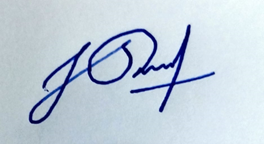
**Student Portfolio**

|  |  |
| --- | --- |
| **Jeevan Sarasram S S** | |
|  | **Register Number: RA2311056010035**  **Mail ID: js9882@srmist.edu.in**  **Department: B.tech CSE-DS**  **Yea** **r / Sem/ Section: 2nd Year/3rd Sem/AL-1** |
| **Subject Title: 21CSC201J Data Structures and Algorithms**  **Handled By: Dr. K.Rajkumar** | |
| **Email: jeevansiva2005@gmail.com**  **LinkedIn: linkedin.com/in/jeevansarasramss/**  **GitHub: github.com/JeevanSarasramSS** | |
| **ELab Completion Status** | |
| **Lab Experiment Completion status** | |
| **REAL WORLD APPLICATION  CRICKET PLAYER RECORDS USING BINARY TREES**  **Overview:** In cricket player management, maintaining player records efficiently is crucial for analyzing performance and retrieving player information quickly. Using a binary tree structure to store player records allows for efficient searching, addition, and deletion of records based on unique Player IDs. This structure is beneficial in scenarios requiring fast access and updates to large sets of player data.  Binary Tree Representation   * Elements: Each node in the binary tree represents a player record. * Attributes: Each record includes Player ID, Name, Matches Played, Runs Scored, and Batting Average.   **Operations:**   * Insert: Adds a new player record to the binary tree, organizing records based on Player ID. * Search: Locates a player’s record using the Player ID, allowing quick retrieval of player data. * Delete: Removes a player’s record based on Player ID, updating the tree to maintain structure. * In-Order Traversal: Displays all player records in sorted order by Player ID, useful for reviewing the entire player database.   **Algorithm Implementation:**   * Binary Search Property: Each node’s left child is less than the node, and the right child is greater, ensuring quick search and access. * Self-Balancing Options: If the number of records becomes large, a self-balancing binary tree (e.g., AVL or Red-Black Tree) can be used to maintain optimal efficiency. * Dynamic Record Management: Supports real-time addition, deletion, and updating of player records to ensure the database is accurate and current.   **Key Components**   * Node-Based Binary Tree:   + Linked Node Structure: A flexible linked structure that accommodates a variable number of player records, dynamically growing as new players are added.   + Binary Search Structure: Efficient for storing and retrieving player information, especially when records are indexed by Player ID. * Traversal Techniques:   + In-Order Traversal: Used for listing player records in ascending order of Player ID.   + Pre-Order and Post-Order Traversals: Can be implemented if specific applications require these traversal types for player record analysis.   **Balancing Techniques**   * Unbalanced Tree (Basic Structure): Simple to implement for smaller datasets but may degrade with a larger number of records. * Self-Balancing Tree (Advanced): Ideal for large datasets, ensuring operations remain efficient even as the number of records grows.   **Dynamic Updates** Supports real-time changes such as updating player statistics, adding new players, and removing players who are no longer active, keeping the player records current.  Skills Demonstrated   * Algorithm Design: Efficiently organizing player records and implementing optimized tree operations for quick access and updates. * Data Structure Analysis: Choosing the right tree structure to handle a growing dataset of player records efficiently. * Programming Proficiency: Advanced tree manipulation for inserting, deleting, and updating records in real time. * User Interface Integration: Developing user-friendly interfaces for inputting, displaying, and managing cricket player records effectively. | |
| **CERTIFICATIONS** | |
|  | |
| **CODING COMPETITIONS** | |

**  
Signature of the Student**