Middle East Technical University | Department of Computer Engineering

CENG 351

Data Management and File Structures

Fall '2024-2025
PA2 – B+ Tree Index
Due: 11 January 2025 23:55
(v.1.1)

1 Introduction

METU has an increasing number of published papers every day, and we need to keep the published papers in a B+ tree with primary and secondary indexing. In this assignment, you should store the academic papers using both primary and secondary B+ Trees. You will implement only certain operations described below. A graphical user interface (GUI) is provided such that you can check the correctness of your implementation.

In the primary B+ tree, you should implement a clustered B+ tree index on the `paperId` field. On the other hand, in the secondary B+ tree, you should implement an unclustered B+ tree on the `journal` field, which is a non-unique attribute.

Note: In the secondary B+ tree, the leaf nodes store the paper IDs associated with each journal. While in real-world scenarios, the number of paper IDs in each leaf node would adhere to the tree's order, we simplify the implementation by not enforcing this restriction. Instead, only the search key, which is the journal in this case, follows the order constraints of the tree.

2 Objectives

There are 5 main operations that you should implement in the assignment:

- **Add Paper**: Adds a new paper to the trees. The key attribute of the academic papers is `paperId`, so you should store the papers according to the `paperId` field in the primary B+ Tree. For the secondary B+ tree, you should consider the `journal` field as an index. Multiple papers may have the same `journal` field. Thus, the leaf nodes should use an **unclustered index structure**. When you add a record, you should update both primary and secondary B+ trees.
- **Search Paper**: Searches for an academic paper with the given `paperId` in the primary B+ tree and prints the visited nodes along the path. Output specifications are detailed in Section 4.

- **Search Journal**: Searches for a journal with the given `Journal Name` in the secondary B+ tree and prints the visited nodes along the path. Output specifications are detailed in Section 4.
- **Print Primary Tree**: Prints the primary tree in depth-first order. Output details are in Section 4.
- **Print Secondary Tree**: Prints the secondary tree in depth-first order. Output details are in Section 4.

3 Project Structure

3.1 GUI Classes

The following classes are implemented to provide a better environment for debugging. These are not expected to be modified. They will only be executed if the program's `guiOptions` parameter is set to greater than 0. Note that grading will be done without using the GUI.

- `CengGUI.java`
- `GUIInternalPrimaryNode.java`
- `GUIInternalSecondaryNode.java`
- `GUILevel.java`
- `GUIPrimaryLeafNode.java`
- `GUISecondaryLeafNode.java`
- `GUITreeNode.java`
- `WrapLayout.java`

3.2 Main Files

The following classes and files are used to parse the input and determine the structure of the databases:

- `CengScholar.java`
- `CengPaper.java`
- `ScholarNode.java`
- `ScholarNodePrimaryLeaf.java`
- `ScholarNodeSecondaryLeaf.java`
- `ScholarNodePrimaryIndex.java`
- `ScholarNodeSecondaryIndex.java`

- `ScholarNodeType.java`
- `ScholarParser.java`
- `ScholarTree.java`

3.3 Implementation

You can make changes to only 3 files listed below:

- `ScholarTree.java`: Builds a B+ tree structure for both primary and secondary indexes. Implement the methods marked as TODO in this file. You can define additional methods and attributes if necessary.
- `ScholarNodePrimaryIndex.java`: Builds internal index nodes of the primary B+ tree. You can define additional methods and attributes if necessary.
- `ScholarNodeSecondaryIndex.java`: Builds internal index nodes of the secondary B+ tree. You can define additional methods and attributes if necessary.

4 Input/Output

The evaluation of the assignment will be done without using the GUI. The GUI is used to visualize the trees and call the necessary functions. An input file is provided to allow the GUI to load examples. The order `d` of the B+ trees will be given as an input.

You should continuously listen for inputs from the command line in an infinite loop until the string `quit` is read. When `quit` is read, terminate the program.

4.1 Add

Input format: `add|<paperId>|<journal>|<title>|<author>|`
Example Input: `add|1|AI Journal|Neural Networks|Alice Johnson`
Output: None

Note: the use of the pipe character | and the lack of spaces between arguments, in order to allow spaces within each argument.

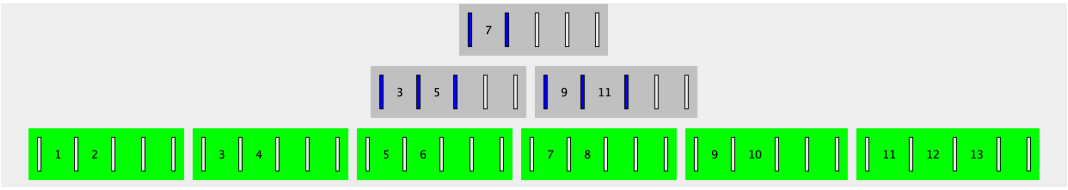

Figure 1: Example primary B+ tree after first 13 add operations of example input file.

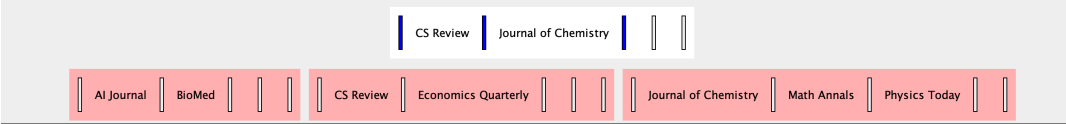

Figure 2: Example secondary B+ tree after first 13 add operations of example input file.

4.2 Search

Should be triggered when **search1** or **search2** are read as input. If input is search1, then you should search given paperId in primary B+ tree. If input is search2, then you should search given journal in secondary B+ tree.
Input format: `search1|<searchKey>`
Example Input: `search1|7`
Output: Prints the visited nodes starting from the root to the data records in the leaf nodes. Indentation with tabs should be added while traversing down each level of the tree. If the record is not found, print `Could not find <search key>`.

Example search output of paper with id 7 on empty tree.

```
search1|7
Could not find 7
```

Example search output of paper with id 7 after the first 13 add operations.

```
search1|7
<index>
7
</index>
        <index>
        9
        11
        </index>
                <data>
                <record>7|AI Journal|Deep Learning|Grace Martin</record>
                </data>
```

Example search output of paper with id 8 after the first 13 add operations.

```
search1|8
<index>
7
</index>
        <index>
        9
        11
        </index>
                <data>
                <record>8|CS Review|Hashing Techniques|Henry Clark</record>
                </data>
```

Example search output of paper with id 21 after the first 13 add operations.

```
search1|21
<index>
7
</index>
        <index>
        9
        11
        </index>
Could not find 21
```

Input format: `search2|<searchKey>`
Example Input: `search2|AI Journal`
Output: Prints the visited nodes starting from the root to the data records in the leaf nodes. Indentation with tabs should be added while traversing down each level of the tree. If the record is not found, print `Could not find <search key>`. The journals papers are also printed in the secondary B+ tree.
Note: To print paper records, they should be fetched from primary B+ tree.

Example search output of journal "AI Journal" after the first 13 add operations.

```
search2|AI Journal
<index>
CS Review
Journal of Chemistry
</index>
        <data>
        AI Journal
                <record>1|AI Journal|Neural Networks|Alice Johnson</record>
                <record>7|AI Journal|Deep Learning|Grace Martin</record>
        </data>
```

Example search output of journal "CS Review" after the first 13 add operations.

```
search2|CS Review
<index>
CS Review
Journal of Chemistry
</index>
        <data>
        CS Review
                <record>2|CS Review|Sorting Algorithms|Bob Smith</record>
                <record>8|CS Review|Hashing Techniques|Henry Clark</record>
        </data>
```

4.3 Print

Should be triggered when **print1** or **print2** are read as input. If input is print1, then you should print primary B+ tree. If input is print2, then you should print secondary B+ tree.
Input format: print1

Output format: All non-leaf and leaf nodes should be printed with a proper indentation (with tabs) for each level in the tree. For non-leaf nodes, you should print the search key enclosed by **<index>** and **</index>** tags. For leaf nodes, you should print the content between **<data>** and **</data>** tags. Records should be printed between with **<record>** and **</record>** tags.

Input format: print2
Output format: All non-leaf and leaf nodes should be printed with a proper indentation (with tabs) for each level in the tree. For non-leaf nodes, you should print the search key enclosed by **<index>** and **</index>** tags. For leaf nodes, you should print the content between **<data>** and **</data>** tags. Records should be printed between with **<record>** and **</record>** tags.

Example print1 output of an empty tree.

```
print1
<data>
</data>
```

Example print1 output after the first 4 add operations.

```
print1
<data>
<record>1|AI Journal|Neural Networks|Alice Johnson</record>
<record>2|CS Review|Sorting Algorithms|Bob Smith</record>
<record>3|BioMed|Genome Sequencing|Carol Lee</record>
<record>4|Math Annals|Number Theory|David Brown</record>
</data>
```

Example print1 output after the first 13 add operations.

```
print1
<index>
7
</index>
        <index>
        3
        5
        </index>
                <data>
                <record>1|AI Journal|Neural Networks|Alice Johnson</record>
                <record>2|CS Review|Sorting Algorithms|Bob Smith</record>
                </data>
                <data>
                <record>3|BioMed|Genome Sequencing|Carol Lee</record>
                <record>4|Math Annals|Number Theory|David Brown</record>
                </data>
                <data>
                <record>5|Physics Today|Quantum Mechanics|Emily Davis</record>
                <record>6|Economics Quarterly|Market Analysis|Frank Wilson</↩
```

```
            record>
        </data>
    <index>
    9
    11
    </index>
        <data>
        <record>7|AI Journal|Deep Learning|Grace Martin</record>
        <record>8|CS Review|Hashing Techniques|Henry Clark</record>
        </data>
        <data>
        <record>9|BioMed|Protein Folding|Ivy Baker</record>
        <record>10|Math Annals|Topology|Jack Turner</record>
        </data>
        <data>
        <record>11|Physics Today|Thermodynamics|Karen Evans</record>
        <record>12|Economics Quarterly|Risk Management|Leo Young</record>
        <record>13|Journal of Chemistry|Molecular Bonds|Mia Collins</↩
            record>
        </data>
```

Example print2 output of an empty tree.

```
print2
<data>
</data>
```

Example print2 output after the first 4 add operations.

```
print2
<data>
AI Journal
        <record>1</record>
BioMed
        <record>3</record>
CS Review
        <record>2</record>
Math Annals
        <record>4</record>
</data>
```

Example print2 output after the first 13 add operations.

```
print2
<index>
CS Review
Journal of Chemistry
</index>
        <data>
        AI Journal
            <record>1</record>
```

```
        <record>7</record>
    BioMed
        <record>3</record>
        <record>9</record>
    </data>
    <data>
    CS Review
        <record>2</record>
        <record>8</record>
    Economics Quarterly
        <record>6</record>
        <record>12</record>
    </data>
    <data>
    Journal of Chemistry
        <record>13</record>
    Math Annals
        <record>4</record>
        <record>10</record>
    Physics Today
        <record>5</record>
        <record>11</record>
    </data>
```

---

5 Submission

You should submit only the files listed in Section 3.3. Ensure that your implementation is covered only under these files. Do not use subdirectories. Archive the files using the following command:

`tar -cvf <e123456>.tar.gz`

`<e123456>` should be replaced with your student ID.

6 Regulations

- **Programming Language:** Java (Version 14).
- Ensure that your submission can be extracted using: `tar -xvf <e123456>.tar.gz`
- Grading will be done using black-box testing. Before submitting, ensure your code runs with the following commands:

```
javac *.java
java CengScholar <order> <guiOptions> <inputFileName>
```

`<guiOptions>`: 0 for no GUI, 1 for primary tree only, 2 for secondary tree only, 3 for both primary and secondary trees.

- Ensure that your output matches the format described in Section 4 exactly.
- Zero-tolerance policy for cheating. Cheating will be punished according to university regulations.