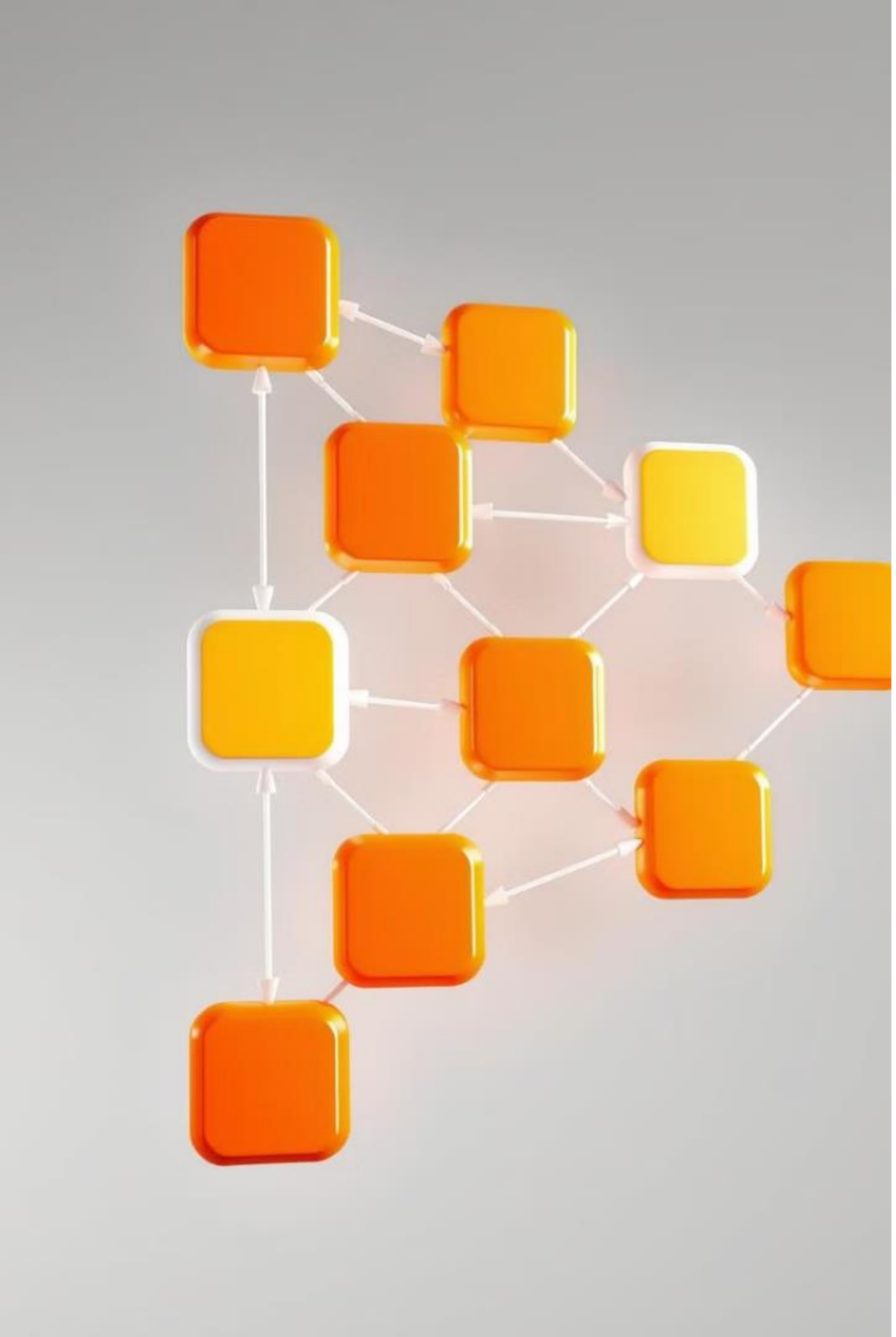
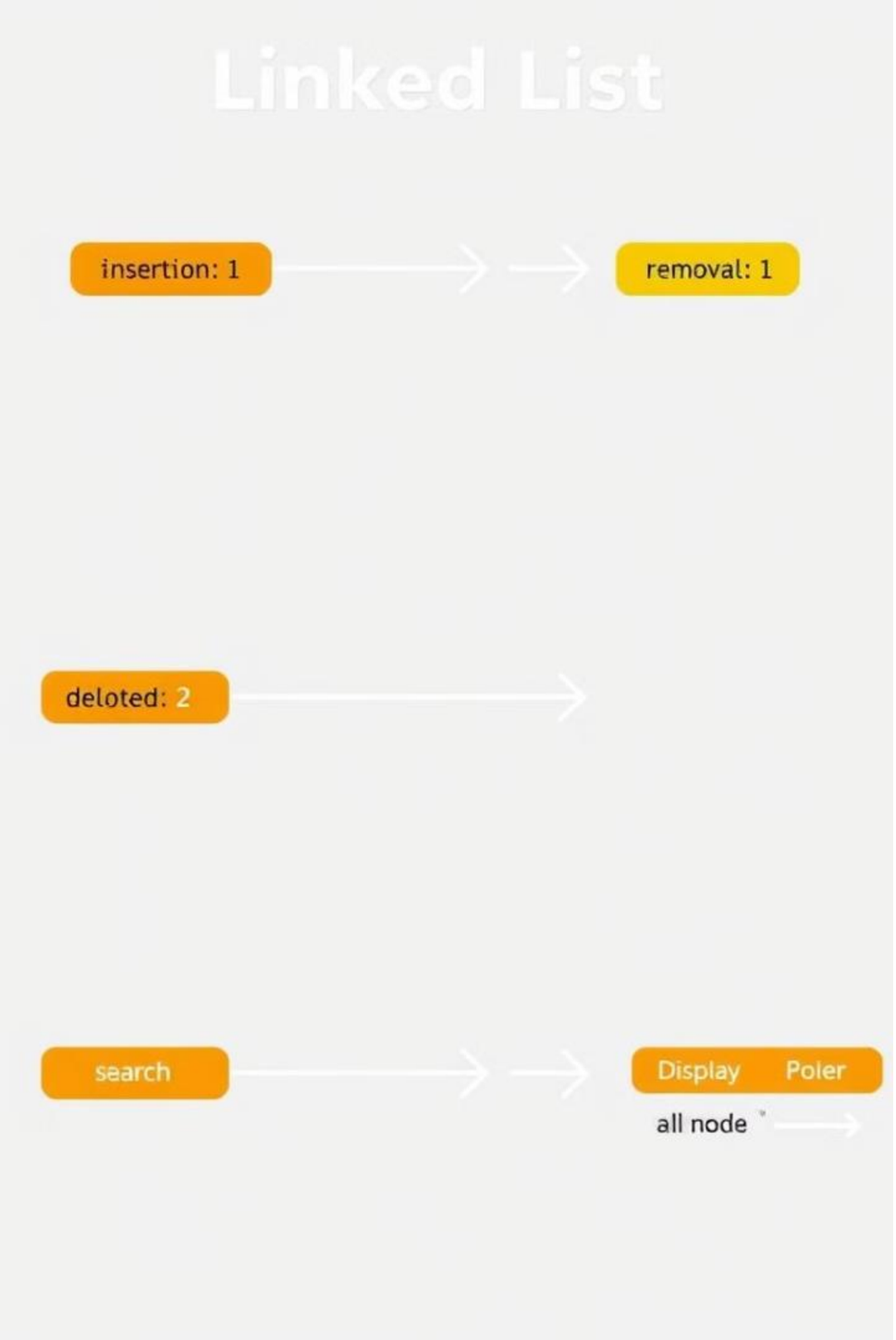


Conteúdo de Revisão: Listas Encadeadas

Nesta aula, revisaremos os conceitos fundamentais de listas encadeadas, explorando suas principais variações: listas simplesmente encadeadas, listas duplamente encadeadas e listas circulares. Cada tipo de lista possui características e aplicações específicas, que serão abordadas em detalhes ao longo desta revisão.

As listas encadeadas são estruturas de dados essenciais na programação, oferecendo flexibilidade na alocação de memória e manipulação de dados. Vamos examinar suas operações básicas e implementações em diferentes contextos.

por William Junior



Conceitos Básicos de Listas Encadeadas

Listas encadeadas são estruturas de dados compostas por nós, onde cada nó armazena um elemento e uma referência (ponteiro) para o próximo nó na sequência. Essa estrutura permite uma alocação dinâmica de memória, sendo ideal para cenários onde o número de elementos pode variar durante a execução do programa.

As operações básicas em listas encadeadas incluem inserção, remoção, busca e exibição de elementos. Essas operações formam a base para a manipulação eficiente dos dados armazenados na lista.

Inserção

Adiciona um novo elemento na lista.

Remoção

Remove um elemento da lista.

Busca

Encontra um elemento específico.

Exibição

Exibe todos os elementos da lista.

Listas Simplesmente Encadeadas

Em uma lista simplesmente encadeada, cada nó contém um valor e um ponteiro para o próximo nó na lista. A navegação é unidirecional, ocorrendo do primeiro para o último nó, sem possibilidade de voltar. Essa estrutura é eficiente para operações que requerem apenas travessia em uma direção.

Vamos examinar o pseudocódigo e o código em C para a inserção na cabeça de uma lista simplesmente encadeada:

Pseudocódigo

1

```
def inserir_na_cabeca(valor): novo_no = No(valor)
    novo_no.proximo = cabeca cabeca = novo_no
```

Código C

2

```c
struct Node{
    int data;
    struct Node* prox;
};
void inserir_na_cabeca(struct Node** cabeca, int novo_dado){
    struct Node* novo_no = (struct Node*) malloc(sizeof(struct Node));
    novo_no->data = novo_dado;
    novo_no->prox = *cabeca;
    *cabeca = novo_no;
}
```

Listas Duplamente Encadeadas

Em uma lista duplamente encadeada, cada nó possui um ponteiro para o próximo nó e um ponteiro para o nó anterior. Essa estrutura permite uma navegação bidirecional, facilitando operações que requerem acesso tanto ao próximo quanto ao nó anterior.

Vamos analisar o pseudocódigo e o código em C para a inserção na cabeça de uma lista duplamente encadeada:

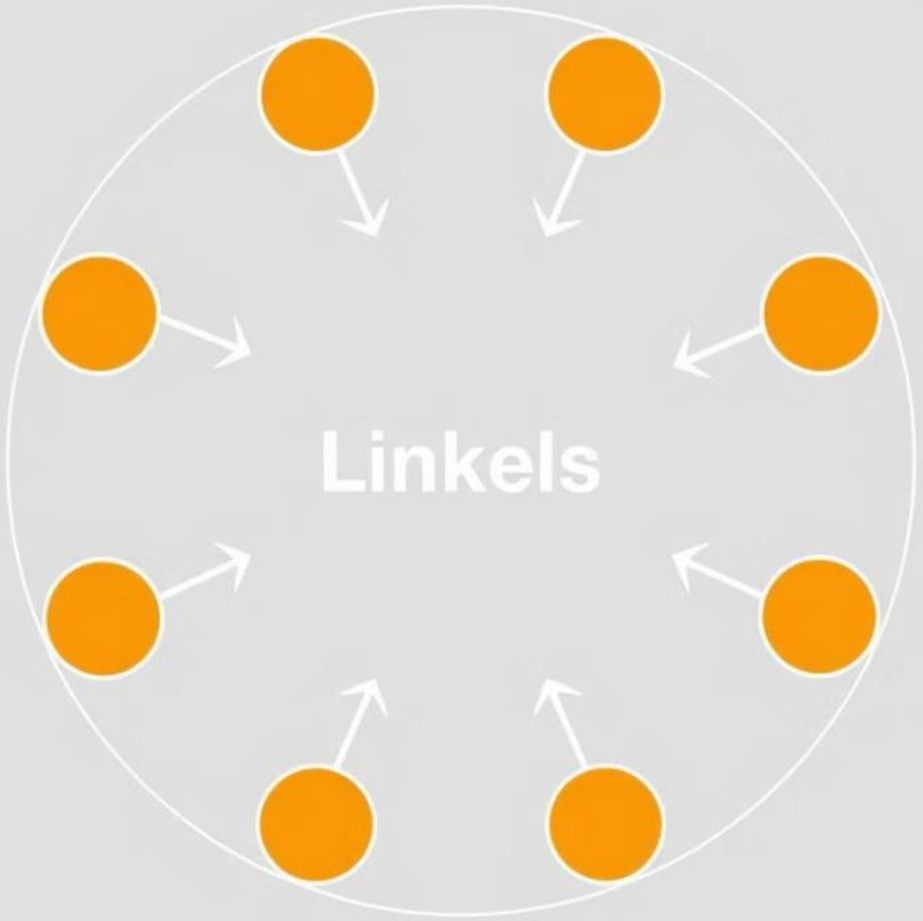
Pseudocódigo

def inserir_na_cabeca(valor):
novo_no = No(valor) novo_no.proximo = cabeca if cabeca != None: cabeca.anterior = novo_no cabeca = novo_no

Código C

```c
struct Node{
    int data;
    struct Node* prox;
    struct Node* ant;
};
void inserir_na_cabeca(struct Node** cabeca, int novo_dado){
    struct Node* novo_no = (struct Node*) malloc(sizeof(struct Node));
    novo_no->data = novo_dado;
    novo_no->prox = (*cabeca);
    novo_no->ant = NULL;
    if ((*cabeca) != NULL)
        (*cabeca)->ant = novo_no;

    (*cabeca) = novo_no;
}
```



Listas Circulares

Nas listas circulares, o último nó aponta de volta para o primeiro, criando um ciclo. Elas podem ser simplesmente ou duplamente encadeadas e são úteis em aplicações onde é necessário fazer uma navegação contínua pelos elementos da lista.

Vamos examinar o pseudocódigo para inserção em uma lista circular simplesmente encadeada:

1

Inicialização

Verificar se a lista está vazia e criar o primeiro nó.

2

Travessia

Percorrer a lista até encontrar o último nó.

3

Inserção

Inserir o novo nó e ajustar os ponteiros para manter a circularidade.

Implementação de Lista Circular em C

Vamos analisar o código C para inserção no final de uma lista circular simplesmente encadeada:

Implementação

Código em C para inserção em lista circular.

```
void inserir_no_final(struct Node** head_ref, int novo_dado) {
    struct Node* novo_no = (struct Node*) malloc(sizeof(struct Node));
    struct Node *temp = *head_ref;
    novo_no->data = novo_dado;
    novo_no->next = *head_ref;
    if (*head_ref == NULL) {
        *head_ref = novo_no;
        novo_no->next = *head_ref;
    } else {
        while (temp->next != *head_ref)
            temp = temp->next;
        temp->next = novo_no;
    }
}
```

Circularidade

Mantém a estrutura circular da lista.

Alocação

Usa alocação dinâmica de memória.

Comparação entre Tipos de Listas Encadeadas

Cada tipo de lista encadeada apresenta vantagens e desafios específicos. A escolha do tipo adequado depende dos requisitos de navegação e atualização do programa.

Tipo de Lista	Vantagens	Desvantagens
Simplesmente Encadeada	Simples implementação, menor uso de memória	Navegação unidirecional
Duplamente Encadeada	Navegação bidirecional, facilita remoções	Maior uso de memória
Circular	Navegação contínua, útil para estruturas cíclicas	Complexidade na implementação

Conclusão e Práticas Recomendadas

As listas encadeadas são estruturas de dados dinâmicas e versáteis, essenciais para muitos algoritmos e programas. Cada tipo de lista encadeada oferece características únicas que se adaptam a diferentes necessidades de programação.

Para consolidar o conhecimento, é fundamental praticar a implementação de cada tipo de lista. Experimente criar, manipular e otimizar diferentes estruturas de listas encadeadas em seus projetos.

1 **Prática Constante**

Implemente diferentes tipos de listas em seus projetos.

2 **Análise de Desempenho**

Compare o desempenho de diferentes tipos de listas em várias operações.

3 **Resolução de Problemas**

Use listas encadeadas para resolver problemas algorítmicos complexos.

4 **Otimização**

Aprenda a otimizar as operações em listas para melhorar a eficiência.