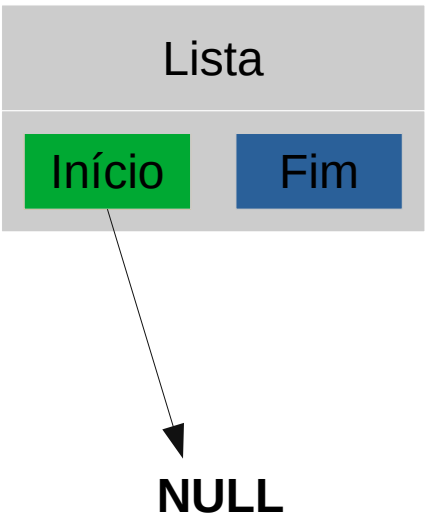
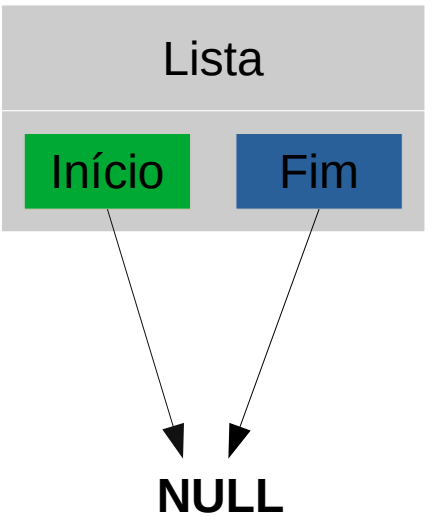
Exercício 1 - Passo 0. Criar lista

```
criarLista():
inicio ← NULO;
fim ← NULO;

tamanho ← 0;
```



Passo 0. Criar lista

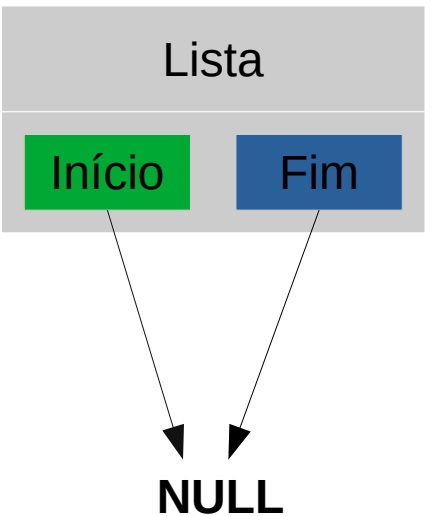
```
criarLista():
inicio ← NULO;
fim ← NULO;

tamanho ← 0;
```



Passo 0. Criar lista

```
criarLista():
inicio ← NULO;
fim ← NULO;

tamanho ← 0;
```

tamanho = 0

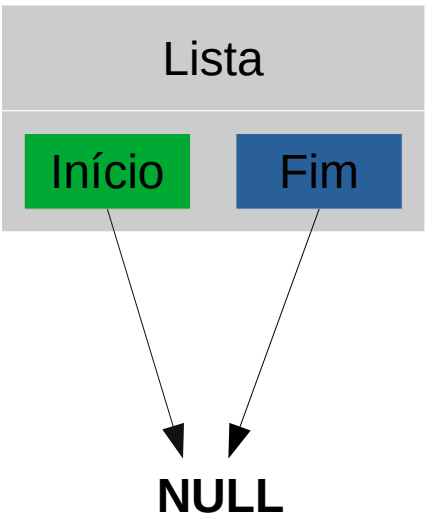


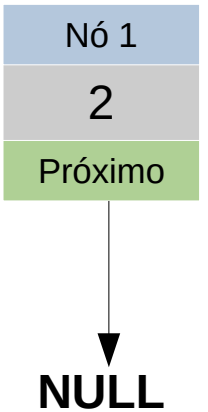
Passo 1. Inserir ordenado o elemento 2

inserirEmListaVazia(valor):

```
novo ← criar_noh(valor);
inicio ← novo;
fim ← novo;

tamanho ← 1;
```

tamanho = 0



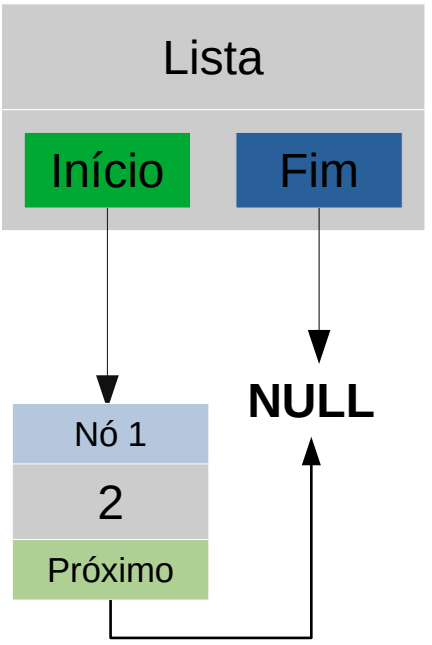


Passo 1. Inserir ordenado o elemento 2

inserirEmListaVazia(valor):

```
novo ← criar_noh(valor);
inicio ← novo;
fim ← novo;

tamanho ← 1;
```

tamanho = 0

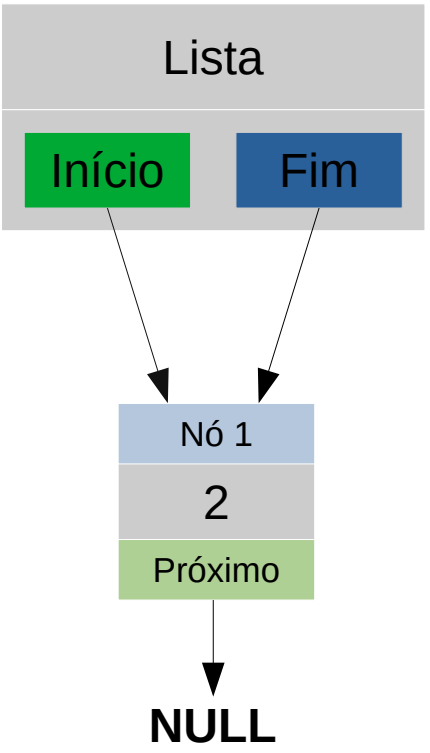


Passo 1. Inserir ordenado o elemento 2

inserirEmListaVazia(valor):

```
novo ← criar_noh(valor);
inicio ← novo;
fim ← novo;

tamanho ← 1;
```

tamanho = 0

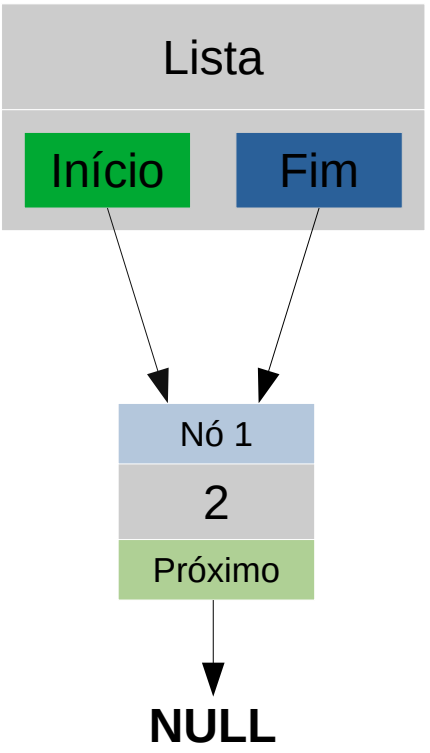


Passo 1. Inserir ordenado o elemento 2

inserirEmListaVazia(valor):

```
novo ← criar_noh(valor);
inicio ← novo;
fim ← novo;

tamanho ← 1;
```

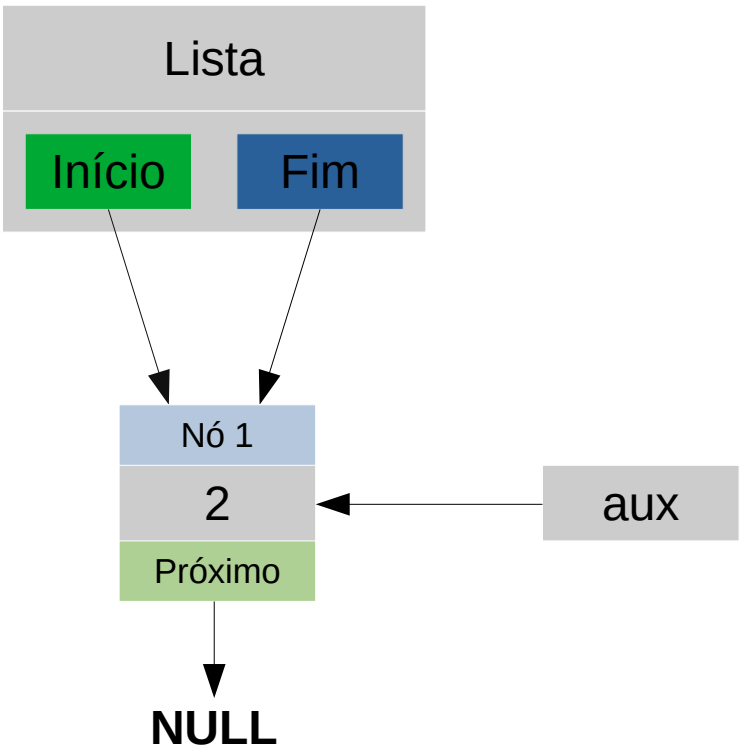
tamanho = 1



Passo 2. Remover elemento 2

```
Se (tamanaho = 1){
     aux = inicio
     Inicio = null
     Fim = null
     delete aux
     tamanho--

}
```

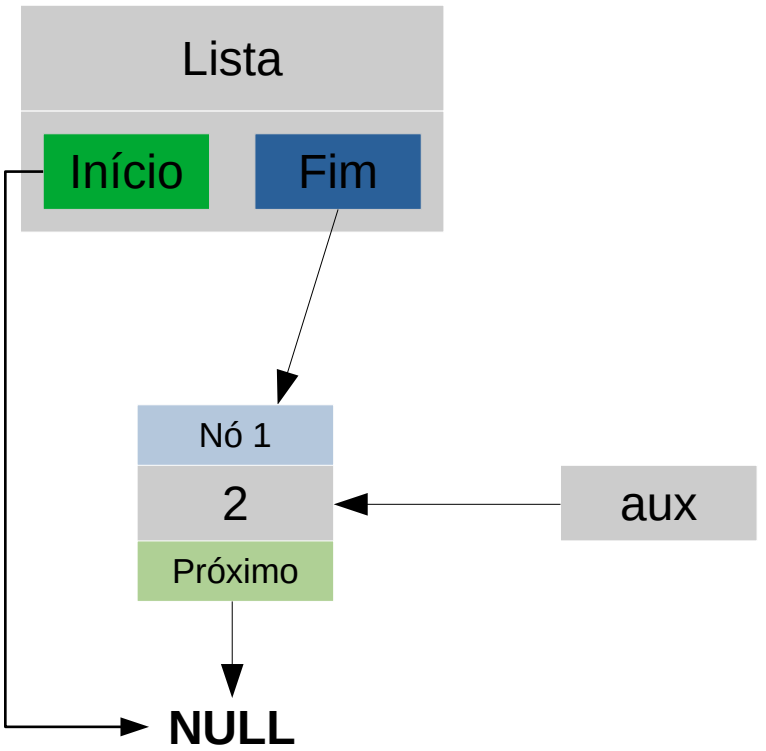
tamanho = 1



Passo 2. Remover elemento 2

```
Se (tamanaho = 1){
     aux = inicio
     Inicio = null
     Fim = null
     delete aux
     tamanho--

}
```

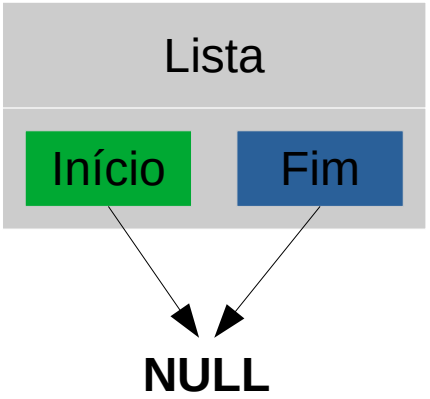
tamanho = 1

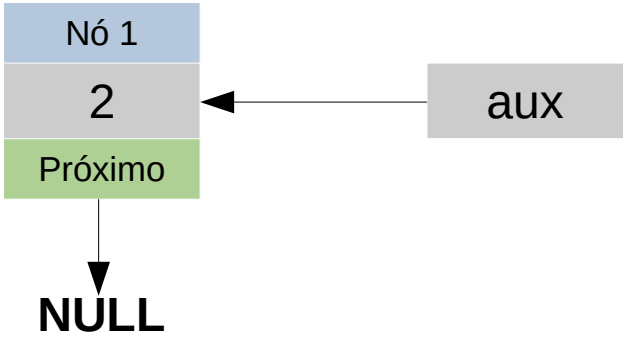


Passo 2. Remover elemento 2

```
Se (tamanaho = 1){
     aux = inicio
     Inicio = null
     Fim = null
     delete aux
     tamanho--

}
```

tamanho = 1

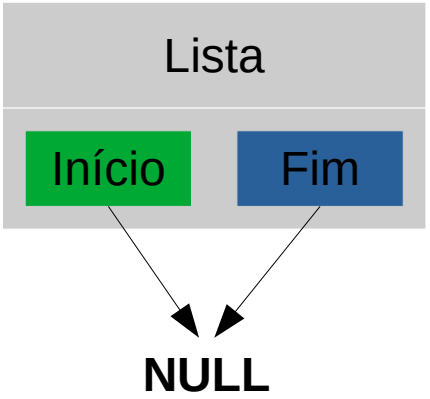


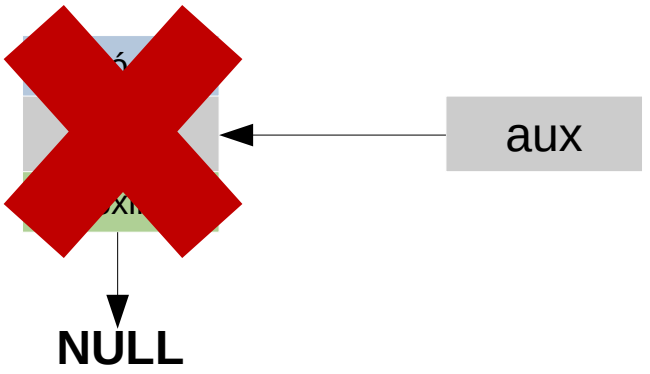


Passo 2. Remover elemento 2

```
Se (tamanaho = 1){
     aux = inicio
     Inicio = null
     Fim = null
     delete aux
     tamanho--

}
```

tamanho = 1

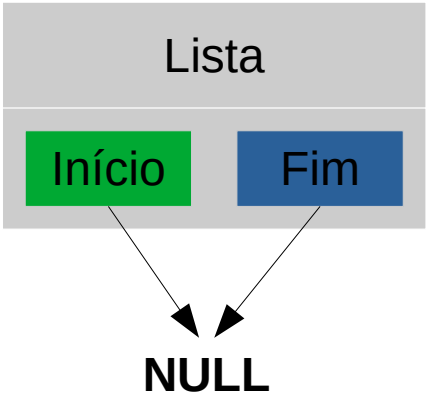




Passo 2. Remover elemento 2

```
Se (tamanaho = 1){
     aux = inicio
     Inicio = null
     Fim = null
     delete aux
     tamanho--

}
```

tamanho = 0

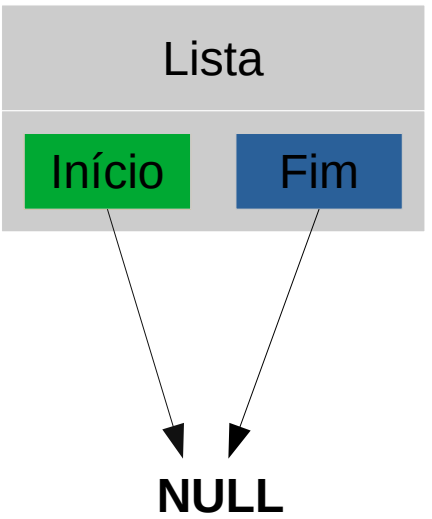


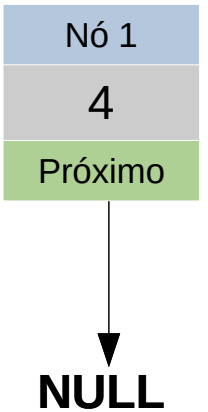
Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

inserirEmListaVazia(valor):

```
novo ← criar_noh(valor);
inicio ← novo;
fim ← novo;

tamanho ← 1;
```

tamanho = 0



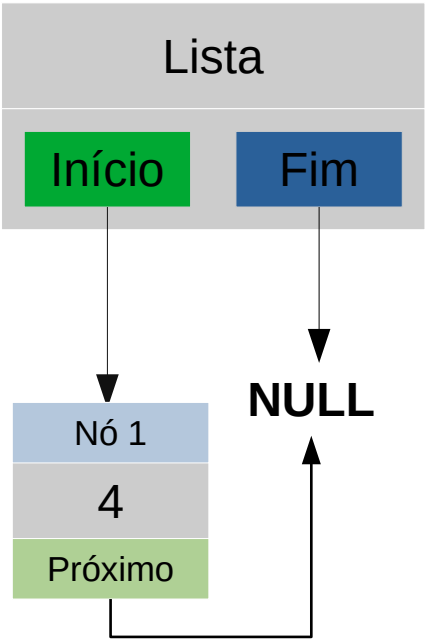


Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

inserirEmListaVazia(valor):

```
novo ← criar_noh(valor);
inicio ← novo;
fim ← novo;

tamanho ← 1;
```

tamanho = 0

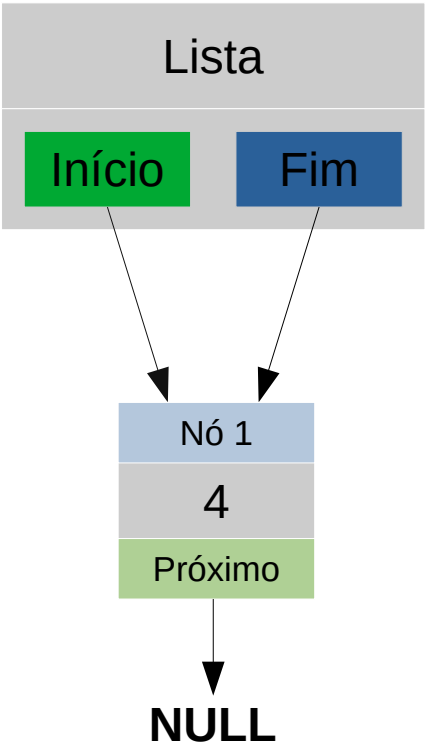


Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

inserirEmListaVazia(valor):

```
novo ← criar_noh(valor);
inicio ← novo;
fim ← novo;

tamanho ← 1;
```

tamanho = 0

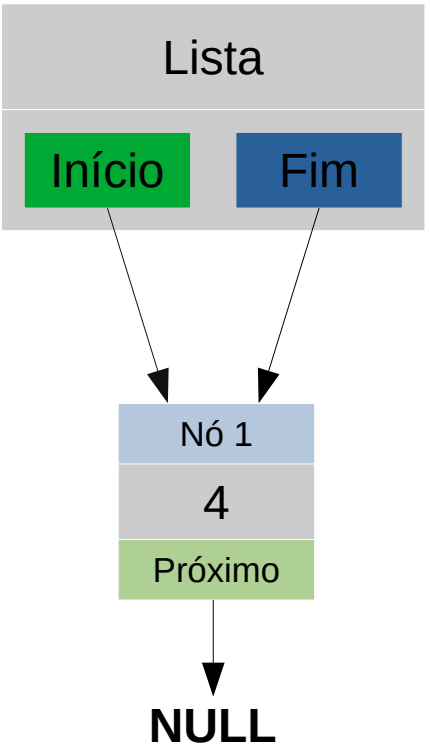


Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

inserirEmListaVazia(valor):

```
novo ← criar_noh(valor);
inicio ← novo;
fim ← novo;

tamanho ← 1;
```

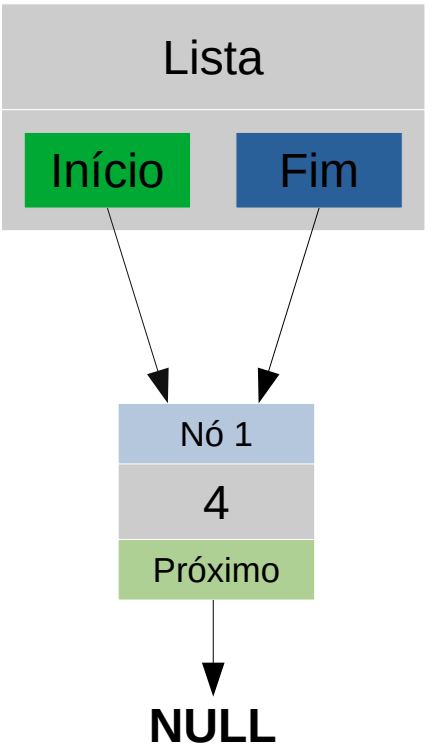
tamanho = 1

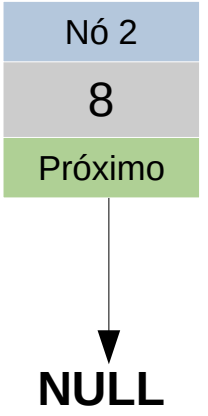


Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

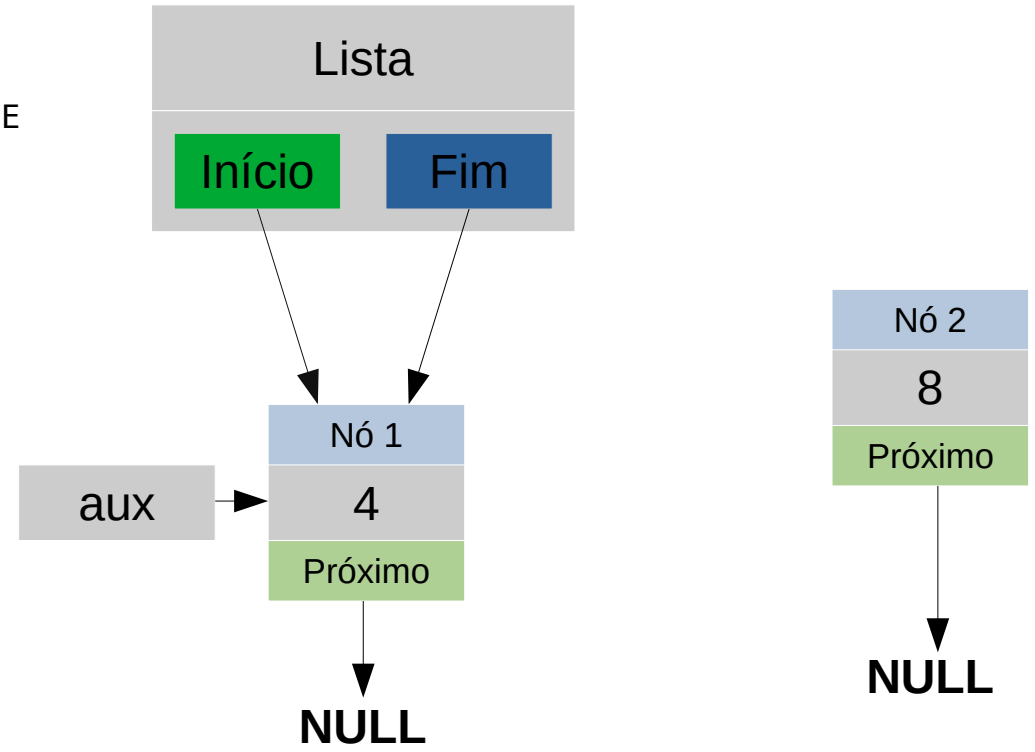
tamanho = 1





Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

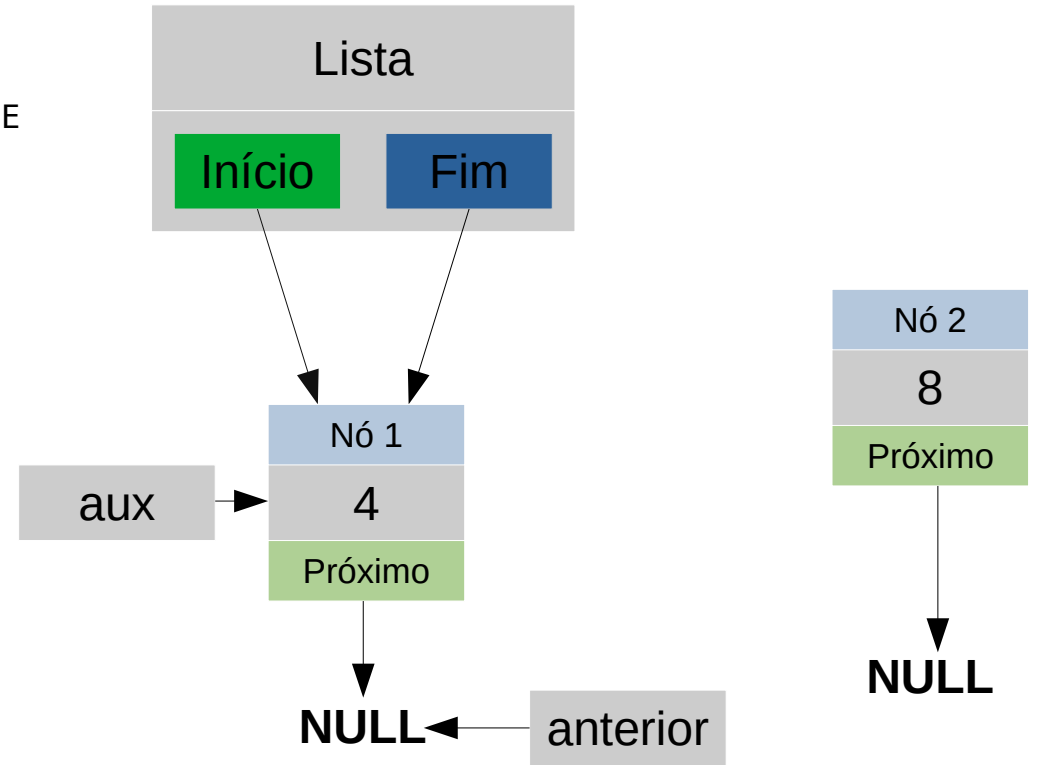
```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```



tamanho = 1

Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

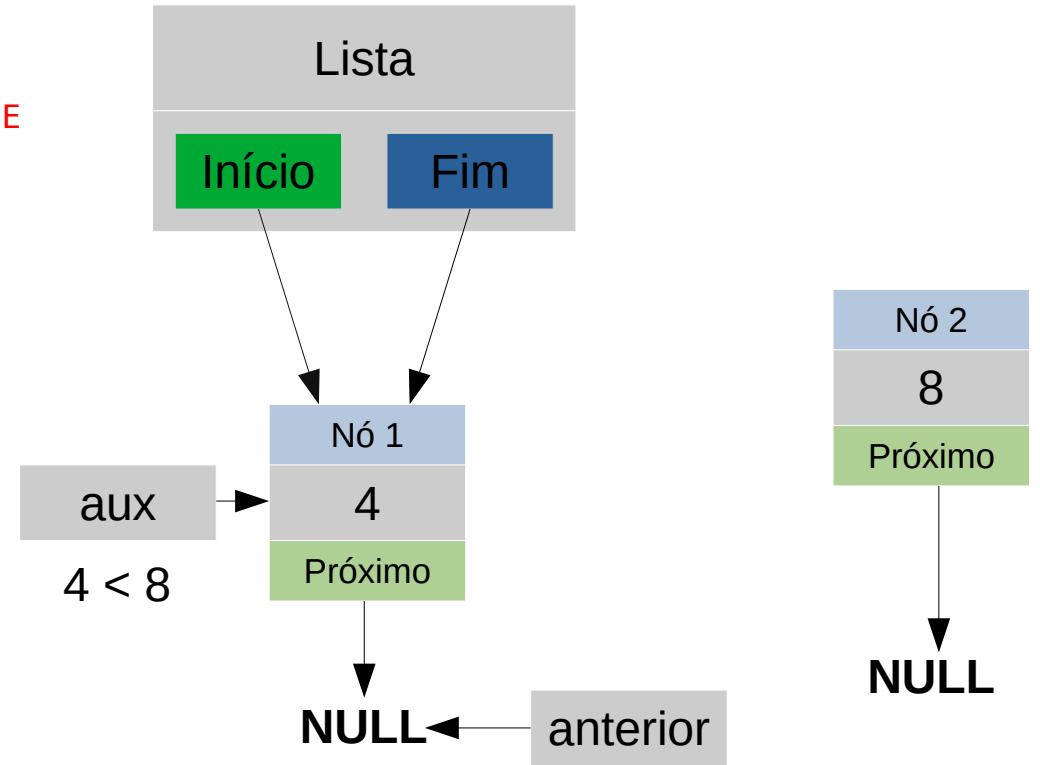
```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```



tamanho = 1

Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

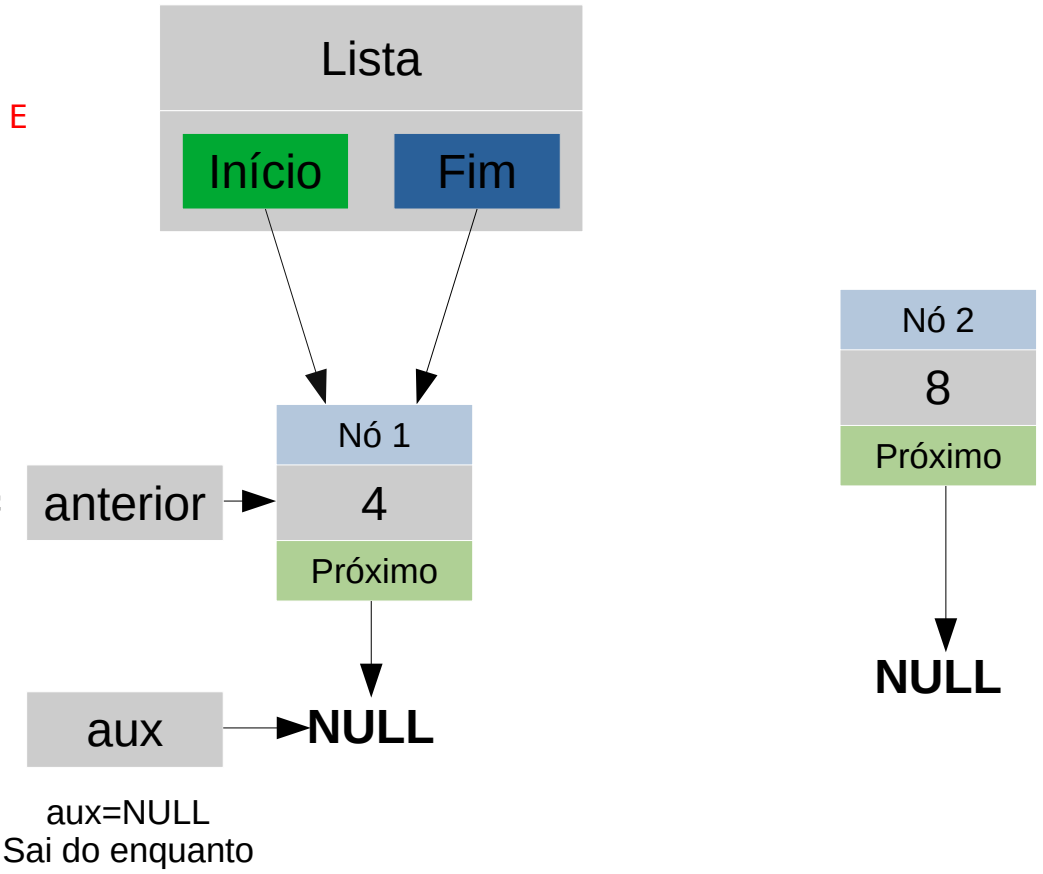


tamanho = 1

Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

tamanho = 1

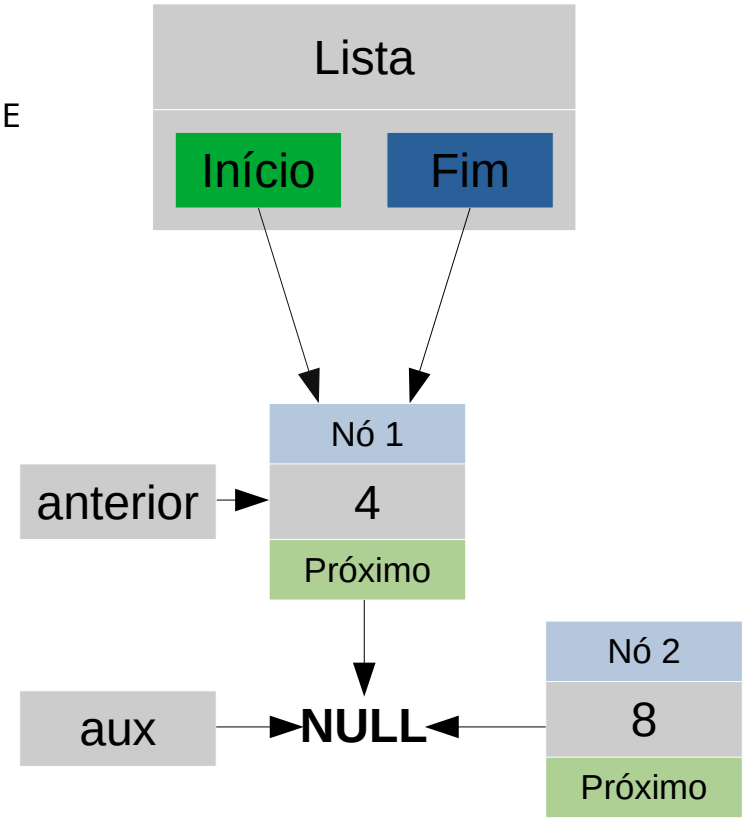


aux=NULL
Sai do enquanto

Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

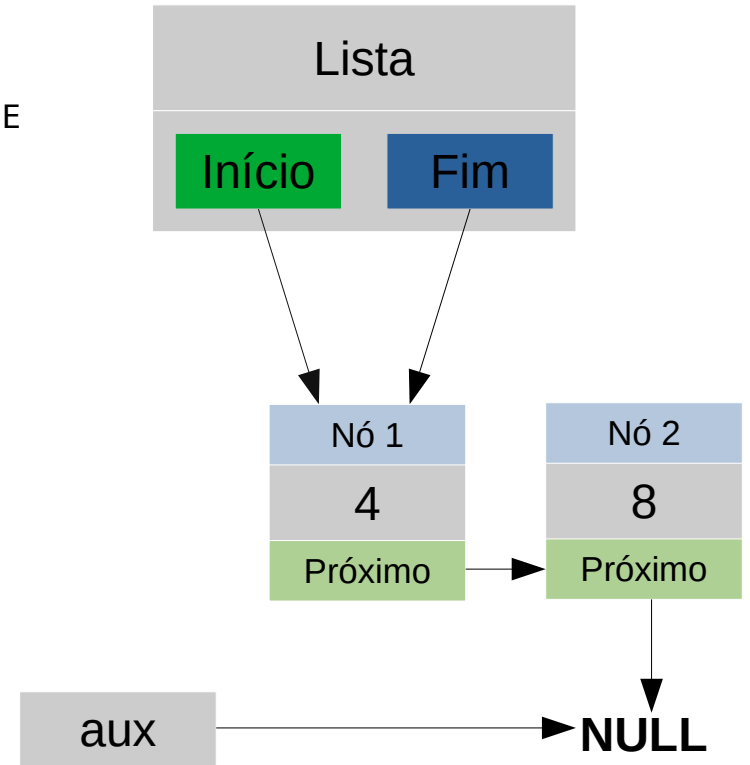
tamanho = 1



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

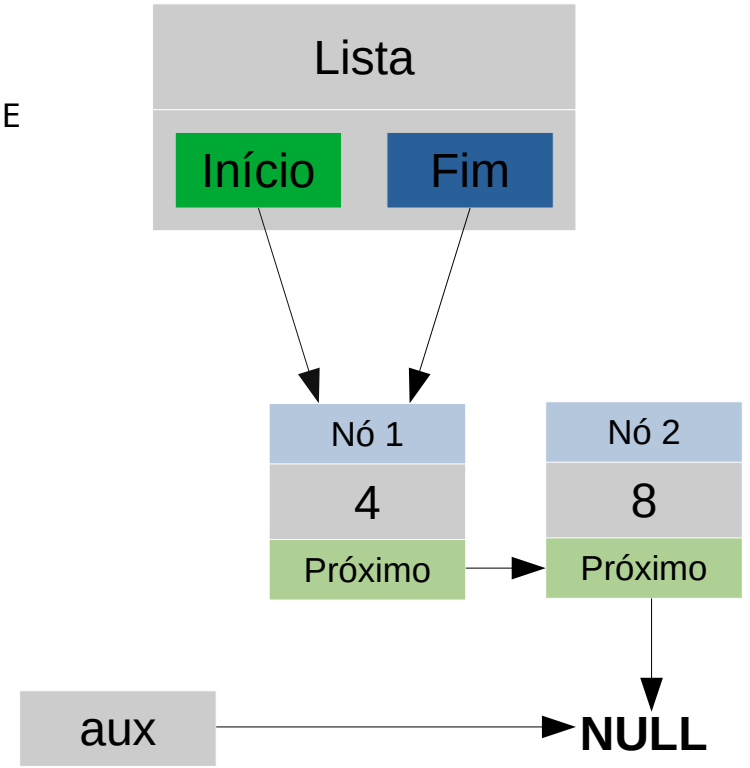
tamanho = 1



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

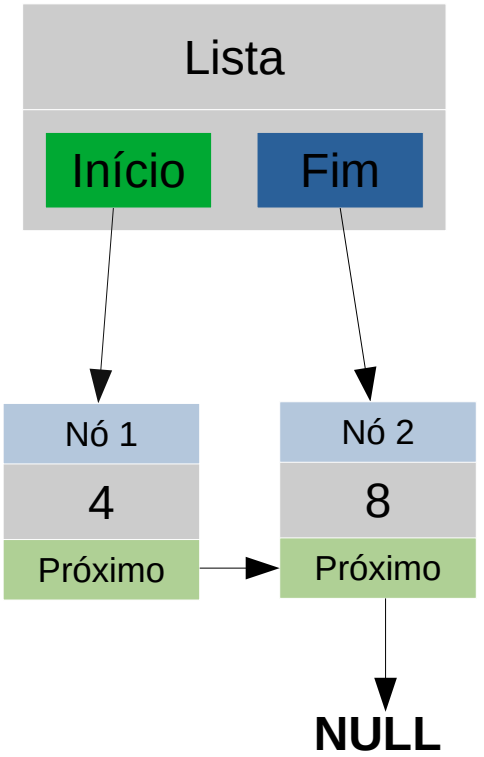
tamanho = 2



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

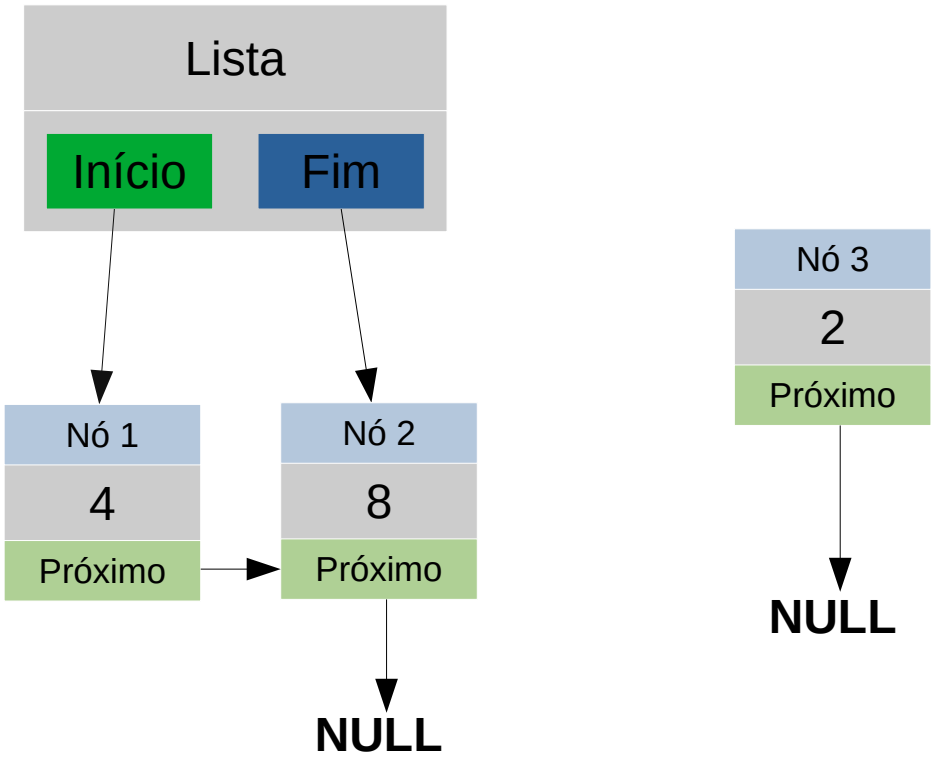
tamanho = 2



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

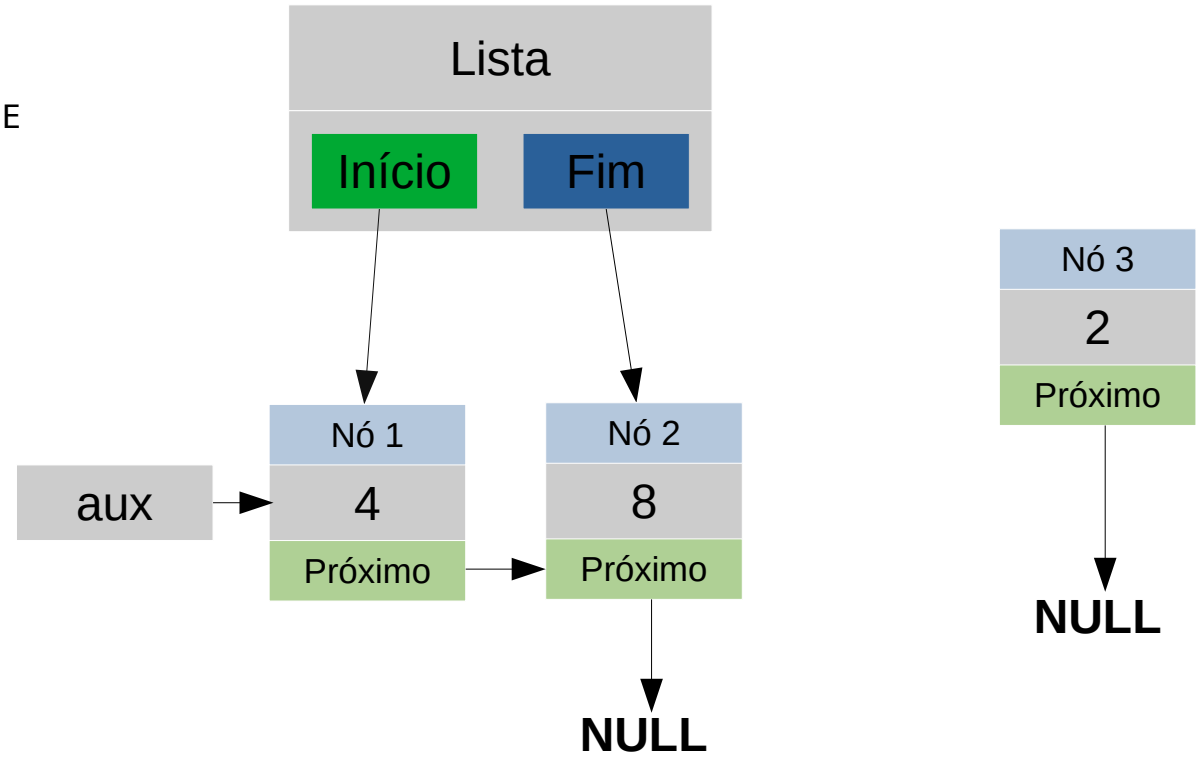
```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

tamanho = 2



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

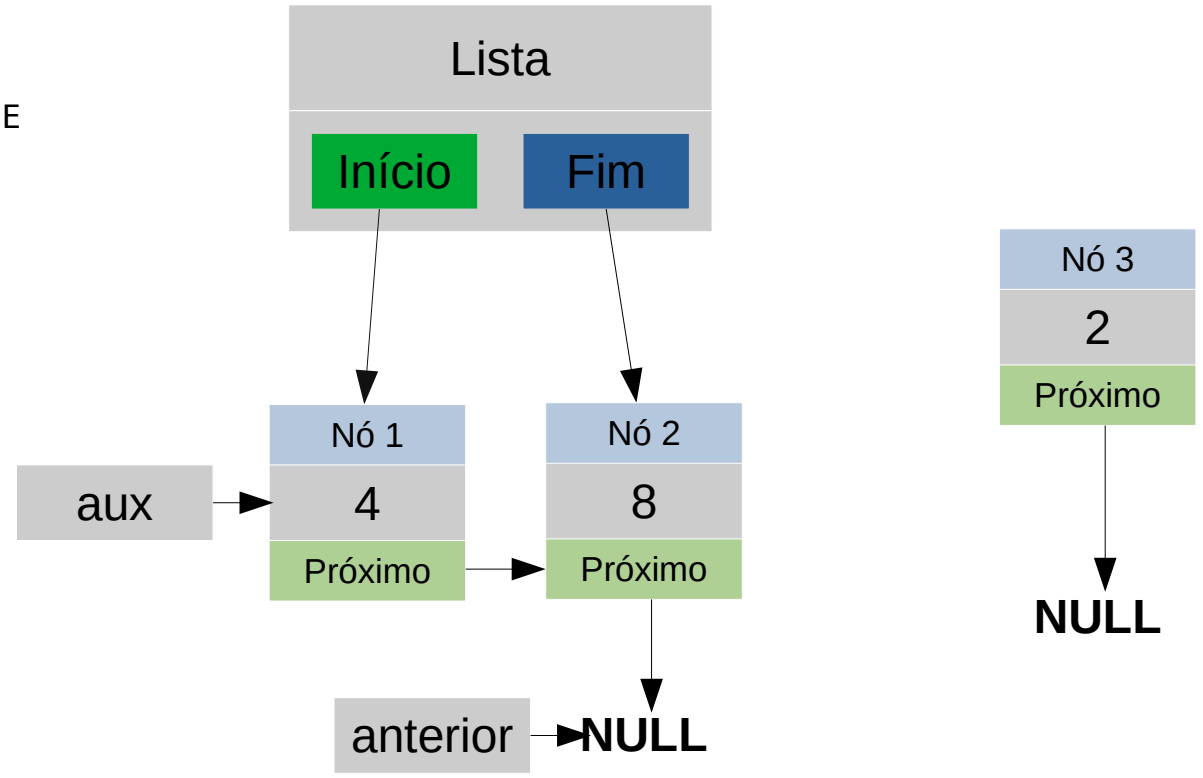


tamanho = 2

Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

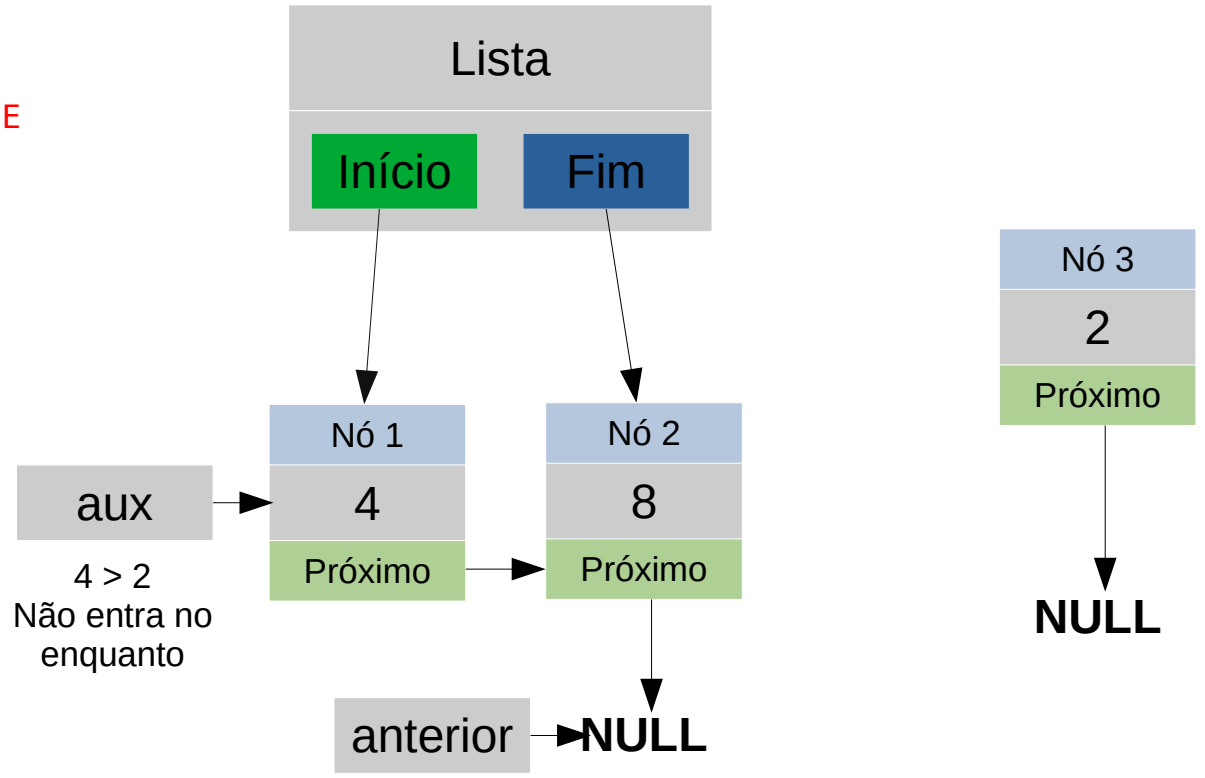
tamanho = 2



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

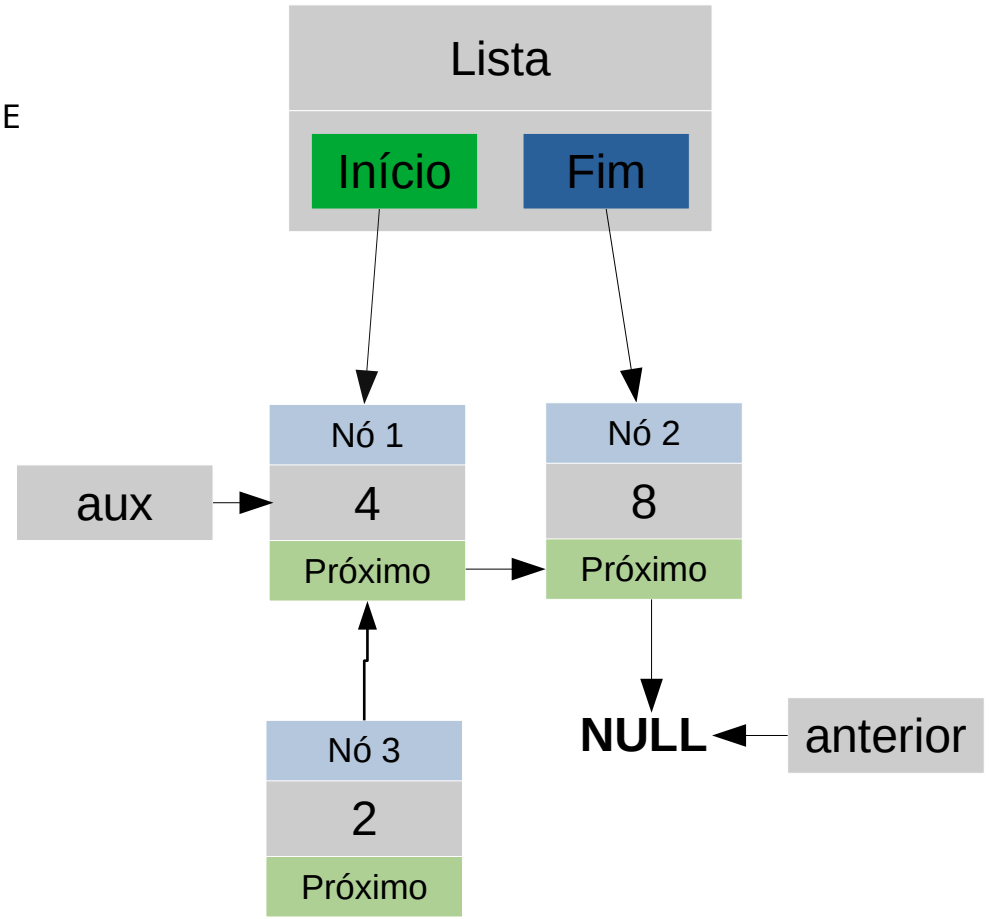
tamanho = 2



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

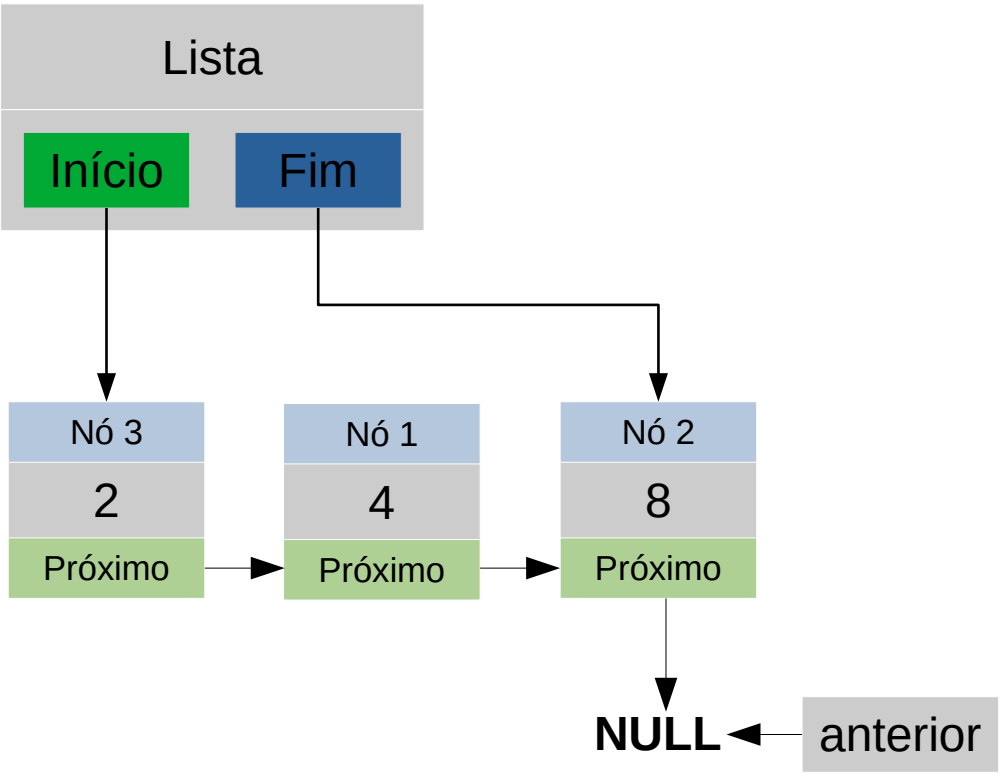
tamanho = 2



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

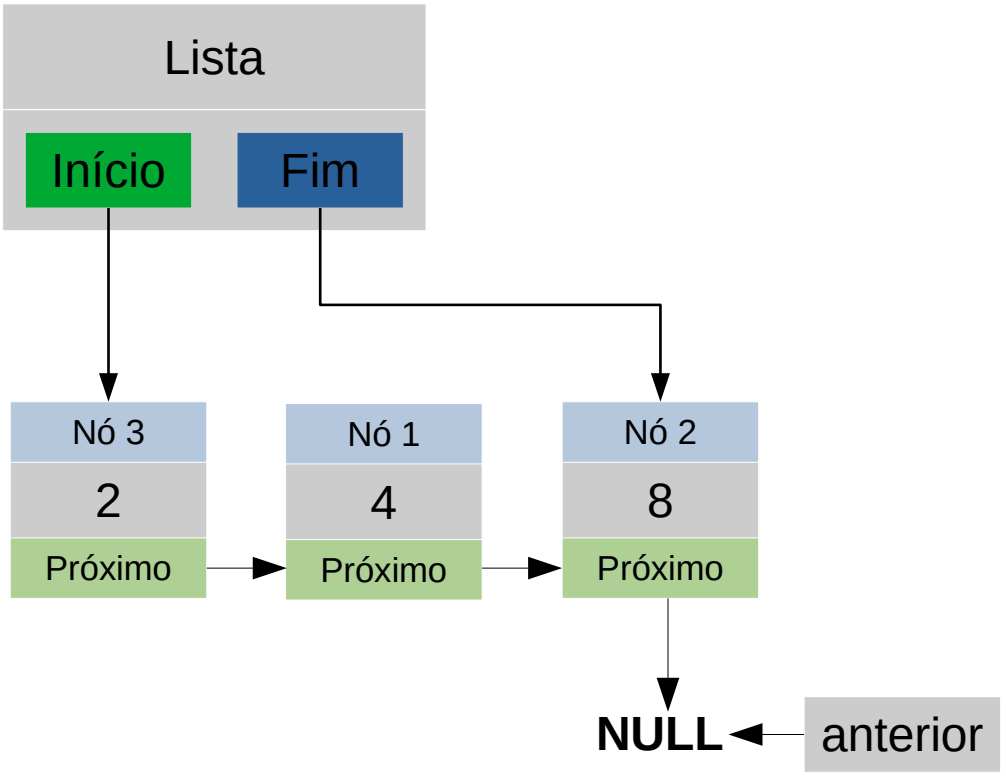
tamanho = 2



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

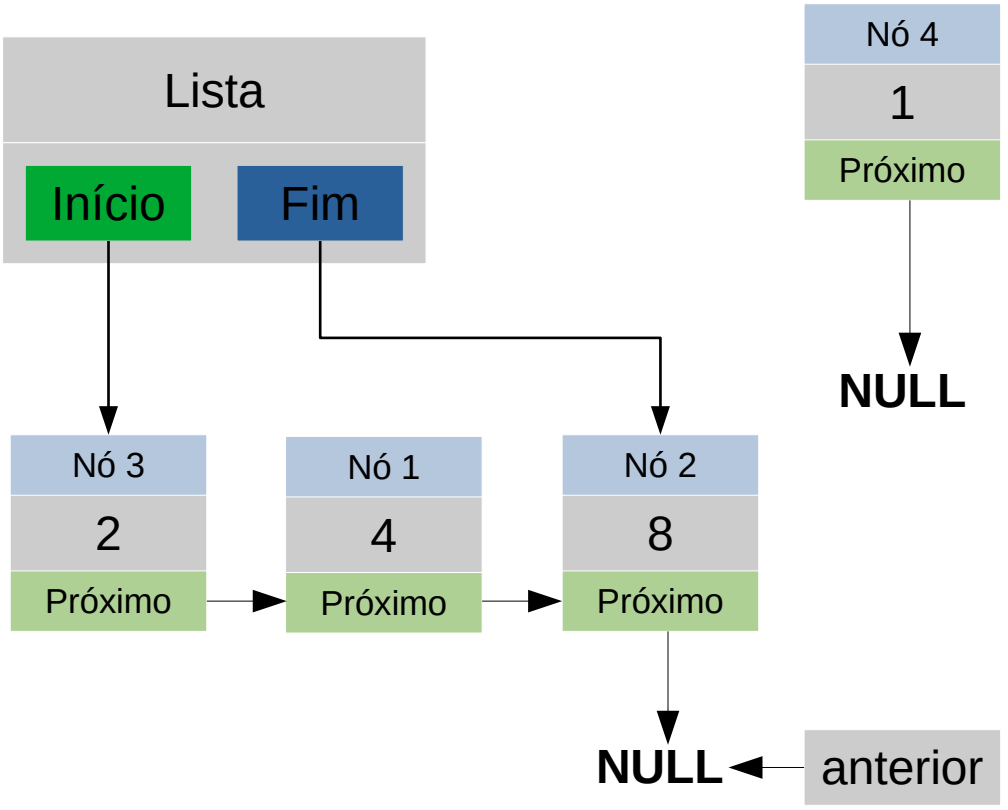
tamanho = 3



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

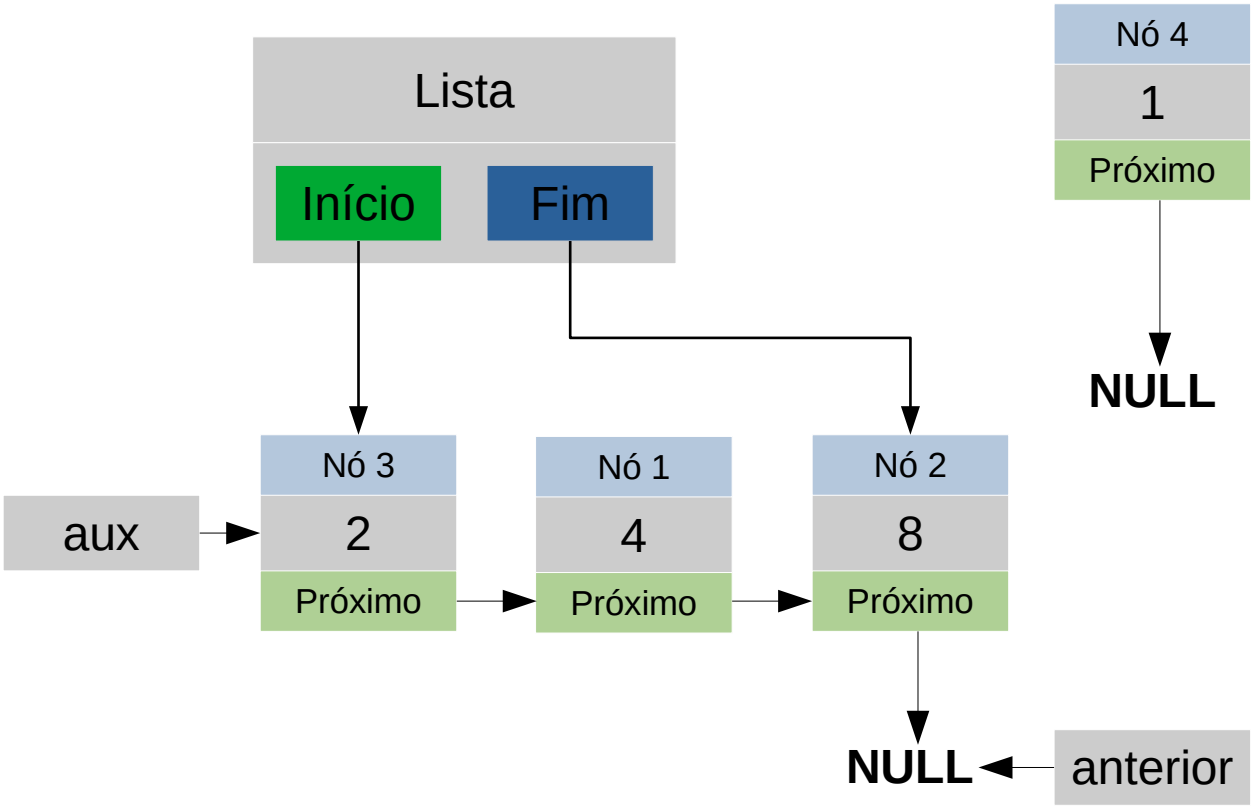
tamanho = 3



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

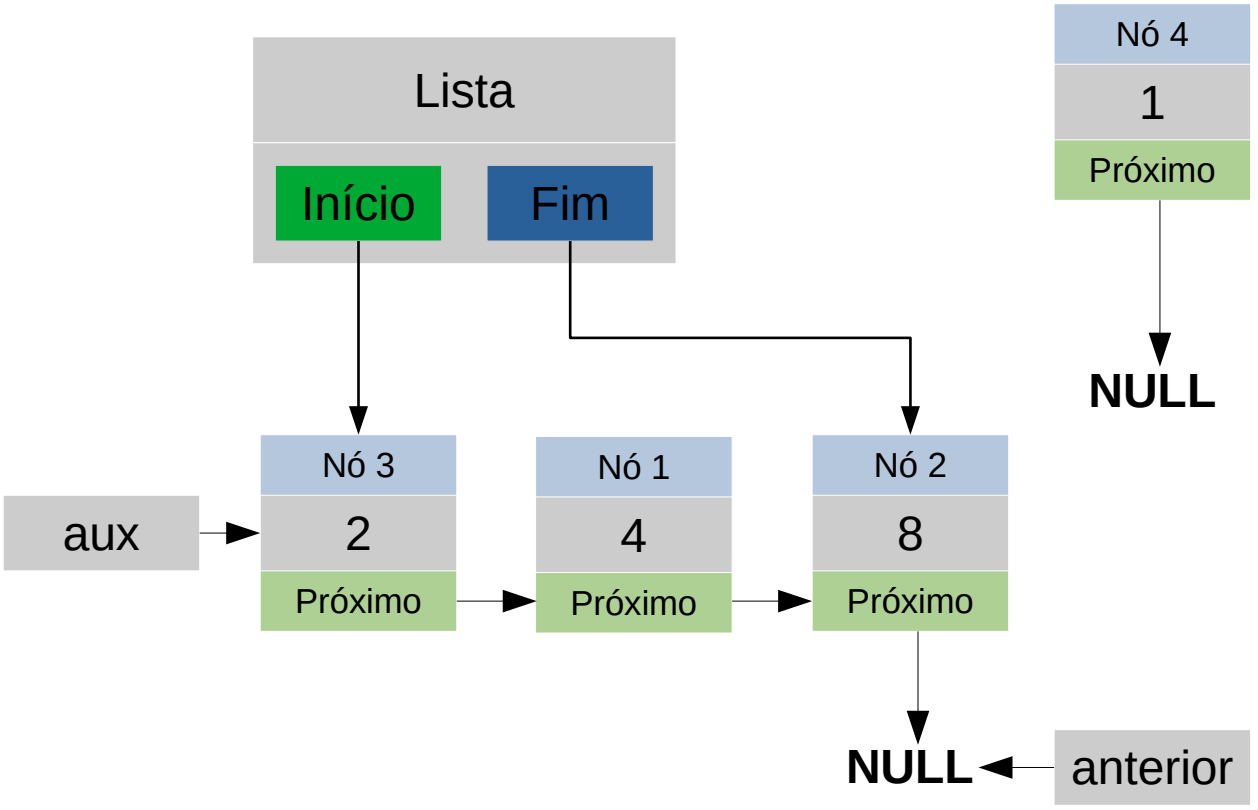
tamanho = 3



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

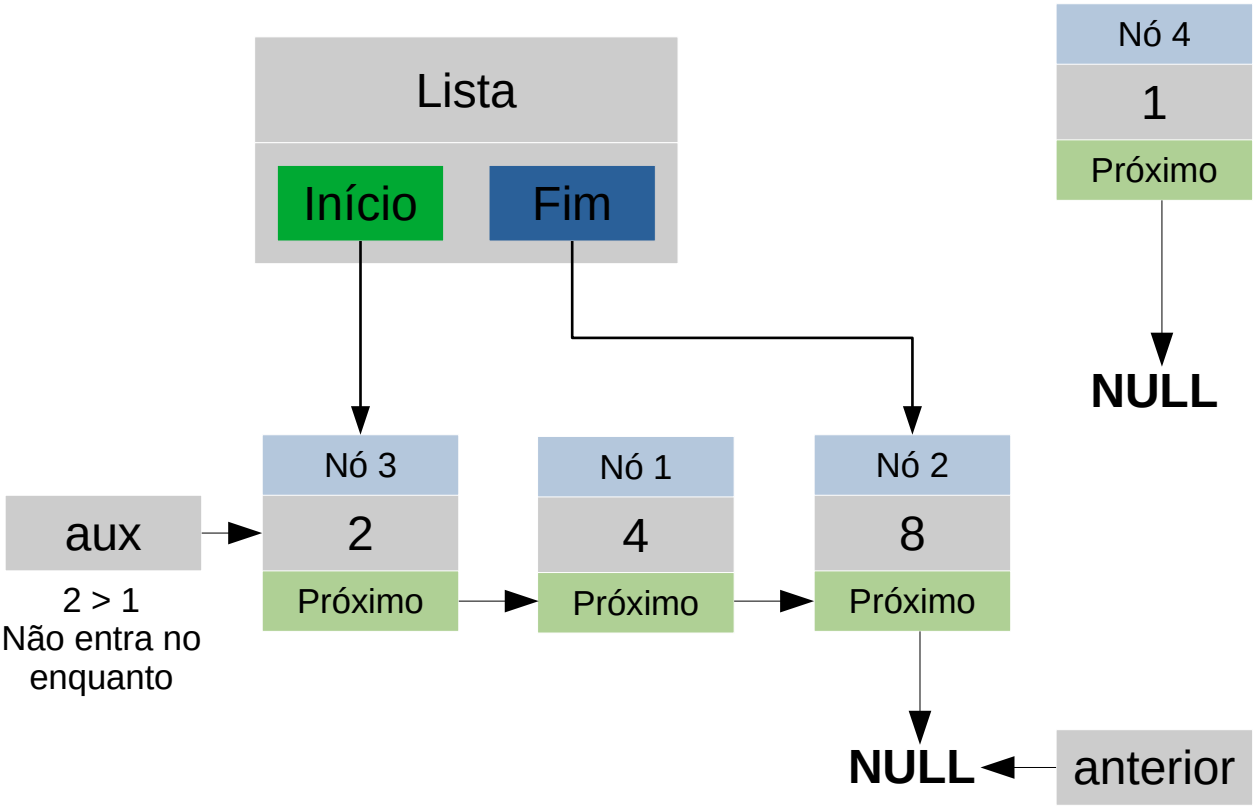
tamanho = 3



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

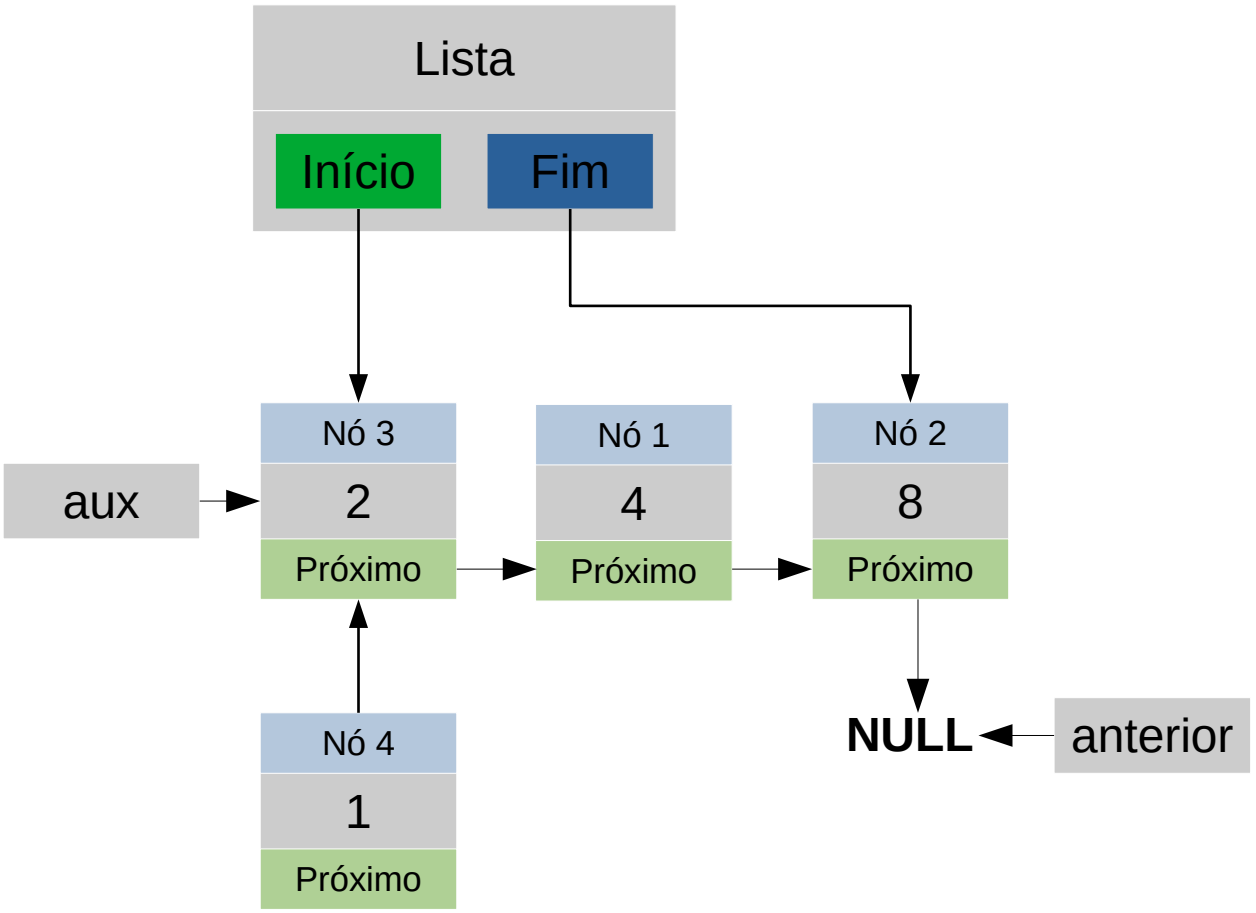
tamanho = 3



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

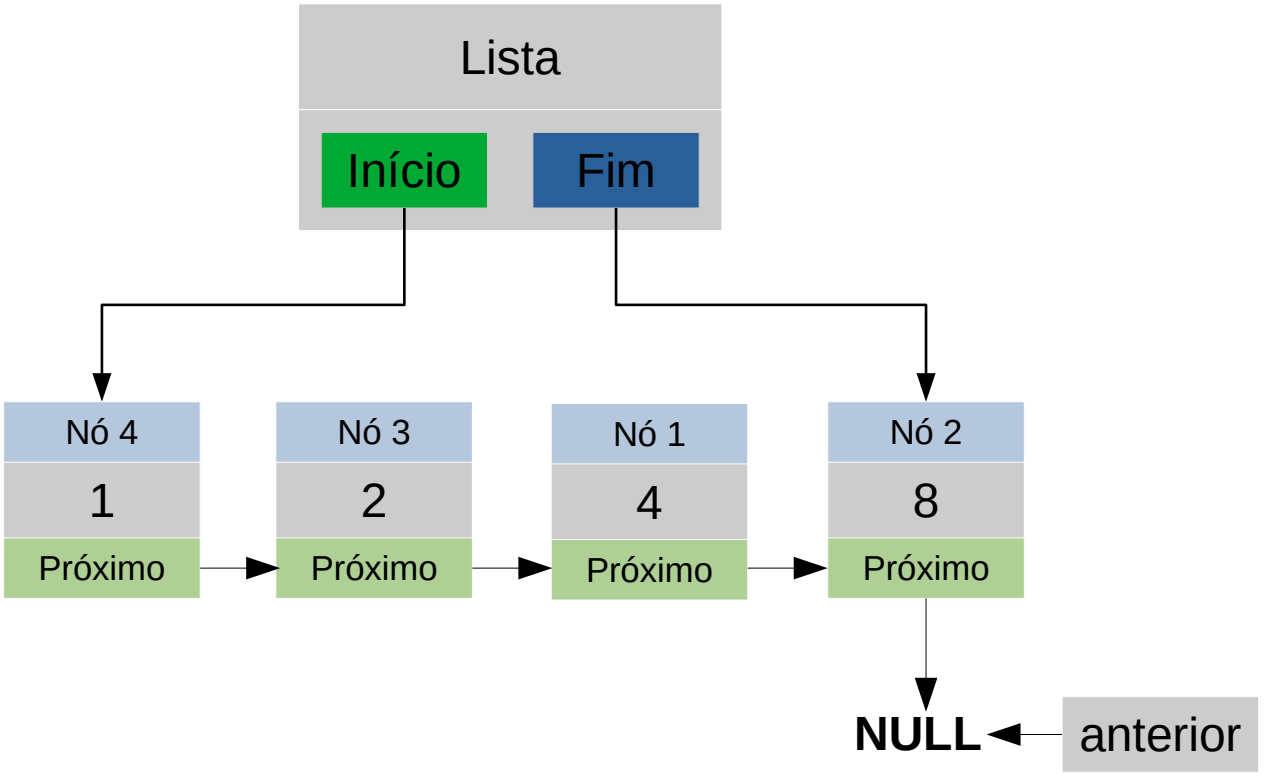
tamanho = 3



Passo 3. Inserir ordenado o elemento 4, 8, 2, 1

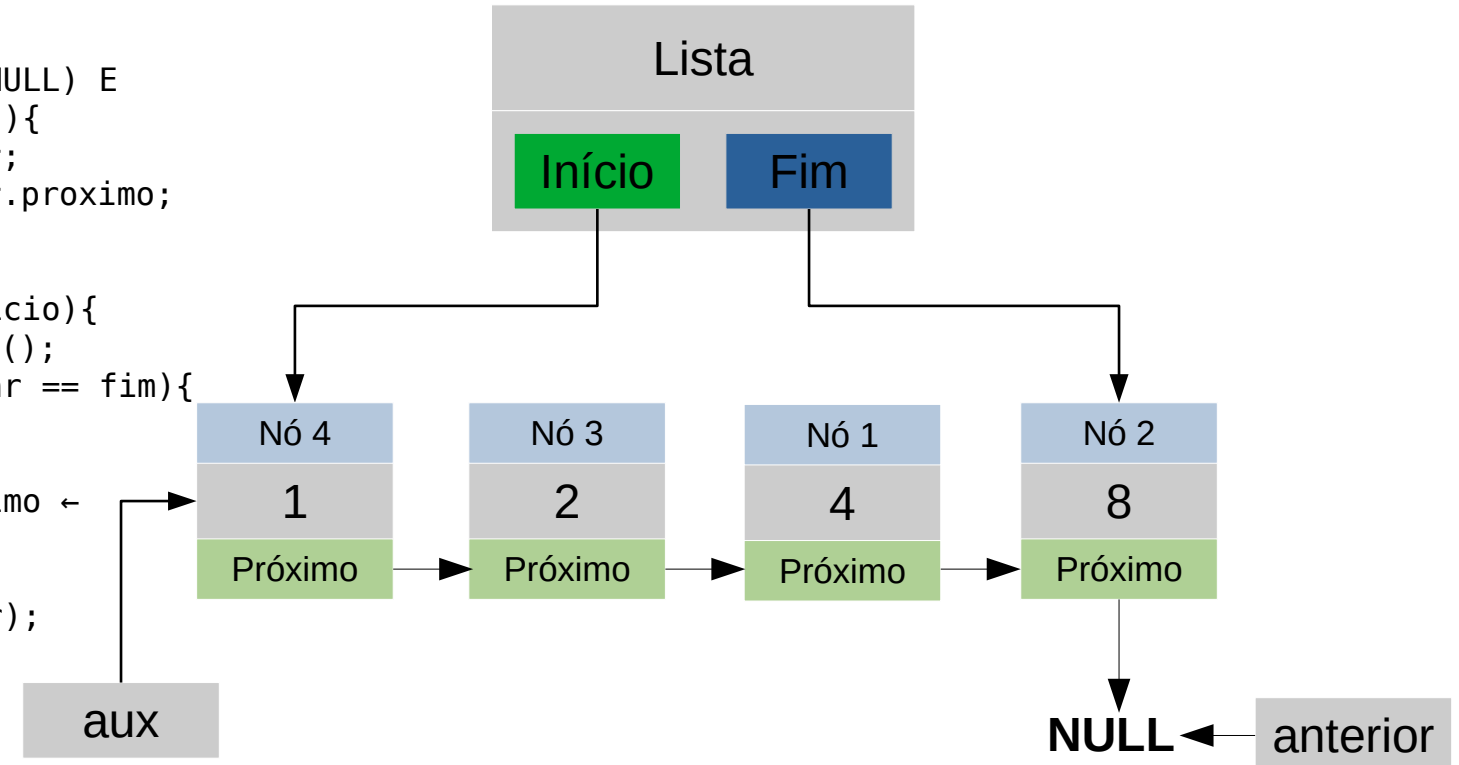
```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

tamanho = 4



Passo 4. Remover elemento 1

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```

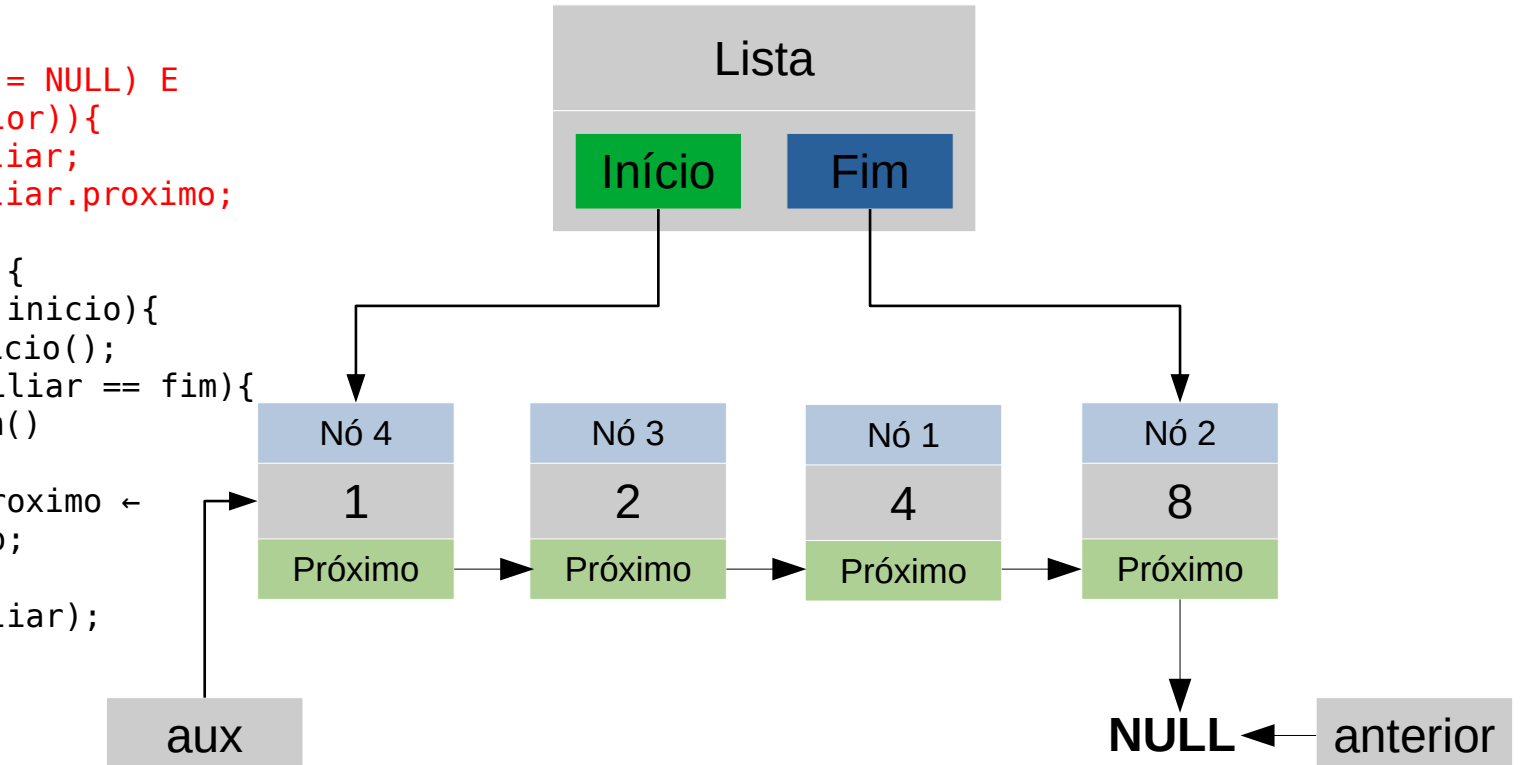


tamanho = 4

Passo 4. Remover elemento 1

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```

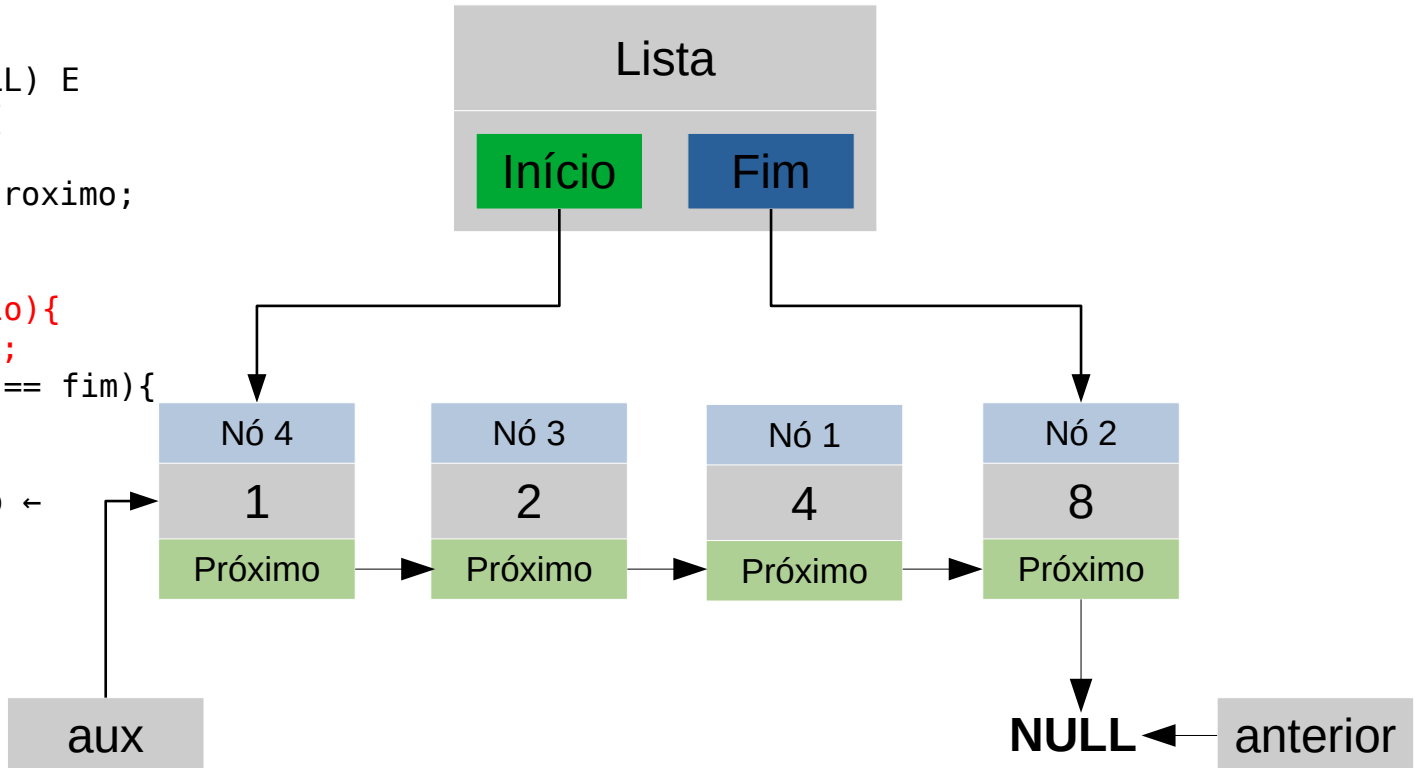
tamanho = 4



Não entra no enquanto
Porque aux.dado = valor

Passo 4. Remover elemento 1

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```



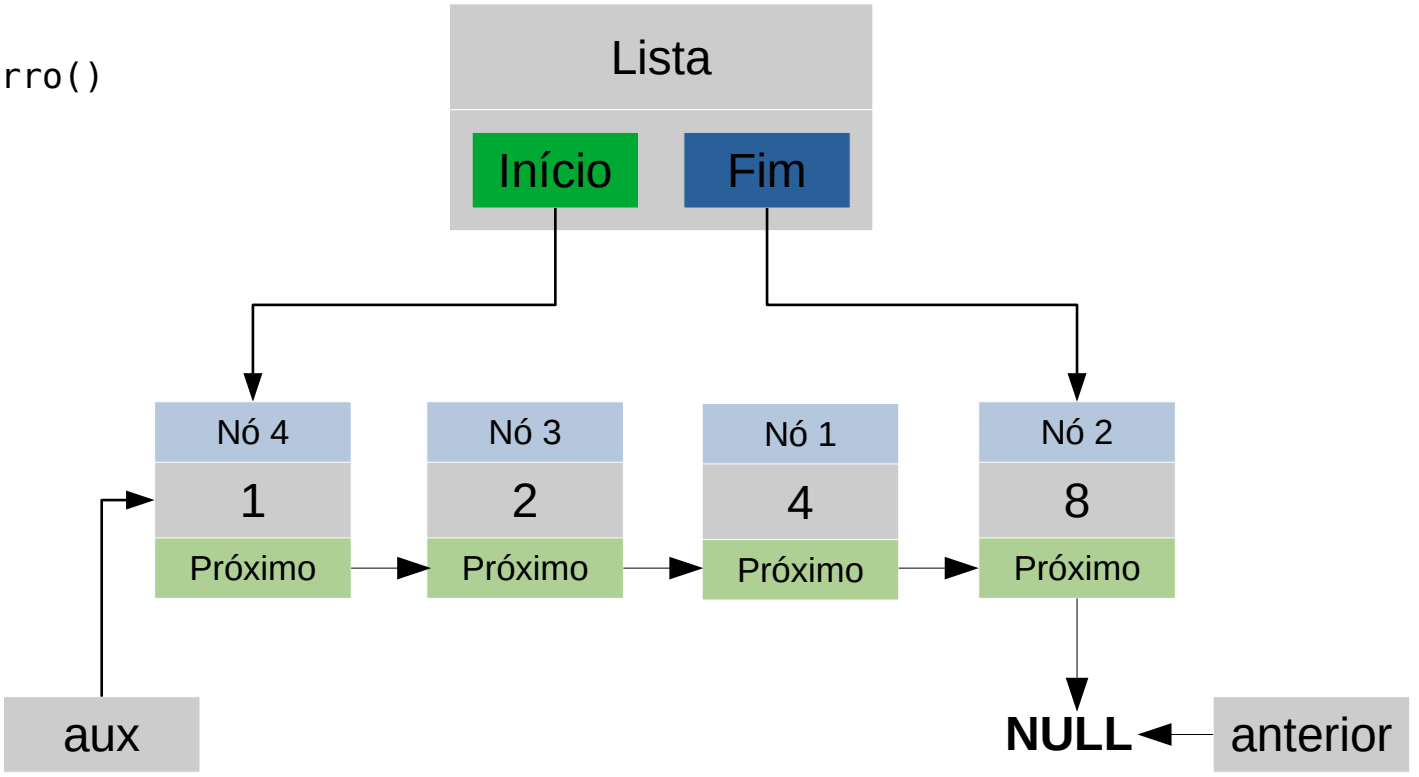
tamanho = 4

Chama função removeNoInicio();

Passo 4. Remover elemento 1

removeNoInicio():

```
se listaVazia() sairComErro()
aux ← inicio;
valor ← aux.dado;
inicio ← aux.proximo;
apagar(aux);
tamanho--;
se listaVazia()
     fim ← NULL;
```



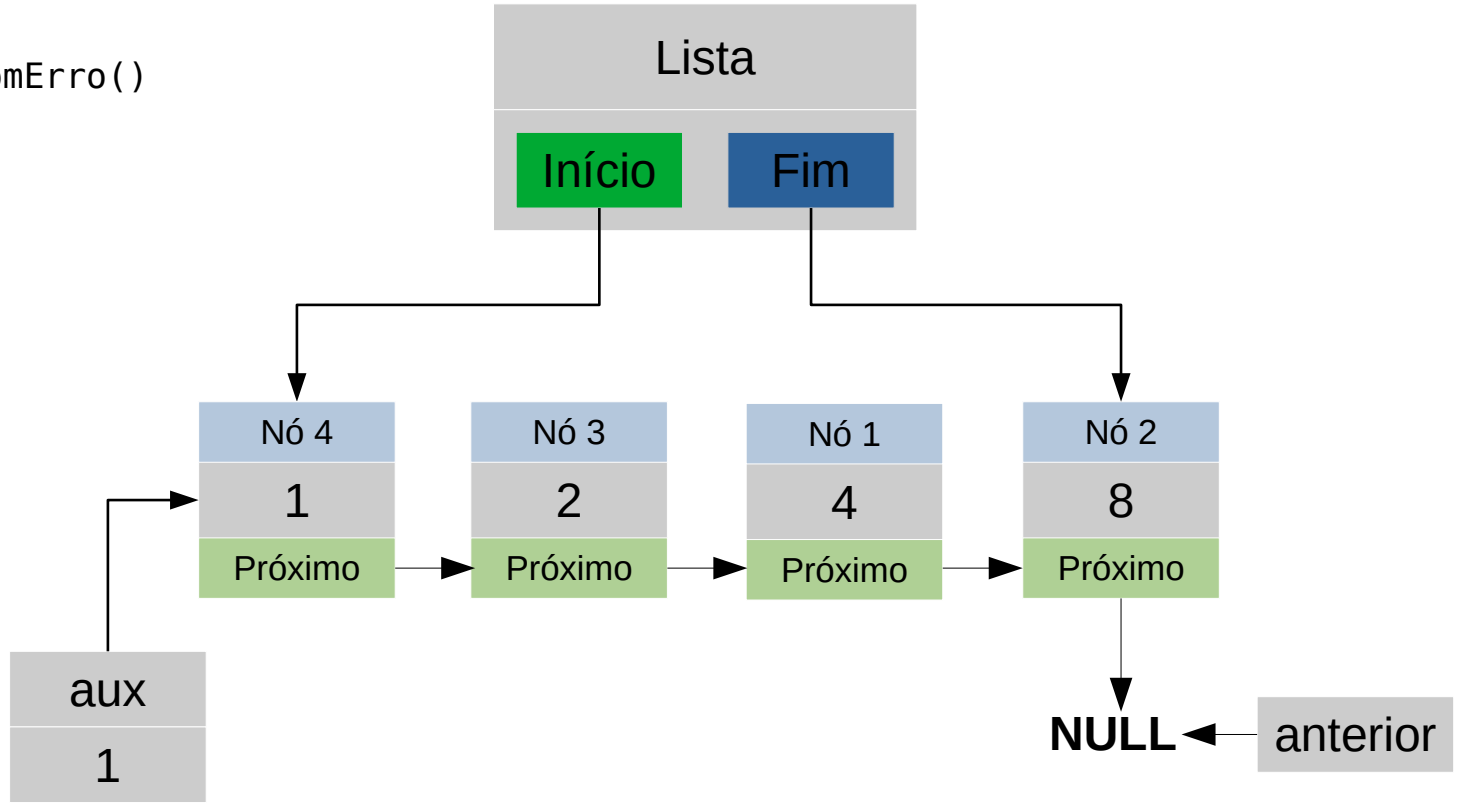
tamanho = 4

Passo 4. Remover elemento 1

removeNoInicio():

```
se listaVazia() sairComErro()
aux ← inicio;
valor ← aux.dado;
inicio ← aux.proximo;
apagar(aux);
tamanho--;
se listaVazia()
     fim ← NULL;
```

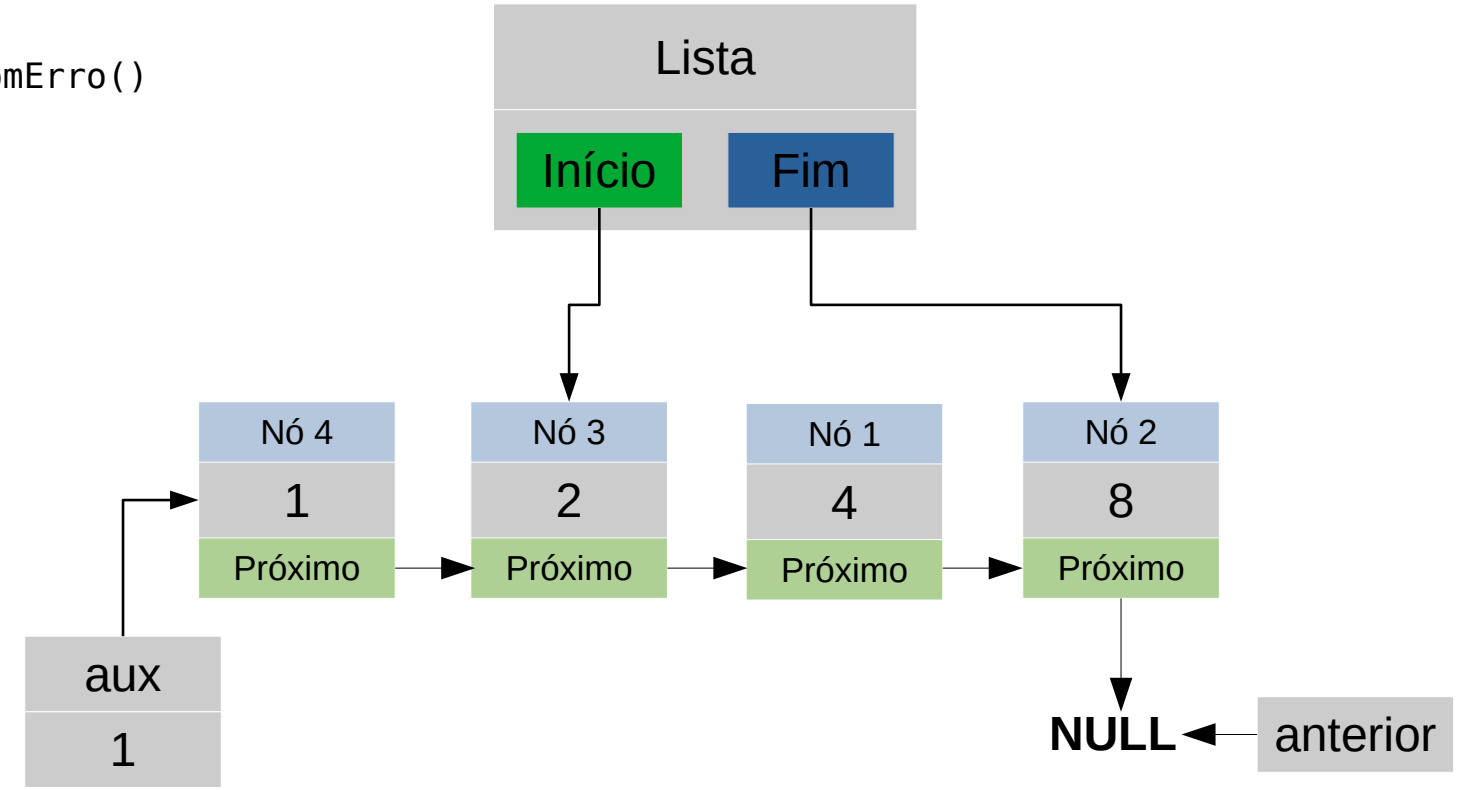
tamanho = 4



Passo 4. Remover elemento 1

removeNoInicio():

```
se listaVazia() sairComErro()
aux ← inicio;
valor ← aux.dado;
inicio ← aux.proximo;
apagar(aux);
tamanho--;
se listaVazia()
     fim ← NULL;
```



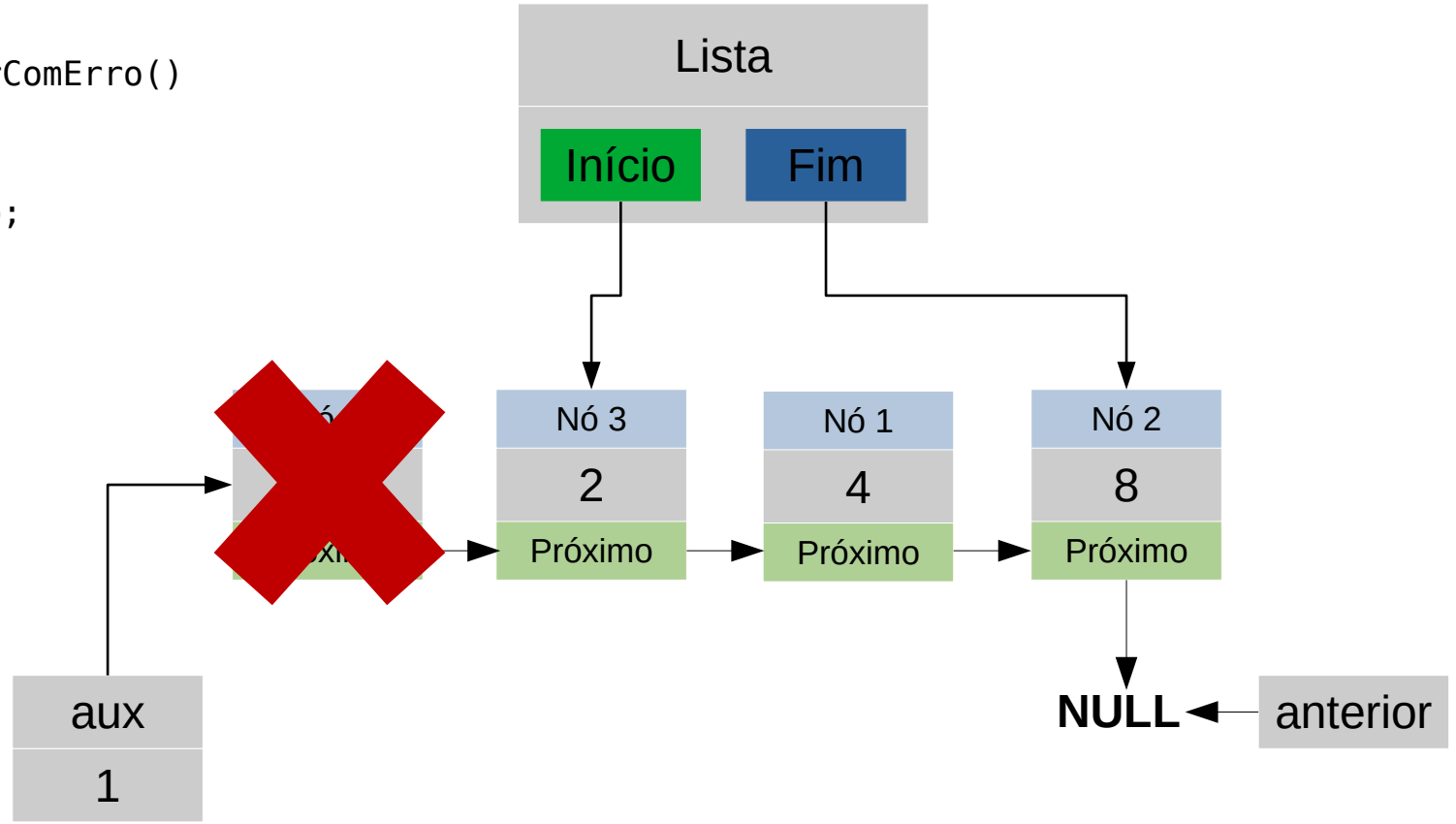
tamanho = 4

Passo 4. Remover elemento 1

removeNoInicio():

```
se listaVazia() sairComErro()
aux ← inicio;
valor ← aux.dado;
inicio ← aux.proximo;
apagar(aux);
tamanho--;
se listaVazia()
     fim ← NULL;
```

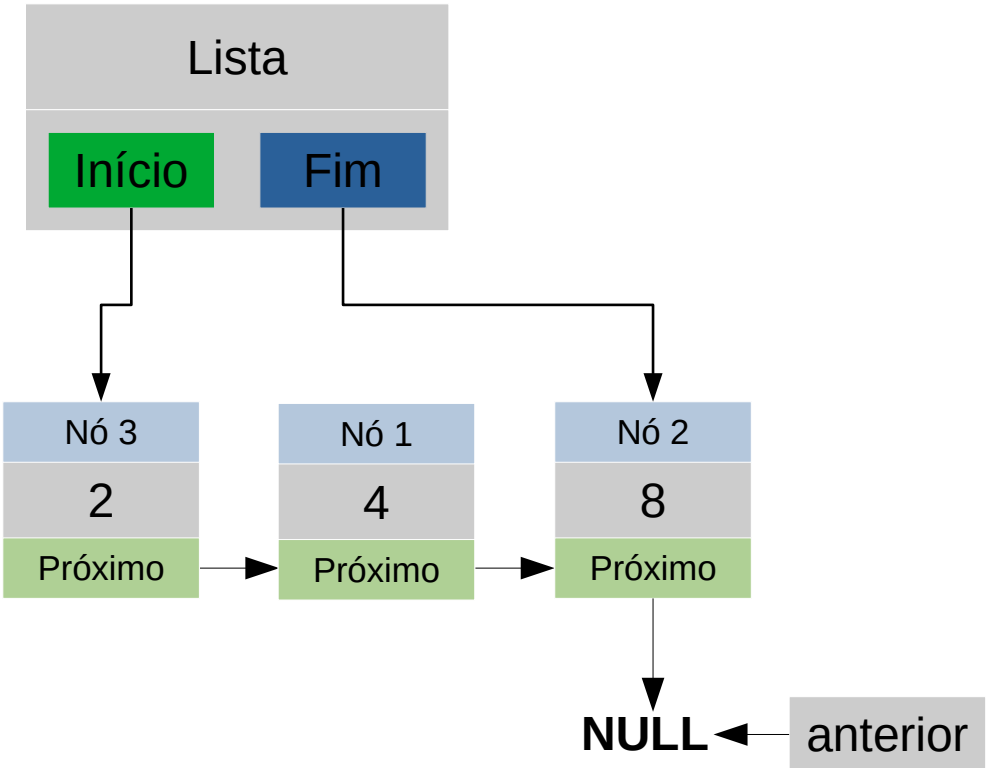
tamanho = 4



Passo 4. Remover elemento 1

removeNoInicio():

```
se listaVazia() sairComErro()
aux ← inicio;
valor ← aux.dado;
inicio ← aux.proximo;
apagar(aux);
tamanho--;
se listaVazia()
     fim ← NULL;
```

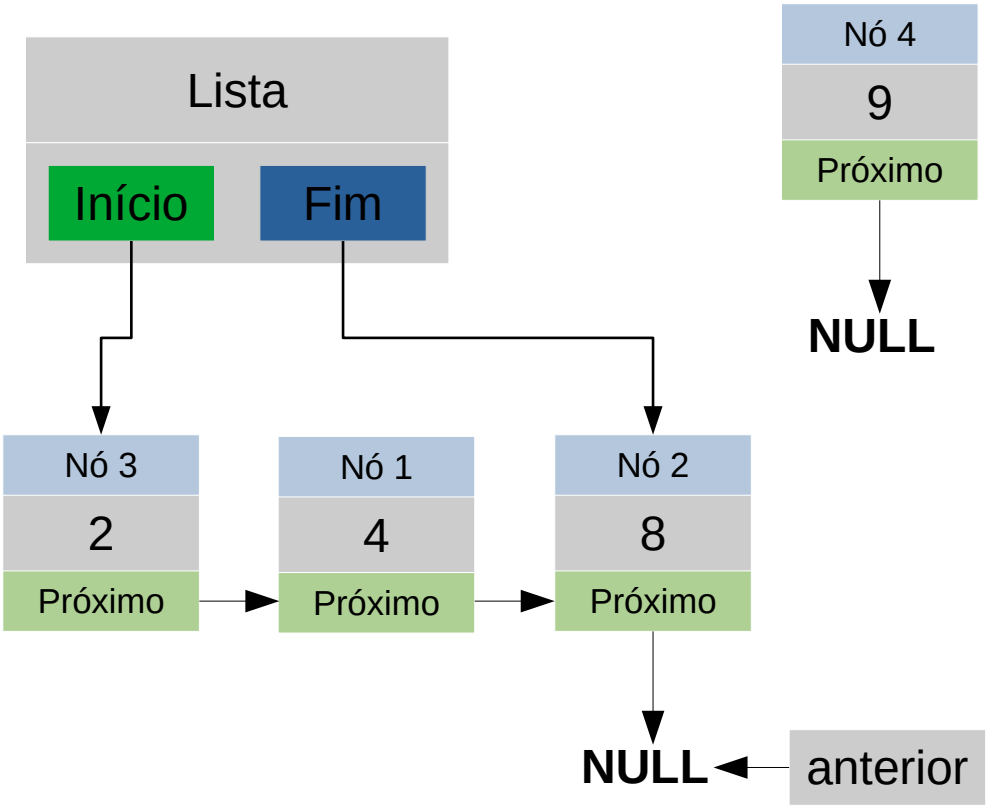


tamanho = 3

Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

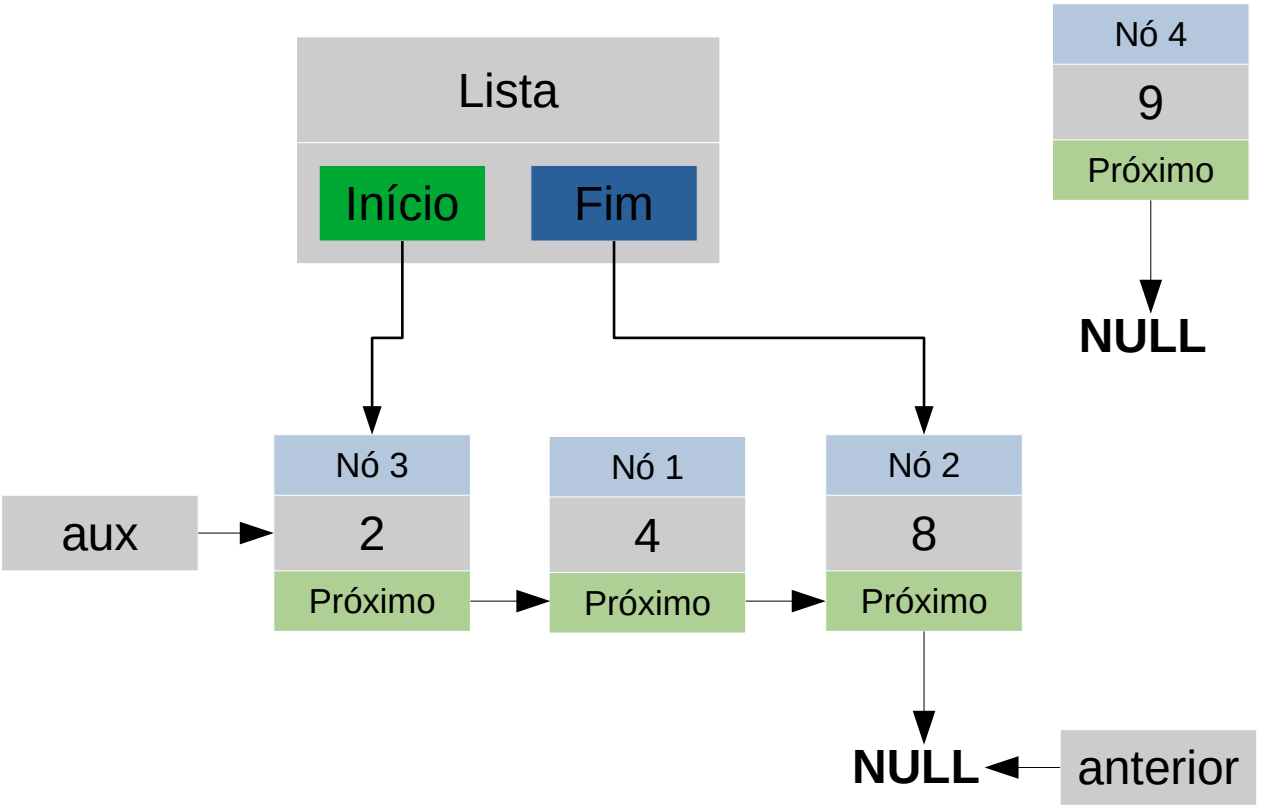
tamanho = 3



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

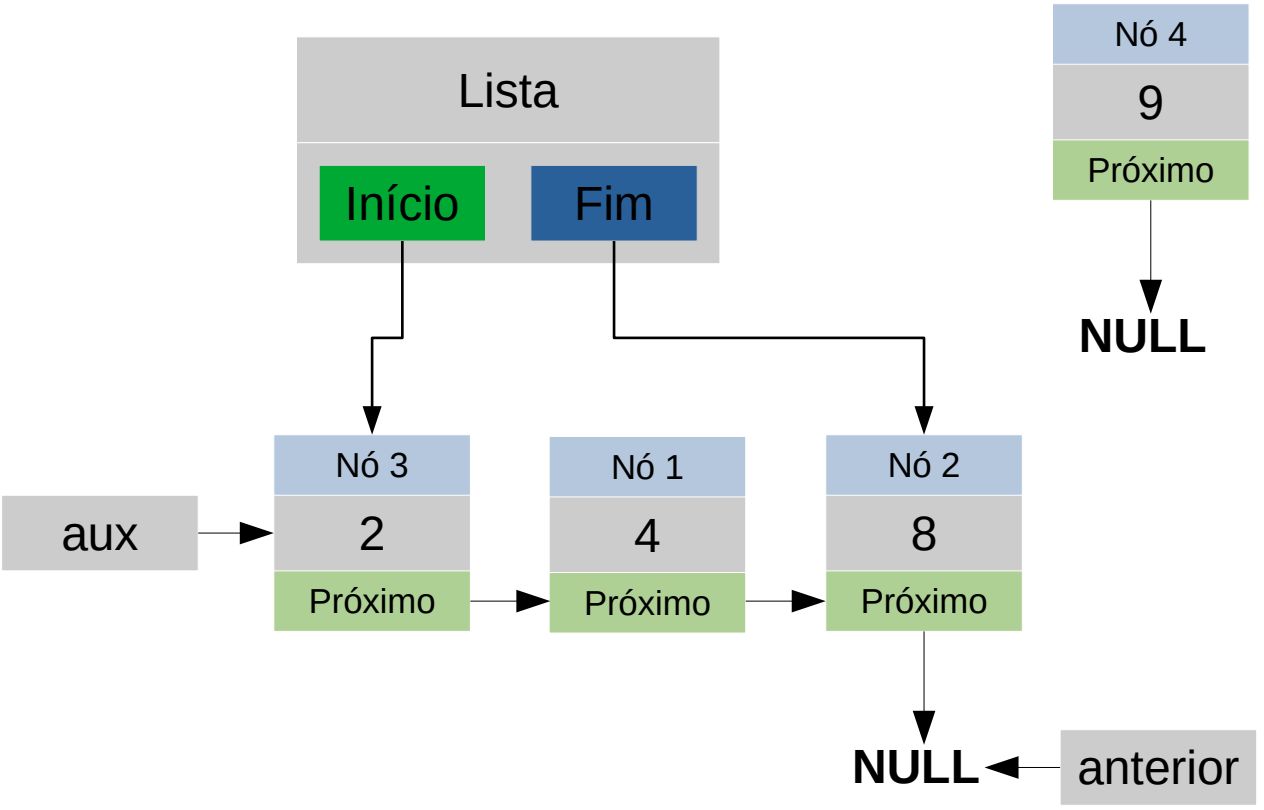
tamanho = 3



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

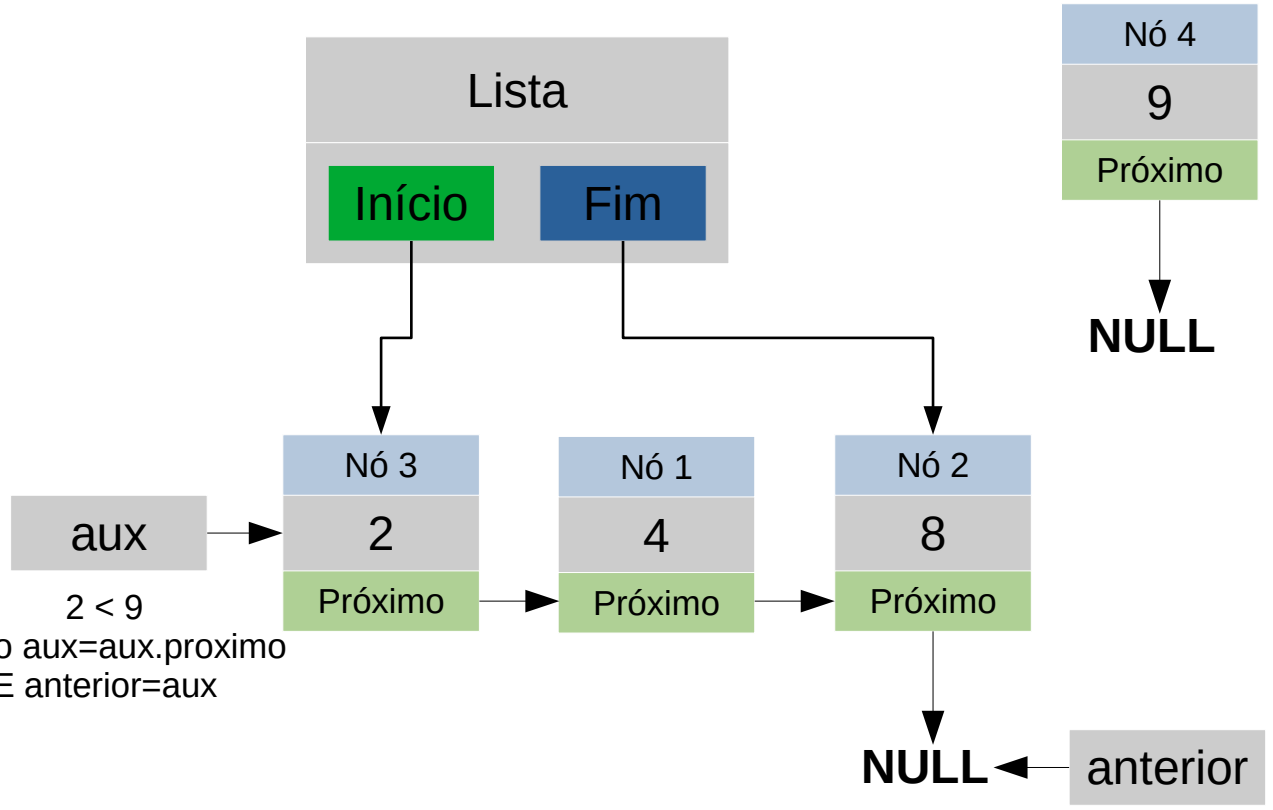
tamanho = 3



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

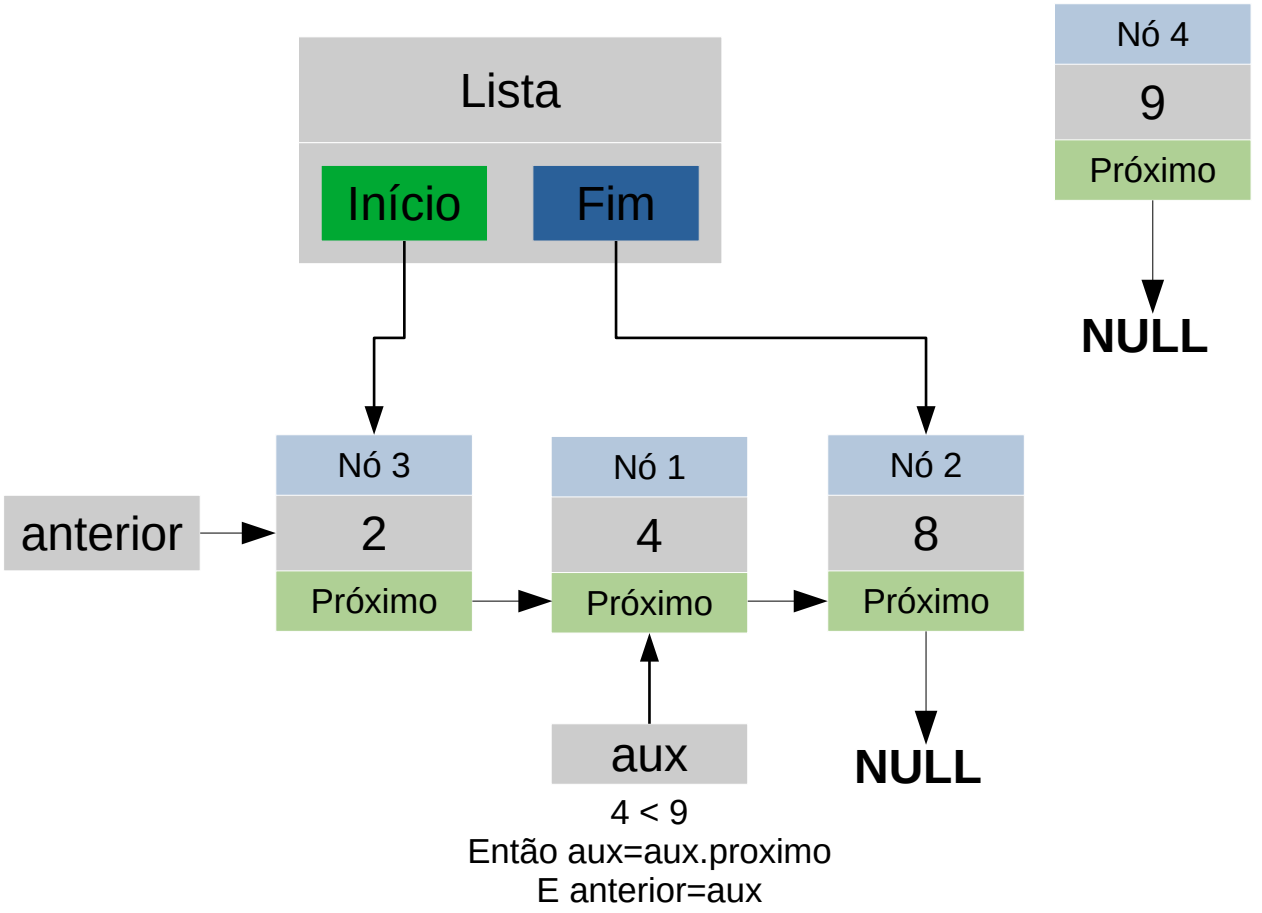
tamanho = 3



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

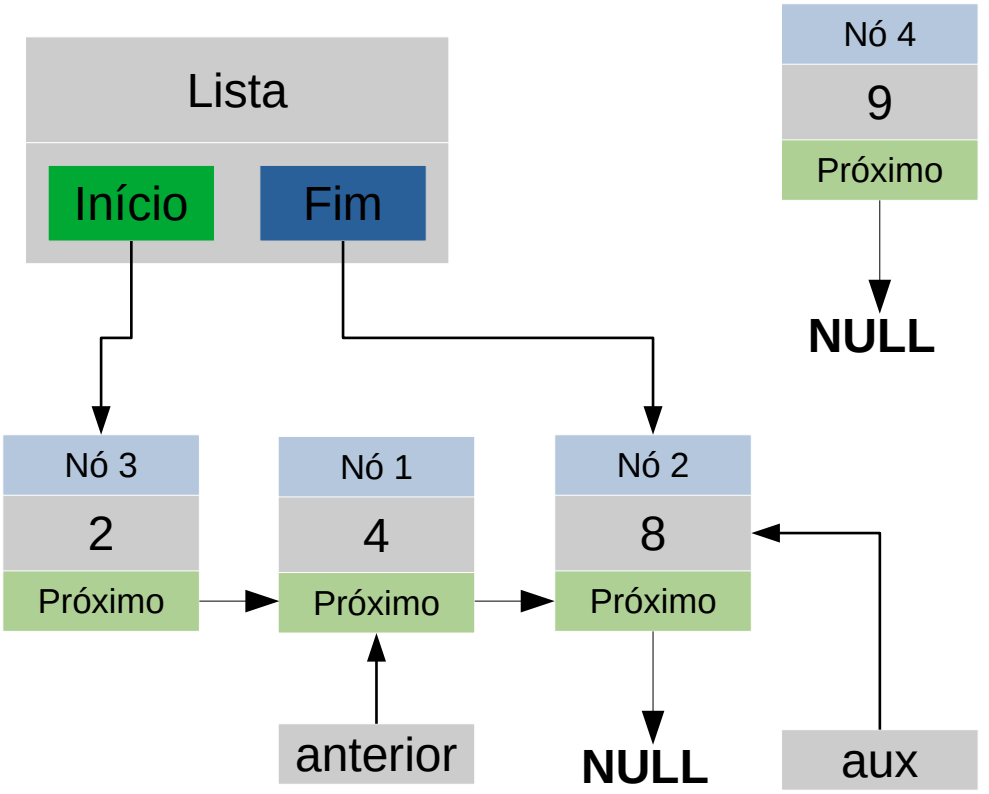
tamanho = 3



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

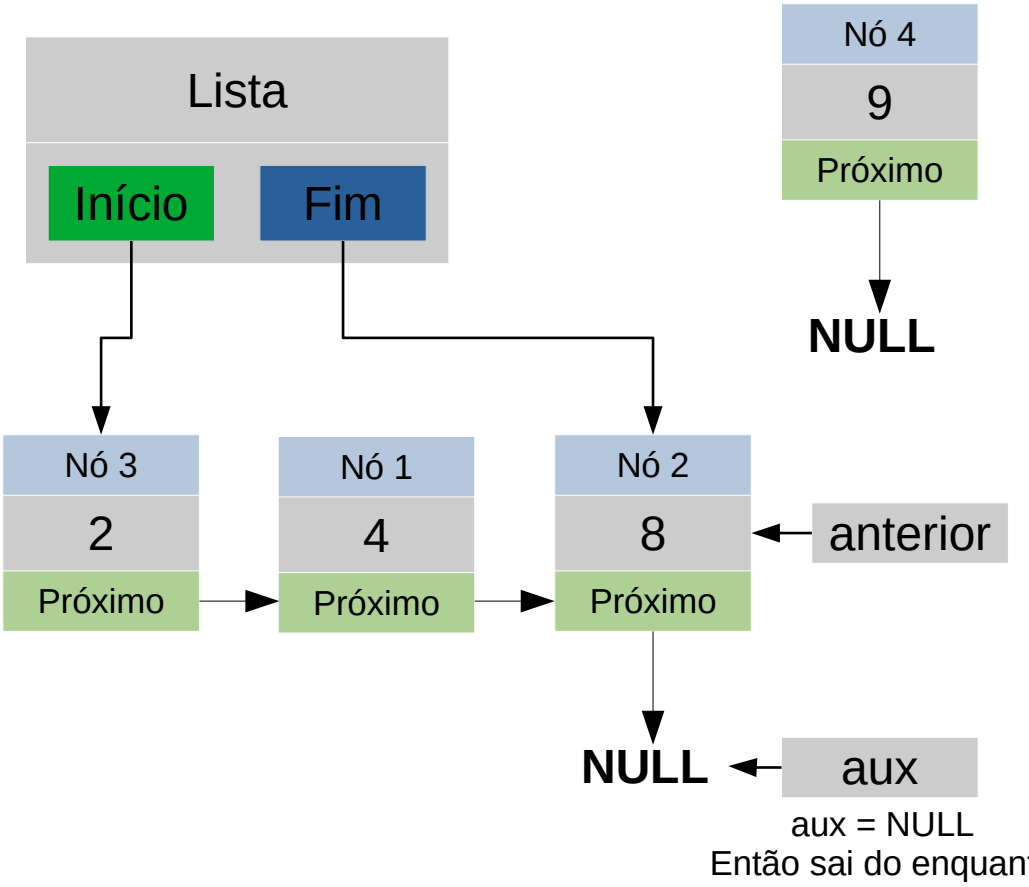
tamanho = 3



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

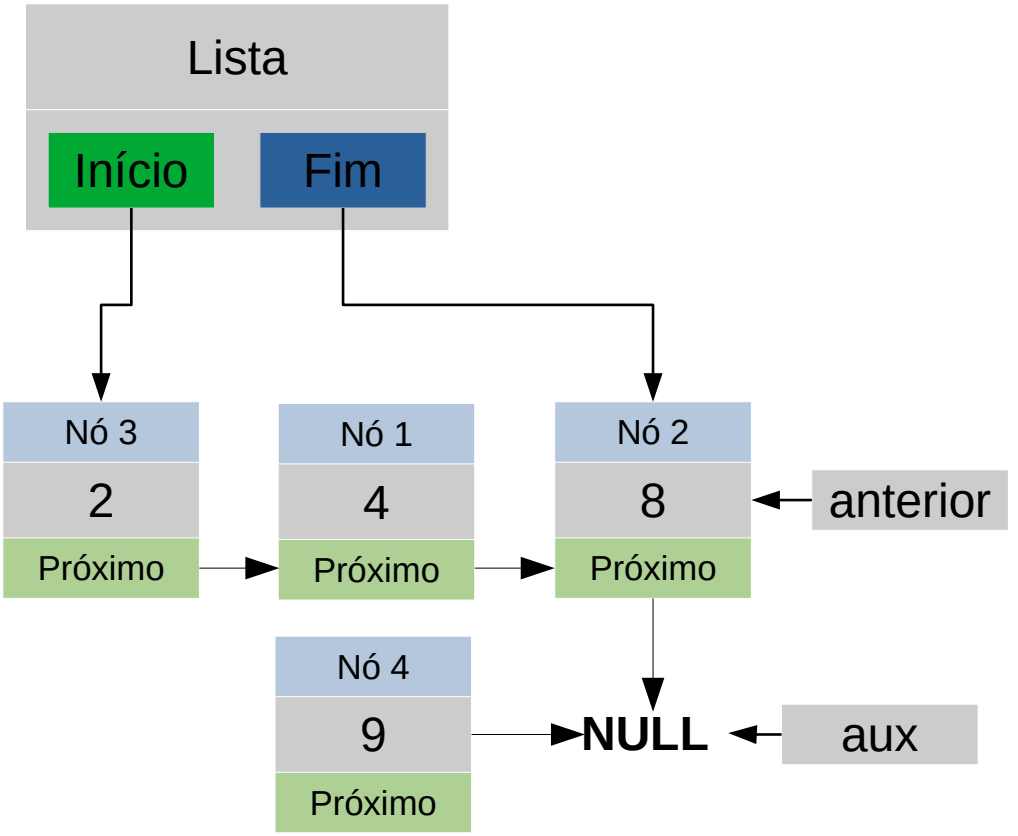
tamanho = 3



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

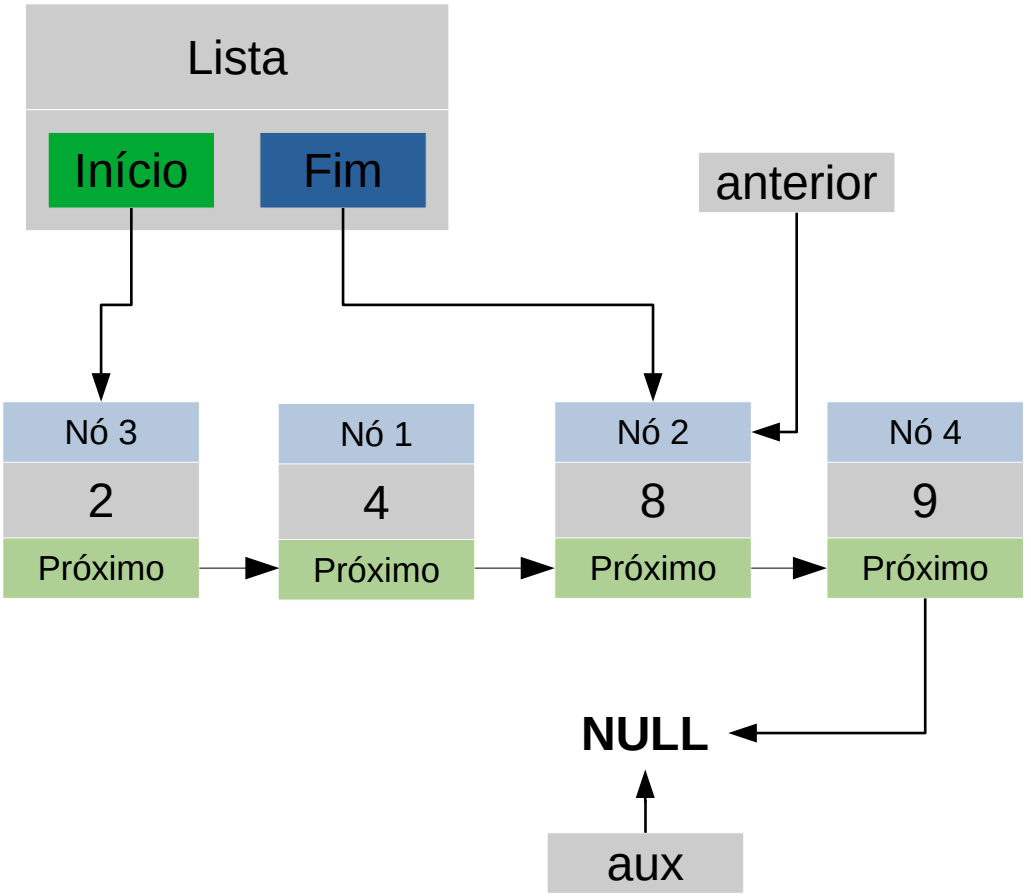
tamanho = 3



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

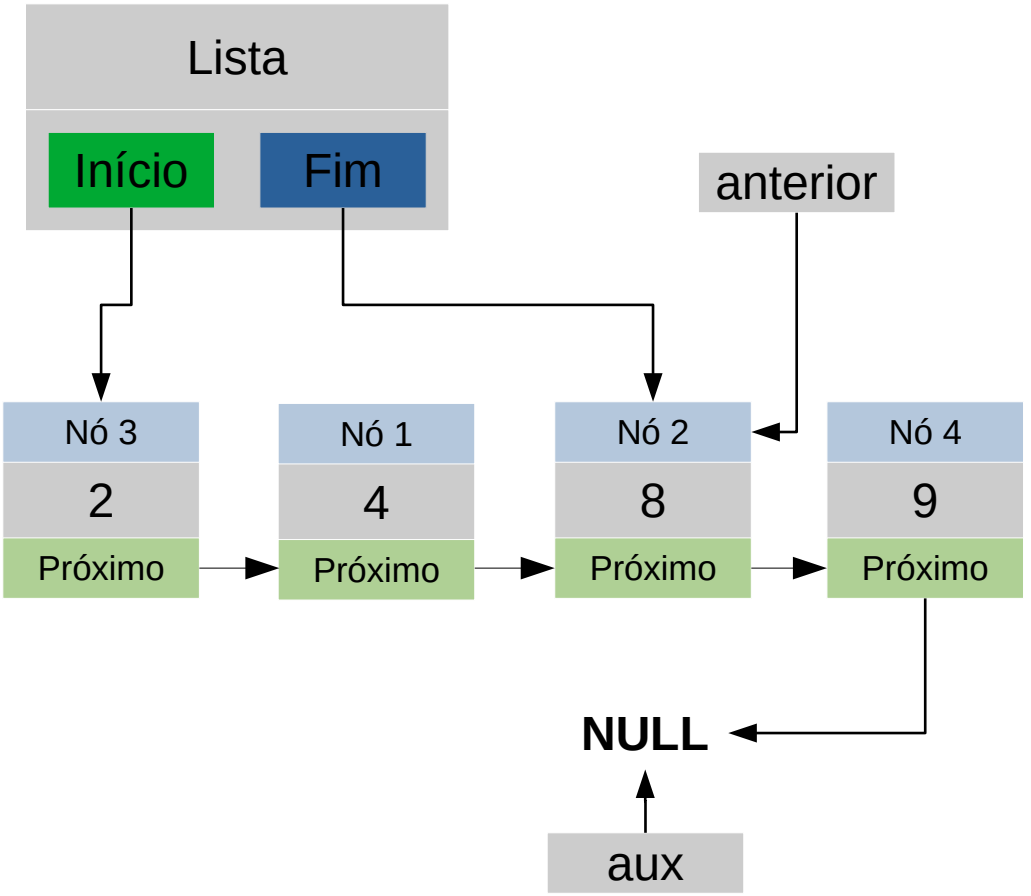
tamanho = 3



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

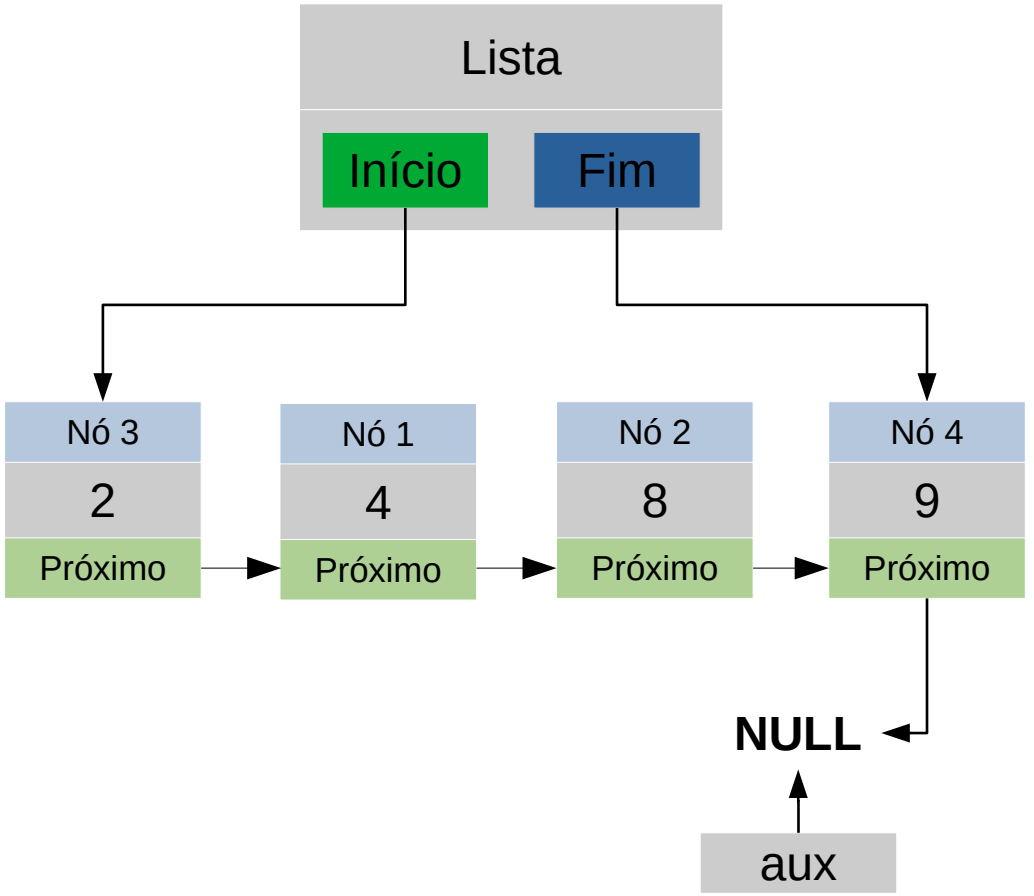
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

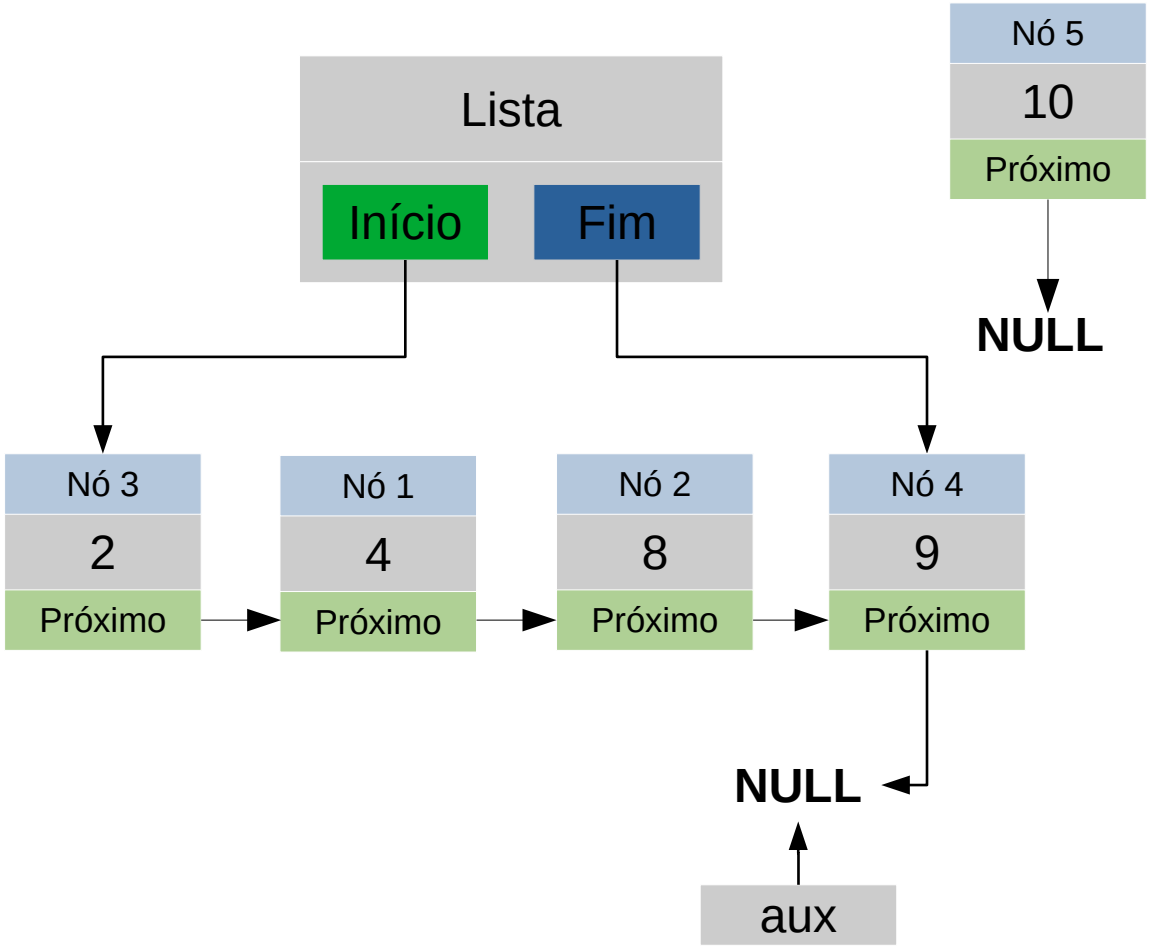
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

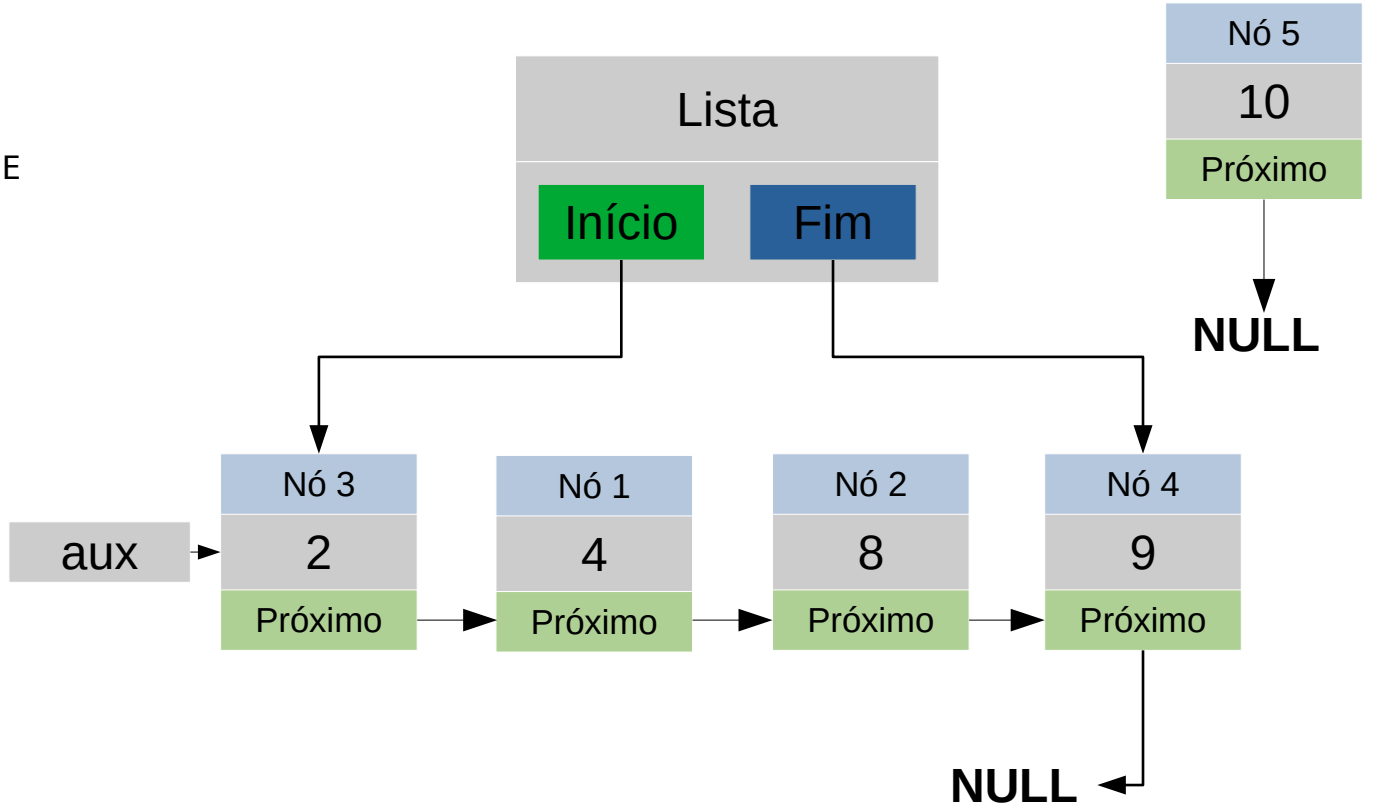
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

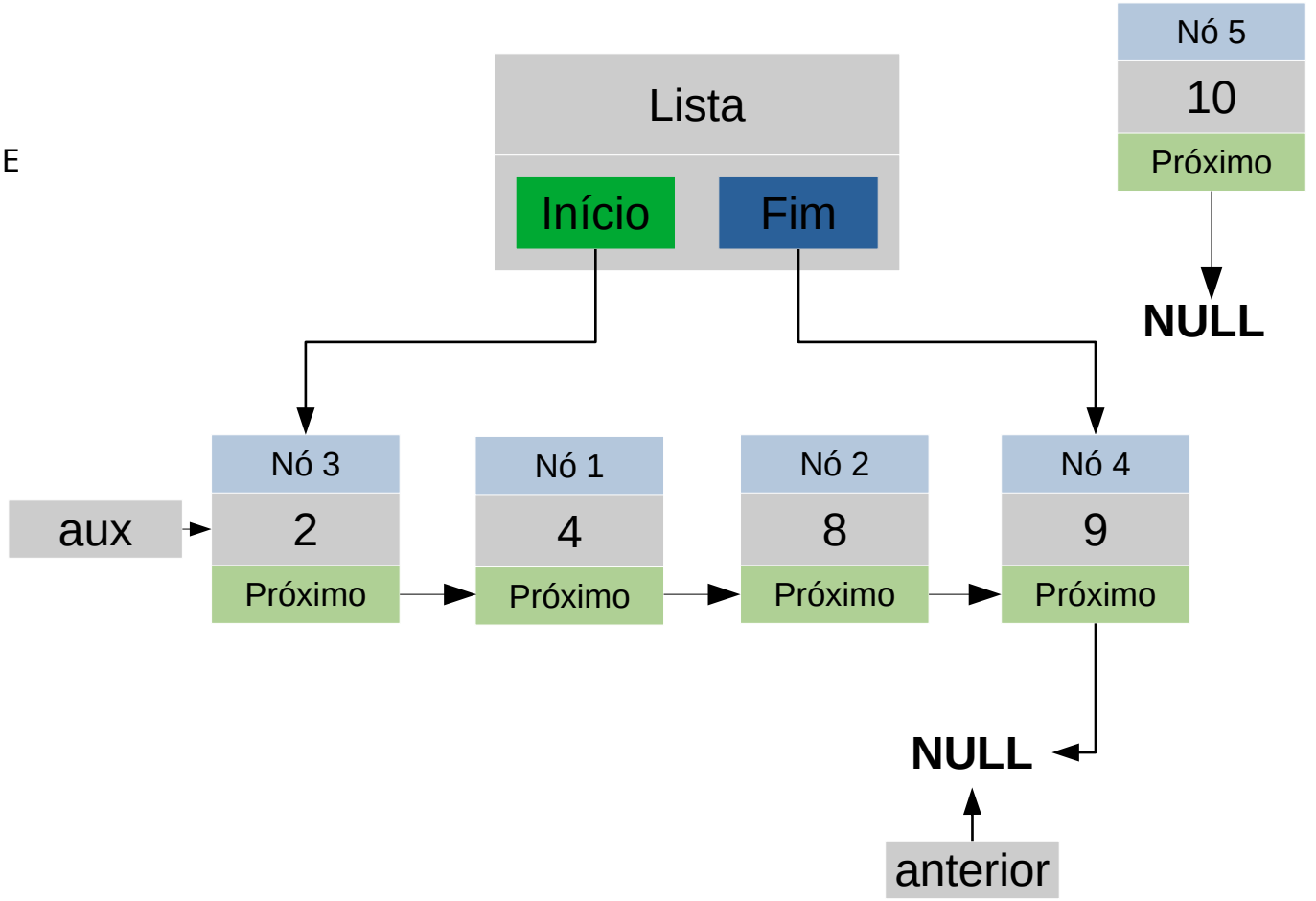
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

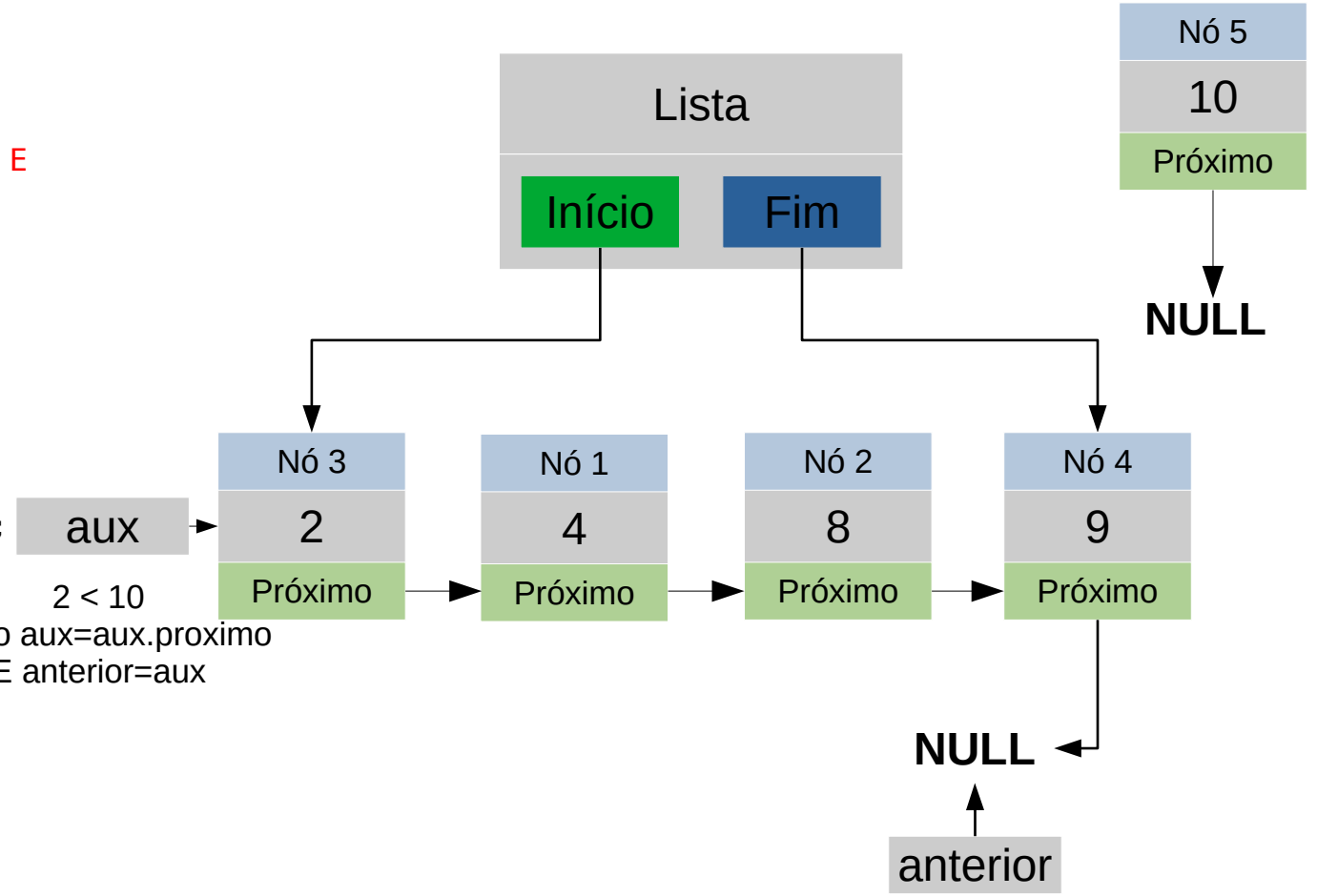
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

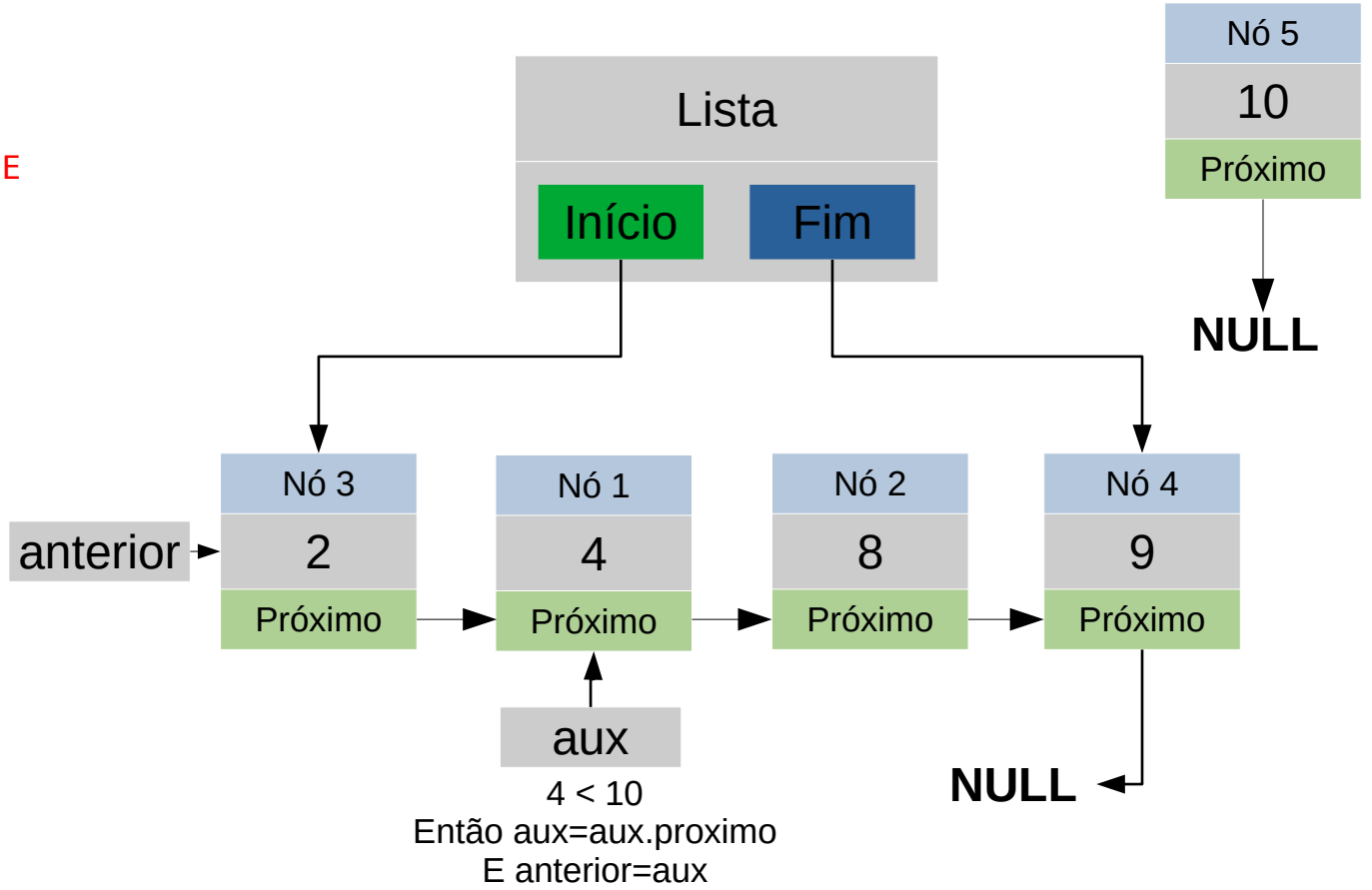
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

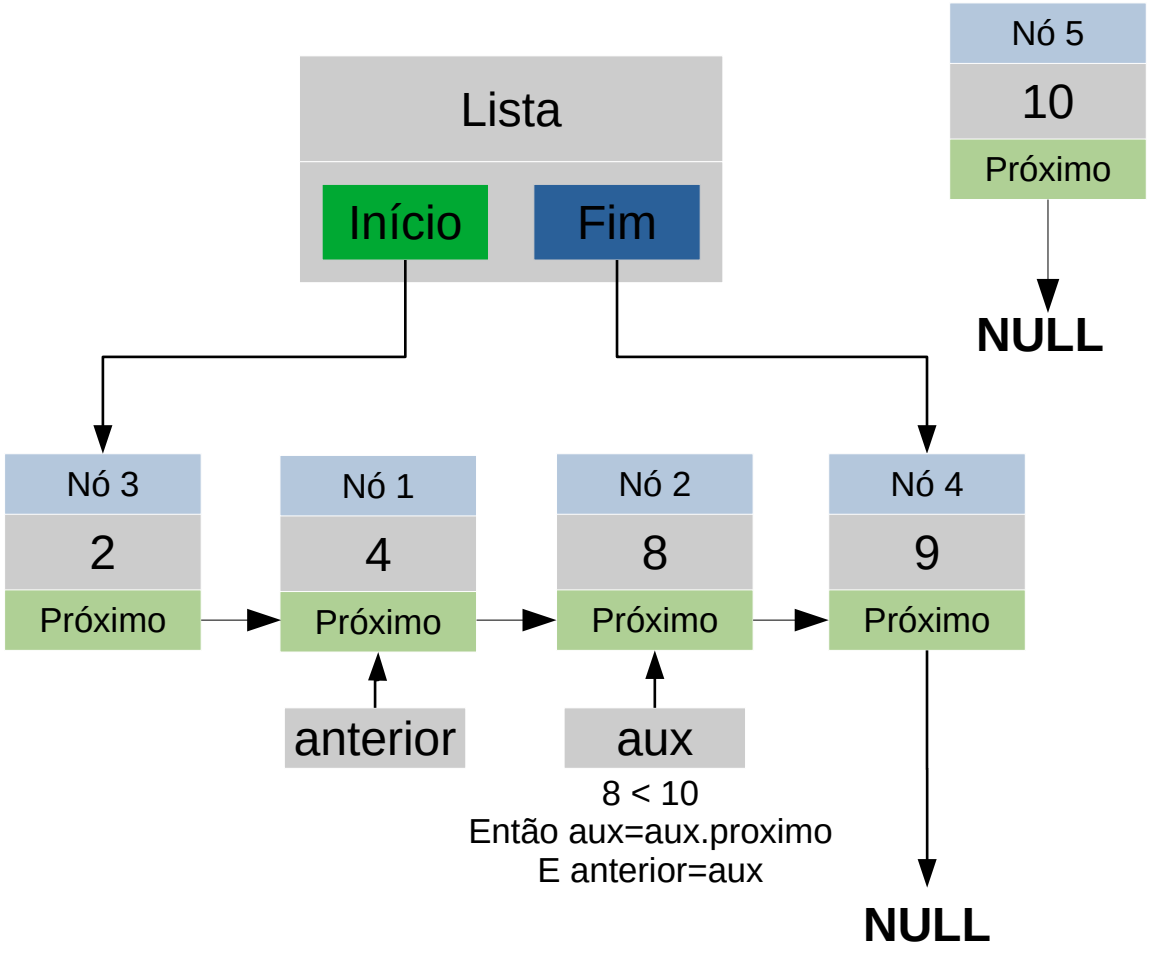
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

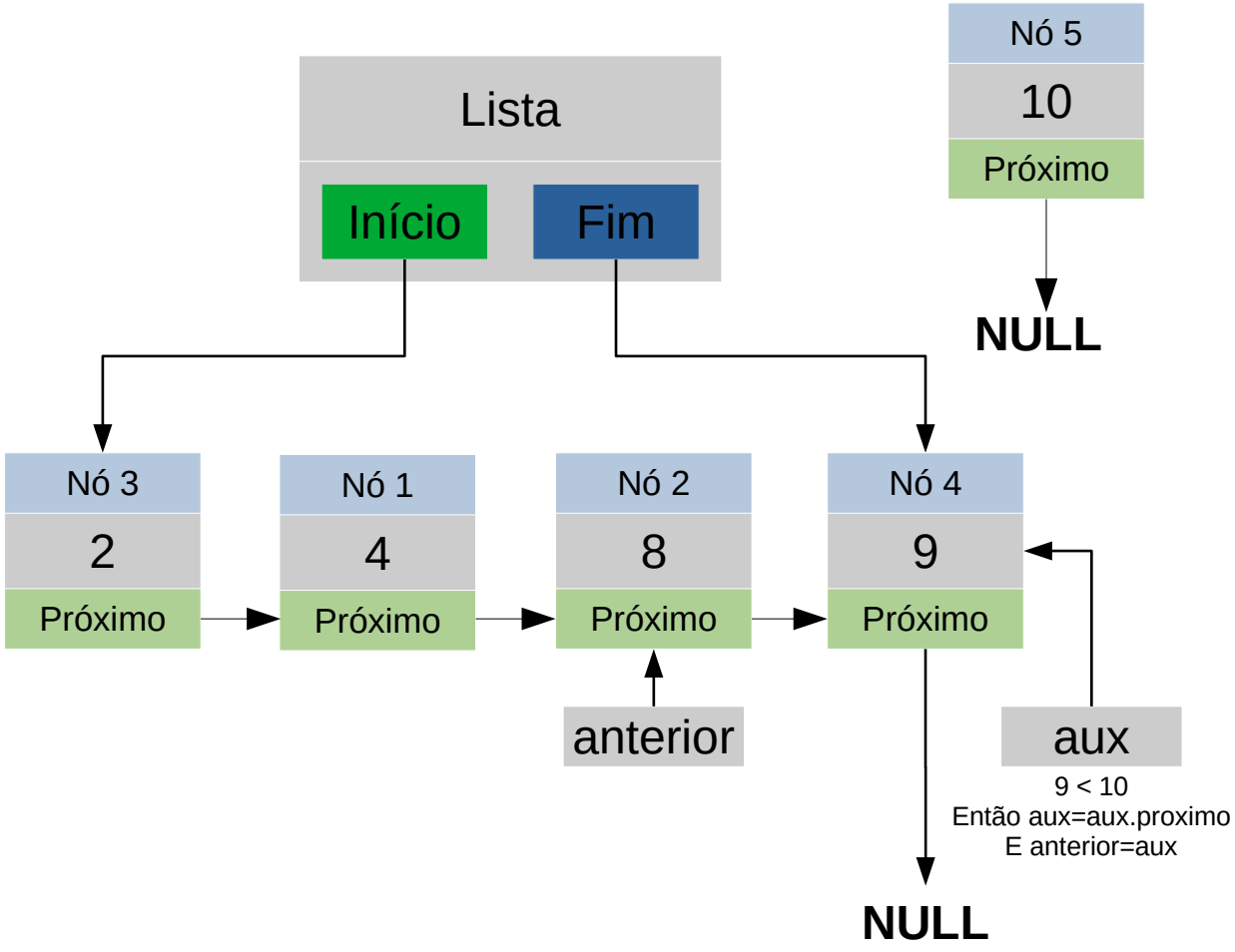
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

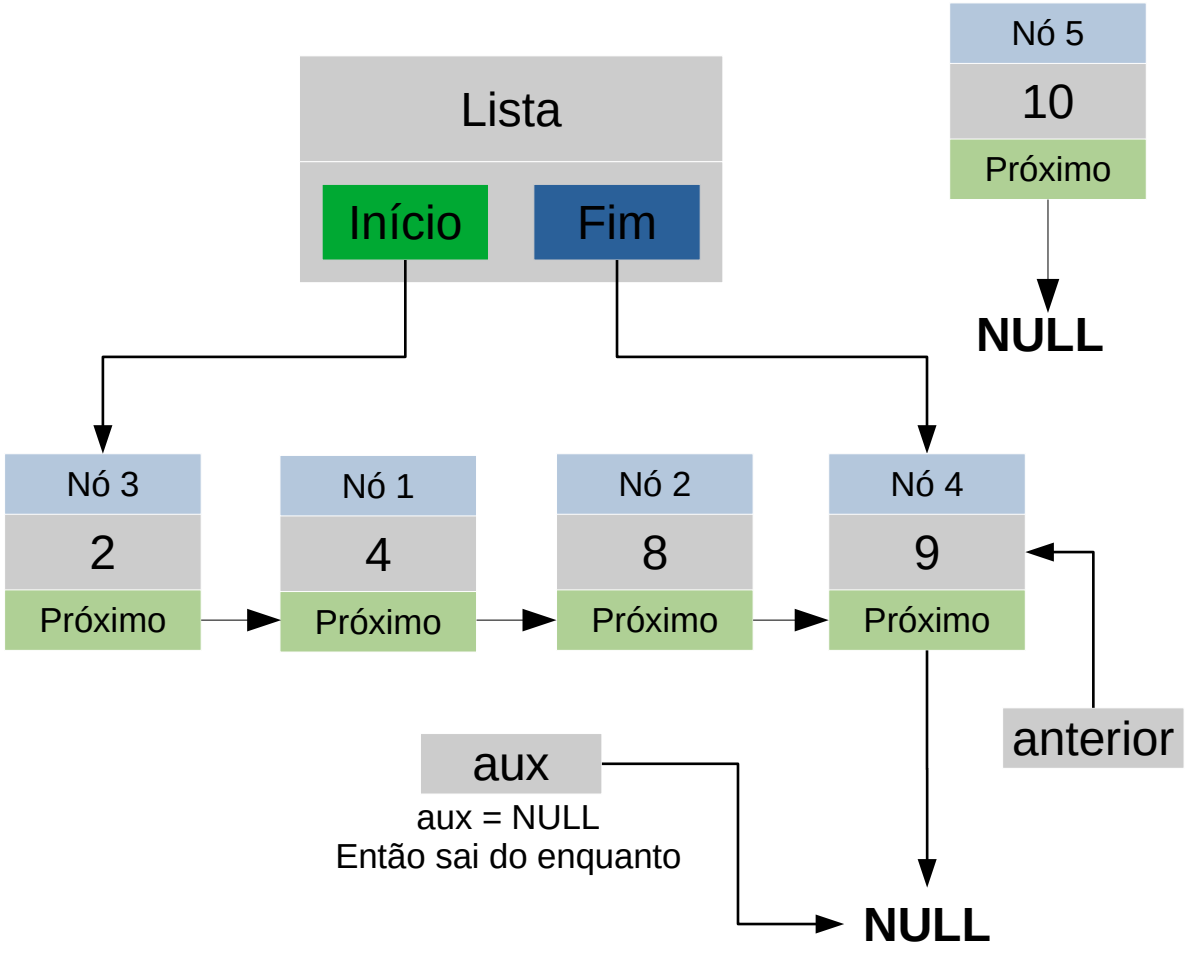
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

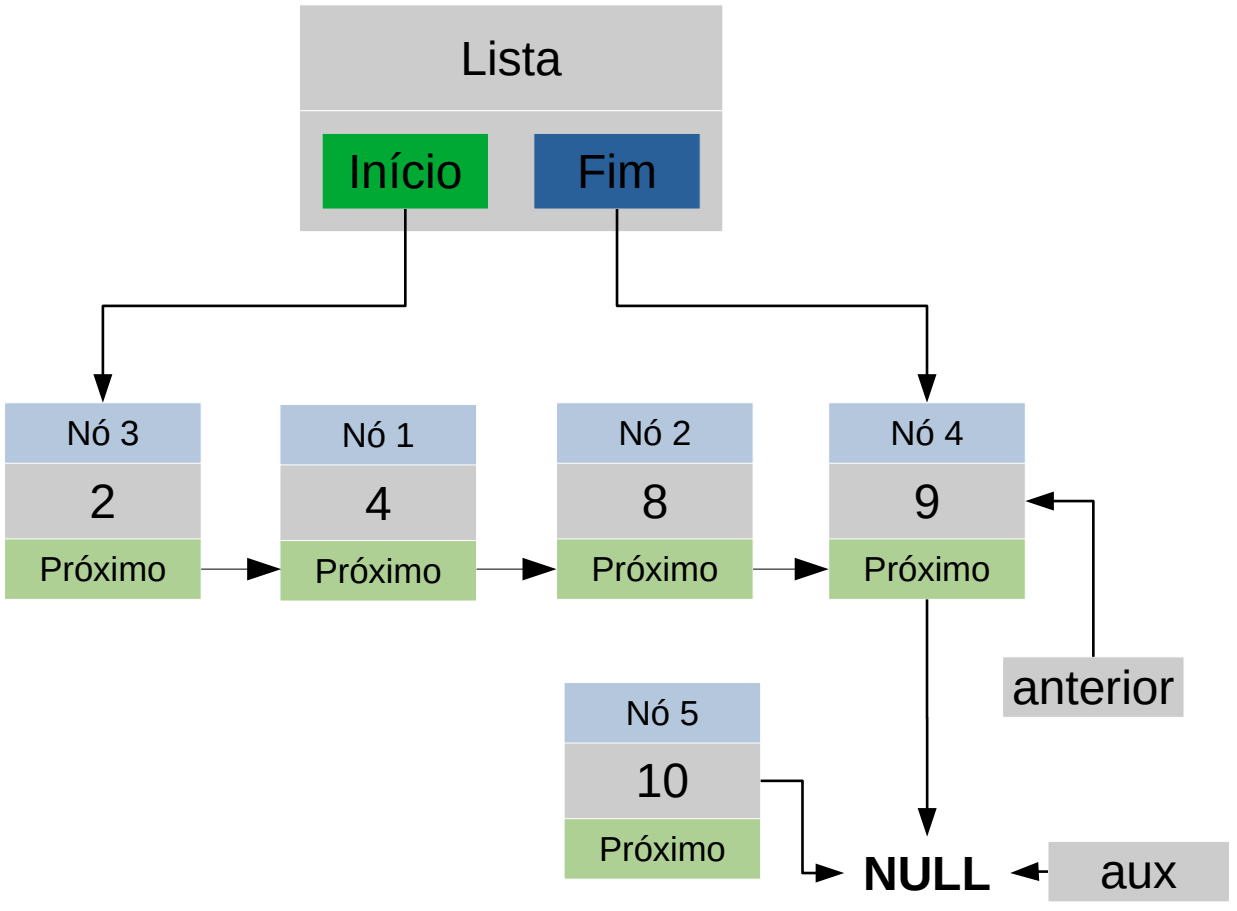
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

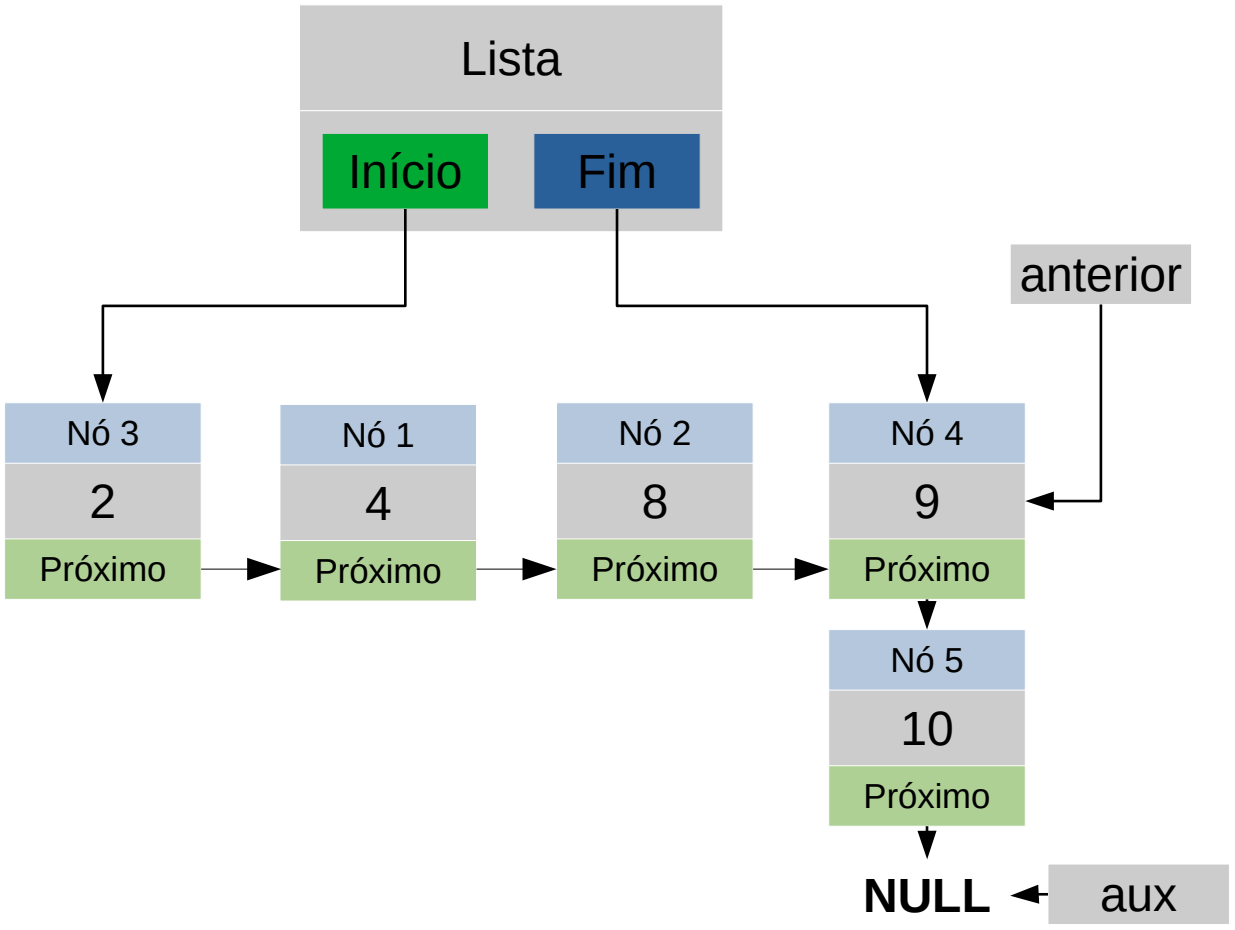
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

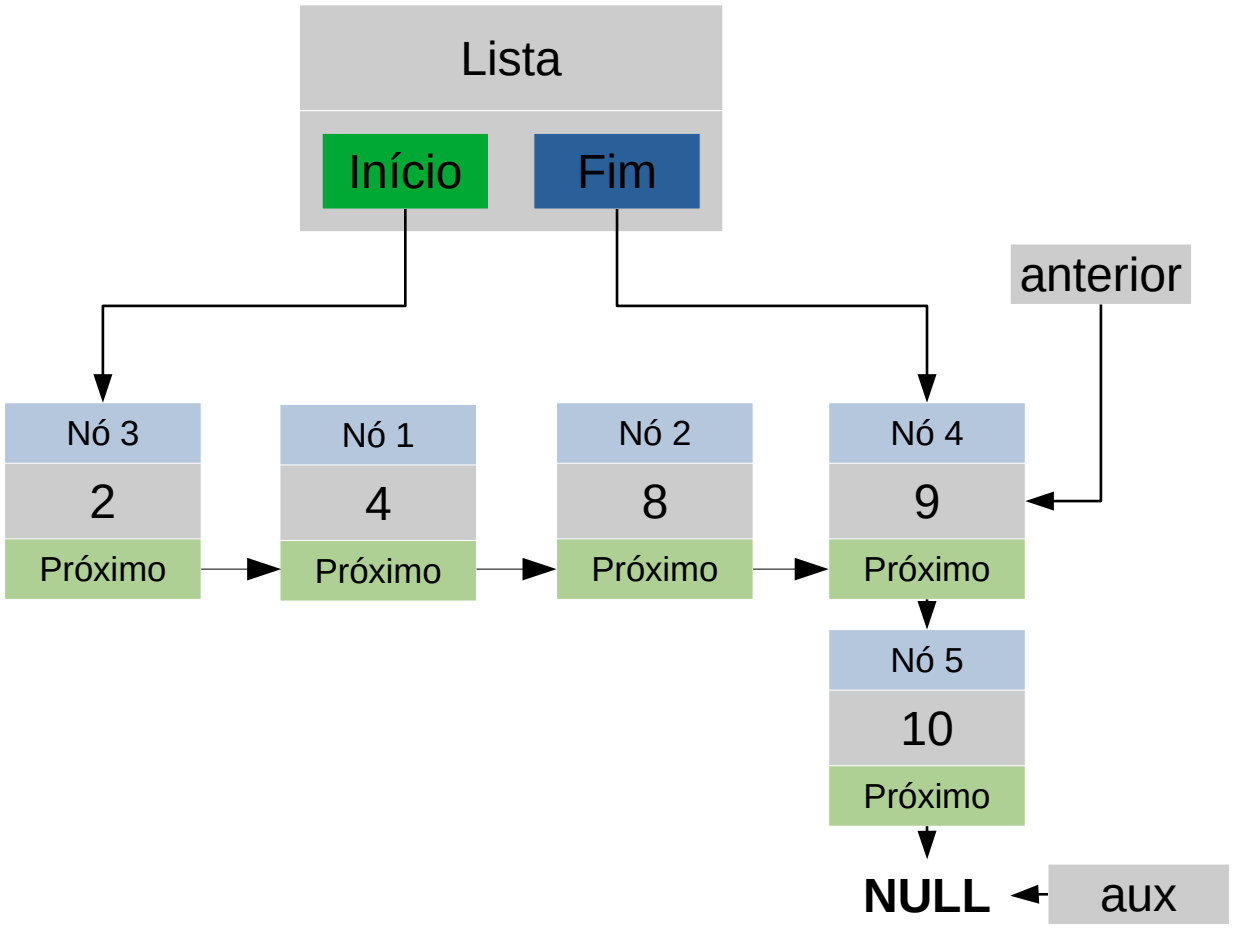
tamanho = 4



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

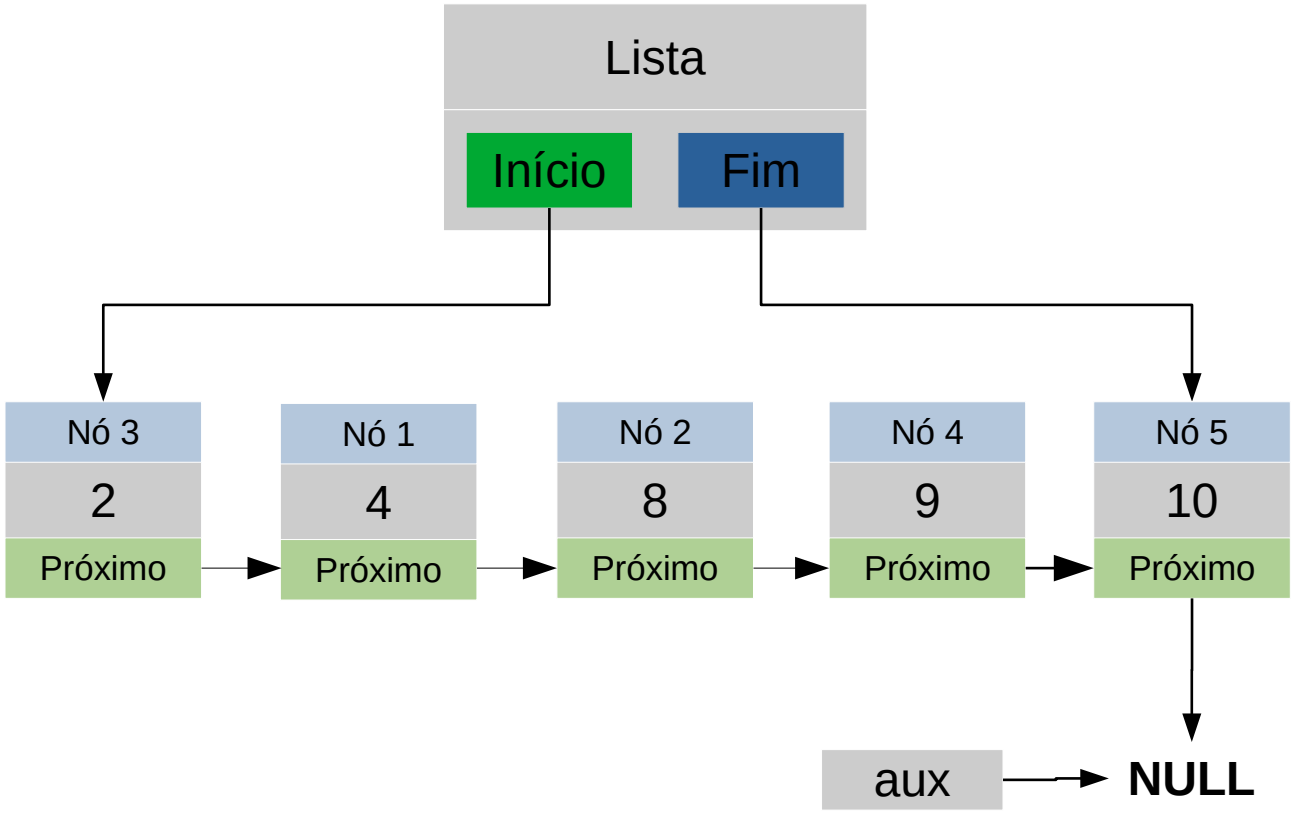
tamanho = 5



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

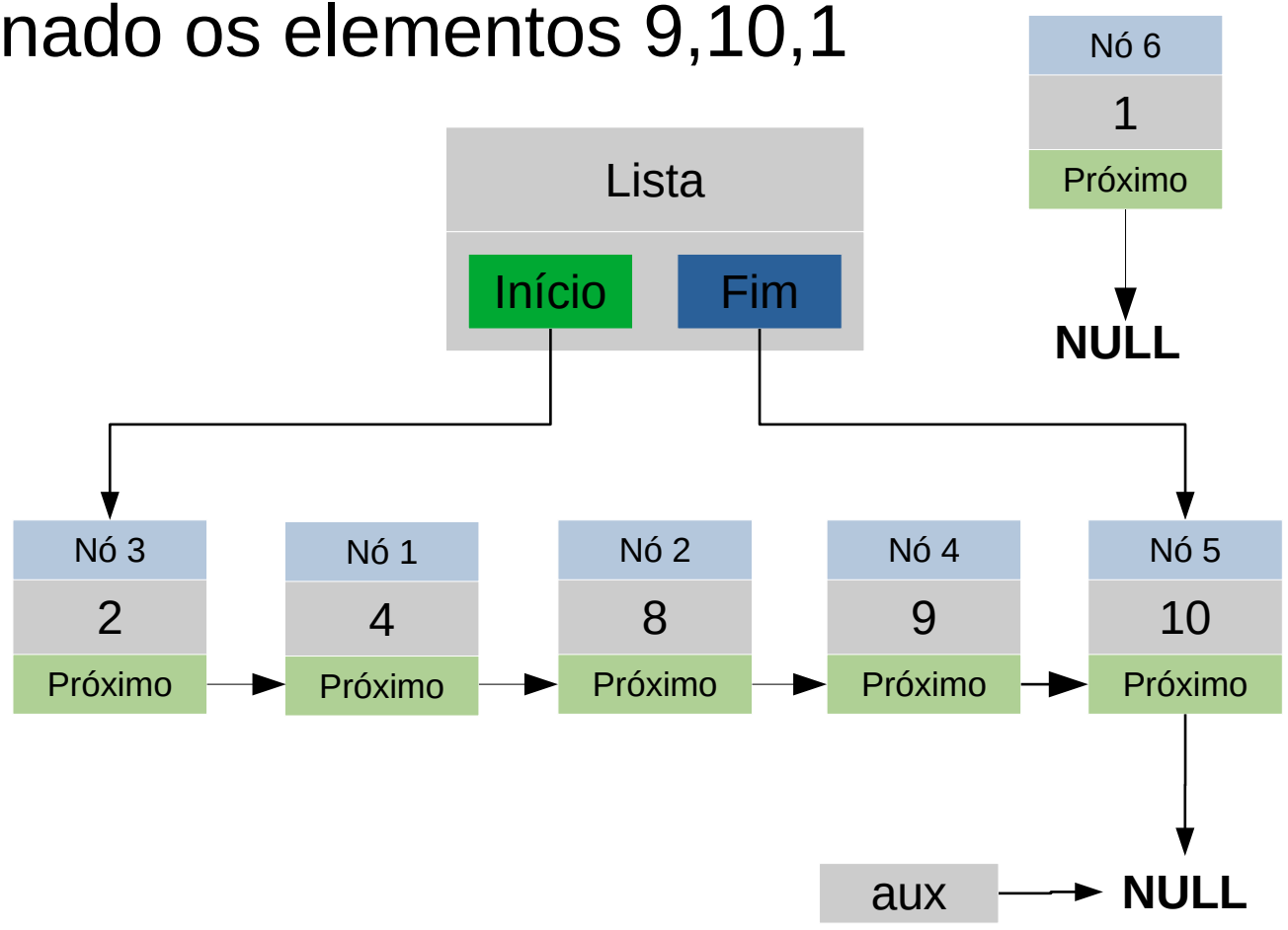
tamanho = 5



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

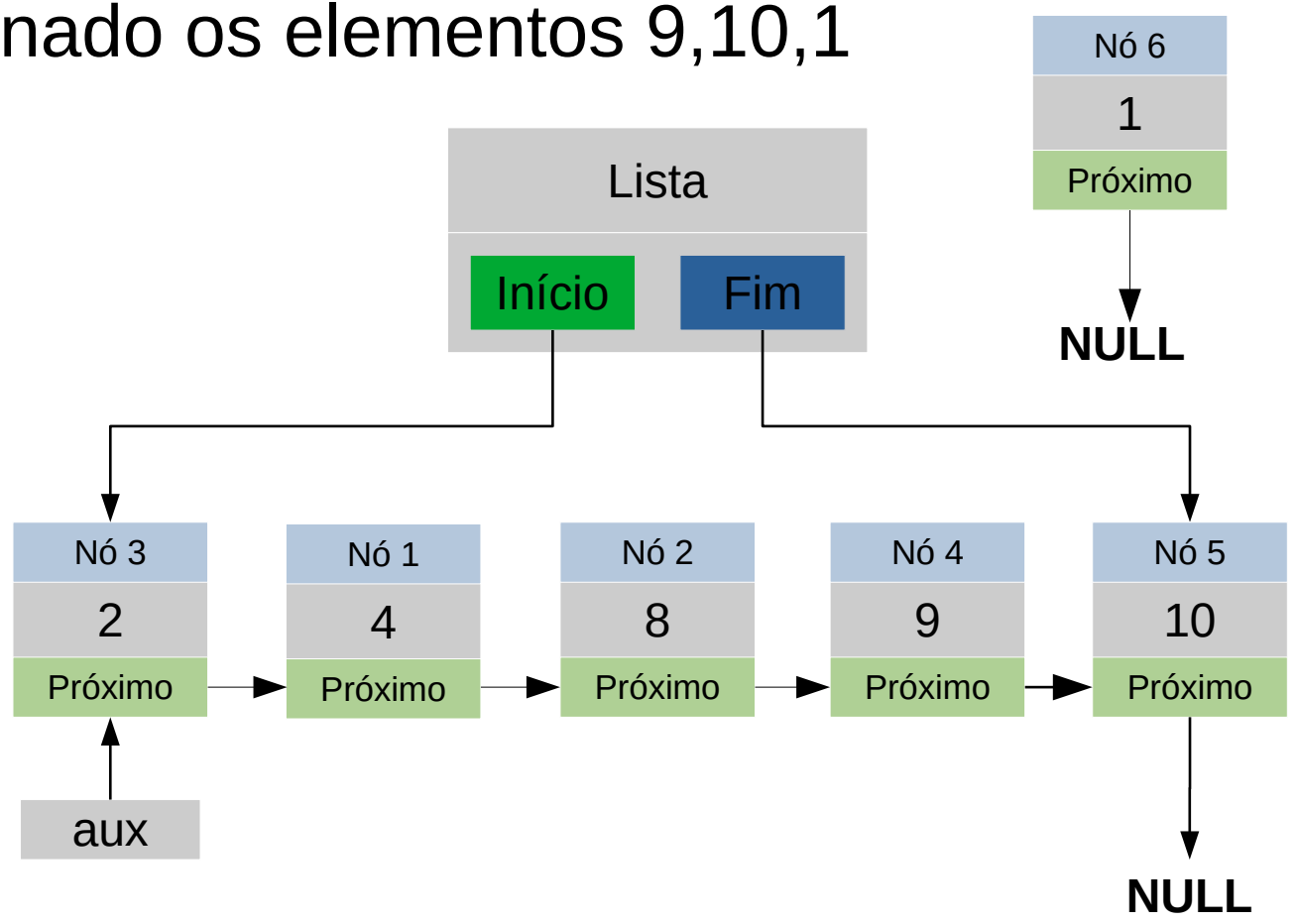
tamanho = 5



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

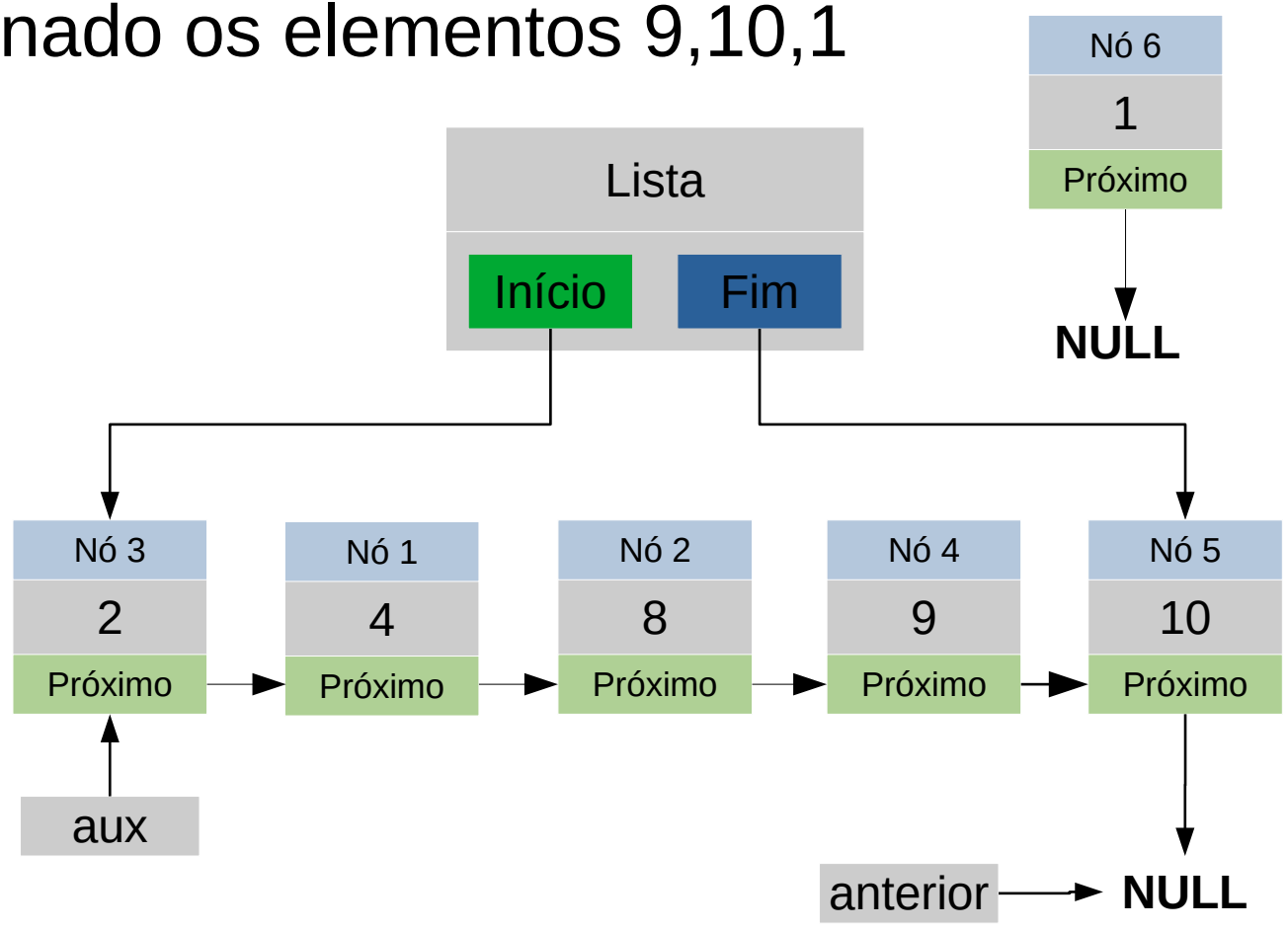
tamanho = 5



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

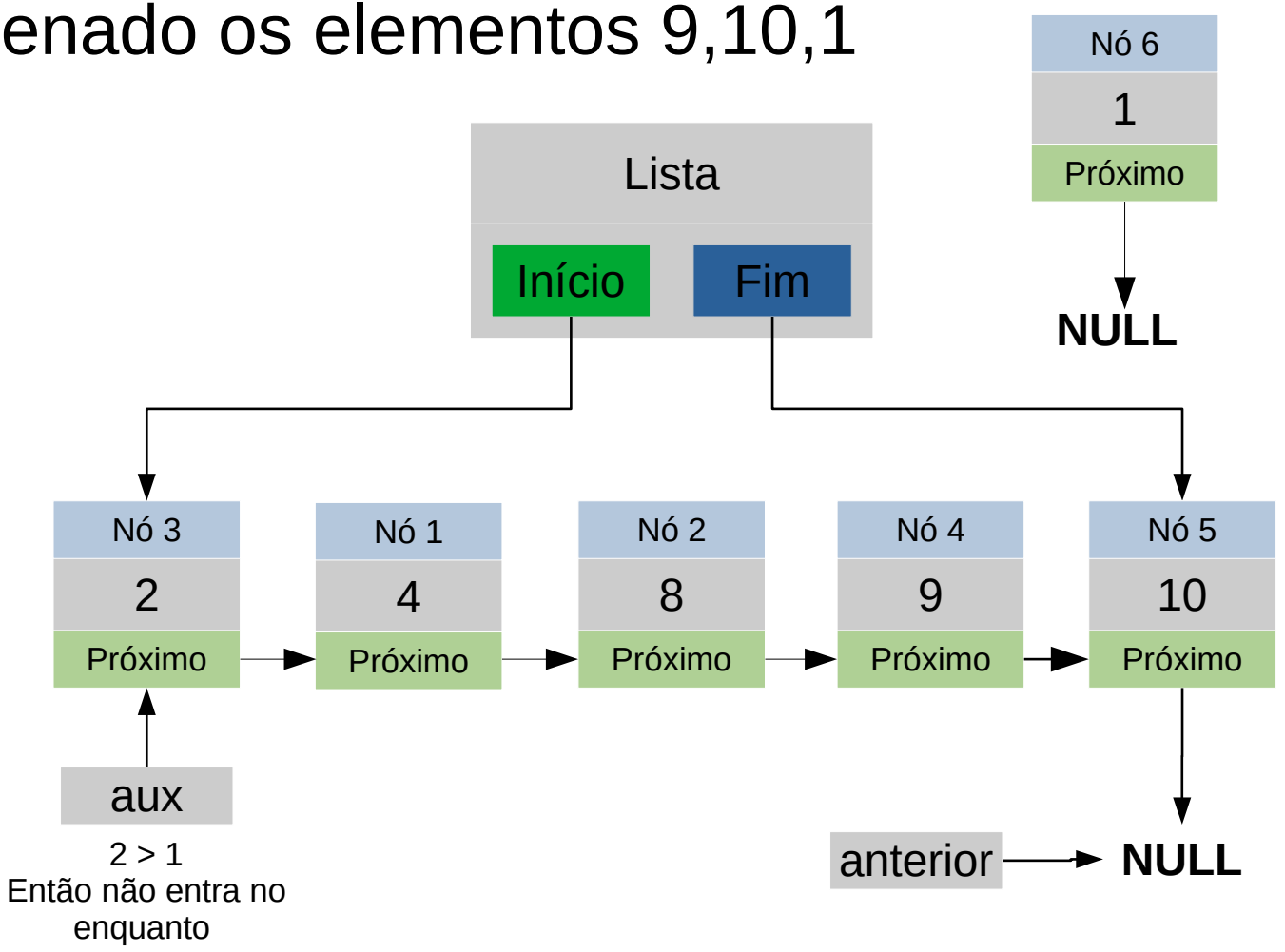
tamanho = 5



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

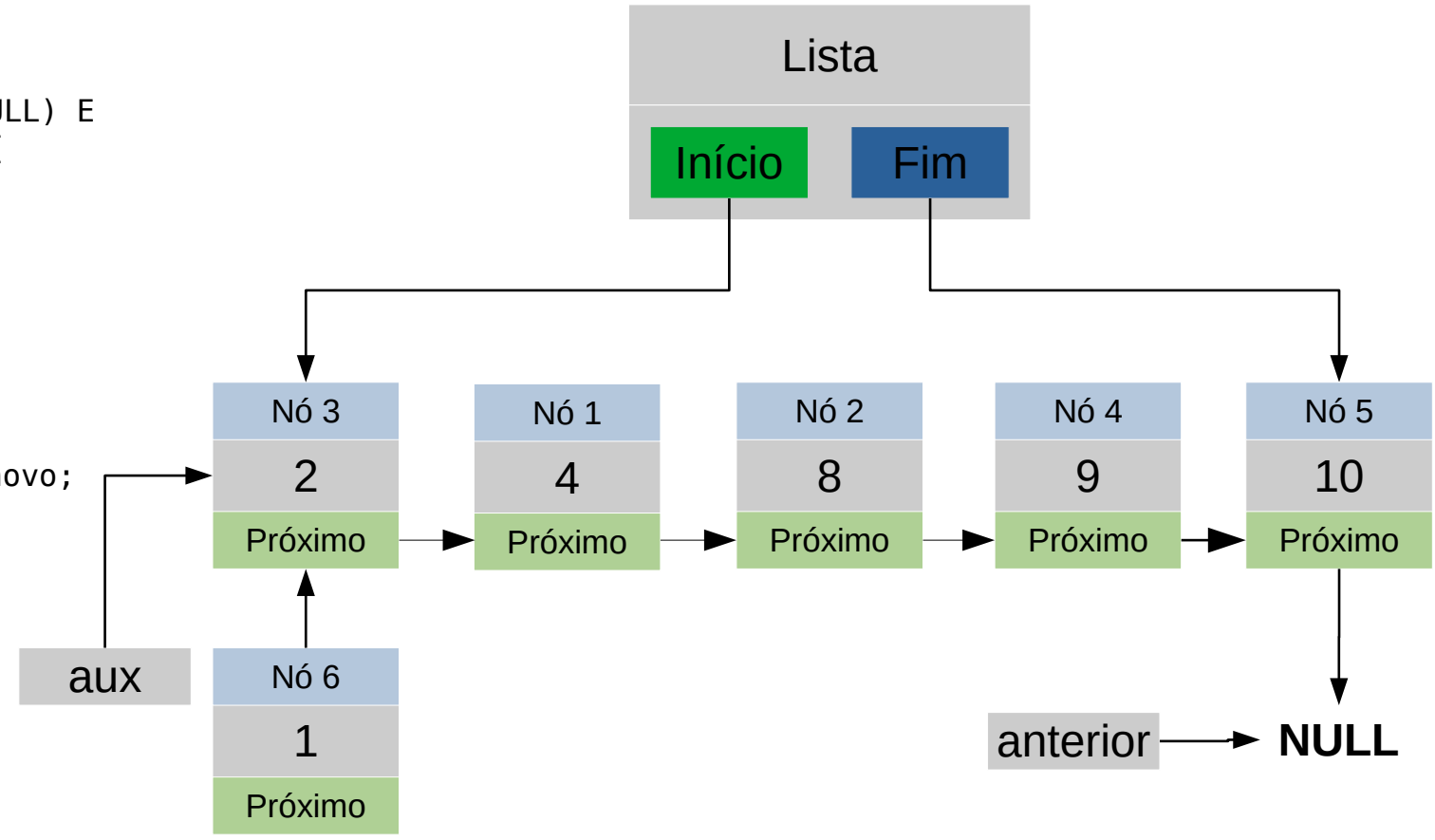
tamanho = 5



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

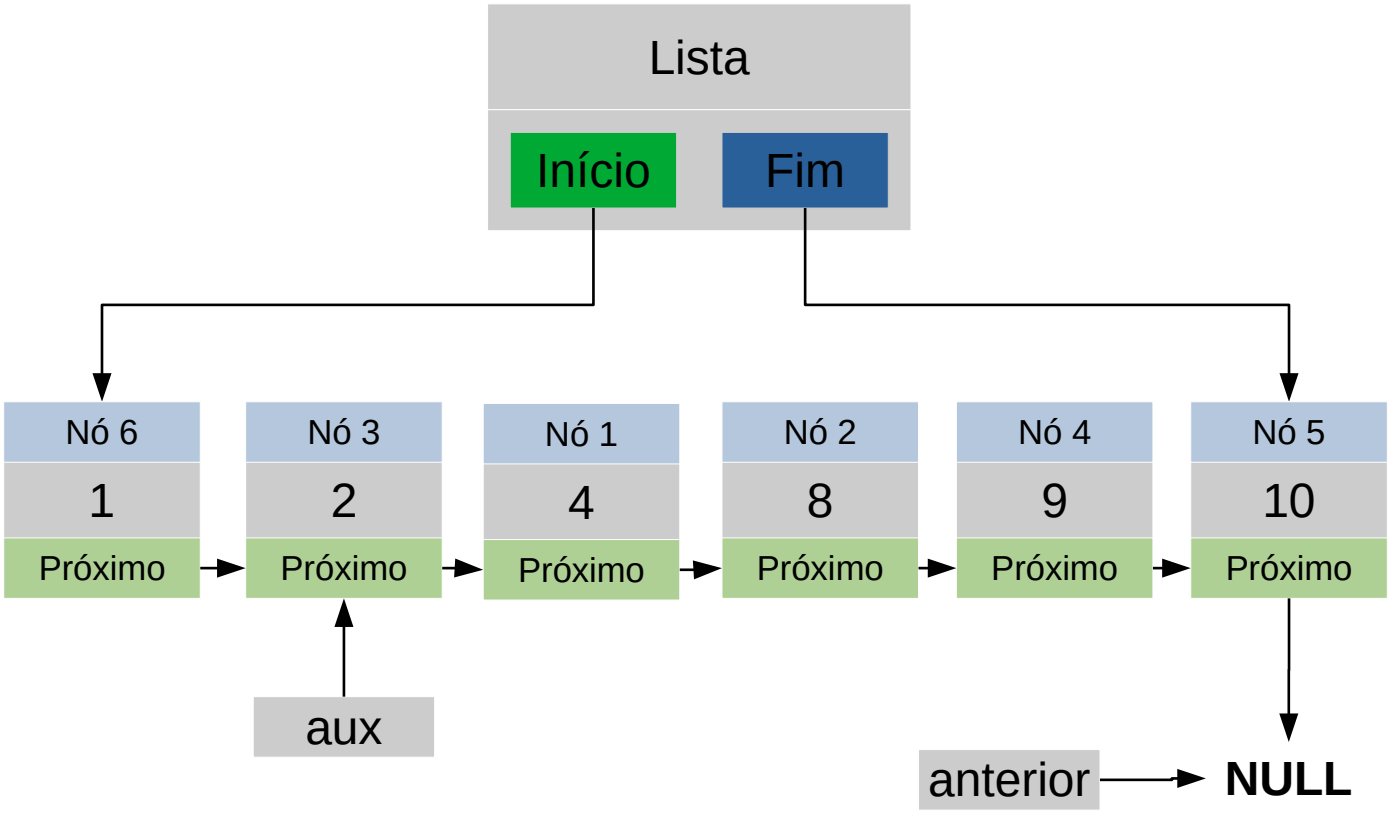
tamanho = 5



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

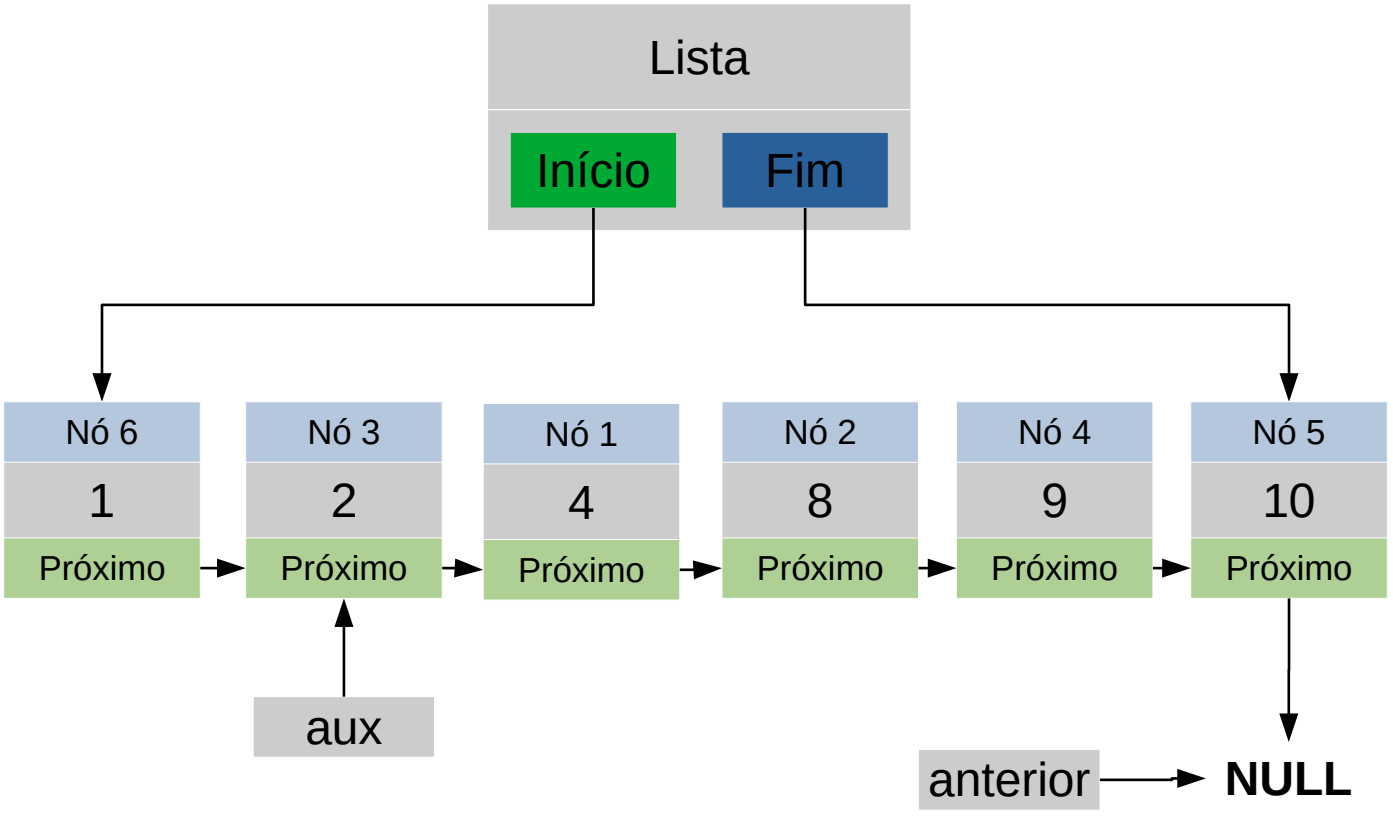
tamanho = 5



Passo 5. Inserir ordenado os elementos 9,10,1

```
novo ← criar_noh(valor);
aux ← inicio;
anterior ← NULL
enquanto ((auxiliar != NULL) E
(auxiliar.dado < valor)){
     anterior ← auxiliar;
     aux = aux.proximo;

}
novo.proximo = aux;
Se (anterior == NULL){
     inicio = novo;
}se nao{
     anteiror.proximo = novo;
}
tamanho++;
Se (aux = NULL){
     fim = novo;
}
```

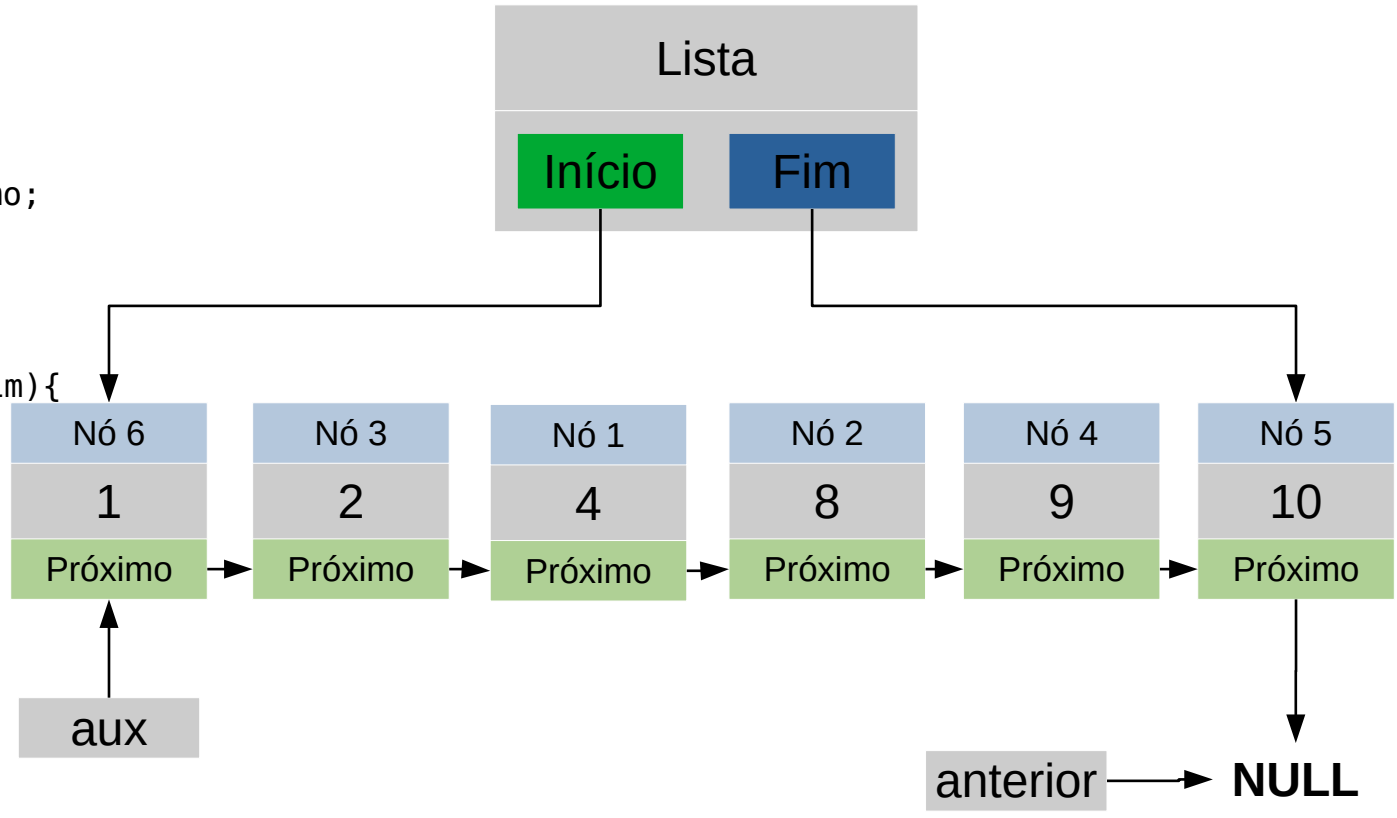
tamanho = 6



Passo 6. Remover elemento 10

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```

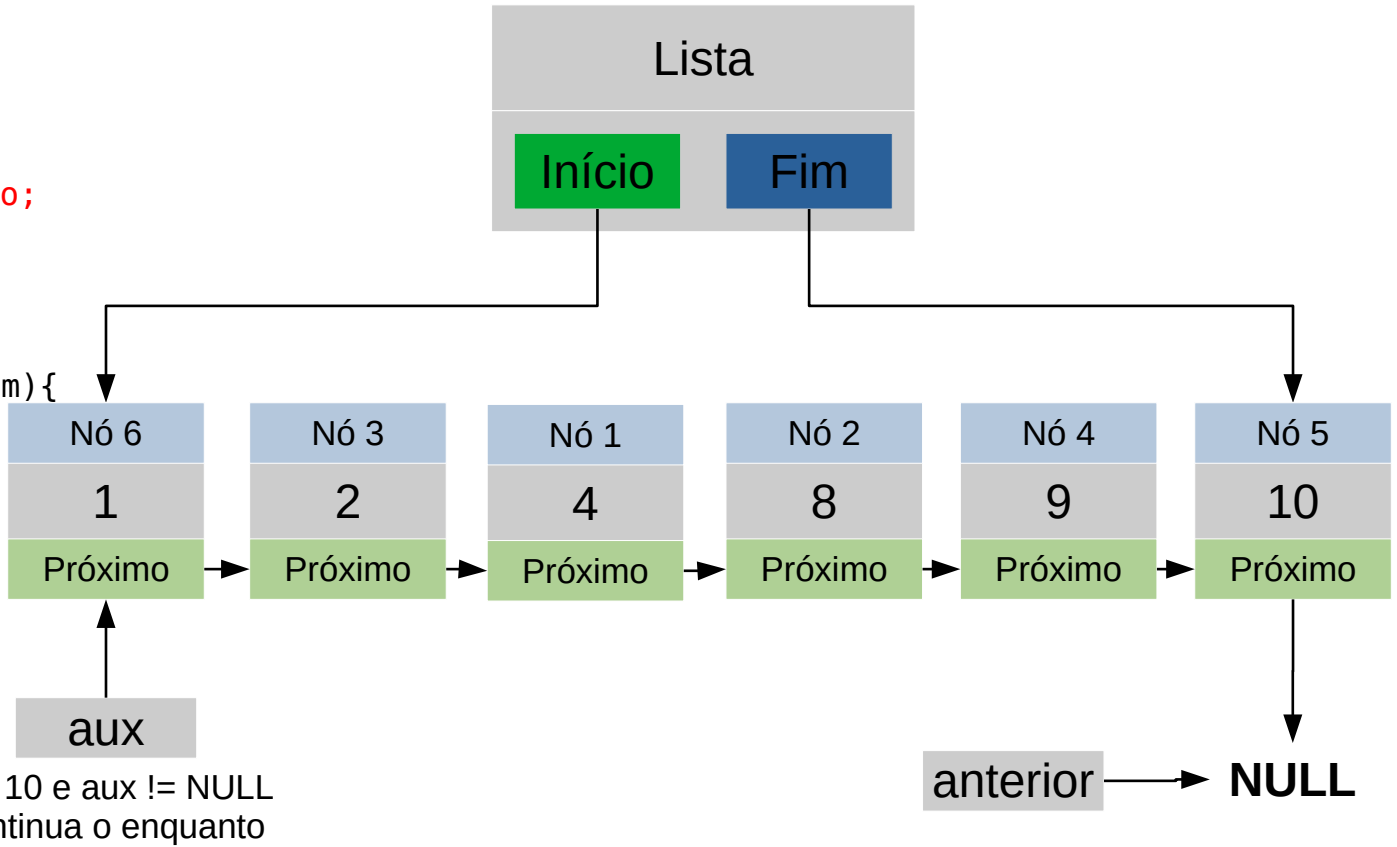
tamanho = 6



Passo 6. Remover elemento 10

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```

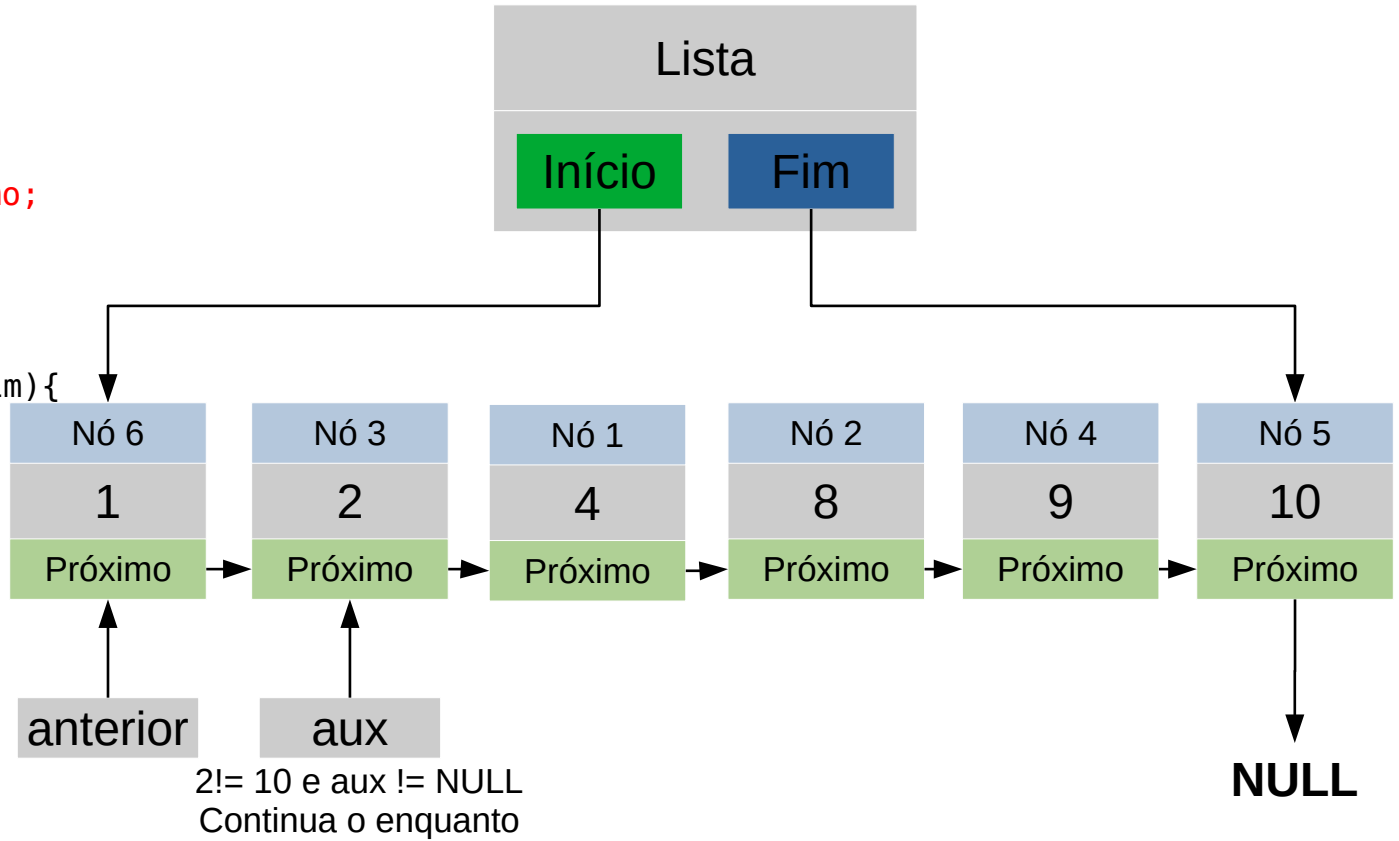
tamanho = 6



Passo 6. Remover elemento 10

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```

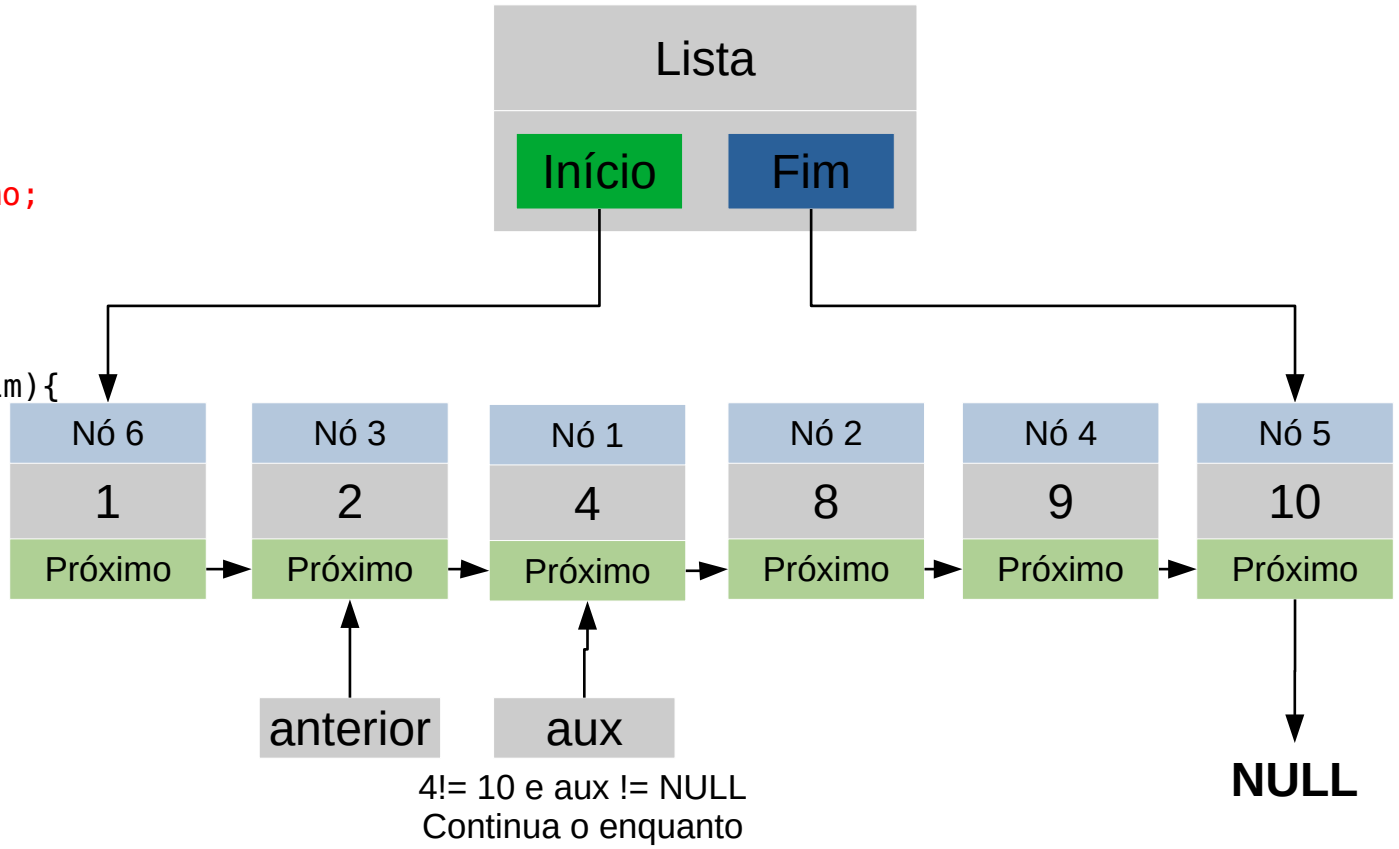
tamanho = 6



Passo 6. Remover elemento 10

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```

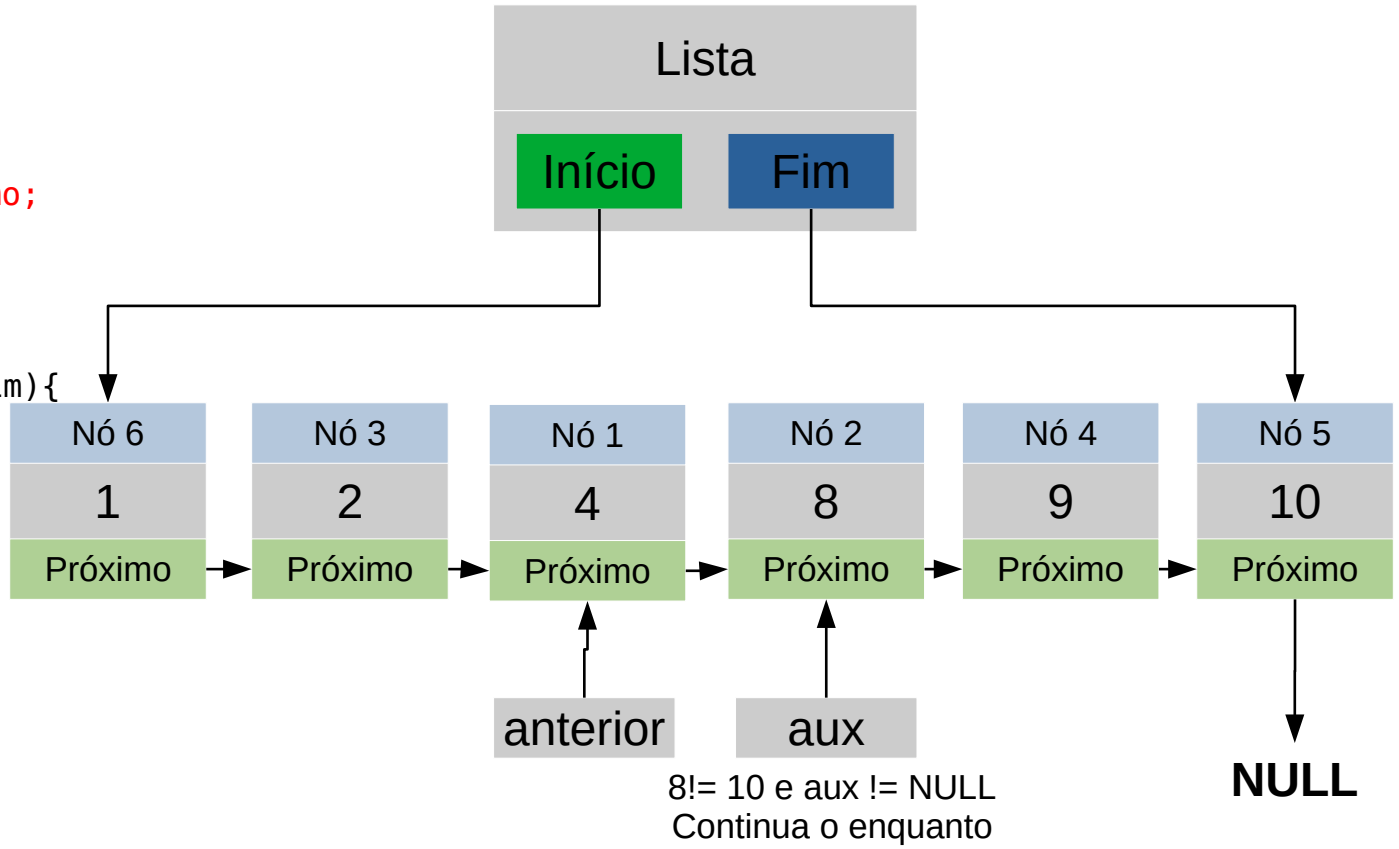
tamanho = 6



Passo 6. Remover elemento 10

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```

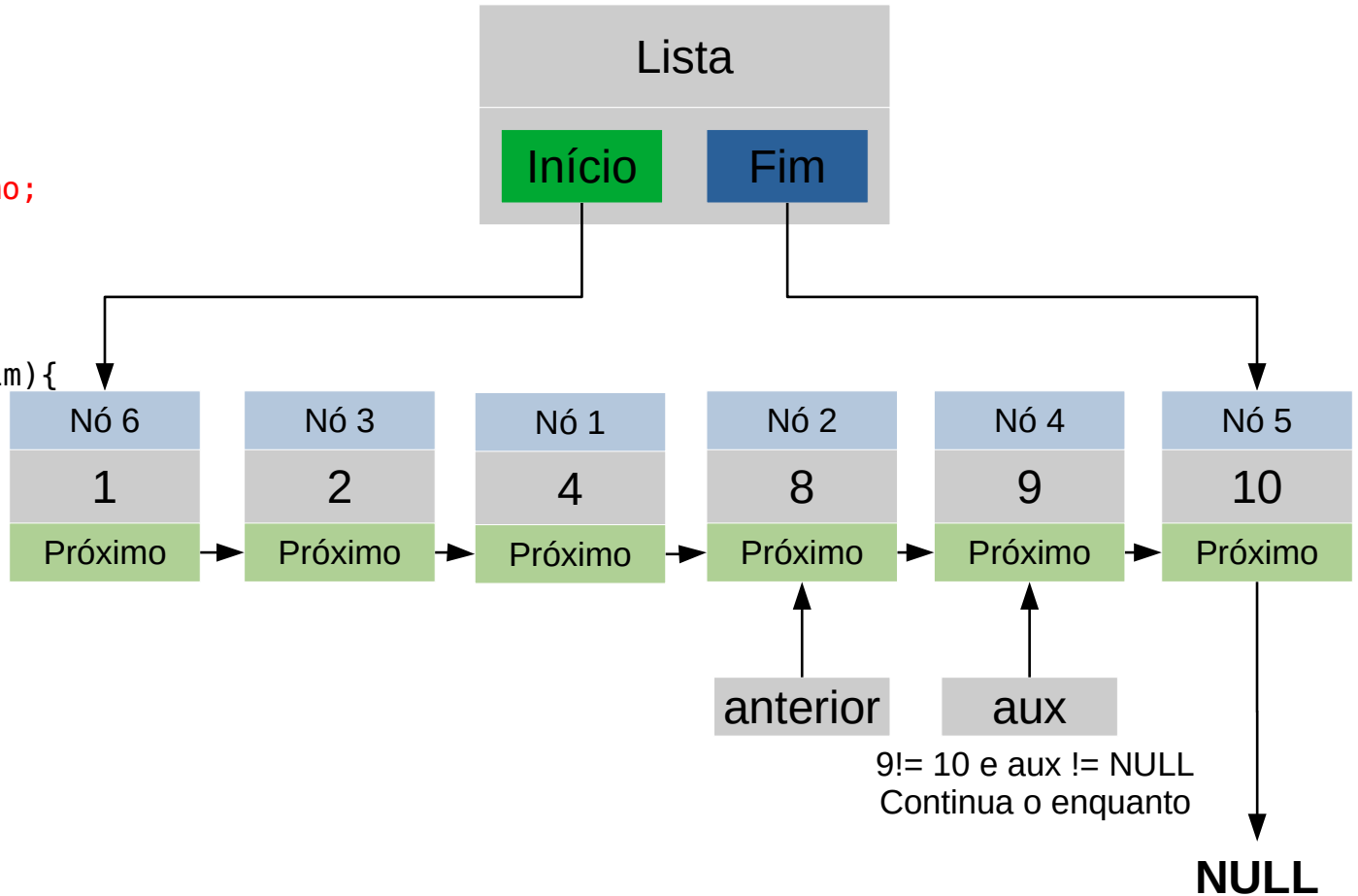
tamanho = 6



Passo 6. Remover elemento 10

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```

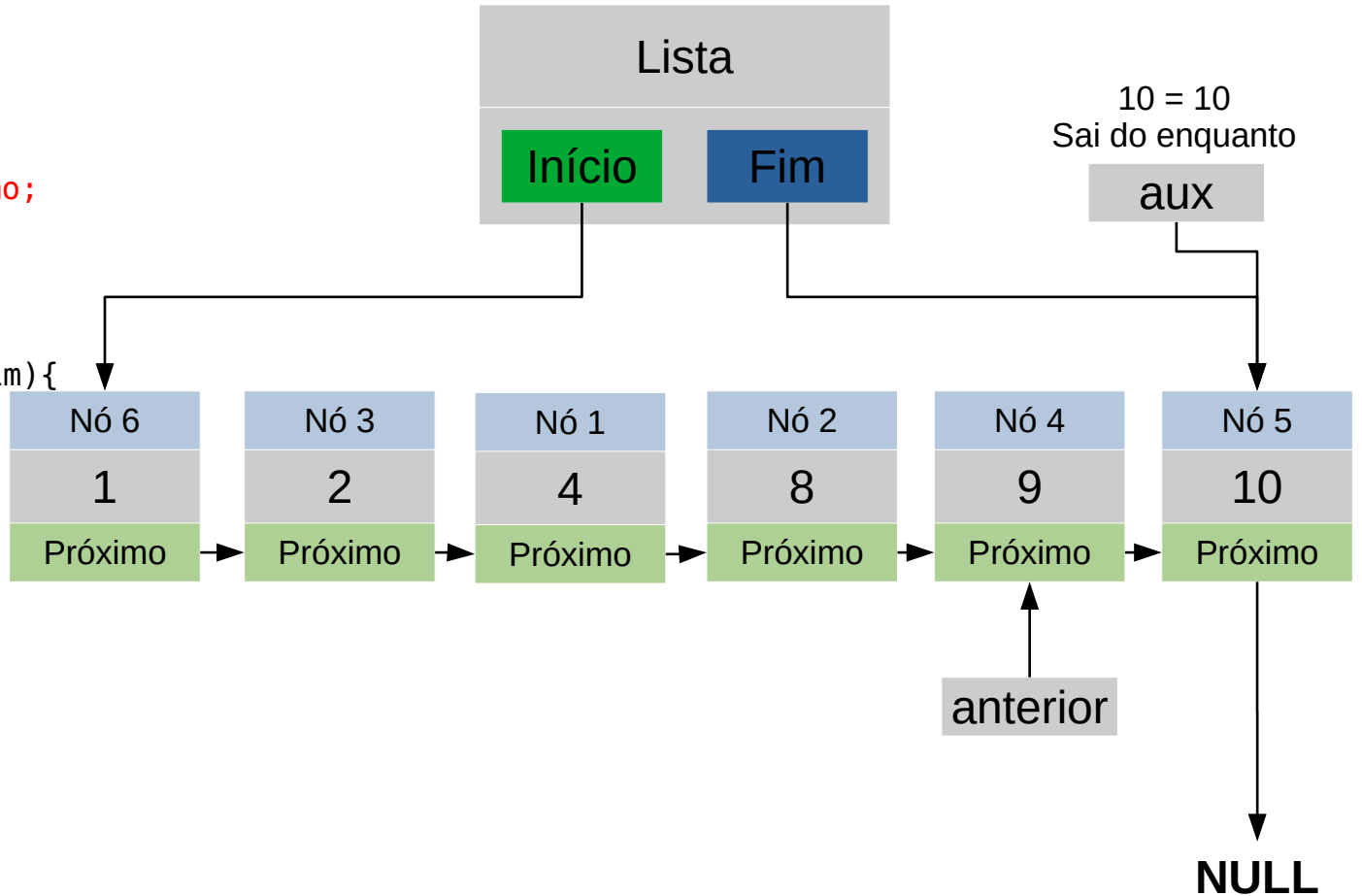
tamanho = 6



Passo 6. Remover elemento 10

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```

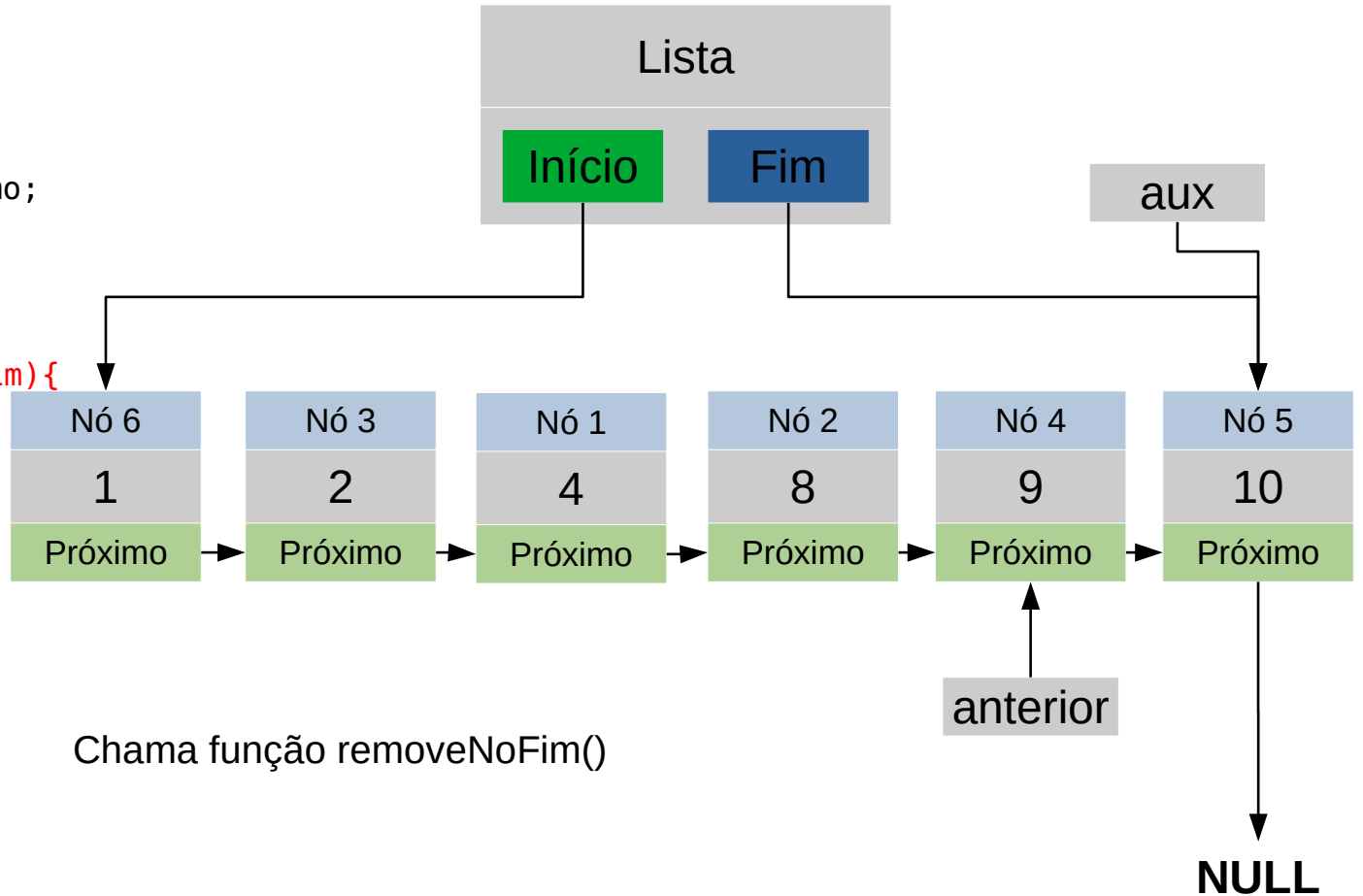
tamanho = 6



Passo 6. Remover elemento 10

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```

tamanho = 6

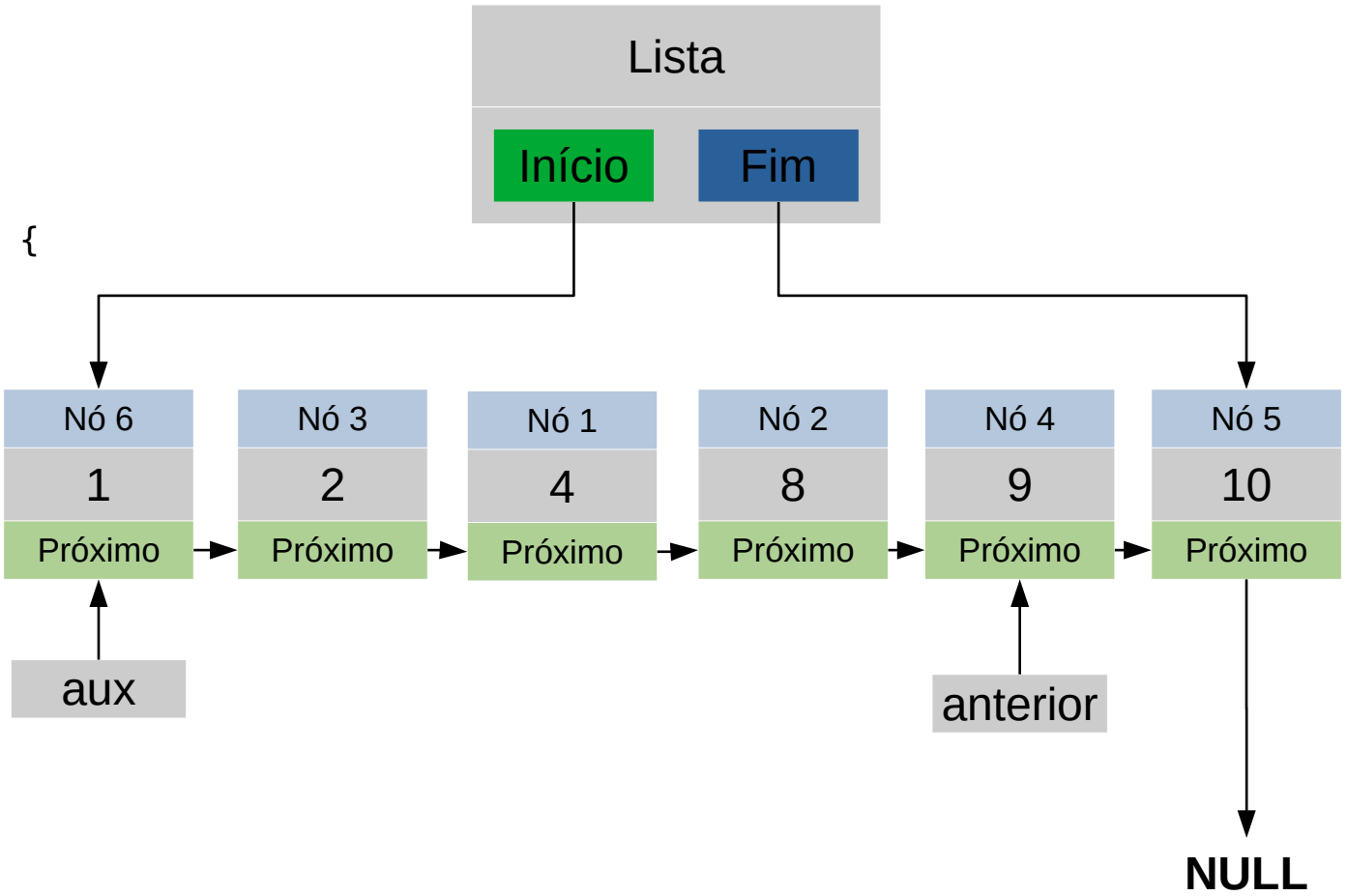


Chama função removeNoFim()

Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

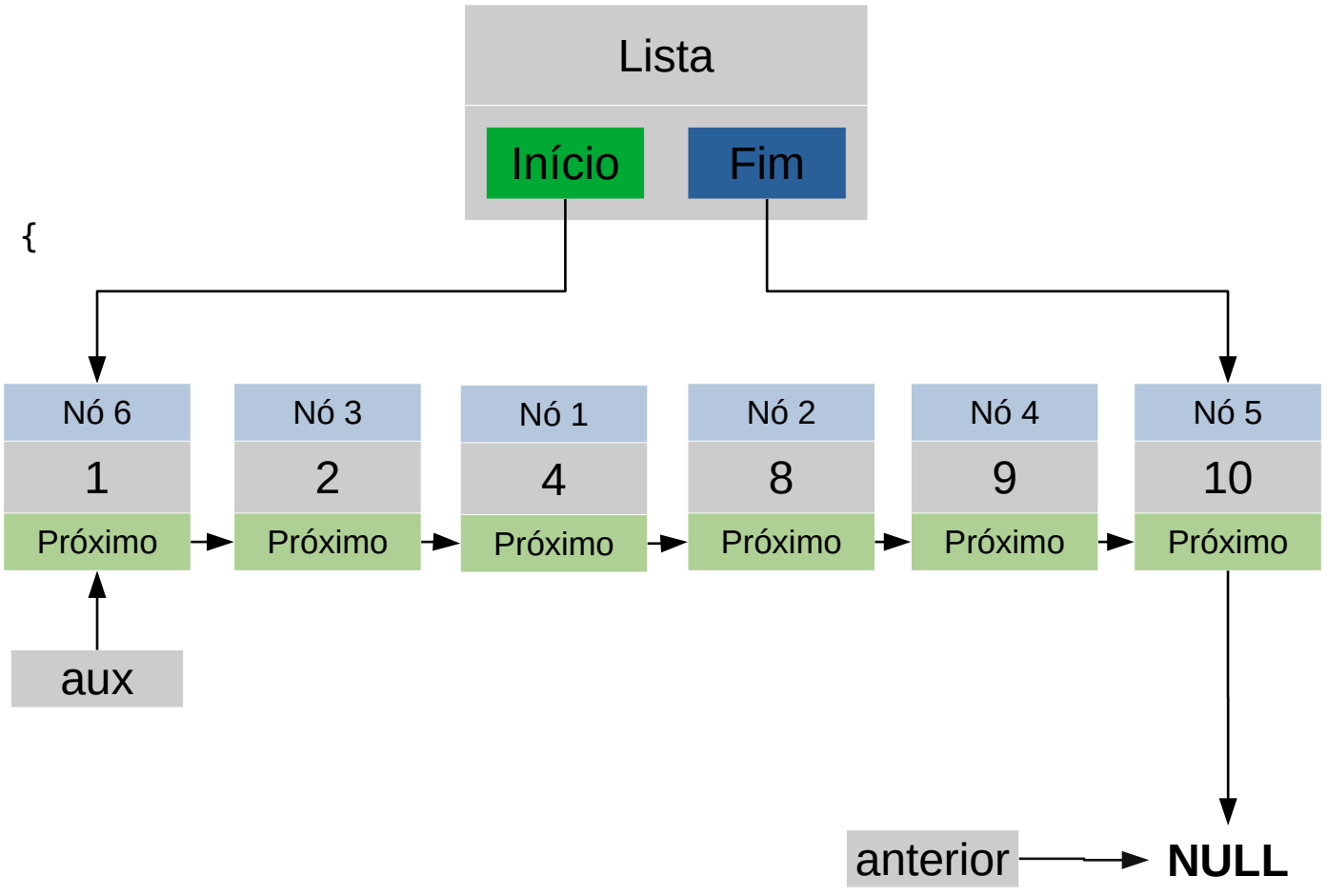
tamanho = 6



Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

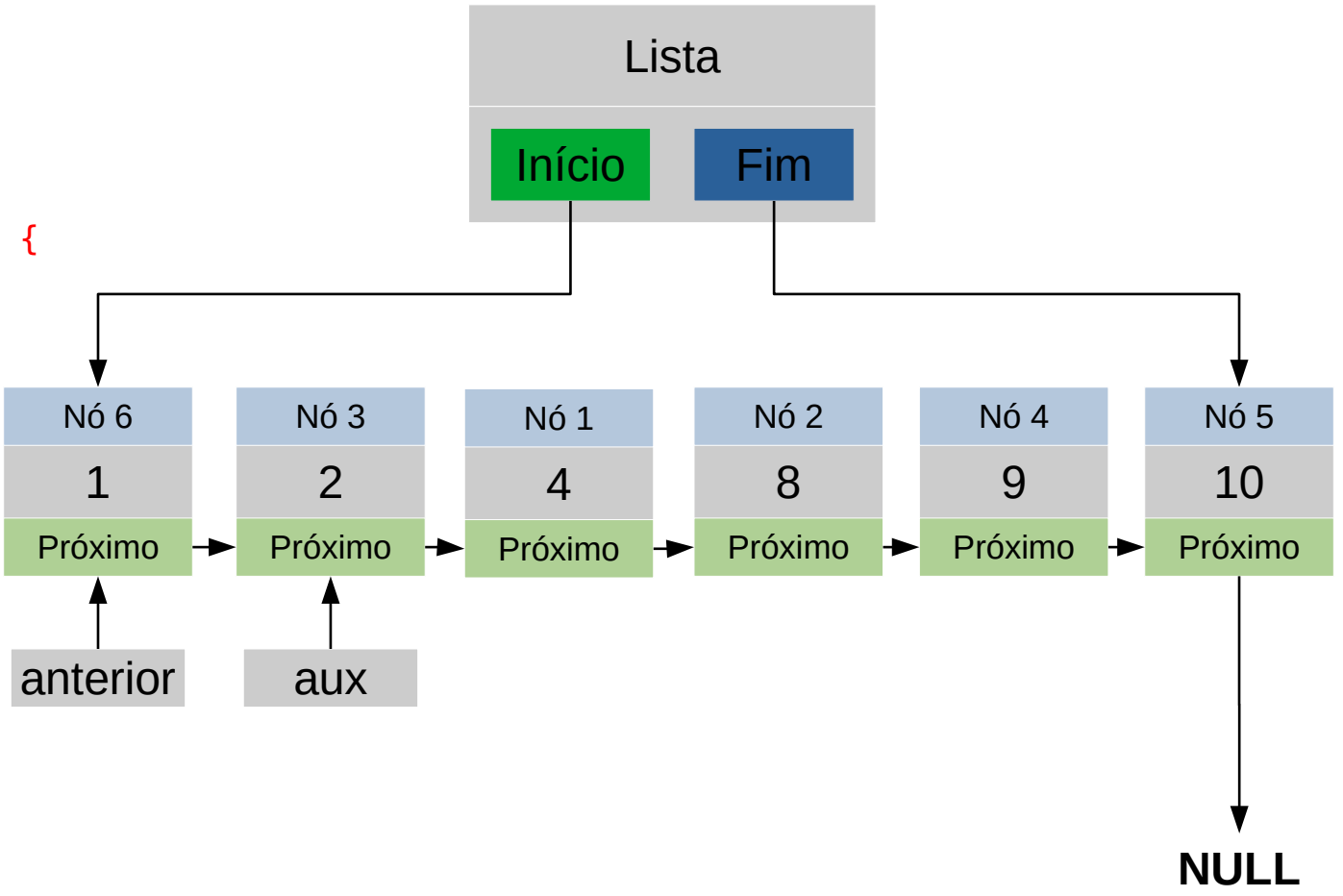
tamanho = 6



Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

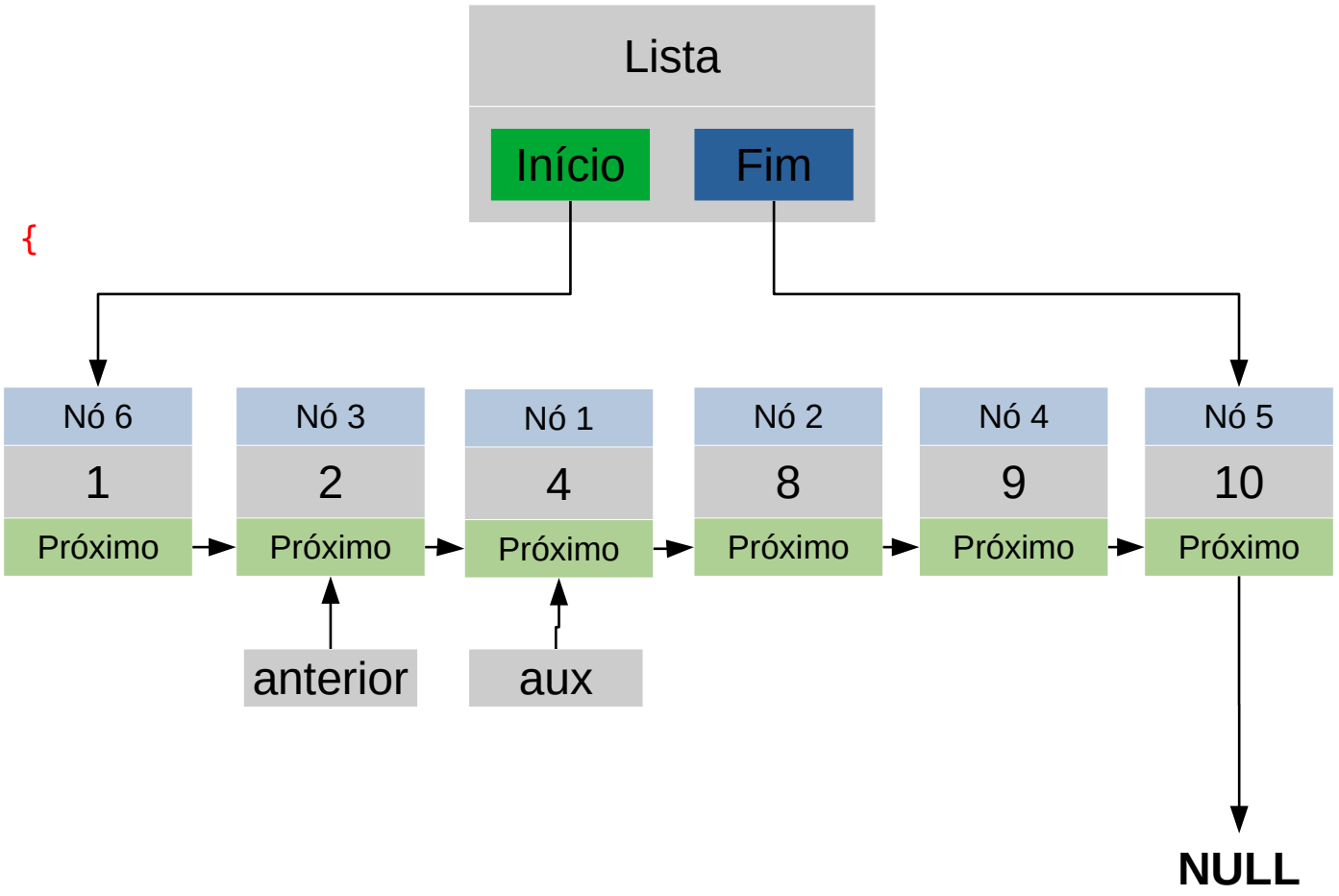
tamanho = 6



Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

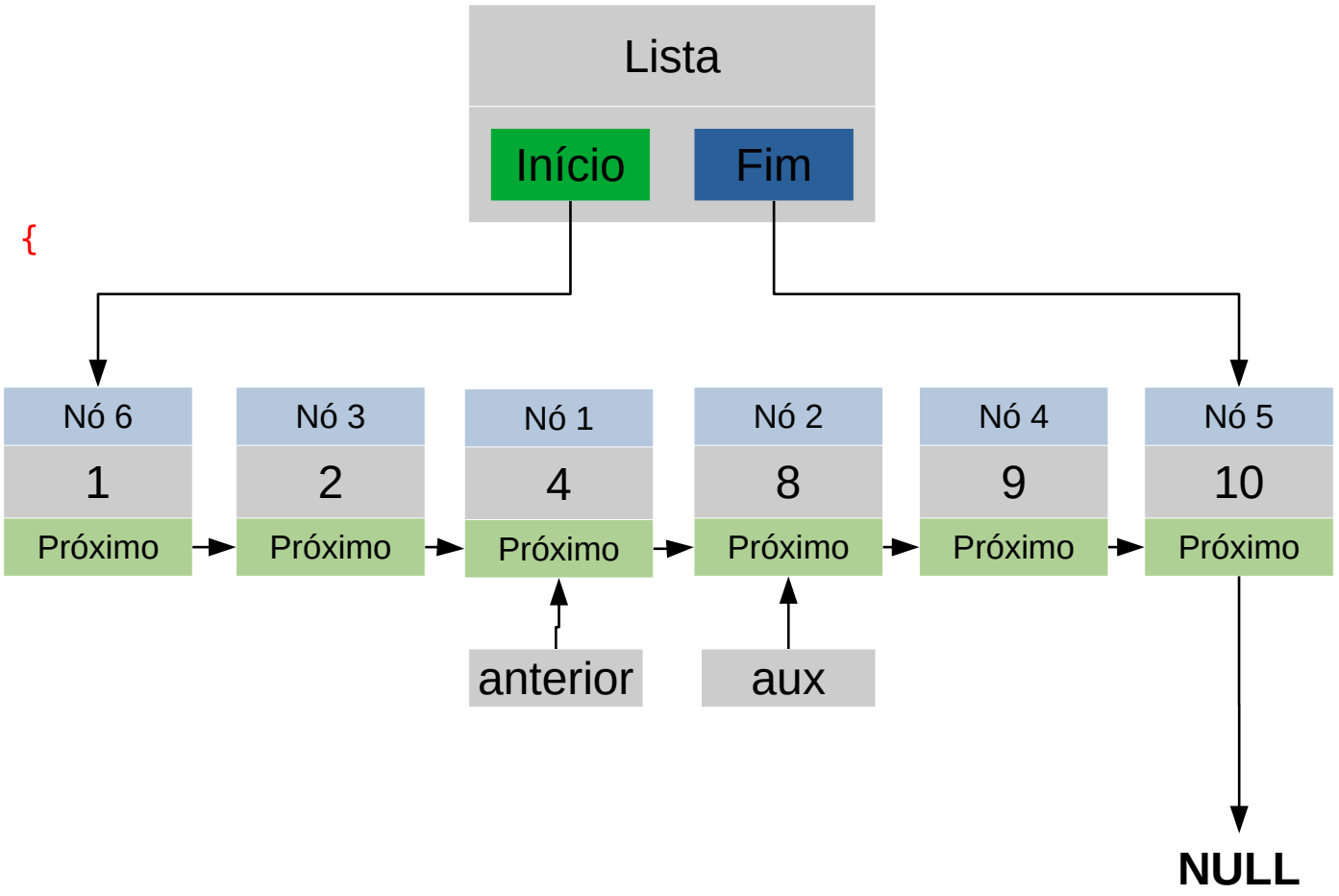
tamanho = 6



Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

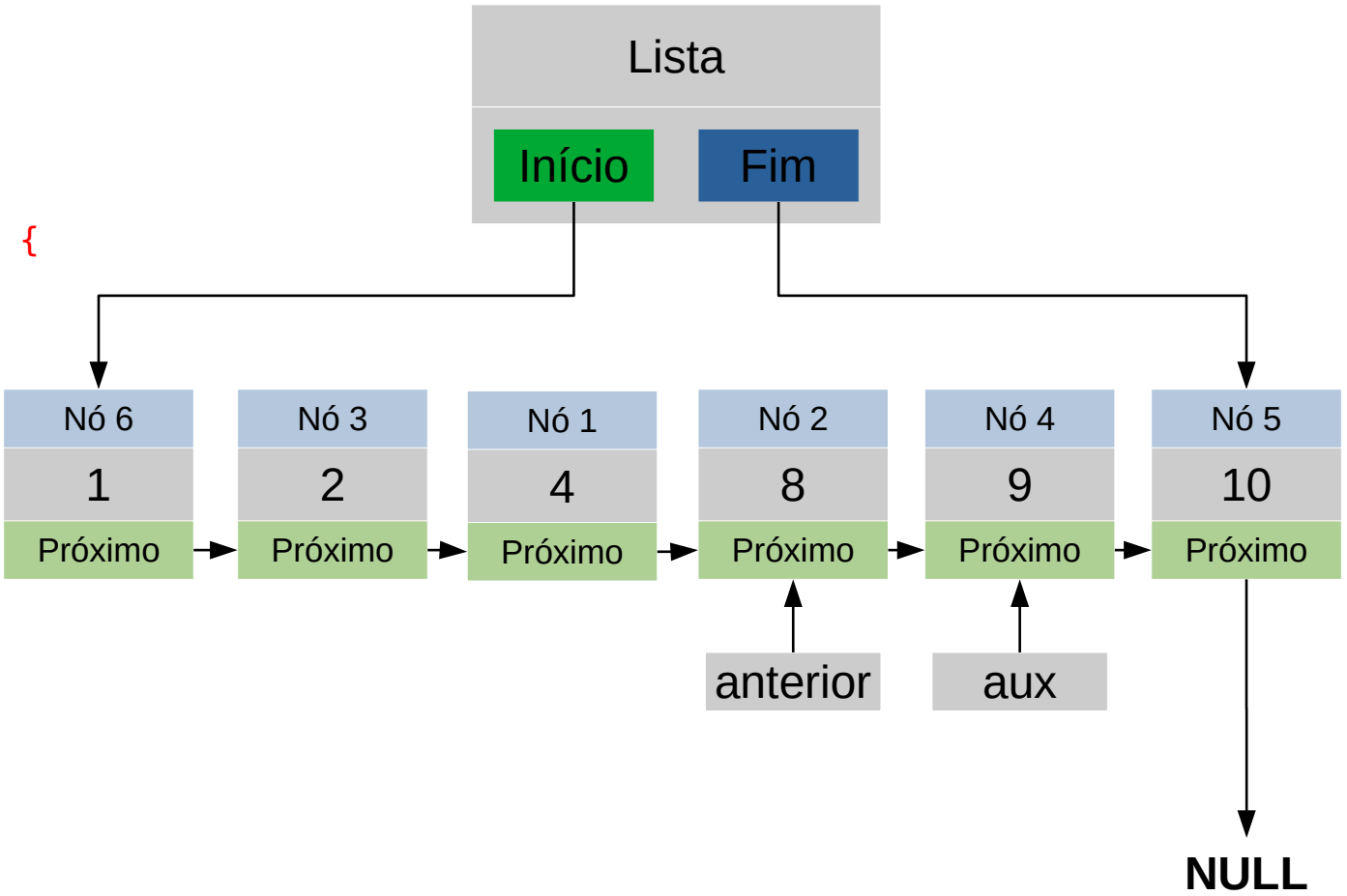
tamanho = 6



Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

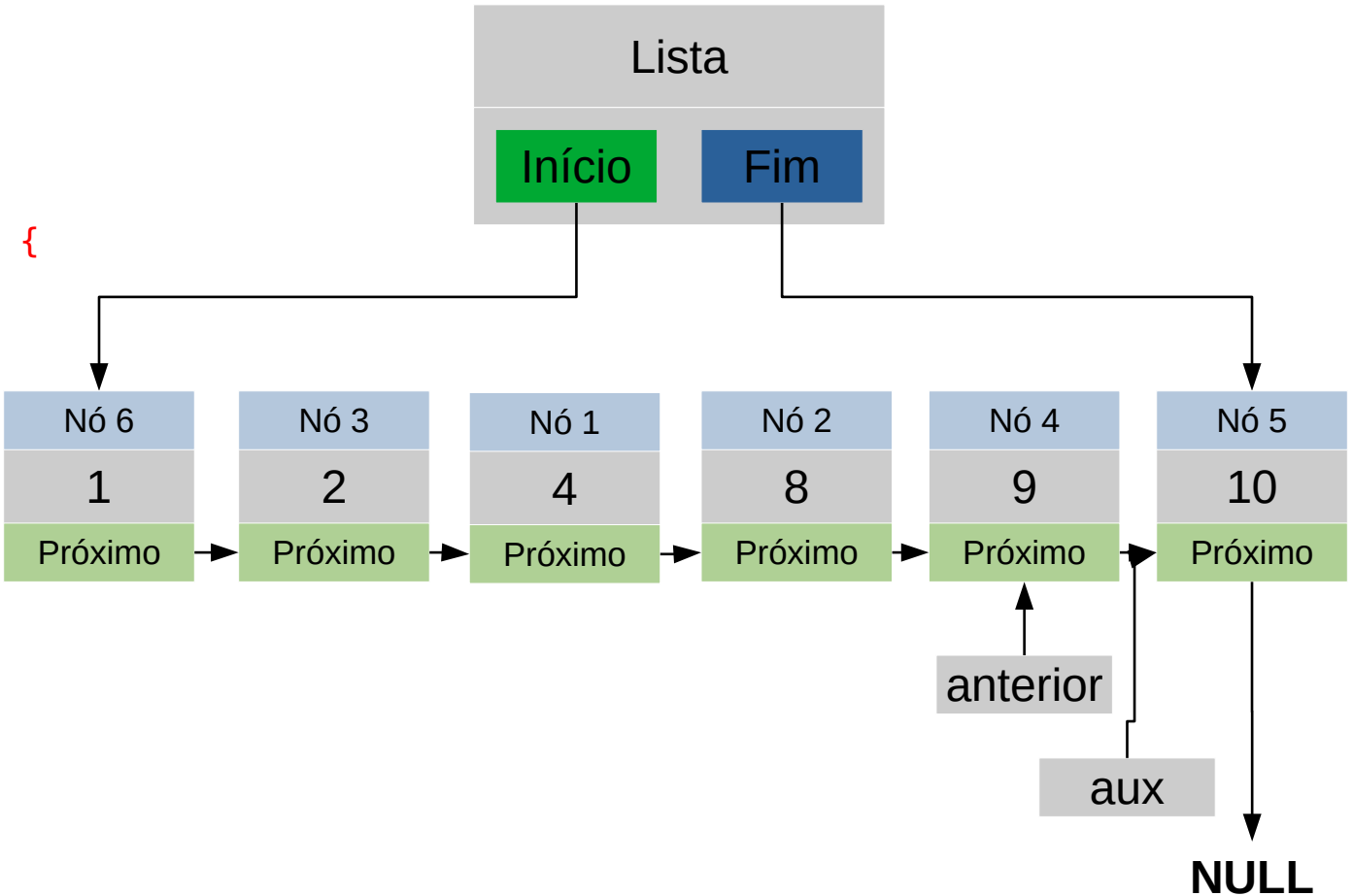
tamanho = 6



Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

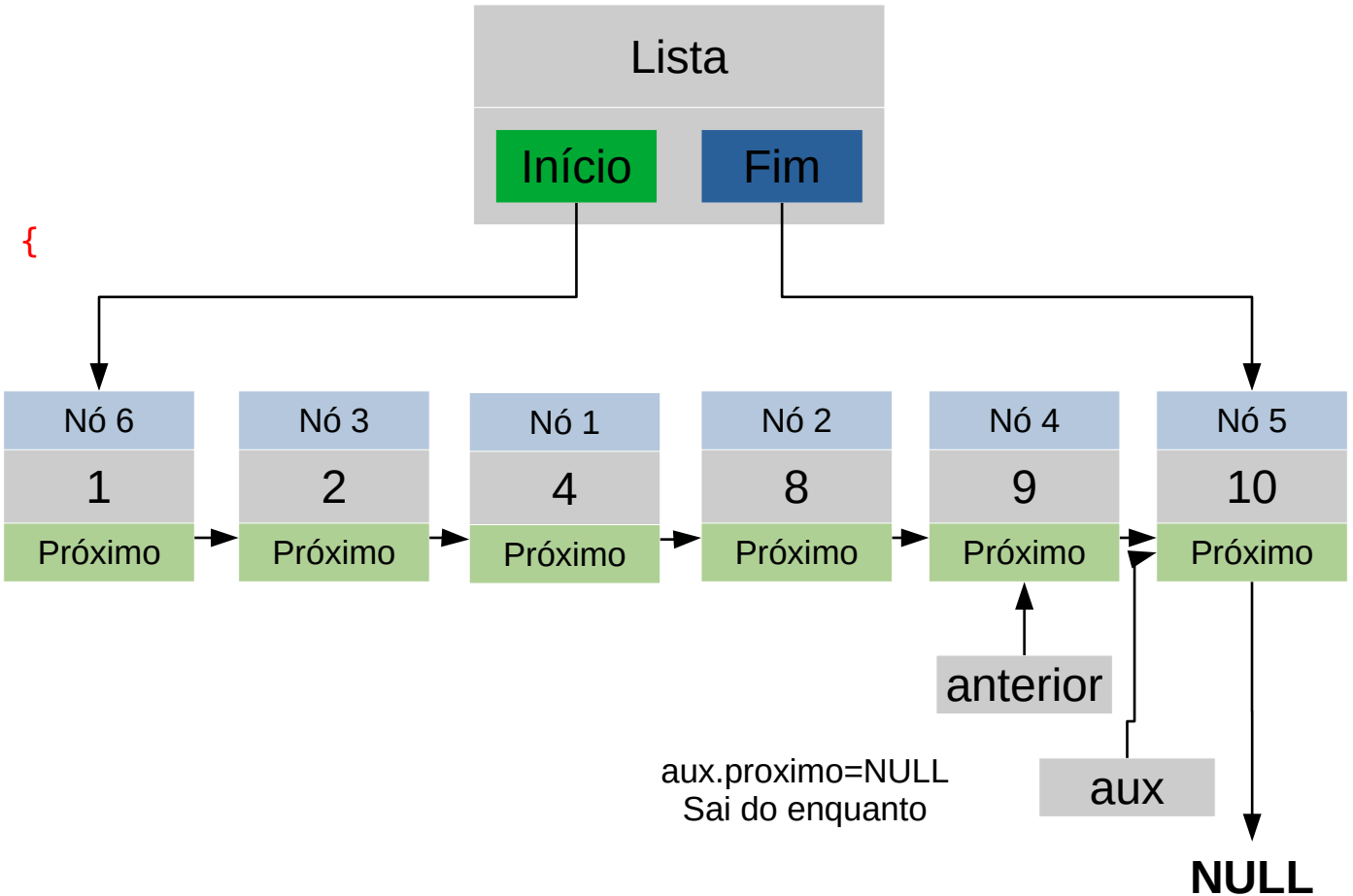
tamanho = 6



Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

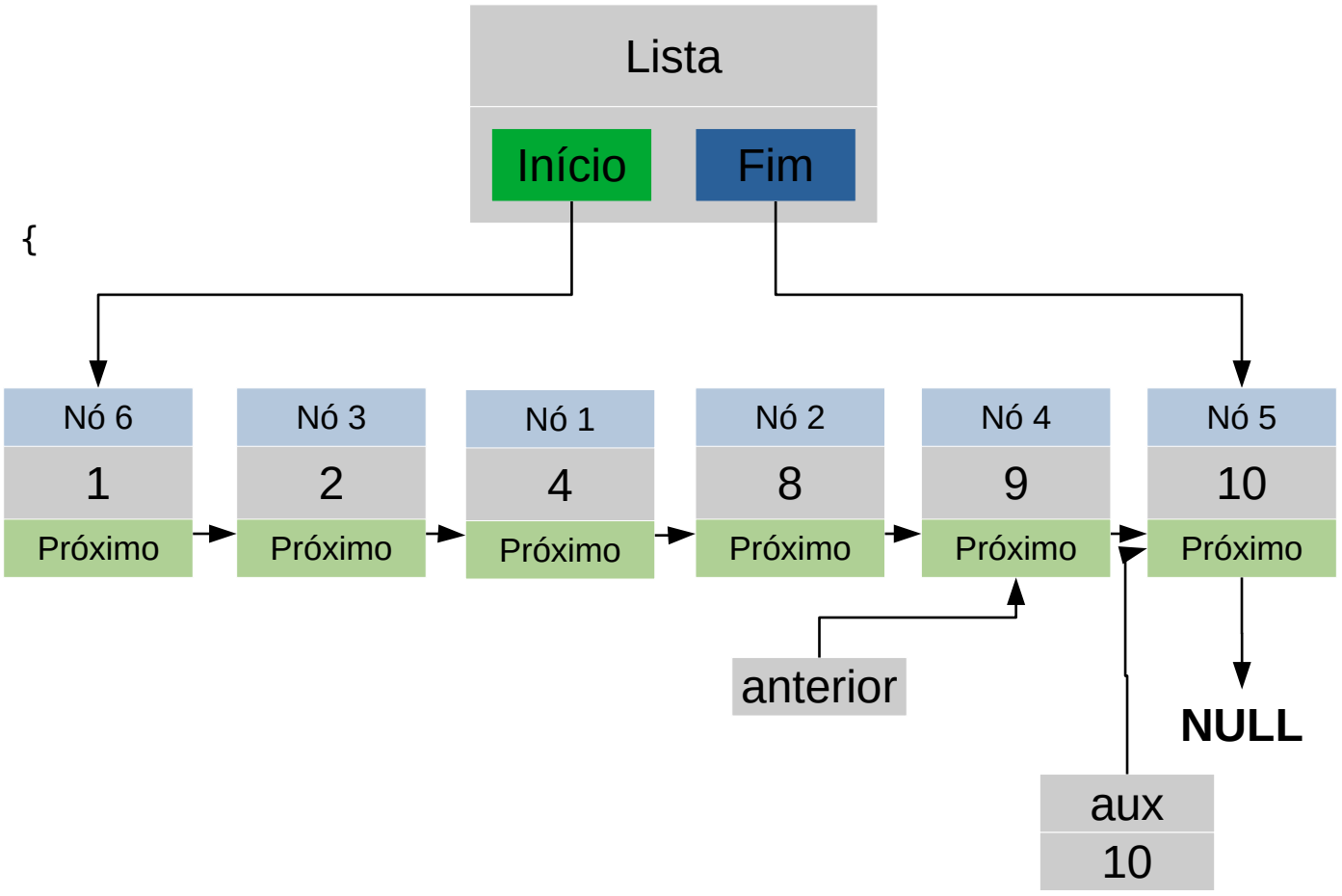
tamanho = 6



Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

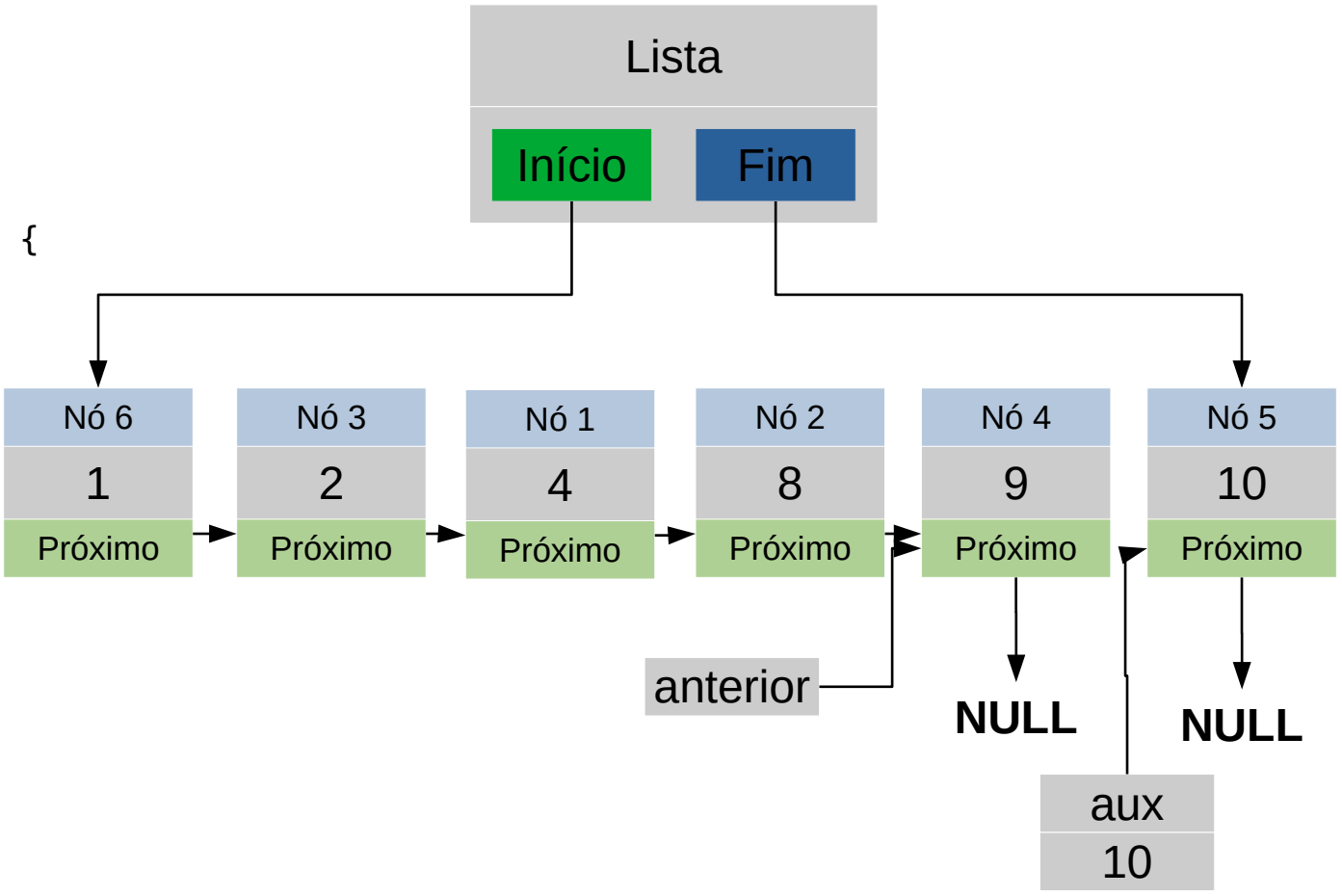
tamanho = 6



Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

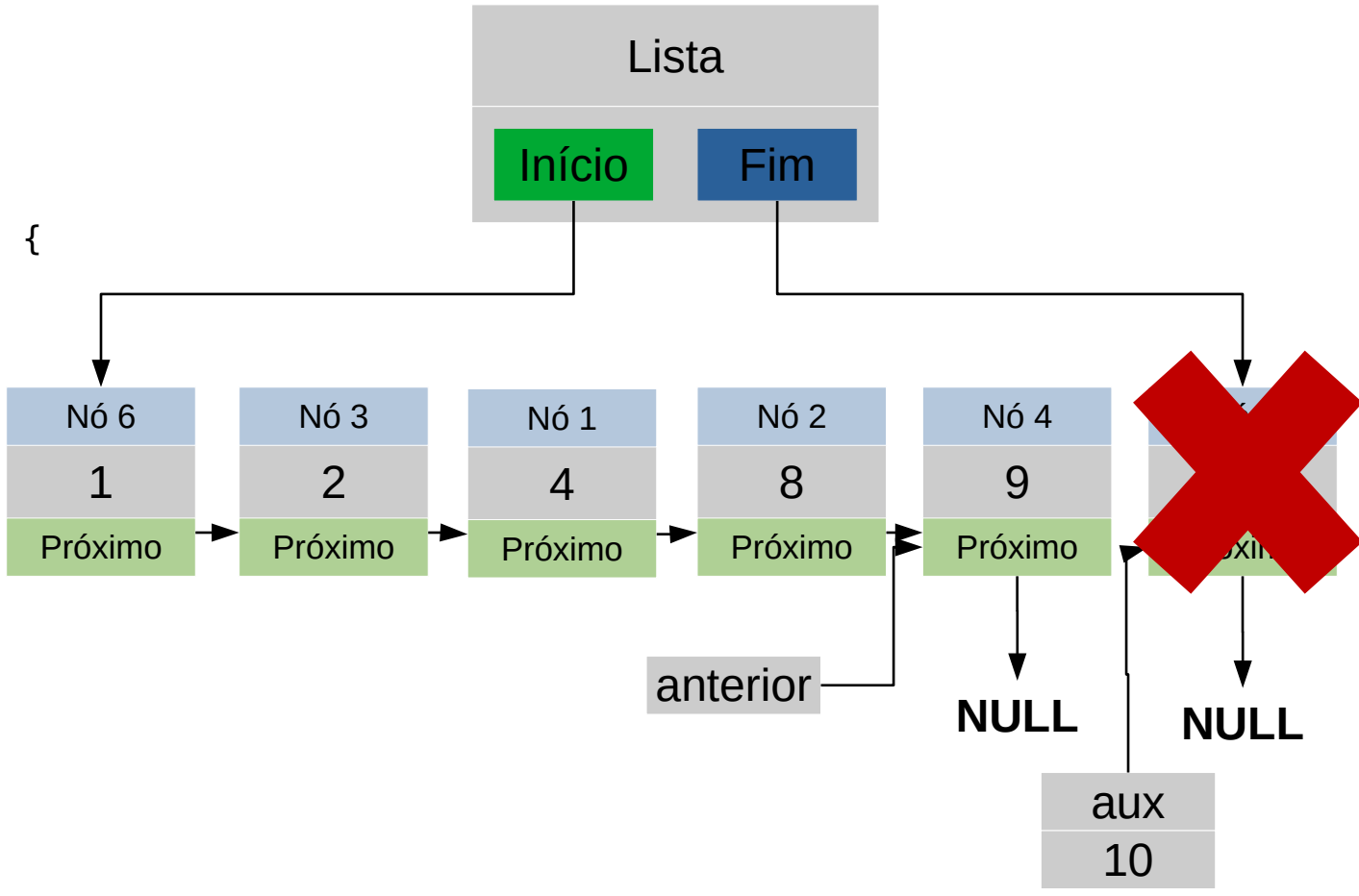
tamanho = 6



Passo 6. Remover elemento 10

```
removeNoFim():
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

tamanho = 6

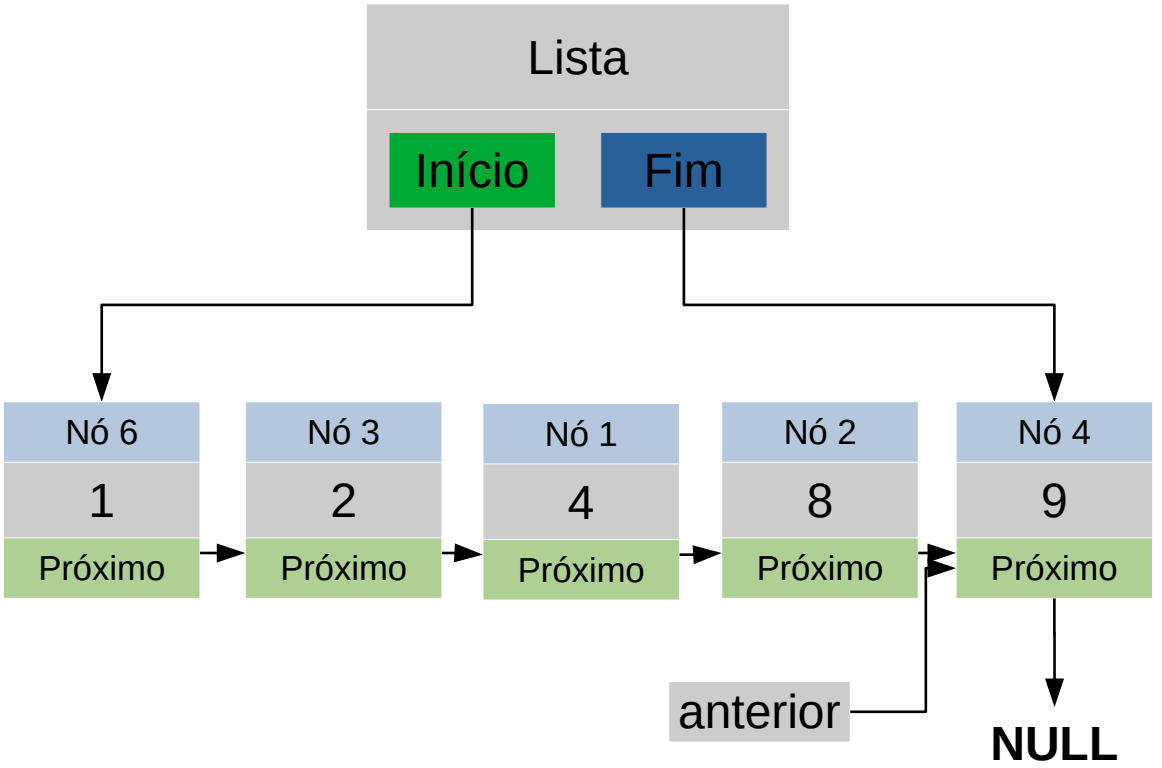


Passo 6. Remover elemento 10

removeNoFim():

```
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

tamanho = 6

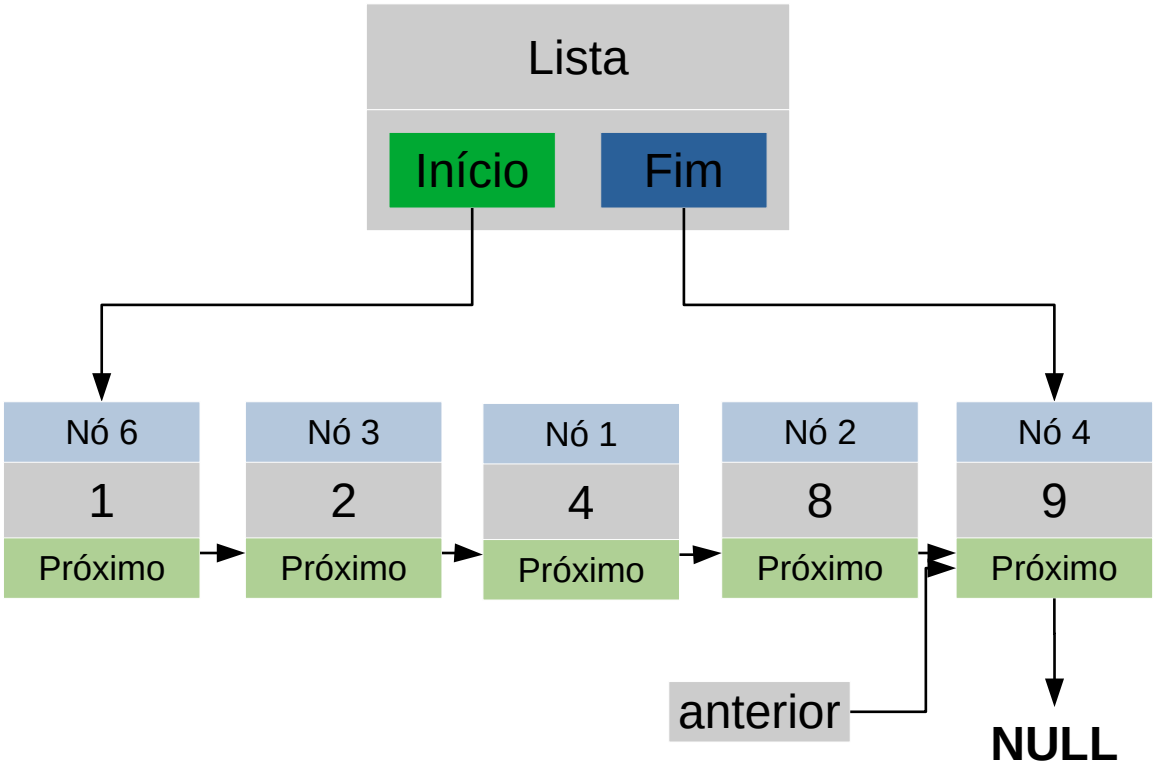


Passo 6. Remover elemento 10

removeNoFim():

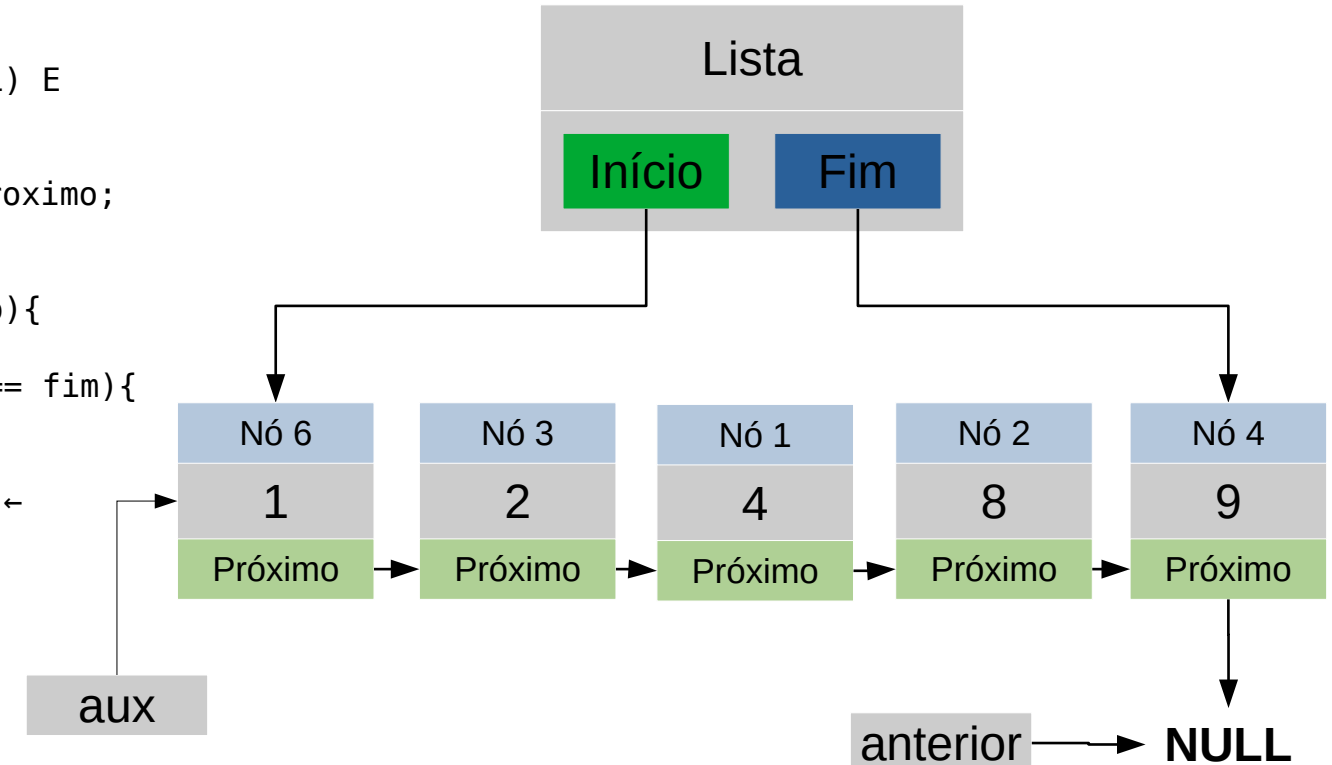
```
se listaVazia() sairComErro()

aux ← inicio;
anterior ← NULL
enquanto (aux.proximo ≠ NULL) {
      anterior ← aux;
      aux ← aux.proximo
}
valor ← aux.dado;
SE(anterior = NULO)
   primeiro ← NULO;
senão
   anterior.proximo ← NULO;
apagar(fim);
fim ← anterior
tamanho--;
```

tamanho = 5



Passo 7. Remover elemento 2

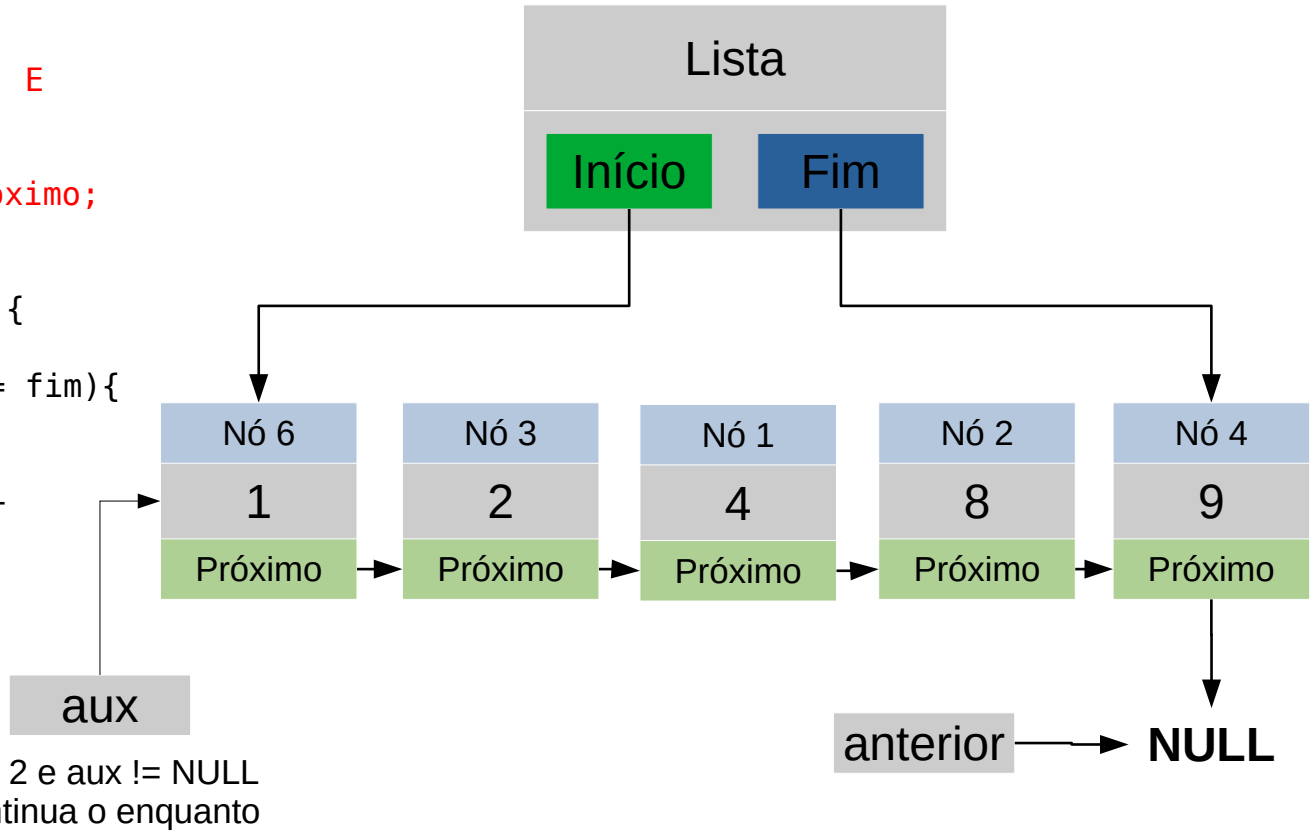
```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```



tamanho = 5

Passo 7. Remover elemento 2

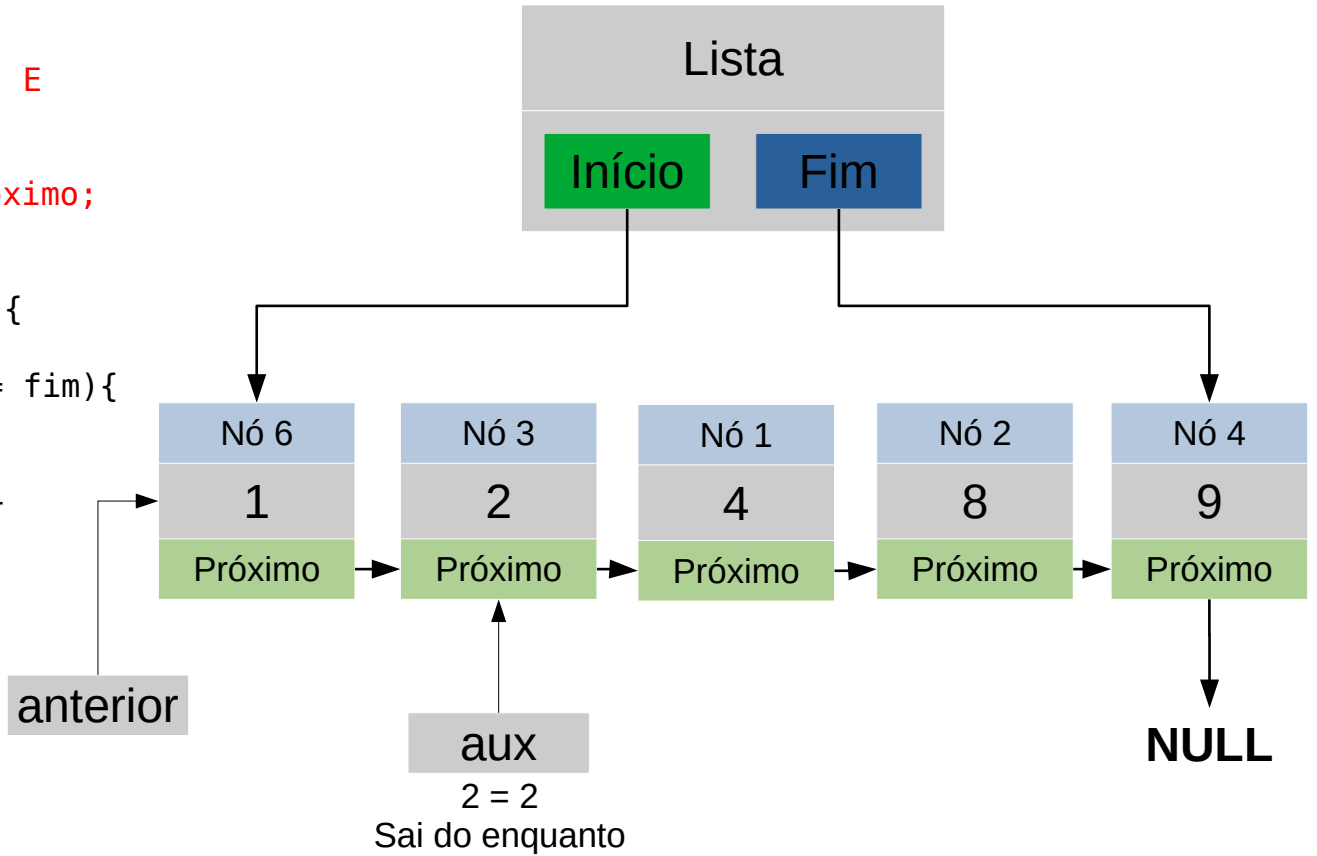
```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```



tamanho = 5

Passo 7. Remover elemento 2

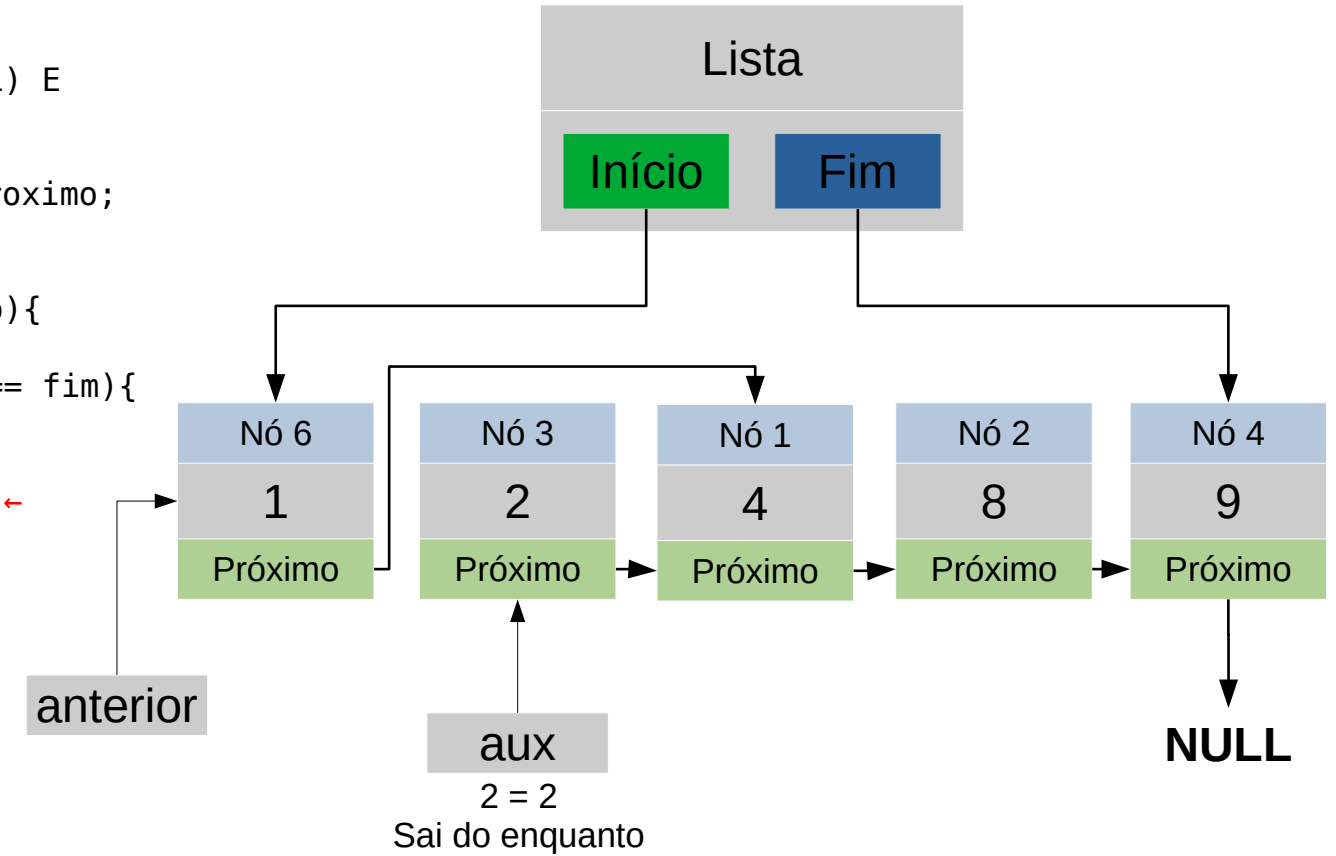
```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```



tamanho = 5

Passo 7. Remover elemento 2

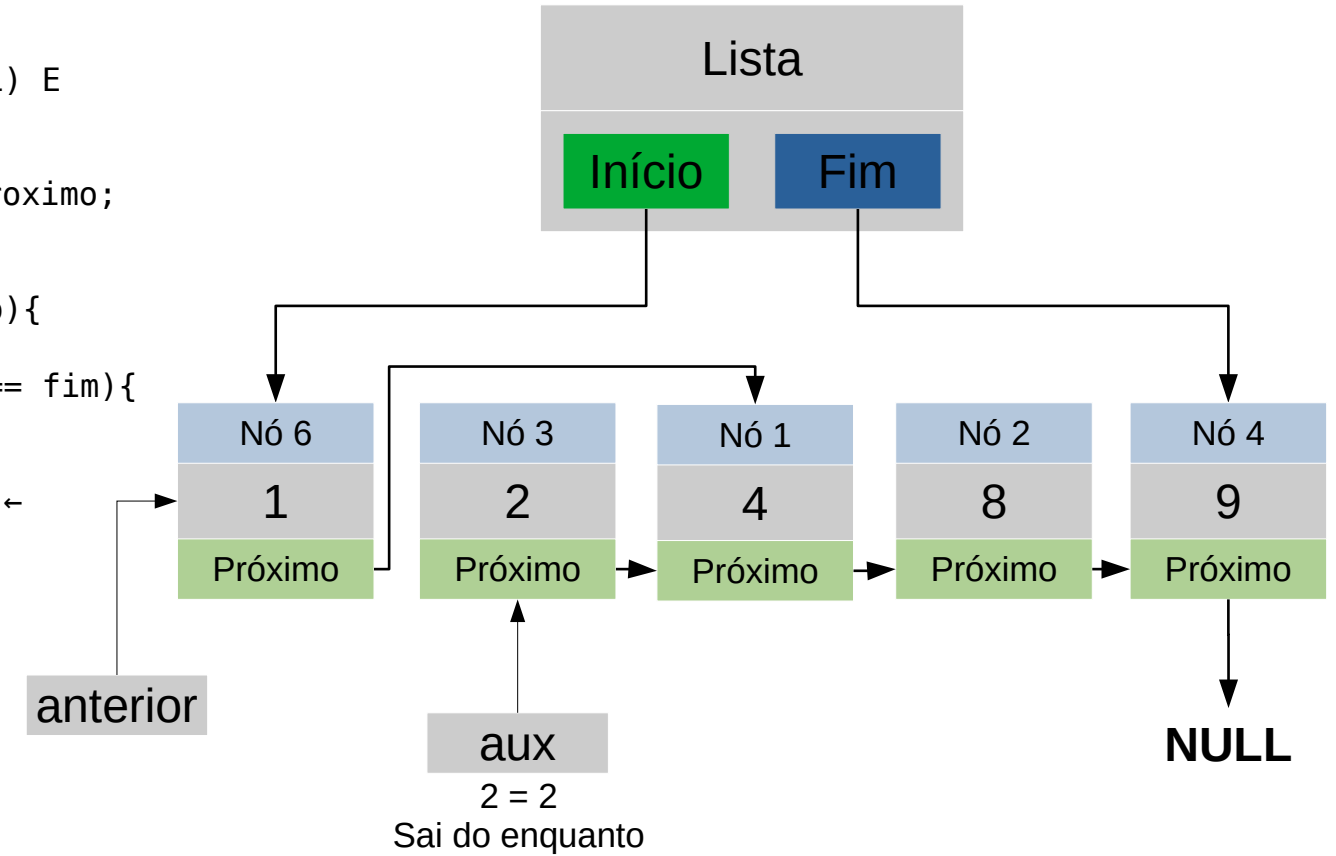
```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```



tamanho = 5

Passo 7. Remover elemento 2

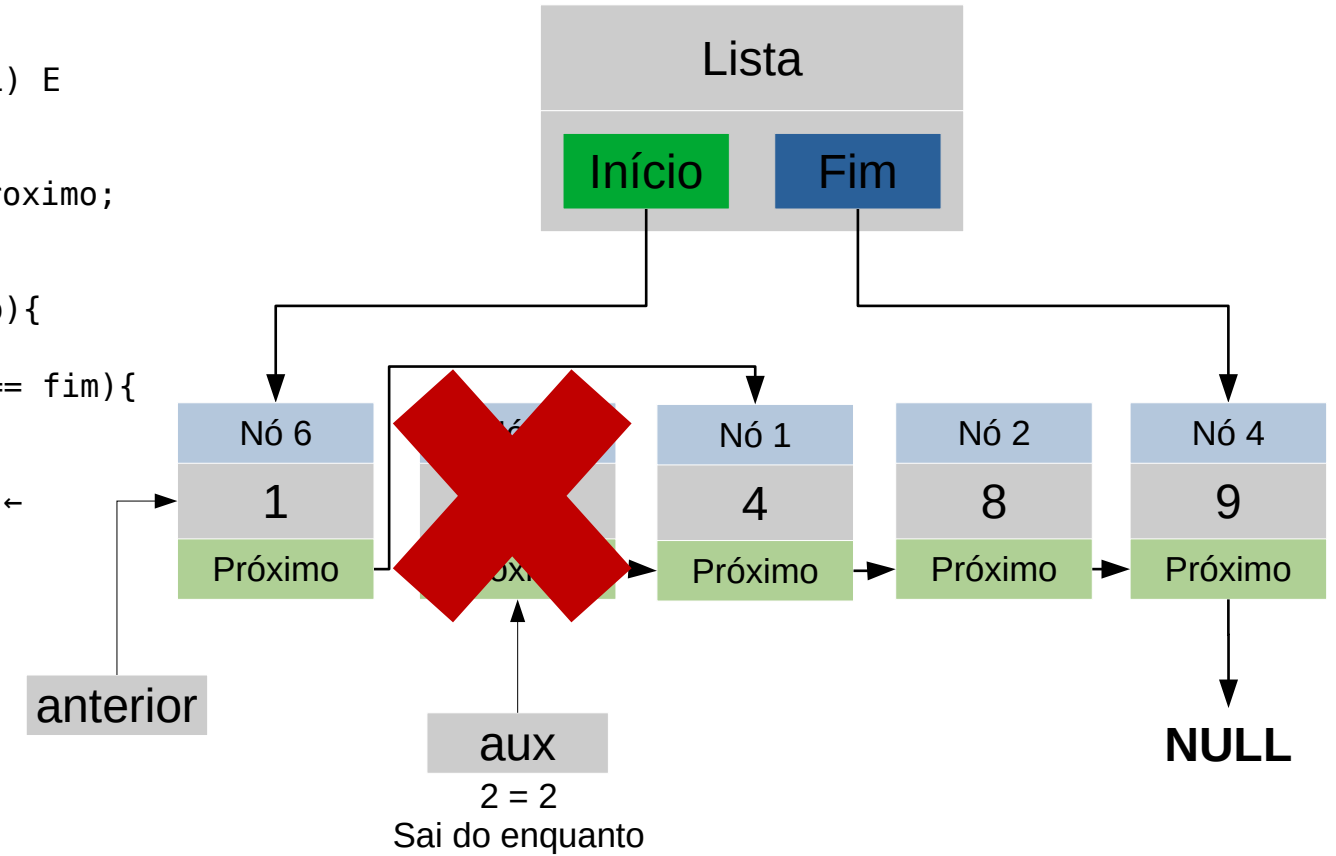
```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```



tamanho = 4

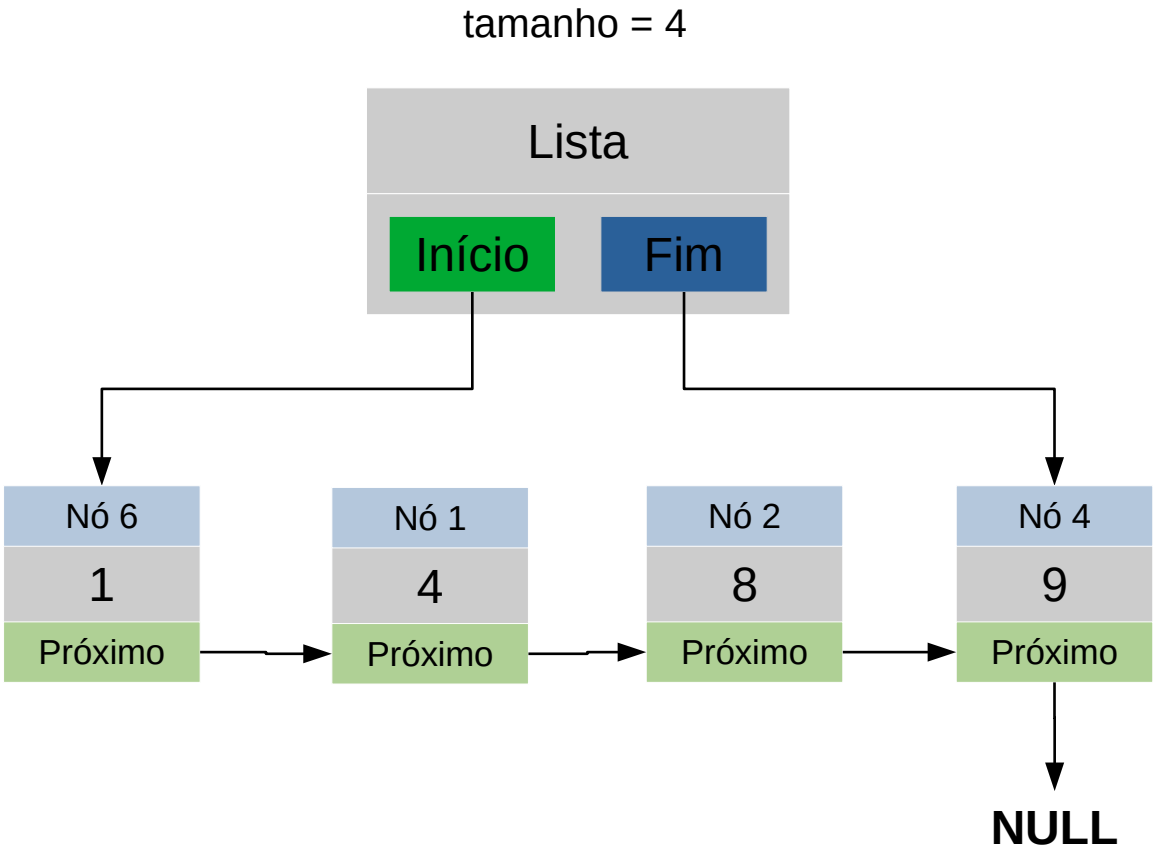
Passo 7. Remover elemento 2

```
auxiliar← inicio;
anterior ← null
enquanto ((auxiliar != NULL) E
(auxiliar.dado != valor)){
     anterior ← auxiliar;
     auxiliar ← auxiliar.proximo;
}
se (auxiliar != NULL){
     se (auxiliar == inicio){
          removeNoInicio();
     } senao se (auxiliar == fim){
          removeNoFim()
     } senao {
          anterior.proximo ←
     auxiliar.proximo;
          tamanho--;
          apaga(auxiliar);
     }
}
```



tamanho = 4

Resultado


tamanho = 4
Lista
Início
Fim
Nó 6
1
Próximo
Nó 1
4
Próximo
Nó 2
8
Próximo
Nó 4
9
Próximo
NULL