**UNIVERSITY OF GHANA**

**DEPARTMENT OF COMPUTER SCIENCE**

**DCIT308: DATA STRUCTURES AND ALGORITHMS 2**

**SECOND SEMESTER 2024/2025**

**SEMESTER PROJECT (GROUP 75)**

**Project Title: Expenditure Management System for Nkwa Real Estate Ltd.**

**Introduction**

The Expenditure Management System is a console-based application developed to assist Nkwa Real Estate Ltd in efficiently managing and analyzing its financial expenditures across various operational phases. The system, which is implemented in Java, makes use of basic data structures such as sets, arrays, stacks, queues, linked lists, maps, trees, and hash maps. Advanced functions including sorting and searching are supported, along with dynamic entry and tracking of expenses, processing of receipts, and balance monitoring. Performance, modularity, and adherence to data structure principles were all given top priority during the system's development.

**Data Structure Justification**

This section outlines the idea behind the selection and application of various data structures in the system. The goal is to match each data structure’s strengths with specific system requirements, ensuring optimal performance and clarity in implementation.

* 1. **Array**
* Arrays are employed temporarily when sorting records, particularly expenditures by category or date
* Arrays work with effective built-in sorting algorithms and offer constant-time access to items using indices. For short-term processing when size is predictable, their static nature is appropriate.
  1. **Linked List**
* Expenditures are sorted in a LinkedList to maintain their order of entry.
* Linked lists offer efficient sequential access and dynamic memory usage, making them suitable for operations where insertions at the end are frequent and the total size is not fixed.
  1. **Stack**
* The system uses stack to manage receipt reviews.
* A stack follows the Last-In-First-Out (LIFO) principle, which is ideal for accessing the most recently added receipts, mirroring real-world approval workflows.
  1. **Queue**
* The queue is used to handle uploaded receipts.
* A queue operates in First-In-First-Out (FIFO) manner, ensuring that receipts are processed in the order they were submitted. This preserves chronological integrity.
  1. **Set (HashSet)**
* Sets manage the collection of unique expenditure categories
* The set data structure enforces uniqueness and provides fast insertion and lookup operations. This is essential to avoid duplicate of category entries.
  1. **Map (HashMap)**
* Maps are used for quick retrieval of expenditure and bank account information using unique codes.
* A hash map provides constant-time average complexity for searches and updates, which significantly enhances system responsiveness when handling frequent queries.
  1. **Tree (Min-Heap)**
* A min-heap tracks bank account with the lowest balance
* Min-heaps support constant-time minimum retrieval and logarithmic insertion. This makes them suitable fir prioritizing accounts that may require attention due to low balances.

1. **Sorting and Searching Techniques**
   1. **Sorting**

* By Category: Implemented using Java’s ArrayList.sort() method and a custom comparator. It uses dual-pivot quicksort under the hood, providing efficient sorting for string data.
* By Date: Sorted using a comparator on ISO-formatted date strings, leveraging lexicographic ordering to reflect chronological order.
  1. **Searching**
* By Category: Implements linear search through the linked list of expenditures.
* By Time Range: Utilizes substrings matching on the date field using startsWith().
* By Cost Range: Performs numeric comparisons to filter expenditures within the specified range
* By Bank Account: Matches expenditures by comparing the bank account Id field.

**Complexity Analysis (Big O / Omega)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Best Case (Ω)** | **Worst Case (O)** | **Structure** |
| Insert Expenditure | Ω (1) | O (1) | LinkedList |
| Add to Receipt Queue | Ω (1) | O (1) | Queue |
| Search by Category/ Cost/ Account | Ω (1) | O (n) | LinkedList |
| Sort by Category/ Date | Ω (n log n) | O (n log n) | ArrayList |
| Hash Lookup | Ω (1) | O (1) | HashMap |
| Add Category to Set | Ω (1) | O (1) | HashSet |
| Min Retrieval in Heap | Ω (1) | O (1) | MinHeap |
| Insert into MinHeap | Ω (n log n) | O (n log n) | MinHeap |

The complexity analysis shows the best-case (Ω) and the worst-case (O) time complexities for the key operations performed by the system, alongside the data structure used to support each operation. Each complexity value indicates how the system’s performance scales as the number of expenditures, accounts, or categories increases.

* **Insert Expenditure (O (1)):** Inserting into LinkedList is fast, as it is always added to the end.
* **Add to Receipt Queue (O (1)):** Adding to a queue is quick and does not depend on the queue’s size.
* **Search by Category/Cost/Account (O (n)):** Linear search is used on the LinkedList, so in the worst case it checks every item.
* **Sort by Category/Date (O (n log n)):** Sorting uses efficient algorithms, but still depends on the number of items.
* **Hash Lookup (O (1)):** Looking up data by key like an expenditure code in a HashMap is fast and does not grow with data size.
* **Add Category to Set (O (1)):** Inserting into a HashSet is fast and automatically avoids duplicates.
* **Min Retrieval in Heap (O (1)):** Retrieving the smallest account balance in a MinHeap is instant.
* **Insert into Minheap (O (n log n)):** Inserting into a heap may involve reorganizing the structure, taking a few steps based on the number of elements.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **ID** | **Activities** | **%Contribution** | **Attendance** | **Days Present** |
| **Prince Biney** | **11015739** | **Worked on Search and Sort under functional requirement.** | **14%** | **1/07/25, 7/07/25, 11/07/25** | **3/3** |
| **Solomon Ayekumse Apiok** | **11285821** | **Worked on the Invoice under functional requirement.** | **14%** | **1/07/25, 7/07/25, 11/07/25** | **3/3** |
| **Jacob Angabey** | **11096501** | **Worked on the project documentation.** | **14%** | **1/07/25, 11/07/25** | **2/3** |
| **Kwodwo Afful Addison** | **11116803** | **Worked on Category Management under functional requirement.** | **14%** | **1/07/25, 7/07/25, 11/07/25** | **3/3** |
| **Markarious Anaba Aporibe** | **11298742** | **Worked on the Cash Flow and Financial Analysis under functional requirement.** | **14%** | **1/07/25, 7/07/25, 11/07/25** | **3/3** |
| **Samuel Tandoh Nsiah** | **11108240** | **Worked on the Expenditure Records under functional requirement, and codes reviewing.** | **16%** | **1/07/25, 7/07/25, 11/07/25** | **3/3** |
| **Jacob Arhin** | **11051341** | **Worked on the Bank Account Ledger under functional requirement.** | **14%** | **1/07/25, 7/07/25, 11/07/25** | **3/3** |