Centro Paula Souza

Fatec Jahu

ALGORITMOS DE ORDENAÇÃO E BUSCA

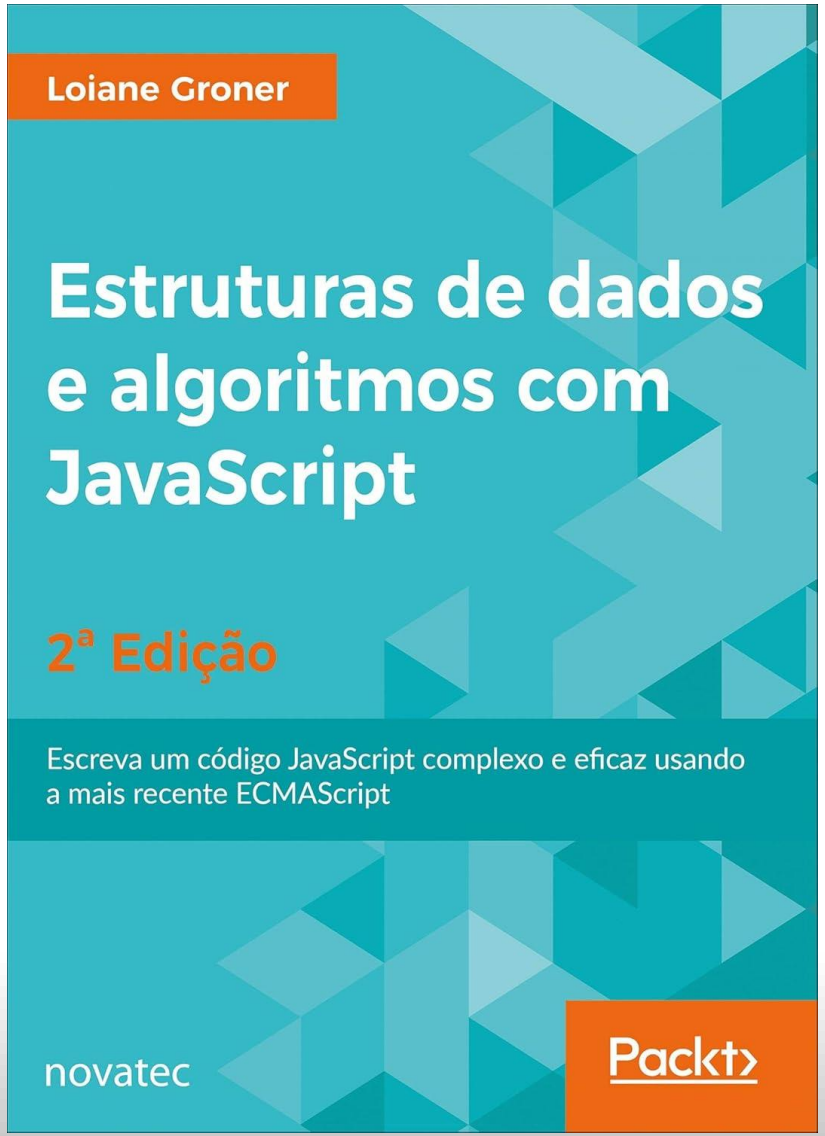
ESTRUTURA DE DADOS

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 ECMASript. **São Paulo: Novatec Editora**, 2019.
- **NESTA AULA:**
  - **Capítulo 13 – Busca e Ordenação**
    - Bubble Sort, Quick Sort e Merge Sort
    - Busca Binária, Sequencial e por Interpolação



PARA SOBREVIVER AO JAVASCRIPT

Non-zero value

0

null

undefined

ALGORITMOS DE ORDENAÇÃO

ALGORITMO BUBBLE SORT

- É um dos algoritmos de ordenação mais simples. Ele compara elementos adjacentes e os troca de lugar se estiverem na ordem errada. Esse processo é repetido até que a lista esteja ordenada.

- Como funciona:
  - O maior elemento "borbulha" para o final do array a cada passagem.
  - Após cada iteração completa, o próximo maior valor também estará em seu lugar correto.
  - O processo continua até que nenhuma troca seja necessária.

```
class Sorter {

  /**
   * Bubble Sort
   */
  static bubbleSort(arr) {
    const array = [...arr]; // copia para não alterar o original
    let n = array.length;
    let trocou;

    do {
      trocou = false;
      for (let i = 0; i < n - 1; i++) {
        if (array[i] > array[i + 1]) {
          [array[i], array[i + 1]] = [array[i + 1], array[i]];
          trocou = true;
        }
      }
      n--;
    } while (trocou);

    return array;
  }
```

ALGORITMO QUICK SORT

- O Quick Sort (ou ordenação rápida) é um algoritmo de ordenação muito eficiente, baseado na estratégia de dividir para conquistar (divide and conquer). É amplamente usado por sua boa performance na prática, mesmo com grandes volumes de dados.

- **Como funciona:**
  1) Escolhe um pivô (um elemento da lista).
  2) Particiona a lista em dois sub-arrays:
     - Um com os elementos menores que o pivô.
     - Outro com os elementos maiores que o pivô.
  3) Aplica o Quick Sort recursivamente em cada sub-array.
  4) Junta tudo: sub-array da esquerda + pivô + sub-array da direita.

IMPLEMENTAÇÃO QUICK SORT

```
25  /**
26   * Quick Sort
27   */
28  static quickSort(arr) {
29    if (arr.length <= 1) return arr;
30
31    const pivot = arr[arr.length - 1];
32    const menores = [];
33    const maiores = [];
34
35    for (let i = 0; i < arr.length - 1; i++) {
36      if (arr[i] < pivot) {
37        menores.push(arr[i]);
38      } else {
39        maiores.push(arr[i]);
40      }
41    }
42
43    return [...Sorter.quickSort(menores), pivot, ...Sorter.quickSort(maiores)];
44  }
```

ALGORITMO MERGE SORT

- O Merge Sort (ou ordenação por mistura) é um algoritmo de ordenação muito eficiente, baseado na estratégia de dividir para conquistar (divide and conquer), assim como o Quick Sort. Porém, sua abordagem é diferente: ele divide a lista completamente antes de começar a ordenar, e depois mescla os pedaços ordenados.

- **Como funciona:**
  - Divide recursivamente o array ao meio até que cada parte tenha apenas 1 elemento.
  - Depois, mescla (merge) os pedaços pequenos de forma ordenada.
  - Ao final, todas as partes estarão fundidas em um array ordenado.

IMPLEMENTAÇÃO MERGE SORT

```
46  /**
47   * Merge Sort
48   */
49  static mergeSort(arr) {
50    if (arr.length <= 1) return arr;
51
52    const meio = Math.floor(arr.length / 2);
53    const esquerda = Sorter.mergeSort(arr.slice(0, meio));
54    const direita = Sorter.mergeSort(arr.slice(meio));
55
56    return Sorter.merge(esquerda, direita);
57  }
58
```

IMPLEMENTAÇÃO MERGE SORT

```
59    static merge(esquerda, direita) {
60      const resultado = [];
61      let i = 0, j = 0;
62
63      while (i < esquerda.length && j < direita.length) {
64        if (esquerda[i] < direita[j]) {
65          resultado.push(esquerda[i++]);
66        } else {
67          resultado.push(direita[j++]);
68        }
69      }
70
71      return resultado.concat(esquerda.slice(i)).concat(direita.slice(j));
72    }
73  }
74
75  module.exports = Sorter;
```

EXEMPLO DE USO ALGORITMOS DE ORDENAÇÃO

```
1   const Sorter = require('./Sorter');
2
3   const lista = [7, 2, 5, 1, 9];
4
5   console.log("Bubble Sort:", Sorter.bubbleSort(lista));
6   console.log("Quick Sort:", Sorter.quickSort(lista));
7   console.log("Merge Sort:", Sorter.mergeSort(lista));
```

ALGORITMOS DE BUSCA

ALGORITMO DE BUSCA SEQUENCIAL

- O algoritmo de busca sequencial, também chamado de busca linear, é um dos **métodos mais simples** para encontrar um elemento dentro de uma estrutura de dados, como um vetor ou uma lista.

- Ele percorre a estrutura elemento por elemento, do início ao fim, comparando cada item com o valor que está sendo buscado. Se encontrar, retorna a posição (índice) do elemento; se não encontrar, retorna -1.

```
1   class Buscas {
2     // Busca Sequencial
3     static sequencial(arr, valorProcurado) {
4       for (let i = 0; i < arr.length; i++) {
5         if (arr[i] === valorProcurado) {
6           return i;
7         }
8       }
9       return -1;
10    }
```

COMO FUNCIONA O ALGORITMO DE BUSCA BINÁRIA?

- A busca binária é um algoritmo muito eficiente para encontrar elementos em uma lista ordenada. Ela funciona dividindo repetidamente o espaço de busca pela metade até encontrar o elemento desejado (ou concluir que ele não existe).

- Como funciona a busca binária?
  1) A lista precisa estar ordenada (do menor para o maior, ou vice-versa).
  2) O algoritmo compara o elemento do meio com o valor procurado.
  3) Se forem iguais, retorna a posição.
  4) Se o valor procurado for menor, repete a busca na metade esquerda.
  5) Se for maior, busca na metade direita.
  6) Repete até encontrar ou acabar a busca.

ALGORITMO DE BUSCA BINÁRIA

```
11
12  // Busca Binária (para array ORDENADO)
13  static binaria(arr, valorProcurado) {
14    let inicio = 0;
15    let fim = arr.length - 1;
16
17    while (inicio <= fim) {
18      let meio = Math.floor((inicio + fim) / 2);
19
20      if (arr[meio] === valorProcurado) {
21        return meio;
22      } else if (arr[meio] < valorProcurado) {
23        inicio = meio + 1;
24      } else {
25        fim = meio - 1;
26      }
27    }
28
29    return -1;
30  }
```

COMO FUNCIONA O ALGORITMO DE BUSCA POR INTERPOLAÇÃO?

- A busca por interpolação (ou interpolation search) é uma variação da busca binária que tenta adivinhar onde está o valor procurado, com base na distribuição dos dados. Ela é mais eficiente que a busca binária em alguns casos específicos — especialmente quando os dados estão ordenados e distribuídos uniformemente.

- Em vez de sempre dividir a lista ao meio (como na busca binária), a interpolação estima a posição provável do valor com base em uma fórmula de interpolação linear.

$$pos = início + \left(\frac{valor - arr[início]}{arr[fim] - arr[início]} \right) \times (fim - início)$$

ALGORITMO DE BUSCA POR INTERPOLAÇÃO

```
31
32  // Busca por Interpolação (para array ORDENADO e UNIFORMEMENTE distribuído)
33  static interpolacao(arr, valorProcurado) {
34    let inicio = 0;
35    let fim = arr.length - 1;
36
37    while (
38      inicio <= fim &&
39      valorProcurado >= arr[inicio] &&
40      valorProcurado <= arr[fim]
41    ) {
42      if (inicio === fim) {
43        return arr[inicio] === valorProcurado ? inicio : -1;
44      }
45
46      const pos = inicio + Math.floor(
47        ((valorProcurado - arr[inicio]) * (fim - inicio)) /
48        (arr[fim] - arr[inicio])
49      );
```

ALGORITMO DE BUSCA POR INTERPOLAÇÃO

```
50
51          if (arr[pos] === valorProcurado) {
52            return pos;
53          }
54
55          if (arr[pos] < valorProcurado) {
56            inicio = pos + 1;
57          } else {
58            fim = pos - 1;
59          }
60        }
61
62        return -1;
63      }
64    }
65
66    module.exports = Buscas;
```

EXEMPLO DE USO ALGORITMOS DE BUSCA

```
1   const Buscas = require('./Buscas.js');
2
3   const dados = [10, 20, 30, 40, 50, 60];
4
5   console.log("Sequencial:", Buscas.sequencial(dados, 40));   // 3
6   console.log("Binária:", Buscas.binaria(dados, 40));         // 3
7   console.log("Interpolação:", Buscas.interpolacao(dados, 40)); // 3
```

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