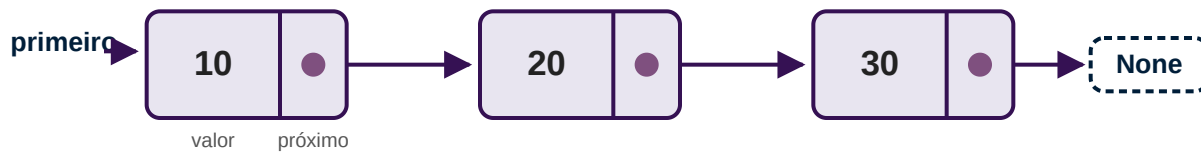


TÓPICO 08: LISTAS ENCADEADAS SIMPLES

Estrutura de Dados - Professor Ramon Venson - SATC 2026.2

Lista Encadeada Simples

Lista Encadeada Simples



Cada nó guarda um valor e uma referência para o próximo nó.

Estrutura de dados **linear e dinâmica**, formada por **nós** que guardam um valor e uma **referência para o próximo nó**.

Ao contrário dos vetores, **não possui tamanho fixo**: a lista cresce e diminui conforme a necessidade.

Por que usar Listas Encadeadas?

Nos vetores, o tamanho é definido na criação e inserir/remover no meio exige **deslocar elementos**.

Na lista encadeada:

- Cada elemento fica em um **nó** independente;
- Os nós são ligados por **referências** (ponteiros);
- Inserir e remover exige apenas **ajustar referências**.

O Nó

O **nó** (*node*) é o bloco básico da lista. Ele possui:

- **valor**: o dado armazenado;
- **próximo**: referência para o próximo nó (ou `None`, se for o último).

```
classe No:  
  valor  
  proximo: No  
  
  construtor(valor):  
    este.valor <- valor  
    este.proximo <- None
```

A Lista

A lista guarda apenas a referência para o **primeiro nó** (na literatura, esse ponteiro é chamado de *head*) e a quantidade de elementos.

```
classe ListaEncadeada:  
  primeiro: No  
  tamanho: inteiro  
  
  construtor():  
    este.primeiro <- None  
    este.tamanho <- 0
```

Quando `primeiro` é `None`, a lista está **vazia**.

Verificando se a Lista está Vazia

```
esta_vazia():  
    retorne primeiro == None
```

A lista está vazia quando `primeiro` não aponta para nenhum nó.

Não existe `esta_cheia()`: a lista encadeada cresce enquanto houver memória disponível.

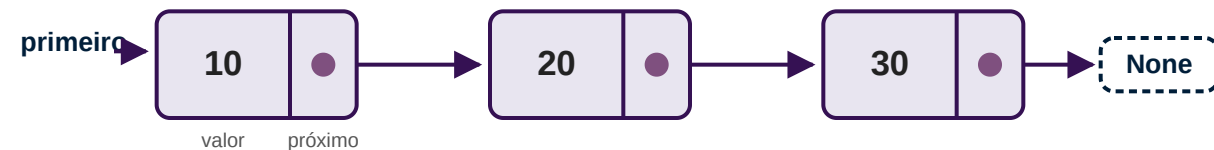
Inserção no Início

Para inserir no início, o novo nó passa a apontar para o antigo `primeiro`, e o `primeiro` passa a apontar para o novo nó.

```
inserir_inicio(valor):  
    novo <- novo No(valor)  
    novo.proximo <- primeiro  
    primeiro <- novo  
    tamanho <- tamanho + 1
```

Inserir no início é **O(1)**: não depende do tamanho da lista.

Lista Encadeada Simples



Cada nó guarda um valor e uma referência para o próximo nó.

Imprimindo os Elementos

Como não há índices, percorremos a lista **nó a nó** a partir do `primeiro`.

```
imprimir():  
    atual <- primeiro  
    enquanto atual != None faça:  
        escreva(atual.valor)  
        atual <- atual.proximo
```

O percurso visita **todos** os nós: complexidade **$O(n)$** .

! Não é possível acessar um elemento diretamente pelo índice, como em um vetor.

Pesquisa de Elementos

Percorremos a lista comparando o valor de cada nó até encontrar o elemento.

```
pesquisar(valor):  
    atual <- primeiro  
    enquanto atual != None faça:  
        se atual.valor == valor então  
            retorne atual  
        atual <- atual.proximo  
    retorne None
```

- **Melhor caso:** o elemento está no primeiro nó (**$O(1)$**).
- **Pior caso:** o elemento está no último nó ou não existe (**$O(n)$**).

Inserção no Final

Para inserir no final, é preciso **percorrer a lista** até o último nó.

```
inserir_fim(valor):  
    novo <- novo No(valor)  
    se esta_vazia() então  
        primeiro <- novo  
    senao:  
        atual <- primeiro  
        enquanto atual.proximo != None faça:  
            atual <- atual.proximo  
        atual.proximo <- novo  
    tamanho <- tamanho + 1
```

Inserir no final é **$O(n)$** , pois depende do tamanho da lista.

Remoção de Elementos

Para remover, localizamos o nó anterior e "pulamos" o nó a ser removido, ajustando a referência `proximo`.

1. Se a lista estiver vazia, retornamos uma mensagem de erro.
2. Se o nó a ser removido for o `primeiro`, apenas ajustamos o ponteiro `primeiro` para o próximo nó da lista.
3. Caso contrário, percorremos a lista até encontrar o nó a ser removido
4. Se o nó não for encontrado, retornamos uma mensagem de erro.
5. Se o nó for encontrado, ajustamos o ponteiro `proximo` do nó anterior para o próximo nó do nó removido e diminuimos o tamanho da lista.

```
remover(valor):  
    se esta_vazia() então  
        retorne "Lista vazia"  
  
    se primeiro.valor == valor então  
        primeiro <- primeiro.proximo  
        tamanho <- tamanho - 1  
        retorne "Removido"  
  
    anterior <- primeiro  
    atual <- primeiro.proximo  
    enquanto atual != None faça:  
        se atual.valor == valor então  
            anterior.proximo <- atual.proximo  
            tamanho <- tamanho - 1  
            retorne "Removido"  
        anterior <- atual  
        atual <- atual.proximo  
  
    retorne "Não encontrado"
```

Removendo o Último Elemento

Ao remover o último nó, o nó anterior passa a apontar para `None`.

```
# lista: 10 -> 20 -> 30
# remover(30)

anterior.proximo <- atual.proximo    # atual.proximo é None
# lista: 10 -> 20 -> None
```

Sempre que removemos o `primeiro`, o novo `primeiro` é o segundo nó (ou `None`).

Lista Encadeada vs Vetor

Característica	Vetor	Lista Encadeada
Tamanho	Fixo	Dinâmico
Acesso por índice	Direto ($O(1)$)	Percorrendo ($O(n)$)
Inserir no início	Custo de deslocar ($O(n)$)	Ajuste de ponteiros ($O(1)$)
Inserir no final	Direto, se houver espaço	Percorrendo ($O(n)$)
Memória	Contígua	Dispersa (nós + ponteiros)

Resumo

- Lista encadeada é **dinâmica** e formada por **nós**.
- Cada nó tem **valor** e **próximo**.
- O ponteiro **primeiro** aponta para o início da lista; `None` indica o fim.
- Inserir/remover no **início** é **$O(1)$** .
- Pesquisar e percorrer é **$O(n)$** .
- Não há `capacidade` nem `esta_cheia()`.