Desafio 1: SPRINT-3
DYNAMIC PROGRAMMING

Integrantes:

-Lucca Borges RM554608

-Ruan Vieira RM557599

-Rodrigo Carnevale RM558148

Sumário

1. Contexto do problema

Unidades de diagnóstico enfrentam baixa visibilidade no consumo de insumos (reagentes e descartáveis). A solução proposta organiza o registro de consumo, permite consultas eficientes e gera alertas para suporte à reposição.

2. Visão geral da solução (arquitetura em memória)

O sistema usa um 'mini banco' em memória (classe MiniDB) e dois modelos:

• Item: dados do insumo (id, nome, categoria, unidade, lote, validade, estoque_min, estoque_atual).

• Consumo: evento (timestamp, item_id, quantidade).

Na MiniDB, cada registro de consumo é gravado em duas estruturas: fila (deque) e pilha (list).

3. Estruturas de Dados e seu papel no problema

3.1 Fila (deque — FIFO)

• Por quê: preservar ordem cronológica dos eventos para auditoria e relatórios.

• Onde: MiniDB.fila_consumo recebe cada Consumo com append().

• Complexidade: inserção/remoção nas extremidades em O(1) amortizado.

3.2 Pilha (list — LIFO)

• Por quê: acessar rapidamente os eventos mais recentes (consultas operacionais).

• Onde: MiniDB.pilha_consumo recebe cada Consumo com append(); o topo é o último evento.

• Complexidade: push/pop no fim em O(1) amortizado.

4. Algoritmos de Busca e como foram aplicados

4.1 Busca Sequencial (linear)

• Uso: localizar um item por nome quando a lista não está ordenada por esse critério.

• Exemplo no fluxo: procurar 'Seringa 5ml' em itens não ordenados.

• Complexidade: O(n).

4.2 Busca Binária

• Pré-condição: a coleção deve estar ordenada pela mesma chave da busca.

• Aplicações no projeto:

– Por validade: ordenar itens por validade e localizar uma data específica.

– Por nome (interativa): ao final da execução, o script ordena por nome e busca o termo digitado.

• Complexidade: O(log n).

5. Algoritmos de Ordenação e seu uso prático

5.1 Merge Sort

• Uso: ordenar pares (Item, total_consumido) por quantidade para produzir rankings.

• Propriedade: estável (mantém ordem relativa de chaves iguais).

• Complexidade: O(n log n).

5.2 Quick Sort

• Uso: ordenar itens por validade para exibir próximos a vencer.

• Propriedade: desempenho médio muito bom; pior caso $O(n^2)$ — aceitável aqui pelo porte dos dados.

• Complexidade média: O(n log n).

6. Relatórios e Regras de Negócio

• Totais por item: varredura na fila de consumo, somando quantidades (O(m), m = eventos).

• Próximos a vencer: ordenação por validade e exibição dos primeiros.

• Alerta de estoque baixo: lista de itens com estoque_atual ≤ estoque_min × (1 + margem). No exemplo, margem = 0,20 (120% do mínimo).

7. Geração de dados (simulação reprodutível)

• Simulador cria eventos diários (dias × eventos_por_dia), com seed fixa para reprodutibilidade.

• Heurística: reagentes consomem menos por evento; descartáveis, mais.

8. Fluxo de execução (demo)

1) Monta banco de itens.

2) Simula 7 dias de consumo.

3) Mostra primeiros/últimos eventos (fila/pilha).

4) Busca sequencial por nome.

5) Ordena por validade e faz busca binária pela data.

6) Ordena por total consumido e por validade (rankings).

7) Gera alertas de estoque.

8) Pergunta um nome e faz busca binária interativa.

9. Complexidade — Resumo

• Registrar consumo: O(1) por evento (append em fila/pilha) + atualização de estoque.

• Busca sequencial (Item por nome): O(n).

• Busca binária (por validade/nome): O(log n) após ordenação.

• Merge Sort / Quick Sort: O(n log n) (Quick Sort: pior caso $O(n^2)$).

• Totais por item: O(m), m = número de eventos.

10. Limitações e Extensões Futuras

• Persistência real (SQLite) para histórico longo.

• API REST para registrar consumo em tempo real.

• Painel de acompanhamento e previsão de ruptura (dashboards/ML).

• Políticas automáticas de reposição baseadas em estoque mínimo e lead time.

Tabela de Complexidade (Big-O)

Etapa	Estratégia	Complexidade
Registro de consumo	Fila/Pilha (append)	O(1)
Busca por nome	Sequencial	O(n)
Busca por validade/nome	Binária (lista ordenada)	O(log n)
Ordenação por consumo	Merge Sort	O(n log n)
Ordenação por validade	Quick Sort	O(n log n) / $O(n^2)$
Totais por item	1 varredura de eventos	O(m)