Problem Statement

A SIMPLE DATABASE SYSTEM

A database is an organized collection of data or information that can be easily accessed and updated. The data is typically stored in the file system and indexed (using different methods such as index trees) making it easier to find or update the relevant information.

In this project, students will develop a simple database system (named DSDB) that can store data in multiple files residing on the computer file system. DSDB will use (1) B tree, (2) AVL tree, and (3) Red-Black tree, to index the data (stored in files) so that different search and update operations can be performed efficiently.

Data stored in the DSDB system

We have provided the data to be stored in the DSDB along with this project. The data is about the leading causes of death in different states of the USA. Each tuple (or entry) in data comprises of the following fields.

- ID (This field is unique for every row/record/tuple within data)
- Year
- Cause name
- State
- Deaths
- Age-adjusted death rate

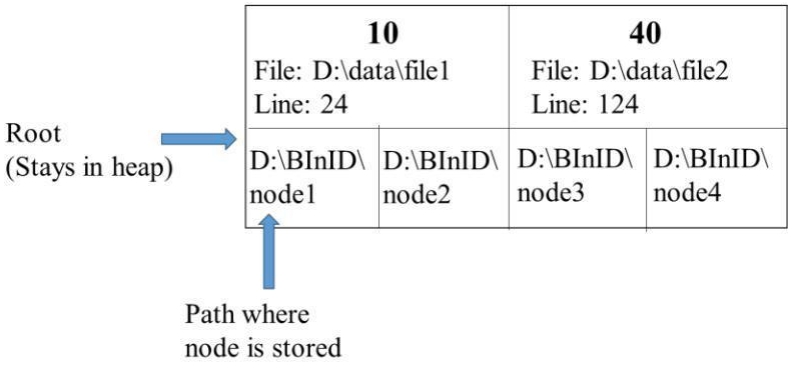
Any one of the above fields can be used for creating the index.

The data is stored in 10 different files. For this project, students are recommended to create a separate directory for storing the data files.

Index Trees

DSDB must support B-tree, AVL tree, and Red-Black tree-based indexing methods on the data. For more detailed information about the working of Red-Black trees, students are recommended to read pages 566 to 576 (i.e., section 12.2.) from the reference book "Data Structures and Algorithm Analysis in C++" by Mark Allen Weiss (edition 4). The book also contains the C++ code for implementing the Red-Black tree, however, an exact copy will be considered plagiarism.

For all trees, the root node will always reside in a heap (i.e., RAM). However, all other nodes will be stored on the file system in separate files. In the Figure below, if the data with key 11 is searched, the DSDB system will open the file D:\BInID\node2 and load the node data to proceed further with the search. Similarly, if data with key 10 is searched, the DSDB will open the file D:\data\file1 and read the required tuple from line 24.



Most importantly, all the files related to a tree index must be stored in a separate directory. For example, if a B tree index is created on the data field "ID", the DSDB system will store the file related to each node in the directory BInID (as shown in the Figure above). Likewise, if the AVL tree index is created on the data field "state" a directory with the name AVLState is used. Students can use their convention to name the directories, but a separate directory must exist for each index.

Indexing with duplicates

Indexing on data fields other than "ID" may result in duplicates, i.e., the same key may point to multiple tuples (or entries) in the data. The duplicates must be handled in the following manner.

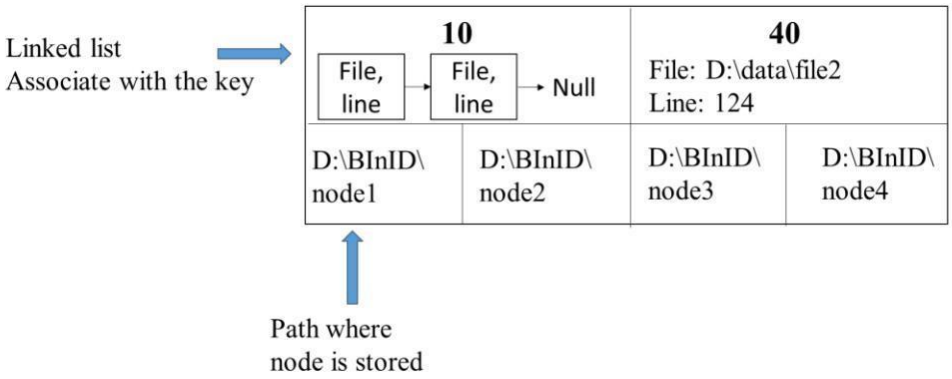
There must be only one entry for each key in both B, AVL, and Red-Black trees (even if the key is pointing to multiple tuples in the data). However, each key maintains the information about the file names and line numbers of all relevant entries (i.e., entries with the same key).

When the node is loaded in a heap (i.e., RAM), a linked list (associated with each key) is created to store the file names and line numbers of duplicate entries (as shown in the figure below).

Problem Statement

In the figure below, key 10 points to multiple data tuples. Therefore, a linked list is maintained, where each node (of the linked list) points to one tuple in the data. If the key 10 is searched, the DSDB returns the data tuples pointing to each node of the linked list.

It is difficult to store a linked list directly into the file system, therefore, when the node is stored in the file, the file names and line numbers of duplicate entries are stored as a list in textual form (i.e., pointers are not stored).



Operations supported by DSDB

- **Create index:** The user should be able to specify the following parameters:
  - B tree index or AVL tree index or Red-Black tree index (at the same time multiple indices can reside).
  - Order of B tree (i.e., the value of m), in case B tree index, is created.
  - On which data field to perform the indexing.

- **Point search:** The user can specify the key and the DSDB system should be able to display the corresponding tuple(s) from the data. For example, in case the indexing is performed on the data field "state" and user specifies "Michigan" as a key, the DSDB system returns all the data tuples containing Michigan.

- **Range search:** The user can specify a range of key values and the DSDB system should be able to display all the relevant tuples.

- **Update key, field, old value, new value:** The user can specify the key, and name of the field to be modified along with its old value and new value. The DSDB system should be able to find the tuple with the given key and change the value of the field to a new value. For example, in case the user specifies: *Update 5105, state Maryland, Michigan*, the DSDB system will find the tuple with ID 5105 and change the state from Maryland to Michigan.

  In case multiple tuples exist with the same key, the old value is used to break the ties. If old value is insufficient to break the tie an error is displayed by the DSDB system.

- **Delete key:** In this case, the DSDB system will delete all the tuples with the given key.

Additionally, the DSDB system should be able to display the performance of each operation in terms of the number of disk I/O operations performed.

Bonus operations

- **where field = value:** As a bonus, DSDB can support where clause in point search, range search and delete operations. For instance, the *"delete key where field = value"* (i.e., delete 2005 where state = Maryland) command by the user should only delete those tuples with the given key (2005) where the state field is equal to Maryland.

Additional details and advice

Your program should be menu driven and enable users to perform the different operations mentioned above. Moreover, memory allocation should be dynamic. Make sure to define a constructor and destructor for all classes. Your destructors must deallocate all dynamically allocated memory.

It is important to mention that Red-Black trees are part of your data structure course and may appear in your final exam.

What to submit

Submit your code for this project, programmed in C++. A document highlighting the design in terms of relationships/associations between different classes of your program must be submitted. The code needs to be well documented so that graders can get a good idea of what each of your procedures does.

An evaluation matrix/rubric is also uploaded along with this project.

Good Luck!