National University of Computer and Emerging Sciences

School of Computing Fall 2018 Islamabad Campus

CS201: Data Structures (Fall 2018)
Semester Project
(Deadline: 7th January, 2019 08:00 AM)

Project groups: This project can be done within a group of three (3) students. There is no restriction on the selection of group members. Students are allowed to make groups according to their preferences. The group members may belong to the same or different sections.

Submission: All submissions MUST be uploaded on slate. Solutions sent to the emails will not be graded. To avoid last minute problems (unavailability of slate, load shedding, network down etc.), you are strongly advised to start working on the project from day one.

You are required to use Visual Studio 2010 for the project. Combine all your work (solution folder) in one .zip file after performing "Clean Solution". Submit zip file on slate within given deadline. If only .cpp file is submitted it will not be considered for evaluation.

Deadline: Deadline to submit project is **7th January, 2019 08:00 AM**. No submission will be considered for grading outside slate or after **7th January, 2019 08:00 AM**. Correct and timely submission of project is responsibility of every group; hence no relaxation will be given to anyone.

Plagiarism: -50% marks in the project if any significant part of project is found plagiarized. A code is considered plagiarized if **more than 20%** code is not your own work.

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 relevant information.

In this project, students will develop a simple database system (named as DSDB) that can store data in multiple files residing on the computer file system. DSDB will use B tree and B+ tree to index the data (stored in files) so that different search and update operations can be performed in an efficient manner.

Data stored in DSDB system

We have provided the data to be stored in the DSDB along with this project. The data is pertaining to the leading causes of deaths in different states of 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

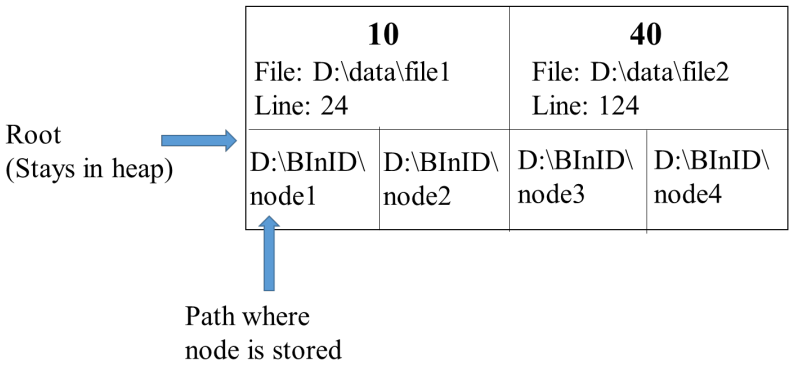
Any one of the above fields can be used for creating the index.

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

Index Trees

DSDB must support both B and B+ tree based indexing on the data. For more detailed information about the working of B+ trees, students are recommended to read pages 368 to 374 (i.e., section 10.5.1.) from the reference book "Data Structures and Algorithm Analysis (C++ version) by Clifford A. Shaffer (edition 3.2)". The book also contains the C++ code for implementing B+ trees. Students can take help from the code, however, exact copy will be considered plagiarism.

For both B and B+ trees, the root node will always reside in the 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 B or B+ tree index must be stored in a separate directory. For example, in case B tree index is created on the data field "ID", DSDB system will store the file related to each node in the directory BInID (as shown in the Figure above). Likewise, if B+ tree index is created on the data field "state" a directory with the name B+State is used. Students can use their own convention to name the directories but a separate directory must exist for each index.

Indexing with duplicates

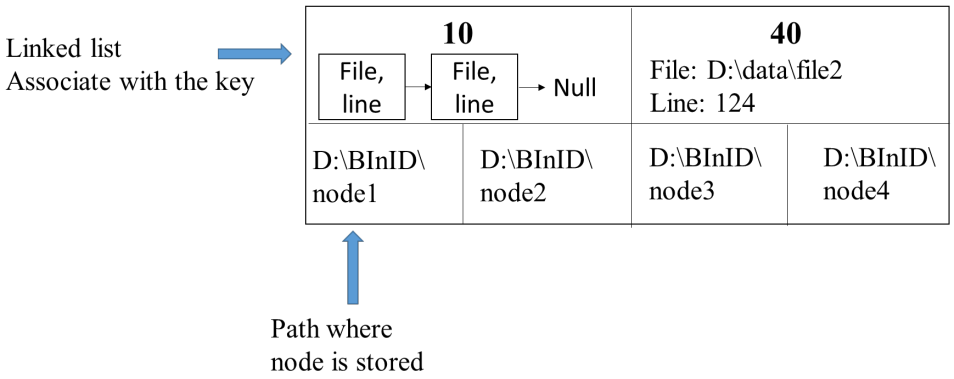
Indexing on data fields other than "ID" may result in duplicates, i.e., same key may point to multiple tuples (or entries) in the data. The duplicates must be handled in the following manner.

There must be only one entry for each key in both B and B+ trees (even if 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 the 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 figure below).

In the figure below, key 10 points to multiple data tuples. Therefore, a linked list is maintain 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 by each node of the linked list.

It is difficult to store 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 B+ tree index (at same time multiple indices can reside).
  - Order of B tree or B+ tree (i.e., value of m).
  - On which data field to perform indexing.

- **Point search:** The user can specify the key and DSDB system should be able to display the corresponding tuple(s) from the data. For example, in case the indexing is performed on 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 DSDB system should be able to display all the relevant tuples.

- **Update key, field, old value, new value:** The user can specify the key, 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 new value. For example, in case 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, 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 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, *"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 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 B+ 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 grader can get a good idea of what each of your procedures do.

Program modularity, clarity, documentation etc., along with correct implementation will be considered for grading.

Good Luck!