CSL 214 : Data Structures and Program Design II
H/W Assignment-2

1. Announcement Date: **17-03-2024**

2. Due date: Submit online by **8:00 AM on Monday 1st April, 2024**

3. The assignment can be done either individually or in a group of 2 students. In case of group of 2, both the partners in the group should belong to the same practical batch.

4. For this assignment, Tree data structure is required to be used. Hence, choose Trees (either AVL tree OR B-tree OR B+tree) for any storage requirement you may have. Use of file operations to store/retrieve persistent data, would be given more credits.

5. No Copying / sharing of code is acceptable. If any such case is identified, the original author(s) and person(s) who has copied, both will be penalised equally and zero marks will be awarded.

6. You need to submit your source files by attaching them to a mail and sending it on **dspd.assignment@gmail.com** by the common deadline. Please attach .c and/or .h and/or .txt files only. No .obj or.exe files should be attached. The subject should be marked as DSPD2-HW-Assignment-1: Your enrolment numbers.

7. The evaluation via will take place in the week of 1st April 2024 to 5th April 2024. The evaluation is not considered complete unless the viva of the project groups during lab hours takes place. All students are required to attend their lab hours without fail in the week of 1st April 2024.

Problem for R1 Batch (Enrolment Numbers BT22CSE001 to BT22CSE025):

Assume that there are four separate databases containing data for people in a state.

First database is Aadhar database, which contains names of people, their addresses and their corresponding Aadhar numbers. Second database is of PAN-cards that contains in every node, name of a person, address, his/her PAN number and his/her Aadhar number. Third database is bank-accounts, which contains name of a person, his/her PAN number and the account details (bank, account number, deposited amount etc). Fourth database is LPG-database which contains name of a person, his/her bank-account-number and whether the person has taken LPG subsidy or not (YES or NO). Assume that Aadhar numbers are unique as they are derived from biometric data of a person that is assumed to be unique. Take as input, above four databases which are already populated.

1. Print names, addresses and Aadhar numbers of all those people who have Aadhar numbers but no PAN numbers.
2. Print names, addresses and Aadhar numbers of all those people who have multiple PAN numbers. Print all the PAN numbers for each such person.
3. Print names, addresses and Aadhar numbers of all those people who have multiple bank accounts registered under multiple (more than 1) PAN numbers. (Note that registration of multiple bank accounts with the *same* PAN number is allowed.)
4. Print details (Aaadhar, PAN, all bank-account details) of a person who has availed LPG subsidy.
5. Take amount X as input. Print names, addresses and Aadhar numbers of all those people who have their total savings (in all bank accounts of theirs) being greater than amount X and they have also availed LPG subsidy (YES). This may include people having one or more PAN numbers.
6. Print inconsistent data, i.e. names, addresses and Aadhar numbers of all those people who have different names mentioned in either of their Aadhar number, PAN number, bank account or LPG connection.
7. Given two different bank account databases, merge these two databases into a single database.
8. Given two Aadhar numbers A1 and A2, print details of all Aadhar numbers in the range of A1 and A2.

For all of the four databases, you are free to choose specific type of tree, whether AVL tree, B tree or B+ tree. Make the decision appropriately that helps the most in the ease of operations and efficiency. You can add extra fields in the tree-node/database if you need.

For implementing this application, it is desirable to use file handling. You can generate a .txt file for data input. While closing the program store the data back from the data structure to the files.

Problem for R2 Batch (Enrolment Numbers BT22CSE026 to BT22CSE050):

Consider an in-air flight-management platform of an air-traffic control (ATC) system, which maintains a digital dashboard for the in-air flights (the flights that have been departed but not yet arrived). The digital dashboard contains a tree based database of in-air flights bucketed in 60 minutes intervals based on the expected time of arrival (ETA) of the flights. Let the current time be 2:00 am. Then, the flights whose ETAs are in between 2:00 am – 2:59 am are kept in the first bucket, the flights whose ETAs are in between 3:00 am – 3:59 am are kept in the second bucket, and so on. Some buckets may be empty if there is no flight to arrive in the time interval. The system maintains a database of active buckets (the buckets which have at least one flight) sorted by the ETA intervals. Every bucket maintains the bucket ID (an integer), the beginning of the ETA interval, the end of the ETA interval, and a database of flight-plans. A flight-plan contains the flight ID (an integer), the actual departure time, and the ETA. In a bucket, the flight-plans are maintained as a database using a tree sorted by the actual departure time of the flight.

(a) Define a C structure to represent a bucket for the digital dashboard. Assume that all the times (ETA, departure time) are maintained using a C structure type ***TIME***.

(b) Consider that a digital dashboard is given. Write a C function to insert a new flight-plan in the digital dashboard. Recall that inside a single bucket, the flight-plans are sorted by the actual departure time. On the other hand, the buckets are sorted by the ETA intervals. Assume that the new flight-plan belongs to an active bucket. Suppose also that the following two C functions are available: (i) ***int timedeff*(*TIME A, TIME B*)** which returns the difference *A-B* between ***TIME A*** and ***TIME B*** in minutes, and (ii) ***int maxtime*(*TIME A, TIME B*)** which returns 0, 1, -1 according as both the times are same, ***A*** precedes ***B***, and ***A*** succeeds ***B***, respectively.

(c) Write a C function to cancel a flight-plan in the digital dashboard for some emergency situations.

(d) Write a C function to show the status of a particular flight-plan.

(e) Write a C function to show the flight-plans in 1 Hr. time period from any given time. In other words input is the current time T (say 2:30 pm), and the buckets get re-arranged in the intervals of 1 hr, that is, first bucket now is for 2:30 pm to 3:30 pm, second bucket is for 3:30 pm to 4:30 pm etc.

(f) Take two time units as input. Display the flight plans which lie between the given time units. Hint : Use/implement range-search function.

For implementing this application, it is desirable to use file handling. You can generate a .txt file for data input. While closing the program store the data back from the data structure to the files.

Problem for R3 Batch (Enrolment Numbers BT22CSE051 to BT22CSE075):

Walmart Store System

Using a tree data structure, design and develop a system for Walmart Store. In Walmart Store, there are different aisles. In one aisle there are dairy products. In the other there are pulses and grains. In the third there are bath and cleaning products. In fourth aisle they have ready to eat food items. In 5th aisle there are vegetables. Maintain an array of pointers to tree nodes. For each aisle, there is a tree. The nodes in the tree are the items in that aisle with information like the item id, item_name, quantity, expiry_date, threshold quantity. Now there is another tree of bills. When a user comes to the Walmart store, he should be billed for whatever items he buys. The corresponding quantity of each item is to be decremented in the main array of tree pointers of items. The array of aisles should be sorted according to the aisle_number. Items are stored by item_id in each tree of the aisle. In short the item id is the key of the node and and , item_name, quantity, expiry_date, threshold quantity is the info at that node.

Implement the following functions :

- Add / update item – Adds or updates an item in the database.
- Delete an item – Delete the item
- Add aisle/update aisle – Adds or updates an aisle
- Delete aisle – Deletes aisle
- Check availability of particular item for particular quantity and within the expiry date.
    - Inputs – Item_id, quantity required for item_id and the expiry date
- Function to generate a message when any particular item goes beyond its defined threshold quantity, so that it can be planned to order it soon.
- Function to generate a message on a certain ready-to-eat item if its expiry is within one week.
- Merge two aisles in a single aisle. For example, Wallmart wishes to add over the counter medicine section to the store. So, they have less space and want to merge their two aisles, namely, dairy and vegetables.
- Write a function that takes item_id as input and provides a list of items which are more often bought with the given item_id.
- Search all items in a particular aisle in the range between two given item_ids and print their information.

File handling is recommended to be used. The input should be scanned from a file to build the database of aisles, and items in each aisle with their data like quantity, expiry date, threshold quantity etc.

Problem for R4 Batch (Enrolment Numbers BT22CSE076 to BT22CSE100):

Using tree data structure, design and develop an audiobook library system where a user database and audiobook database have to be maintained. Efficient management of audiobooks and user interactions, including listening progress, library management, and user preferences, should be the focus of this system. Quick access to audiobooks, effective tracking of users' listening progress, and support for user libraries and preferences should all be provided by the system.

Data Structures

- **User Tree**: Manage user profiles using the chosen tree data structure. Each user node should contain:
  - UserID (unique ID)
  - Name
  - Email
  - Preferences (genres, authors, narrators, etc.)
  - A reference to another tree for the user's audiobook library, sorted by AudiobookID.

- **Audiobook Tree**: Manage audiobooks using the chosen tree data structure. Each audiobook node includes:
  - AudiobookID (unique identifier)
  - Title
  - Author
  - Narrator
  - Duration
  - Genre
  - Sum_Rating
  - Count_Rating(Total count of rating done eg: 4 users)

  (Note: Sum_Rating and Count_Rating will further help to calculate the average rating of an audiobook)

Required Functions/Operations

The following functions/operations need to be implemented:

1. Add_audiobook() and Edit_audiobook():
   Input: Audiobook details including ID, title, author, narrator, duration, genre, and rating.
   And for edit: with Audiobook ID and the details that can be modified (e.g., title, author, duration).
   Output: Confirmation message indicating the audiobook has been successfully added or modified to the library.
   Note: Audiobooks will be inserted into the database sorted based on title. If titles are the same, sort them by author.
2. Delete_audiobook():
   Input: Criteria for identifying the audiobook to be deleted (e.g., title, author).
   Output: A message indicating the audiobook is deleted.

Note: If the audiobook does not exist, indicate the deletion operation failed.

3. Add_user_profile():
   Input: User details including name, email, and preferences.
   Output: A message indicating successful user account creation.
   Note: Email addresses should remain unique within the system.
4. Add to User_library():
   Input: User action (add/remove audiobooks to/from library), and audiobook details.
   Output: Updated user library with a success or failure message.
   Note: Each user's library should be a tree, allowing for efficient management of their audiobooks without duplicates.
5. Search_audiobook():
   Input: Search criteria (e.g., title, author, genre).
   Output: List of audiobooks matching the criteria.
   Note: Provide options for flexible search criteria.
6. Listening_progress():
   Input: User ID, audiobook ID, and new progress (timestamp or percentage).
   Output: A message showing the updated listening progress for the specified audiobook.
   Note: Ensure accurate tracking of listening progress for each user.
7. Display_user_libraries():
   Input: User ID and filter criteria (e.g., all audiobooks, by genre).
   Output: A list of audiobooks in the user's library matching the filter criteria.
   Note: Allow users to view their audiobook collections based on different filters.
   Note: Sort the report by listening time to highlight the most listened-to audiobooks.
8. Rate_Audiobook(): Allow users to rate an audiobook, contributing to its overall feedback score.
   Input parameters: User ID, Audiobook ID, Rating (assuming a scale of 1 to 5).
   Output: Confirmation message indicating the rating has been successfully recorded
9. Most_poplular_audiobook()
   Input parameters: audiobook-ID
   Output: Returns the title, author, and average rating of the most popular audiobook. If no audiobooks have been rated yet, it returns an appropriate message indicating no ratings are available to determine popularity.
10. Listening_history_report_user():
    Input parameters: UserID
    Output: A report detailing the audiobooks listened to, including titles, authors, and total listening time.
11. Merge_user_libraries():
    Input: UserId's for two user library to be merged.
    Output: A single-user library containing the union of audiobooks from both libraries, using title and author as the key, first use title for the same title use author.
12. Listening_history_report_audiobook():
    Input parameters: AudiobookID
    Output: A report detailing the AudiobookID, title, author, and total listening time.
13. Range_Search_Audiobook
    Search and print titles of audiobooks with ID which have audiobook IDs between audiobook id1 and audiobook id2.

For implementing this application, it is desirable to use file handling. You can generate a .txt file for data input. The file should contain data for 5 users and 10 audiobooks.

Problem for R5 Batch (Enrolment Numbers BT22CSE101 to BT22CSE129 + ex-students):

A step tracking application lets a user calculate daily steps, set goals for a week, create groups to achieve group goals, and gives rewards to individuals who complete their goals. It also maintains leader boards to encourage people to complete steps with great rewards for top 3 individuals. It also has a group leader board where groups are ranked according to steps completed by each group as a whole. We have to design this step tracking application using trees.

1. Generate a tree of individuals with following fields-
    a. ID (unique ID)
    b. Name
    c. Age
    d. Daily Step goal
    e. Array of weekly step count (7 days step count recorded)

    This tree should be sorted on the basis of ID.

2. Generate a tree of groups with following fields-
    a. Group-ID
    b. Group name
    c. Member IDs with pointers to individuals in individuals tree
    d. Weekly group goal

    This tree should be sorted on the basis of Group-ID. A group can contain maximum 5 individuals.

3. The application should have the following functionalities
    a. Add_Person: This function should add a new individual to the tree of individuals. The tree should remain sorted
    b. Create_group: This function should create a new group and be able to add existing individuals to it from the individual tree. If an individual already belongs to a group, he cannot be added to a new group.
    c. Get_top_3: this function should display the top 3 individuals from the individual tree who have completed their daily steps goals and achieved highest number of steps. The individuals who have not completed daily goals but have higher number of steps should be excluded.
    d. Check_group_achievement(Group-ID): This function should display whether the given group has completed its weekly group goal
    e. Generate_leader_board: This function should take the group tree, sort it in the order of highest number of steps (Descending) completed by each group such that the root of the tree has the leading group, and display the group name with number of steps based on rank using this sorted tree.
    f. Check_individual_rewards(ID): This function should display the rewards earned by the given individual if he is in the top 3 individuals. Rank 1 gets 100 points, rank 2 gets 75 points, and rank 3 gets 50 points

g. Delete_individual(ID): This function should delete an individual from the tree of individuals as well as remove him from his groups.
h. Delete_group(Group-ID): This function should delete a group but retain its individuals. The individuals are now available to be added to a different group
i. Merge_groups(Group_ID_1, Group_ID_2): Create a new group by merging two groups and set new goals. The original groups should be deleted. The group should be merged with group_ID_1 as new ID and group_ID_2 should be deleted.
j. Display_group_range_info(): This function should take input as two group ids and display information about members in the range of these groups as well as group goals and ranks.
k. Suggest_goal_update(): This function should suggest a daily goal update for an individual such that he/she can consistently appear in the top 3 individuals

For implementing this application, it is desirable to use file handling. You can generate a .txt file for data input. The file should contain data for 20 individuals. Generate 5 groups with varying number of members and some members who do not belong to any group.