Atividade - Deque

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 *deque* (*double-ended queue*, ou fila de duas pontas) consiste em uma coleção linear de elementos que permite a inserção e a remoção em ambas as extremidades, isto é, tanto no início quanto no final da sequência. Trata-se de uma generalização das estruturas clássicas de *pilha* (LIFO — *last in, first out*) e *fila* (FIFO — *first in, first out*). Seu uso é particularmente relevante em cenários onde é necessário gerenciar elementos de forma dinâmica em ambas as direções. As principais operações associadas à estrutura de dados *deque* são as seguintes:

- `pushFront(elemento)`: insere um elemento no início;
- `pushBack(elemento)`: insere um elemento no final;
- `popFront()`: remove e retorna o elemento do início;
- `popBack()`: remove e retorna o elemento do final;
- `peekFront()`: consulta o primeiro elemento;
- `peekBack()`: consulta o último elemento;
- `isEmpty()`: verifica se o *deque* está vazio;
- `size()`: retorna o número de elementos.

O *deque* admite operações fundamentais como a adição de elementos na frente ou atrás da estrutura, a remoção dos mesmos em ambas as extremidades, e também consultas ao primeiro e ao último elemento. Tais operações, quando implementadas de maneira eficiente, garantem desempenho constante (tempo $O(1)$) na maioria das aplicações práticas.

No contexto da linguagem TypeScript, a implementação de um *deque* pode ser realizada utilizando objetos associativos e controle explícito de índices. A seguir, apresenta-se uma implementação no desenvolvimento de soluções algorítmicas:

```typescript
class Deque<T> {
  private items: { [index: number]: T } = {};
  private frontIndex: number = 0;
  private backIndex: number = 0;

  pushFront(element: T): void {
    this.frontIndex--;
    this.items[this.frontIndex] = element;
  }

  pushBack(element: T): void {
    this.items[this.backIndex] = element;
    this.backIndex++;
  }

  popFront(): T | undefined {
    if (this.isEmpty()) return undefined;

    const element = this.items[this.frontIndex];
    delete this.items[this.frontIndex];
    this.frontIndex++;
    return element;
  }

  popBack(): T | undefined {
    if (this.isEmpty()) return undefined;

    this.backIndex--;
    const element = this.items[this.backIndex];
    delete this.items[this.backIndex];
    return element;
  }

  peekFront(): T | undefined {
    return this.items[this.frontIndex];
  }

  peekBack(): T | undefined {
    return this.items[this.backIndex - 1];
  }

  isEmpty(): boolean {
    return this.size() === 0;
  }

  size(): number {
    return this.backIndex - this.frontIndex;
  }

  clear(): void {
    this.items = {};
```

```
52      this.frontIndex = 0;
53      this.backIndex = 0;
54    }
55
56    print(): void {
57      console.log(this.toArray());
58    }
59
60    toArray(): T[] {
61      const result: T[] = [];
62      for (let i = this.frontIndex; i < this.backIndex; i++) {
63        result.push(this.items[i]);
64      }
65      return result;
66    }
67  }
```