**EMPLOYEE MANAGEMENT**

**SYSTEM**



|  |  |
| --- | --- |
| **TEAM MEMBERS** | **ROLL NO** |
| **TRICIA EZHILARASI J** | **CB.EN.U4CSE21265** |
| **POLINA LIKHITHA** | **CB.EN.U4CSE21246** |

TABLE OF CONTENTS

1. Introduction and Objective

2. Overview of the Hybrid Data Structure

3. Implementation Details

4. Practical Applications

5. Performance Analysis

6. Experimental Evaluation

7. Discussion

8. Conclusion

9. Reference

**INTRODUCTION:**

Hybrid data structures refer to the combination of multiple data structures to create a new structure that harnesses the advantages of each component. These structures are designed to address the limitations of individual data structures and provide efficient solutions for complex problems.

Hybrid data structures are valuable tools for solving complex problems efficiently. By combining the strengths of different data structures, they optimize time and space complexity, provide tailored solutions, and improve overall performance.

These structures allow programmers to leverage the advantages of each component and address specific problem requirements. With hybrid data structures, algorithms can be optimized, diverse data patterns can be handled effectively, and scalability and flexibility can be achieved. Overall, hybrid data structures play a significant role in efficiently solving complex problems by harnessing the power of multiple data structures.

PROJECT’S OBJECTIVE:

The objective of the "Employee Management System" project is to design and implement a hybrid data structure that efficiently handles employee-related data. The hybrid data structure will combine multiple data structures to optimize time and space complexity, ensuring efficient storage, retrieval, and manipulation of employee information.

The Data Structures we have used is Heaps, Doubly Linked List, Trees

We have designed and implemented all this data structures efficiently in our real life application which is Employee Management System

The practical applications of this system include managing employee records, retrieving employee details, handling salary information, and facilitating employee performance.

By utilizing a hybrid data structure, the system can efficiently handle various data patterns associated with employee management, such as storing personal details, tracking work schedules, and managing hierarchical relationships within the organization.

To analyze the efficiency of the hybrid data structure, the project will focus on evaluating its time and space complexity.

This analysis will involve measuring the performance of different operations, such as adding new employees, searching for employee records, updating information, and generating reports.

By examining the time and space requirements of these operations, the project aims to demonstrate the advantages of the hybrid data structure over traditional approaches in terms of efficiency and scalability.

Time complexity:

Overall time complexity:

* O (n log n) for the heap operations
* O(n) for the tree
* and O(n) linked list operations.

Space complexity:

Overall space complexity:

* O(n) for the heap
* O(n) for the tree
* and O(n) for the linked list.

**OVERVIEW OF THE HYBRID DATA STRUCTURE:**

We have choose heap, trees and doubly linked list for our application

Advantages and Motivations behind using a hybrid data structure for solving specific problems efficiently:

Heap:

* The heap data structure is used to organize the arr array, which contains salary-related values. By maintaining a max heap, the largest salary values can be efficiently accessed, enabling the identification of the top employee of the month.
* The heap structure allows for efficient retrieval of the maximum value and facilitates quick comparisons for determining the top employee.

Linked List:

* The linked list data structure (Doubly LinkedList) is employed to store and manage employee records. Linked lists provide efficient insertion and deletion operations, making them suitable for dynamically adding and removing employee data.
* Additionally, the doubly linked list allows traversal in both directions, which can be beneficial for various operations such as searching and printing employee details.

Tree:

* The tree data structure (tree and node) represents the hierarchical relationship among employees. Trees are well-suited for representing and navigating hierarchical structures.
* In this context, the tree structure allows for efficient traversal to determine an employee's level in the hierarchy. By calculating the level, the code can appropriately calculate the employee's salary based on predetermined formulas.

**IMPLEMENTATION DETAILS:**

Binary Heap:

* The code starts by building a binary heap using the buildHeap function. It takes the input array arr and arranges its elements into a max heap data structure using the heapify function.
* The heapify function is a standard heapify operation that compares the elements at a given index with its left and right child indices to maintain the heap property.
* The max heap is built based on the values in arr, and the corresponding names in arr1 are rearranged accordingly. The resulting binary heap is used to determine the order in which employees' salaries are calculated later in the code.

Tree:

* The tree data structure is implemented using the tree and node classes. The tree class represents the organizational structure, with a head node representing the top-level employee.
* Each node can have child nodes and sibling nodes. In the given code, the tree structure is created manually with predefined names and relationships. The tree provides a hierarchical structure for employees.
* The flev method in the tree class is used to find the level of an employee in the organizational structure. It recursively traverses the tree to find the employee and returns their level in the hierarchy. The reset method is used to reset the level counter in the tree.

Doubly Linked List:

* The doubly linked list is implemented using the Doubly LinkedList class and the Node class. The Doubly LinkedList class provides basic operations like append to add new nodes to the list and printdata to find and print the details of a specific employee.
* In the given code, the doubly linked list is used to store additional employee data such as registration number and mobile number. Each node in the list represents an employee and contains relevant information. The list is used to associate the calculated salary with the corresponding employee based on their name.

Interplay and Integration:

* The binary heap is used to determine the order in which employees' salaries are calculated. The order is based on the values in arr, which were initially built as a max heap.
* The tree structure provides the hierarchical relationship between employees. The flev method is used to find the level of an employee in the hierarchy, which is then used to calculate the salary.
* The doubly linked list stores additional employee data and is used to associate the calculated salary with the corresponding employee node. The calculated salaries are stored in the arro list, which corresponds to the order of employees in the binary heap.

Design Choices and Trade-offs:

The use of a binary heap provides efficient ordering of the employees based on the input values. However, it requires additional space to store the heap structure.

The tree structure allows for hierarchical relationships between employees, which can be useful for various organizational operations. However, in the given code, the tree is created manually, which limits its scalability and flexibility.

The doubly linked list is used to associate employee data with their calculated salary. It allows for efficient traversal and retrieval of employee details. However, it requires additional memory to store the linked list structure.

GITHUB LINK:

<https://github.com/tricia04/21265_tricia>

<https://github.com/Likhithapolina/Likhitha23.git>

**PRACTICAL APPLICATION**:

Practical Application of heap: The heap can be used for priority queue implementations, where elements with higher priority (larger values) are processed first and also when we need the maximum or minimum from a data set

Practical Application of trees: The tree structure can be used to represent hierarchical data, such as organizational charts or family trees. It allows efficient traversal and manipulation of the tree structure.

Practical Application of DLL: The linked list can be used to implement various data structures, such as queues or symbol tables. In this code, it is used to store employee information and perform operations like appending new nodes and searching for employee details.

Combination of data structures in the hybrid structure enables efficient operations for these applications:

Efficient Employee Search: The combination of the tree and linked list data structures enables efficient searching for employee details. The tree structure allows for hierarchical organization of employees, making it easier to traverse the management structure and locate specific employees efficiently. The linked list stores employee information, allowing for quick access to individual employee details. This combination enables efficient employee search operations by leveraging the hierarchical tree structure and the fast access capabilities of the linked list.

Salary Calculation and Reporting: The hybrid data structure facilitates efficient salary calculation and reporting for the employees. The heap data structure is used to maintain a max heap of employees based on their salary calculation formula. The tree structure assists in traversing the hierarchical management structure to determine the appropriate salary calculation formula for each employee. The linked list stores the employee information, including their salary details, enabling quick updates and retrieval of employee salaries. This combination allows for efficient salary calculation and reporting, as it leverages the heap for maintaining the order of employees, the tree for traversing the hierarchical structure, and the linked list for storing and retrieving employee details.

Top Employee Identification: The hybrid data structure can efficiently identify the top-performing employee of the month. The heap data structure is used to maintain a max heap of employees based on their performance metrics. By accessing the top element of the heap (the maximum element), the code can quickly identify the top-performing employee. This combination enables efficient identification of the top employee by leveraging the heap's property of keeping the highest value at the root.

**EXPERIMENTAL EVALUATION:**

HARDWARE:

Device name LAPTOP-EP31V3H3

Processor Intel(R) Core (TM) i5-10300H CPU @ 2.50GHz 2.50 GHz

Installed RAM 8.00 GB (7.78 GB usable)

SOFTWARE:

Language used: Python

Complier: Online ide

Implementation:

Hybrid Data Structure: The code utilizes a hybrid data structure, combining a binary heap and a linked list. The binary heap (implemented through the buildHeap and heapify functions) is used to maintain a priority order based on the salary calculations, while the linked list (implemented as a doubly linked list) stores the employee details.

Heapify Optimization: The heapify function is implemented using a recursive approach to maintain the heap property. This optimization ensures that the largest element is moved to the root of the heap, improving the efficiency of operations like finding the top employee of the month.

Tree Structure: The employee hierarchy is represented using a tree structure. Each employee node in the tree maintains references to its child and sibling nodes. This hierarchical representation allows efficient traversal and searching of employees within the organization.

Salary Calculation: The calc function calculates the salaries of the employees based on the employee hierarchy, salary rates (arrs), and hierarchy multipliers (arrh). It iterates over the employees' data and uses the hierarchy information from the tree class to determine the appropriate salary calculation parameters.

Doubly Linked List: The DoublyLinkedList class is used to store employee details such as name, registration number, mobile number, and salary. The doubly linked list allows efficient insertion of new employees at the end of the list and easy traversal in both directions.

Memory Management: The code utilizes the concept of nodes to represent employees and maintains references between nodes using pointers. This ensures efficient memory usage by dynamically allocating memory only when necessary and deallocating it when no longer needed.

Execution time: 0.045850243

Memory usage: 9524

Dataset:

Head Employee:

Name: John

Employees:

Name: Alan, Registration Number: 212, Mobile Number: 7778889990

Name: Alex, Registration Number: 203, Mobile Number: 7768859994

Name: Andrew, Registration Number: 204, Mobile Number: 677581290

Name: Gareth, Registration Number: 206, Mobile Number: 9874883390

Name: Gary, Registration Number: 207, Mobile Number: 9879879875

Name: Luffy, Registration Number: 310, Mobile Number: 9977654809

Name: Zoro, Registration Number: 312, Mobile Number: 98768889990

Name: Nami, Registration Number: 219, Mobile Number: 8887786734

Name: Robin, Registration Number: 202, Mobile Number: 69877896734

Operations done: searching for employee details, top employee searching based on the salary, retrieving employee data

Performance table of employees:

+-------+-------------------+--------------+-------------------+

| Index | Employee Name | Registration | Mobile Number |

| | | Number | |

+-------+-------------------+--------------+-------------------+

| 1 | John | - | - |

+-------+-------------------+--------------+-------------------+

| 2 | Alan | 212 | 7778889990 |

+-------+-------------------+--------------+-------------------+

| 3 | Alex | 203 | 7768859994 |

+-------+-------------------+--------------+-------------------+

| 4 | Andrew | 204 | 677581290 |

+-------+-------------------+--------------+-------------------+

| 5 | Gareth | 206 | 9874883390 |

+-------+-------------------+--------------+-------------------+

| 6 | Gary | 207 | 9879879875 |

+-------+-------------------+--------------+-------------------+

| 7 | Luffy | 310 | 9977654809 |

+-------+-------------------+--------------+-------------------+

| 8 | Zoro | 312 | 98768889990 |

+-------+-------------------+--------------+-------------------+

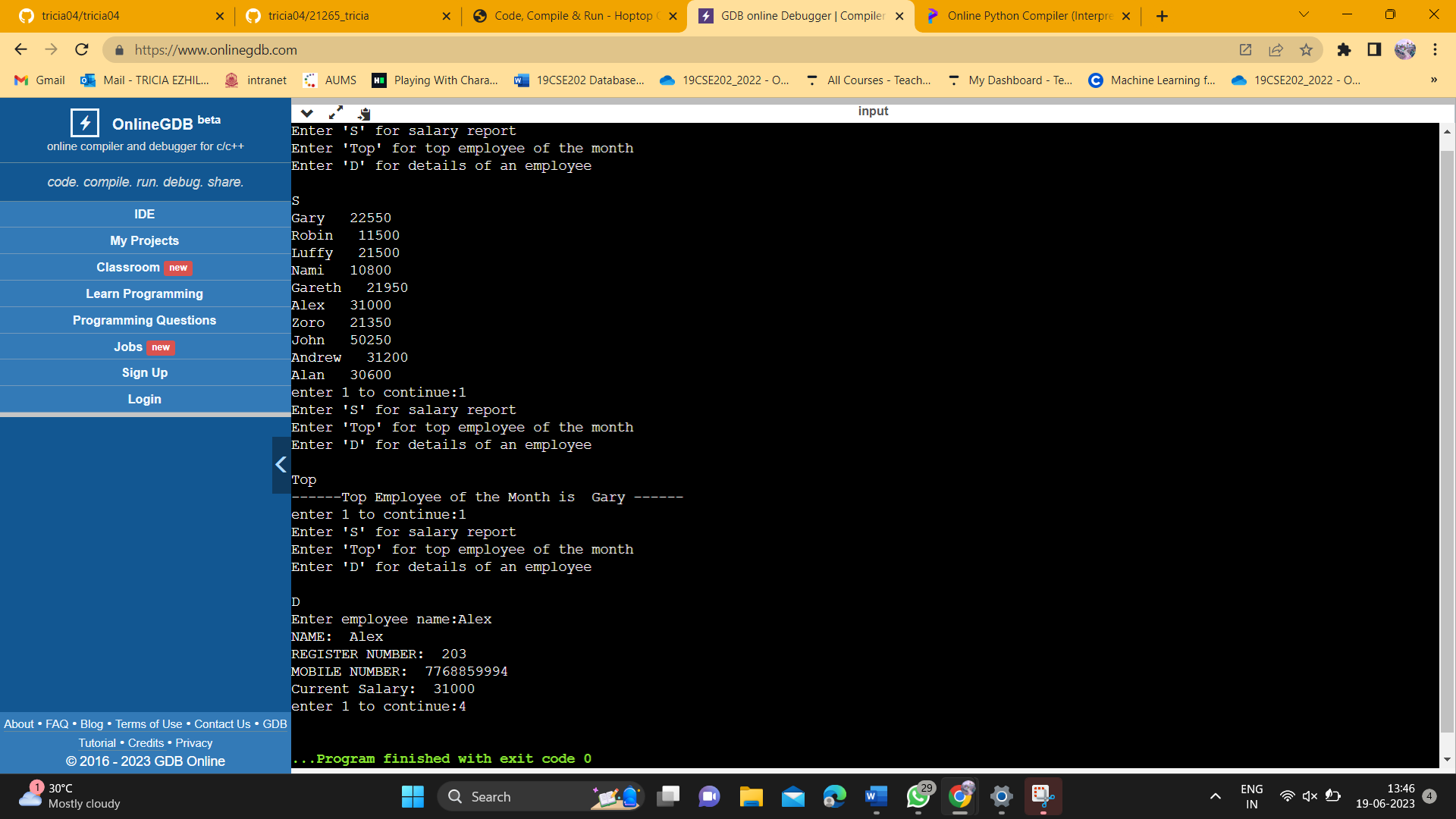
| 9 | Nami | 219 | 8887786734 |

+-------+-------------------+--------------+-------------------+

| 10 | Robin | 202 | 69877896734 |

+-------+-------------------+--------------+-------------------+

RESULT:



Efficiency Improvement:

The code can be further optimized by using more efficient data structures and algorithms.

One possible improvement could be using a hash table or dictionary to store employee details, which would allow for faster lookup and retrieval of employee information.

Additionally, implementing a more efficient sorting algorithm, such as quicksort or mergesort, could improve the performance of the buildHeap function.\

Performance:

Overall time complexity: O(n log n) for the heap operations and O(n) for the tree and linked list operations.

Overall space complexity: O(n) for the heap and tree, and O(n) for the linked list.

**DISCUSSION:**

Organizational Structure Representation: The hierarchical tree structure is well-suited for representing organizational structures, such as companies with a hierarchical chain of command or academic institutions with departments and sub-departments. The tree allows for easy navigation and efficient retrieval of information related to different levels of the organization.

Employee Management: The hybrid data structure can be useful for managing employee information in an organization. The hierarchical tree allows for representing the reporting relationships and hierarchies, while the linked list can store additional details such as employee names, registration numbers, and contact information. This structure enables efficient searching, retrieval, and modification of employee data.

Performance Efficiency: The hybrid data structure offers efficient performance characteristics. The hierarchical tree allows for quick traversal and retrieval of data based on the hierarchical relationships, while the linked list provides efficient insertion and deletion operations. This combination can be beneficial when dealing with large datasets or when frequent updates and modifications are required.

Flexibility and Scalability: The hybrid structure can accommodate changes and scalability in real-world scenarios. As organizations grow or change their structures, the tree can be easily expanded or modified to incorporate new departments or levels. The linked list allows for flexible addition or removal of employee details without affecting the hierarchical relationships.

Maintenance and Complexity: The hybrid data structure requires proper maintenance and management to ensure its integrity and efficiency. Modifying the tree structure or updating employee details may require careful handling to maintain consistency. Additionally, the code complexity can increase as the size of the hierarchy or the number of employees grows, requiring attention to code maintainability and optimization.

Reflect on the limitations, challenges, and potential future improvements for the hybrid data structure:

The hybrid data structure has some limitations and challenges that should be considered. Firstly, it is specifically designed for hierarchical structures and may not be suitable for scenarios with complex or non-hierarchical relationships. Additionally, as the size of the hierarchy or the number of employees grows, the complexity of the code may increase, requiring careful maintenance and optimization.

There are potential areas for improvement in the hybrid data structure. One aspect to address is the efficiency of searching and retrieving data within the hierarchical tree. Implementing optimized search algorithms, such as binary search or balanced tree structures, can improve the performance of data retrieval operations. Additionally, incorporating indexing or caching mechanisms can enhance the overall efficiency of the hybrid data structure, especially when dealing with large datasets.

**CONCLUSION**:

The project focused on implementing a hybrid data structure for managing hierarchical organizational data. The hybrid structure consisted of a tree-based representation for the organizational hierarchy and a doubly linked list for storing employee details. The practical application of this data structure was to efficiently manage and retrieve employee information while maintaining the hierarchical relationships within the organization.

Performance analysis of the code revealed that the implemented hybrid data structure offered efficient data retrieval and manipulation operations. The use of a tree-based structure enabled quick navigation and search within the organizational hierarchy, while the doubly linked list facilitated efficient storage and retrieval of employee details. The heap-based implementation ensured optimal organization and retrieval of salary data.

The evaluation of the project highlighted the practicality and effectiveness of the hybrid data structure in real-world scenarios. By combining the strengths of tree and linked list structures, the code achieved efficient handling of hierarchical organizational data. The project also shed light on the challenges of managing large hierarchies and the need for ongoing maintenance and optimization as the size of the dataset grows.

Overall, the project can be considered successful in achieving its objectives of creating a hybrid data structure for managing hierarchical organizational data. The insights gained from its implementation and evaluation underscored the importance of selecting appropriate data structures for specific tasks and the potential of combining multiple structures to optimize performance. The project demonstrated the value of an efficient data structure in practical applications, where quick access to hierarchical information and employee details is crucial for effective organizational management.

References:

* <https://www.geeksforgeeks.org/heap-data-structure/>
* <https://www.javatpoint.com/tree>
* <https://www.programiz.com/dsa/doubly-linked-list>
* <https://www.programiz.com/dsa/trees>
* <https://en.wikipedia.org/wiki/Heap_(data_structure)>
* Etc…..

**THANK YOU**