1. Desafio

Sistemas críticos da nossa infraestrutura foram comprometidos. A única pista é um arquivo de log detalhado, forensic_logs.csv, que registra cada ação realizada. Sua missão, como especialista em perícia digital, é usar seu conhecimento em estruturas de dados para dissecar esses logs, identificar padrões anômalos e reconstruir a cadeia de ataque. Cada desafio o levará um passo mais perto de entender o que aconteceu.

2. Requisitos de Entrega

Para que sua solução seja avaliada corretamente, siga rigorosamente os seguintes requisitos de entrega:

a. **Formato do Arquivo:** Você deve entregar um **único arquivo .jar**. Este JAR deve ser um **"fat JAR"** (também conhecido como "uber JAR" ou "JAR with dependencies"), o que significa que ele deve conter não apenas o seu código compilado, mas também todas as bibliotecas de terceiros que você possa ter utilizado.
b. **Nome do Arquivo:** O nome do arquivo deve seguir o padrão matricula_dos_integrantes.jar. (por exemplo: 45789_454889_2025487.jar)
c. **Informação Adicional:** Junto com o .jar, você deve entregar um arquivo README.txt contendo o **APENAS nome completo da classe** que implementa a interface (ex: br.edu.icev.aed.SolucaoForense). Esta informação é **essencial** para que a ferramenta de avaliação automática funcione.

3. O Contrato: A Interface AnaliseForenseAvancada.java

Sua solução deve ser uma classe Java que implementa rigorosamente esta interface. O sucesso na avaliação automática depende da sua total aderência a este contrato.

```java
import java.io.IOException;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.Set;

/**
 * Contrato para a solução dos desafios avançados de análise forense.
 * Sua classe deve implementar esta interface e possuir um construtor público sem argumentos.
 */
public interface AnaliseForenseAvancada {

    /**
     * Desafio 1 (Pilha): Encontra sessões de usuário que foram corrompidas ou
     * deixadas abertas, indicando uma possível falha ou ataque. Uma sessão é
     * inválida se um usuário tenta um novo LOGIN antes de um LOGOUT, ou se a sessão
     * termina sem um LOGOUT correspondente.
     * @param caminhoArquivoCsv O caminho para o arquivo de logs.
     * @return Um Set contendo os IDs de todas as sessões (SESSION_ID) inválidas.
     * @throws IOException Se ocorrer um erro de leitura do arquivo.
     */
    Set<String> desafio1_encontrarSessoesInvalidas(String caminhoArquivoCsv) throws IOException;

    /**
     * Desafio 2 (Fila): Reconstrói a sequência exata de ações de um usuário dentro
     * de uma sessão específica, da primeira à última ação.
     * @param caminhoArquivoCsv O caminho para o arquivo de logs.
     * @param sessionId O ID da sessão a ser reconstruída.
     * @return Uma List<String> contendo os ACTION_TYPE na ordem cronológica em que
     * ocorreram dentro da sessão. Retorna uma lista vazia se a sessão não for encontrada.
     * @throws IOException Se ocorrer um erro de leitura do arquivo.
     */
    List<String> desafio2_reconstruirLinhaDoTempo(String caminhoArquivoCsv,
        String sessionId) throws IOException;


    /**
     * Desafio 3 (Fila de Prioridade): Identifica os N eventos de maior risco
     * para que a equipe de resposta a incidentes possa priorizá-los. O risco é
     * determinado pelo campo SEVERITY_LEVEL.
     * @param caminhoArquivoCsv O caminho para o arquivo de logs.
     * @param n O número de eventos de maior risco a serem retornados.
     * @return Uma List<Alerta> contendo os 'n' alertas mais severos, ordenados do
     * mais severo para o menos severo. Em caso de empate na severidade, a ordem
     * não importa.
     * @throws IOException Se ocorrer um erro de leitura do arquivo.
     */
    List<Alerta> desafio3_priorizarAlertas(String caminhoArquivoCsv, int n) throws IOException;
```

```java
/**
 * Desafio 4 (Pilha Monotônica): Detecta anomalias em transferências de dados.
 * Para cada evento de transferência, encontra o próximo evento no tempo que
 * envolveu uma transferência de dados MAIOR. Isso ajuda a identificar
 * escalonamentos súbitos na exfiltração de dados.
 * @param caminhoArquivoCsv O caminho para o arquivo de logs.
 * @return Um Map<Long, Long> onde a chave é o TIMESTAMP de um evento e o valor
 * é o TIMESTAMP do próximo evento com BYTES_TRANSFERRED maior. Se não houver
 * um evento maior subsequente, a chave não deve estar no mapa.
 * @throws IOException Se ocorrer um erro de leitura do arquivo.
 */
Map<Long, Long> desafio4_encontrarPicosDeTransferencia(String caminhoArquivoCsv) throws IOException;

/**
 * Desafio 5 (Grafo): Mapeia o caminho de contaminação do invasor através do
 * sistema, mostrando como ele se moveu de um recurso para outro. O caminho é
 * a sequência mais curta de recursos acessados entre o ponto de entrada e o alvo final.
 * @param caminhoArquivoCsv O caminho para o arquivo de logs.
 * @param recursoInicial O ponto de entrada do ataque (e.g., "/usr/bin/sshd").
 * @param recursoAlvo O alvo final do ataque (e.g., "/var/secrets/key.dat").
 * @return Um Optional<List<String>> contendo a sequência de recursos que formam
 * o caminho mais curto. Retorna Optional.empty() se não houver caminho.
 * @throws IOException Se ocorrer um erro de leitura do arquivo.
 */
Optional<List<String>> desafio5_rastrearContaminacao(String caminhoArquivoCsv,
    String recursoInicial, String recursoAlvo) throws IOException;
```

a. Classe de Apoio

```java
public class Alerta {
    public final long timestamp = 0;
    public final String acao = "";
    public final int nivelSeveridade = 0;
}
```

4. Configuração do Projeto

Para garantir que todas as soluções sigam exatamente o mesmo contrato, fornecemos a interface AnaliseForenseAvancada e a classe Alerta compiladas dentro de um único arquivo chamado **analise-forense-api.jar**. Este arquivo funciona como uma biblioteca ou dependência que você deve adicionar ao seu projeto Java.

Por que usar um .jar?

- **Padrão de Mercado:** No desenvolvimento de software profissional, é prática comum trabalhar com bibliotecas de terceiros (.jar) que fornecem as "ferramentas" e os "contratos" para construir uma aplicação.
- **Integridade do Contrato:** Garante que a estrutura da interface e das classes de apoio não seja alterada, permitindo que nossa ferramenta de avaliação automática funcione corretamente com a solução de todos os grupos.

Como Adicionar o .jar ao seu Projeto:

Você precisa adicionar o arquivo analise-forense-api.jar ao *classpath* (caminho de compilação) do seu projeto. O processo varia ligeiramente dependendo da sua IDE (Ambiente de Desenvolvimento Integrado):

- **IntelliJ IDEA:**
  1. Crie uma pasta chamada lib (ou libs) na raiz do seu projeto e copie o arquivo .jar para dentro dela.
  2. Clique com o botão direito sobre o arquivo .jar na árvore de projetos.
  3. Selecione a opção **"Add as Library..."**.
  4. Confirme a caixa de diálogo. Agora você poderá importar as classes da API no seu código.
- **Eclipse IDE:**
  1. Crie uma pasta chamada lib (ou libs) na raiz do seu projeto e copie o arquivo .jar para dentro dela.
  2. Clique com o botão direito sobre o seu projeto no "Package Explorer".
  3. Vá em **"Build Path" > "Configure Build Path..."**.
  4. Na aba **"Libraries"**, clique em **"Add JARs..."** ou **"Add External JARs..."** e navegue até o local do seu arquivo analise-forense-api.jar.
  5. Clique em "Apply and Close".
- **Visual Studio Code (com Java Extension Pack):**
  1. No painel "Java Projects", encontre a seção "Referenced Libraries".
  2. Clique no ícone de + ao lado de "Referenced Libraries".
  3. Navegue até o local do arquivo analise-forense-api.jar e o selecione.

Após configurar a dependência, você poderá usar o import no seu código para acessar a interface e a classe Alerta (ex: import br.edu.icev.aed.forense.AnaliseForenseAvancada;).

5. Descrição Detalhada dos Desafios

A seguir, cada desafio é apresentado com seu contexto, lógica sugerida e os requisitos exatos de retorno.

Desafio 1: Encontrar Sessões Inválidas

- **Contexto:** Invasores frequentemente deixam sessões abertas ou causam falhas que impedem o LOGOUT.
- **Lógica:** Use um estrutura onde a chave é o USER_ID. Ao ler os logs:
  1. Para uma ação LOGIN de um usuário, verifique sua pilha. Se não estiver vazia, a sessão atual (SESSION_ID) é inválida (LOGIN aninhado). Adicione-a ao conjunto de resultados. De qualquer forma, empilhe a SESSION_ID.
  2. Para uma ação LOGOUT, verifique a pilha. Se estiver vazia ou o topo não corresponder à SESSION_ID atual, a sessão é inválida. Se corresponder, desempilhe.

3. Após ler todos os logs, qualquer USER_ID em que a pilha não esteja vazia tem sessões inválidas. Adicione todas as SESSION_IDs restantes nas pilhas ao conjunto de resultados.

- **Tipo de Retorno:** Set<String>
- **Conteúdo:** O conjunto deve conter as SESSION_ID (Strings), exatamente como aparecem no arquivo de log, para todas as sessões consideradas inválidas.
- **Casos Específicos:**
  - Se **nenhuma sessão inválida** for encontrada, o método deve retornar um Set vazio (Collections.emptySet() ou new HashSet<>()).
  - **NUNCA** retorne null.
  - Exemplo de Saída Programática

```
// Supondo que as sessões 'session-gamma-915' e 'session-alpha-723' sejam inváli
Set<String> resultado = new HashSet<>();
resultado.add("session-gamma-915");
resultado.add("session-alpha-723");
return resultado;
```

Desafio 2: Reconstruir Linha do Tempo

- **Contexto:** É preciso saber exatamente o que um invasor fez, na ordem correta, dentro de uma sessão suspeita.
- **Lógica:** A estrutura FIFO da Fila é perfeita para isso.
  1. Primeiro, filtre o arquivo para obter todos os eventos da sessionId especificada.
  2. Adicione o ACTION_TYPE de cada um desses eventos a uma Queue.
  3. Construa a lista de resultados desenfileirando os itens um por um até que a fila esteja vazia. A ordem será naturalmente a correta.
- **Tipo de Retorno:** List<String>
- **Conteúdo:** A lista deve conter as ACTION_TYPE (Strings) de uma sessionId específica.
- **Propriedades Chave:**
  - **A Ordem é Crucial:** A lista **deve** estar em ordem cronológica estrita, refletindo a sequência em que as ações apareceram no arquivo de log (que já está ordenado por TIMESTAMP). O primeiro elemento da lista deve ser a primeira ação da sessão, e o último elemento, a última ação.

- **Casos Específicos:**
  - Se a sessionId fornecida **não existir** no arquivo de log, o método deve retornar uma List vazia (Collections.emptyList() ou new ArrayList<>()).
  - **NUNCA** retorne null.

Exemplo de Saída Programática:

```
// Para uma sessão com as ações LOGIN, FILE_ACCESS e LOGOUT nessa ordem.
List<String> resultado = new ArrayList<>();
resultado.add("LOGIN");
resultado.add("FILE_ACCESS");
resultado.add("LOGOUT");
return resultado;
```

Desafio 3: Priorizar Alertas

- **Contexto:** Com milhares de logs, é impossível analisar tudo. A equipe de resposta a incidentes precisa focar nos eventos mais críticos primeiro.
- **Lógica:**
    1. Crie uma PriorityQueue<Alerta>. Você precisará fornecer um Comparator que organize os Alertas pela SEVERITY_LEVEL em ordem decrescente (o maior número tem a maior prioridade).
    2. Leia cada linha do log, crie um objeto Alerta e adicione-o à PriorityQueue.
    3. Após processar todos os logs, extraia os n primeiros itens da fila usando poll() n vezes e adicione-os à lista de resultados.
- **Tipo de Retorno:** List<Alerta>
- **Conteúdo:** Uma lista de n objetos da classe Alerta, contendo os eventos de maior severidade.
- **Propriedades Chave:**
    - **Ordenação por Severidade:** A lista **deve** ser ordenada em ordem decrescente de SEVERITY_LEVEL. O primeiro elemento deve ser o alerta com o maior nível de severidade.
    - **Critério de Desempate:** Se dois ou mais alertas tiverem o mesmo SEVERITY_LEVEL, a ordem relativa entre eles na lista de retorno **não importa**.
- **Casos Específicos:**
    - Se o número de eventos no log for **menor que n**, a lista deve conter todos os eventos do log, ordenados por severidade.
    - Se n for 0, ou se o arquivo de log estiver vazio, retorne uma List vazia.
    - **NUNCA** retorne null.

Exemplo de Saída Programática:

```java
// Supondo n=2 e os alertas mais graves são (severidade 10) e (severidade 9)
Alerta alerta1 = new Alerta(1665857900L, "COMMAND_EXEC", 10);
Alerta alerta2 = new Alerta(1665857850L, "FILE_DELETE", 9);

List<Alerta> resultado = new ArrayList<>();
resultado.add(alerta1);
resultado.add(alerta2);
return resultado;
```

Desafio 4: Encontrar Picos de Transferência

- **Contexto:** A exfiltração de dados geralmente ocorre com transferências que aumentam drasticamente de tamanho. Este método detecta esses saltos.
- **Lógica:** Este é um problema clássico de "Próximo Elemento Maior" (Next Greater Element).
    1. Crie uma Stack para armazenar eventos (ou apenas seus timestamps e bytes). A pilha será mantida em ordem decrescente de BYTES_TRANSFERRED.
    2. Itere pelos logs **em ordem cronológica inversa** (do fim para o começo).
    3. Para cada evento e:
        - Enquanto a pilha não estiver vazia e os BYTES_TRANSFERRED do evento no topo da pilha forem menores ou iguais aos do evento e, desempilhe.
        - Se a pilha não estiver vazia após isso, o evento no topo é o "próximo elemento maior". Adicione a entrada (e.timestamp, topo.timestamp) ao seu mapa de resultados.
        - Empilhe o evento e.

- **Tipo de Retorno:** Map<Long, Long>
- **Conteúdo:** Um mapa onde cada chave é o TIMESTAMP de um evento de transferência de dados, e seu valor correspondente é o TIMESTAMP do **primeiro evento subsequente** com um valor de BYTES_TRANSFERRED estritamente maior.
- **Propriedades Chave:**
    - **Condição de Inclusão:** Uma chave timestamp_A só deve existir no mapa se houver um timestamp_B (com timestamp_B > timestamp_A) tal que bytes_B > bytes_A. O valor associado a timestamp_A será o menor timestamp_B que satisfaça essa condição.

- Se para um dado evento não existir nenhum evento subsequente com mais bytes transferidos, sua chave **NÃO DEVE** ser incluída no mapa.

- **Casos Específicos:**
  - Se o arquivo de log estiver vazio ou não contiver eventos com BYTES_TRANSFERRED > 0, retorne um Map vazio.
  - **NUNCA** retorne null.

Exemplo de Saída Programática:

```
// Evento A (t=100, b=50), Evento B (t=120, b=30), Evento C (t=150, b=80)
// O próximo maior que A é C. B não tem próximo maior.
Map<Long, Long> resultado = new HashMap<>();
resultado.put(100L, 150L);
return resultado;
```

Desafio 5: Rastrear Contaminação

- **Contexto:** O invasor não ataca o alvo final diretamente. Ele se move lateralmente pelo sistema, de um recurso para outro. Precisamos mapear esse caminho.
- **Lógica:** Use a Busca em Largura (BFS), que é ideal para encontrar o caminho mais curto em grafos não ponderados.
  1. **Construa o Grafo:** Os nós (vertices) são os TARGET_RESOURCE (Strings). As arestas são as transições. Crie um Map<String, List<String>> para sua lista de adjacências. Itere pelos logs, agrupando por SESSION_ID. Dentro de cada sessão, um acesso ao recursoA seguido por um acesso ao recursoB cria uma aresta direcionada recursoA -> recursoB.
  2. **Execute o BFS:**
     - Inicie uma fila para o BFS e adicione o recursoInicial.
     - Use um Map<String, String> chamado predecessor para rastrear o caminho (predecessor.put(B, A) significa que você chegou em B vindo de A).
     - Execute o algoritmo BFS padrão. Quando você encontrar o recursoAlvo, pare.
     - Se encontrou, use o mapa predecessor para reconstruir o caminho, começando do recursoAlvo e voltando até o recursoInicial. Inverta esta lista para obter o caminho correto. Encapsule-a em um Optional.
     - Se o BFS terminar e o alvo não for encontrado, retorne Optional.empty().
- **Tipo de Retorno:** Optional<List<String>>

- **Conteúdo:** Um Optional que, se presente, contém uma lista de TARGET_RESOURCE (Strings) representando o caminho mais curto entre o recursoInicial e o recursoAlvo.
- **Propriedades Chave:**
  - **Uso do Optional:**
    - Se um caminho for encontrado, retorne Optional.of(caminho).
    - Se **não houver caminho** entre o recurso inicial e o alvo, retorne Optional.empty().
  - **Estrutura da Lista (se presente):**
    - A lista **deve** estar ordenada.
    - O primeiro elemento **deve** ser o recursoInicial.
    - O último elemento **deve** ser o recursoAlvo.

- **Casos Específicos:**
  - Se recursoInicial for igual a recursoAlvo, e este recurso existir no log, o caminho é uma lista contendo apenas esse único elemento.
  - **NUNCA** retorne um Optional contendo null (Optional.of(null)).
  - **NUNCA** retorne null em vez de um Optional.

- **Exemplo de Saída Programática:**

```
// Caso 1: Caminho encontrado
List<String> caminho = new ArrayList<>();
caminho.add("/usr/bin/sshd");
caminho.add("/var/log/auth.log");
caminho.add("/var/secrets/key.dat");
return Optional.of(caminho);

// Caso 2: Caminho não encontrado
return Optional.empty();
```

6. Metodologia de Avaliação

A avaliação foi desenhada para recompensar não apenas a **corretude**, mas também a **eficiência** e **qualidade** do código, sendo dividida em duas fases.

A definição do grupo vencedor é baseada em um critério duplo que recompensa tanto a excelência absoluta quanto o desempenho relativo. Para ser coroado vencedor, um grupo

precisa satisfazer duas condições simultaneamente: primeiro, deve alcançar a maior Pontuação Base entre todos os competidores, uma métrica que consolida a corretude, eficiência e qualidade do código. Segundo, e mais importante, essa pontuação deve ultrapassar o limiar de excelência de 95 pontos. Portanto, não basta ser o melhor; é preciso ser comprovadamente excelente. O grupo que cumprir ambos os requisitos garante a nota máxima de 100 pontos, estabelecendo-se como o padrão de performance contra o qual todas as outras equipes serão comparadas. Caso o grupo de maior pontuação não atinja esse limiar, a competição não terá um vencedor absoluto, e as notas serão calculadas com base no desempenho geral. Em termos gerais, a pontuação é dada como segue:

1. **Cálculo da Pontuação Base:** Cada solução é avaliada objetivamente contra um conjunto de critérios, gerando uma pontuação que pode ultrapassar 100 pontos.
2. **Determinação da Nota Final (0-100):** A Pontuação Base é usada para ranquear as equipes. O grupo com o melhor desempenho (acima de um limiar de excelência) crava a nota 100, e as demais notas são calculadas em relação a este padrão de excelência.

Fase 1: Cálculo da Pontuação Base (Máximo de 120 Pontos)

A Pontuação Base é a soma de três pilares: Corretude, Eficiência e Qualidade.

A. Corretude (Peso: 70 Pontos)

- Este é o pilar mais importante. Se a solução não produz o resultado correto, a performance é irrelevante.
- **Método:** A ferramenta de avaliação rodará uma bateria de testes unitários secretos contra cada um dos 5 desafios. Os testes incluirão casos de uso normais, casos extremos (edge cases) e arquivos de log massivos.
- **Cálculo:** Cada desafio vale **14 pontos**. A pontuação de um desafio é calculada como:

$$Pontos_{desafio} = 14 \times (N_c/T_t)$$

Onde N_c é a quantidade de testes que passaram e T_t é o total de testes para o desafio.

B. Eficiência de Execução (Peso: 40 Pontos)

- Aqui é onde a competição realmente acontece. Mediremos o tempo total que cada solução leva para processar um conjunto de arquivos de log de grande volume.
- **Método:** O avaliador medirá o tempo de execução de todos os desafios em sequência, para cada grupo. Para garantir justiça, cada solução rodará 3 vezes na mesma máquina, e a média dos tempos será considerada.
- **Cálculo:**
  - Identifica-se o menor tempo médio entre todos os grupos ($Tempo_{Mais\ Rápido}$).

- O grupo com o $Tempo_{Mais\ Rápido}$ recebe os **40 pontos** integrais de eficiência.
- Para os outros grupos, a pontuação é inversamente proporcional à sua lentidão em relação ao mais rápido:

  $Pontos_{eficiência} = 40x\ (Tempo_{Mais\ Rápido} / Tempo_{Seu\ Grupo})$

C. Qualidade e Boas Práticas (Peso: 10 Pontos)

- Um código que funciona rápido é bom. Um código que também é legível e bem estruturado é profissional.
- Esta etapa será uma avaliação qualitativa baseada em um checklist. Uma Inteligência Artificial será utilizada para calcular a legibilidade do código.
- **Critérios (2 pontos por item):**
  1. **Legibilidade:** O código é claro, bem formatado e utiliza nomes de variáveis/métodos intuitivos?
  2. **Estrutura:** O uso das estruturas de dados é apropriado para cada problema? (Ex: Não usou uma ArrayList onde uma LinkedList seria drasticamente mais eficiente para a lógica aplicada).
  3. **Comentários:** O código complexo é devidamente comentado, explicando o "porquê" daquela solução?
  4. **Tratamento de Exceções:** A IOException é tratada ou propagada corretamente, sem blocos catch vazios?
  5. **Modularidade:** O código evita repetição excessiva (princípio DRY - Don't Repeat Yourself)?

Fase 2: Cálculo da Nota Final (Escala de 0 a 100)

1. O Limiar de Excelência

Para que um grupo seja considerado "Vencedor", ele não precisa apenas ser o melhor, mas também ser excelente.

- Limiar: 95 Pontos Base.
- O Grupo Vencedor: É o grupo com a maior Pontuação Base, desde que essa pontuação seja igual ou superior a 95.

2. Regras de Atribuição da Nota Final

- Para o Grupo Vencedor:
  - Se um Grupo Vencedor for identificado (maior pontuação e ≥ 95), sua Nota Final é 100.
- Para os Demais Grupos (Cálculo Relativo):
  - A nota dos outros grupos é calculada em relação à performance do vencedor, para refletir o quão próximos eles chegaram da excelência.
  - A fórmula é:

$$\text{Nota Final}_{\text{Grupo}} = 99 \times (\text{Pontuação Base}_{\text{Grupo}} / \text{Pontuação Base}_{\text{Grupo Vencedor}})$$

- Usamos **99** como multiplicador para garantir que apenas o vencedor absoluto receba a nota máxima de 100.

Caso Nenhum Grupo Atinja o Limiar:

- Se o grupo com a maior pontuação **não atingir os 95 Pontos Base**, ninguém recebe 100.
- Neste cenário, a nota final de cada grupo será sua própria Pontuação Base, com um teto de 95.

$$\text{Nota Final}_{\text{Grupo}} = \min(95, \text{Pontuação Base}_{\text{Seu Grupo}})$$

Cláusula de Salvaguarda (Corretude Mínima):

- Para evitar que um código muito rápido, mas incorreto, obtenha uma boa nota, há uma regra final: se a pontuação de **Corretude (0-70) de um grupo for inferior a 40**, sua Nota Final será simplesmente essa pontuação de corretude.
- **Exemplo:** Um grupo obteve 35/70 em corretude. Mesmo que sua Pontuação Base total seja 60, sua Nota Final será 35.

Exemplo prático

Critério	**Grupo A (Vencedor)**	**Grupo B(Vice)**	**Grupo C (Lento)**	**Grupo D (Incorreto)**
Corretude (0-70)	70.0	70.0	70.0	35.0
Tempo de Execução	15s	18s	60s	16s
Eficiência (0-40)	40 *(40 * 15/15)*	33.3 *(40*(15/18)*	10.0 *(40 * 15/60)*	37.5 *(40 * 15/16)*
Qualidade (0-10)	9.0	8.0	7.0	6.0
Pontuação Base	119.0	111.3	87.0	78.5
Atingiu o Limiar (95)?	Sim	Sim	Não	Não
É o vencedor?	Sim	Não	Não	Não
Cálculo Final	Nota 100	99 * (111.3/119)	99*(87/119)	Cláusula de

				Salvaguarda
Nota Final	100	92,4	72,3	35,0
Nota na P2	4,00	3,69	2,89	1,4

7. Critérios para anulação da nota

Existem condições sob as quais um trabalho, independentemente do esforço aparente, será desconsiderado e receberá a **nota final zero**. Essas regras são inegociáveis e visam garantir a integridade acadêmica e o cumprimento dos requisitos mínimos do projeto. A nota será zerada nos seguintes casos:

- **Plágio ou Desonestidade Acadêmica:** A constatação de plágio, seja de código de outros grupos (deste ou de semestres anteriores) ou de fontes online não autorizadas, resultará na anulação da nota para **todos os grupos envolvidos**. O trabalho deve ser uma criação original da equipe.
- **Falha de Execução:** O arquivo .jar entregue que não puder ser executado pela ferramenta de avaliação — seja por erros de compilação, falha ao empacotar as dependências ("fat JAR" incorreto) ou qualquer outro erro que impeça a inicialização — receberá nota zero. Se o avaliador não consegue rodar, o trabalho não pode ser medido.
- **Não Cumprimento do Contrato:** Uma solução cujo código não implemente corretamente a interface AnaliseForenseAvancada será zerada. A incapacidade da ferramenta de carregar e instanciar a classe do aluno por não seguir o contrato é considerada uma falha fundamental.
- **Resultados Pré-Fixados ("Hardcoded"):** Qualquer solução que, em vez de processar os dados de forma algorítmica, contenha resultados "hardcoded" para contornar a lógica dos desafios será considerada fraude. Por exemplo, retornar uma lista de alertas pré-definida se o nome do arquivo de teste for teste_cenario_1.csv levará à anulação da nota. A solução deve ser genérica e funcional para qualquer arquivo de log que siga a estrutura definida.

8. Conjunto de Dados

- **Colunas:**
  1. TIMESTAMP (long - Unix epoch time, **garantido estar em ordem crescente**)
  2. USER_ID (String)
  3. SESSION_ID (String)
  4. ACTION_TYPE (String - e.g., LOGIN, LOGOUT, FILE_ACCESS, COMMAND_EXEC)
  5. TARGET_RESOURCE (String - e.g., um caminho de arquivo ou um IP)
  6. SEVERITY_LEVEL (int - de 1 a 10, sendo 10 o mais crítico)
  7. BYTES_TRANSFERRED (long)

Exemplo de Arquivo de Entrada (CSV)

TIMESTAMP,USER_ID,SESSION_ID,ACTION_TYPE,TARGET_RESOURCE,SEVERITY_LEVEL,BYTES_TRANSFERRED

1697396401,alice,session-alpha-723,LOGIN,/usr/bin/sshd,5,0

1697396415,alice,session-alpha-723,COMMAND_EXEC,/bin/ls,3,1024

1697396428,bob,session-beta-112,LOGIN,/usr/bin/sshd,5,0

1697396445,alice,session-alpha-723,FILE_ACCESS,/var/log/auth.log,7,4096

1697396458,eve,session-gamma-915,LOGIN,/usr/bin/sshd,5,0

1697396472,bob,session-beta-112,FILE_ACCESS,/home/bob/docs/project.txt,4,8192

1697396485,eve,session-gamma-915,LOGIN,/usr/bin/telnetd,9,0

1697396501,alice,session-alpha-723,DATA_TRANSFER,198.51.100.2,8,512000

1697396515,bob,session-beta-112,LOGOUT,/usr/bin/sshd,5,0

1697396528,carlos,session-delta-404,LOGIN,/usr/bin/sshd,5,0

1697396542,alice,session-alpha-723,FILE_ACCESS,/var/secrets/key.dat,10,256

1697396555,carlos,session-delta-404,COMMAND_EXEC,/usr/bin/whoami,3,512

1697396570,eve,session-gamma-915,FILE_ACCESS,/etc/shadow,10,2048

1697396588,carlos,session-delta-404,DATA_TRANSFER,203.0.113.50,8,1024000

1697396602,alice,session-alpha-723,LOGOUT,/usr/bin/sshd,5,0

1697396615,carlos,session-delta-404,COMMAND_EXEC,/bin/rm,9,0

1697396630,dave,session-epsilon-500,LOGIN,/usr/bin/sshd,5,0

1697396645,eve,session-gamma-915,DATA_TRANSFER,192.0.2.10,8,768000

1697396660,carlos,session-delta-404,LOGOUT,/usr/bin/sshd,5,0