Estruturas de Dados Elementares: LISTAS

Profª. Rose Yuri Shimizu

Roteiro

Estrutura de Dados Elementares

- Estrutura de dados
  - Organizam uma coleção de dados
  - Possuem um conjunto de operações
- Elementar
  - Utilizados por outras estruturas
- Estrutura elementar: lista

Roteiro

LISTA ESTÁTICA

- Conjunto do **mesmo tipo** de dado
- Espaço **consecutivo** na memória RAM
- **Acesso aleatório**: qualquer posição pode ser acessada facilmente através de um **index**
- Nome $\rightarrow$ corresponde ao **endereço de memória**
- Tamanho **fixo** (stack) ou alocado **dinamicamente** (heap)
- Operações: `https://www.ime.usp.br/~pf/algoritmos/aulas/array.html`

LISTA ESTÁTICA

- VANTAGEM: fácil acesso
- DESVANTAGEM: difícil manipulação
- Alternativa: LISTAS ENCADEADAS
- `https://www.ime.usp.br/~pf/algoritmos/aulas/lista.html`
- `https://www.ime.usp.br/~pf/mac0122-2002/aulas/llists.html`

Roteiro

LISTA SIMPLESMENTE ENCADEADAS

- Conjunto de nós ou células
- Cada nó é tipo um contêiner que armazena **item + link (para outro nó)**

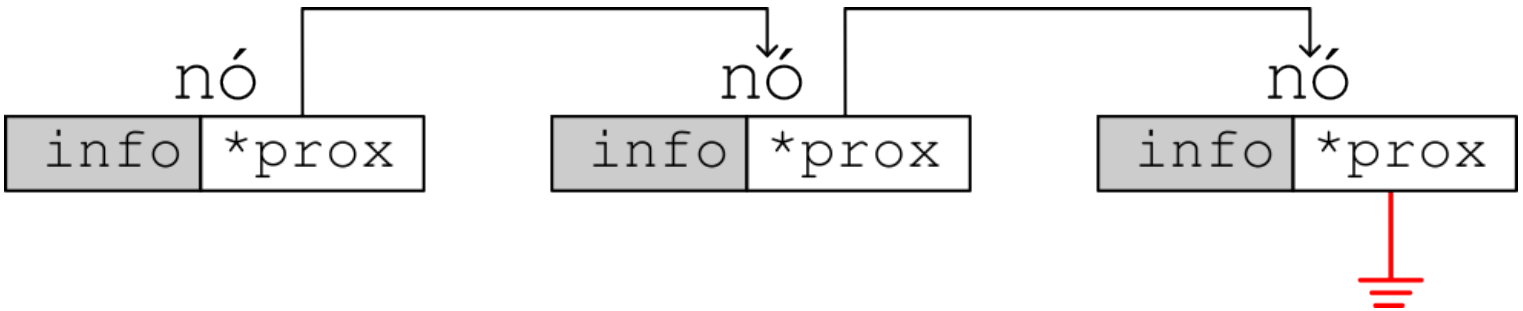
- Mais adequado para **manipulações** do que acessos:
  - Maior eficiência para **rearranjar os itens** (reapontamentos)
  - **Não tem acesso direto** aos itens
- Operações: buscar, inserir, remover

LISTA SIMPLESMENTE ENCADEADAS

Nós da lista

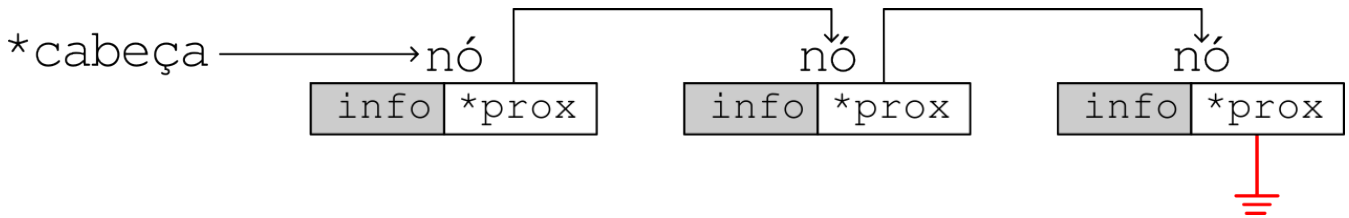
```c
//typedef struct item Item;
typedef int Item;
```

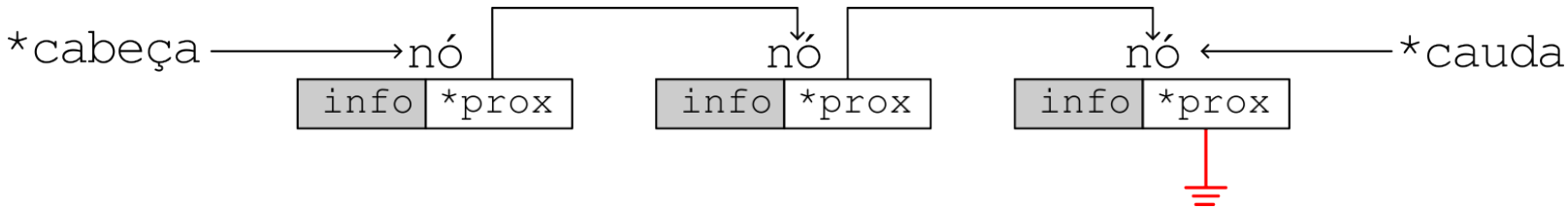
```c
typedef struct node no;
struct node {
    Item info;
    no *prox;
};
```



LISTA SIMPLESMENTE ENCADEADAS - Tipos

- Fim da lista: último nó aponta para NULL
- Início sem cabeça
  - Primeiro nó é o primeiro item da lista
  - Ponteiro (auxiliar) pode armazenar o endereço do primeiro nó
  - Com ou sem cauda

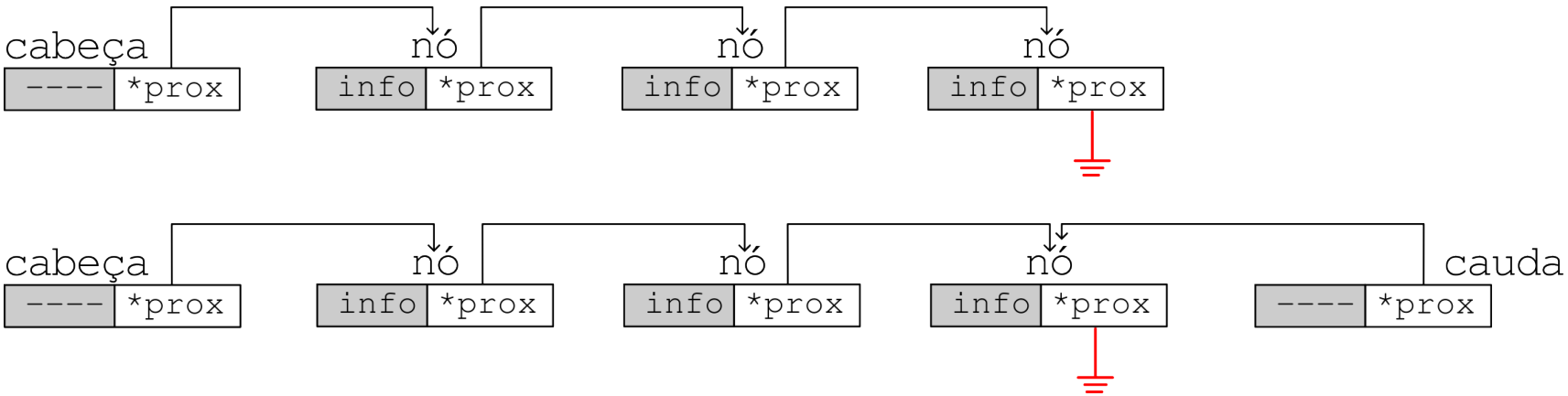





```
1 no *lista = NULL;
2
3 no *novo = malloc(sizeof(no));
4 novo->prox = NULL;
5 novo->info = 2;
6
7 lista = novo;
```

LISTA SIMPLESMENTE ENCADEADAS - Tipos

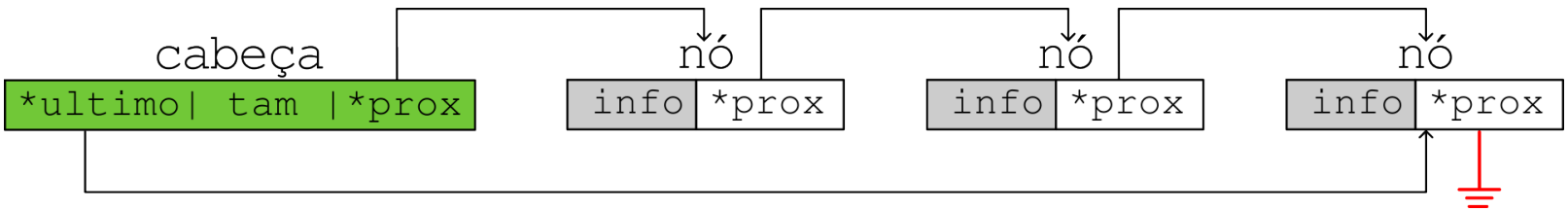
- Fim da lista: último nó aponta para NULL
- Início sem cabeça
- Início com cabeça do tipo **nó**
  - Conteúdo é ignorado
  - Elementos da lista: a partir do segundo nó
  - Com ou sem cauda




```
1 no *lista = malloc(sizeof(no));
2
3 no *novo = malloc(sizeof(no));
4 novo->prox = NULL;
5 novo->info = 2;
6
7 lista->prox = novo;
```

LISTA SIMPLESMENTE ENCADEADAS - Tipos

- Fim da lista: último nó aponta para NULL
- Início sem cabeça
- Início com cabeça do tipo **nó**
- Início com cabeça do tipo **cabeca** (específico)
  - Aproveita para guardar metadados
  - Tamanho da lista e fim da lista(cauda), por exemplo
  - Elementos da lista: a partir do segundo nó




```
1 typedef struct head cabeca;
2 struct head {
3     int tam;
4     no *prox;
5     no *ultimo;
6 };
7
```

LISTA SIMPLESMENTE ENCADEADAS

- Lista com tipo "cabeça"

```c
//nós da lista
typedef struct node no;
struct node no {
    Item info;
    node *prox;
};

//especifico para cabeca
typedef struct head cabeca
    ;
struct head {
    int tam;
    no *prox;
    no *ultimo;
};
```

```c
//novo elemento
no *novo = malloc(sizeof(no));
novo->prox = NULL;
novo->info = 2;

//criando a lista e
//  inserindo o novo elemento
cabeca *lista = malloc(sizeof(cabeca));
lista->tam = 1;
lista->prox = novo;
lista->ultimo = novo;
```

LISTA SIMPLESMENTE ENCADEADAS

- Implementado na STL (Standard Template Library) do C++
- Implementado na <sys/queue.h> da libc
- Algumas operações
  - Códigos na página da disciplina

LISTAS SIMPLESMENTE ENCADEADAS

1. Escreva uma função que conte o número de células de uma lista encadeada. Faça duas versões: uma iterativa e uma recursiva.

2. Escreva uma função que concatene duas listas encadeadas. Faça duas versões: uma iterativa e uma recursiva.

3. Escreva uma função que insira uma nova célula com conteúdo x imediatamente depois da k-ésima célula de uma lista encadeada. Faça duas versões: uma iterativa e uma recursiva.

4. Escreva uma função que troque de posição duas células de uma mesma lista encadeada.

LISTAS SIMPLESMENTE ENCADEADAS

1. **Altura**. A altura de uma célula c em uma lista encadeada é a **distância entre c e o fim da lista**. Escreva uma função que calcule a altura de uma dada célula.

2. **Profundidade**. A profundidade de uma célula c em uma lista encadeada é **distância entre o início da lista e c**. Escreva uma função que calcule a profundidade de uma dada célula.

3. Escreva uma função que inverta a ordem das células de uma lista encadeada (a primeira passa a ser a última, a segunda passa a ser a penúltima etc.). Faça isso sem usar espaço auxiliar, apenas alterando ponteiros. Dê duas soluções: uma iterativa e uma recursiva.

LISTAS SIMPLESMENTE ENCADEADAS

1. Escreva uma função que encontre uma célula com **conteúdo mínimo**. Faça duas versões: uma iterativa e uma recursiva.

2. Escreva uma função para remover de uma lista encadeada todas as células que contêm y.

3. Escreva uma função que verifique se **duas listas encadeadas são iguais**, ou melhor, se têm o mesmo conteúdo. Faça duas versões: uma iterativa e uma recursiva.

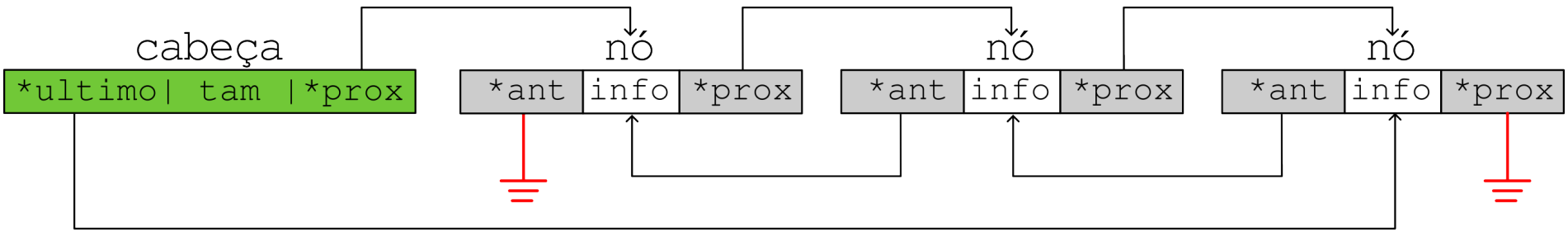
4. Listas de strings. Este exercício trata de listas encadeadas que contêm strings ASCII (cada célula contém uma string). Escreva uma função que verifique se uma lista desse tipo está em ordem lexicográfica. As células são do seguinte tipo:

```
1 typedef struct reg {
2     char *str; struct reg *prox;
3 } celula;
```

Roteiro

LISTA DUPLAMENTE ENCADEADAS

- Armazena a informação do nó **anterior** e **posterior**
- Útil quando ocorrem muitas inserções e remoções, principalmente de elementos intermediários
- Anterior do primeiro e posterior do último: NULL

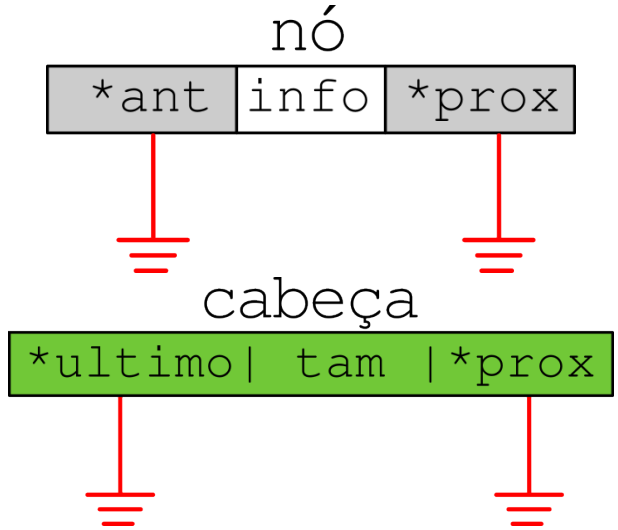


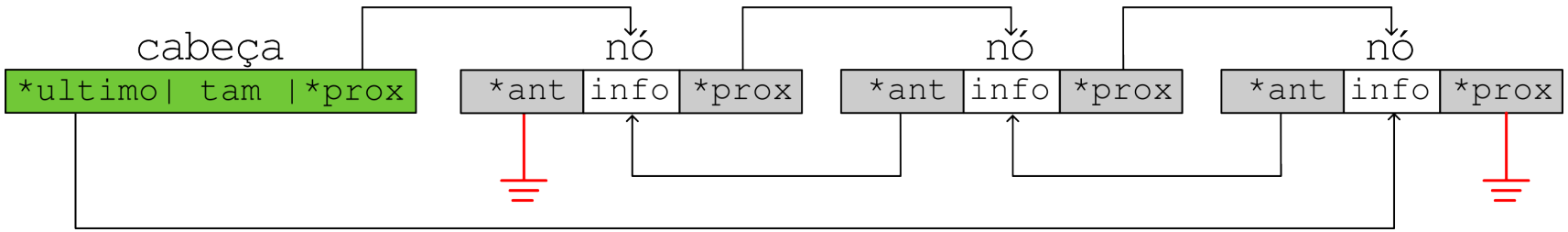
LISTA DUPLAMENTE ENCADEADAS

Lista com cabeça

```c
typedef struct node no;
struct node {
    Item info;
    no *ant; //<<<<
    no *prox;
};
```

```c
typedef struct head cabeca;
struct head {
    int tam;
    no *prox;
    no *ultimo;
};
```





LISTA DUPLAMENTE ENCADEADAS

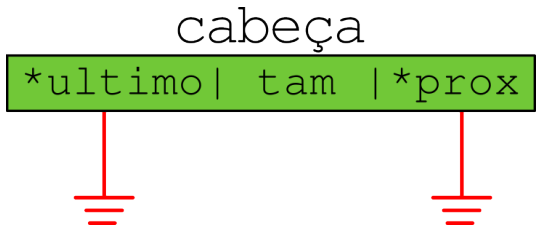
Operações

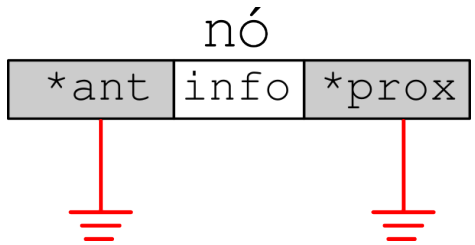
```
1  cabeca *criar();
2  no *criar_no(Item);
3
4  int vazia(cabeca*);
5  int tamanho(cabeca *);
6
7  no *inicio(cabeca *);
8  no *anterior(no *);
9  no *proximo(no *);
10 no *fim(cabeca *);
11
12 void insere_inicio(cabeca *, no *);
13 void insere_fim(cabeca *, no *);
14 void insere_depois(cabeca *, no *, no *);
15 void insere_antes(cabeca *, no *, no *);
16
17 void remove_no(cabeca *, no *);
```

- Implementado na STL (Standard Template Library) do C++
- Implementado na <sys/queue.h> da libc
- Algumas possibilidades de implementações serão apresentadas a seguir.

Criação

```
1  cabeca *criar()  {
2       cabeca *l = malloc(sizeof(cabeca));
3       l->tam = 0;
4       l->prox = NULL;
5       l->ultimo = NULL;
6
7       return l;
8  }
9
10 no *criar_no(Item x)  {
11      no *novo = malloc(sizeof(no));
12      novo->ant = NULL;  //<<<<<<<<<
13      novo->prox = NULL;
14      novo->info = x;
15
16      return novo;
17 }
18
```





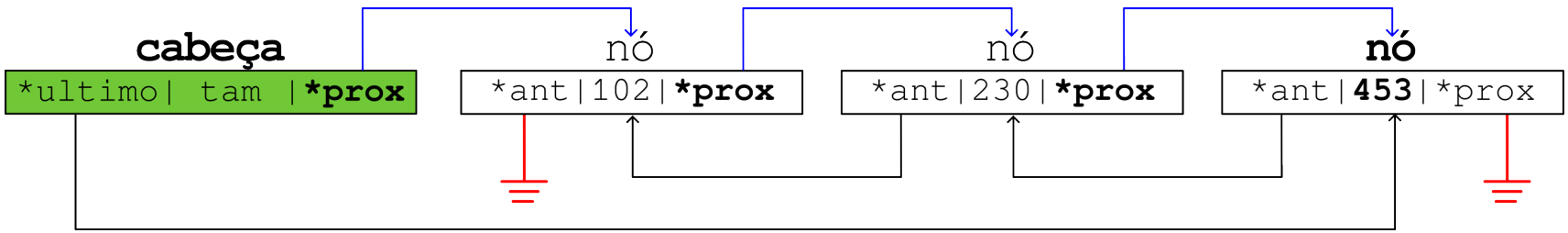

```
1  cabeca *lista = criar();
2
3  scanf("%d", &x);
4  no *elemento = criar_no(x);
```

Percorrer

```
1  int vazia(cabeca *lista) {
2      return (lista->prox==NULL);
3  }
4
5  int tamanho(cabeca *lista){
6      return (lista->tam);
7
8      //int tam = 0;
9      //for(no *a=lista->prox; a; a=a->prox, tam++);
10     //return tam;
11 }
12
13 no *inicio(cabeca *lista) {
14     return lista->prox;
15 }
16
17 no *fim(cabeca *lista) {
18     return lista->ultimo;
19
20     //no *a;
21     //for(a=lista->prox; a->prox; a=a->prox);
22     //return a;
23 }
```

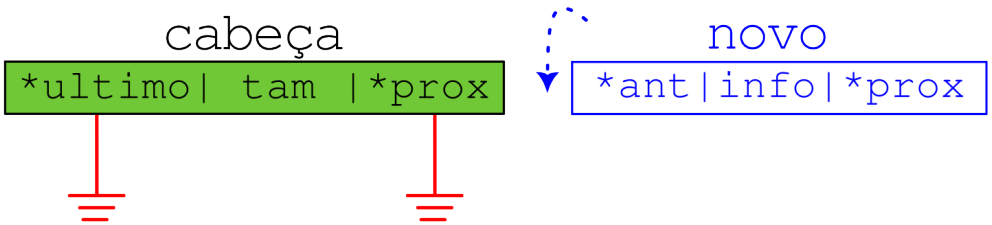
Percorrer

```
1  //Custo??
2  no *anterior(no *elem) {
3      return elem->ant;
4  }
5
6  //Custo??
7  no *proximo(no *elem) {
8      return elem->prox;
9  }
10
11 //Custo??
12 no *busca(cabeca *lista, Item x){
13     no *a;
14     for(a=lista->prox; a && a->info!=x; a=a->prox);
15     return a;
16 }
17
18 busca(lista, 453);
19
```

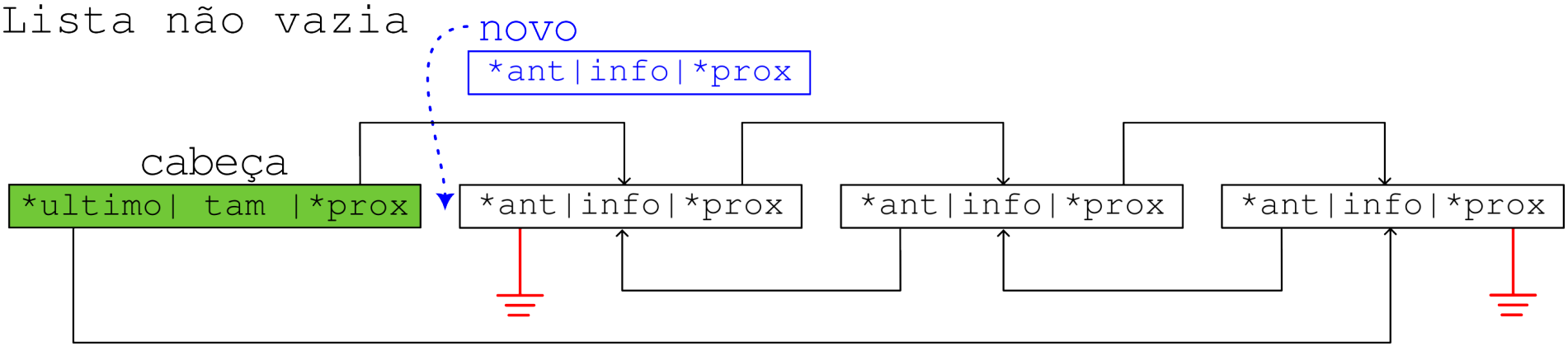


Inserção no início

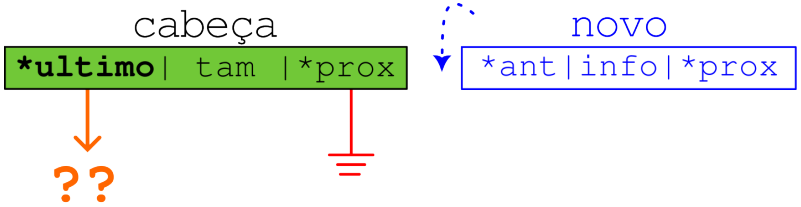
Lista vazia

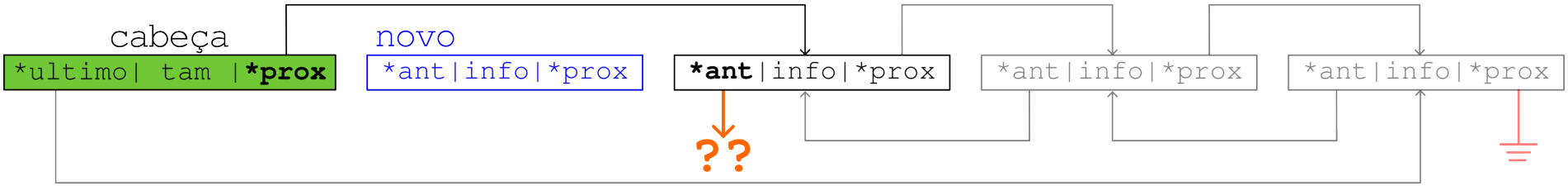


Lista não vazia



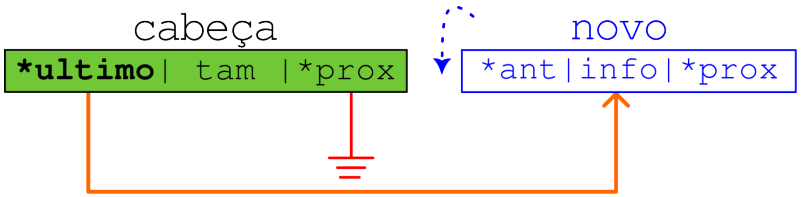
Inserção no início

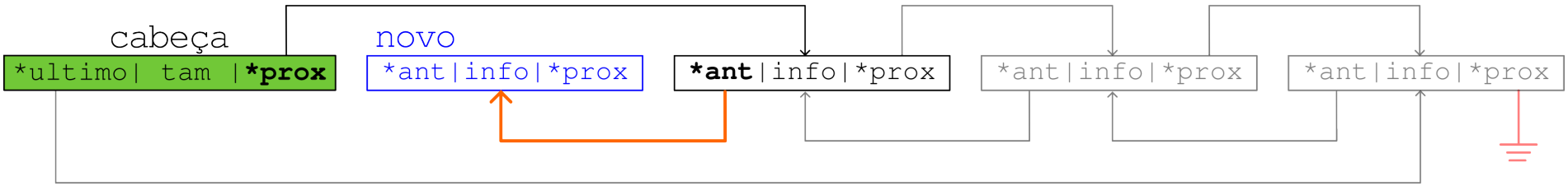





```
1  void insere_inicio(cabeca *lista, no *novo){
2  {
3      if(vazia(lista)) lista->ultimo = ??
4      else lista->prox->ant = ??
5
6
7
8
9
10
11
12 }
13
```

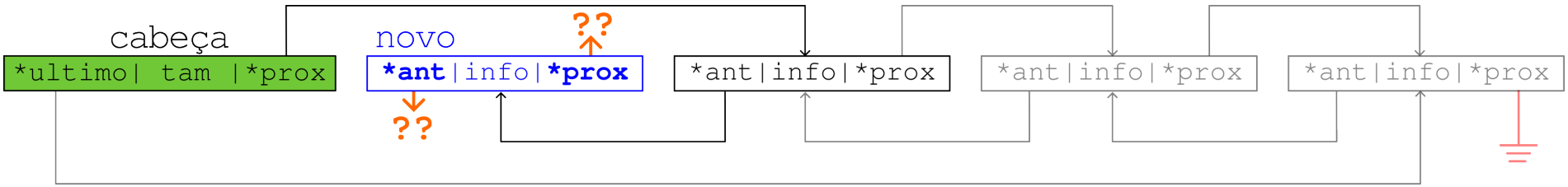
Inserção no início





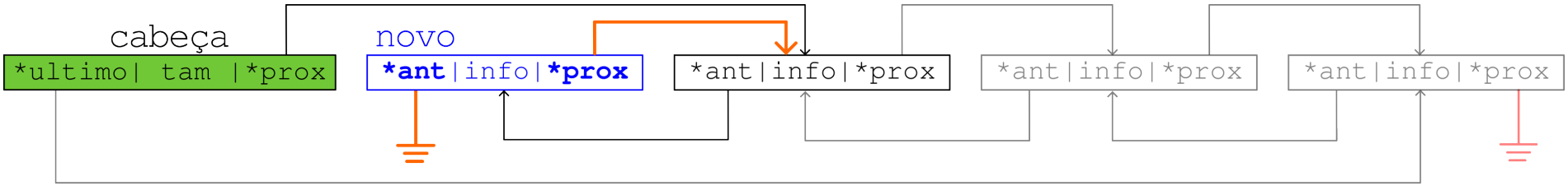

```
1  void insere_inicio(cabeca *lista, no *novo){
2      if(vazia(lista)) lista->ultimo = novo;
3      else lista->prox->ant = novo;
4
5
6
7
8
9
10
11 }
12
```

Inserção no início



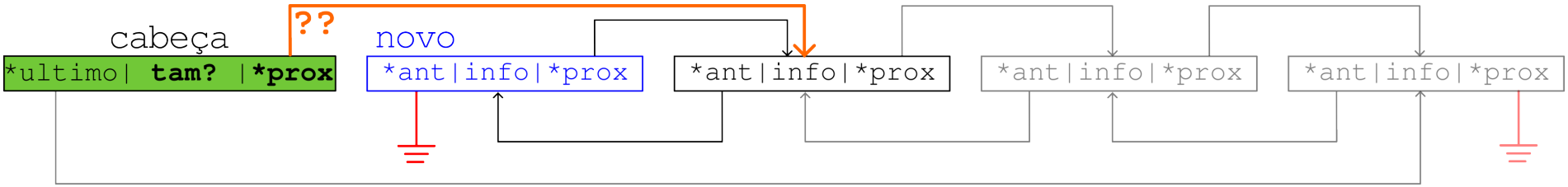

```
1  void insere_inicio(cabeca *lista, no *novo){
2      if(vazia(lista)) lista->ultimo = novo;
3      else lista->prox->ant = novo;
4
5      novo->ant = ??
6      novo->prox = ??
7
8
9
10
11 }
12
```

Inserção no início



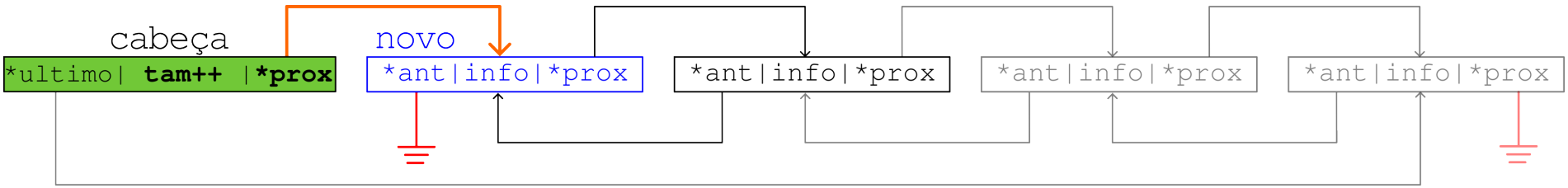

```
1  void insere_inicio(cabeca *lista, no *novo){
2      if(vazia(lista)) lista->ultimo = novo;
3      else lista->prox->ant = novo;
4
5      novo->ant = NULL;
6      novo->prox = lista->prox;
7
8
9
10
11 }
12
```

Inserção no início



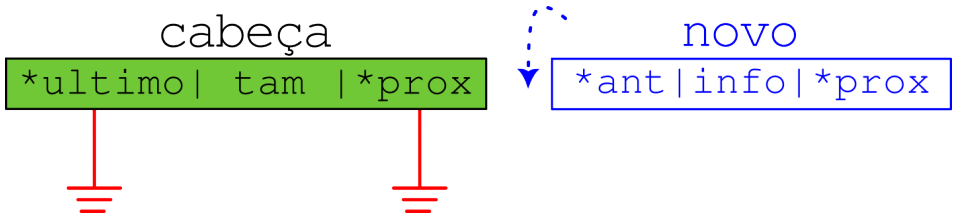

```
1  void insere_inicio(cabeca *lista, no *novo){
2      if(vazia(lista)) lista->ultimo = novo;
3      else lista->prox->ant = novo;
4
5      novo->ant = NULL;
6      novo->prox = lista->prox;
7
8      lista->prox = ??
9
10
11 }
12
```

Inserção no início



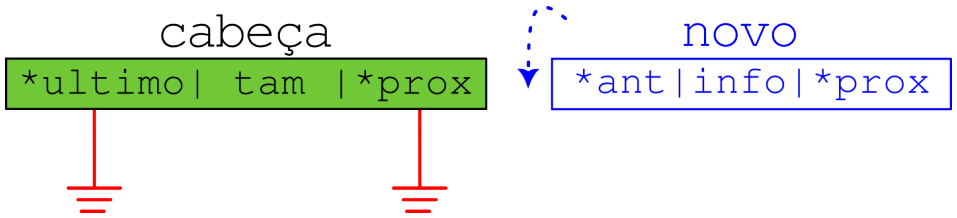

```
1  void insere_inicio(cabeca *lista, no *novo){
2      if(vazia(lista)) lista->ultimo = novo;
3      else lista->prox->ant = novo;
4
5      novo->ant = NULL;
6      novo->prox = lista->prox;
7
8      lista->prox = novo;
9      lista->tam++;
10
11 }
12
```

Inserção no fim



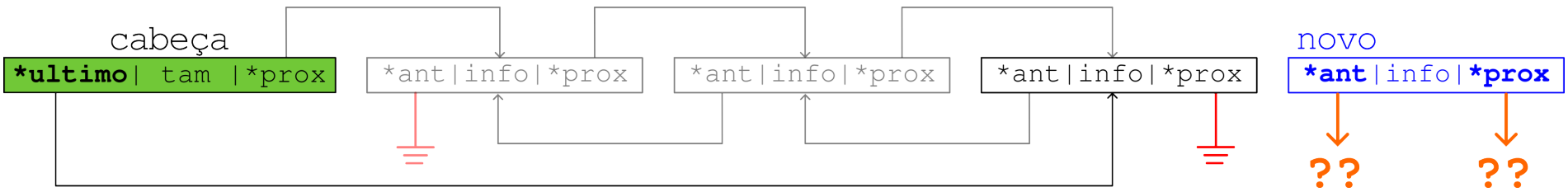

```
1  void insere_fim(cabeca *lista, no *novo){
2      if(vazia(lista)) ??
3
4
5
6
7
8
9
10
11 }
12
```

Inserção no fim



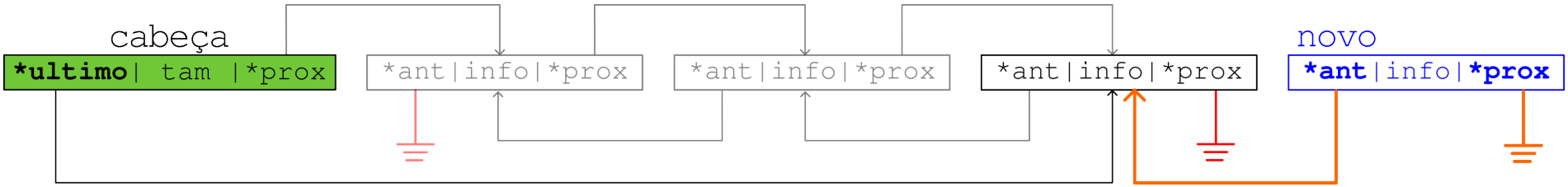

```
1  void insere_fim(cabeca *lista, no *novo){
2      if(vazia(lista)) return insere_inicio(lista, novo);
3
4
5
6
7
8
9
10
11 }
12
```

Inserção no fim



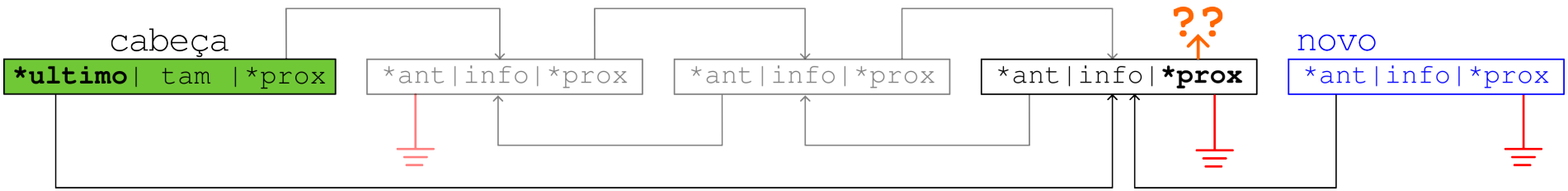

```
1  void insere_fim(cabeca *lista, no *novo){
2      if(vazia(lista)) return insere_inicio(lista, novo);
3
4      novo->ant = ??
5      novo->prox = ??
6
7
8
9
10
11 }
12
```

Inserção no fim



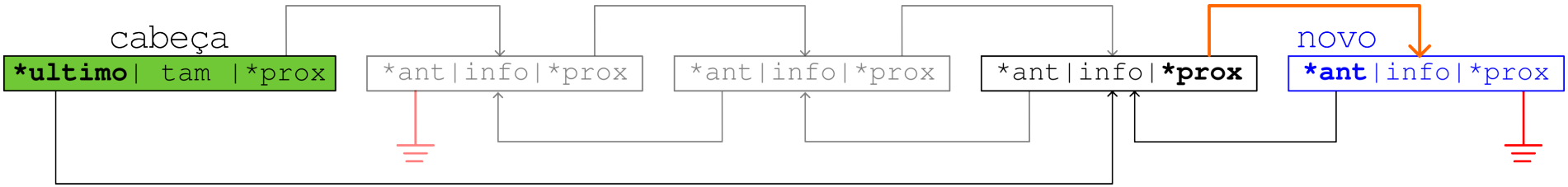

```
1  void insere_fim(cabeca *lista, no *novo){
2      if(vazia(lista)) return insere_inicio(lista, novo);
3
4      novo->ant = lista->ultimo;
5      novo->prox = NULL;
6
7
8
9
10
11 }
12
```

Inserção no fim



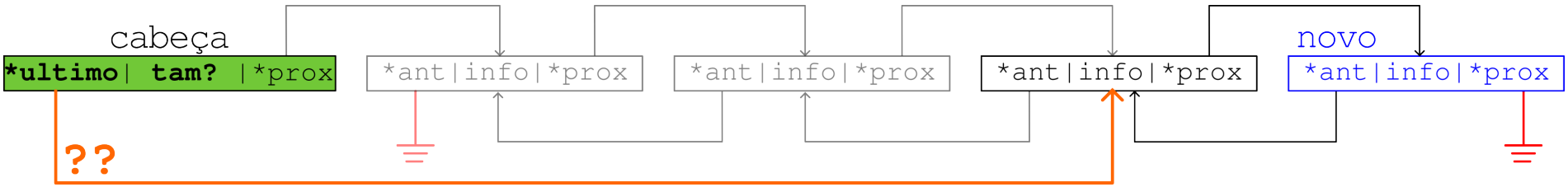

```
1  void insere_fim(cabeca *lista, no *novo){
2      if(vazia(lista)) return insere_inicio(lista, novo);
3
4      novo->ant = lista->ultimo;
5      novo->prox = NULL;
6
7      lista->ultimo->prox = ??
8
9
10
11 }
12
```

Inserção no fim



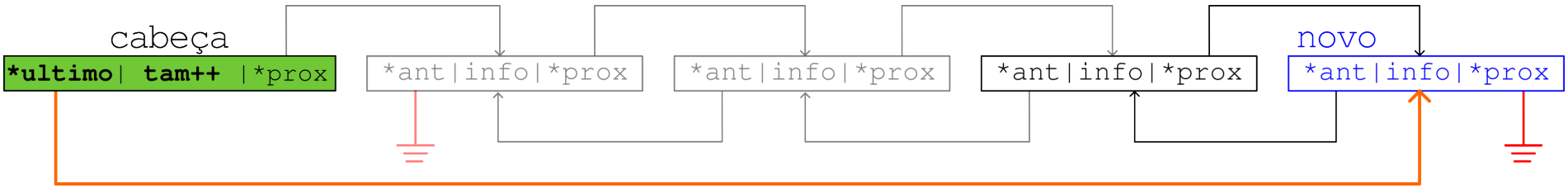

```
1  void insere_fim(cabeca *lista, no *novo){
2      if(vazia(lista)) return insere_inicio(lista, novo);
3
4      novo->ant = lista->ultimo;
5      novo->prox = NULL;
6
7      lista->ultimo->prox = novo;
8
9
10
11 }
12
```

Inserção no fim




```
1  void insere_fim(cabeca *lista, no *novo){
2      if(vazia(lista)) return insere_inicio(lista, novo);
3
4      novo->ant = lista->ultimo;
5      novo->prox = NULL;
6
7      lista->ultimo->prox = novo;
8      lista->ultimo = ??
9
10
11 }
12
```

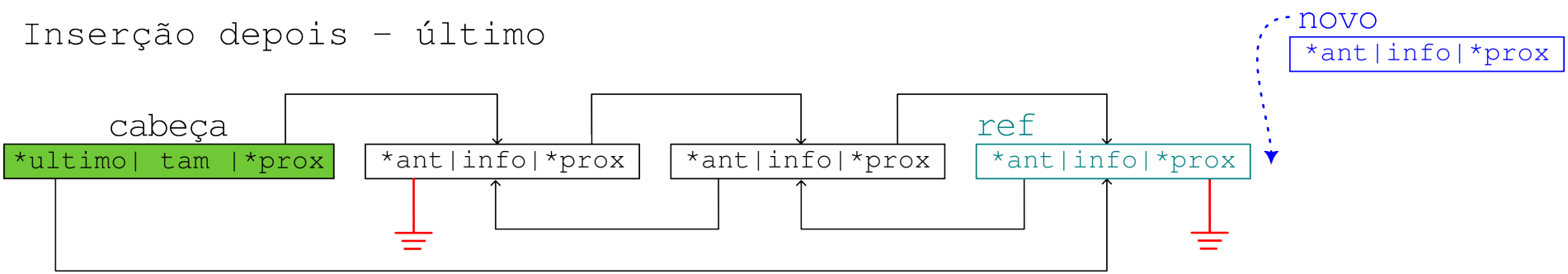
Inserção no fim



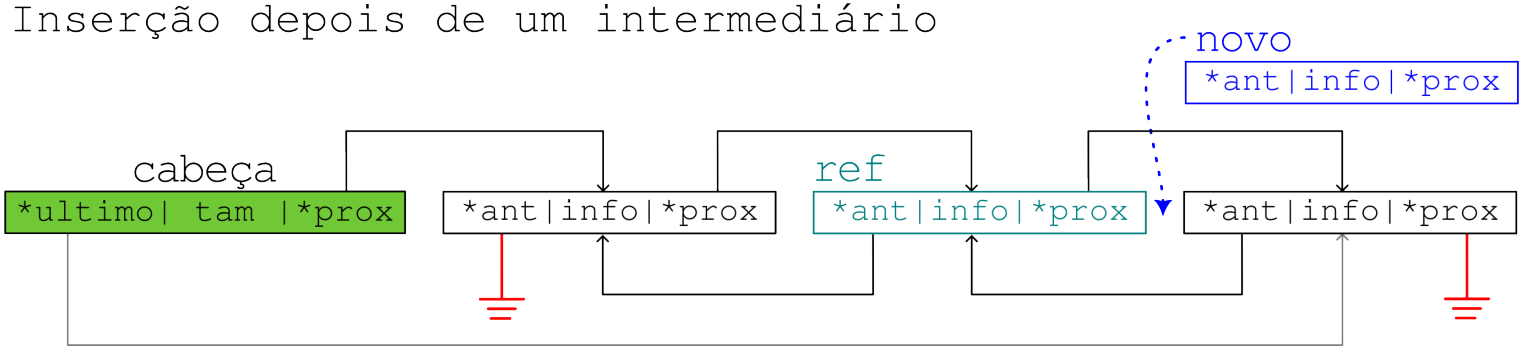

```
1  void insere_fim(cabeca *lista, no *novo){
2      if(vazia(lista)) return insere_inicio(lista, novo);
3
4      novo->ant = lista->ultimo;
5      novo->prox = NULL;
6
7      lista->ultimo->prox = novo;
8      lista->ultimo = novo;
9      lista->tam++;
10
11 }
12
```

Inserção depois de um nó

Inserção depois - último

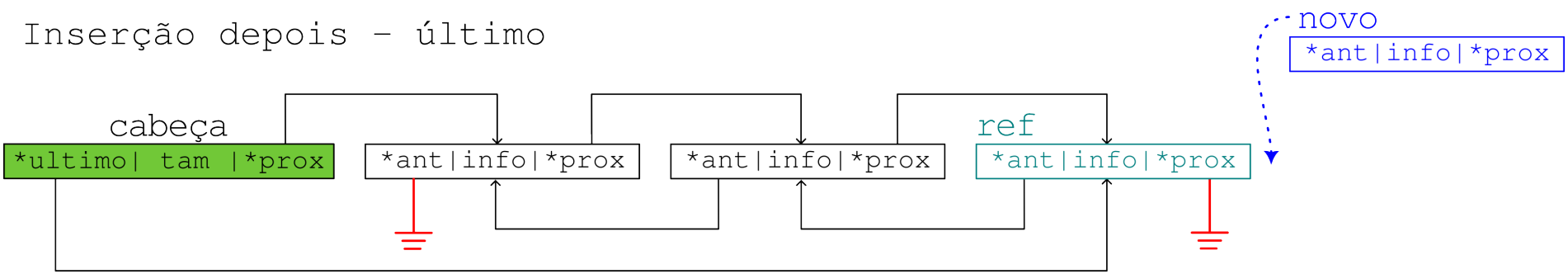


Inserção depois de um intermediário



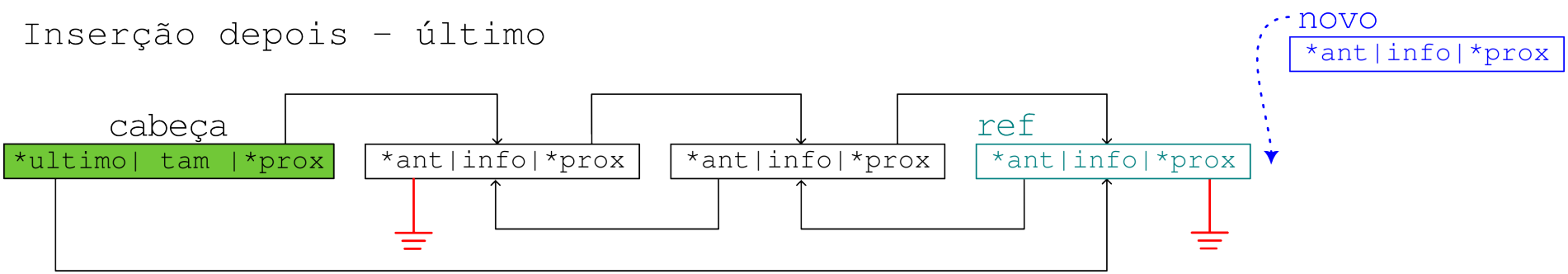
Inserção depois de um nó

Inserção depois - último



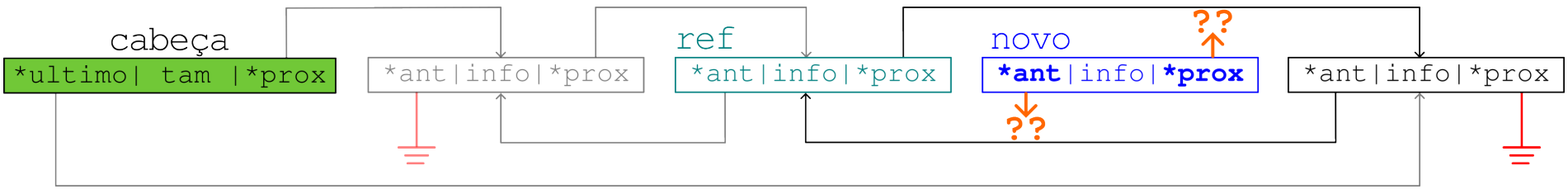

```
1  void insere_depois(cabeca *lista, no *ref, no *novo)
2  {
3      if(lista->ultimo == ref) ??
4
5
6
7
8
9
10
11
12
13 }
14
```

Inserção depois de um nó



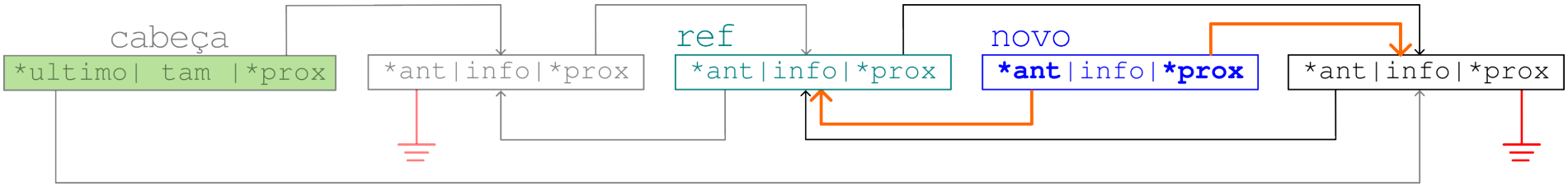

```
1  void insere_depois(cabeca *lista, no *ref, no *novo)
2  {
3      if(lista->ultimo == ref)
4          return insere_fim(lista, novo);
5
6
7
8
9
10
11
12
13 }
14
```

Inserção depois de um nó



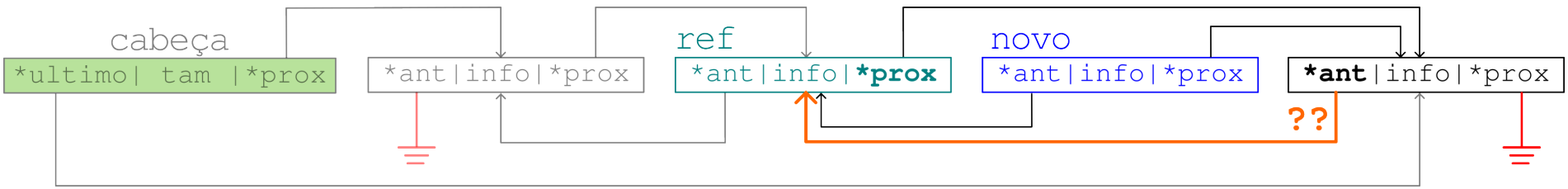

```
1  void insere_depois(cabeca *lista, no *ref, no *novo)
2  {
3      if(lista->ultimo == ref)
4          return insere_fim(lista, novo);
5
6      novo->ant = ??
7      novo->prox = ??
8
9
10
11
12
13 }
14
```

Inserção depois de um nó



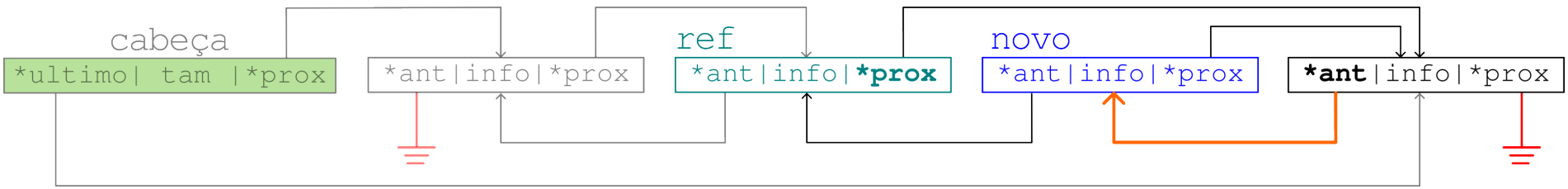

```
1  void insere_depois(cabeca *lista, no *ref, no *novo)
2  {
3      if(lista->ultimo == ref)
4          return insere_fim(lista, novo);
5
6      novo->ant = ref;
7      novo->prox = ref->prox;
8
9
10
11
12
13 }
14
```

Inserção depois de um nó



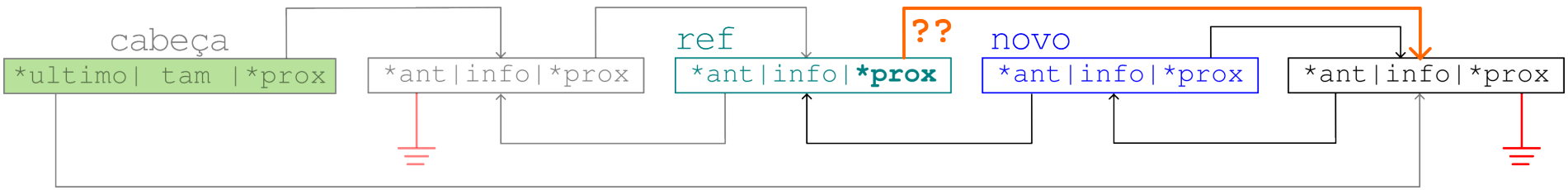

```
1  void insere_depois(cabeca *lista, no *ref, no *novo)
2  {
3      if(lista->ultimo == ref)
4          return insere_fim(lista, novo);
5
6      novo->ant = ref;
7      novo->prox = ref->prox;
8
9      ref->prox->ant = ??
10
11
12
13 }
14
```

Inserção depois de um nó



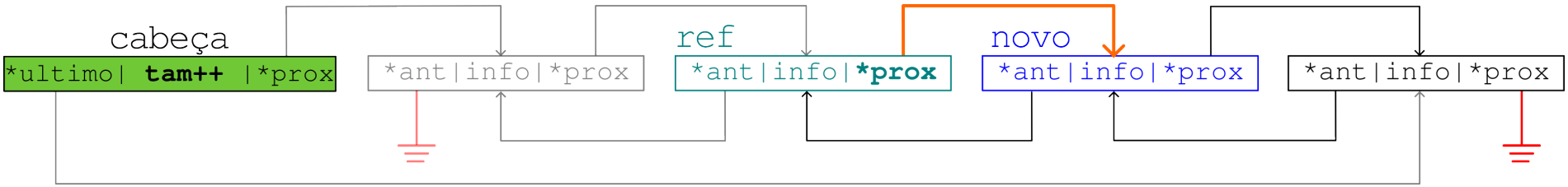

```
1  void insere_depois(cabeca *lista, no *ref, no *novo)
2  {
3      if(lista->ultimo == ref)
4          return insere_fim(lista, novo);
5
6      novo->ant = ref;
7      novo->prox = ref->prox;
8
9      ref->prox->ant = novo;
10
11
12
13 }
14
```

Inserção depois de um nó




```
1  void insere_depois(cabeca *lista, no *ref, no *novo)
2  {
3      if(lista->ultimo == ref)
4          return insere_fim(lista, novo);
5
6      novo->ant = ref;
7      novo->prox = ref->prox;
8
9      ref->prox->ant = novo;
10     ref->prox = ??
11
12
13 }
14
```

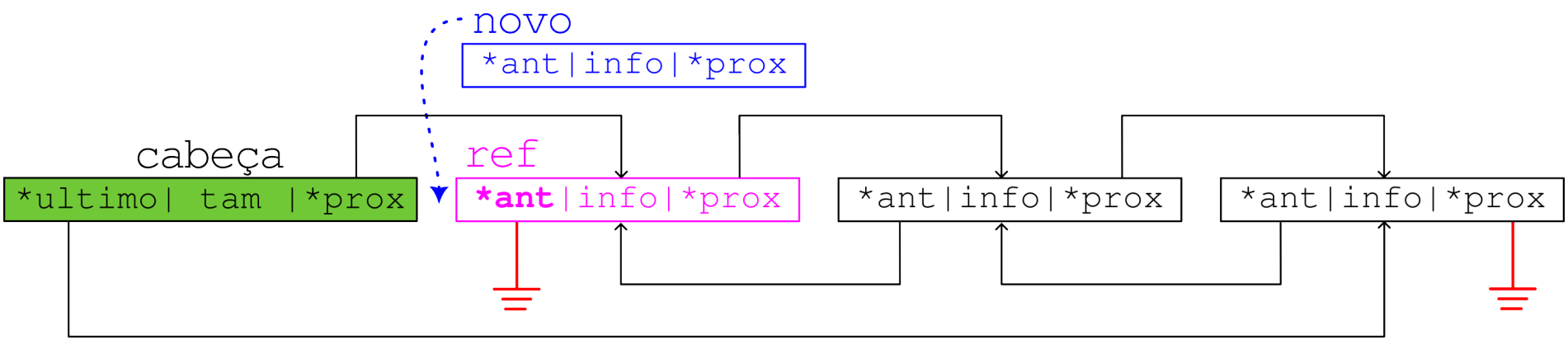
Inserção depois de um nó




```
1  void insere_depois(cabeca *lista, no *ref, no *novo)
2  {
3      if(lista->ultimo == ref)
4          return insere_fim(lista, novo);
5
6      novo->ant = ref;
7      novo->prox = ref->prox;
8
9      ref->prox->ant = novo;
10     ref->prox = novo;
11
12     lista->tam++;
13 }
14
```

Inserção antes de um nó

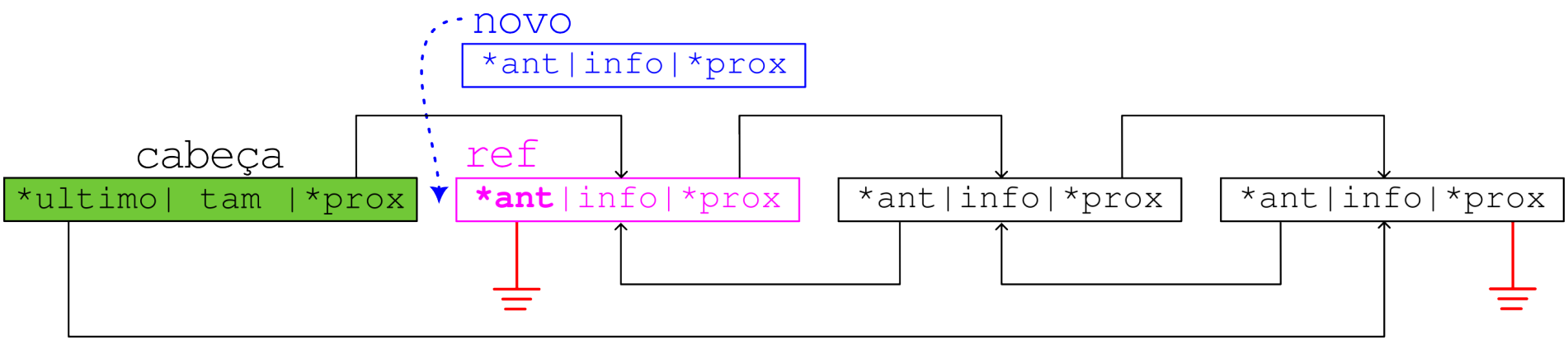
Inserção antes - primeiro




```
1 void insere_antes(cabeca *lista, no *ref, no *novo)
2 {
3     if(lista->prox == ref) ??
4
5
6
7 }
8
```

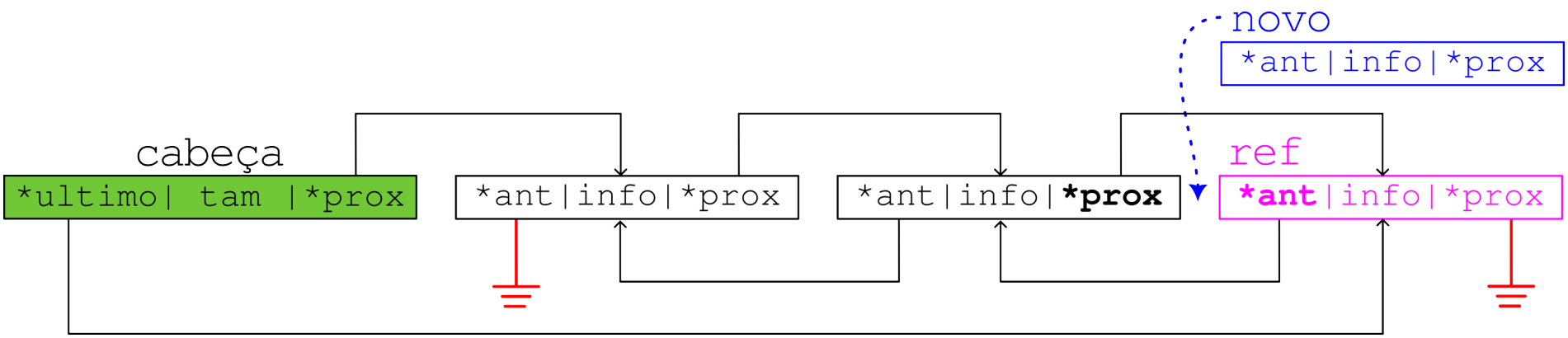
Inserção antes de um nó

Inserção antes - primeiro



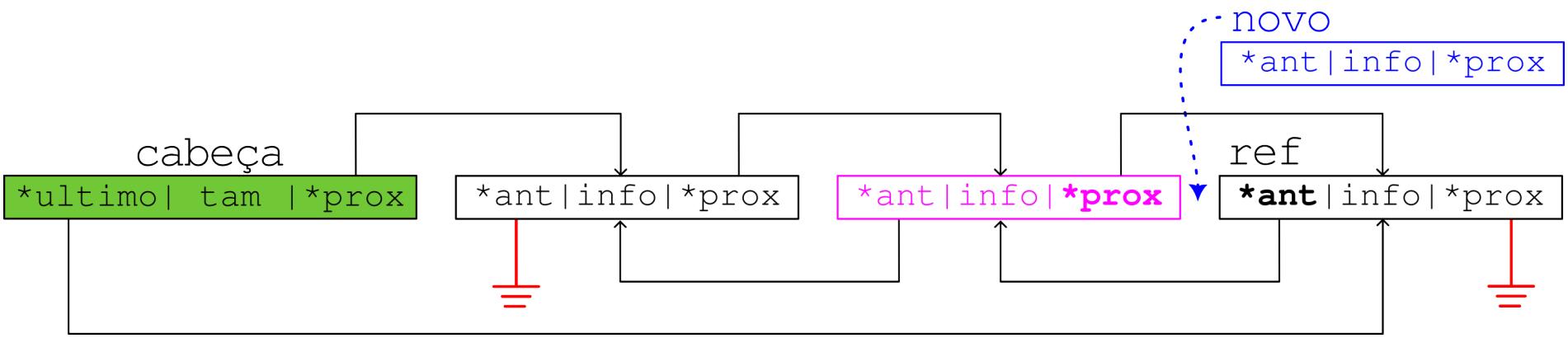

```
1  void insere_antes(cabeca *lista, no *ref, no *novo)
2  {
3      if(lista->prox == ref)
4          return insere_inicio(lista, novo);
5
6
7  }
8
```

Inserção antes de um nó



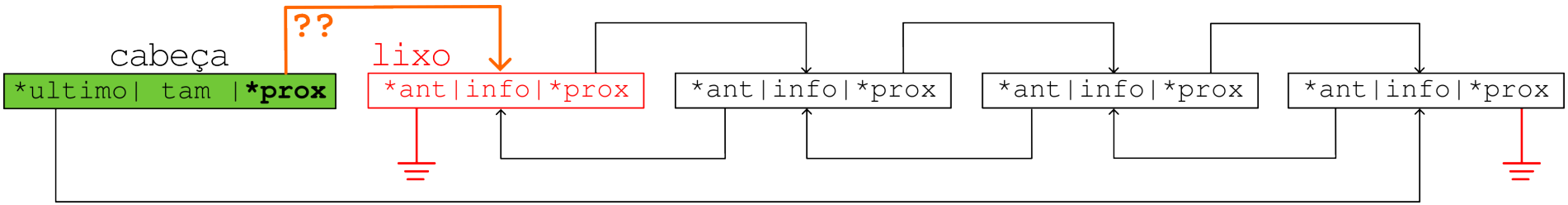

```
1 void insere_antes(cabeca *lista, no *ref, no *novo)
2 {
3     if(lista->prox == ref)
4         return insere_inicio(lista, novo);
5
6     ??
7 }
8
```

Inserção antes de um nó



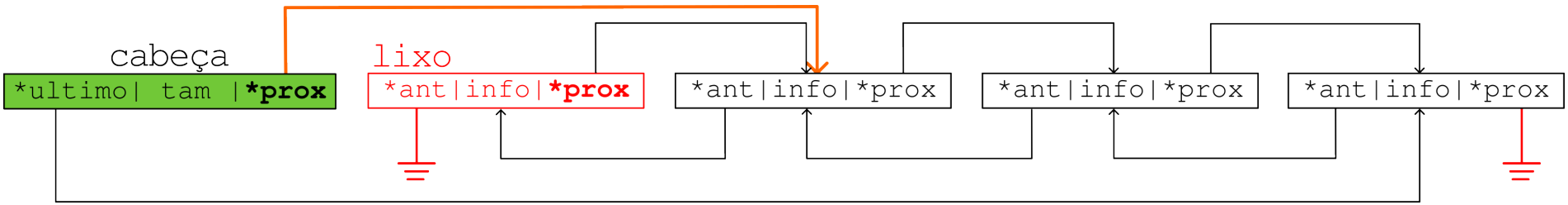

```
1 void insere_antes(cabeca *lista, no *ref, no *novo)
2 {
3     if(lista->prox == ref)
4         return insere_inicio(lista, novo);
5
6     return insere_depois(lista, ref->ant, novo);
7 }
8
```

Remoção



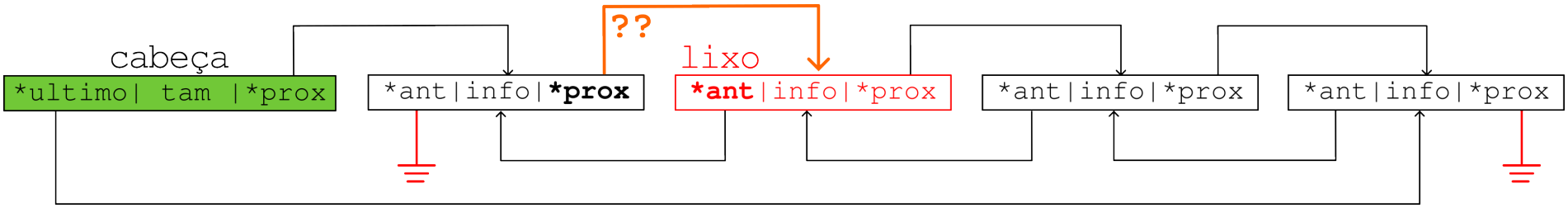

```
1  void remove_no(cabeca *lista, no *lixo)
2  {
3
4      if(lista->prox == lixo) lista->prox = ??
5
6
7
8
9
10
11
12 }
13
```

Remoção



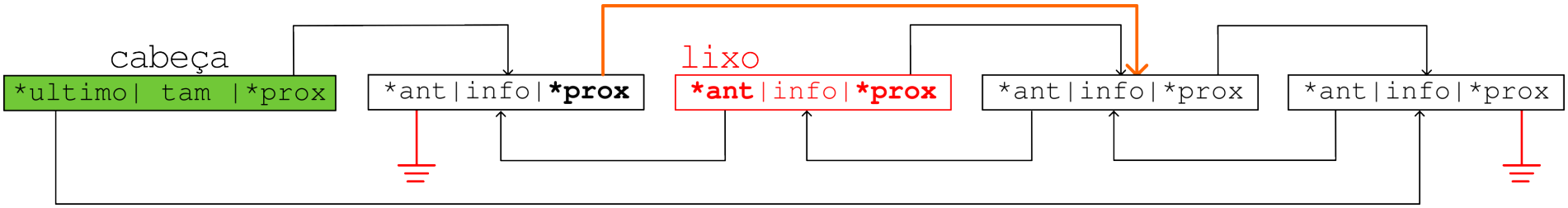

```
1  void remove_no(cabeca *lista, no *lixo)
2  {
3
4      if(lista->prox == lixo) lista->prox = lixo->prox;
5
6
7
8
9
10
11
12 }
13
```

Remoção



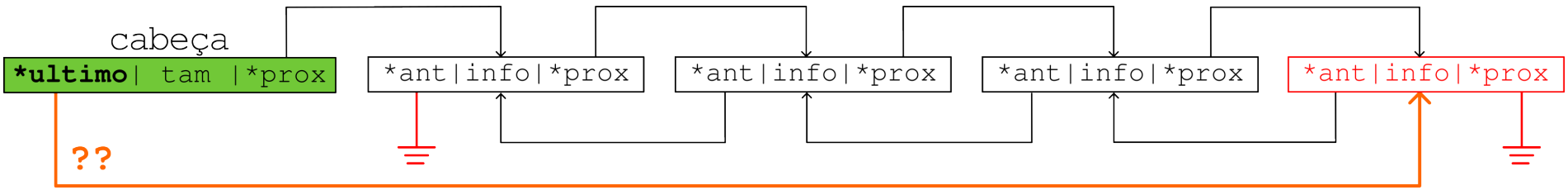

```
1  void remove_no(cabeca *lista, no *lixo)
2  {
3
4      if(lista->prox == lixo) lista->prox = lixo->prox;
5      else lixo->ant->prox = ??
6
7
8
9
10
11
12 }
13
```

Remoção



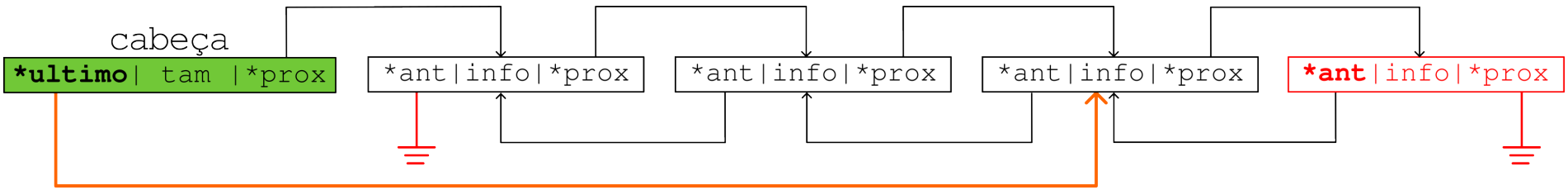

```
1  void remove_no(cabeca *lista, no *lixo)
2  {
3
4      if(lista->prox == lixo) lista->prox = lixo->prox;
5      else lixo->ant->prox = lixo->prox;
6
7
8
9
10
11
12 }
13
```

Remoção



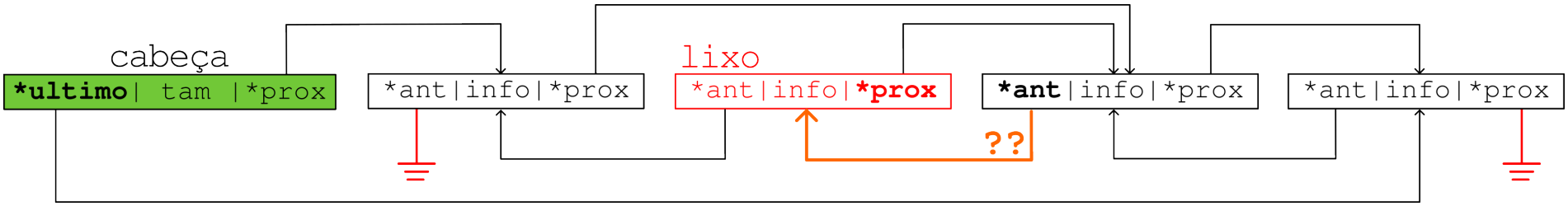

```
1  void remove_no(cabeca *lista, no *lixo)
2  {
3
4      if(lista->prox == lixo) lista->prox = lixo->prox;
5      else lixo->ant->prox = lixo->prox;
6
7      if(lista->ultimo == lixo) lista->ultimo = ??
8
9
10
11
12 }
13
```

Remoção




```
1  void remove_no(cabeca *lista, no *lixo)
2  {
3
4      if(lista->prox == lixo) lista->prox = lixo->prox;
5      else lixo->ant->prox = lixo->prox;
6
7      if(lista->ultimo == lixo) lista->ultimo = lixo->ant;
8
9
10
11
12 }
13
```

Remoção

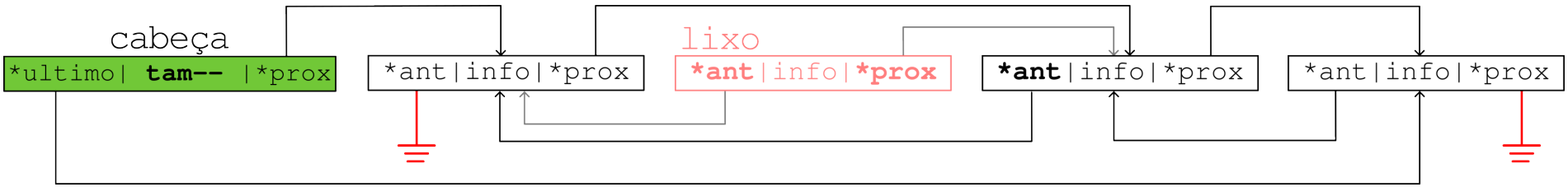



```
1  void remove_no(cabeca *lista, no *lixo)
2  {
3
4      if(lista->prox == lixo) lista->prox = lixo->prox;
5      else lixo->ant->prox = lixo->prox;
6
7      if(lista->ultimo == lixo) lista->ultimo = lixo->ant;
8      else lixo->prox->ant = ??
9
10
11
12 }
13
```

Remoção


```
1  void remove_no(cabeca *lista, no *lixo)
2  {
3
4      if(lista->prox == lixo) lista->prox = lixo->prox;
5      else lixo->ant->prox = lixo->prox;
6
7      if(lista->ultimo == lixo) lista->ultimo = lixo->ant;
8      else lixo->prox->ant = lixo->ant;
9
10
11
12 }
13
```

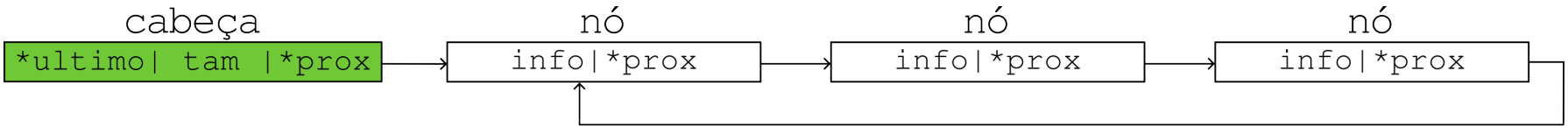
Remoção

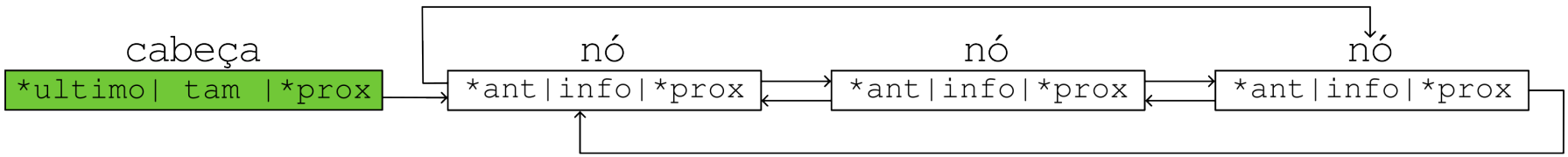


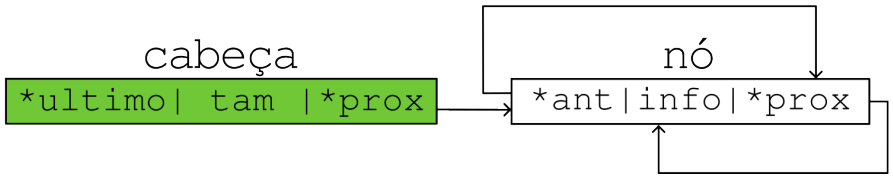

```
1  void remove_no(cabeca *lista, no *lixo)
2  {
3
4      if(lista->prox == lixo) lista->prox = lixo->prox;
5      else lixo->ant->prox = lixo->prox;
6
7      if(lista->ultimo == lixo) lista->ultimo = lixo->ant;
8      else lixo->prox->ant = lixo->ant;
9
10     lista->tam--;
11
12 }
13 //lixo continua na memória?
```

Outras listas encadeadas

- Circular:
  - Simplesmente: último aponta para o primeiro
  - Duplamente: primeiro elemento aponta para o último e vice-versa
  - Último elemento da lista??
  - Único elemento da lista??
  - Implementem as operações básicas

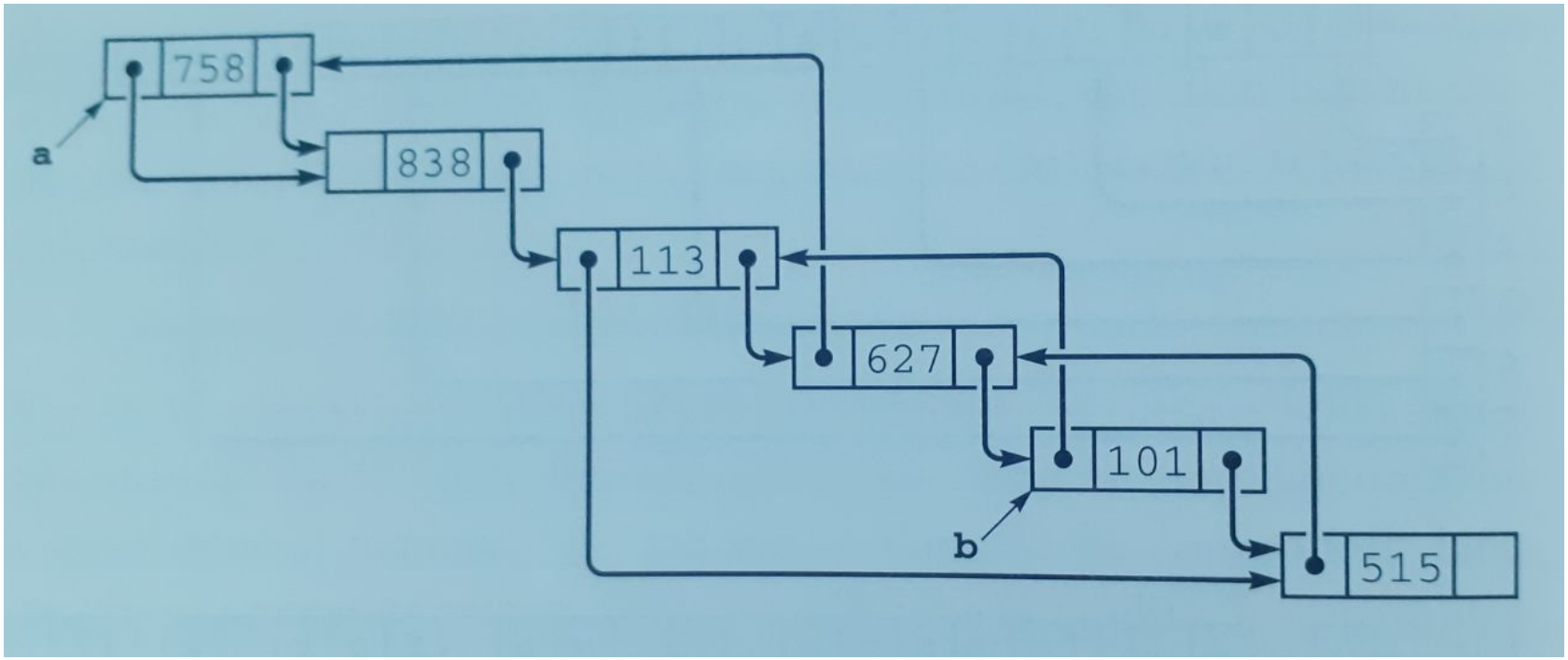






Outras listas encadeadas

- Multilista:
  - Apontamentos para o próximo e anterior são independentes, não necessariamente um nó aponta de volta para o nó que aponta para ele
  - Exemplo: começando por A, temos a ordem na qual os itens foram inseridos; por B, temos a lista ordenada



- Duplamente encadeada:
  - Multilista
  - x->prox->ant = x = x->ant->prox

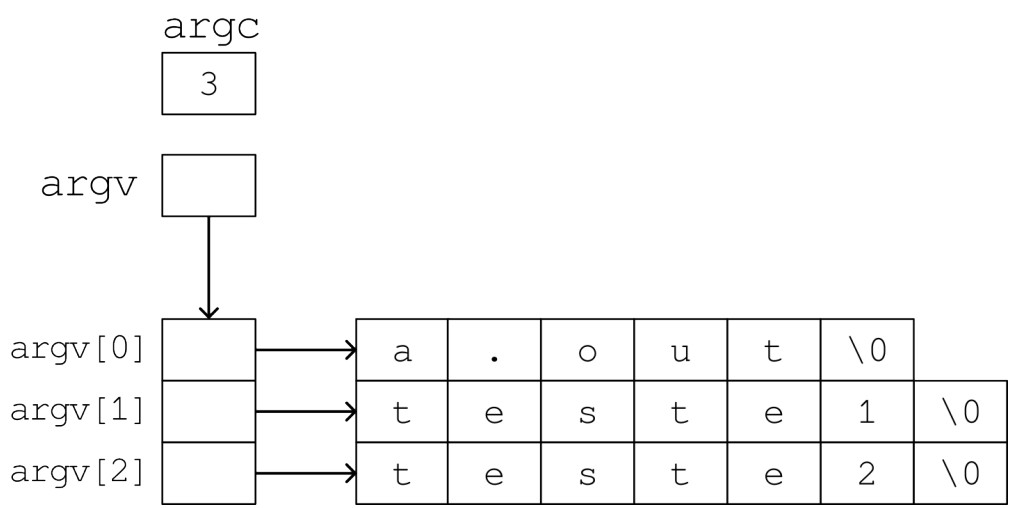
Estruturas de dados elementares - relembrando

- Representam um conjunto de dados
  - Listas estáticas: arrays
  - Listas encadeadas: simplesmente, duplamente, circular
- Composição de estruturas de dados (arrays, listas encadeadas, strings)
  - Vetor de strings

```c
int main(int argc, char *argv[]){
    for(int cont=0; cont < argc; cont++)
        printf("%d Parametro: %s\n", cont, argv[cont]);

    return 0;
}
```

```
./a.out teste1 teste2
```



  - Vetor de vetores (matriz)

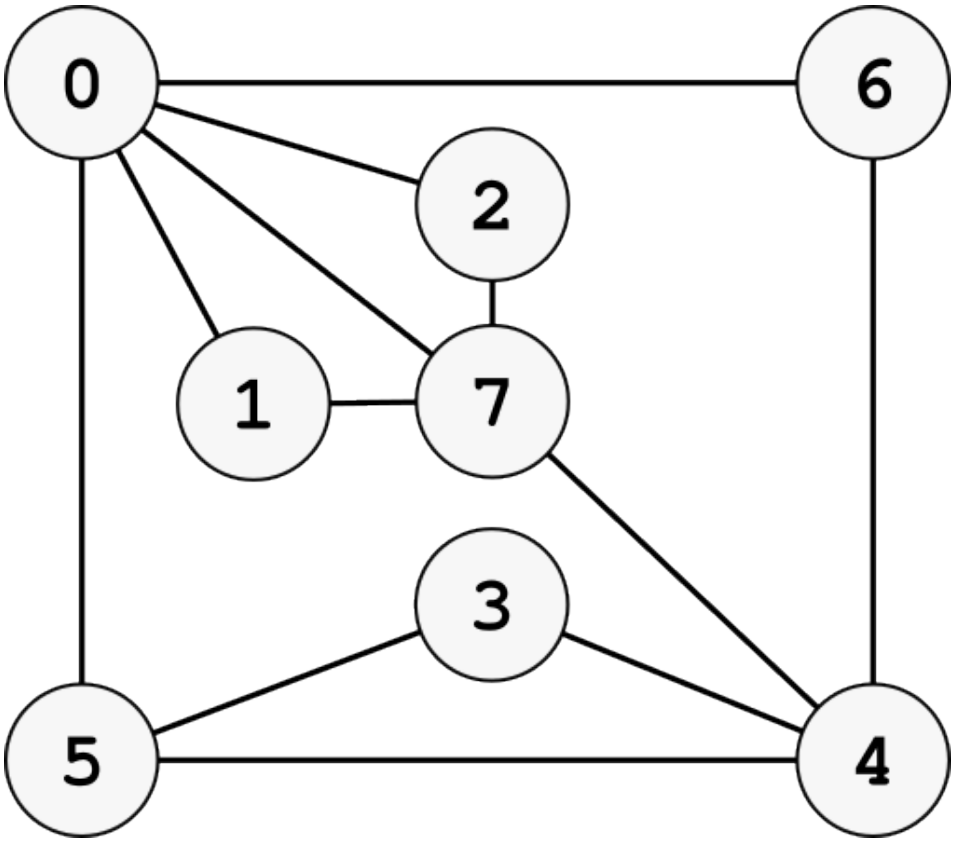
Estruturas de dados elementares - relembrando

- Representam um conjunto de dados
  - Listas estáticas: arrays
  - Listas encadeadas: simplesmente, duplamente, circular
- Composição de estruturas de dados (arrays, listas encadeadas, strings)
  - Vetor de strings
  - Vetor de vetores (matriz)
    - ⋆ **Matrizes com listas encadeadas**: bom **desempenho espacial** em casos de **matrizes esparsas**
    - ⋆ Matriz esparsa: grande quantidade dos elementos são não-válidos (zeros)
    - ⋆ Conceito de esparsidade/dispersão: frequentes em mineração de dados, análises numéricas, combinatórias, aplicações científicas e de engenharia (fenômenos eletrostática, eletrodinâmica, eletromagnetismo, dinâmica dos fluidos, difusão do calor, propagação de ondas), tabela hash (estrutura de dados que associa chave de pesquisa a valores de indíces - ótimo desempenho em inserções, remoções e buscas), grafos esparsos
  - Multilistas: representação de árvores

Exemplo de aplicação: Grafos

- Estuda a relação entre objetos para a obtenção de informações
  - Objetos são chamados de **vértices** e as relações de **arestas**
- A organização em grafos contribui para a resolução de problemas com rotas, combinatórias, logística, fluxo, etc.
- Auxiliando em operações para definir o melhor (menor, mais rápido) caminho, geração de preferências, perfis, classificações, hierarquias, combinações, etc.
- Exemplos:
  - Escalonamento de vôos, transporte de mercadorias, mapas, rede de computadores, links nas páginas web, relacionamento nas redes sociais, banco de dados, máquinas de aprendizagem, mineração de dados, busca na Internet, escalonamento de tempo, engine de jogos.
- Possível representação por uma matriz de adjacências
  - Dado matriz[i][j] = 1, diz-se que o vértice i está conectado ao vértice j
  - Quando há poucas conexões, com muitos dos pares (i,j) iguais a 0, diz-se que é uma matriz esparsa
  - Grafos esparsos, são eficientemente representados com **listas de adjacências (array de listas)**

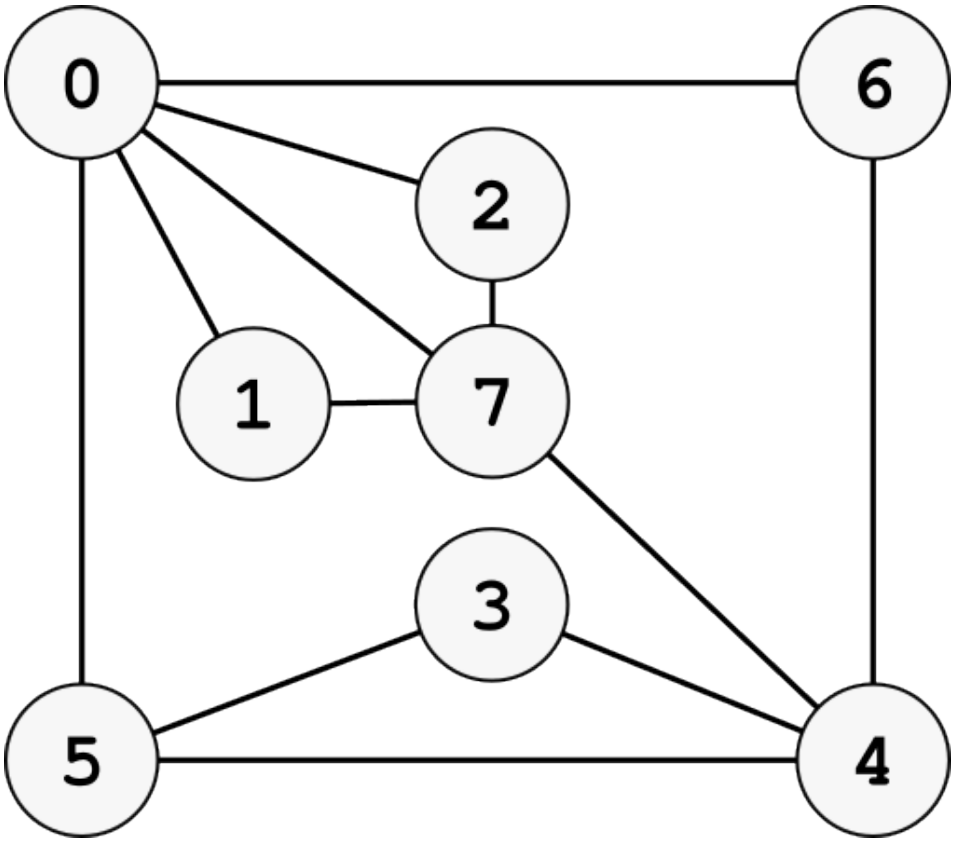
Exemplo: Ligações entre pessoas (rede sociais)



Matriz de adjacências

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	1	1	1
1	1	1	0	0	0	0	0	1
2	1	0	1	0	0	0	0	1
3	0	0	0	1	1	1	0	0
4	0	0	0	1	1	1	1	1
5	1	0	0	1	1	1	0	0
6	1	0	0	0	1	0	1	0
7	1	1	1	0	1	0	0	1

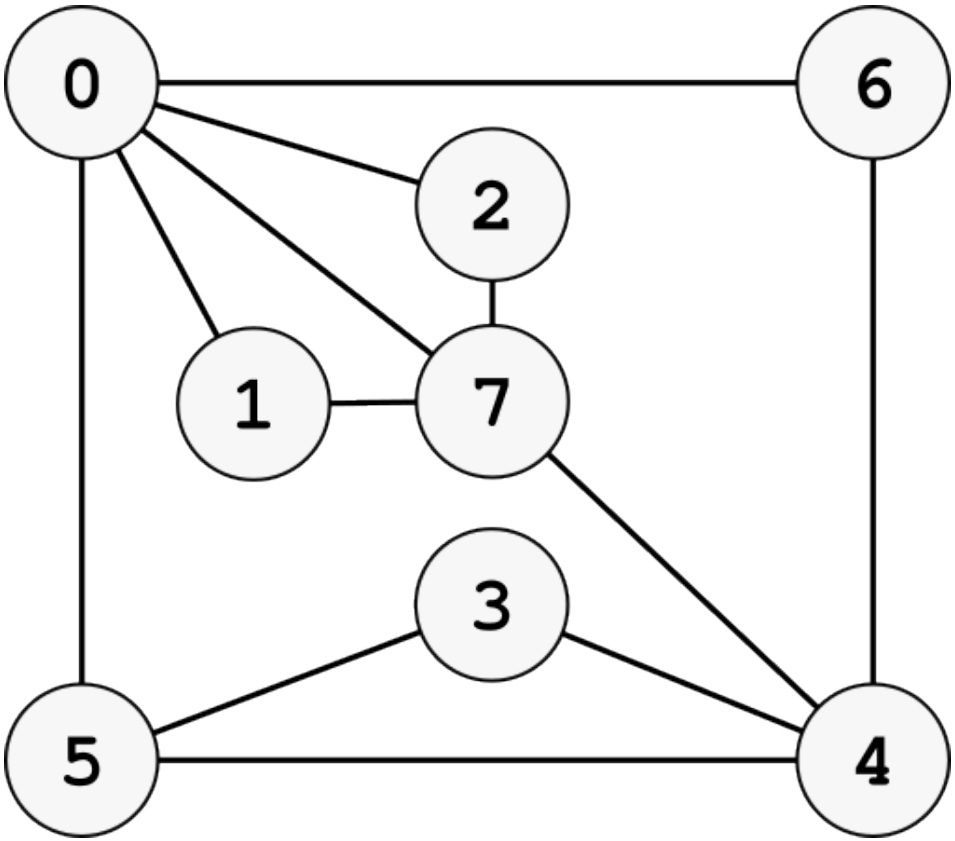
Exemplo: Ligações entre pessoas (rede sociais)



Matriz de adjacências

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	1	1	1
1	1	1	0	0	0	0	0	1
2	1	0	1	0	0	0	0	1
3	0	0	0	1	1	1	0	0
4	0	0	0	1	1	1	1	1
5	1	0	0	1	1	1	0	0
6	1	0	0	0	1	0	1	0
7	1	1	1	0	1	0	0	1

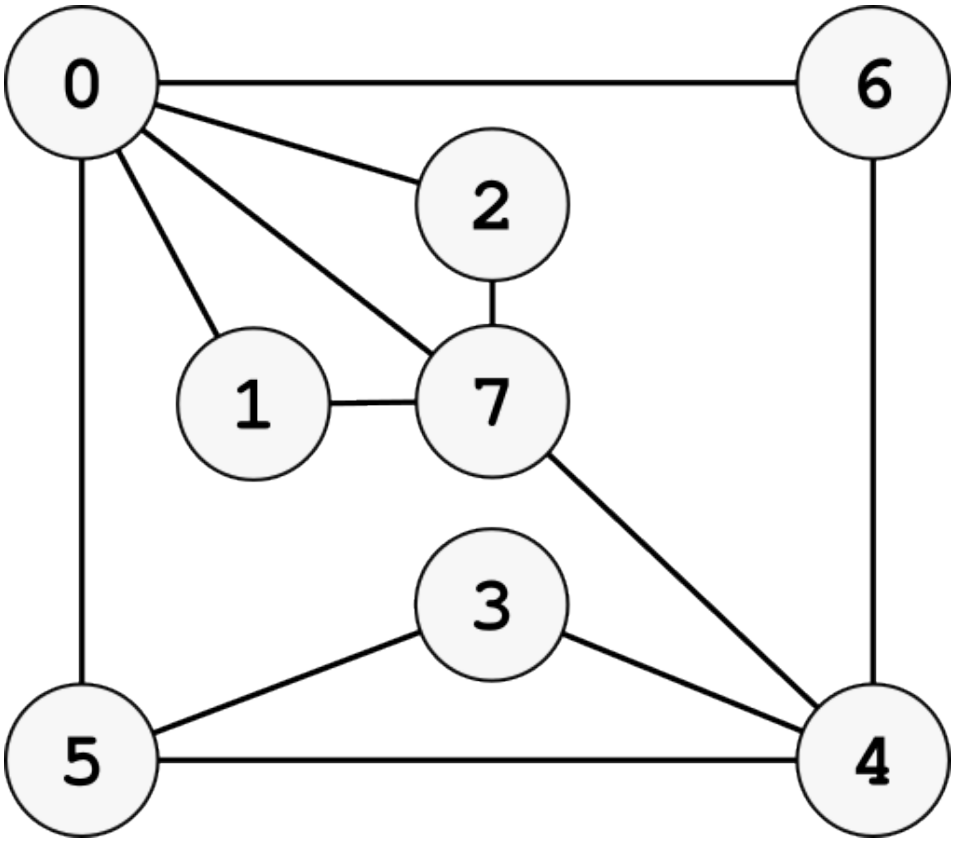
Exemplo: Ligações entre pessoas (rede sociais)



Matriz de adjacências

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	1	1	1
1	1	1	0	0	0	0	0	1
2	1	0	1	0	0	0	0	1
3	0	0	0	1	1	1	0	0
4	0	0	0	1	1	1	1	1
5	1	0	0	1	1	1	0	0
6	1	0	0	0	1	0	1	0
7	1	1	1	0	1	0	0	1

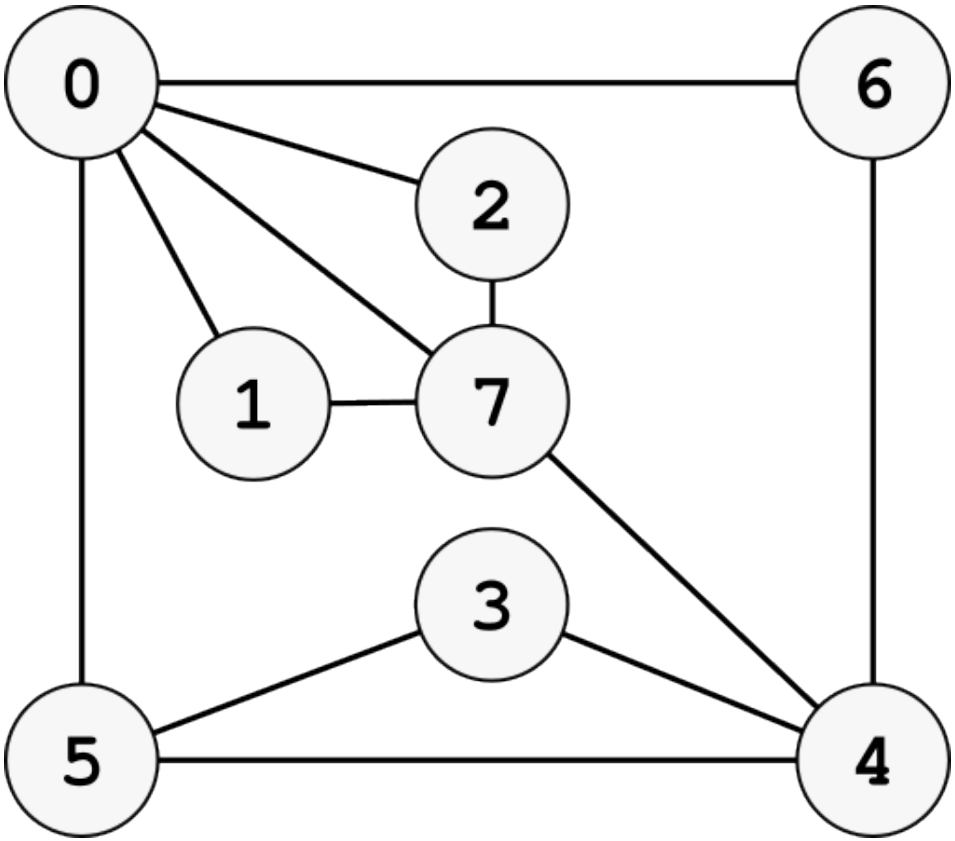
Exemplo: Ligações entre pessoas (rede sociais)



Matriz de adjacências

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	1	1	1
1	1	1	0	0	0	0	0	1
2	1	0	1	0	0	0	0	1
3	0	0	0	1	1	1	0	0
4	0	0	0	1	1	1	1	1
5	1	0	0	1	1	1	0	0
6	1	0	0	0	1	0	1	0
7	1	1	1	0	1	0	0	1

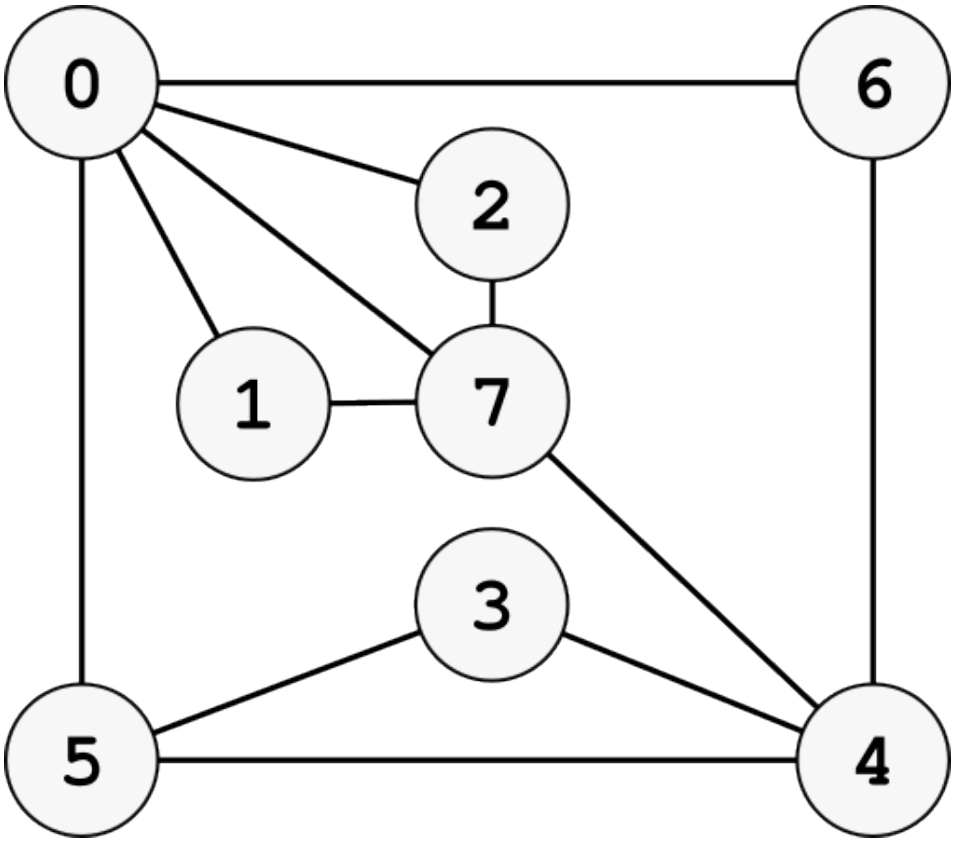
Exemplo: Ligações entre pessoas (rede sociais)



Matriz de adjacências

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	1	1	1
1	1	1	0	0	0	0	0	1
2	1	0	1	0	0	0	0	1
3	0	0	0	1	1	1	0	0
4	0	0	0	1	1	1	1	1
5	1	0	0	1	1	1	0	0
6	1	0	0	0	1	0	1	0
7	1	1	1	0	1	0	0	1

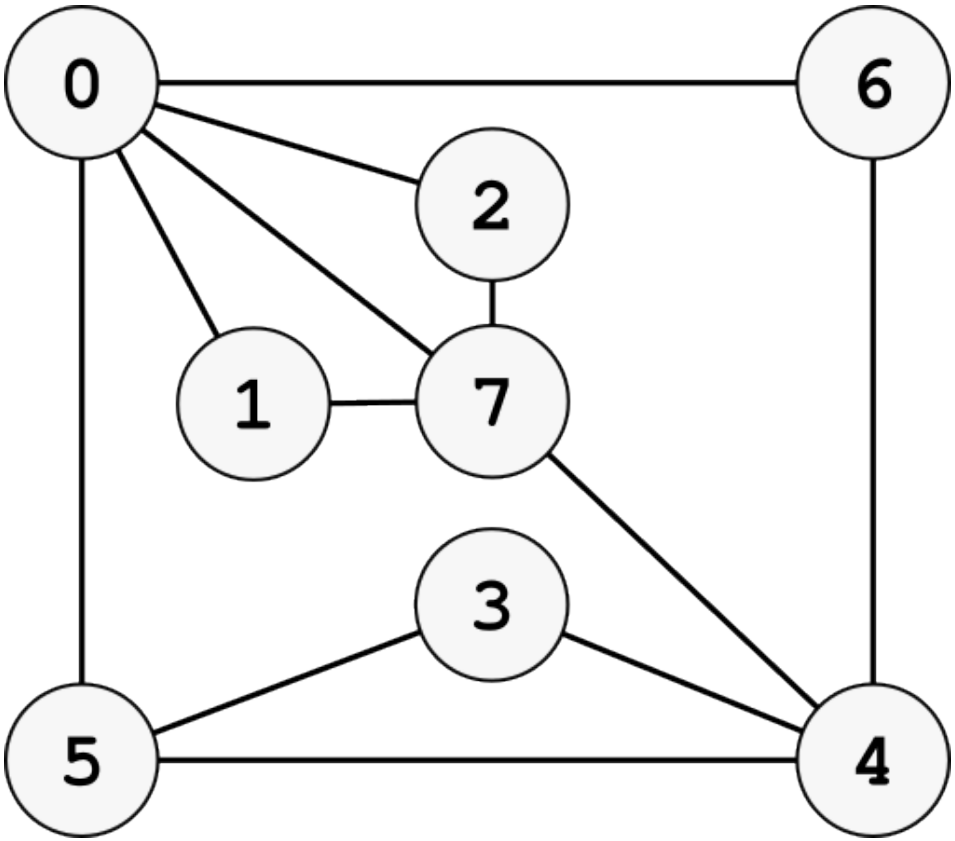
Exemplo: Ligações entre pessoas (rede sociais)



Matriz de adjacências

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	1	1	1
1	1	1	0	0	0	0	0	1
2	1	0	1	0	0	0	0	1
3	0	0	0	1	1	1	0	0
4	0	0	0	1	1	1	1	1
5	1	0	0	1	1	1	0	0
6	1	0	0	0	1	0	1	0
7	1	1	1	0	1	0	0	1

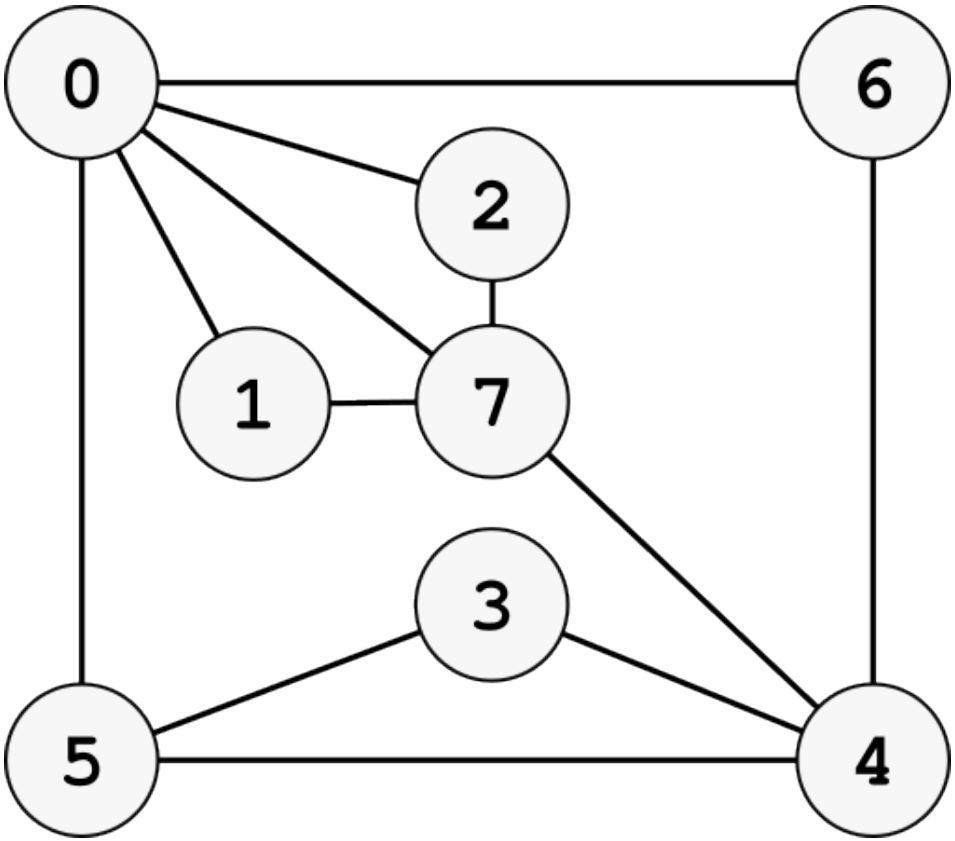
Exemplo: Ligações entre pessoas (rede sociais)



Matriz de adjacências

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	1	1	1
1	1	1	0	0	0	0	0	1
2	1	0	1	0	0	0	0	1
3	0	0	0	1	1	1	0	0
4	0	0	0	1	1	1	1	1
5	1	0	0	1	1	1	0	0
6	1	0	0	0	1	0	1	0
7	1	1	1	0	1	0	0	1

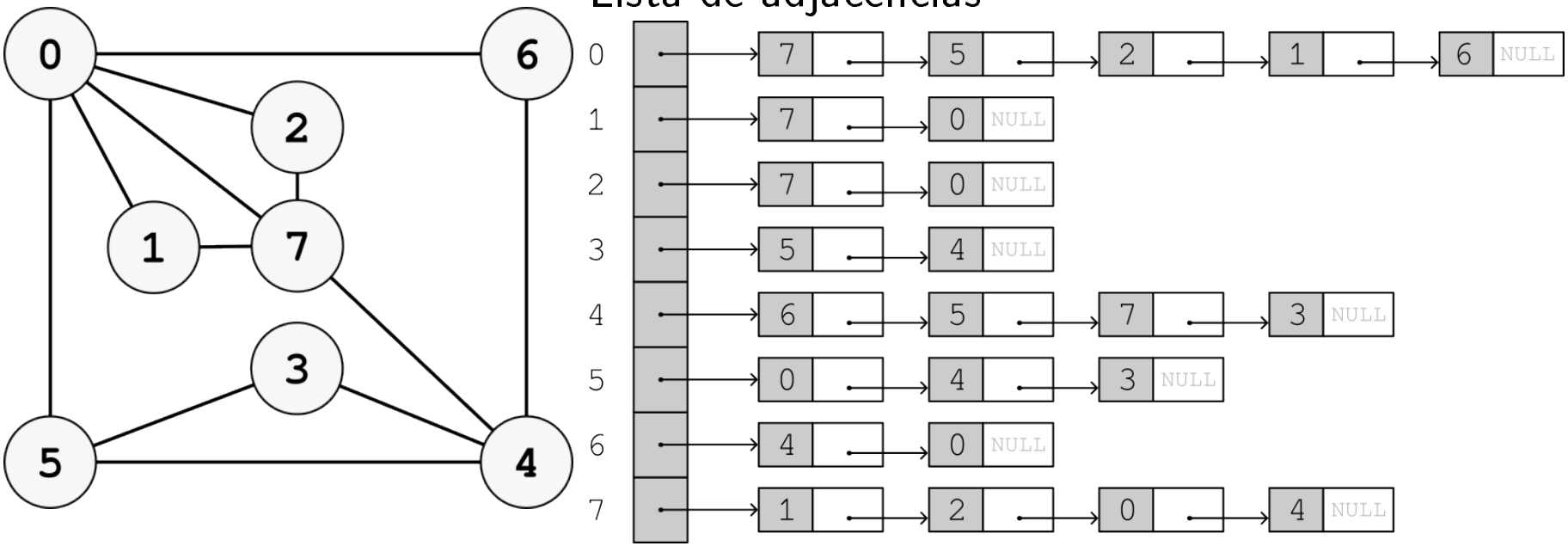
Exemplo: Ligações entre pessoas (rede sociais)



Matriz de adjacências

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	1	1	1
1	1	1	0	0	0	0	0	1
2	1	0	1	0	0	0	0	1
3	0	0	0	1	1	1	0	0
4	0	0	0	1	1	1	1	1
5	1	0	0	1	1	1	0	0
6	1	0	0	0	1	0	1	0
7	1	1	1	0	1	0	0	1

Exemplo: Ligações entre pessoas (rede sociais)



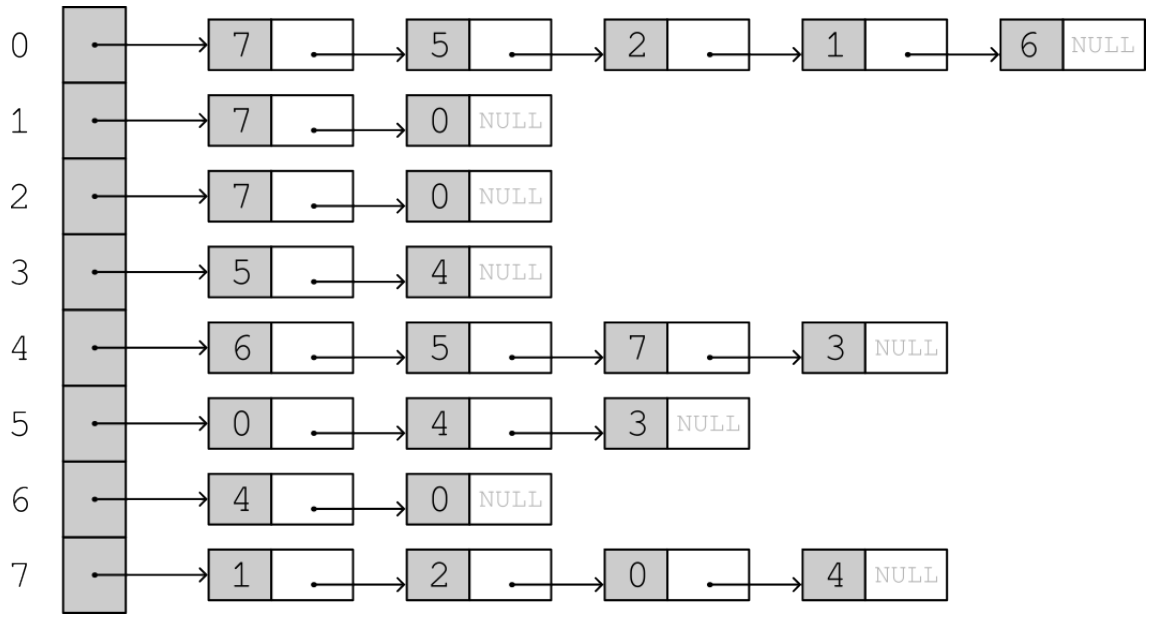
Exemplo: lista de adjacências

```
1     //V número de vértices
2     //A número de arestas
3
4     //Entradas: pares de vértices conectados
5
6     no *adj[V];
7     for(int i=0; i<V; i++){
8         adj[i] = criar();
9     }
10
11    while(scanf("%d%d", &i, &j) == 2){
12        inserir_inicio(adj[j], criar_no(i));
13        inserir_inicio(adj[i], criar_no(j));
14    }
15
```

Exemplo: lista de adjacências

```
1     no *adj[V];
2     for(int i=0; i<V; i++){
3         adj[i] = criar();
4     }
5
6     while(scanf("%d%d", &i, &j) == 2){
7         inserir_inicio(adj[j], criar_no(i));
8         inserir_inicio(adj[i], criar_no(j));
9     }
10
```

(7,1) (2,7)
(4,6) (7,0)
(0,5) (5,4)
(2,0) (4,7)
(3,5) (1,0)
(1,6) (3,4)



Exemplo: Representações das adjacências

- Matriz:
  - Acesso aleatório e direto $O(1)$
  - Em grafos esparsos, desperdício de espaço $O(V^2)$;
  - Processar todos os elementos $O(V^2)$
- Lista:
  - Em grafos esparsos, custo espacial é menor $O(V + A)$
  - Processar todos elementos $O(V + A)$
  - Acesso sequencial $O(V)$
  - Em grafos densos, com a maioria dos vértices conectados entre si, a vantagem espacial é menor do que a desvantagem do acesso
    $O(V + A) = O(V + V * (V - 1)) = O(V^2)$