HOW DATA STRUCTURES AND ALGORITHMS ARE APPLIED IN SYSTEMS

Introduction

For this assignment, I researched how data structures and algorithms work within real-world systems, specifically focusing on a modern banking system. Banks handle millions of transactions daily while needing to be accurate, fast, and secure, making them a perfect example to study. Systems like those used by ABSA Bank or other financial institutions deal with challenges like data integrity, high traffic, and regulatory compliance, and the way they use data structures and algorithms is what makes it all possible.

When we talk about systems, we have to consider the hardware too. Accessing data from RAM is fast, but reading from a disk is much slower. This physical reality affects which data structures make sense to use. For example, a **B-Tree** is designed to minimize slow disk reads, while a **hash table** in a cache lives in fast RAM for quick lookups. Understanding this connection between hardware and software is the first step to understanding why certain data structures are chosen over others.

Data Structures in a Banking System

The first major data structure is the **B-Tree**. This is used in the core database that stores all account information and transaction histories. Banks use databases like Oracle , and they rely heavily on B-Trees for indexing. The reason B-Trees are so important is that they keep data in a balanced tree structure that requires very few disk reads to find anything. If you have millions of accounts, a B-Tree might only need to perform three or four disk seeks to locate a specific record. They also support range queries, which are essential when you need to pull up a month's worth of transactions. The trade-off is that they are slightly slower than hash tables for simple lookups and have some storage overhead, but for disk-based storage, they are the gold standard.Trees are used to maintain sorted data and support fast searching. B-Trees are especially used in databases, which store bank records.

Use in banking:

- Database indexing for account numbers
- Organizing customer loan records
- Managing large datasets efficiently

Advantage:

- Fast search and retrieval (O(log n))

Disadvantage:

- Complex to implement compared to arrays

Another critical data structure is the **hash table**, which you find in the caching layer. Banks use systems like Redis to store frequently accessed data like account balances in RAM. When a user checks their balance on a mobile app, the system first looks in this cache. Hash tables give O(1) lookup time, meaning the balance is retrieved almost instantly without touching the slower database. This is crucial for handling the massive number of read requests that come in every second. The downside is that hash tables don't store data in order, so they can't do range queries, and they don't persist data to disk, but for caching, that's perfectly fine because the data can always be fetched from the main database if needed.

Trees are used to maintain sorted data and support fast searching. B-Trees are especially used in databases, which store bank records.

Use in banking:

- Searching account details quickly
- PIN/password verification
- Storing user session data during online banking

Advantage:

- Very fast searching (average O(1))

Disadvantage:

- Collision problems if hashing is not well managed

The third structure is the **queue**, used in message brokers like RabbitMQ. When you use an ATM, your request doesn't go straight to the database. Instead, it goes into a queue. This acts as a **buffer**. If the core banking system is busy, the queue holds your request safely until it can be processed. Queues follow a **First-In-First-Out (FIFO)** discipline, which ensures fairness. They decouple the front-end services from the back-end, meaning a traffic spike at ATMs won't crash the main database. The trade-off is that queues add a tiny bit of latency since requests have to wait their turn, but the resilience they provide is worth it.

Use in banking:

- Processing withdrawals and deposits in ATM systems
- Handling online banking requests
- Managing customer service ticket systems

Advantage:

- Ensures fairness (First-In-First-Out system)

Algorithms in a Banking System

Moving on to algorithms, one of the most interesting ones is the **LRU (Least Recently Used) cache eviction algorithm**. Caches can't hold everything, so when they fill up, something has to be omitted. LRU solves this by removing the item that hasn't been accessed for the longest time. LRU is typically implemented using a hash table for fast lookups and a doubly linked list to keep track of the order items were used. When an item is accessed, it moves to the front of the list. When eviction happens, the item at the back gets omitted. It's a perfect example of how algorithms and data structures work together.

Dijkstra's algorithm also plays a role, mostly in inter-bank networks. When you make a payment to someone at a different bank, the transaction needs to find the most efficient route through the financial network. This network can be modelled as a graph, with banks as nodes and payment rails as edges that have costs or speeds attached. Dijkstra's algorithm finds the shortest path from the source bank to the destination bank, ensuring the transaction is routed quickly and cost-effectively.

For end-of-day processing, banks use **external sorting algorithms**, which are variations of merge sort. When calculating interest for millions of accounts, the data is often too large to fit into memory. External sorting reads chunks of data into memory, sorts them, writes them back to disk, and then merges all the sorted chunks together. This allows the bank to process accounts in order and update them efficiently, even with massive datasets.

Conclusion

In conclusion, it is seen that data structures and algorithms are not just theoretical concepts. They are the actual building blocks that make complex systems like banks work. The choice between a B-Tree and a hash table directly impacts whether a user waits milliseconds or seconds for their balance. Queues prevent systems from falling over during peak times. Algorithms like LRU optimize how resources are used. Without these fundamentals, modern banking wouldn't be possible. This assignment has given me a much deeper appreciation for why we study these topics and how they apply to the real world.