Lista Circular

Algoritmos e Programação II

Introdução

- Variação das listas encadeadas, chamada de lista encadeada circular, é usada em várias aplicações que envolvem jogos de roda como "Corre Cotia".

- Minnesota praticam uma versão chamada de "Duck, Duck, Grey Duck", mas não perguntem o porquê.

- Em Indiana, esse jogo é chamado de "The Mosh Pot".
- As crianças da República Checa e do Brasil joga versões cantadas do jogo, conhecidas, respectivamente, como "Pesek" e "Corre Cotia".

Lista encadeada circular

Lista encadeada circular

- Lista tem o mesmo tipo de nós que uma lista encadeada simples.
  - cada nó tem um ponteiro para o próximo nó e um valor.
  - não existe cabeça (início) ou cauda (fim) em uma lista circular.
  - último nó de uma lista circular aponta para o primeiro nó.

- Mesmo que uma lista circular não tenha início ou fim, sempre será necessário que algum nó seja marcado de forma especial, sendo chamado de *cursor*. O nó *cursor* serve como ponto de partida sempre que for necessário percorrer a lista circular.

Operações sobre a lista

- adicionar(v):
  - Insere um nó novo, **v**, imediatamente após o ***cursor***; se a lista está vazia, então **v** torna-se o ***cursor*** e seu ponteiro ***prox*** aponta para si mesmo.
- remover():
  - remove e retorna o nó **v** que se encontra imediatamente após o ***cursor*** (não o ***cursor*** propriamente dito, a menos que ele seja o único nó); se a lista ficar vazia, o ***cursor*** é definido como ***null***.
- avançar(): avança o cursor para o próximo nó da lista.

Lista encadeada circular

DEFINIÇÃO

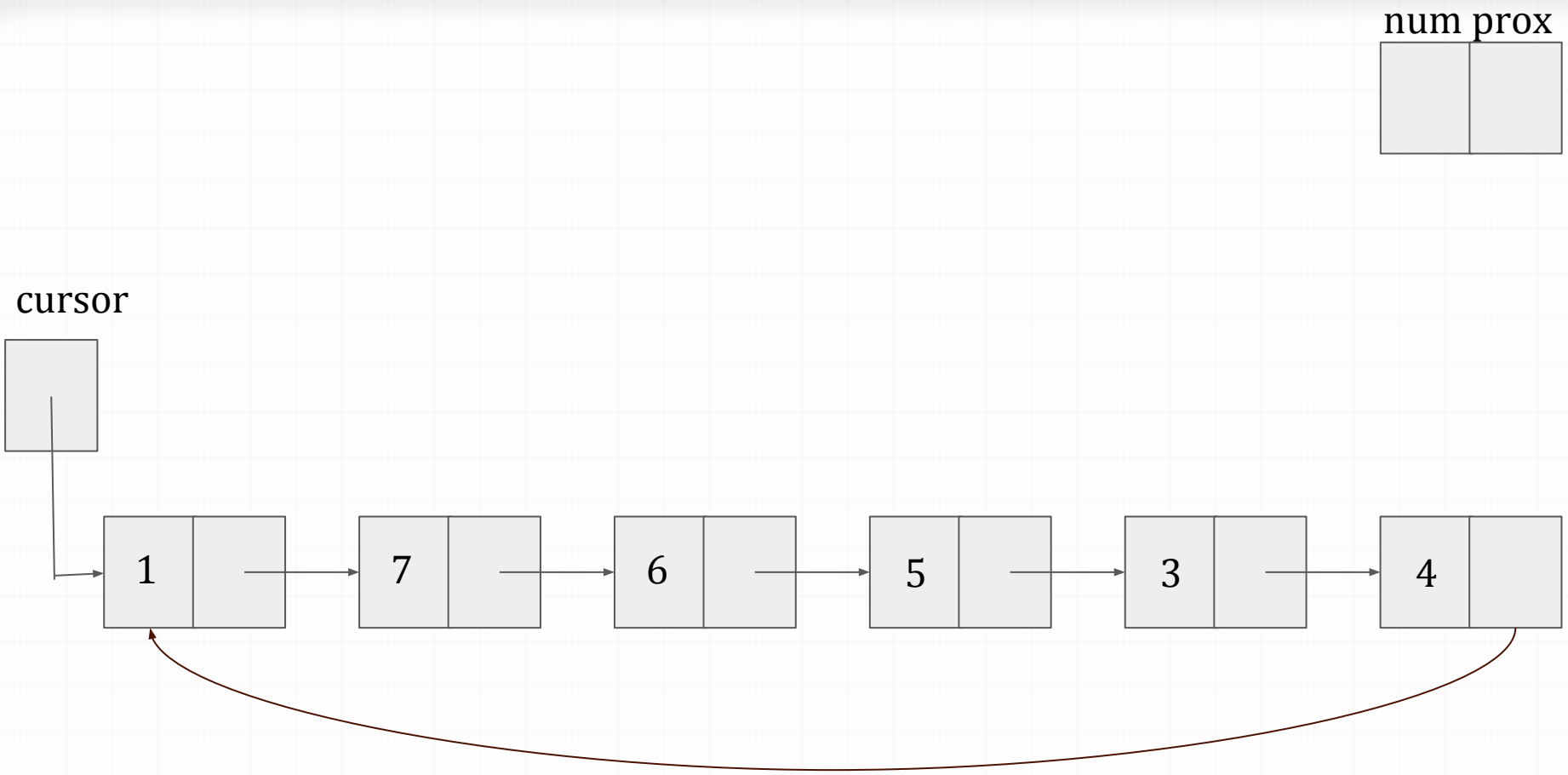
- Considere que as células da lista são do tipo abaixo:

C

```
typedef struct cel {
   int num;
   struct cel *prox;
} celulaCirc;
```

C++

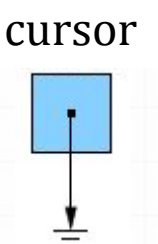
```
struct celulaCirc{
    int num;
   struct celulaCirc *prox;
};
```


num prox
cursor
1
7
6
5
3
4

DEFINIÇÃO

- A inicialização de uma lista circular em alocação encadeada é dada a seguir:

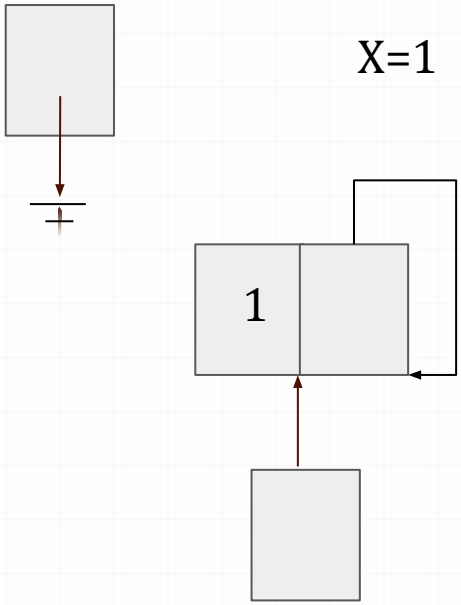
```
celulaCirc *cursor;

cursor = NULL;
```



REPRESENTAÇÃO DE UMA **LISTA CIRCULAR VAZIA** EM ALOCAÇÃO ENCADEADA

Inserção: todo novo nó é inserido após o cursor

cursor

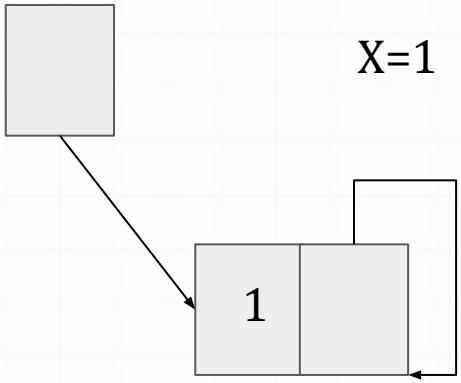


nova

X=1

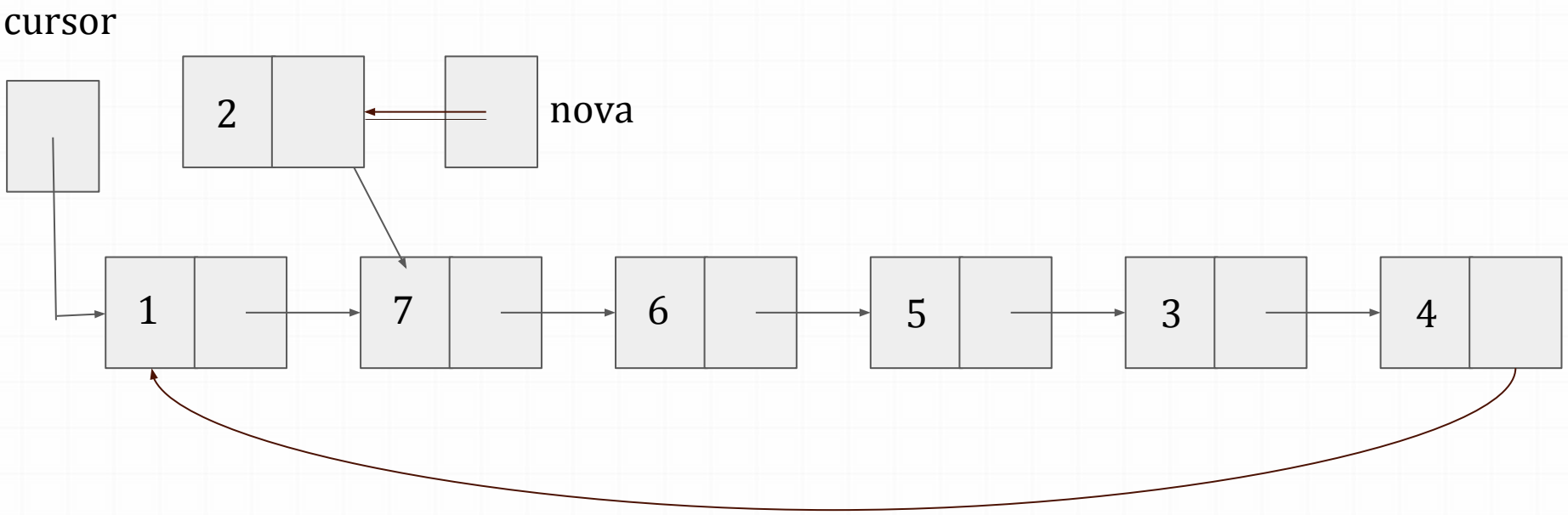
LISTA ESTÁ VAZIA?

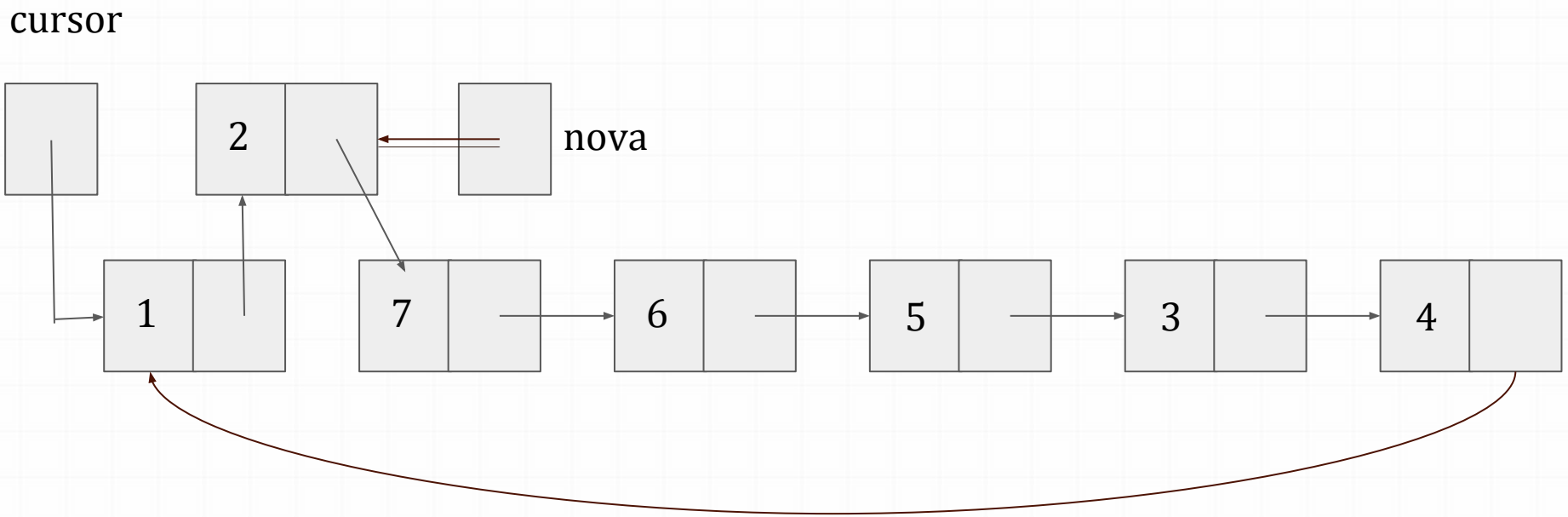
cursor



LISTA **não** ESTÁ VAZIA

X=2




cursor
2
nova
1
7
6
5
3
4

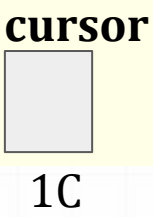
Operação de adicionar

Protótipo da função (parâmetros: o valor a ser inserido e o ponteiro para a lista passado por referência).

void adicionar (int , celulaCirc*&);

Chamada da função adicionar(), por exemplo:

```
/*dada uma lista circular 'cursor' e um dado na variável 'numero'*/
celulaCirc *cursor=NULL;
int numero;
……
                    adicionar(numero,  cursor);
```



Operação de inserção

```
/* Recebe uma chave x e uma lista circular e insere x imediatamente
após o cursor*/
void adicionar(int x, celulaCirc *&cur)
{
  celulaCirc *nova;

  nova = (celulaCirc *) calloc(sizeof (celulaCirc));
  nova->chave = x;

  if(cur == NULL)
  {
    nova->prox = nova;
    cur = nova;
  }
  else{

    nova->prox = cur->prox;

    cur->prox = nova;
  }
}
```

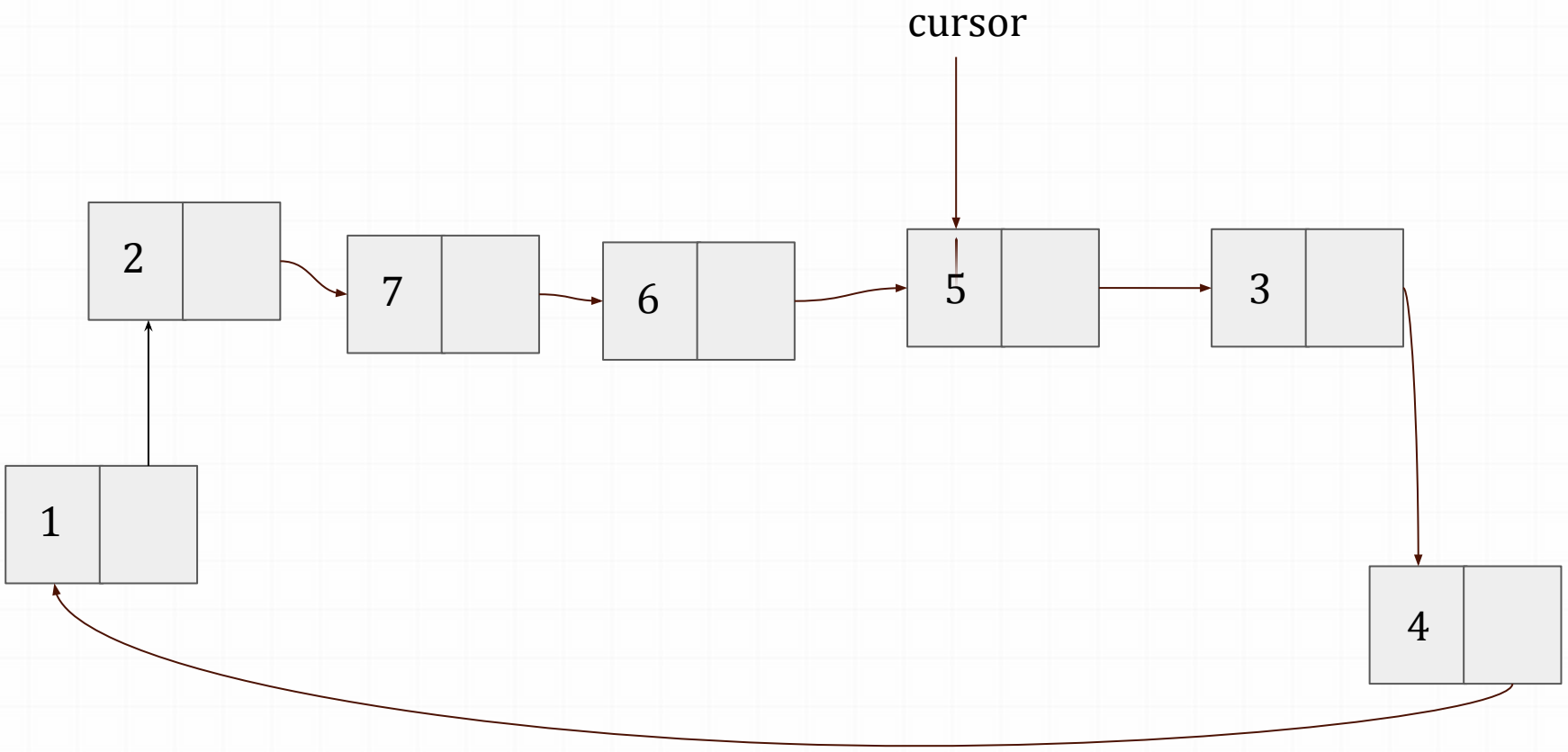
cursor

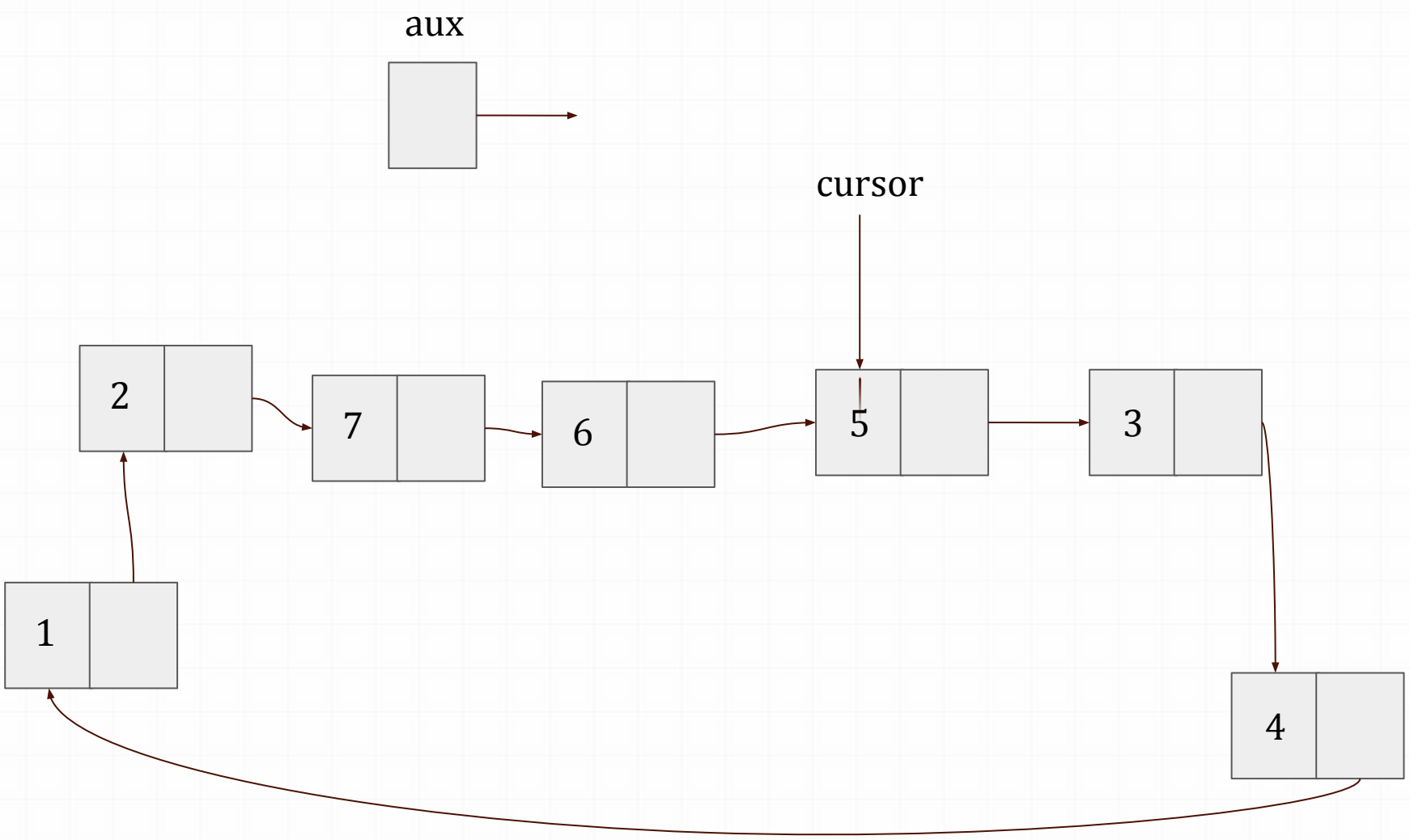
1C

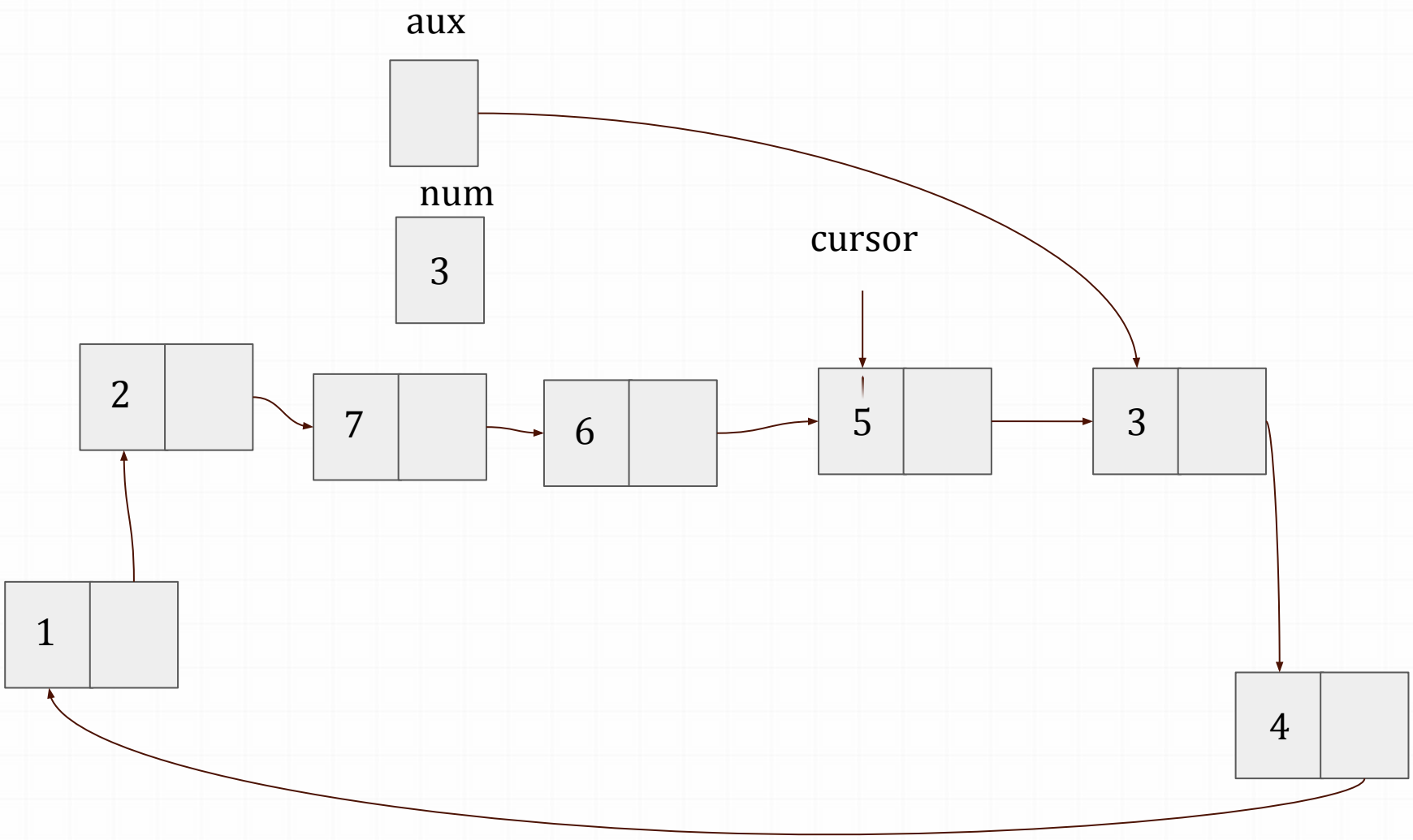
cur

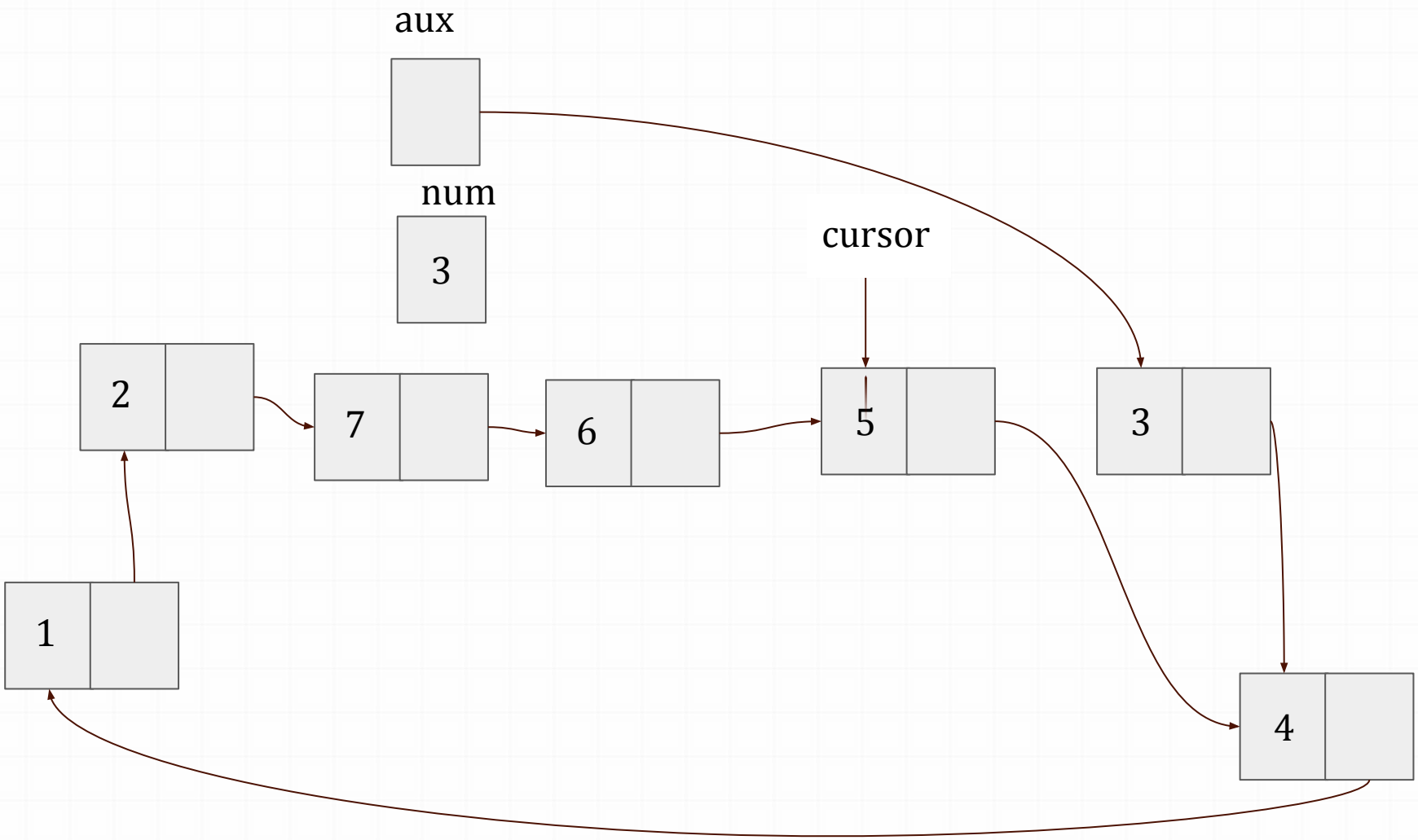
cur é o apelido de cursor

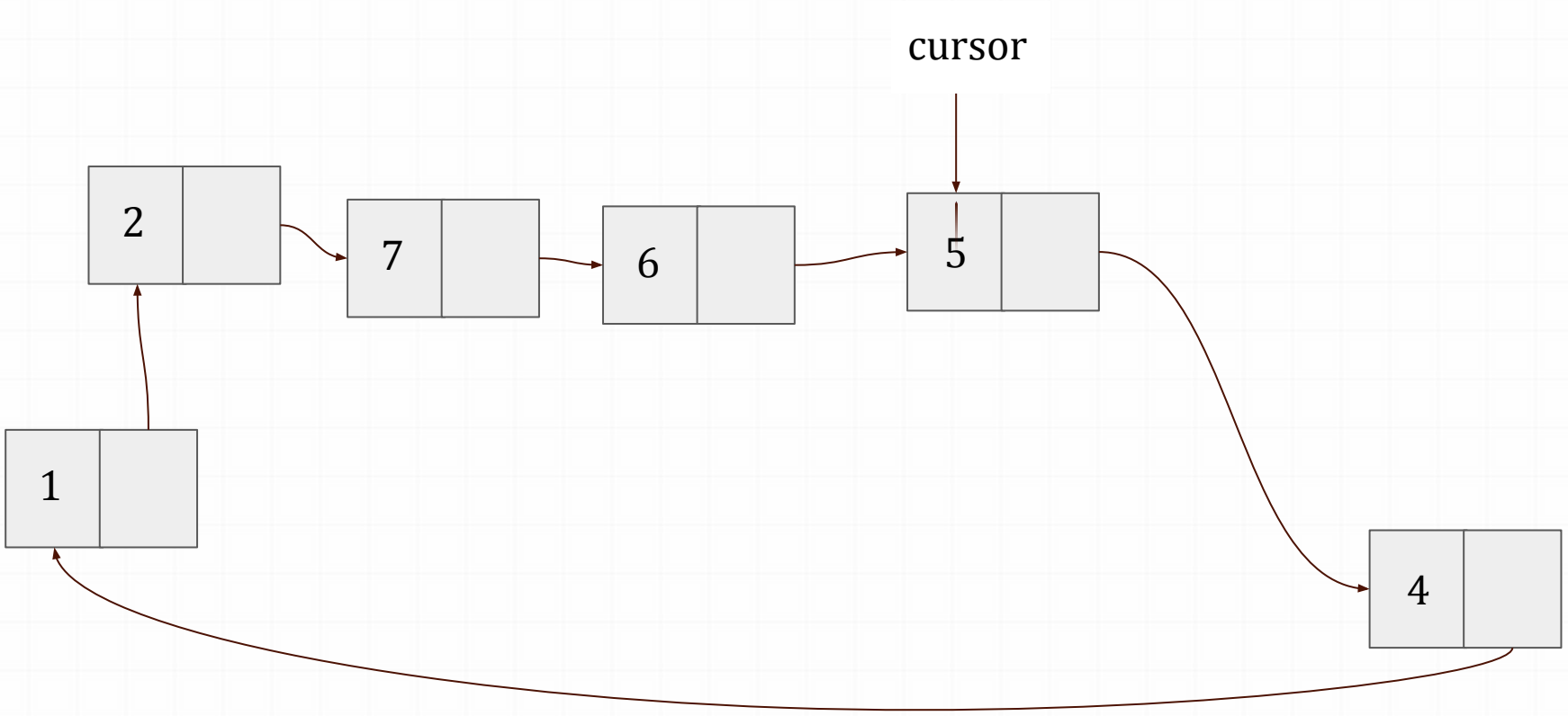
Remoção de um nó: o nó após o cursor


cursor
2
7
6
5
3
1
4


aux
cursor
2
7
6
5
3
1
4


aux
num
3
cursor
2
7
6
5
3
1
4


aux
num
3
cursor
2
7
6
5
3
1
4


cursor
2
7
6
5
1
4

Operação de remover

Protótipo da função (parâmetros: ponteiro para a lista passado por referência).

int remover(celulaCirc*&);

Chamada da função remover(), por exemplo:

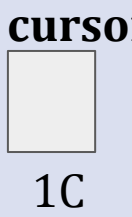
```
/*dada uma lista circular 'cursor' */
celulaCirc *cursor;
int num;

num = remover( cursor );
```

cursor

1C

Operação de remoção

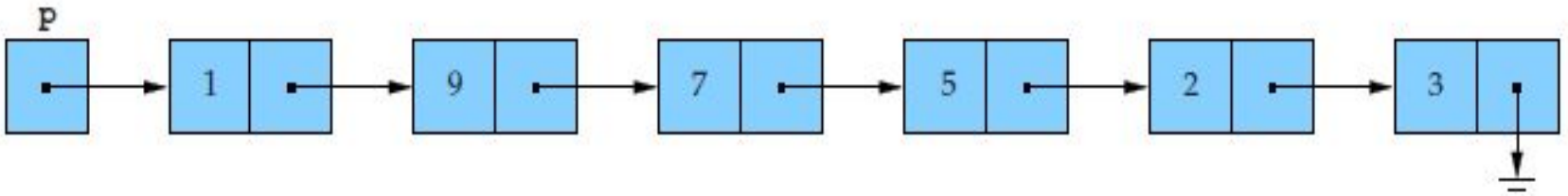
```
/*Recebe o ponteiro para um elemento da lista (cursor) e remove o
elemento que vem imediatamente após.*/
int remover(celula *&cur)
{
    celula *aux;
    int num;

    if(cur == NULL)
        return 0;
    else{
        aux = cur->prox;
        num = aux->chave;

        if(aux == cur)
            cur = NULL;
        else
            cur->prox = aux->prox;

        free(aux);
        return num;
    }
}
```



cur é o apelido de cursor

Listas e suas complexidades

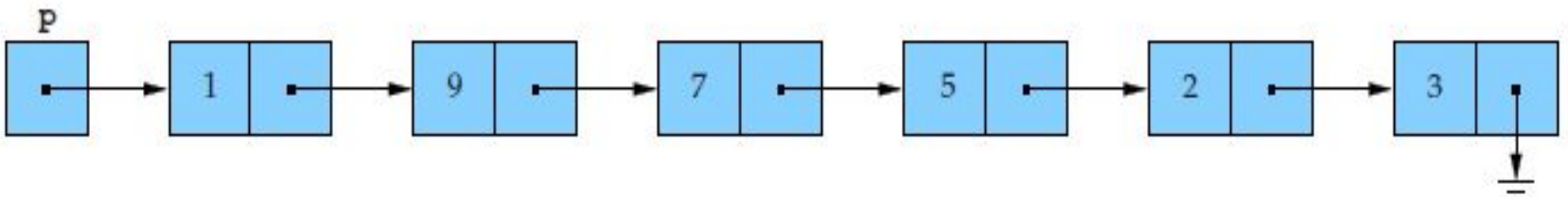
- Lista Simplesmente Encadeada ***p***



- Operação de inserção no início – $O(1)$ – tempo constante
- Operação de inserção no fim - $O(n)$ – n é o número de nós
- Busca - $O(n)$ – n é o número de nós
- Remoção de um elemento qualquer - $O(n)$ – n é o número de nós

Listas e suas complexidades

- Lista Simplesmente Encadeada ***p***

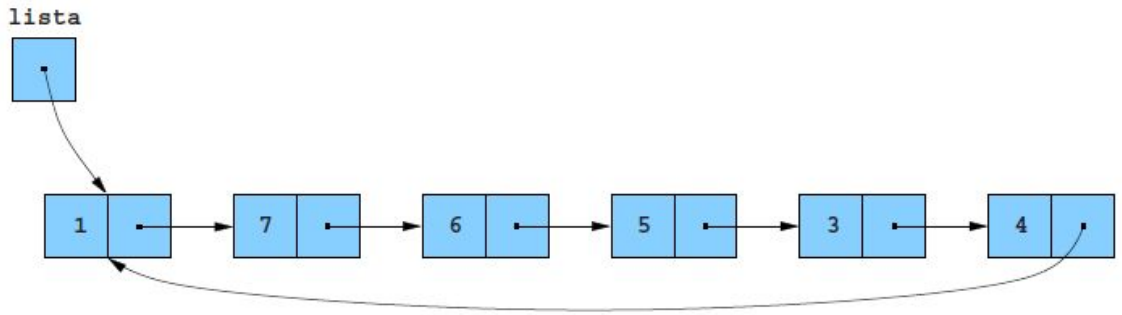


- Remoção no início – $O(1)$ – tempo constante

- Remoção de um elemento qualquer - $O(n)$ – n é o número de nós

Listas e suas complexidades

- Lista Encadeada Circular ***p***

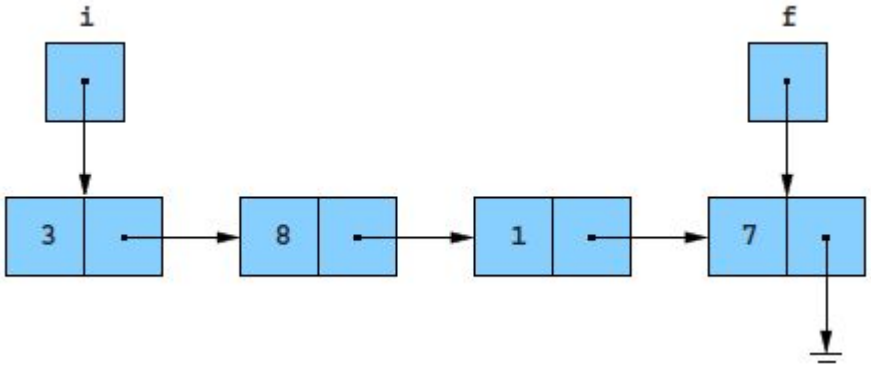


- Inserção – $O(1)$ – tempo constante

- Remoção – $O(1)$ – tempo constante

Listas e suas complexidades

- Fila ***i***

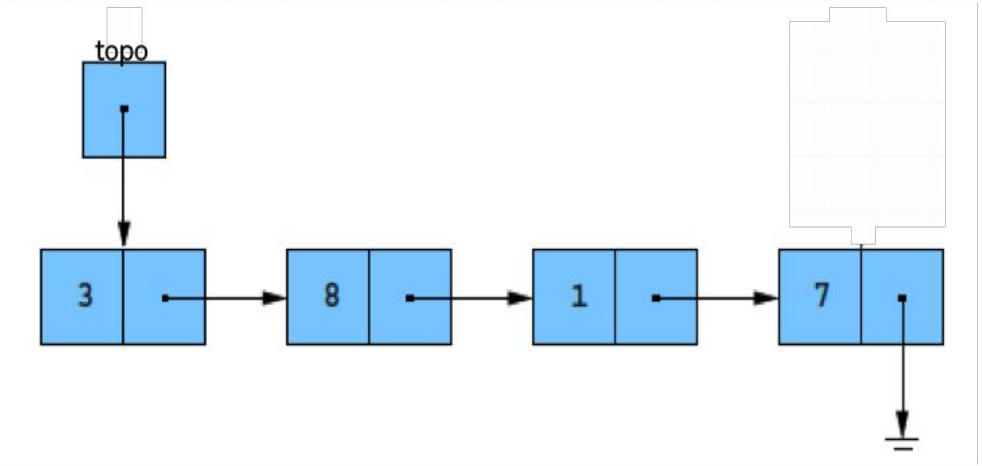


- inserção no fim ($O(1)$ – tempo constante)

- remoção do primeiro ($O(1)$ – tempo constante)

Listas e suas complexidades

- Pilha ***topo***

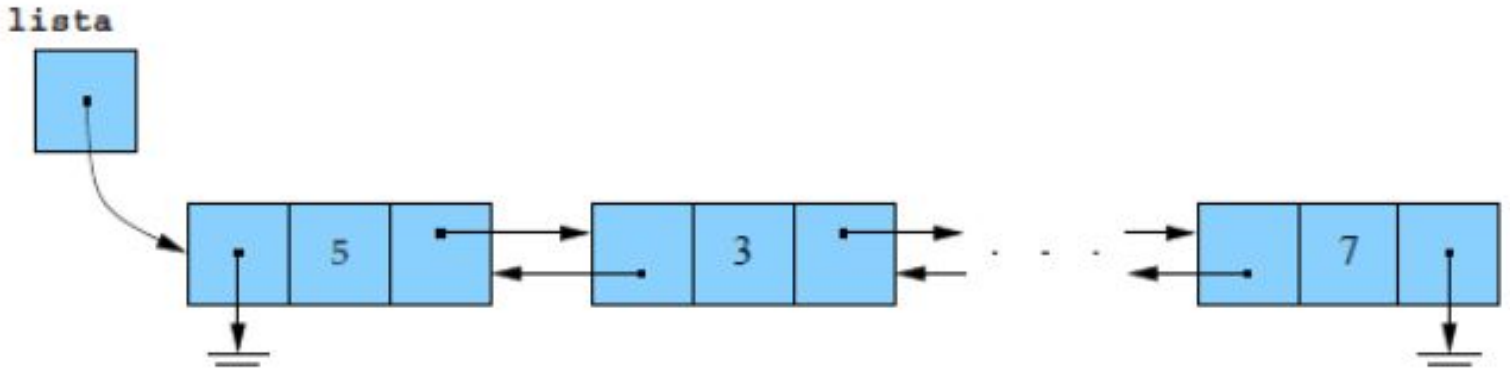


- inserção no topo: $O(1)$ – tempo constante

- remoção do topo: $O(1)$ – tempo constante

Listas e suas complexidades

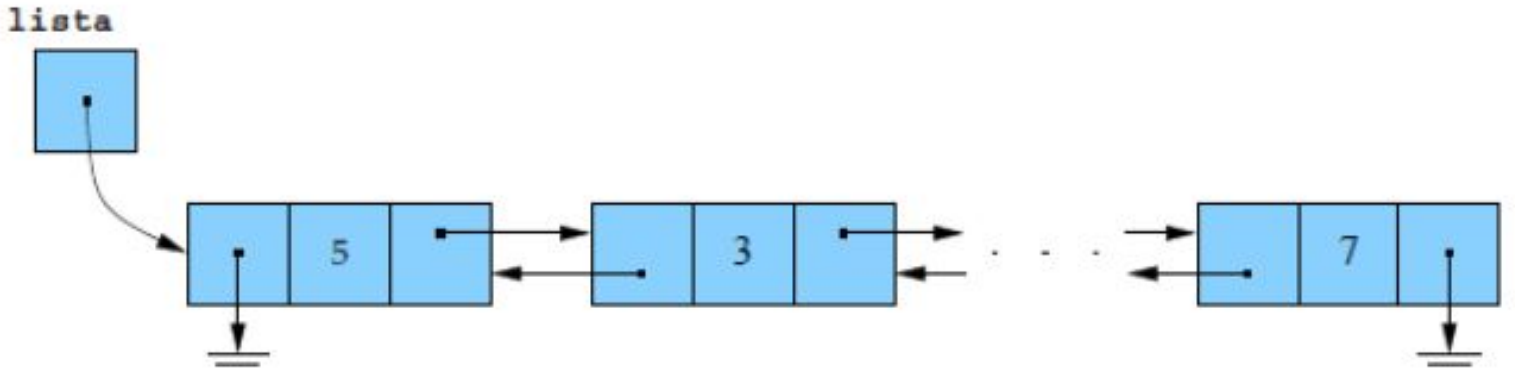
- Lista duplamente encadeada ***lista***



- Operação de inserção no início: $O(1)$ – tempo constante
- Operação de inserção no fim: $O(n)$ – n é o número de nós
- Busca: $O(n)$ – n é o número de nós

Listas e suas complexidades

- Lista duplamente encadeada ***lista***



- Remoção no início – $O(1)$ – tempo constante

- Remoção de um elemento qualquer - $O(n)$ – n é o número de nós