Atividade - Árvore Binária de Busca

Tecnólogo em Análise e Desenvolvimento de Sistemas
Estrutura de Dados II
Jonathas Jivago de Almeida Cruz

José Nilton Silva Lima

Introdução

A estrutura de dados denominada *Árvore Binária de Busca* (BST - *Binary Search Tree*) consiste em uma coleção hierárquica de nós onde cada nó possui no máximo dois filhos (esquerdo e direito), seguindo uma propriedade fundamental de ordenação. Trata-se de uma generalização das estruturas lineares que permite operações eficientes de busca, inserção e remoção quando balanceada.

As principais operações associadas à estrutura de dados *Árvore Binária de Busca* são as seguintes:

- `inserir(valor)`: insere um novo valor na árvore mantendo a propriedade de BST;
- `contem(valor)`: verifica se um valor está presente na árvore;
- `buscaEmLargura()`: retorna os elementos em ordem de nível (BFS);
- `preOrdem()`: retorna os elementos em pré-ordem (raiz-esquerda-direita);
- `emOrdem()`: retorna os elementos em ordem crescente (esquerda-raiz-direita);
- `posOrdem()`: retorna os elementos em pós-ordem (esquerda-direita-raiz);
- `altura()`: retorna a altura da árvore;
- `tamanho()`: retorna o número de elementos.

No contexto da linguagem TypeScript, a implementação de uma BST pode ser realizada utilizando classes e recursão. A seguir, apresenta-se uma implementação completa:

```
1  class No<T> {
2      public valor: T;
3      public esquerda: No<T> | null;
4      public direita: No<T> | null;
5
6      constructor(valor: T) {
```

```
        this.valor = valor;
        this.esquerda = null;
        this.direita = null;
    }
}

class ArvoreBinariaBusca<T> {
    private raiz: No<T> | null;
    private tamanho: number;

    constructor(private comparar: (a: T, b: T) => number) {
        this.raiz = null;
        this.tamanho = 0;
    }

    public inserir(valor: T): void {
        this.raiz = this.inserirNo(this.raiz, valor);
        this.tamanho++;
    }

    private inserirNo(no: No<T> | null, valor: T): No<T> {
        if (no === null) return new No(valor);

        const comparacao = this.comparar(valor, no.valor);
        if (comparacao < 0) no.esquerda = this.inserirNo(no.esquerda, valor);
        else if (comparacao > 0) no.direita = this.inserirNo(no.direita, valor);
        else this.tamanho--; // Valor duplicado

        return no;
    }

    public contem(valor: T): boolean {
        return this.buscarNo(this.raiz, valor) !== null;
    }

    private buscarNo(no: No<T> | null, valor: T): No<T> | null {
        if (no === null) return null;

        const comparacao = this.comparar(valor, no.valor);
        if (comparacao < 0) return this.buscarNo(no.esquerda, valor);
        if (comparacao > 0) return this.buscarNo(no.direita, valor);
        return no;
    }

    public buscaEmLargura(): T[] {
        const resultado: T[] = [];
        if (!this.raiz) return resultado;

        const fila: No<T>[] = [this.raiz];
        while (fila.length > 0) {
            const no = fila.shift()!;
```

```
58              resultado.push(no.valor);
59              if (no.esquerda) fila.push(no.esquerda);
60              if (no.direita) fila.push(no.direita);
61          }
62          return resultado;
63      }
64
65      public preOrdem(): T[] {
66          const resultado: T[] = [];
67          this.preOrdemNo(this.raiz, resultado);
68          return resultado;
69      }
70
71      private preOrdemNo(no: No<T> | null, resultado: T[]): void {
72          if (no) {
73              resultado.push(no.valor);
74              this.preOrdemNo(no.esquerda, resultado);
75              this.preOrdemNo(no.direita, resultado);
76          }
77      }
78
79      public emOrdem(): T[] {
80          const resultado: T[] = [];
81          this.emOrdemNo(this.raiz, resultado);
82          return resultado;
83      }
84
85      private emOrdemNo(no: No<T> | null, resultado: T[]): void {
86          if (no) {
87              this.emOrdemNo(no.esquerda, resultado);
88              resultado.push(no.valor);
89              this.emOrdemNo(no.direita, resultado);
90          }
91      }
92
93      public posOrdem(): T[] {
94          const resultado: T[] = [];
95          this.posOrdemNo(this.raiz, resultado);
96          return resultado;
97      }
98
99      private posOrdemNo(no: No<T> | null, resultado: T[]): void {
100         if (no) {
101             this.posOrdemNo(no.esquerda, resultado);
102             this.posOrdemNo(no.direita, resultado);
103             resultado.push(no.valor);
104         }
105     }
106
107     public altura(): number {
108         return this.alturaNo(this.raiz);
```

```
    }

    private alturaNo(no: No<T> | null): number {
        if (!no) return -1;
        return Math.max(this.alturaNo(no.esquerda), this.alturaNo(no.direita)) +
        ↪  1;
    }

    public tamanho(): number {
        return this.tamanho;
    }

    public estaVazia(): boolean {
        return this.tamanho === 0;
    }
}
```

Exemplo de Uso

```
// Função de comparação para números
function compararNumeros(a: number, b: number): number {
    return a - b;
}

const arvore = new ArvoreBinariaBusca<number>(compararNumeros);

// Inserção de elementos
arvore.inserir(10);
arvore.inserir(5);
arvore.inserir(15);
arvore.inserir(3);
arvore.inserir(7);

// Consultas
console.log("Contém 7?", arvore.contem(7));
console.log("Travessia em ordem:", arvore.emOrdem());
console.log("Altura:", arvore.altura());
console.log("Tamanho:", arvore.tamanho());
```