Jeff Morgan

John Widner

Murad Isgandarli

Project 2 Report

When planning for this project we decided on a binary search tree to hold the books early, because we had just covered it in class. For holding the customer’s ratings of each book, we considered a few different data structures. Linked lists and binary trees where both discussed but ended on maps due to the ease of search and storage.

Assumptions:

When creating a weight for each customer, the books that was rated in common with mine will be compared. For example, if the rates are same, we will add 1 to a temporary stack “a”, if customers rate is off by 1 point we will add 0.8 to the stack. If it is off by 2 points we will add 0.6 to the stack… and continue doing that for all books in common. When done, values in stack “a” will be added and divided by the number of books in common.

Big O notation:

In the Binary\_Search\_Tree<Item\_Type> class there are three main functions used, the find and the two insert functions.

The find function find(BTNode<Item\_Type>\* local\_root, int& target), which receives a pointer to a BTNode that is the root of the tree you wish to search and a reference to an integer, the IBSN number as arguments. It then recursively searches the binary search tree by the ISBN number and returns the item if found. If the item is not found in the tree, it returns an error. The find function has an average performance of O(log(n)) and a worst case of O(n).

The insert function insertByISBN(BTNode<Book>\*& local\_root, Book& item) receives a pointer to the address of the root node of the tree and the reference to the book item you are trying to insert as arguments. The function recursively traverses the tree checking the ISBN numbers of each node against the ISBN number of the book you are insert until it finds the correct place to insert. It returns a Boolean value to indicate whether the book was inserted or not. The insertByISBN function has an average performance of O(log(n)) and a worst case of O(n).

The insertByTitle(BTNode<Book>\*& local\_root, Book& item) receives a pointer to the address of the root node of the tree that you are trying to insert the book into and a reference to a book item that you wish to insert as arguments. It recursively searches the tree in order and checks the title of the node against the title of the book that you are trying to insert. It returns a Boolean value to indicate whether the book was inserted or not. The insertByTitle function has an average performance of O(log(n)) and a worst case of O(n).

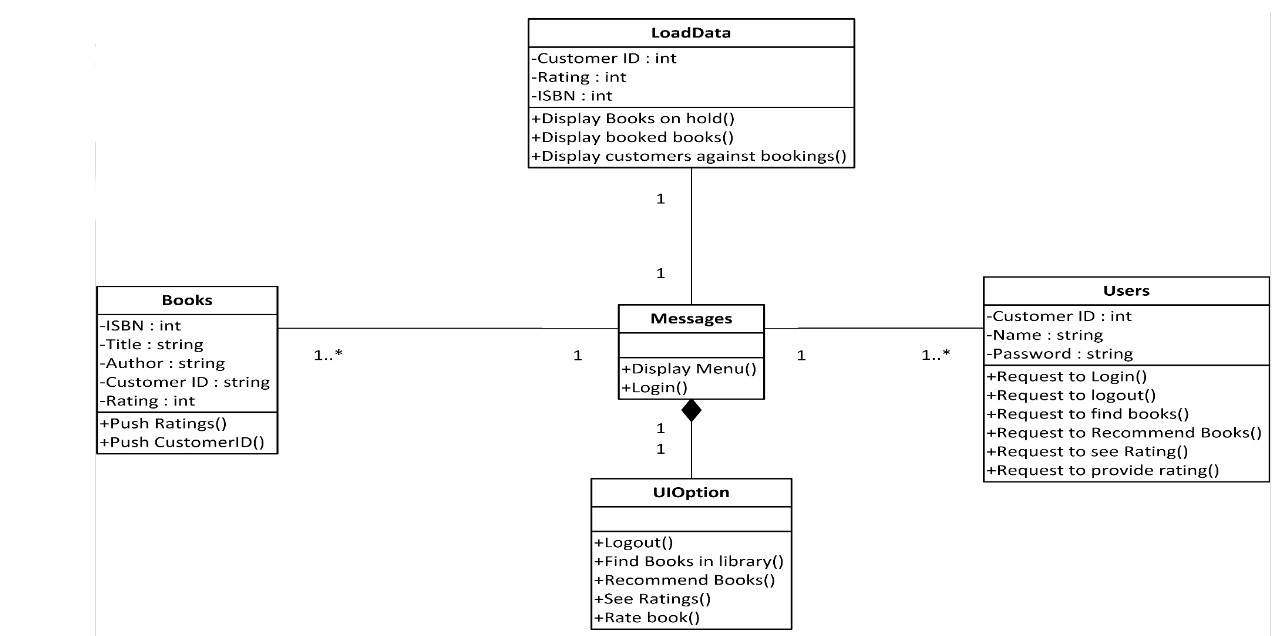
In the Customer class there are also three main functions used, the rateBook, the booksRated, and convertList.

The rateBook(Binary\_Search\_Tree<Book>& BookTree, int IBSN, int theRating) receives a reference to the Book Tree, the ISBN number of the book you are going to rate, and the rating that the book will be given as arguments. The rateBook function the uses the find function to locate the book to be rated and sets the rating. If the book has already been rated you have the option of changing the rating or canceling the process. The rateBook function has an average performance of O(log(n)) and a worst case of O(n).

The booksRated(vector<Book> theBookList) function is passed a vector to hold books as an argument. It iterates through both the book list and the map of the ratings and adds each rating into the vector. It has an average performance of O(1) and a worst case of O(n).

The convertList(vector<Book>& bookTitle) function receives the reference to a vector of book as an argument. It creates a list of the titles of the books that have been rated to output them to the customer. The convertList function has an average performance of O(1) and worst case of O(n).

The recommend function, RecommendBooks(vector<Customer>& customers, int& custID) receives a reference to a vector of customers and the customers ID number as arguments. The function searches for a customer who has similar books in their ratings vector. Once a customer is found with the most books in common with the user, suggestions are made by returning books that the customer has read but the user has not. The RecommendBooks function has an average performance of O(n2).

UML Diagram:

