|  |
| --- |
| C:\Users\mabarkat\Documents\KSU_LOGO.BMPKing Saud University  College of Computer and Information Sciences  Department of Computer Science  CSC 212 Data Structures Project Report – 1st Semester 2016-2017  Developing a Ratings Query Application |

**Authors**

|  |  |  |
| --- | --- | --- |
| Name | ID | Section |
| Abdurrahmane Rikli | **435107582** | **21889** |
| Abdulaziz Mogrin al Mogren | **435103275** | **21897** |
| Khalid al Ahaidib | **435101351** | **21889** |

# Introduction

This project helped us in understanding the many ways to use ADTs. Specifically linked lists and binary search trees. The idea was to store and manipulate sets of data (ratings) in the most efficient way possible.

This report will contain the specifications of the methods used in this project, the general idea of the design, and the implementation of the methods.

# Specification

***Phase 1***

**Class: item**

Elements  
‘data’ of type LinkedList of type Rating.  
‘Id’ of type int.  
‘next’ of type item.

Structure  
The class is practically a Node with an extra piece of data.

Operations  
getters only.

**Class: user**

Elements  
‘data’ of type LinkedList of type Rating.  
‘Id’ of type int.  
‘next’ of type user.

Structure  
The class is practically a Node with an extra piece of data.

Operations  
getters only.

**Class: RatingManager**“Description of extra methods”

Operations  
Method itemExists(int itemId)  
Requires: LinkedList Item is not empty.  
Input: itemId.  
Results: Sets the current on the item’s position if it exists. Otherwise, sets it at last.  
Output: Returns whether the item exists in the Item LinkedList.

Method userExists(int userId)  
Requires: LinkedList User is not empty.  
Input: userId.  
Results: Sets the current on the user’s position if he exists. Otherwise, sets it at last.  
Output: Returns whether the user exists in the User LinkedList.

***Phase 2***

“No new classes or methods were added”.

# Design

[Write a detailed description of how the project was designed. The description should be readable by less technical people, and at the same time helpful for more technical people. Use tables and diagrams if needed]

***Phase 1***

The idea of the project is to attach to a ‘user’ every ‘rating’ they made on an ‘item’, and then link all users in a ‘linked list’. Vice Versa was done to ‘items’.  
Whenever a rating is added, the program first checks if the ‘user’ who made the rating and/or the ‘item’ he rated was inserted previously in the ‘User’ and ‘Item’ linked lists respectively. If the ‘user’ didn’t exist previously, the program creates a ‘user’ object for him, adds the ‘rating’ to it, and adds him to the ‘User’ linked list. If the ‘item’ didn’t exist previously, the program creates an ‘item’ object for it, adds the ‘rating’ to it, and adds it to the ‘Item’ linked list. Either, neither, or both operations might be run. If the ‘item’/ ‘user’ was already inserted in either, the program would insert the rating into the ‘item’ / ‘user’ ‘s linked list.  
The program can also compute the average rating given to an ‘item’ / by a user. It does so by finding the appropriate object on the ‘Item’ / ‘User’ linked list, copying all the ratings made on it/by him, and calculating their average values.  
Finally, the program can find the highest rated ‘item’ by comparing between the average rating of every ‘item’ on the ‘Item’ linked list.

***Phase 2***

The phase 2 program functions much the same way as the phase 1 program except for two main differences: The ‘ratings’ are stored on a binary search tree, and the program does not use ‘item’ or ‘user’ objects, as the id of the user/item is stored as a key, and the related ratings are stored in its data as a binary search tree.  
This program also sports some extra features:  
- It counts the number of nodes traversed to reach a specified item rated by a specified user.  
- It estimates the rating a user may place on an item by calculating the average rating placed on that item by users who share similar reviews to items rated by said user. If no users rated items similar to the specified user, the program instead computes the average rating given to the specified item by all users.

# Implementation

[Write a detailed description of the technical implementation details. The description should cover only the critical implementation parts that need to be clarified. Use code snippets if needed]

***Phase 1***

**addRating(Rating rating)**The method checks if the ‘Item’ and the ‘User’ linked lists are empty or do not contain the ‘rating’s’ relevant objects (userId, itemId). If so, the program will create new ‘item’ and ‘user’ objects and insert them into their linked lists.  
If only one of them is empty or does not contain the ‘rating’s’ relevant object, then the program creates and inserts an appropriate object into the appropriate list, and just inserts the ‘rating’ into the appropriate object on the other linked list.  
If neither are empty and both contain the ‘rating’s’ relevant objects, then the program just inserts them into the linked lists’ appropriate objects.

**getHighestRatedItems()**The program traverses the ‘Item’ linked list, computing the average rating of the item, and comparing between each to find the one with the highest average rating. The method then re-traverses the ‘Item’ linked list, adding each rating with an average value similar to the highest average rating and placing its ‘itemId’ into a linked list. The method then returns the list.

***Phase 2***

**getDist(int ui, int uj)**The method checks if the ‘User’ BST contains both of the given users, and returns POSITIVE\_INFINITY if it does not.  
It then extracts one user’s ratings as a linked list, and the other’s as a BST. It then checks each item a rating was placed on in the linked list, checks if that same item is present in the BST, and if it is, it adds the distance between the ratings placed on that item, and adds it to a local variable. If the users share no common items, then the method returns POSITIVE\_INFINITY. Else, it returns the average distance.

**kNNUsers(int I, LinkedList<Integer> users, int k)**The method computes the length of the linked list “users” and creates a heap implemented as a priority queue of the same length. It then uses the method getDist(int, int) to compute their distance, checks if it is POSITIVE\_INFINITY, and adds them to the PQ with a priority of their distance \*-1 it they aren’t. It then transfers k or less of them to a linked list which it returns.

# Conclusion

[Conclude with summarizing all what have been done in the project and stating any limitations or future work]

The program stores records of ratings placed on items by users in two ways. The first is a linear, inefficient way (LinkedList). The second is a more, if not the most efficient way to store the records (BST).

The project has a few shortcomings, such as the creation of unnecessary classes ‘item’ and ‘user’ in the first phase, which should have been implemented as Nodes of type (class containing a linked list of ratings and an int value); and the getDist() method in the second phase, which should have returned POSITIVE\_INFINITY if either users made no ratings, but instead returned 0.

We are proud to have learned so much, and to have produced such work that just a few months ago, would have seemed impossible to us.

# Source Code

On webcat.