UNIVERSIDADE ESTADUAL PAULISTA
Instituto de Ciência e Tecnologia – Sorocaba
Engenharia de Controle e Automação

Enrico de Castro Garcia
Érick Domingos da Silva
Leonardo Pereira de Lima

RELATÓRIO TÉCNICO: APLICAÇÃO DE PREVISÃO DE FRAUDES BANCÁRIAS

SOROCABA
JUNHO / 2025

Sumário

Introdução

A Aplicação de Previsão de Fraudes Bancárias, programada em linguagem C, utiliza-se de diversas estruturas de dados para mostrar ao usuário a probabilidade de uma certa transação bancária ser fraudulenta. O banco de dados usado neste trabalho pode ser achado no site Kaggle e foi gerado pelo usuário da plataforma Aryan Kumar. Este dataset contém dados sintéticos que simulam transações do mundo real, que podem ser fraudulentas ou não, e suas características.

Apesar de não ser possível afirmar a nível global, segundo a Febraban, no ano de 2023, foram realizadas aproximadamente 186 bilhões de transações bancárias só no Brasil, um aumento de 13,9% em relação a 2022. Destas 186 bilhões, 216.046 foram notificadas como fraudulentas, representando um aumento de quase 58% em comparação ao último ano. Este número representa 0,007% das transações totais e, apesar de parecer uma baixa porcentagem, as perdas por fraudes no sistema de pagamentos foi de R$2,911 bilhões.

Com esses números crescendo cada vez mais e com o uso de Big Data também cada vez maior, é necessário a utilização de estruturas de dados adequados, visando realizar operações como análise estatística e busca de informações.

O dataset apresenta um extenso número de dados (5 milhões). Apesar de serem gerados sinteticamente, o conjunto funciona pelas diversas variáveis que o compõem e pela ausência quase total de valores nulos.

A aplicação visa auxiliar na análise de transferências bancárias, que são feitas cada vez mais, muito por conta da facilidade de acesso e realização. Por esse motivo, é essencial trabalhar com um grande conjunto de dados a fim de analisar diferentes características de movimentação financeira para que possam ser observados padrões fraudulentos e assim seja possível detectar e alertar ao usuário quando este poderá estar sendo vítima de uma ação criminosa.

Atestando a plena funcionalidade da aplicação, o sistema poderá ser utilizado por instituições financeiras privadas e públicas para garantir maior segurança ao usuário.

Diante disso, o objetivo principal é o desenvolvimento de uma aplicação que utiliza de diferentes estruturas de dados visando a análise de um conjunto de dados de transações financeiras. Sendo necessário estreitar essa meta à objetivos mais

específicos como: aplicação prática de conceitos teóricos, implementação de cinco estruturas de dados diferentes, implementação de cálculos estatísticos, desenvolvimento de interface para visualização dos resultados, realizar testes de performance através de benchmarks para otimização da aplicação.

1. Apresentação do Dataset

1.1 Fonte e Estrutura dos Dados

O conjunto de dados utilizado é “Financial Transactions Dataset for Fraud Detection”, desenvolvido pelo usuário da plataforma Kaggle, Aryan Kumar. O dataset é composto por dados gerados sinteticamente a fim de simular características e comportamentos de transações financeiras do mundo real ao longo do ano de 2023.

O dataset possui 5 milhões de registros de transações, definido pelas seguintes categorias:

Identificação única: Cada uma das transações são identificadas por um ID, do tipo alfanumérico, visando individualizar cada movimentação financeira.

Dados Geográficos e Temporais: Cidade onde cada transação ocorreu e Timestamp no formato ISO, que indica a data e o horário exato.

Características das Transações: Montante da transação; Tipo de transação; Categoria do comércio envolvido; Dispositivo utilizado e identificação de fraude (Booleano).

1.2 Qualidade e Integridade dos Dados

Vazio de dados: 99,99994% de preenchimento de dados fundamentais (3 linhas da variável ‘timestamp’ possuem valores inválidos); 100% de preenchimento de dados opcionais nos campos escolhidos para análise.

Informações espaço-temporais: Aproximadamente 99,87% das transações ocorrem no ano de 2023 (0,13% ocorreram em 31/12/2022 e/ou 01/01/2024), 99,99996% das marcas de hora estão dentro do intervalo 00:00-23:59 (3 linhas possuem valores inválidos); Todas as transações analisadas têm origem em uma das sete seguintes cidades: Tóquio, Nova Iorque, cidade de Singapura, Berlim, Sydney, Toronto, Dubai e Londres.

Valores referenciais: IDs utilizados não possuem duplicação.

1.3 Distribuição Estatística dos Dados

Abaixo, há o cálculo de distribuição de dados para quando a transação é classificada como fraude.

Distribuição dos tipos de transação: Não há grande concentração em um dos quatro tipos, dados são bem distribuídos (Transferências: 45.328 fraudes,

Empréstimo: 44.874 fraudes, Depósito: 44.786 fraudes, Pagamento: 44.565 fraudes).

Distribuição de Dispositivos: Também há uma distribuição homogênea entre esses dados (Caixa eletrônico: 45.217 fraudes, Caixa de comércios: 44.853 fraudes, Computadores: 44.807 fraudes, Móveis: 44.677 fraudes).

Distribuição das finalidades: Distribuição homogênea dos dados (Entretenimento: 22.573 fraudes, Mercados: 22.516 fraudes, Viagens: 22.503 fraudes, Varejo: 22.453, Compras Online: 22.324 fraudes, Restaurantes: 22.367 fraudes, Utilidades: 22.261 fraudes, Outros: 22.556 fraudes).

Distribuição de quantia: Maior concentração no intervalo 0-100 (81.687 fraudes); menor concentração nos intervalos 500-1000 e 1000-5000 (22.456 e 22.574 fraudes, respectivamente).

1.4 Adequação para Testes de Estrutura de Dados

A coleção de 5.000.000 de registros oferece volume suficiente para realizar testes de performance, identificando gargalos computacionais; Análise de comportamento a partir dos dados fornecidos.

Possui diversidade de padrões de acesso por meio de consultas e buscas através de um ID único e filtragem partindo de diversos critérios, que variam de acordo com a estrutura utilizada.

Tendo também uma diversidade de dados como: campos numéricos (data e hora, quantia) - Campos categóricos (IDs das contas a realizar a transação, IDs das contas a receber, tipo de transação, categoria, localização, dispositivo utilizado); Campos Booleanos (Fraude ou não fraude); Identificadores únicos (ID alfanumérico de cada transação)

2. Estruturas de Dados Escolhidas

2.1 Estruturas Clássicas

Hash table: utilizado por ser uma estrutura eficiente de busca e de acesso rápido. Ideal para consultas por ID único, usada como chave para a busca das transações financeiras, e também foi empregada como estrutura para armazenamento das transações. A escolha se deve à sua capacidade de fornecer inserções, buscas e remoções muito rapidamente, com complexidade média constante de (O(1)). Isso a torna ideal para trabalhar com grandes volumes de dados, como neste dataset.

Árvore AVL: utilizada para armazenar e organizar as transações. Por ser uma árvore binária de busca auto-balanceada, garante que todas as operações principais (inserção, remoção, busca) ocorram em tempo logarítmico ($O(\log n)$), mesmo no pior caso. Essa estrutura se mostrou vantajosa especialmente em funcionalidades que exigem ordenação, como listagens crescentes por valor da transação ou filtros por intervalo.

Lista encadeada simples: por funcionar de maneira sequencial, essa estrutura é mais simples, mas ainda eficaz para percorrer todas as transações. Embora tenha um desempenho inferior para buscas diretas, ela oferece uma base útil para filtragens e operações que percorrem os dados de forma completa.

Fila: usada para representar o fluxo de chegada das transações, o que a torna útil em sistemas que precisam operar de forma sequencial ou em tempo real, como em filas de validação ou auditoria.

Pilha: aplicada para cenários em que se deseja manipular os dados de forma reversa, como no caso de histórico de ações do usuário ou operações que podem ser desfeitas.

2.2 Estruturas Recentes

Cuckoo Hashing: é uma variação avançada da tabela hash tradicional, foi adotado para melhorar a previsibilidade do tempo de acesso. Essa estrutura lida melhor com colisões ao alocar itens entre múltiplas posições usando duas funções hash diferentes, mantendo o tempo de busca garantido como constante, mesmo no pior caso.

Bloom filter: utilizado em conjunto com a hash table, foi aplicado para verificar rapidamente a existência de elementos, como transações duplicadas. Essa estrutura probabilística é extremamente eficiente em termos de memória e desempenho, permitindo respostas quase instantâneas. Embora possa retornar falsos positivos, ela nunca retorna falsos negativos, sendo ideal como etapa preliminar de validação antes de consultas mais pesadas.

Trie: utilizada para representar eficientemente dados textuais, como nomes de comerciantes ou categorias. A Trie permite buscas rápidas por prefixo, o que viabiliza funcionalidades como autocompletar e agrupamento hierárquico. O uso da AVL nos nós internos da Trie mantém os ramos balanceados, melhorando o desempenho geral das buscas em dados não numéricos.

3. Operações Essenciais

3.1 Operação de busca

- **Lista Encadeada:** A busca é linear com complexidade O(n), que indica que o tempo para achar um elemento cresce de acordo com o número de elementos n, já que é necessário percorrer os nós em sequência até encontrar o elemento desejado.
- **Hash Table:** Busca com complexidade O(1) média. A busca é eficiente pelo fato de ser feita procurando o array interno da própria função hash correspondente à identificação única.
- **Árvore AVL:** Busca com complexidade O(log n), isso a torna útil pois o fato do tempo estimado acompanhar o logaritmo de n, isso indica que o tempo é reduzido pela metade conforme seu avanço.
- **Fila:** Também realiza buscas com complexidade O(n), que acompanha o número de elementos n, pois também percorre todos os elementos até achar um específico.
- **Pilha:** Outra estrutura que busca com complexidade O(n), a principal diferença é que, apesar de percorrer todos os elementos, esta percorre de forma inversa.
- **Cuckoo Hashing:** Por ser uma estrutura de implementação de tabela hash, a complexidade de sua busca é de O(1). A diferença é que essa, por usar duas ou mais funções hash, guarda a informação em dois arrays, ou seja, ao realizar a busca, a estrutura procura diretamente em 2 acessos de memória.
- **Bloom Filter com hash table:** Buscas de complexidade O(k), onde k equivale ao número de funções hash, e apenas verificam se o elemento está presente, que podem resultar em falsos positivos.
- **Trie com Árvore AVL:** Busca com complexidade O(L), onde L é o comprimento da palavra, tornando a busca de IDs eficiente.

3.2 Operação de inserção

A operação de inserção foi implementada de forma específica para cada estrutura, considerando suas características particulares:

- **Lista Encadeada:** Inserção no início com complexidade O(1), ideal para carregamento sequencial dos dados. O novo nó sempre se torna a nova cabeça da lista.
- **Hash Table:** Inserção com tratamento de colisões por sondagem linear. A complexidade média é O(1), mas pode degradar com alta taxa de ocupação.
- **Árvore AVL:** Inserção com balanceamento automático através de rotações. Complexidade O(log n) garantida, mantendo propriedades da árvore balanceada.
- **Fila:** Há uma inserção no final com complexidade O(1), considerada ideal para verificar dados na ordem em que chegam (sistema FIFO)
- **Pilha:** Há uma inserção no final com complexidade O(1), que também é considerada como ideal para controle de chamadas.
- **Cuckoo Hashing:** Realiza inserções de complexidade de O(1) amortizado, ou seja, complexidade média de O(1), recomendada para reposicionamento de elementos que colidiram.
- **Bloom Filter com hash table:** Inserções de complexidade O(k), onde k equivale ao número de funções hash, adequado para conjuntos de dados volumosos, que aceitam falsos positivos mas não falsos negativos.
- **Trie com Árvore AVL:** Inserção com complexidade O(L). L é o comprimento da chave, sendo ideal para armazenamento e busca eficiente de strings com prefixos comuns, nesse caso os IDs.

3.3 Operação de remoção

- **Lista Encadeada:** A remoção é linear com complexidade O(n), já que é necessário identificar o nó anterior àquele elemento a ser removido.
- **Hash Table:** A remoção, com complexidade O(1), é realizada após a localização do elemento, já que a estrutura vai diretamente ao array que está localizado o dado a ser removido.
- **Árvore AVL:** Remoção com complexidade O(log n) para que seja possível restaurar o balanço da árvore.
- **Fila:** A remoção é realizada no início da fila e tem complexidade O(1), preferida para consumir dados de acordo com a ordem recebida.

- **Pilha:** Remoção é feita no topo da pilha, complexidade O(1), a melhor para desfazer ações.
- **Cuckoo Hashing:** Após localizar o elemento, a sua remoção é feita de forma simples, isto é, ocorre sem a necessidade de realocação ou reorganização dos dados, com uma complexidade de O(1).

4. Operações Adicionais

4.1 Filtragem

- **Lista Encadeada Simples:** É feita através de uma varredura simples, começando no primeiro nó da lista e percorrendo até o último, selecionando elementos a partir do critério que o usuário escolheu.
- **Fila:** Realiza uma varredura sequencial, passando pelo primeiro item da lista e indo até o último, exibindo apenas os dados que atendam ao critério selecionado pelo usuário. A diferença entre as duas varreduras é que esta permite começar a filtragem de onde são adicionados os elementos (cauda) ou onde são removidos elementos (cabeça), enquanto a da lista é obrigada a seguir a hierarquia dos nós.
- **Tabela Hash:** Examina cada array interno das funções hash e listas encadeadas, com complexidade O(n).
- **Cuckoo Hashing:** Varre a tabela completa, complexidade O(n).
- **Bloom Filter:** Capaz de verificar se é válido procurar um dado em bases maiores de dados, tornando-a mais eficiente.
- **Trie:** Utilizada para filtrar os IDs, que possuem prefixos que podem ser semelhantes.
- **Árvore AVL:** Muito eficiente para intervalos, já que utiliza a travessia em ordem, que percorre a árvore de forma sequencial, começando na raiz, percorrendo a subárvore à esquerda e logo após à direita, com cada nó sendo examinado apenas uma vez.

4.2 Agrupamento

- **Lista Encadeada Simples:** Feito por contadores durante a varredura..
- **Fila:** Foram utilizadas diversas filas para a separação dos dados durante a varredura.
- **Pilha:** Feito com pilhas auxiliares.
- **Tabela Hash:** Também utilizou de contadores durante a varredura.
- **Trie:** A estrutura em si tem como natureza a de agrupar a partir dos prefixos dos IDs.

4.3 Cálculos Estatísticos

- **Lista Encadeada Simples:** São calculados utilizando complexidade O(n) e varreduras.
- **Fila:** Resultados são obtidos após uma única varredura com complexidade O(n).
- **Pilha:** Percorre todos os elementos (complexidade O(n)) para chegar nos resultados.
- **Tabela Hash:** Resultados são obtidos através da análise de todos os arrays internos das funções hash. Complexidade O(n).
- **Cuckoo Hashing:** Feitos através da varredura completa da tabela, complexidade O(n).
- **Trie:** Utiliza-se da travessia em ordem completa de n IDs inseridos com complexidade O(n).
- **Árvore AVL:** Cálculos são realizados utilizando travessia ordenada, com complexidade O(n).

5. Solução da Tarefa e Otimização

Como parte da tarefa principal do sistema, foi feita uma detecção de transações financeiras potencialmente fraudulentas, e foi implementado um modelo de predição heurística diretamente associado à inserção de novas transações.

A heurística implementada, utilizada apenas na Tabela Hash, foi baseada na análise de maiores ocorrências de um determinado valor de cada variável para quando uma transação do próprio dataset é identificada como fraudulenta. Essa análise permitiu a realização de predições para novas transações inseridas pelo usuário, apresentando uma mensagem com o nível de risco (baixa, moderada ou alta).

Para Árvores AVL, seria utilizado bitfields, compostos por apenas 4 bits, visando economizar memória, que pode ser útil para dispositivos com restrições de hardware, como sistemas embarcados. Porém foi detectado que a utilização dessa ferramenta estava causando overflow. Logo, a ferramenta usada foi uint8_t, que utiliza de inteiros de 8 bits, para também otimizar memória. A altura máxima da árvore com 5 milhões de amostras seria em torno de 32 níveis, e o uint8_t suporta 255 níveis, garantindo segurança e evitando overflow.

6. Benchmark e Categorias de Restrição

6.1 Método de medições de benchmark

Os benchmarks foram executados em critério científico para garantir que os resultados tenham sido confiáveis e comparáveis entre as estruturas. Foram 10 repetições com dados gerados aleatoriamente para calcular os tempos médios e desvio padrão.

6.2 Ambiente de Simulação Controlada

Processador: Intel I5-12450H; Memória: 8GB RAM DDR4 3200MHz; Armazenamento: SSD NVMe M.2 256GB Leitura 2000 MB/s Gravação 1000 MB/s; SO: Windows 11 Home; Compilador: GCC 9.2.0 64-bit Release.

6.3 Análise dos Benchmarks Detalhados de Cada Operação

Análise das Inserções (10.000 elementos)

Tabela Hash: Tempo médio das inserções (13.0 ms ± 0.59 ms) - Coeficiente de Variação (5.53 %).

Árvore AVL: Tempo médio das inserções (11.4 ms ± 0.63 ms) - Coeficiente de Variação (4.61 %).

Lista Encadeada Simples: Tempo médio das inserções (12.9 ms ± 1.94 ms) - Coeficiente de Variação (15.0 %).

Análise das Buscas (10.000 elementos)

Tabela Hash: Tempo médio das buscas (0.75 ms ± 0.13 ms) - Coeficiente de Variação (17.0 %).

Árvore AVL: Tempo médio das buscas (0.73 ms ± 0.23 ms) - Coeficiente de Variação (31.1 %).

Lista Encadeada Simples: Tempo médio das buscas (342 ms ± 21.5 ms) - Coeficiente de Variação (6.30 %).

Análise das Remoções (1.000 elementos)

Tabela Hash: Tempo médio das remoções (0.10 ms ± 0.02 ms) - Coeficiente de Variação (20.4 %).

Árvore AVL: Tempo médio das remoções (0.21 ms ± 0.01 ms) - Coeficiente de Variação (6.90 %).

Lista Encadeada Simples: Tempo médio das remoções (36.9 ms ± 4.20 ms) - Coeficiente de Variação (11.3 %).

Análise de Memória (população estrutura 10.000 inserções)

Tabela Hash: Memória total estimada: 2320072 bytes (2265.70 KB).

Árvore AVL: Memória total estimada: 2159104 bytes (2108.50 KB).

Lista Encadeada: Memoria total estimada: 2240016 bytes (2187.52 KB).

6.4 Categoria de Restrição

Restrições utilizadas

As restrições utilizadas foram: R2, R10, R13 e R18.

R2: Limitação de tamanho: A estrutura de dados teve seu tamanho máximo limitado, simulando ambientes com recursos de memória restritos ou políticas de cache.

R10: Interrupções Periódicas: Pequenas pausas (1 milissegundo) foram introduzidas a cada 100 operações de inserção. Esta restrição simula latências e interrupções comuns em sistemas operacionais.

R13: Delay Artificial por Lote: Um atraso mais significativo (50 milissegundos) foi imposto a cada 100 transações inseridas.

R18: Inserção de Anomalias: Um percentual das transações geradas (10%) foi intencionalmente marcado como anômalo (com IDs, valores e tipos de transação incomuns ou negativos).

- **Análise das Buscas (10.000 elementos)**

  Tabela Hash: Tempo médio das buscas (0.338 ms ± 0.302 ms) - Coeficiente de Variação (89.53 %).

  Árvore AVL: Tempo médio das buscas (0.490 ms ± 0.465 ms) - Coeficiente de Variação (94.88 %).

Lista Encadeada Simples: Tempo médio das buscas (7.676 ms ± 0.438 ms) - Coeficiente de Variação (5.71 %).

- **Análise das Remoções (1.000 elementos)**

  Tabela Hash: Tempo médio das remoções (0.007 ms ± 0.000 ms) - Coeficiente de Variação (2.00 %).
  Árvore AVL: Tempo médio das remoções (0.021 ms ± 0.001 ms) - Coeficiente de Variação (3.84 %).
  Lista Encadeada Simples: Tempo médio das remoções (0.167 ms ± 0.012 ms) - Coeficiente de Variação (7.20 %).

- **Tempo Total (com 4 restrições aplicadas):**

  Tabela Hash: 714.614 ms
  Lista Encadeada Simples: 709.875 ms
  Árvore AVL: 735.540 ms

- **Análise de Memória (população estrutura com máximo de 500 elementos)**

  Tabela Hash: Memória total estimada: 192072 bytes (187.57 KB).
  Árvore AVL: Memória total estimada: 124016 bytes (121.11 KB).
  Lista Encadeada: Memória total estimada: 112016 bytes (109.39 KB).

7. Resultados e Conclusões

Os resultados dos benchmarks mostram dados interessantes sobre a performance de cada estrutura de dados essencial para as operações essenciais. Os testes mostraram que a árvore AVL leva ligeira vantagem ao realizar inserções, sendo 11,63% mais rápida que a Lista Encadeada e 12,31% mais rápida que a Tabela Hash.

Durante os testes de busca, a estrutura que destoa das outras é a Lista Encadeada Simples, sendo 456x mais lento que a Tabela Hash e até 468,5x mais lento que a Árvore AVL.

Nos testes de remoção, novamente a Lista Encadeada apresenta resultados piores que a Tabela Hash e a Árvore AVL, levando em média 36,9 ms para conclusão, contra 0.10 ms e 0.21 ms, respectivamente.

A Aplicação de Previsão de Fraudes Bancárias concluiu com êxito os objetivos traçados, demonstrando como o programa funcionaria em cenários da vida real. O funcionamento das estruturas de dados são comprovados a partir da comparação entre os resultados que apontavam transações fraudulentas já presentes no dataset e as informações exibidas sobre a transação inserida pelo usuário que exibe a probabilidade de essa transação ser fraudulenta (baixa, moderada ou alta), entre outras informações.

Visando um melhor funcionamento para possível aplicação profissional por parte de instituições financeiras privadas e públicas, foram realizados testes de benchmarks para que fossem escolhidas as estruturas que trouxessem os melhores resultados de otimização de tempo.

Mesmo que os dados sejam sintéticos, estes simulam fielmente características de transações financeiras reais. Por conta disso, é possível que outros setores utilizem a aplicação, como empresas ou repartições públicas responsáveis por segurança digital.

Referências

IAs generativas:

OPENAI. ChatGPT. Disponível em: <https://chatgpt.com/>.

Gemini: conversas que vão potencializar suas ideias. Disponível em: <https://gemini.google.com/app?hl=pt-br>.

Material contendo códigos:

ESD - Google Drive. Disponível em: <https://drive.google.com/drive/folders/1tDqUWIzdrpYXo4vnGCX-NcEjKbm93Z9a?usp=drive_link>. Acesso em: 10 jun. 2025.

Dataset

FINANCIAL_FRAUD_DETECTION_DATASET.CSV. financial_fraud_detection_dataset.csv. Disponível em: <https://drive.google.com/file/d/1u1KW4zuTAGuVeZRmaqyJ9kvNBdiUnZmc/view?usp=sharing>. Acesso em: 10 jun. 2025.