**Department of Computer Science**

**B. Tech. (CS)/ Sem-III (2024-2025)**

**Subject : Data Structures and Applications**

**Pedagogy :** Role Playing

**Evaluation:** As Group Assignment

 The Role-playing scenario to represent a **B** using volunteer students as nodes. Here’s the updated scenario:

**Student Record Management System**

**Objective:** Use a binary search tree to manage student records.

1. **Introduction:**
   * **Setting:** Imagine our classroom is now a university’s student record office, and we need to keep all student details in order. We will use a binary search tree to organize these records like a neat filing system.
   * **Props:** We will use the whiteboard and markers to draw a tree diagram, where each node represents a student’s record, and the branches show how they are connected.
   * **Roles:**
     + **Team Leader:** Supervises the project, explains how the tree organizes student records, and guides the team members with their tasks.
     + **Developer 1 (addStudent):** Adds new student records into the tree, making sure they go in the right place based on a unique ID.
     + **Developer 2 (removeStudent):** Removes student records when needed, making sure the tree stays organized after a record is deleted.
     + **Developer 3 (searchStudent):** Finds a student’s record in the tree when we need specific information, using the unique ID to locate them quickly.
2. **Role Play Script**

**Scene:** A group of developers is working on a project to create a Student Record Management System using a Binary Search Tree (BST). Each developer is responsible for implementing a specific feature. The Team Leader (TL) is coordinating the process.

**Team Leader (TL):** "Alright team, we need to get this Student Record Management System running. The first task is to insert student records into our binary search tree. "

**Developer 1 (D1):** "I've got this! Let me handle the insertion. I’ll start by adding the first student to the tree."

* *D1 inserts the first student into the binary search tree.*

**TL:** "Nice! Now, let's add another student."

**D1:** "Got it! Let me add the second student to the tree."

* *D1 inserts the second student into the binary search tree.*

**TL:** "Great work! Let's do it again for the next student."

**D1:** "No problem, I’ll add the third student now."

* *D1 inserts the third student into the binary search tree.*

**TL:** "Nice! Once more, Developer 1."

**D1:** "On it! Adding the fourth student to the tree."

* *D1 inserts the fourth student into the binary search tree.*

**TL:** "You're doing great! One last time for now."

**D1:** "I’m adding the fifth student."

* *D1 inserts the fifth student into the binary search tree.*

**TL:** "Fantastic work! Let’s add one more student for now."

**D1:** "I’m on it! Adding the sixth student to the tree."

* *D1 inserts the sixth student into the binary search tree.*

**Algorithm 2: Searching for a Student Record**

**TL:** "Okay team, I want to find information about a specific student. Can someone help me with this?"

**Developer 2 (D2):** "I'll take that! Just give me the student’s unique ID, and I’ll search for it."

* *TL provides the student ID. D2 searches for the student in the BST.*

**D2:** "Here we go, I found that student’s data!"

**TL:** "Great job! Now the system can search and return student records by ID."

**Algorithm 3: Deleting a Student Record**

**TL:** "Some student left our college so I wanted to deleted their student data. Who can help me in it?”

**Developer 3 (D3):** "I’ll do it!"

**D3:** "Since this student has no child nodes, we can safely remove it from the tree."

* *D3 deletes the student with no child nodes from the binary search tree.*

**TL:** "Great! Now, can you delete one more student’s data."

**D3:** "I’m on it!."

**D3:** "Since this student has one child, we’ll remove it and connect its parent to its child."

* *D3 deletes the student with one child from the binary search tree.*

**TL:** "Awesome! Now, can you remove one more student’s data for final time."

**D3:** "Got it! Let’s delete the student data."

**D3:** "Since the student has two children, I’ll replace it with the minimum id present in it."

* *D3 deletes the student with two child nodes and replaces it appropriately in the binary search tree.*

**Algorithm 2: Searching for a Student Record (Attempt 2)**

**TL:** "Okay team, I need to find another student's information. Can you search for it again?"

**D2:** "Sure, just give me the student’s ID."

* *TL provides the student ID. D2 searches the BST.*

**D2:** "Sorry, it looks like this student’s data is not present in the system."

**TL:** "That’s alright, we’ll double-check later."

---End Scene---

**Conclusion:**

In summary, the team worked well together to create a Student Record Management System using a Binary Search Tree. Developer 1 took care of adding student records, Developer 2 made it possible to search for students, and Developer 3 managed the deletion of records in different situations. The Team Leader praised their teamwork and stressed the importance of testing the system and adding more features. The team celebrated their progress and discussed their next steps, focusing on making the system even better.

**Algorithms to be used**

**Various Operations:**

#1. Student Struct

struct Student {

int id;

char name[100];

float cgpa;

};

#2. Node Struct

struct Node {

Student data;

Node\* left;

Node\* right;

};

**# 1. Create New Node**

createNode(data):

node = new Node()

node.value = data

node.left = NULL

node.right = NULL

return node

**# 2. Insertion**

insertNode(root,data):

if (root==nullptr):

root=createNode(data)

else if(data.id<root.data.id):

root.left=insertNode(root.left,data)

else:

root.right=insertNode(root.right,data)

return root

**# 3. Search**

searchNode(root, id):

if (root == nullptr or root->data.id == id):

return root

else if (id < root.(data.id)):

return searchNode(root.left, id)

else:

return searchNode(root.right, id)

**# 4. Minimum value**

MinVal(root):

current = root

while (current && current.left != nullptr):

current = current.left;

return current

**# 5. Deletion**

DeleteFromBST(root, id):

if (root == nullptr):

return root

if (id < root.(data.id)):

root.left = DeleteFromBST(root.left, id)

else if (id > root->data.id) :

root.right = DeleteFromBST(root.right, id)

else :

// Case 1: No child

if (root.left == nullptr and root.right == nullptr):

delete root

return nullptr

// Case 2: One child (left or right)

if (root.left != nullptr and root.right == nullptr):

temp = root.left

delete root

return temp

if (root.left == nullptr and root.right != nullptr):

temp = root.right

delete root

return temp

// Case 3: Two children

if (root.left != nullptr and root.right != nullptr):

temp = MinVal(root.right)

root.data = temp.data

root.right = DeleteFromBST(root.right, temp.(data.id))

return root

**# 6. Display**

displayNodes(root):

if (root != nullptr):

displayNodes(root.left)

print root.(data.id)

print root.(data.name)

print root.(data.cgpa)

displayNodes(root->right);

**# Main Function**

root = nullptr;

// Inserting student records

Student student1 = {1, "Alice", 3.5};

Student student2 = {2, "Bob", 3.8};

Student student3 = {3, "Charlie", 3.2};

Student student4 = {4, "David", 3.9};

root = insertNode(root, student1);

root = insertNode(root, student2);

root = insertNode(root, student3);

root = insertNode(root, student4);

// Displaying all student records

cout << "Student Records:" << endl;

displayNodes(root);

// Searching for a student record by ID

int idToSearch = 2;

Node\* foundNode = searchNode(root, idToSearch);

if (foundNode != nullptr)

cout << "Found student record: \nID: " << foundNode->data.id << endl;

cout << "Name: " << foundNode->data.name << endl;

cout << "GPA: " << foundNode->data.cgpa << endl;

else

cout << "Student record not found." << endl;

// Deleting a student record by ID

int idToDelete = 3;

root = DeleteFromBST(root, idToDelete);

cout << "Student record deleted (if it existed)." << endl;

// Displaying all student records after deletion

cout << "Student Records after deletion:" << endl;

displayNodes(root);

**:Real life Examples:**

**1) Storing Student Records by Roll Number or Student ID :**

**Each node in the Binary Search Tree can represent a student, with the roll number or student ID. This allows for quick searches, insertions, and deletions.**

**2)Handling Admission and Deletion of Student Records :**

**When new students are admitted or existing students leave the system, their records need to be added or deleted. Binary Search Tree make these operations efficient, as insertion and deletion can be done.**

**3)Managing Course Enrollment :**

**Binary Search Tree can be used to manage course enrollments by storing student records based on course ID or course name. This enables fast access to students enrolled in a particular course and makes adding or removing students efficient.**

**4) Sorting and Ranking Students :**

**If students need to be ranked based on academic performance, attendance, or extracurricular involvement, a Binary Search Tree can help by storing students based on these criteria.**

**5) Fee Payment Records :**

**A Binary Search Tree can store students based on their fee payment status, in the form of date or payment ID. This allows the administration to quickly check who has paid or who is pending.**

**Why to use Binary Search Tree:**

**1) Ordered Structure -**

**Sorted Data: BST inherently maintains an ordered structure, where the left subtree contains smaller values and the right subtree contains larger values. This makes it easy to retrieve data in sorted order by performing an in-order traversal.**

**2) Efficient Insertion and Deletion -**

**Dynamic Structure: Unlike arrays, which require shifting elements when inserting or deleting, a BST allows efficient insertion and deletion without needing to reorganize the entire data structure.**

**3) Memory Efficiency -**

**Dynamic Memory Usage: Unlike arrays, which allocate memory for all elements upfront, BSTs dynamically allocate memory as needed for new nodes. This makes them more memory-efficient, especially when the number of elements is not known in advance.**

**4) Easy Modification and Extension -**

**Customizable Operations: BSTs are easily customizable. You can modify the basic structure to add more features (e.g., balancing algorithms, duplicate key handling, etc.) or optimize them for specific use cases.**

**5) Faster Minimum/Maximum Search -**

**Direct Access to Extremes: In a BST, the minimum value is always located at the leftmost node, and the maximum value is at the rightmost node, enabling constant time access to these elements.**

**Gen AI Queries Used to generate the Script:**

**Write down the required GEN AI Queries Used here…**

**Rubric for Evaluation**

* + - 1. Two marks for the query fired on Gen-AI tool for preparing the script.
      2. Three marks for the Effectiveness of the script for the chosen topic. (i,e, : Documentation Prepared by the Teams and its readiness to deliver/present the scenario. )
      3. Five Marks for the Actual Role Playing, which includes
         1. Three marks for Team Work
         2. Two marks for individual contribution

• Generate a script which can be performed live on stage to make audience understand insertion in binary search tree of Student Record Management System. The contact node will be demonstrated by people and the narrator should use terms from algorithm of binary search tree.

• Generate a script which can be performed live on stage to make audience understand deletion in binary search tree of Student Record Management System. The contact node will be demonstrated by people and the narrator should use terms from algorithm of binary search tree.

• Generate a script which can be performed live on stage to make audience understand searching in binary search tree of Student Record Management System. The contact node will be demonstrated by people and the narrator should use terms from algorithm of binary search tree.

**Students Attendance**

**B. Tech. (CS)/ Sem-III (2024-2025)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **SAP ID** | **Roll No** | **Student Name** | **Student Sign** |
| 1. | 70552300059 | E257 | Manas Borse |  |
| 2. | 70552300069 | E267 | Vyom Adhyaru |  |
| 3. | 70552300076 | E274 | Yanshu Baria |  |
| 4. | 70552300077 | E275 | Ankit Jangid |  |

**Students Evaluation**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **Roll No** | **Gen-AI Query**  **(02)** | **Script**  **(03)** | **Role Play**  **(05)** | | **Total**  **(10)** |
|  |  |  |  | **Team Work (03)** | **Individual Contribution (02)** |  |
| 1. | E257 |  |  |  |  |  |
| 2. | E267 |  |  |  |  |  |
| 3. | E274 |  |  |  |  |  |
| 4. | E275 |  |  |  |  |  |

**Name & Signature of the Faculty: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**