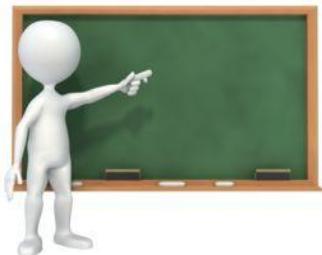




LISTAS DINAMICAMENTE ENCADEADAS

ESTRUTURA DE DADOS

CST em Desenvolvimento de Software Multiplataforma



PROF. Me. TIAGO A. SILVA



LIVRO DE REFERÊNCIA DA DISCIPLINA

- **BIBLIOGRAFIA BÁSICA:**

- GRONER, Loiane. **Estrutura de dados e algoritmos com JavaScript**: escreva um código JavaScript complexo e eficaz usando a mais recente ECMAScript. São Paulo: Novatec Editora, 2019.



- **NESTA AULA:**

- **Capítulo 6 – Listas Ligadas**

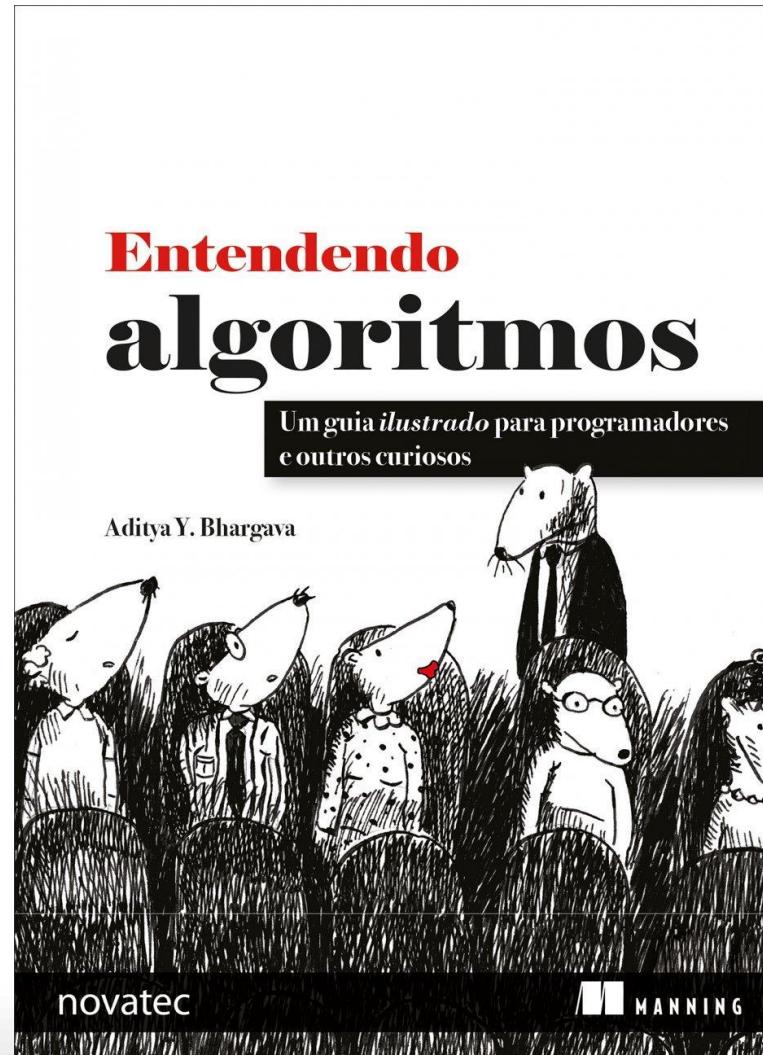
LIVRO DE APOIO DESSA AULA

- **BIBLIOGRAFIA COMPLEMENTAR:**

- BHARGAVA, Aditya Y. **Entendendo Algoritmos: Um guia ilustrado para programadores e outros curiosos.** São Paulo: Novatec Editora, 2018.

- **NESTA AULA:**

- Capítulo 2 – Listas Encadeadas



PARA SOBREVIVER AO JAVASCRIPT

Non-zero value



0



null



undefined

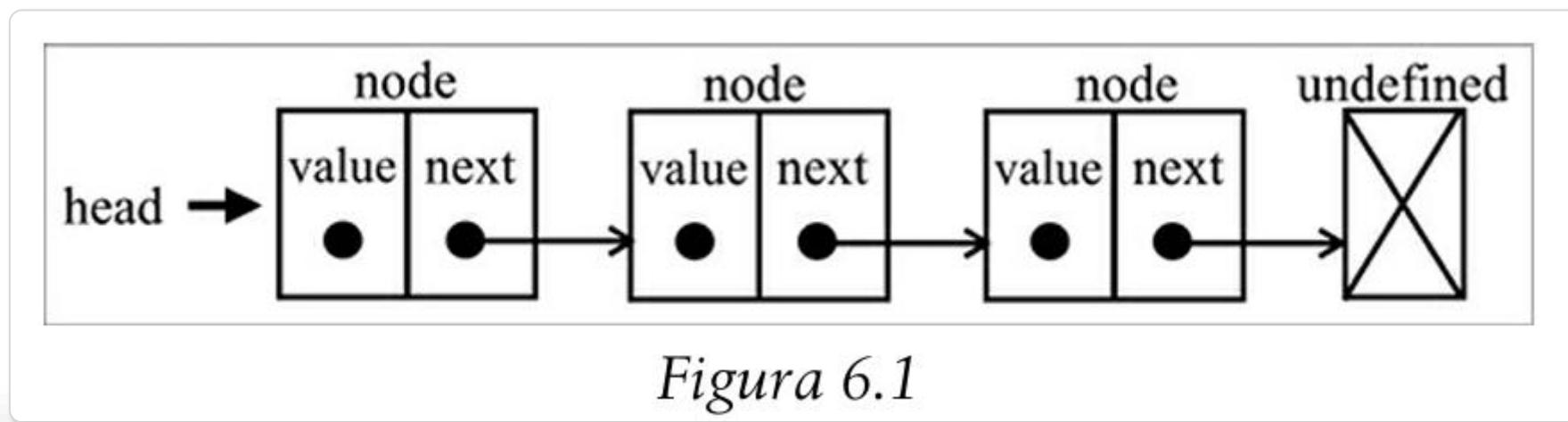


LISTA DINAMICAMENTE ENCADEADA

*Primeira estrutura que não usará um array
como base*

O QUE É UMA LISTA DINAMICAMENTE ENCADEADA?

- 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.



CLASSE QUE SERÁ CADA UM DOS NÓS DA LISTA

- 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.

● ● ● Aula 06 - LinkedList - LinkedList.js

```
1 class Node {  
2  
3     constructor(value) {  
4  
5         this.value = value;  
6  
7         // Ponteiro para o próximo nó  
8         this.next = undefined;  
9     }  
10 }  
11
```

IMPLEMENTAÇÃO DO PROTÓTIPO DA CLASSE

● ● ● Aula 06 - LinkedList - LinkedList_prototipo.js

```
1 class LinkedList {  
2  
3     #head = undefined;  
4     #length = 0;  
5  
6     insertAtBeginning(value) {    }  
7     insertAtEnd(value) {    }  
8     removeByValue(value) {    }  
9     find(value) {    }  
10    size() {    }  
11    isEmpty() {    }  
12    toString() {    }  
13}  
14  
15 module.exports = LinkedList;
```



IMPLEMENTAÇÃO DO MÉTODO `insertAtBeginning(value)`

- 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.



Aula 06 - LinkedList - LinkedList.js

```
21 // Inserir no início da lista
22 insertAtBeginning(value) {
23
24     // Cria um novo nó
25     let newNode = new Node(value);
26
27     // O novo nó aponta para o antigo "head"
28     newNode.next = this.#head;
29
30     // Atualiza o "head" para o novo nó
31     this.#head = newNode;
32
33     // Incrementa o tamanho da lista
34     this.#length++;
35 }
```

IMPLEMENTAÇÃO DO MÉTODO `insertAtEnd(value)`

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



Aula 06 - LinkedList - LinkedList.js

```
37 // Inserir no fim da lista
38 insertAtEnd(value) {
39     let newNode = new Node(value);
40
41     // Se a lista estiver vazia, o novo nó se torna o "head"
42     if (this.#head === undefined) {
43         this.#head = newNode;
44         return;
45     }
46
47     // Percorre até o último nó
48     let current = this.#head;
49     while (current.next !== undefined) {
50         current = current.next;
51     }
52
53     // Faz o último nó apontar para o novo nó
54     current.next = newNode;
55
56     this.#length++;
57 }
```

IMPLEMENTAÇÃO DO MÉTODO `removeByValue(value)`



Aula 06 - LinkedList - LinkedList.js

```
59 // Remover um nó por valor
60 removeByValue(value) {
61
62     // A lista está vazia
63     if (this.#head === undefined) {
64         return;
65     }
66
67     // Se o valor a ser removido estiver no primeiro nó
68     if (this.#head.value === value) {
69
70         // Atualiza o "head" para o próximo nó
71         this.head = this.#head.next;
72         return;
73     }
74
75     // Percorre a lista procurando o nó a ser removido
76     let current = this.#head;
77     while (current.next !== undefined && current.next.value !== value) {
78         current = current.next;
79     }
80
81     // Se o nó foi encontrado, ajusta o ponteiro para pular o nó removido
82     if (current.next !== undefined) {
83         current.next = current.next.next;
84     }
85
86     this.#length--;
87 }
```

- 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.

IMPLEMENTAÇÃO DO MÉTODO `find(value)`



Aula 06 - LinkedList - LinkedList.js

```
89 // Buscar um nó por valor
90 find(value) {
91     let current = this.#head;
92
93     // Percorre a lista até encontrar o nó ou até o final
94     while (current !== undefined) {
95         console.log(current.value);
96         if (current.value === value) {
97             return current; // Nó encontrado
98         }
99         current = current.next;
100    }
101
102    return undefined; // Se não encontrar, retorna null
103 }
```



IMPLEMENTAÇÃO DO MÉTODO size()



Aula 06 - LinkedList - LinkedList.js

```
116 // Retorna o tamanho da lista
117 size() {
118     let count = 0;
119     let current = this.#head;
120     while (current !== undefined) {
121         count++;
122         current = current.next;
123     }
124     return count;
125 }
```



IMPLEMENTAÇÃO DO MÉTODO isEmpty()



Aula 06 - LinkedList - LinkedList.js

```
127 // Retorna true se a lista estiver vazia  
128 isEmpty = () => this.#head === undefined;
```



IMPLEMENTAÇÃO DO MÉTODO `toString()`



Aula 06 - LinkedList - LinkedList.js

```
105 // Exibir a lista (opcional, para facilitar a visualização)
106 toString() {
107     let current = this.#head;
108     let list = '';
109     while (current !== undefined) {
110         list += current.value + ' -> ';
111         current = current.next;
112     }
113     console.log(list + 'undefined');
114 }
```



VANTAGENS DA LISTA ENCADEADA

- **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.

DESVANTAGENS DA LISTA ENCADEADA

- **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.

COMO USAR A CLASSE?



Aula 06 - LinkedList - app.js

```
1 const LinkedList = require('./LinkedList.js');
2
3 const lista = new LinkedList();
4
5 console.log("\n-----");
6 console.log(`\x1b[35m${`\x1b[0m`}` , 'Lista Inicial'});
7 lista.isEmpty() ? console.log("A lista está vazia.") : console.log("A lista não está vazia.");
8
9
10 console.log("\n-----");
11 console.log(`\x1b[32m${`\x1b[0m`}` , 'Inserindo no início:');
12 lista.insertAtBeginning(30);
13 lista.toString(); // Saída: 30 -> undefined
14 lista.insertAtBeginning(20);
15 lista.toString(); // Saída: 20 -> 30 -> undefined
16 lista.insertAtBeginning(10);
17 lista.toString(); // Saída: 10 -> 20 -> 30 -> undefined
18 console.log(`Tamanho da lista: ${lista.size()}`); // Saída: Tamanho da lista: 3
```

SAÍDA NO CONSOLE

Lista Inicial

A lista está vazia.

Inserindo no início:

30 -> undefined

20 -> 30 -> undefined

10 -> 20 -> 30 -> undefined

Tamanho da lista: 3



COMO USAR A CLASSE?



Aula 06 - LinkedList - app.js

```
21 console.log("\n-----");
22 console.log(`\x1b[32m${`x1b[0m`}`, 'Inserindo no fim:'});
23 lista.insertAtEnd(40);
24 lista.toString(); // Saída: 10 -> 20 -> 30 -> 40 -> undefined
25 lista.insertAtEnd(50);
26 lista.toString(); // Saída: 10 -> 20 -> 30 -> 40 -> 50 -> undefined
27 console.log(`Tamanho da lista: ${lista.size()}`); // Saída: Tamanho da lista: 5
28
29
30 console.log("\n-----");
31 console.log(`\x1b[32m${`x1b[0m`}`, 'Exibindo a lista:'});
32 lista.toString(); // Saída: 10 -> 20 -> 30 -> 40 -> 50 -> null
```

SAÍDA NO CONSOLE

```
Inserindo no fim:
```

```
10 -> 20 -> 30 -> 40 -> undefined
```

```
10 -> 20 -> 30 -> 40 -> 50 -> undefined
```

```
Tamanho da lista: 5
```

```
Exibindo a lista:
```

```
10 -> 20 -> 30 -> 40 -> 50 -> undefined
```



COMO USAR A CLASSE?



Aula 06 - LinkedList - app.js

```
35 console.log("\n-----");
36 console.log('\x1b[31m%s\x1b[0m', 'Removendo Itens:');
37 // Removendo um valor
38 lista.removeByValue(30);
39 lista.toString(); // Saída: 10 -> 20 -> 40 -> 50 -> null
40
41 console.log("\n-----");
42 console.log('\x1b[34m%s\x1b[0m', 'Procurando um Valor:');
43 let node = lista.find(40);
44 console.log(node ? `Valor encontrado: ${node.value}` : "Valor não encontrado");
```

COMO USAR A CLASSE?

Removendo Itens:

10 -> 20 -> 40 -> 50 -> undefined

Procurando um Valor:

10

20

40

Valor encontrado: 40



LISTA DUPLAMENTE ENCADEADA

*Além de referenciar o próximo nó, agora
também referenciamos o anterior*

O QUE É UMA LISTA DUPLAMENTE ENCADEADA?

- Uma lista duplamente encadeada é uma estrutura de dados onde cada nó possui um ponteiro que aponta para o próximo e para o nó anterior.
- Diferentemente das **listas encadeadas simples, que só têm um ponteiro** para o próximo nó, a lista duplamente encadeada permite percorrer tanto no sentido direto (do início ao fim) quanto no inverso (do fim ao início).
- Isso a torna bastante útil quando precisamos realizar inserções, exclusões ou travessias em ambas as direções.

O QUE É UMA LISTA DUPLAMENTE ENCADEADA?

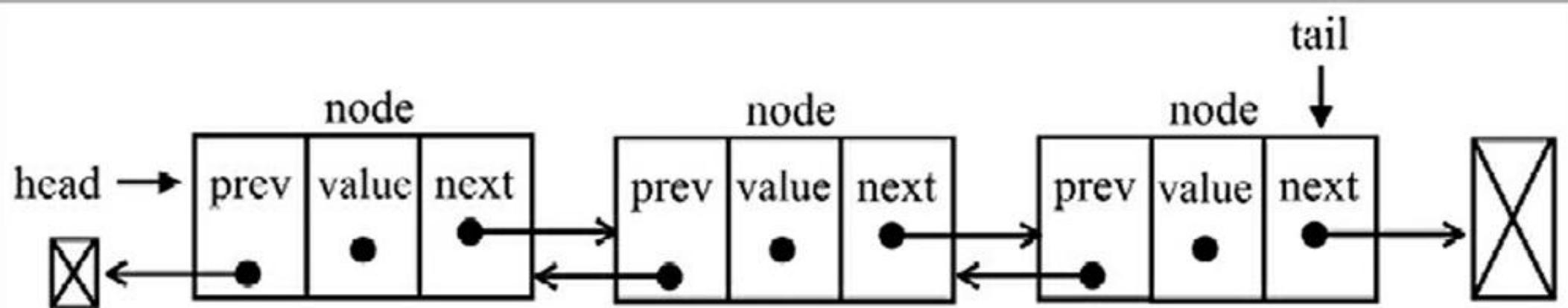


Figura 6.11

CLASSE QUE SERÁ CADA UM DOS NÓS DA LISTA



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
1 class Node {
2     constructor(value) {
3         this.value = value;
4         this.next = undefined; // Próximo nó
5         this.prev = undefined; // Nó anterior
6     }
7 }
8
```



IMPLEMENTAÇÃO DO PROTÓTIPO DA CLASSE

● ● ● Aula 07 - DoublyLinkedList - DoublyLinkedListPrototipo.js

```
10  class DoublyLinkedList {  
11  
12      #head = undefined; // Primeiro nó (cabeça)  
13      #tail = undefined; // Último nó (cauda)  
14      #length = 0; // Tamanho da lista  
15  
16      append(value) { }  
17      prepend(value) { }  
18      removeLast() { }  
19      removeFirst() { }  
20      traverse() { }  
21      traverseReverse() { }  
22      insertAt(value, index) { }  
23      find(value) { }  
24      removeAt(index) { }  
25      size = () => this.#length;  
26      isEmpty = () => this.#length === 0;  
27      toString() { }  
28  }  
29  
30  module.exports = DoublyLinkedList;
```



IMPLEMENTAÇÃO DO MÉTODO: append(value)



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
16 // Adicionar um nó ao final da lista
17 append(value) {
18     const newNode = new Node(value);
19
20     if (this.#head == undefined) {
21         this.#head = newNode;
22         this.#tail = newNode;
23     } else {
24         this.#tail.next = newNode;
25         newNode.prev = this.#tail;
26         this.#tail = newNode;
27     }
28
29     this.#length++;
30 }
```



IMPLEMENTAÇÃO DO MÉTODO prepend(value)

● ● ● Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
32 // Adicionar um nó ao início da lista
33 prepend(value) {
34     const newNode = new Node(value);
35
36     if (this.#head == undefined) {
37         this.#head = newNode;
38         this.#tail = newNode;
39     } else {
40         newNode.next = this.#head;
41         this.#head.prev = newNode;
42         this.#head = newNode;
43     }
44
45     this.#length++;
46 }
```



IMPLEMENTAÇÃO DO MÉTODO `removeLast()`



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
48 // Remover o nó do final da lista
49 removeLast() {
50     if (this.#tail == undefined) return undefined;
51
52     const removedNode = this.#tail;
53     if (this.#tail === this.#head) {
54         this.#head = null;
55         this.#tail = null;
56     } else {
57         this.#tail = this.#tail.prev;
58         this.#tail.next = undefined;
59     }
60
61     this.#length--;
62     return removedNode.value;
63 }
```



IMPLEMENTAÇÃO DO MÉTODO removeFirst()



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
65 // Remover o nó do inicio da lista
66 removeFirst() {
67     if (this.#head == undefined) return undefined;
68
69     const removedNode = this.#head;
70     if (this.#head === this.#tail) {
71         this.#head = undefined;
72         this.#tail = undefined;
73     } else {
74         this.#head = this.#head.next;
75         this.#head.prev = undefined;
76     }
77
78     this.#length--;
79     return removedNode.value;
80 }
```



IMPLEMENTAÇÃO DO MÉTODO `traverse()`



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
82 // Percorrer a lista do início ao fim
83 traverse() {
84
85     if(this.isEmpty()) {
86         console.log("A lista está vazia.");
87         return;
88     }
89
90     let current = this.#head;
91     while (current) {
92         console.log(current.value);
93         current = current.next;
94     }
95 }
```



IMPLEMENTAÇÃO DO MÉTODO `traverseReverse()`



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
97 // Percorrer a lista do fim ao início
98 traverseReverse() {
99
100    if(this.isEmpty()) {
101        console.log("A lista está vazia.");
102        return;
103    }
104
105    let current = this.#tail;
106    while (current) {
107        console.log(current.value);
108        current = current.prev;
109    }
110 }
```



IMPLEMENTAÇÃO DO MÉTODO `insertAt(value, index)`



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
112 // Inserir um nó em uma posição específica
113 insertAt(value, index) {
114     if (index < 0 || index > this.#length) return undefined;
115
116     const newNode = new Node(value);
117
118     if (index === 0) {
119         this.prepend(value);
120         return;
121     }
122
123     if (index === this.#length) {
124         this.append(value);
125         return;
126     }
```



IMPLEMENTAÇÃO DO MÉTODO `insertAt(value, index)`

● ● ● Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
128     let current = this.#head;
129     let previous;
130     let count = 0;
131
132     while (count < index) {
133         previous = current;
134         current = current.next;
135         count++;
136     }
137
138     newNode.next = current;
139     newNode.prev = previous;
140     previous.next = newNode;
141     current.prev = newNode;
142
143     this.#length++;
144 }
```



IMPLEMENTAÇÃO DO MÉTODO `find(value)`



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
146 // Encontrar o índice de um valor específico
147 find(value) {
148     let current = this.#head;
149     let index = 0;
150
151     while (current) {
152         if (current.value === value)
153             return index; // Valor encontrado
154
155         current = current.next;
156         index++;
157     }
158
159     return -1; // Valor não encontrado
160 }
```



IMPLEMENTAÇÃO DO MÉTODO removeAt(index)



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
162 // Remover um nó em uma posição específica
163 removeAt(index) {
164     if (index < 0 || index >= this.length) return null;
165
166     if (index === 0) return this.removeFirst();
167     if (index === this.length - 1) return this.removeLast();
168
169     let current = this.head;
170     let count = 0;
171
172     while (count < index) {
173         current = current.next;
174         count++;
175     }
176
177     current.prev.next = current.next;
178     current.next.prev = current.prev;
179
180     this.#length--;
181     return current.value;
182 }
```



IMPLEMENTAÇÃO DOS MÉTODOS size E isEmpty



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
184 // Retorna o tamanho da lista  
185 size = () => this.#length;  
186  
187 // Verifica se a lista está vazia  
188 isEmpty = () => this.#length === 0;
```



IMPLEMENTAÇÃO DO MÉTODO `toString()`



Aula 07 - DoubleLinkedList - DoublyLinkedList.js

```
190 // Exibir a lista (opcional, para facilitar a visualização)
191 toString() {
192     let current = this.#head;
193     let list = '';
194     while (current !== undefined) {
195         list += current.value + ' -> ';
196         current = current.next;
197     }
198     console.log(list + 'undefined');
199 }
```



VANTAGENS DA LISTA DUPLAMENTE ENCADEADA

- **Travessia bidirecional:** Uma lista duplamente encadeada pode ser percorrida em ambas as direções, o que é uma vantagem em comparação com listas encadeadas simples.
- **Inserção e remoção eficientes:** Como cada nó contém uma referência ao nó anterior, as operações de inserção e remoção em qualquer posição são mais rápidas do que em listas encadeadas simples.

DESVANTAGENS DA LISTA DUPLAMENTE ENCADEADA

- **Maior uso de memória:** Cada nó armazena dois ponteiros adicionais, o que aumenta o consumo de memória.
- **Maior complexidade de implementação:** O gerenciamento dos ponteiros `prev` e `next` exige mais cuidado para evitar erros.

EXERCÍCIOS

Use a classe implementada acima para resolver os exercícios

LinkedList - EXERCÍCIO 1

- Você está ajudando um explorador a planejar sua trilha de aventura na floresta. Cada ponto da trilha (nó) contém um local interessante, como uma cachoeira, uma caverna ou um mirante. O explorador quer começar a trilha em um ponto específico e adicionar novos pontos durante a jornada. Sua missão é ajudá-lo a:
 - Inserir pontos da trilha no início e no fim da lista de locais a serem visitados.
 - Depois, o explorador decide remover um local que descobriu ser muito perigoso.
 - Por fim, ele quer verificar se o mirante ainda faz parte da trilha.
 - Implemente uma lista encadeada para representar a trilha e resolva essas tarefas.

LinkedList - EXERCÍCIO 2

- Na corrida anual das tartarugas, cada tartaruga segue a outra formando uma fila (como uma lista encadeada). As tartarugas são muito pacientes, mas a tartaruga "Lenta" decide desistir da corrida, e outra, "Veloz", quer entrar na fila em uma posição específica.
- Sua missão é:
 - Inserir uma nova tartaruga no início da fila e outra no final.
 - Remover a tartaruga "Lenta" da corrida, se ela estiver na fila.
 - Encontrar a tartaruga "Veloz" para saber em que posição ela está.
 - Ajude as tartarugas a organizarem a fila e implemente essas operações usando uma lista encadeada.

LinkedList - EXERCÍCIO 3

- Em uma batalha épica, um grupo de heróis possui uma sequência de poderes especiais (representados por uma lista encadeada), e cada herói pode ativar um poder em sequência. No entanto, durante a batalha, eles precisam reorganizar e ajustar seus poderes:
 - Adicione um novo poder no início e outro no fim da lista de poderes.
 - Um dos poderes, "Raio Congelante", acabou se tornando ineficaz, então ele precisa ser removido.
 - Verifique se o poder "Escudo de Fogo" está disponível para o herói ativar.
 - Implemente essa lista encadeada para ajudar os heróis a gerenciar seus poderes durante a batalha.

LinkedList - EXERCÍCIO 4

a) Remover um valor específico

- Implemente a remoção de um valor específico e teste a remoção de diferentes valores da lista encadeada.

b) Buscar um valor na lista

- Implemente uma função de busca e teste-a para verificar se um valor está presente na lista.

TRABALHO SALVA-VIDAS

*Orientações de como realizar o trabalho
dessa semana*

TRABALHO SALVA-VIDAS DESSA AULA

- Instruções:
 - Orientações gerais de como enviar: veja o tópico “Trabalho Salva-vidas” da **Aula 1**.
 - Não se esqueça de marcar os perfis e das hashtags.
 - Escolha um dos exercícios de cada assunto para gravar.
 - Duração do Vídeo: aproximadamente **90 segundos**.
 - **Prazo: 25/09/2025**



OBRIGADO!

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

