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

1. Announcement Date: **01-04-2021**

2. Due date: Submit online by **11:59PM on Thursday 8th April, 2021**

3. The assignment has to be done individually.

4. For this assignment, ***either AVL-tree or B-tree or B+-tree*** are the preferred data structure. Hence, choose the tree (any type that you are comfortable with) for any storage requirement you may have, although slight preference will be given to B+tree/B-tree/AVL-tree in that order.

5. No Copying / sharing of code is allowed for this assignment. If any such case is identified, the original author and person who has copied, both will be penalised equally and zero marks will be awarded.

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-2: Your enrolment no.

Problem for R1 Batch(Enrolment Numbers BT19CSE001 to BT19CSE030):

Physical Environment Sensor Network

A sensor network consists of spatially dispersed and dedicated sensors for monitoring and recording physical conditions of environment like temperature, sound, humidity, wind, and so on for multiple stations (you can consider different areas in city). The data from all sensors is collected at central location. Each sensor records the corresponding data at continuous time interval daily. Each sensor is represented by sensor ID (integer), sensor type, data it senses, time interval during which it senses the conditions continuously.

Write function for the following assuming that sensor database is stored ***either in AVL tree or B-tree or B+ tree*** data structure.

1. Write a function **create_sensor_network(struct sensor_node* new_node)** which will formulate the above mentioned sensor network in software. Data fields for every sensor node in the linked list should have following attributes -

a. sensor_ID(integer)
b. sensor_type (char)
c. sensor_location(charater)(or sensor station)
d. duration(time interval ex. 5 min- it means that a sensor senses the temperature , humidity etc. after every 5 min.)

Central repository is the location where data from all sensors is collected and it should include following things

a. **sensor_ID**
b. **Date**
c. **Time**
d. **data(integer or float)**

Write a function **central_repository (struct record* new_record)** which will create a database of information collected from all sensors.

2. Implement a function **Install_new_Sensor()** to add sensor to the existing sensor network at specified station.
   Note: New sensor will be added only if the type of sensor to be added is not present at specified station.
3. Idle sensors are those which are not sending any information to central repository for more than 2months. Identify such idle sensors and remove them from database. (remove means permanently delete them from database)
4. **Retrieve_info**() functions retrieves the data for sensors specified by following conditions
   A. Depending on sensor type (retrieves till date data)
   B. Depending on specified date range (multiple dates) for specific sensor type
5. Write function **sensors_in_between()** to print information of all sensors located in between any stations specified by sensor_ID range. (EX. if sensor ID range is 100- 200. Expected output is the information of all sensors with ID>=100 and ID<=200)

6. Adapt the existing data structure for sensor type which records multiple quantities. Ex. **Air quality index (AQI)** sensor which records entities like PM10, PM 2.5, nitrogen dioxide, sulphur dioxide, carbon monoxide, ground level ozone etc. and tries to find out Air quality level and pollution level. AQI is measures based on the average quantity of a particular entity measured over a standard time interval. Standard time interval for measuring averages is different for different entities (24 hours for most of the entities, 8 hrs. For PM 2.5). There should be provision for storing standard time interval for each independent entity in existing data structure. Final AQI is the highest of the AQI values calculated separately for each entity. AQI value for finding health status is as follows

AQI	Status

1-50	Good
51-100	Satisfactory
101-200	Moderately polluted
201-300	Poor
301-400	May cause respiratory illness
401-500	Severe
501 onwards	Hazardous

a. Write a function to report or display the month during which maximum AQI is reported for all years for all stations.
b. Write a function to display the dates on which hazardous health status is recorded for all stations.

Problem for R2 Batch(Enrolment Numbers BT19CSE031 to BT19CSE060):

Flight Management System:

A Flight Records Management System can be implemented as a tree (***either AVL or B-tree or B+ tree***) of structures, where each node of the tree represents a flight record and each record has fields like flight name, flight id, flight capacity, flight arrival time, flight departure time, flight class (VIP, VVIP, public), and any other field you may want.

The combined pair of flight id and arrival time of the flight can be thought of as a key in the tree and represents a unique record in a tree. Also kept are 3 trees, one each for every class of the flight.

Write following functions and a program that lets user to use any of these operations.

-insert

- Inserts (if the entry is not present in the tree) or updates an element (if the entry is present in the tree)
- I/p parameters: *flight name, flight id, flight capacity, flight arrival time, flight departure time, flight class (VIP, VVIP, public)*.
- O/p – Entry should be added/updated appropriately. The return parameter can specify if the insert operation is successful or not.

-delete

- Deletes an element
- i/p parameters:
    - tree (treeof records from which the entry needs to be deleted)
    - *flight id* of the flight whose records need to be deleted.
- o/p parameters: To know the operation is successful or not you have to print the appropriate message.

-getNumFlights

- i/p parameters: tree (tree of records)
- o/p: number of flights in the tree.

-isEmpty

- i/p parameters: tree (tree of records)
- o/p: To know if the tree is empty or not

-isFull

- i/p parameters: tree (tree of records)
- o/p: To know if the tree is fully occupied or not

- *geFlightWithtLongeststay*

- i/p parameters: tree (tree of records)
- o/p : flight(s) with max stay time (departure time – arrival time)

- *getSortedOnArrivalTime*
- i/p parameters: tree (tree of records)
- o/p parameters : print sorted records based on arrival time

- *getSortedOnDepartureTime*
- i/p parameters: tree (tree of records)
- o/p parameters : print sorted records based on departure time

- *getSortedOnStayTime*
- i/p parameters: tree (tree of records)
- o/p parameters : print sorted records based on stay time (departure time – arrival time)

- *UpdateFlightStatus*
- i/p paraments: : tree (original tree of records), tree2 (each record containing flight id, flight name, status (DELAY, ON_TIME, CANCELLED, PROMOTED), delay time. • Description : If status is DELAY, then delay_time gives us the time by which the flight is delayed for arrival. The stay time (difference between arrival and departure times) for the

flight remains the same. If status is PROMOTED, it promotes the flight class, so the original flight class and promoted flight class to be provided. • o/p: Updates the original tree according to status. The Output must be sorted on key. Also updates the 3 trees maintained for each flight class.

- *list Unique*
  - i/p parameters – A tree of records which contains duplicate entries (with same flight name and flight id)
  - Output – Remove duplicate entries if present. Retain the occurrence with minimum delay, in case of duplicates.

- *searchFlight*
  - *i/p parameters* – A tree of records and the range of start & end time (24-hours scale) • *Output* – The flight details which have scheduled departure & arrival time withing the time range. The details should be printed according to flight class.

Problem for R3 Batch(Enrolment Numbers BT19CSE061 to BT19CSE090):

Library management System

In an institute library, the following information are stored corresponding to each book:

1. Name of the book
2. Subject
3. Authors (surname first) of the book
4. Accsession number (an accession number is a sequential number assigned to each record or item as it is added to a library collection or database and which indicates the chronological order of its acquisition)
5. New print or reprint with year of publish
6. Issued/available

All books are stored in the database made up of ***either AVL-tree, B-tree or B+-tree***. All books of the same subject, are to be sorted according to title. All books of same title are sorted according to names of authors (if at all exists). All copies of the same book are sorted according to year of print.

One *member database* is maintained with the following information:

1. Member name
2. Department
3. Status/designation (student/faculty)
4. Member ID
5. Number of books issued

Members name are arranged according to the Department and membership no. (for members in the same department). When a new member is formed, his/her name is inserted in proper place.

To search the availability of a book in the library, the title of the book should be used as the search key (if the entire title is not known then at least two keywords of the title can be used as the search key (e.g., for *"Design and Analysis of Algorithm"*, the search key may be *"Design Algorithm"*). If the title is not known properly then the name of the author can be used for searching a book. For the first case, if the book is enlisted in the library then all the corresponding information (author's name, year of print, number of copies present in the library with their availability status [issued (0), non-issued (1)]) is shown. For the second case, all books of the author will be displayed.

The issue operation displays the date of return/renew (for faculty, duration is one month and for student it is 2 weeks). A member can also check the return or renew date of all the books he/she has issued. A renewed permission can be extended upto 3 times. The maximum number of issues (i.e, number of books) for a faculty is restricted to 4 and that for students is restricted to 2 books.

A fine against late return is computed as 50 paise/day unanimously.
The insertion of a book in the library, should consider two different cases. A new print of an available book or a new book which was not available earlier in the library.

(i) Design such a library database using *suitable data structures* (***either AVL-tree, B tree or B+-tree***). Separate *structures* should be created for faculty and student member.
(ii) Define functions for insertion of a book by considering various conditions mentioned above.
(iii) Define functions to create new faculty or student member.
(iv) Find and display the names of the faculty and the student who currently issued maximum number of books.
(v) Write a function to calculate fine against a member and display the names of the borrower having maximum fine till date.
(vi) Sort (rearrange the nodes physically, do not move the data) and display the names of the borrowers in descending order of number of borrows currently in their possession. If more than one person have same number of books borrowed then faculty name comes first. If the borrowers having same number of borrows are in the same category (both are faculties or students) then display their name in alphabetical (dictionary) order.
(vii) Create a 2-3 ordered tree (B/B+) for keep tracking the name of the defaulters who has to pay fine (e.g., members paying fine for the first time, members paying fine second time and members paying fine more than twice). Display the names who have paid fine atleast trice.

Problem for R4 Batch(Enrolment Numbers BT19CSE091 to BT19CSE127):

Accommodation Management System

The following structure defines the accommodation records maintained for all residents of VNIT campus.

```
struct accommodation
{
        char firstname [50];
        char lastname [50];
        char accommodation_type [20];
        int idtype;
        struct idnum
        {
                char aadhaar [15];
                char passport [15];
                char empcode [15];
        };
        char address [100];
}
```

Create a database of all residents of VNIT campus using ***either AVL-tree, B-tree or B+-tree as per your choice***.

The field idtype takes a value 0, 1, or 2, indicating whether the ID is Aadhaar Card number, Passport number, or Employee code respectively.

(a) Write a function, *InsertRecord()*, which reads a record from user and insert into tree of structures. The records with idtype 0 must be stored first, followed by records with idtype 1 and 2. This order must be preserved throughout.
Example records:
Suman Dey Type-II 2 2022513 Formal methods lab, VNIT, Nagpur.
Pramod Gupta Type-I 1 ZF351076 Dept. of CSE, VNIT, Nagpur.
After reading idnum you must treat the rest of the line as the address.
The other fields of idnum should be null.

(b) Write a function *removeDuplicates()* to remove the duplicate entries from the records. (c) Write a function, *printRecords()*, which prints the records in tree so that all those who have provided Aadhar number are listed first, all those who have provided Passport number are listed next, and all those who have provided Employee code are listed at the end.

(d) We wish to print the records in tree in alphabetic order of names (sorted by firstname and ties resolved by lastname). Instead of sorting the records in tree, we will create a thing called index. For this purpose, create a tree, index, of N nodes, and then for each k, $0 \leq k < N$, populate k^{th} node with the index of the record containing the k^{th} name in alphabetic order. The order of nodes can be considered as top to bottom and left to right. Write a

function *printSortedRecords()* that uses the index tree, to print the records in alphabetic order of names.

(e) Write a *search()* function that prints the whole information for a given Employee name. If there are multiple records with same name then print them in alphabetic order. (f) Write a function *deleteRecord()* to delete the record of a given employee, the input to this function should be idtype and idnum. Make sure to change the index tree after deletion. (g) Write the *updateRecord()* function to update the record of an employee. The field to update should be the user input. Make sure to change the index tree after updating firstname or lastname.

(h) Create an additional structure having 10 blocks of each type of accommodation. The types of accommodation are type-A, type-B, type-C, and type-D. These blocks should be addressed with type, followed by serial number. For example, blocks in type-A will be named as A1, A2, A3, A4, … & so on for other blocks also.Mark the blocks as accommodated/not accommodated.

1. Write a function *allocateBlock()* to allocate block whenever the new resident record is inserted.
2. Write a function *deallocateBlock()* to deallocate the block whenever the resident record is deleted & mark that block as “not accommodated”.
3. Write a function *addressSearch()* to print the list of residents living in the given range of addresses. (For e.g., B5 to B9, C1 to C8).
4. Write a function *specialRequestAllocation()* to change a type of block requested by an employee depending on the availability of that type of block.