Algorithms and Data Structures

Algorithms

Our system includes the implementation of several algorithms, primarily to detect fraudulent activities. The most notable is the Hidden Markov Model (HMM) used for fraud detection.

Fraud Detection Algorithm (HMM):

* The fraud detection algorithm uses an HMM where each state corresponds to the likelihood of a transaction being fraudulent or legitimate based on historical patterns.
* Step 1: Initialization: Initialize the HMM model using a set of known transactions categorized as either fraud or legitimate.
* Step 2: Observation Sequence: Each transaction (with details like amount, time, location) serves as an observation in the sequence.
* Step 3: State Transition: Based on the new transaction, compute the transition between states (fraud and legitimate).
* Step 4: Output Probabilities: Compute the likelihood that the current transaction is fraudulent using the forward-backward algorithm of HMM.
* Step 5: Decision: If the likelihood surpasses a certain threshold, flag it as a fraudulent transaction.

Other Algorithms:

* Two's Complement Algorithm for signed binary calculations in financial systems. This algorithm checks the sign bit, inverts bits for negative numbers, and adds 1 to generate the two’s complement representation.
* Binary Search Algorithm is used to search for specific transaction IDs quickly in a sorted list of transactions, optimizing retrieval times.

Activity Diagram:

An activity diagram could be drawn to depict how transactions flow through the system and are processed by the fraud detection module, showing decision points where transactions are flagged or processed normally.

Data Structures

Our system uses several key data structures:

1. Hash Maps:
   * Used for mapping transaction IDs to transactions for quick lookup.
   * Chosen for performance reasons as hash maps offer O(1) average-case complexity for search and retrieval, which is crucial when processing large datasets in real-time.
2. Dynamic Arrays:
   * Used to store lists of transactions and fraud alerts.
   * Chosen for their flexibility as the transaction list grows dynamically over time.
3. Linked Lists:
   * Used to store recent fraudulent transactions that need to be monitored continuously.
4. Concurrent Data Structures:
   * These data structures are lock-free and help in handling multiple threads that perform simultaneous reads and writes to the database of transactions and alerts.

Concurrency:

Yes, our system employs multiple threads to handle different tasks:

* Fraud Detection Thread: Monitors transactions in real-time.
* Transaction Logging Thread: Logs each transaction asynchronously.
* Synchronization: Mutexes and condition variables are used to prevent race conditions between threads, ensuring that data consistency is maintained.

User Interface Design and Implementation

Based on the functional requirements identified in the initial phase of development, we focused on ensuring ease-of-use and minimal user effort. The design prioritizes intuitive navigation and logical grouping of controls and data fields to minimize the number of user interactions needed to complete tasks.

While no formal mock-up was presented, the user interface has been iteratively designed through direct user feedback and interface refinement. The current UI layout ensures that key functionalities are easily accessible and does not rely on unnecessary visual elements that could detract from usability.

In future iterations, should the system’s scope evolve to require more complex interactions, detailed mock-ups and prototypes may be developed to ensure user-friendly design principles are followed throughout the process.

Design of Tests

Unit Testing:

Each method in the system, particularly those responsible for transaction processing and fraud detection, will be tested independently.

Test Cases:

* Test Case 1: Valid transaction inputs should not trigger fraud alerts.
  + Expected Outcome: Transactions pass without being flagged.
* Test Case 2: Fraudulent transaction input triggers an alert.
  + Expected Outcome: Fraudulent transactions are flagged correctly by the system.
* Test Case 3: Hash map-based transaction lookup performance.
  + Expected Outcome: Transaction lookup time stays within O(1) for large datasets.
* Test Case 4: Two’s complement conversion for negative transactions.
  + Expected Outcome: Binary conversion works accurately for negative numbers.

Test Coverage:

Our test suite aims for at least 85% code coverage, focusing on the core functionality of the fraud detection system and critical transaction processing paths.

Integration Testing:

* We will conduct integration tests to ensure that the User, Transaction, and FraudDetector components work seamlessly together.
* These tests will verify that transactions are submitted by users, processed by the fraud detection system, and stored in the database without data loss or corruption.

Algorithm Testing:

For the Hidden Markov Model, we will design tests using a controlled set of transactions (both fraud and legitimate) to validate that the model correctly identifies fraud patterns based on training data.

Non-functional Requirements:

* Performance Testing: Ensure that the system can handle at least 100 transactions per second during peak usage times.
* Stress Testing: Subject the system to high volumes of transactions to check if it crashes or fails to flag fraud.

User Interface Testing:

* We will conduct usability tests to ensure that users can flag and approve transactions with minimal effort. This includes testing screen layouts and user navigation.