Searching catalogues in library using binary search tree ➢

● Brief Description:

My main objective in this project is to create a library management system wherein students can issue books and the admin or librarian can update/delete the record of books kept in the library. So we have the system into two parts : from user’s perspective and from admin’s perspective. First of all, the admin must login to handle the accounts where the username and password are already set. After he has logged in successfully, he can add, delete and update the books. He can add any new book in the already existing list of books. Similarly he can also delete any existing book. In the update option, the admin can update the quantity of books as well as the name of the book. As and when the admin adds the books, a binary search tree will be created where the nodes contain the name of books and are put in sorted order. Now if a student wants to issue/return any book, then he/she must login into the system, by entering their valid university ID. The student will be allowed to issue/return only if their ID matches the list of university ID’s of students. When the student enters the name of book to be issued, that particular book will be searched by it’s name, in the already created binary search tree. If the book is not found in the tree, then a message will be printed “Book is not available in the library”. And if the book is out of stock, then this message will be printed, “This book is currently unavailable. Please try after some days.” Moreover, the student cannot issue more than 2 books simultaneously. When the student issues a book, the issuing date and time is recorded by the librarian. And if the student misses the due date of returning the book, then he has to pay that particular fine.

● List of Data Structures used with logic design

1) Binary Search Tree: Binary search is used to search for a particular subject where the searching operation performs an in-order traversal on the created binary search tree to get the elements in sorted order, where the binary search tree is created using Linked List. ➢ Doubly linked list

2) Hash-map: Generally Hash-map, maps keys to values such that no duplicate keys are generated. So, in our project, unique keys are generated using hash map and it is assigned to every book such that using this keys we can store and fetch records of books in 2-D array.

3) 2-D array Here, we have used 2-D array to store the record of each book where rows indicate unique value for particular book which hash-map returns and columns indicate total quantity of books and available quantity of books.

• Operations to be perform on each data structure:

Insertion: We are using insertion operation to add books in the binary search tree. And the tree will contain the name of the added books.

Deletion: We use deletion operation to delete the node of that particular book from the binary search tree. After the node is deleted, the remaining elements are again rearranged in the tree.

Updation : The librarian can update the quantity of already existing set of books which is stored in the 2D array.

List or print all the values: There is also an option where all the details of the books are printed. Name of the book, available quantity of the books that can be issued and total number of books that library has.

Print book in-order: Prints the contents of the binary search tree i.e. the names of the books in ascending order as the tree is balanced.

Requirements of processing on data structures

Keeping elements in a specific order (ascending order) in Binary Search tree is balanced.

• List of programs Filename:

Insertion:

```
 void insert(String key)

{ root = insertRec(root, key);

 }

Node insertRec(Node root, String key) {

 if(root == null) { root = new Node(key); return root;

} //If book name < root then place it as left child

 if (key.compareTo(root.key) root then place it as Right child

else if (key.compareTo(root.key)>0) root.right = insertRec(root.right, key); return root;

}
```

Deletion:

```
void deleteKey(String key) {

root = deleteRec(root, key);

 } Node deleteRec(Node root, String key) {

if (root == null) return root; //If book name < root then search it at left side and delete

if (key.compareTo(root.key) root then search it at right side and delete

else if (key.compareTo(root.key)< space ; i++) System.out.print( " ");

System.out.print("[" +t.key+ "]"); return printTreeRec(t.left ,space);

 }
```

Print in-order:

```
 void printInorder(Node node) { if (node == null) return;
 printInorder(node.left); System.out.print(node.key + " ");
printInorder(node.right);
 }
void printInorder() { printInorder(root);
 }
void inorder() {
inorderRec(root);
 }
void inorderRec(Node root) {
if (root != null)
 { inorderRec(root.left);
 System.out.println(root.key);
inorderRec(root.right);
 }
}
```

Searching the book:

```
 public boolean containsNode(String value) {
return containsNodeRecursive(root, value);
 }
private boolean containsNodeRecursive(Node current, String key) {
 if (current == null) { return false; } //If book name < root then place it as left child
if (key.equalsIgnoreCase(current.key)) { return true;
 } //If book name < root then search at left side of root else right side return
key.compareTo(current.key)
```