FIAP – Engenharia de Software
Challenge 2025 – 2º Semestre

Data: Setembro / 2025

Sprint 3 de Dynamic Programming

Equipe:

- 558488 Anthony Motobe
- 555342 Arthur Rodrigues
- 554743 Guilherme Abe
- 554779 Gustavo Paulino
- 558017 Victor Dias

Cliente: DASA

1. Introdução

Nas unidades de diagnóstico, o consumo diário de insumos (como reagentes e descartáveis) não foi registrado com precisão, o que nos dificultou o controle de estoque e a previsão da reposição.

Para resolvermos esse problema, desenvolvemos um sistema em Python que organiza os dados de consumo utilizando **estruturas de dados clássicas** e algoritmos que aprendemos na **programação dinâmica**, permitindo uma melhor visibilidade do estoque e reduzindo os desperdícios.

---

2. Objetivos do Sistema

- Simular o registro de consumo diário de insumos.
- Controlar estoques em diferentes almoxarifados.
- Identificar os itens em excesso ou em falta.
- Propor redistribuição automáticas entre estoques.
- Aplicar estruturas de dados (Fila e Pilha), buscas (Sequencial e Binária) e algoritmos de ordenação (Merge Sort e Quick Sort).
- Demonstrar o uso de memorização para otimização de cálculos repetidos.

---

3. Estruturas de Dados Utilizadas

3.1 Fila (Queue)

- Utilizada para registrar consumos em **ordem cronológica**.
- Permite visualizar o histórico desde o consumo mais antigo até o mais recente.

3.2 Pilha (Stack)

- Utilizada para registrar consumos em **ordem inversa**.
- Permite consultas rápidas ao consumo mais recente (último que entrou, primeiro a sair).

---

4. Estruturas de Busca

4.1 Busca Sequencial

- Percorre a lista de insumos elemento por elemento.
- Simples, mas menos eficiente para grandes volumes.

4.2 Busca Binária

- Exige lista ordenada previamente.
- Divide o espaço de busca ao meio a cada iteração.
- Muito mais eficiente em grandes listas (complexidade O(log n)).

---

5. Algoritmos de Ordenação

5.1 Merge Sort

- Ordenação estável baseada em divisão recursiva.
- Complexidade O(n log n).
- Útil quando precisamos garantir consistência em grandes volumes de dados.

5.2 Quick Sort

- Ordenação eficiente em média, escolhendo um **pivô** e particionando a lista.
- Complexidade média O(n log n).
- Mais rápido que o Merge Sort em listas pequenas, porém menos estável.

---

6. Programação Dinâmica (Memoização)

- Implementamos uma função CALCULATE_DIFF que utiliza **memorização** para evitar cálculos repetidos da diferença entre a quantidade atual e o ideal de cada insumo.
- Isso reduz o custo de operações repetitivas, otimizando o desempenho do sistema.

7. Funcionalidades do Sistema

- **Visualização de Estoques**: mostra insumos com quantidade atual, ideal e validade.
- **Registro de Consumo**: adiciona consumos na Fila e na Pilha.
- **Histórico de Consumo**: consulta dados em ordem cronológica e inversa.
- **Busca de Insumos**: via busca sequencial ou binária.
- **Ordenação de Insumos**: com Merge Sort e Quick Sort.
- **Redistribuição Automática**: sugere transferências entre almoxarifados para corrigir excessos/faltas.

8. Conclusão

O sistema desenvolvido cumpre os requisitos, aplicando estruturas de dados e algoritmos clássicos em um problema realista de controle de estoque.

Nossa solução:

- Melhora a visibilidade do consumo.
- Facilita o controle de insumos críticos.
- Reduz o risco de desperdícios ou faltas.
- Demonstra o uso prático de conceitos de Fila, Pilha, Busca, Ordenação e Programação Dinâmica.

---

9. Repositório

O código está disponível no GitHub da equipe, acompanhado deste relatório.