Recommendation System

This program focuses on using data structures in complex ways. Turn your solution to GradeScope. You will need several txt files containing ratings from repository for testing.

<https://github.com/upstate-csci-236-master/master-program-recommender/blob/main/README.md#background>Background:

If you've ever bought a book online, the bookseller's website has probably told you what other books you might like. This is handy for customers, but also very important for business. In 2010, online movie-rental company Netflix awarded one million dollars to the winners of the Netflix Prize (http://www.netflixprize.com/). The competition simply asked for an algorithm that would perform 10% better than their own algorithm. Making good predictions about people's preferences was that important to this company. It is also an important current area of research in machine learning, which is part of the area of computer science called artificial intelligence.

<https://github.com/upstate-csci-236-master/master-program-recommender/blob/main/README.md#what-you-will-do>What you will do:

So how might we write a program to make recommendations for books? Consider a user named Carlos. How is it that the program should predict books Carlos might like? The simplest approach would be to make almost the same prediction for every customer. In this case the program would simply calculate the average rating for all the books in the dataset and display the highest rated book. With this simple approach no information about Carlos is used.

We could make a better prediction about what Carlos might like by considering his actual ratings in the past and how these ratings compare to the ratings given by other customers. Consider how you decide on movie recommendations from friends. If a friend tells you about a number of movies that s(he) enjoyed and you also enjoyed them, then when your friend recommends another movie that you have never seen, you probably are willing to go see it. On the other hand, if you and a different friend always tend to disagree about movies, you are not likely to go to see a movie this friend recommends.

Your task for this project is to write a program that reads the name of a ratings file from the command line and takes people's book ratings and makes book recommendations to them using both techniques described above. We will refer to the people who use the program as "users".

<https://github.com/upstate-csci-236-master/master-program-recommender/blob/main/README.md#terminology>Terminology

program-user: the person running your program who is looking for book recommendations

recommendation-user: these come from the recommendation file and are compared against the requested-user

requested-user: recommendation-user that the program user is asking for their recommendations to be based upon

rating averages: the average recommendation for each book

books-list: list of the names of all the books known to your program, read in from a file, the name of the file comes from the command line

average-ratings: a list of tuples that contains the averages of the ratings from the top three recommendation for a each book

ratings-dictionary: a dictionary containing each recommendation-user and their list of ratings for each book

program-user commands: a list of commands the program must recognize and process; these are: averages, recommend, quit

ratings.dat: a large text file of book ratings used in testing

small_ratings.dat: a small text file of books ratings used in testing

<https://github.com/upstate-csci-236-master/master-program-recommender/blob/main/README.md#implementation-details>Implementation Details:

Your program will read user date from a file (name of this data file comes from command line) that contains sets of three lines. The first will contain a username, the second a book title and the third the rating that user gave the book. Examples attached to this assignment.

When your program starts it should read through the file and create a list with one occurrence of each book from the file. For example, the file ratings-small.txt might produce the following list:

['1984', 'Cats', 'Harry Potter', 'Animal Farm', 'Watership Down', 'Lord of the Rings']

Note that the order of the books does not matter. We suggest that

you create this list by putting all books in the file into a set and then convert that set to a vector. https://www.geeksforgeeks.org/convert-set-to-vector-in-cpp/

Once you have this vector, create a dictionary (map) to store the rating data and loop through the file again. This time add each person in the file as a key to the dictionary. The value that is associated with them should be a list the same length as the list of books you created in your first pass through the file. You should store the rating at the same index that book's name appears at in in the list of books. For example, when the first three lines of the file in the repository are read, we would add a mapping from the key Bob to a value of [1, 0, 0, 0, 0, 0]. Books that the user has not rated (or whose ratings we have not read yet, as in this case) should be represented by 0s. https://stackoverflow.com/questions/15912877/how-do-i-convert-values-from-a-vector-to-a-map-in-c

The dictionary created with the file ratings-small.txt in the repository would be as follows:

{ 'Kalid' : [0, 0, 3, -3, 5, 0], 'Carlos' : [-5, 0, 5, 1, 0, 0], 'Suelyn' : [5, 0, 1, -3, 0, 0], 'Bob' : [0, 5, -3, 0, 0, 5] }

Now your program is ready to make recommendations. It should then wait for the program-user to enter a command.

* recommend : recommend books for a particular recommendation-user
* averages : output the average ratings of all books in the system
quit : exit the program
* quit

DO NOT OUTPUT ANY PROMPTS.

The program should repeat processing commands until quit is entered.

Here is what each command should do:

<https://github.com/upstate-csci-236-master/master-program-recommender/blob/main/README.md#averages>averages

If the program-user selects the averages option the program should output all of the books in the file sorted by average rating from highest to lowest. For example, see the files in the repository.

We suggest that you figure this out by building up a list of tuples containing the average rating for a book first and the title of that book second. You can build up this list by going through the list of books one at a time and for each person in the dictionary adding up their rating of that book and counting how many people in the dictionary rated it something other than 0. The average score for that book is the sum scores divided by the count of non-zero ratings. Once you have created this list you can sort it by using the list's sort function.

Since the averages will stay the same throughout the run of the program you may want to calculate them once at the start of the program.

<https://github.com/upstate-csci-236-master/master-program-recommender/blob/main/README.md#recommendations>recommendations

When the program-user selects the recommend option the

program should read the requested-user that the program-user wants recommendations for. If the name of the requested-user isn't in the dictionary of ratings, the program should output the same list of books as when the program-user selects the averages option.

If the name the program-user inputs is in the dictionary of ratings you should use the data in the dictionary to find the other recommendation-users that are most similar to the requested-user you are looking for recommendations for.

<https://github.com/upstate-csci-236-master/master-program-recommender/blob/main/README.md#how-to-calculate-recommendations>How to calculate recommendations

The first step to do this is to calculate the similarities between the requested-user and the other recommendation-users. We will use the dot product between the users' lists of ratings to calculate their similarity. This means that we will multiply each element in your user's list with the element at the same index in the other user's list and sum the result. For example, if we were looking for a recommendation for Kalid we would do the following to calculate his similarity to Carlos:

(0 * -5) + (0 * 0) + (1 * 3) + (-3 * 1) + (3 * 0) + (0 * 0) = 0 + 0 + 3 + -3 + 0 + 0 = 0

Compute this similarity for each recommendation-user in the dictionary. Store tuples containing first the similarity number and second the name of the other recommendation-user in a list. You can use the list sort function to sort this list.

Note that the requested-user that you are looking for will always be in the dictionary. We are not interesting in how similar the

requested-user is to themselves. You may find it helpful not to add the user to the list or to remove them after sorting. Note that the requested-user will always be most similar to themselves.

Now that we have a list of the most similar recommendation-users, we can use this to figure out which books to recommend. To generate recommendations take an average of the ratings of the three recommendation-users with the highest similarity to the requested-user you are looking for.

To average the ratings create a new list the same length as the list of books and filled with 0s. Loop through this list. For every index of this list loop through the first three users, add up their ratings and then divide by the number of non-zero ratings. If there are no non-zero ratings for a book you should not divide as you will get a divide by 0 error.

Once you have calculated these averages, create a list of tuples that contain first the average rating and then the book title for all books that have non-zero ratings in the averages list. Then, sort this list. Now you have a list of books to recommend.

Note that it must be possible for the program-user to choose options multiple times in any order and get the correct results each time.

Your program should match the expected output files provided in the repository.

<https://github.com/upstate-csci-236-master/master-program-recommender/blob/main/README.md#development-strategy-and-hints>Development Strategy and Hints:

We suggest that you complete the assignment in the order

described above. You will find the list sort function very helpful for this assignment. The sort function sorts elements in a list from smallest to largest. If the list contains tuples it sorts by the first element in the tuple. Therefore, it is very important to order your tuple contents as described in the instructions above. If you have a variable called my_list, the following code will sort it: my_list.sort(). Note that sort alters the list, it doesn't return a new list.

Use print statements to view the state of your structures. Create small input files for yourself to help you debug.

<https://github.com/upstate-csci-236-master/master-program-recommender/blob/main/README.md#style-guidelines-and-grading>Style Guidelines and Grading:

Part of your grade will come from appropriately using data structures and following the algorithms described in this document.

Redundancy is always a major grading focus; avoid redundancy and repeated logic as much as possible in your code. Divide your code into a set of functions that captures the structure of the program.

Follow good general style guidelines such as: appropriately using control structures like loops and if/else statements; avoiding redundancy using techniques such as functions, loops, and if/else factoring; good variable names, and naming conventions; and not having any lines of code longer than 80 characters. You may have no global variables (except constants) and you may not nest functions inside other functions

Comment descriptively at the top of your program, each function,

and on complex sections of your code. Comments should explain each function's behavior, parameters and returns.

<https://github.com/upstate-csci-236-master/master-program-recommender/blob/main/README.md#submission>submission

Submit your code to GradeScope. No need to upload test files, they will be provided in GradeScope for you.