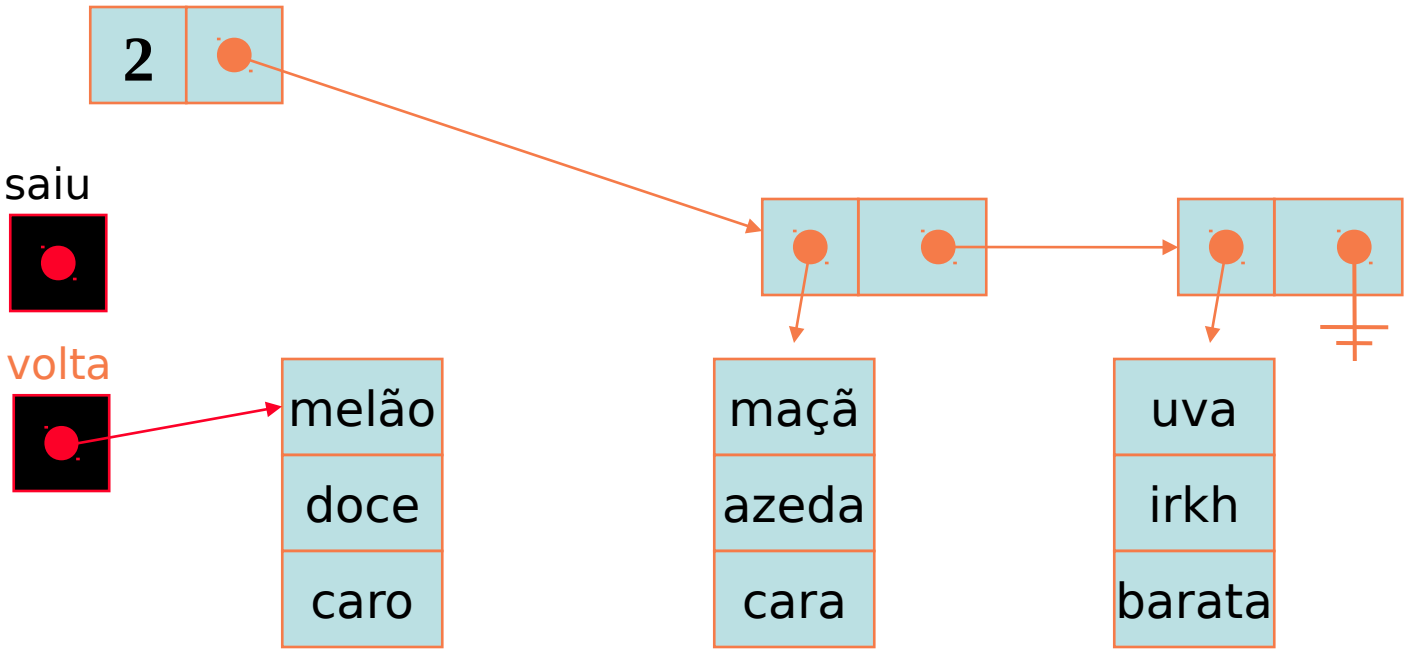
Algoritmo Desempilha = RetiraDoInício



Algoritmo Desempilha



Algoritmo Desempilha



Algoritmo Desempilha

```
T* MÉTODO desempilha()
   //Elimina o primeiro elemento de uma pilha.
   //Retorna a informação do elemento eliminado ou NULO.
   variáveis
        tElemento *saiu; //Variável auxiliar para o primeiro elemento.
        T *volta; //Variável auxiliar para o dado retornado.
   início
        SE (Vazia()) ENTÃO
          THROW(PILHAVAZIA);
        SENÃO
          saiu <- dados;
          volta <- saiu->info;
          dados <- saiu->próximo;
          tamanho <- tamanho - 1;
          LIBERE(saiu);
          RETORNE(volta);
        FIM SE
   fim;
```

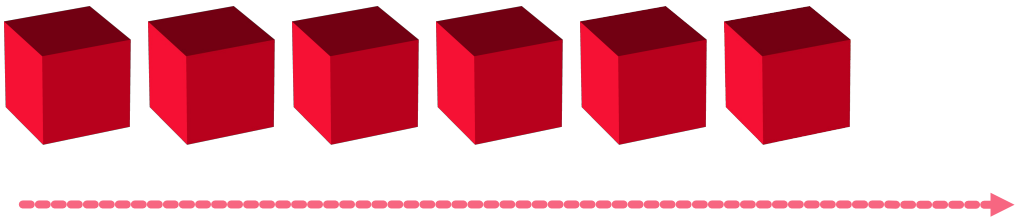
Exercício

- Implemente uma classe PilhaEncadeada
- Implemente a pilha usando Templates
- Use as melhores práticas de orientação a objetos
- Documente todas as classes, métodos e atributos.
- Aplique os testes unitários disponíveis no moodle da disciplina para validar sua estrutura de dados.

Filas

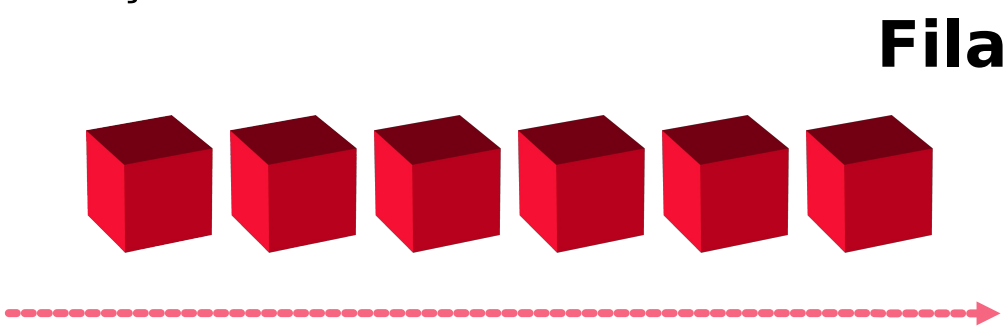
- A Fila é uma estrutura de dados que simula uma fila da vida real.
- Possui duas operações básicas:
  - incluir no fim da fila;
  - retirar do começo da fila;
  - chamada de Estrutura-FIFO:
    First-In, First-Out - O primeiro que entrou é o primeiro a sair…

Fila



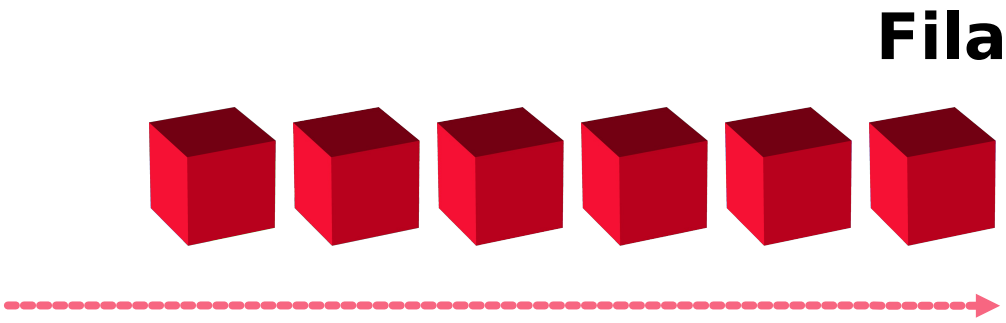
Filas

- É uma estrutura de dados importantíssima para:
  - gerência de dados/processos por ordem cronológica:
    - Fila de impressão em uma impressora de rede;
    - Fila de pedidos de uma expedição ou tele-entrega.
  - simulação de processos seqüenciais:
    - chão de fábrica: fila de camisetas a serem estampadas;
    - comércio: simulação de fluxo de um caixa de supermercado;
    - tráfego: simulação de um cruzamento com um semáforo.



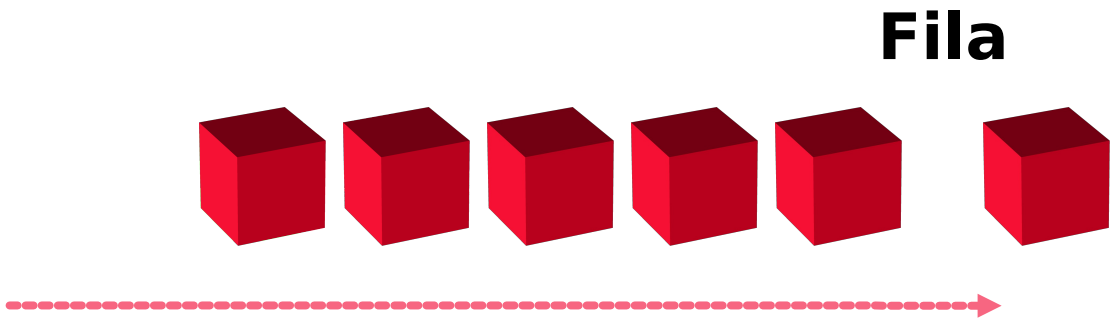
Filas

- É uma estrutura de dados importantíssima para:
  - gerência de dados/processos por ordem cronológica:
    - Fila de impressão em uma impressora de rede;
    - Fila de pedidos de uma expedição ou tele-entrega.
  - simulação de processos seqüenciais:
    - chão de fábrica: fila de camisetas a serem estampadas;
    - comércio: simulação de fluxo de um caixa de supermercado;
    - tráfego: simulação de um cruzamento com um semáforo.



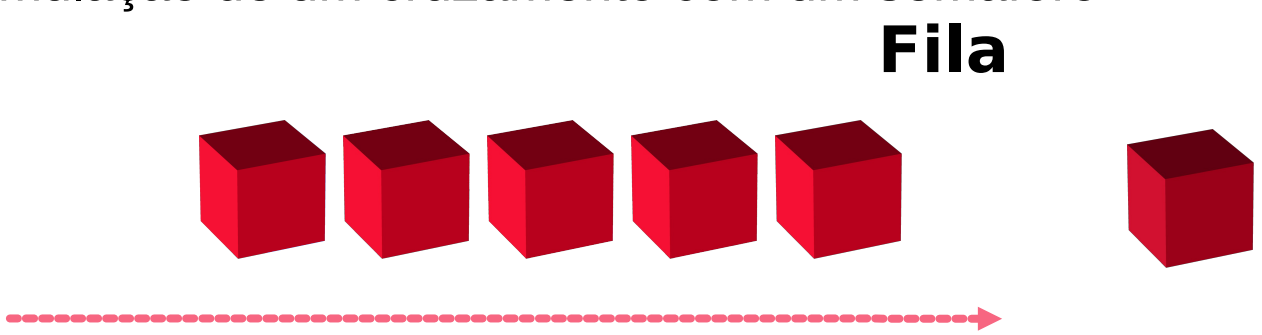
Filas

- É uma estrutura de dados importantíssima para:
  - gerência de dados/processos por ordem cronológica:
    - Fila de impressão em uma impressora de rede;
    - Fila de pedidos de uma expedição ou tele-entrega.
  - simulação de processos seqüenciais:
    - chão de fábrica: fila de camisetas a serem estampadas;
    - comércio: simulação de fluxo de um caixa de supermercado;
    - tráfego: simulação de um cruzamento com um semáforo.



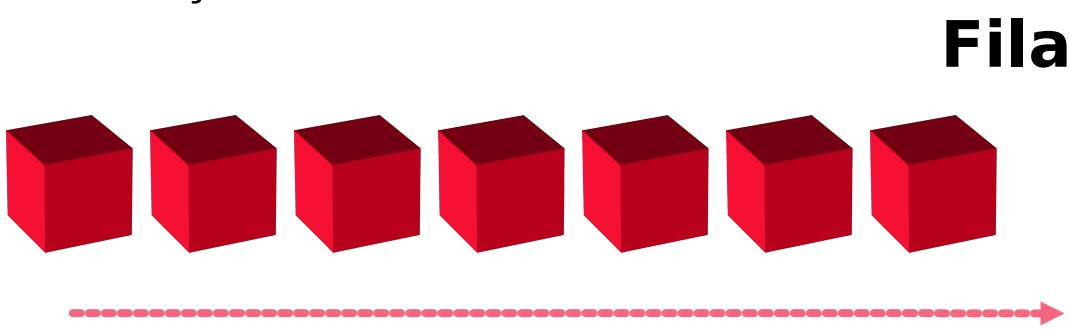
Filas

- É uma estrutura de dados importantíssima para:
  - gerência de dados/processos por ordem cronológica:
    - Fila de impressão em uma impressora de rede;
    - Fila de pedidos de uma expedição ou tele-entrega.
  - simulação de processos seqüenciais:
    - chão de fábrica: fila de camisetas a serem estampadas;
    - comércio: simulação de fluxo de um caixa de supermercado;
    - tráfego: simulação de um cruzamento com um semáforo.



Filas

- É uma estrutura de dados importantíssima para:
  - gerência de dados/processos por ordem cronológica:
    - Fila de impressão em uma impressora de rede;
    - Fila de pedidos de uma expedição ou tele-entrega.
  - simulação de processos seqüenciais:
    - chão de fábrica: fila de camisetas a serem estampadas;
    - comércio: simulação de fluxo de um caixa de supermercado;
    - tráfego: simulação de um cruzamento com um semáforo.

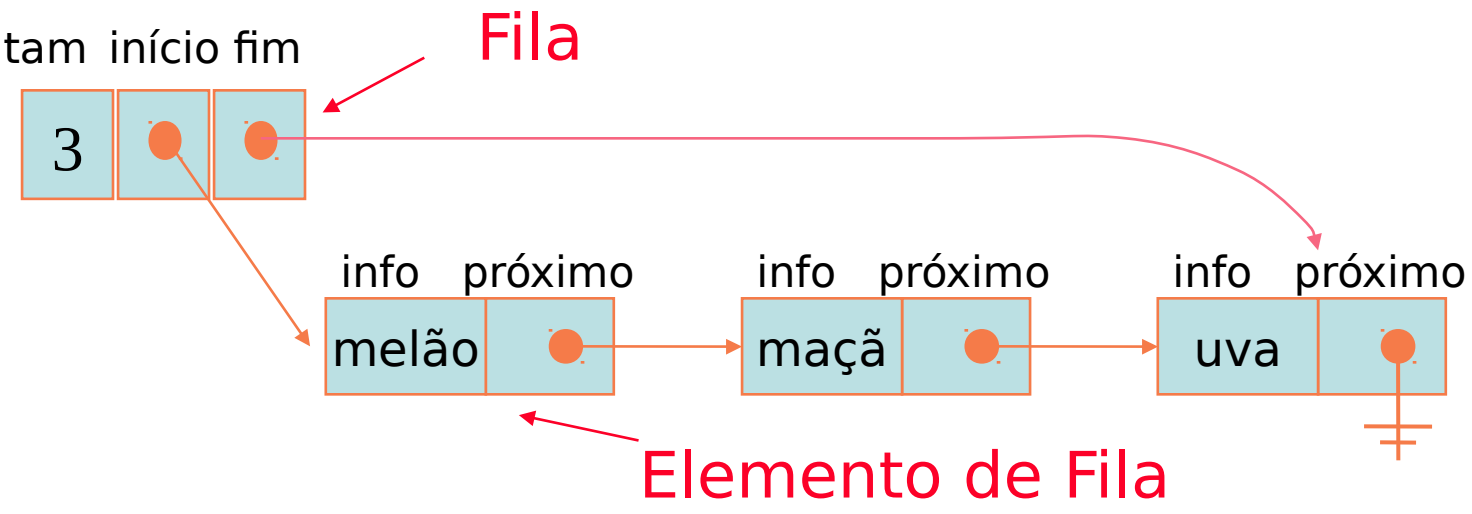


Filas: representação

- Extensão da lista encadeada:
  - referenciamos o último elemento também;
  - adicionamos no fim;
  - excluímos do início.

Pseudo-código:

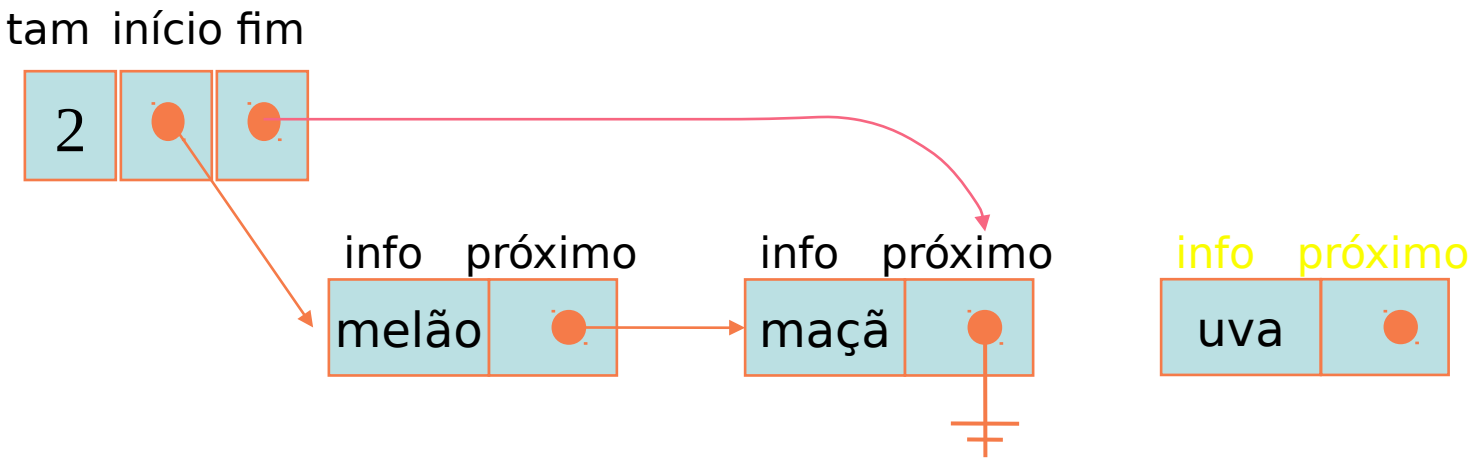
```
classe tFila {
  tElemento *início;
  tElemento *fim;
  inteiro tamanho;
};
```



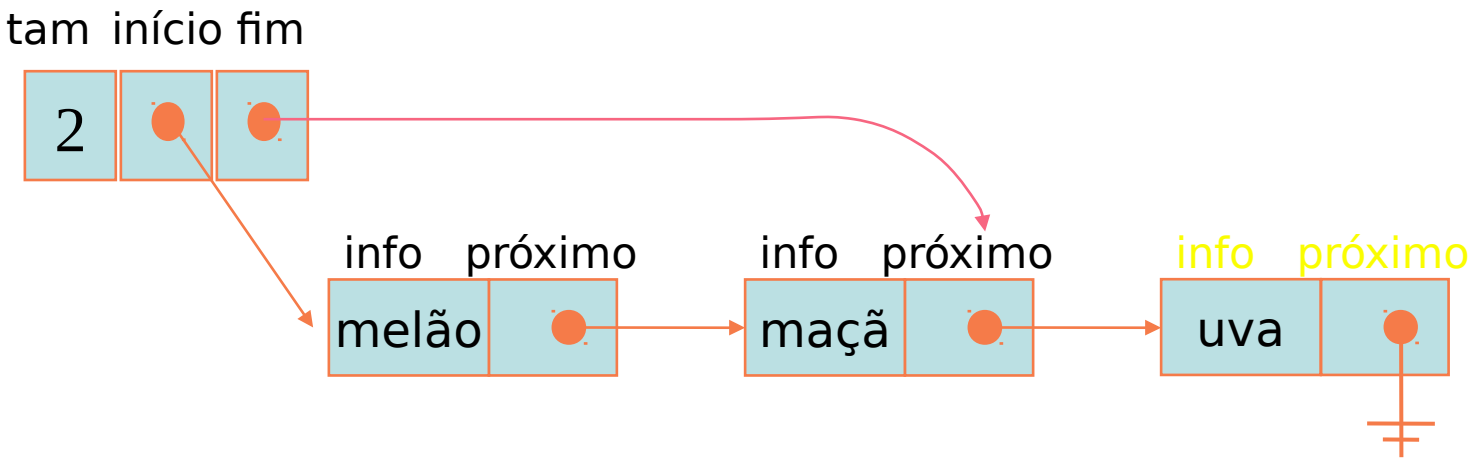
Algoritmo CriaFila

```
MÉTODO criaFila()
   //Inicializa a cabeça e o tamanho da fila
   início
        inicio <- NULO;
        fim <- NULO;
        tamanho <- 0;
        RETORNE(aFila);
   fim;
```

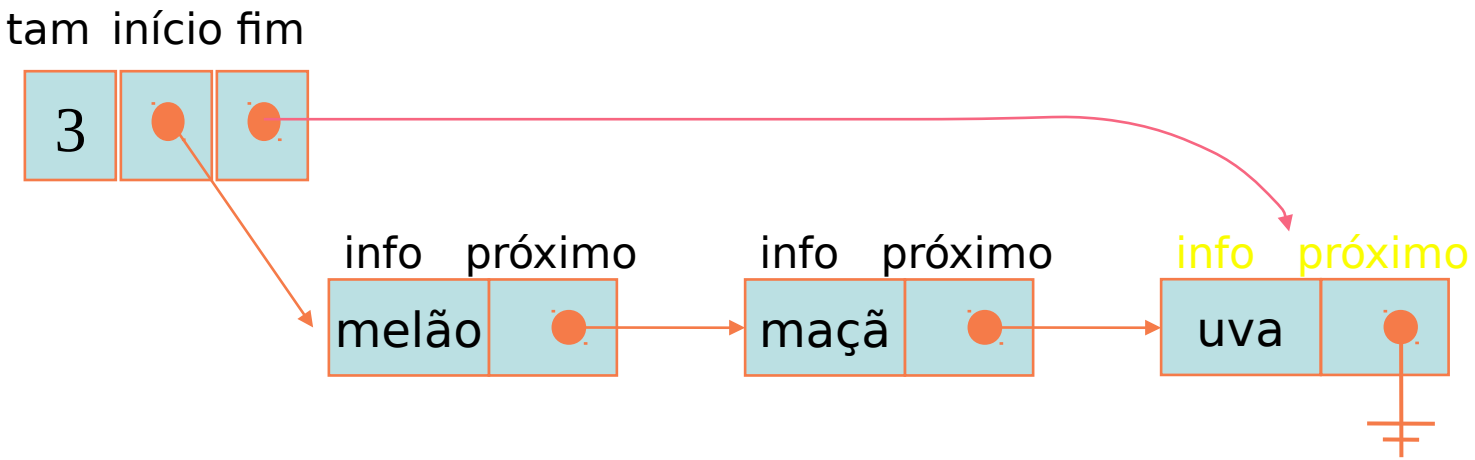
Algoritmo Adiciona (Fila)



Algoritmo Adiciona (Fila)

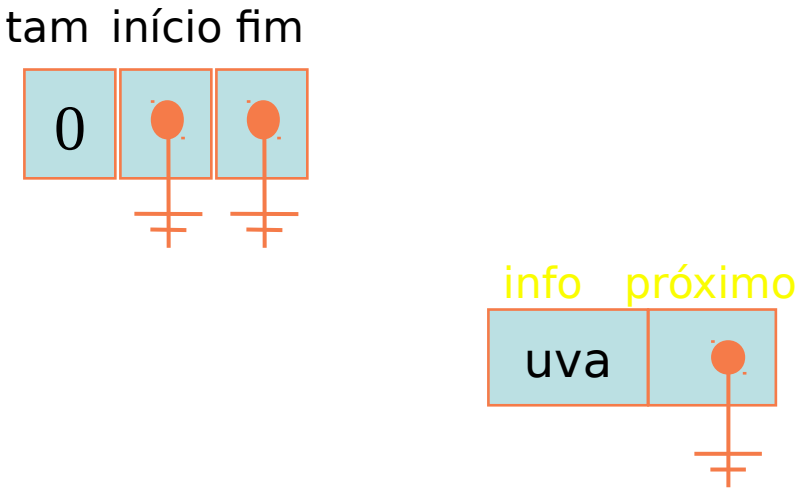


Algoritmo Adiciona (Fila)



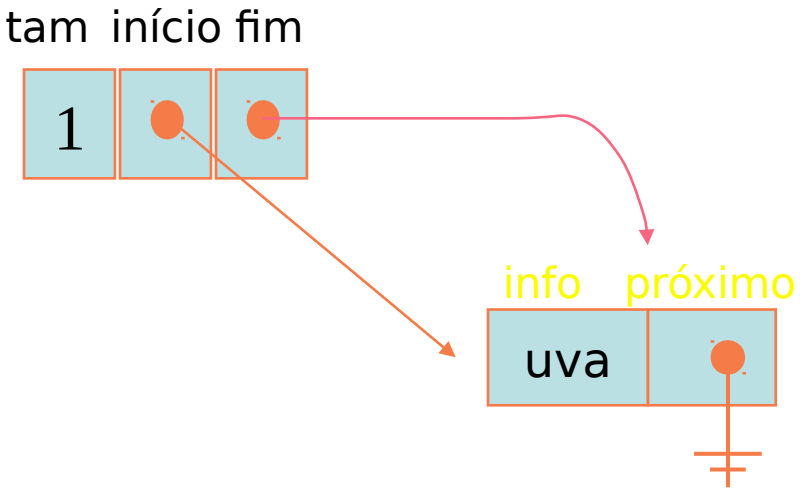
Algoritmo Adiciona (Fila)

Caso especial: Fila Vazia



Algoritmo Adiciona (Fila)

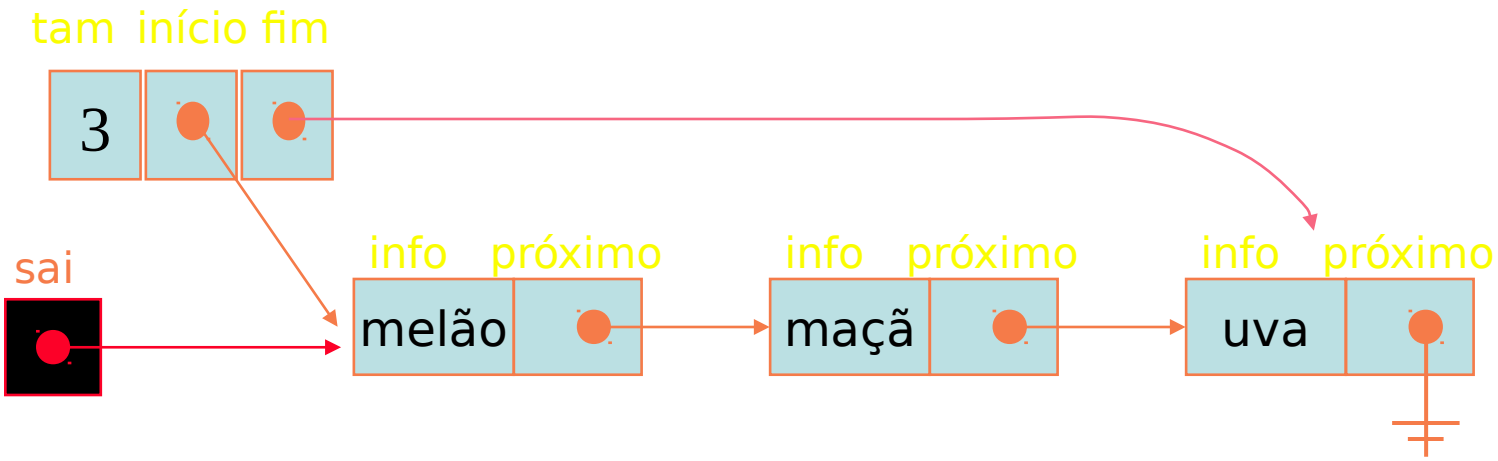
Caso especial: Fila Vazia



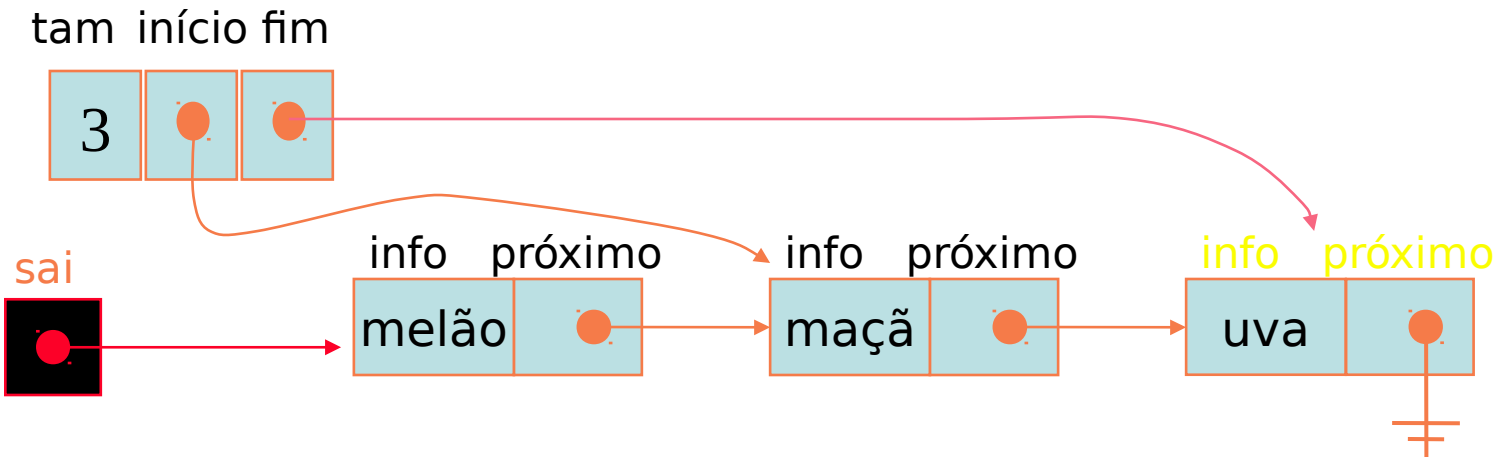
Algoritmo Adiciona (Fila)

```
inteiro MÉTODO adiciona(T *dado)
  variáveis
       tElemento *novo; //Variável auxiliar para o novo elemento.
  início
       novo <- aloque(tElemento);
        SE ( novo == NULO)
           THROW FILACHEIA;
       SE filaVazia() ENTÃO
           início <- novo
       SENÃO
           fim->próximo <- novo;
       FIM SE
           novo->próximo <- NULO;
           novo->info <- dado;
           fim <- novo;
           tamanho <- tamanho + 1;
         RETORNE(tamanho);
       FIM SE
  fim;
```

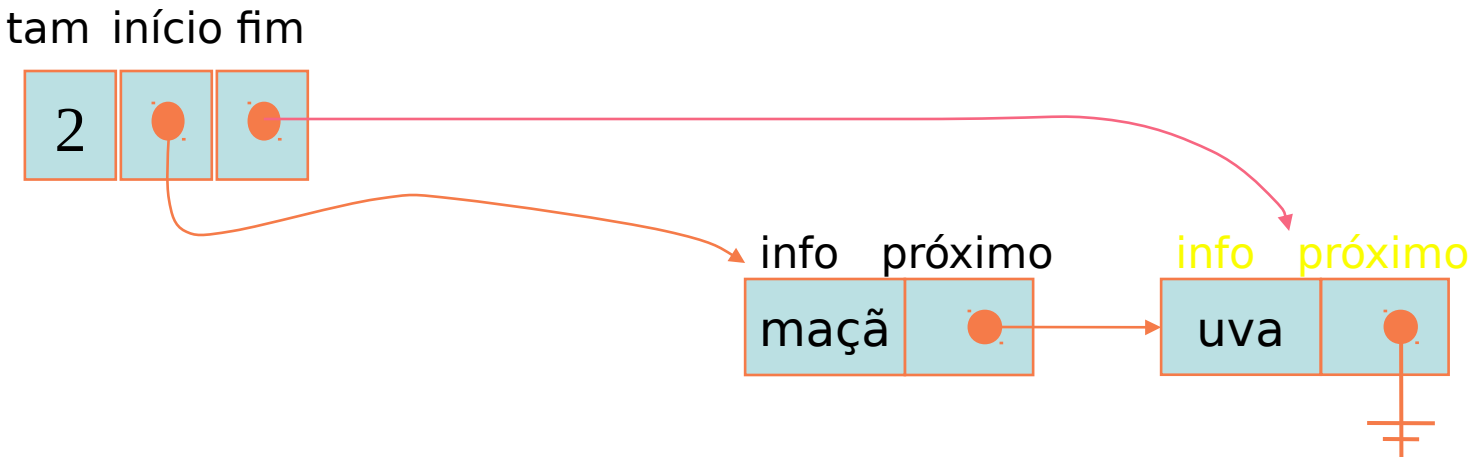
Algoritmo Retira (Fila)



Algoritmo Retira (Fila)

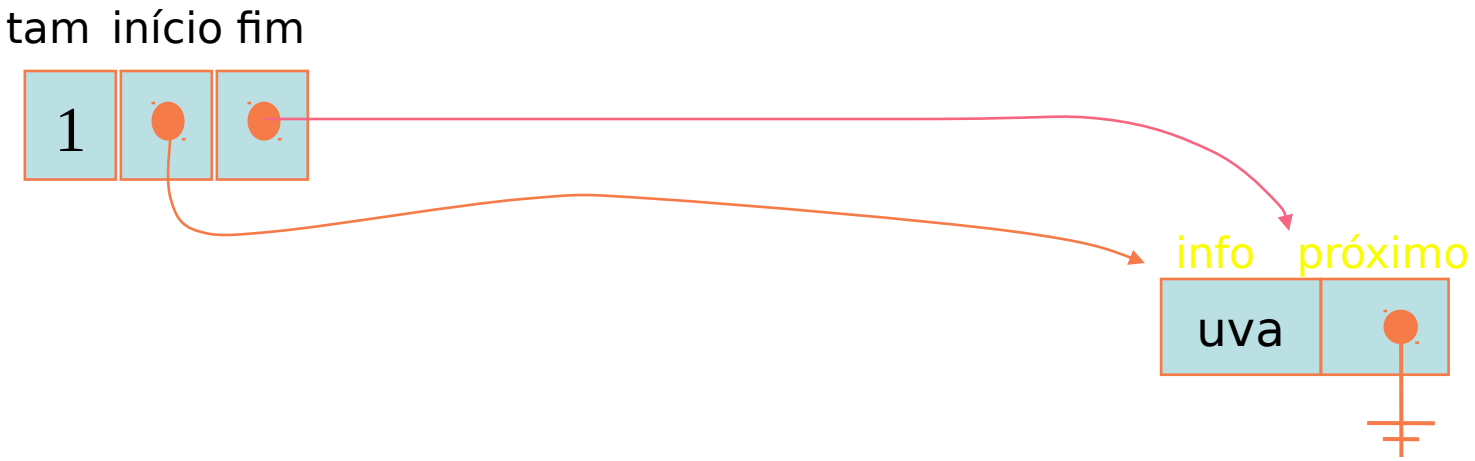


Algoritmo Retira (Fila)



Algoritmo Retira (Fila)

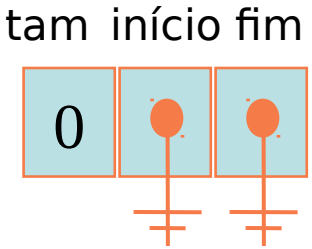
Caso especial: Fila Unitária



Não preciso de uma variável auxiliar sai.

Algoritmo Retira (Fila)

Caso especial: Fila Unitária



Algoritmo RetiraDoInício (Fila)

```
T* MÉTODO retiraDoInício()
   //Elimina o primeiro elemento de uma fila.
   //Retorna a informação do elemento eliminado ou NULO.
   variáveis
         tElemento *saiu; //Variável auxiliar para o primeiro elemento.
         T *volta; //Variável auxiliar para o dado retornado.
   início
         SE (filaVazia()) ENTÃO
           THROW FILAVAZIA;
         SENÃO
           saiu <- início;
           volta <- saiu->info;
           início <- saiu->próximo;
           //Se SAIU for o único, próximo é NULO e está certo.
           SE (tamanho = 1) ENTÃO
                  //Fila unitária: devo anular o fim também.
                  fim <- NULO;
           FIM SE
           tamanho <- tamanho - 1;
           LIBERE(saiu);
           RETORNE(volta);
         FIM SE
   fim;
```

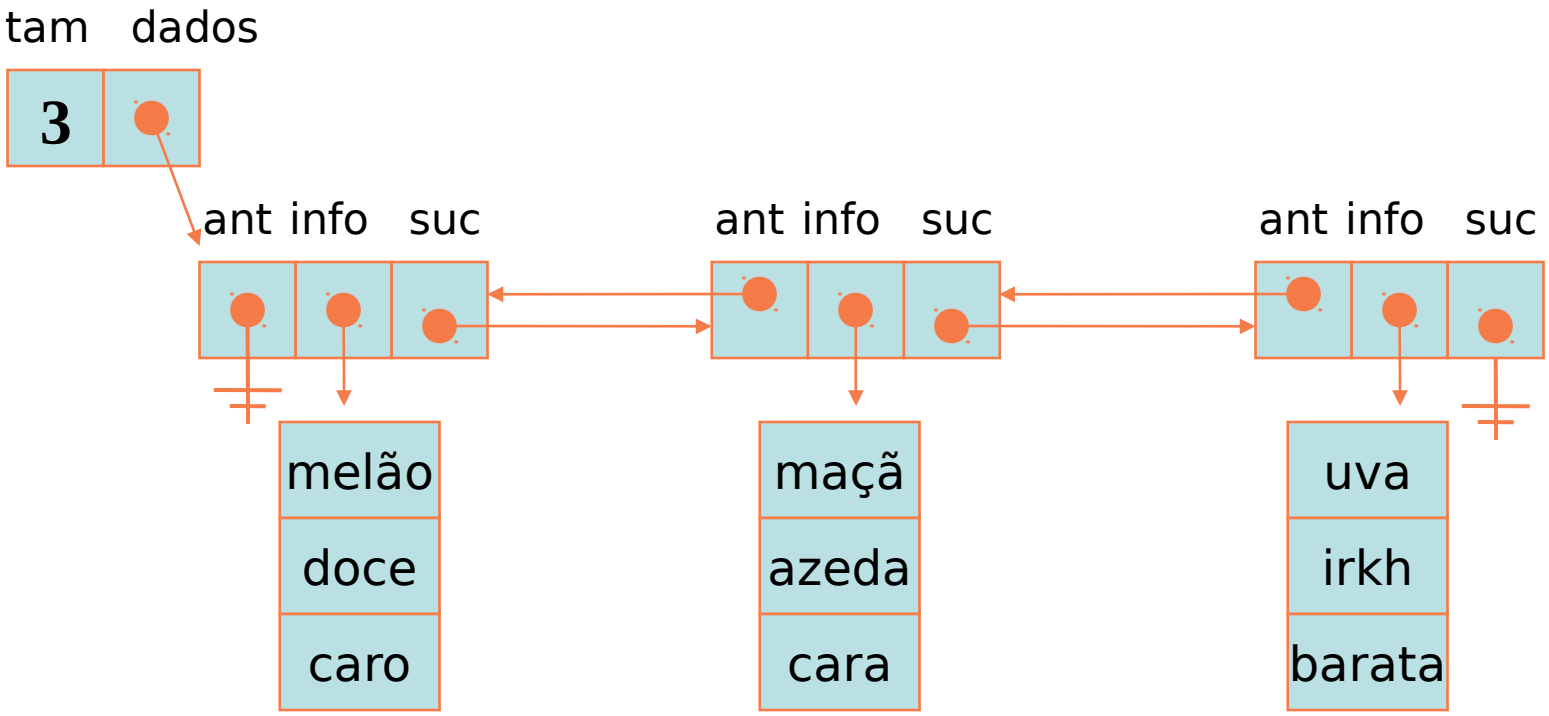
Exercício

- Implemente uma classe FilaEncadeada
- Implemente a fila usando Templates
- Use as melhores práticas de orientação a objetos
- Documente todas as classes, métodos e atributos.
- Aplique os testes unitários disponíveis no moodle da disciplina para validar sua estrutura de dados.

Listas Duplamente Encadeadas

- A Lista Encadeada e a Fila Encadeada possuem a desvantagem de somente podermos caminhar em uma direção:
  - vimos que para olhar um elemento pelo qual “acabamos de passar” precisamos de uma variável auxiliar “anterior”;
  - para olhar 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 a frente e para trás.

Listas Duplamente Encadeadas - Modelagem



Modelagem: Cabeça de ListaDupla

- Necessitamos:
  - um ponteiro para o primeiro elemento da lista;
  - um inteiro para indicar quantos elementos a lista possui.

- Pseudo-código:

```
classe tListaDupla {
   tElementoDuplo *dados;
   inteiro tamanho;
   };
```

Modelagem: Elemento de ListaDupla

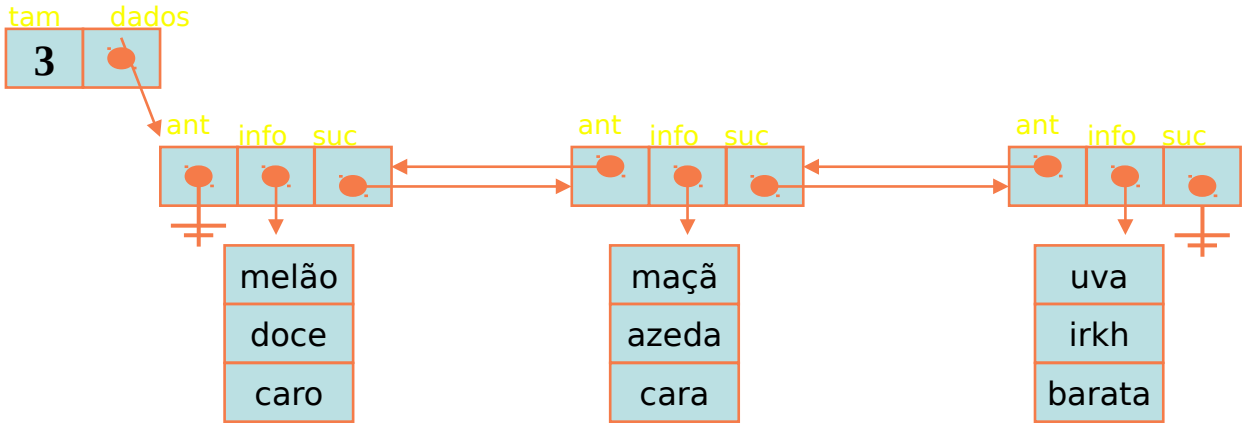
- Necessitamos:
  - um ponteiro para o elemento anterior na lista;
  - um ponteiro para o elemento sucessor na lista;
  - um ponteiro para a informação que vamos armazenar.

- Pseudo-código:

```
classe tElementoDuplo {
  tElementoDuplo *anterior;
  tElementoDuplo *sucessor;
  T *info;
  };
```

Modelagem da Lista Duplamente Encadeada

- Aspecto Funcional:
  - colocar e retirar dados da lista;
  - testar se a lista está vazia e outros testes;
  - inicializá-la e garantir a ordem dos elementos.



Modelagem da Lista Duplamente Encadeada

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

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

Modelagem da Lista Duplamente Encadeada

- Operações - testar a lista e outros testes:
  - ListaVaziaDuplo()
  - PosiçãoDuplo(dado)
  - ContémDuplo(dado)
- Operações - inicializar ou limpar:
  - CriaListaDupla()
  - DestróiListaDupla()

Algoritmo CriaListaDupla

```
Método criaListaDupla()
   //Inicializa as variáveis
       dados <- NULO;
       tamanho <- 0;
   fim;
```

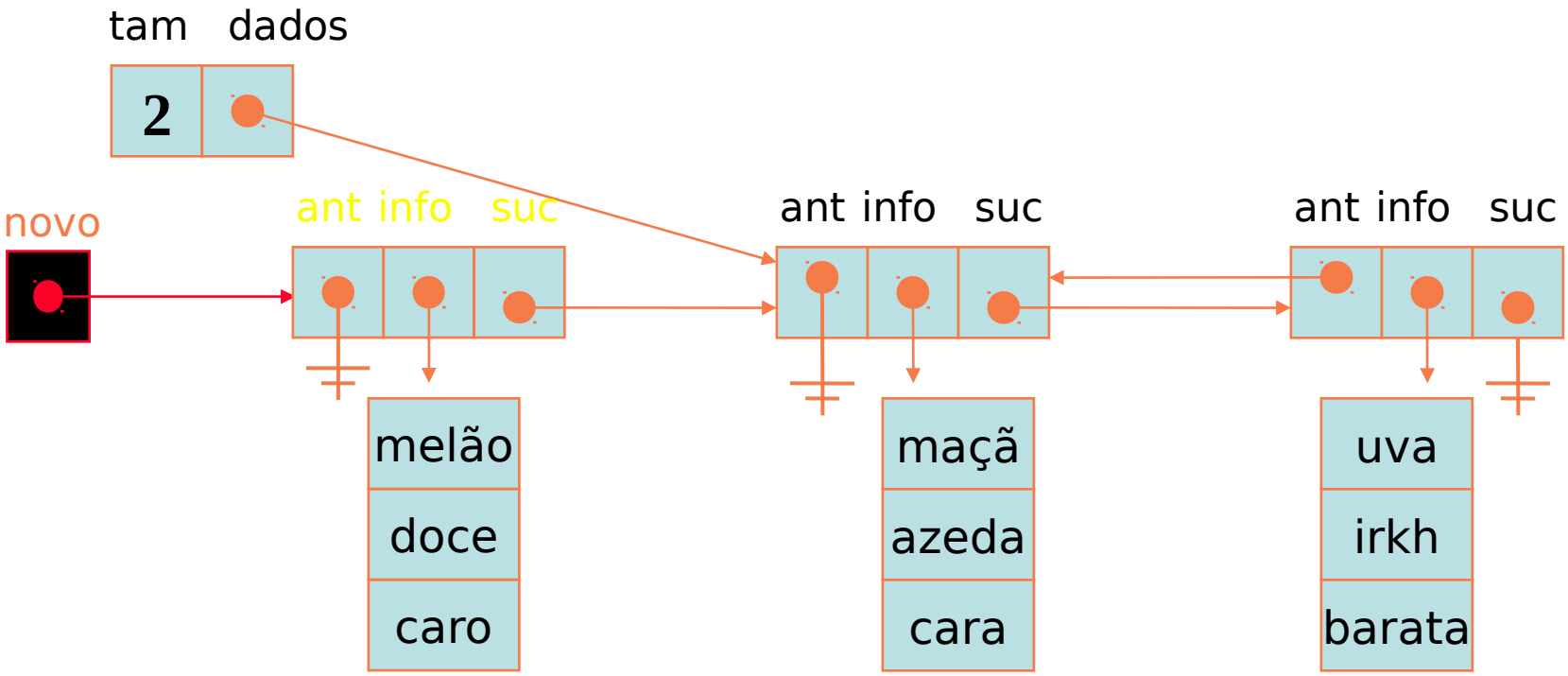
Algoritmo ListaVaziaDuplo

```
Booleano Método listaVaziaDuplo()
  início
      SE (tamanho = 0) ENTÃO
            RETORNE(Verdadeiro)
      SENÃO
            RETORNE(Falso);
  fim;
```

Algoritmo AdicionaNoInícioDuplo

- Procedimento:
  - 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 elemento.
- Parâmetros:
  - o dado a ser inserido;

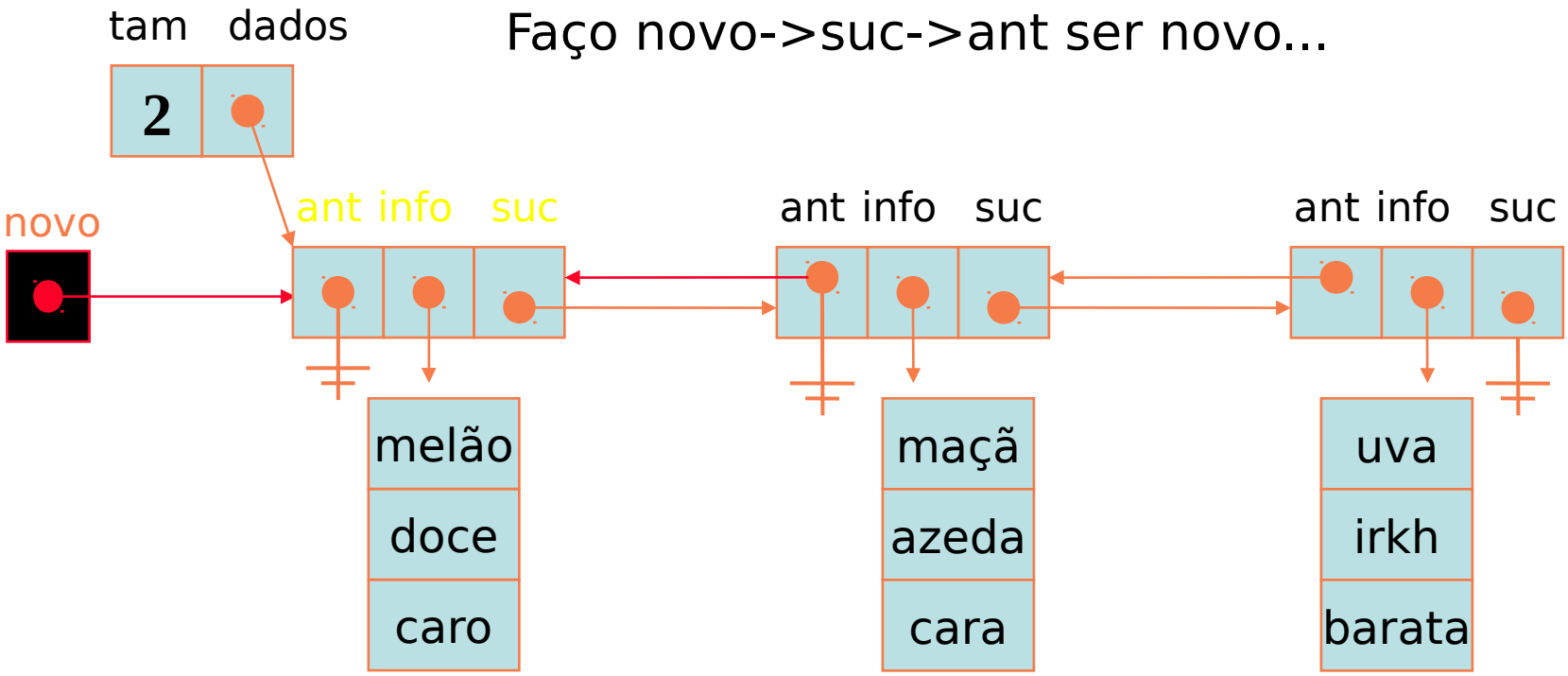
Algoritmo AdicionaNoInícioDuplo



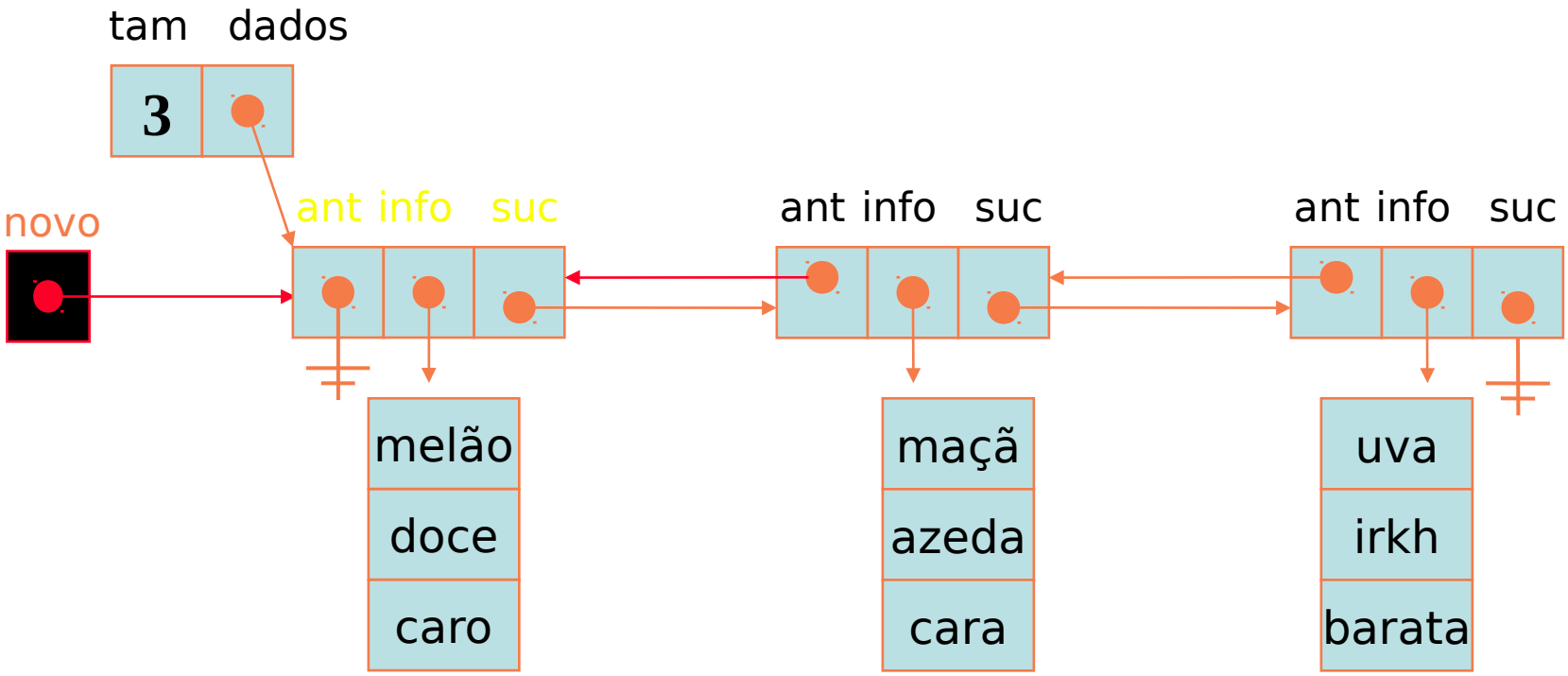
Algoritmo AdicionaNoInícioDuplo

Caso novo->suc não seja nulo...

Faço novo->suc->ant ser novo...



Algoritmo AdicionaNoInícioDuplo



Algoritmo AdicionaNoInícioDuplo

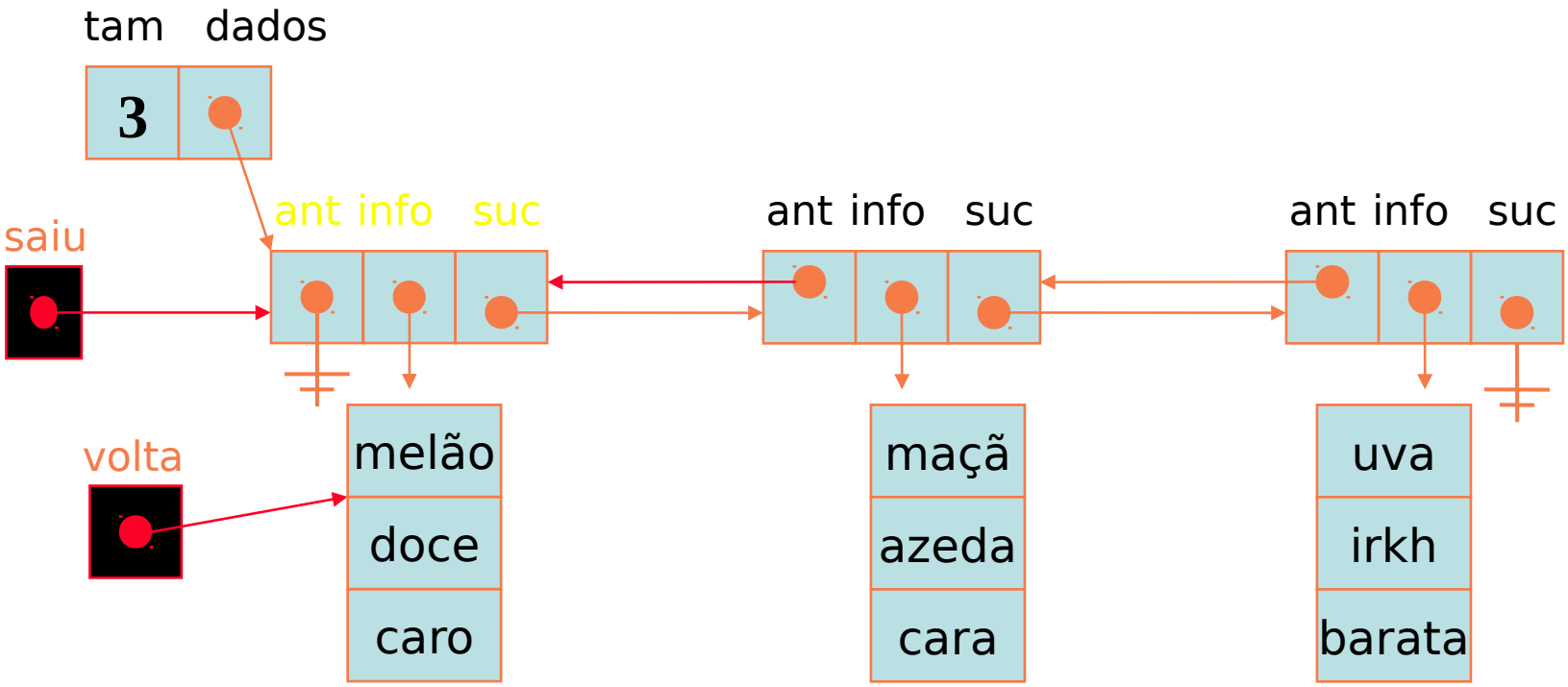
```
Inteiro MÉTODO adicionaNoInícioDuplo(T *dado)
   variáveis
        tElementoDuplo *novo; //Variável auxiliar para o novo elemento.
   início
        novo <- aloque(tElementoDuplo);
       SE ( novo == NULO )
         THROW LISTACHEIA;
        novo->suc <- dados;
          novo->ant <- NULO;
          novo->info <- dado;
          dados <- novo;
          SE (novo->suc ~= NULO) ENTÃO
                 novo->suc->ant <- novo;
          FIM SE;
          tamanho <- tamanho + 1;
          RETORNE(1);

   fim;
```

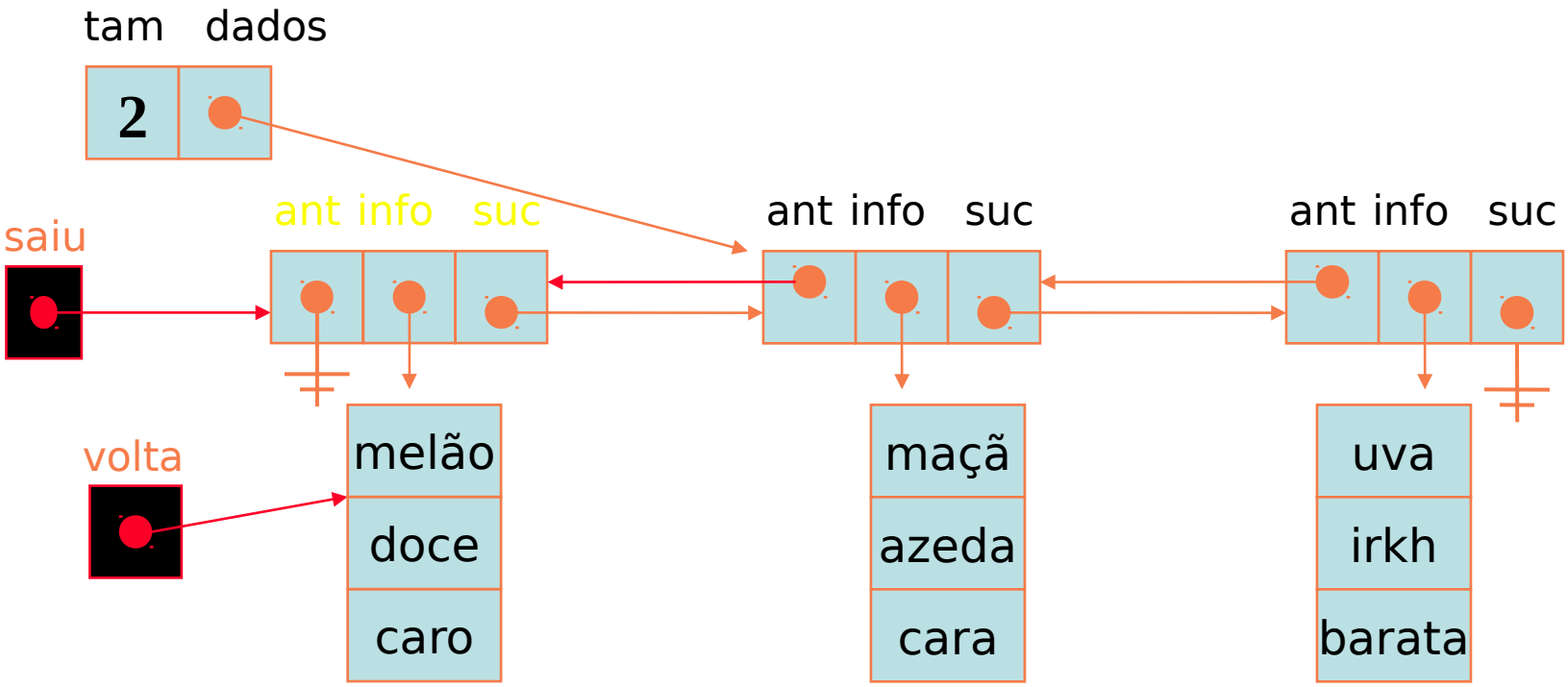
Algoritmo RetiraDoInícioDuplo

- 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.

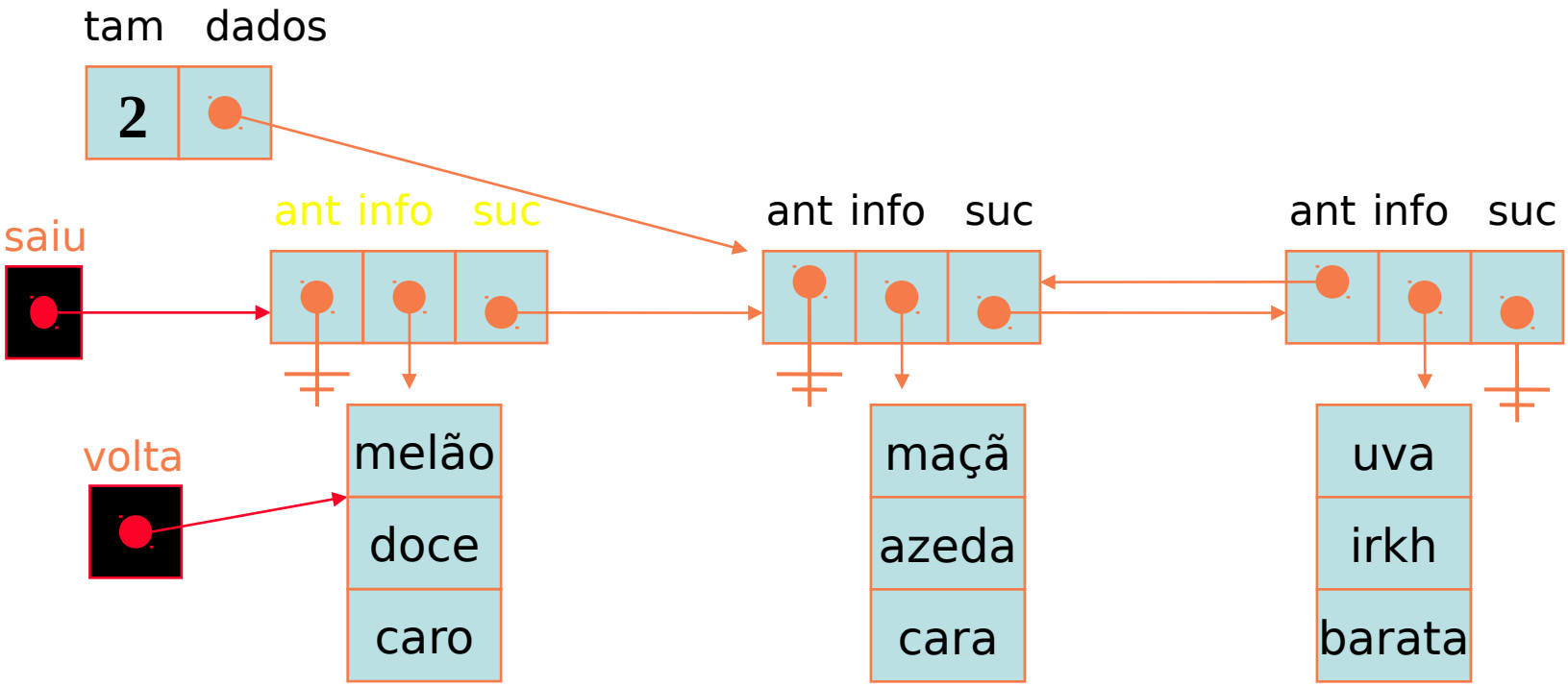
Algoritmo RetiraDoInícioDuplo



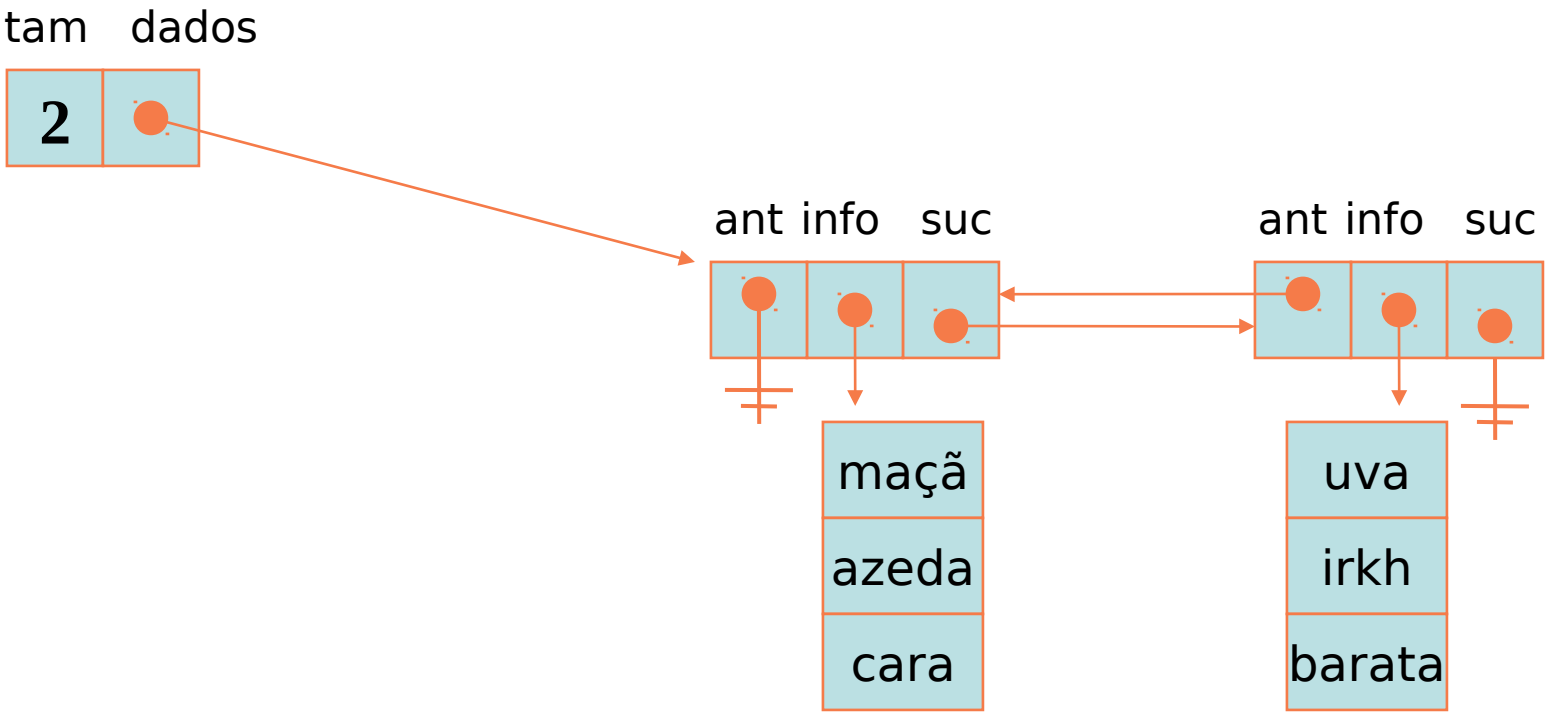
Algoritmo RetiraDoInícioDuplo



Algoritmo RetiraDoInícioDuplo



Algoritmo RetiraDoInícioDuplo



Algoritmo RetiraDoInícioDuplo

```
T* MÉTODO retiraDoInícioDuplo()
   //Elimina o primeiro elemento de uma lista duplamente encadeada.
   //Retorna a informação do elemento eliminado ou NULO.
   variáveis
        tElementoDuplo *saiu; //Variável auxiliar para o primeiro elemento.
        T *volta; //Variável auxiliar para o dado retornado.
   início
        SE (listaVaziaDuplo()) ENTÃO
          THROW LISTAVAZIA;
        SENÃO
          saiu <- dados;
          volta <- saiu->info;
          dados <- saiu->suc;
          SE (dados ~= NULO) ENTÃO
                 dados->ant <- NULO;
          FIM SE
          tamanho <- tamanho - 1;
          LIBERE(saiu);
          RETORNE(volta);
        FIM SE
   fim;
```

Algoritmo AdicionaNaPosiçãoDuplo

- Praticamente idêntico à lista encadeada;
- Procedimento:
  - caminhamos até a posição;
  - adicionamos o novo dado na posição;
  - incrementamos o tamanho.
- Parâmetros:
  - o dado a ser inserido;
  - a posição onde inserir;

Algoritmo AdicionaNaPosição

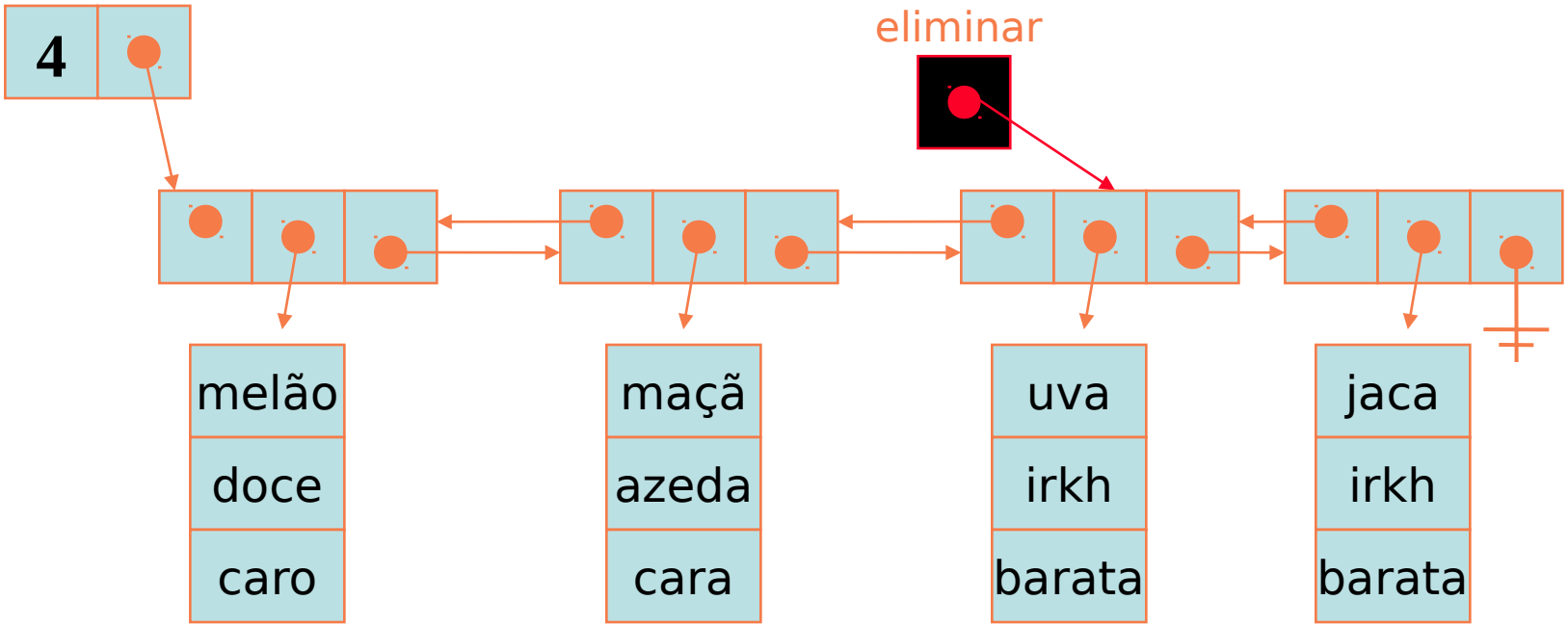
```
Inteiro MÉTODO adicionaNaPosiçãoDuplo(T *info, inteiro posição)
    //Adiciona novo elemento na posição informada.
    //Retorna o novo número de elementos da lista ou erro.
    variáveis
      tElementoDuplo *novo, *anterior; //Ponteiros auxiliares.
    início
      SE (posição > tamanho + 1 OU posição < 1) ENTÃO
           THROW(ERROPOSIÇÃO)
      SENÃO
           SE (posição = 1) ENTÃO
             RETORNE(adicionaNoInícioDuplo(info))
           SENÃO
             novo <- aloque(tElemento);
             SE (novo = NULO) ENTÃO
               THROW(ERROLISTACHEIA)
             SENÃO
               anterior <- 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; //Faço o antecessor do sucessor do novo.
               FIM SE
               novo->info <- info;
               anterior->suc <- novo;
               novo->ant <- anterior;
               tamanho <- tamanho + 1;
               RETORNE(tamanho);
             FIM SE
        FIM SE
      FIM SE
    fim;
```

Algoritmo RetiraDaPosiçãoDuplo

- Mais simples que na 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;

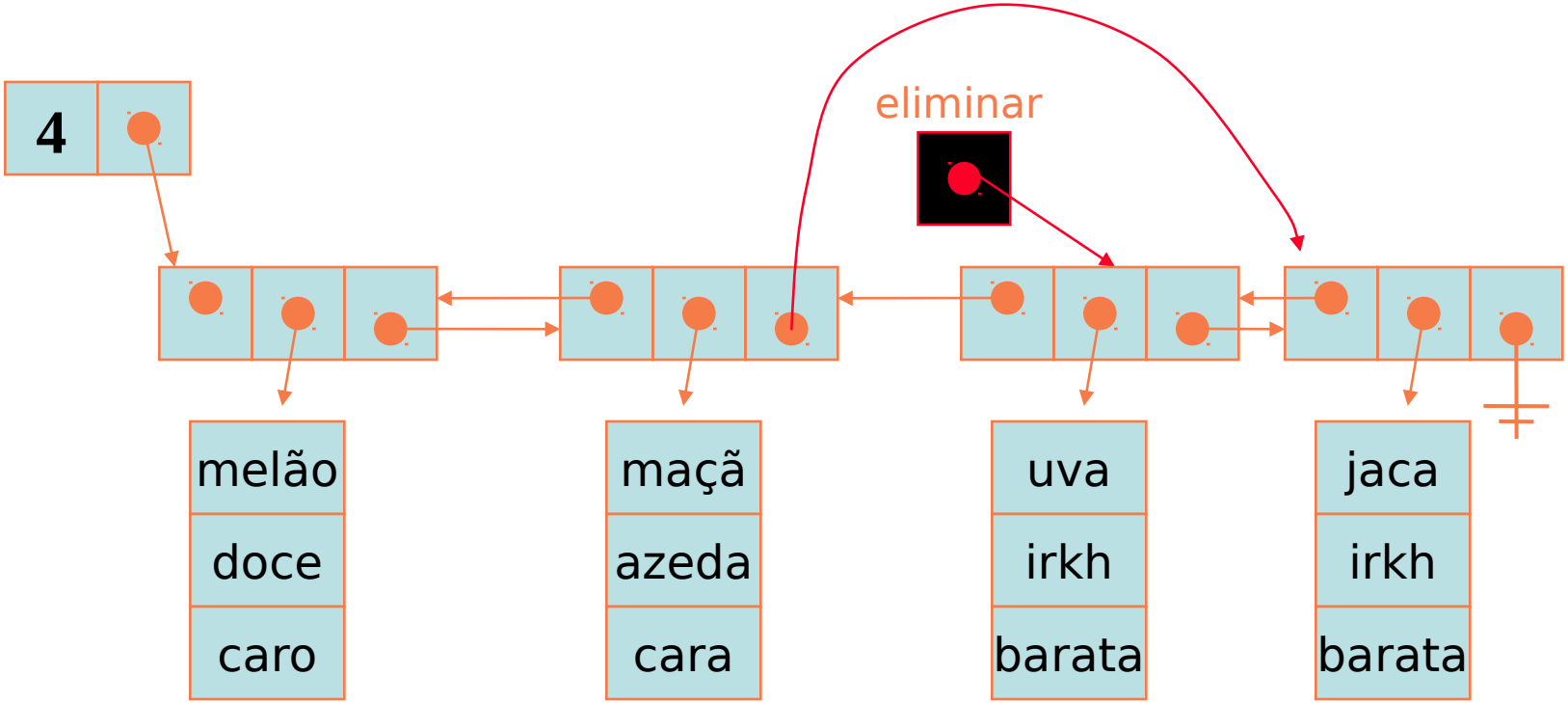
Algoritmo RetiraDaPosiçãoDuplo

Posições > 1



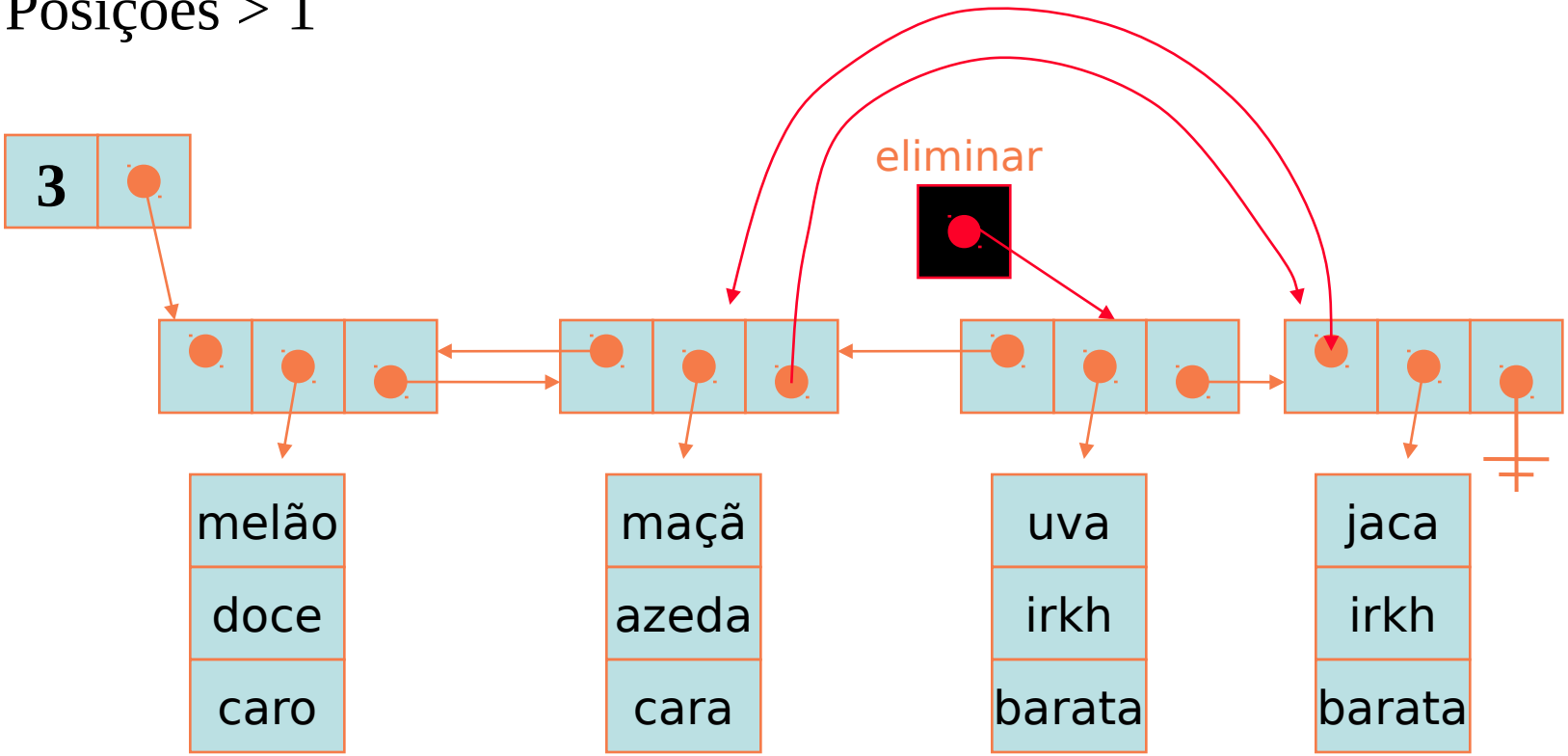
Algoritmo RetiraDaPosiçãoDuplo

Posições > 1



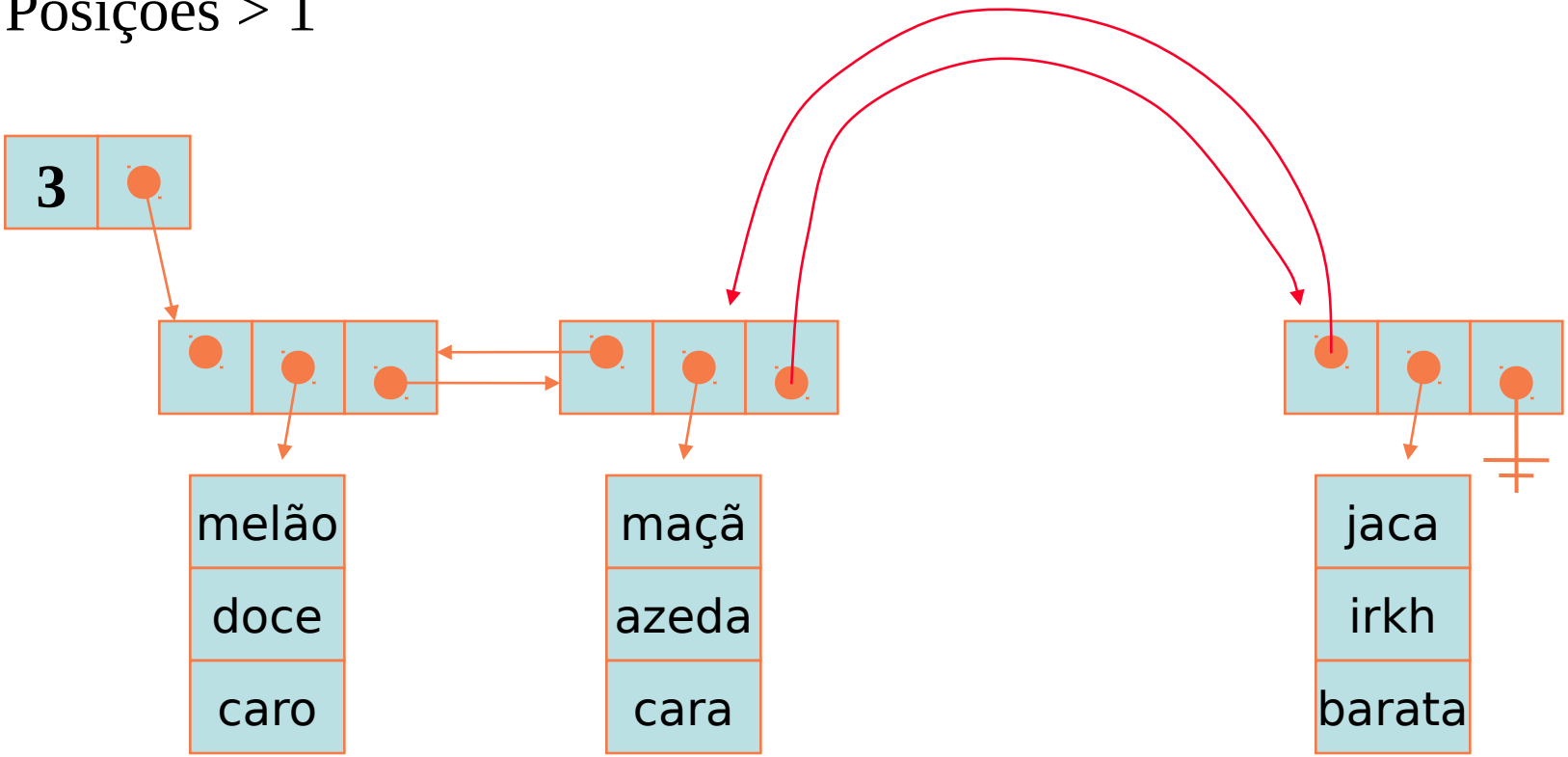
Algoritmo RetiraDaPosiçãoDuplo

Posições > 1



Algoritmo RetiraDaPosiçãoDuplo

Posições > 1



Algoritmo RetiraDaPosiçãoDuplo

```
T* MÉTODO retiraDaPosiçãoDuplo(inteiro posição)
    //Elimina o elemento da posição informada.
    //Retorna a informação do elemento eliminado ou NULO.
    variáveis
           tElementoDuplo *anterior, *eliminar; //Variável auxiliar para elemento.
           T *volta; //Variável auxiliar para o dado retornado.
    início
      SE (posição > tamanho OU posição < 1) ENTÃO
           THROW (ERROPOSICAO)
      SENÃO
           SE (posição = 1) ENTÃO
             RETORNE(retiraDoInícioDuplo())
           SENÃO
             anterior <- dados;
             REPITA (posição - 2) VEZES
                      anterior <- anterior->suc;
             eliminar <- anterior->suc;
             volta <- eliminar->info;
             anterior->suc <- eliminar->suc;
             SE eliminar->suc ~= NULO ENTÃO
                      eliminar->suc->ant <- anterior;
             FIM SE
             tamanho <- 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.

Algoritmo AdicionaEmOrdemDuplo

```
Inteiro Método adicionaEmOrdemDuplo(T *dado)
   variáveis
     tElementoDuplo *atual; //Variável auxiliar para caminhar.
     inteiro posição;
   início
     SE (listaVaziaDupla()) ENTÃO
       RETORNE(adicionaNoInícioDuplo( dado))
     SENÃO
       atual <- dados;
       posição <- 1;
       ENQUANTO (atual->suc ~= NULO E maior(dado, atual->info)) FAÇA
          //Encontrar posição para inserir.
          atual <- atual->suc;
          posição <- posição + 1;
        FIM ENQUANTO
        SE maior(dado, atual->info) ENTÃO //Parou porque acabou a lista.
          RETORNE(adicionaNaPosiçãoDuplo(dado,posição + 1))
        SENÃO
          RETORNE(adicionaNaPosiçãoDuplo(dado, posição));
        FIM SE
     FIM SE
   fim;
```

Por conta do aluno:

- Operações de inclusão e exclusão:
  - AdicionaDuplo(dado)
  - RetiraDuplo()
  - RetiraEspecíficoDuplo(dado)
- Operações - inicializar ou limpar:
  - DestróiListaDupla()