CS 564 Project Checkpoint 3: Understanding B+ Tree Index

Goal: To develop a hands-on understanding of indexing by creating a B+ tree index to help facilitate the efficient searching and accessing of data.

In this checkpoint, you are given data in the form of a csv file ('Student.csv') which contains a Student table for you to index. Each group will implement a B+ tree library to accomplish this. Your implementation should have the ability to:

- Create a B+ tree index file for the Student table on the *StudentID* field.
    - The leaf nodes should contain <StudentID, recordID>
- Search for an existing student using their *StudentID* and return the *RecordID*.
    - If not found, print out a message that the given *StudentID* does not exist.
- Insert a new student with a new *StudentID*.
    - Use a random generator to generate a *RecordID* for this student, then update both the B+ tree index and the Student table.
- Delete an existing student given a *StudentID*.
    - Return True if the deletion is completed successfully, otherwise, return False.
- Print the leaf nodes of the B+ tree from left to right.
    - Return the list of *recordIDs*.

Additional Specifications:

- The library will take user input and commands in the form of a text file ('input.txt').
- *RecordID* represents the address of each row in this table. The data for this table can be found in the 'Student.csv' file.
- For the B+ tree's insert algorithm, base your code on the pseudo-code provided in the textbook. Pay special attention to how they handle splitting.
- For the B+ tree's delete algorithm, base your code on the pseudo-code provided in the textbook. Pay special attention to how they handle merging.
- Each leaf node in your B+ tree should contain the pair (*key, rid*). In this application, the key corresponds to *StudentID* while *rid* corresponds to *RecordID*.
- Your library should allow users to create a B+ tree with a specific degree/order, as specified in the input file.

Code Guidelines:

- You are required to use the provided skeleton code.
- You are to complete the code by implementing:
    - The *getStudents* method in BTreeMain.java
    - The logic for generating *RecordID* in BTreeMain.java
    - The *insert, delete, search*, and *print* methods in BTree.java
- For your convenience, TODO comments have been added to the files to indicate what code needs to be written and where it goes.
- You are permitted to add to or modify the classes and methods, but you should NOT modify the code related to IO in BTreeMain.java.
    - Make sure to verify that the Student.csv file reflects all added entries.

- Make sure to add meaningful comments describing what your code is doing.

Testing:

- You should initially test your code using the sample test input file ('input.txt').
- You should also modify the sample test input file to test different use cases to ensure the functionality of your library.
- Verify that your results are added to the 'Student.csv' file AND your B+ Tree.
- Make sure your implementation runs correctly on the CSL machines.

Assumptions:

- You may assume that all the values being indexed are unique.
- The schema of the Student table is given as:

Student(*StudentID:* bigint,
StudentName: varchar(255),
Major: varchar(255),
Level: char(2),
Age: int,
RecordID: bigint)

Deliverables:

- Compress ALL java files needed to run your program into a zip folder.
- Submit the zip folder to Canvas.
- One submission per group.

Grading: See attached rubric.

Criteria	Unsatisfactory (0-2 points)	Fair 3 points	Good 4-5 points	Excellent 6 points
B+ tree created for the original Student.csv	Code still has compiling errors or run time errors	Code is compiled and runs, but B+ tree is created incorrectly	B+ tree is created correctly except for 1-2 minor errors/things needing improvement	B+ tree is created correctly
Insert	Code still has compiling errors or run time errors	The code is compiled and runs; however, insert does not function entirely correctly	Successful implementation of insert except for minor errors/things which could be improved	Successful implementation of insert
Search	Code still has compiling errors or run time errors	The code is compiled and runs; however, search does not function entirely correctly	Successful implementation of search except for minor errors/things which could be improved	Successful implementation of search
Delete	Code still has compiling errors or run time errors	The code is compiled and runs; however, delete does not function entirely correctly	Successful implementation of delete except for minor errors/things which could be improved	Successful implementation of delete
Readability	The code is poorly organized and very difficult to read	The code is readable only by someone who knows what it is supposed to be doing	The code is fairly easy to read	The code is exceptionally well organized and very easy to follow. Comments are plentiful

*We reserve the right to deduct additional points for failure to comply with instructions (i.e. wrong submission format, modifying IO code, not using the skeleton code, etc.)