**Event Management System**



**BTech/II Year CSE/IV Semester 19CSE212/Data Structures and Algorithms Case Study Report**

|  |  |
| --- | --- |
| **Roll.No** | **Name** |
| CB.EN.U4CSE21218 | Devanshu Vasishth |
| CB.EN.U4CSE21256 | Sasidhar Maddali |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Department of Computer Science and Engineering** | | |  | |
| **Amrita School of Computing, Coimbatore** | | |
| **Amrita Vishwa Vidyapeetham** | | |
| **2022 -2023 Even Semester** | | |
| **Table of Contents** | | |
| **Title**  Chapter 1 | Introduction | | | **PageNo**  4 | |
| 1.1 | Objective | | | 4 | |
| 1.2 | Significance | | | 4 | |
| Chapter 2 | Implementation | | | 5 | |
| 2.1 | Integration Of Unordered Map | | | 6 | |
| 2.2 | Integration Of List | | |  | |
| 2.3 | Design choices and Trade Offs | | | 6 | |
| Chapter 3 | Practical Applications | | | 7 | |
| Chapter 4 | Performance Analysis | | | 8 | |
| 4.1  4.2  4.3 | Time Complexity  Space Complexity  Output | | | 8  9  10 | |
| Chapter 5 | Discussion | | |  | |
|  | |  |  | |
|  | |  |  | |
|  |  | | |  | |
| Chapter 6  Chapter 7 | Conclusion  References | | | 12  13 | |

# Chapter 1 Introduction:

The event management system is a software solution designed to facilitate the organization and management of various events. It provides a streamlined approach to handling event-related tasks, including event creation, attendee management, and event details retrieval. The system described in the provided code implements a basic event management functionality using various data structures and algorithms.

# Objective:

The objective of the project is to develop an Event Management System using a combination of different data structures to efficiently store and manipulate event data. The project utilizes a hybrid data structure comprising of a  hash-map and list to store events, allowing fast access based on event names. The system supports operations like adding and removing events, adding and removing attendees, viewing event details, and maintaining a waiting list for events with conflicting dates.

Practical applications of this data structure could include any problem that requires low time complexity.

When analyzing the time complexity of the implemented operations, most of them have a time complexity of O(1) due to the efficient access provided by the unordered map. However, some operations like removing an attendee or undoing the last operation may require traversing the list of attendees or undo stack, resulting in a complexity of O(n) in the worst case, where n is the number of attendees or operations.

Regarding space complexity, the project's data structures consume memory proportional to the number of events, attendees, and operations performed. The space complexity can be approximated as O(E + A + U), where E represents the number of events, A represents the total number of attendees across all events, and U represents the number of operations stored in the undo stack.

# Significance of Hybrid Data Structures:

Hybrid data structures combine the strengths of different data structures to solve complex problems more efficiently. They leverage the unique characteristics and operations of each data structure to optimize performance, memory usage, or both.

The significance of hybrid data structures lies in their ability to address the limitations of individual data structures when dealing with specific problem domains. By combining different data structures, they can provide better time and space complexity for certain operations, leading to more efficient algorithms and improved overall performance.

# Chapter 2 Implementation:

The chosen hybrid data structure in the provided project is a combination of an unordered map (hash map) and a list. The unordered map is used to store and retrieve events efficiently based on their names. The list is employed to maintain a list of attendees for each event.

Advantages and Motivations:

Using a hybrid data structure that combines an unordered map and a list offers several advantages:

1. Efficient Access and Retrieval: The unordered map provides fast access and retrieval of events based on their names, thanks to its hashing and indexing mechanisms. This ensures efficient searching and lookup operations.
2. Memory Efficiency: The list is utilized to store the attendees' names, which allows for flexible and efficient memory allocation. Unlike an array-based structure, the list doesn't require contiguous memory, enabling efficient memory management for `variable-sized lists of attendees.

1. Flexibility and Scalability: The hybrid data structure can handle a varying number of events and attendees. The unordered map allows for quick addition, removal, and lookup of events, while the list allows for efficient management and manipulation of the attendees' list.

1. Simplified Implementation: Combining an unordered map and a list simplifies the implementation of various operations. For instance, adding and removing attendees becomes more straightforward and efficient by utilizing the list's insertion and deletion capabilities.

By leveraging the strengths of both the unordered map and list, the hybrid data structure provides an efficient solution for managing events and attendees, offering fast access, memory efficiency, flexibility, and simplified implementation for the specific problem of an Event Management System.

**Chapter 2.1 Integration of the Unordered Map (Hash Map):**

* The unordered\_map container is used to store events, with the event name as the key and the corresponding Event object as the value.
* Events can be efficiently added, removed, and accessed using their names, leveraging the fast lookup capabilities of the unordered map.
* The events unordered map is declared as a global variable to ensure its availability across different functions.

**Chapter 2.2 Integration of the List:**

* Each Event object stored in the unordered map contains a list to store the list of attendees.
* The list is implemented using the list container from the STL.
* Attendees can be added or removed from the list associated with a specific event.
* The list allows for efficient insertion and deletion of attendees, ensuring flexible management of attendees' lists.

**Chapter 2.3 Design Choices and Trade-offs:**

Use of Unordered Map:

The choice of an unordered map allows for fast event lookup based on their names, providing O(1) average-case complexity for operations like adding, removing, and accessing events.However, the unordered map doesn't maintain the events in a specific order, which might be a trade-off if ordered traversal or sorting of events is required.

Use of List for Attendees:

The list provides efficient insertion and deletion of attendees, allowing flexible memory allocation for variable-sized attendee lists.However, accessing a specific attendee or searching for an attendee within the list requires traversing the list, resulting in O(n) time complexity in the worst case, where n is the number of attendees.

# Chapter 3 Practical Applications

Ticketing Systems:

* Ticketing systems, such as those for concerts, theaters, or sports events, can benefit from the hybrid data structure.
* The unordered map can store information about different events and their corresponding ticket availability, allowing quick lookup and updating of ticket availability.
* The list can be used to maintain a list of attendees who have purchased tickets, facilitating efficient ticket validation and attendee management.

Reservation Systems:

* Hybrid data structures can be employed in reservation systems, such as hotel bookings or restaurant reservations.
* The unordered map can store reservation details for different entities (e.g., hotel rooms, tables), enabling fast retrieval and modification of reservation information.
* The list can be utilized to track the guests associated with each reservation, allowing efficient addition and removal of guests.

Project Management Tools:

* Project management tools often require efficient tracking of tasks, deadlines, and team members.
* The unordered map can store information about different tasks or projects, providing quick access and management of task details.
* The list can be used to maintain a list of team members associated with each task, allowing efficient assignment and tracking of team members.
* The combination of an unordered map and a list in the hybrid data structure enables efficient operations for these applications:

The unordered map provides fast access to events, tasks, reservations, or other entities based on their unique identifiers, facilitating quick look-up and modification. The list allows for flexible management of attendees, guests, or team members associated with each entity, supporting efficient addition, removal, and traversal of the list.

The hybrid data structure combining an unordered map and a list has its limitations, challenges, and potential areas for future improvements:

* Memory Overhead: The hybrid data structure might have a higher memory overhead compared to a single data structure. The combination of an unordered map and a list requires memory allocation for both structures, potentially consuming more memory.
* Lack of Concurrency Support: The provided implementation of the hybrid data structure does not incorporate concurrency control mechanisms. In scenarios where multiple threads or processes access and modify the data structure simultaneously, concurrency-related issues like race conditions can arise. Introducing appropriate synchronization mechanisms or exploring thread-safe data structures can address this challenge
* Optimizing Memory Usage: Although the list provides flexibility in managing variable-sized lists of attendees, it can lead to memory fragmentation. Exploring alternative data structures that offer efficient memory utilization, such as dynamic arrays or self-balancing binary trees, could be beneficial in certain scenarios.

# Chapter 4 Performance Analysis

* 1. **Time Complexity:**

Adding an Event: When adding an event, the code iterates over the existing events to check for date conflicts. This requires iterating through the events map, resulting in a time complexity of O(n), where n is the number of events.  
  
 Removing an Event: When removing an event, the code searches for the event in the events map, which has an average time complexity of O(1) using an unordered map. However, if there is an event in the waiting list with the same date, the code iterates through the waiting list to find it, resulting in a time complexity of O(m), where m is the number of events in the waiting list.  
  
Adding and Removing an Attendee: Adding and removing an attendee involves finding the event in the events map, which has an average time complexity of O(1) using an unordered map. The attendee is then added or removed from the event's list of attendees, which takes O(1) time complexity.  
   
Viewing Event Details and Viewing All Events: These operations involve iterating through the events map and accessing each event's details. Therefore, the time complexity is O(n), where n is the number of events.   
Viewing Waiting List: The code iterates through the waiting list to display the details of each event. Therefore, the time complexity is O(m), where m is the number of events in the waiting list.   
  
Undoing Last Operation: Undoing the last operation involves popping an event from the undoStack and updating the events map. Both operations have a time complexity of O(1).  
   
Overall, the time complexity of the code depends on the specific operation performed but can be summarized as follows:  
O(1) for adding/removing an attendee and undoing the last operation.   
O(1) average case and O(n) worst case for adding/removing an event.   
O(n) for viewing event details and viewing all events.   
O(m) for viewing the waiting list and removing an event with a waiting event on the same date.

# Space Complexity:

The space complexity of the code is primarily determined by the storage of events in the events map. It requires O(n) space, where n is the number of events.  
The undoStack uses O(u) space, where u is the number of performed operations that can be undone.   
The waitingList queue requires O(m) space, where m is the number of events in the waiting list.   
The Event struct stores the details of each event and requires additional space for strings and the list of attendees.   
Overall, the space complexity of the code is determined by the number of events, performed operations, and events in the waiting list.   
  
In summary, the time complexity of the code varies depending on the specific operation, while the space complexity depends on the number of events and operations performed.

# Outputs:

# 

# 

# Chapter 6 Discussion

# The implemented hybrid data structure in the Event Management System demonstrates practicality and effectiveness in managing events. By utilizing a combination of an unordered map, stack, and queue, the system efficiently handles event operations, maintains a waiting list, and provides the ability to undo previous actions.

# The use of an unordered map (events) allows for fast retrieval and modification of events using the event name as a key. This data structure provides an average constant-time complexity for operations such as adding, removing, and finding events. It ensures efficient event management, particularly when the number of events increases.

# The stack (undoStack) plays a crucial role in enabling undo functionality. It stores the previous events, allowing the system to revert the last operation with a constant-time complexity. This feature enhances the user experience by providing a safety net for accidental or incorrect operations.

# The queue (waitingList) is utilized to handle date conflicts between events. If a new event conflicts with an existing event, it is added to the waiting list. The waiting list maintains the order of events based on their arrival time, ensuring fairness when allocating slots to conflicting events. The use of a queue data structure facilitates a first-in, first-out (FIFO) order, which is suitable for managing events on a waiting list.

# However, the implemented hybrid data structure does have limitations. As the number of events or waiting list entries grows significantly, the performance may degrade due to linear search operations. For example, when removing an event, if there is a waiting event on the same date, the code iterates through the waiting list to find the corresponding event. This can result in increased time complexity as the number of waiting events grows.

# Additionally, the current implementation lacks advanced features such as sorting events by date or time, searching events by date range, or handling recurring events. These limitations could impact the scalability and flexibility of the Event Management System for more complex event management scenarios.

**Conclusion:**

In conclusion, the implemented hybrid data structure effectively manages events within the Event Management System. The combination of an unordered map, stack, and queue allows for efficient event handling, maintains a waiting list, and provides undo functionality. The system demonstrates practicality and usefulness in organizing events, tracking attendees, and displaying event details. While the implemented hybrid data structure showcases its effectiveness in managing events, there are certain limitations to consider. The performance may degrade when dealing with a large number of events or waiting list entries, and the system lacks certain advanced features for complex event management sce

# Chapter 7 References

## Books:

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. , Introduction to Algorithms.
2. Narsimha Karumanchi, Data Structures and Algorithms Made Easy

## Websites:

1. N.E.Goller, “Hybrid Data Structures Defined by Indirection” [Online]

Available URL: https://academic.oup.com/comjnl/article/28/1/44/468048

1. ] “ Big O Analysis and Hybrid Data Structures” [ Online]. Available: URL: https://docs.onenetwork.com/NeoHelp/devnet/big-o-analysis- and-hybrid-data-structures-79141067.html
2. Dario Radecic, “Data Structures and Algorithms With Python” *IEEE Explore*. [Online] Available URL: https://towardsdatascience.com/data- structures-and-algorithms-with-python-learn-stacks-queues-and-deques-in-10- minutes-e7c6a2a1c5d5
3. JavaTpoint, “Data Structures and Algorithms in Python” . [Online] Available URL: https://[www.javatpoint.com/data-structures-and-algorithms-in-](http://www.javatpoint.com/data-structures-and-algorithms-in-) python-set-1