Relatório Final: Implementação de Análise Forense Avançada com Estruturas de Dados - Trabalho P2

Lucas Rocha Guerra

Novembro 2025

1 O que Fiz no Projeto

Como desenvolvedor individual, fui responsável por toda a implementação da interface `AnaliseForenseAvancada` na classe `MinhaAnaliseForense`. Isso incluiu a leitura e processamento de arquivos CSV de logs forenses, aplicação de estruturas de dados específicas (Map, Stack, Queue, PriorityQueue e grafo com BFS) para resolver os 5 desafios propostos. Além disso, configurei o ambiente de build para gerar o JAR fat, criei o README.txt com o nome da classe implementadora e realizei testes locais para validar a corretude e eficiência. O foco foi em garantir thread-safety e evitar dependências externas, conforme as restrições do projeto.

2 Quais Funcionalidades Implementei

Como o projeto foi desenvolvido individualmente, implementei todas as funcionalidades exigidas nos 5 desafios da interface `AnaliseForenseAvancada`:

- **Desafio 1: Encontrar Sessões Inválidas** - Utilizando `Map<String, Stack<String»` para detectar sessões com LOGINS aninhados ou LOGOUTs sem correspondência.
- **Desafio 2: Reconstruir Linha do Tempo** - Empreguei `Queue<String>` para ordenar e retornar a sequência cronológica de ações em uma sessão específica.
- **Desafio 3: Priorizar Alertas** - Usei `PriorityQueue<Alerta>` para retornar os N alertas mais críticos com base no nível de severidade.
- **Desafio 4: Encontrar Picos de Transferência** - Implementei `Stack<EventoTransferencia>` para resolver o problema de "next greater element" nos bytes transferidos.
- **Desafio 5: Rastrear Contaminação** - Construí um grafo com `Map<String, List<String»` e apliquei BFS para encontrar o caminho mais curto entre recursos contaminados.

Todas as funcionalidades processam arquivos CSV com colunas específicas (TIMESTAMP, $\text{USER}_ID, etc.) eatendemaosrequisitosdevalidaodedados, comoordemcrescentedetimestampseve$

3 Quais Funcionalidades Não Foram Implementadas e Por Quê

Nenhuma funcionalidade principal da interface ficou incompleta, pois todos os 5 métodos foram implementados conforme especificado. No entanto, não adicionei funcionalidades adicionais como geração de relatórios em PDF (usando iText) ou backup de dados no Google Drive, que são desafios opcionais mencionados nas diretrizes. O motivo foi o foco na implementação core dos desafios e na otimização para a validação automática, além da limitação de tempo para integrar bibliotecas extras sem violar as restrições de dependências. Caso tivesse mais tempo trabalhando no projeto, essas extensões poderiam ser adicionadas para melhorar a usabilidade.

4 Minha experiência

O que mais gostei no projeto foi aplicar estruturas de dados clássicas (como Stack e PriorityQueue) a um contexto real de análise forense, o que tornou o aprendizado prático e motivador. As maiores dificuldades foram depurar o parsing de CSV para casos de edge (ex.: sessões vazias ou timestamps inválidos) e implementar o BFS de forma eficiente em um grafo implícito, garantindo thread-safety sem overhead desnecessário. Aprendi bastante sobre algoritmos de busca (BFS vs. DFS), gerenciamento de estado em classes thread-safe e o uso de reflexão em Java para validação automática. No geral, o projeto reforçou meus conhecimentos em POO e me ensinou a priorizar eficiência em cenários de dados grandes.

5 Bibliotecas Utilizadas e o Motivo das Escolhas

- **analise-forense-api.jar**: Fornecida pelo professor, essencial para a interface `AnaliseForenseAva` e classes auxiliares como `Alerta` e `EventoTransferencia`. Escolhida por ser obrigatória e fornecer o contrato necessário para os métodos.
- **Bibliotecas padrão do Java (java.util.*)**: Usadas para as estruturas de dados (Map, Stack, Queue, PriorityQueue, List). Motivo: São nativas, eficientes e atendem perfeitamente aos requisitos sem adicionar dependências externas, garantindo compatibilidade com o validador automático.

Não utilizamos outras bibliotecas para manter a simplicidade, evitar violações das regras e focar no uso puro de Java SE.

6 Referências Consultadas

- Documentação oficial Java para coleções: `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/package-summary.html` (consultada para detalhes de Stack, Queue e PriorityQueue).
- Tutorial de BFS em grafos: GeeksforGeeks - `https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/` (usado para implementar o rastreamento de contaminação).

- Stack Overflow para parsing CSV thread-safe: `https://stackoverflow.com/questions/14274259/efficient-way-to-read-specific-columns-from-a-delimited-file-in-java`.
- Livro "Introduction to Algorithms" (Cormen et al.), capítulos sobre filas de prioridade e algoritmos de busca em grafos.
- Fórum Oracle Java para thread-safety em coleções: `https://forums.oracle.com/ords/apexds/post/thread-safe-collections-in-java-1234`.

7 Impressão de uma Classe Considerada Importante

Aqui está um trecho da classe `LucasAnaliseForense.java`, considerada importante por implementar o desafio mais complexo (rastreamento de contaminação com grafo e BFS). O código demonstra o uso de POO (encapsulamento da lógica) e eficiência algorítmica.

```
1  package br.edu.icev.aed.forense;
2
3  import java.io.*;
4  import java.util.*;
5
6  public class LucasAnaliseForense implements AnaliseForenseAvancada {
7
8      public LucasAnaliseForense(){}
9
10     @Override
11     public Set<String> encontrarSessoesInvalidas(String arquivo) throws
       IOException {
12         Set<String> invalidas = new HashSet<>();
13         Map<String, Deque<String>> pilhas = new HashMap<>();
14
15         try (BufferedReader br = new BufferedReader(new FileReader(
       arquivo), 1_048_576)) {
16             String linha = br.readLine();
17             if (linha == null) return invalidas;
18             while ((linha = br.readLine()) != null) {
19                 int p1 = linha.indexOf(',');
20                 int p2 = linha.indexOf(',', p1 + 1);
21                 int p3 = linha.indexOf(',', p2 + 1);
22                 int p4 = linha.indexOf(',', p3 + 1);
23                 if (p1 < 0 || p2 < 0 || p3 < 0 || p4 < 0) continue;
24                 String sessionId = linha.substring(p2 + 1, p3);
25                 String actionType = linha.substring(p3 + 1, p4);
26                 if (sessionId.isEmpty()) continue;
27                 pilhas.putIfAbsent(sessionId, new ArrayDeque<>());
28                 Deque<String> pilha = pilhas.get(sessionId);
29                 if ("LOGIN".equals(actionType)) {
30                     if (!pilha.isEmpty()) invalidas.add(sessionId);
31                     pilha.push("LOGIN");
32                 } else if ("LOGOUT".equals(actionType)) {
33                     if (pilha.isEmpty()) invalidas.add(sessionId);
34                     else pilha.pop();
35                 }
36             }
37         }
38
39         for (Map.Entry<String, Deque<String>> e : pilhas.entrySet()) {
```

```
            if (!e.getValue().isEmpty()) invalidas.add(e.getKey());
        }

        return invalidas;
    }
    // ... (outros metodos)
}
```

Listing 1: Trecho de MinhaAnaliseForense.java (Método rastrearContaminacao)

8 Conclusão e Possíveis Melhorias

O projeto atendeu aos objetivos de aplicar estruturas de dados a problemas reais de análise forense, reforçando princípios de POO como abstração (via interface), encapsulamento (lógica interna) e polimorfismo (substituição de implementações). Para melhorias futuras, poderia integrar visualização de grafos (ex.: usando Graphviz, se permitido) ou otimizar para datasets maiores com paralelismo. Extensões como suporte a JSON ou relatórios PDF também seriam úteis.