



LISTAS DINAMICAMENTE ENCADEADAS EM JAVASCRIPT

ESTRUTURA DE DADOS

CST em Desenvolvimento de Software Multiplataforma



PROF. Me. TIAGO A. SILVA



PARA SOBREVIVER AO JAVASCRIPT

Non-zero value



null



0

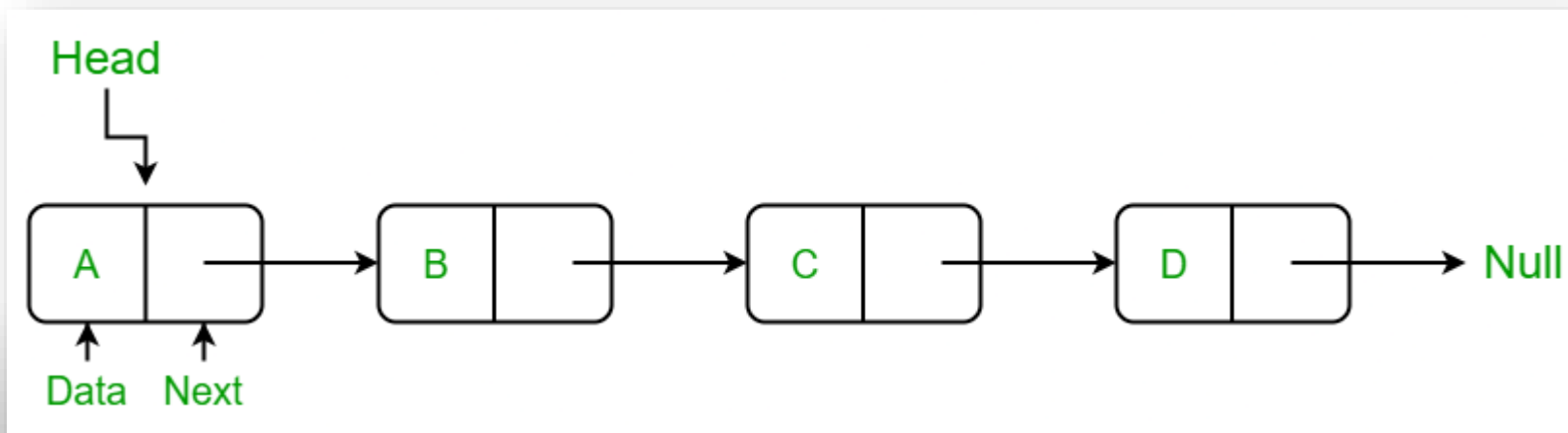


undefined



O QUE SÃO LISTAS ENCADEADAS?

- As listas dinamicamente encadeadas (ou simplesmente listas encadeadas) são estruturas de dados compostas por nós (Node) que, juntos, formam uma sequência. Cada nó contém um elemento de dados e uma referência (ou ponteiro) para o próximo nó na sequência. Isso difere dos arrays tradicionais, que armazenam elementos em locais contíguos na memória.



ESTRUTURA DE UMA LISTA ENCADEADA

- Cada elemento de uma lista encadeada é chamado de nó. Um nó geralmente possui duas partes:
 - **Dados:** O valor que queremos armazenar.
 - **Referência:** Um ponteiro para o próximo nó na lista.
- Em JavaScript, podemos representar um nó (Node) como um objeto. Veja um exemplo básico da estrutura de um nó:

```
1  class Node {  
2      constructor(value) {  
3          this.value = value;  
4          this.next = null; // Ponteiro para o próximo nó  
5      }  
6  }
```

O QUE VAMOS IMPLEMENTAR?

```
8  class LinkedList {
9      constructor() {
10         this.head = null; // A lista começa vazia
11     }
12
13     // Inserir no início da lista
14 > insertAtBeginning(value) { ...
18     }
19
20     // Inserir no fim da lista
21 > insertAtEnd(value) { ...
38     }
39
40     // Remover um nó por valor
41 > removeByValue(value) { ...
62     }
63
64     // Buscar um nó por valor
65 > find(value) { ...
77     }
78
79     // Exibir a lista (opcional, para facilitar a visualização)
80 > printList() { ...
88     }
89 }
```

- A lista encadeada, por sua vez, pode ser representada por uma classe que mantém referência ao primeiro nó da lista, chamado de cabeça (ou **head**). Inicialmente, a lista começa vazia, então **head** é **null**.

```
8  class LinkedList {  
9      constructor() {  
10         this.head = null;  
11     }  
12 }
```

INSERÇÃO NO INÍCIO

- **Adicionar um novo nó no início de uma lista encadeada é uma operação direta.** Precisamos criar um novo nó e fazer com que sua referência (**next**) aponte para o antigo primeiro nó. Em seguida, atualizamos o ponteiro **head** para o novo nó.

```
15  class LinkedList {
16      constructor() {
17          this.head = null;
18      }
19
20      insertAtBeginning(value) {
21          let newNode = new Node(value);
22          newNode.next = this.head;
23          this.head = newNode;
24      }
25  }
```


INSERÇÃO NO FIM

- Para adicionar um nó ao final da lista, percorremos a lista até o último nó, ou seja, o nó cuja referência **next** é **null**. Então, fazemos o **next** desse nó apontar para o novo nó.

```
27 class LinkedList {
28     constructor() {
29         this.head = null;
30     }
31
32     insertAtEnd(value) {
33         let newNode = new Node(value);
34
35         if (this.head === null) {
36             this.head = newNode;
37             return;
38         }
39
40         let current = this.head;
41         while (current.next !== null) {
42             current = current.next;
43         }
44         current.next = newNode;
45     }
46 }
```

REMOÇÃO DE UM NÓ

- Para remover um nó, devemos encontrar o nó anterior ao que queremos remover. Depois, atualizamos o ponteiro **next** desse nó anterior para pular o nó que será removido.
- Veja a seguir a remoção de um nó específico (por valor):

```
48 class LinkedList {
49     constructor() {
50         this.head = null;
51     }
52
53     removeByValue(value) {
54         if (this.head === null) {
55             return;
56         }
57
58         // Se o nó a ser removido for o primeiro
59         if (this.head.value === value) {
60             this.head = this.head.next;
61             return;
62         }
63
64         let current = this.head;
65         while (current.next !== null && current.next.value !== value) {
66             current = current.next;
67         }
68
69         if (current.next !== null) {
70             current.next = current.next.next;
71         }
72     }
73 }
```

BUSCA DE UM NÓ

- Para buscar um nó com um valor específico, simplesmente percorremos a lista comparando o valor de cada nó com o valor que estamos procurando.

```
75 class LinkedList {
76     constructor() {
77         this.head = null;
78     }
79
80     find(value) {
81         let current = this.head;
82
83         while (current !== null) {
84             if (current.value === value) {
85                 return current;
86             }
87             current = current.next;
88         }
89
90         return null; // Retorna null se o valor não for encontrado
91     }
92 }
```

MOSTRANDO CONTEÚDO DA LISTA

```
93     printList() {  
94         let current = this.head;  
95         let list = '';  
96         while (current !== null) {  
97             list += current.value + ' -> ';  
98             current = current.next;  
99         }  
100        console.log(list + 'null');  
101    }
```

VANTAGENS E DESVANTAGENS DAS LISTAS ENCADEADAS

- As listas encadeadas têm várias vantagens sobre arrays, mas também apresentam desvantagens que devem ser levadas em conta.
- **VANTAGENS:**
 - **Inserções e remoções rápidas:** Inserir ou remover um elemento de uma lista encadeada pode ser feito em tempo constante, desde que a posição do nó seja conhecida. Em um array, inserir ou remover elementos pode exigir o deslocamento de vários elementos, especialmente em listas grandes.
 - **Tamanho dinâmico:** Ao contrário dos arrays, que têm tamanho fixo, listas encadeadas podem crescer ou diminuir dinamicamente conforme necessário.

VANTAGENS E DESVANTAGENS DAS LISTAS ENCADEADAS

- **DESVANTAGENS:**

- **Acesso sequencial:** Diferente de arrays, onde o acesso a um elemento é feito por índice de maneira constante, nas listas encadeadas o acesso a um nó específico requer a travessia da lista, resultando em um tempo de busca linear.
- **Mais uso de memória:** Cada nó em uma lista encadeada armazena não apenas os dados, mas também um ponteiro para o próximo nó. Isso pode resultar em um uso adicional de memória comparado a um array.

EXERCÍCIOS DE FIXAÇÃO

- **1) Inserir elementos no início e no fim da lista**
 - Implemente uma função que crie uma lista encadeada e insira os valores 10, 20 e 30 no início da lista e os valores 40, 50 e 60 no fim.

```
95  let list = new LinkedList();
96
97  // Inserindo no início
98  list.insertAtBeginning(30);
99  list.insertAtBeginning(20);
100 list.insertAtBeginning(10);
101
102 // Inserindo no fim
103 list.insertAtEnd(40);
104 list.insertAtEnd(50);
105 list.insertAtEnd(60);
```


EXERCÍCIOS DE FIXAÇÃO

- **2) Remover um valor específico**
 - Implemente a remoção de um valor específico e teste a remoção de diferentes valores da lista encadeada.
- **3) Buscar um valor na lista**
 - Implemente uma função de busca e teste-a para verificar se um valor está presente na lista.

OBRIGADO!

- Encontre este **material on-line** em:
 - www.tiago.blog.br
 - Plataforma Teams
- Em caso de **dúvidas**, entre em contato:
 - **Prof. Tiago:** tiago.silva238@fatec.sp.gov.br

