NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

# CZ4031
# Database System Principles

# Project 1 Report
# Group 4

**Members:**

Rosario Gelli Ann Labayen (U1822818E)

Tang Heem Yew (U1820996J)

Ngiam Jing Xiang (U1820038H)

Lexx Audrey Pecson Manalansan (U1822109H)

# Introduction

This report aims to design and implement a movie database management system, storage and indexing. The following specifications are set

- Programming language : Java
- Indexing component: B+ tree

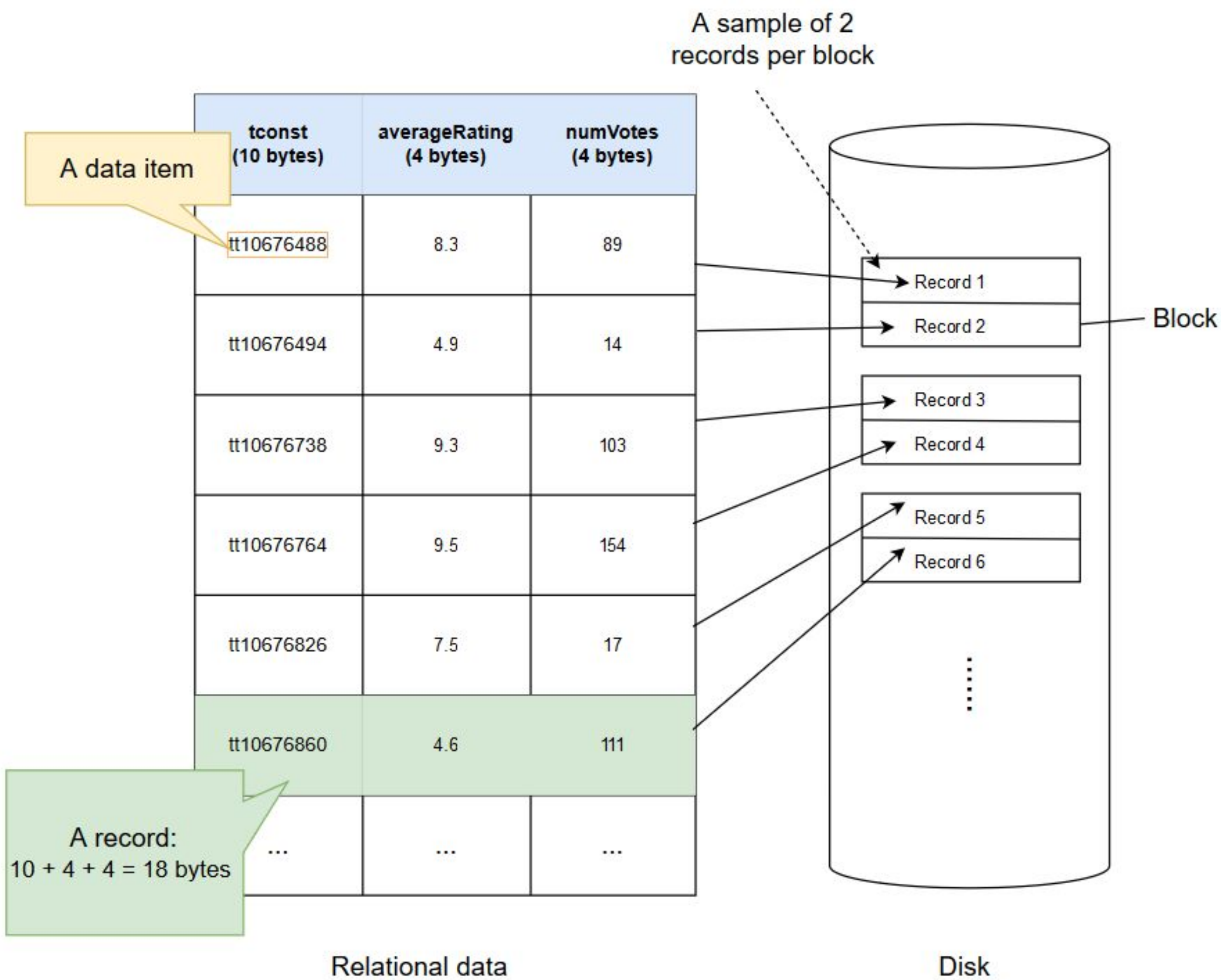
The data contains a movie information list that each consist of tconst (unique movie id), averageRating, and numVotes. The data contains over 1 million records. Table below shows a sample of the data.

| tconst | averageRating | numVotes |
|---|---|---|
| tt10676488 | 8.3 | 89 |
| tt10676494 | 4.9 | 14 |
| tt10676738 | 9.3 | 103 |

## Storage Design Consideration

Based on the data list provided, the following design considerations are made for the storage of the components, a class named *Record* will store the following :

| Data Column | Data Type | Consideration |
|---|---|---|
| *tconst* | String | String is a suitable data type as it will only take up 10 bytes. |
| *averageRating* | float | As decimal is part of the required input for the calculation of the average rating of each movie, and with 10.0 as the possible maximum average rating for a record, float and double can be used.<br><br>Float can store about 7 decimal digits. **(Size : 4 bytes)**<br>Double can store about 14 decimal digits. **(Size : 8 bytes)**<br><br>Given our data that only needs 1 decimal digit, considering the size and the number of decimal digits, float is sufficient for this data. |
| *numVotes* | int | *numVotes* refers to the number of votes the title has received. Inputs required for this data are whole numbers.<br><br>Hence, int with the size of 4 bytes will be the ideal data type for this data. |



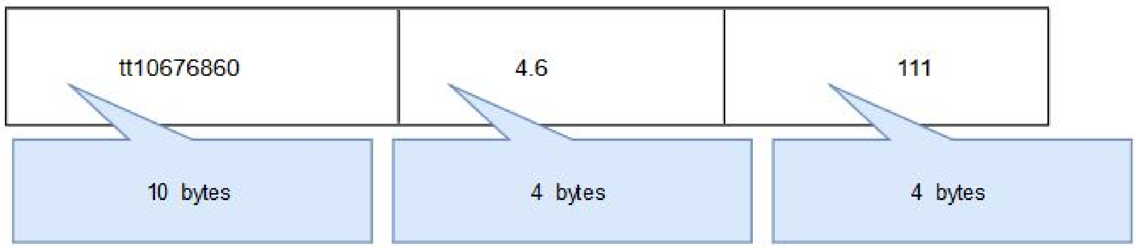

The format we have decided to use is fixed-length fields. The rationale we choose fixed-length fields is because it is easier to interpret, however some space is wasted in the memory as compared to variable-length fields.

Storing each data item as a field:

Each data item is represented with a fixed number of bytes and they are stored as a field. The following shows the sample of our data item.

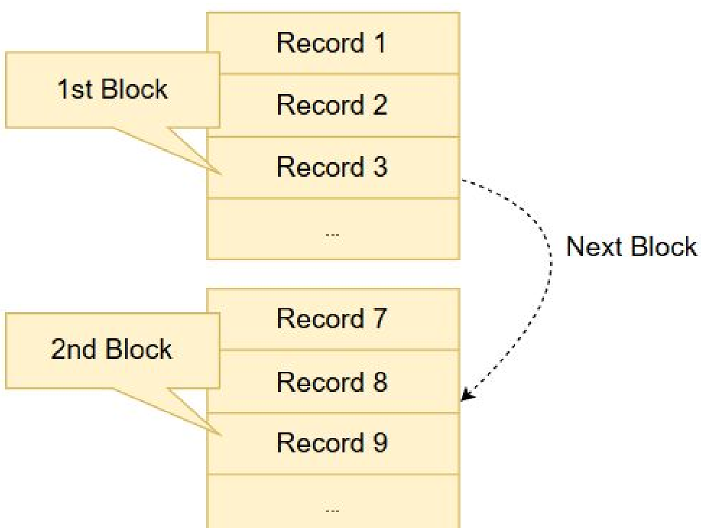
1) The string attribute “tconst” -> “tt10676860” has 10 characters and 10 bytes, since each character is considered as one byte.
2) The float attribute “averageRating” -> “4.6” requires only 1 decimal digit, hence 4 bytes is sufficient for this data type.
3) The integer attribute “numVotes” -> highest value is “2,279,223”, which requires at least approximately 22 bits to represent the data type, hence 4 bytes will be the ideal data type.

Packing fields into records:
A typical record with a fixed format with fixed length is as shown below:

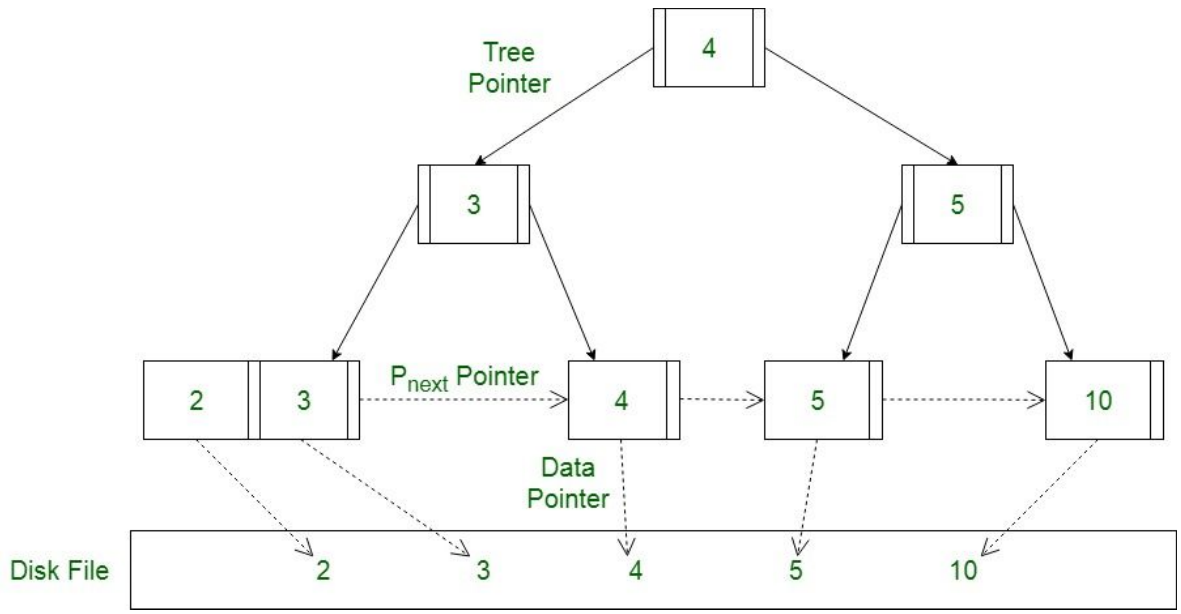



Packing records into blocks:

The records are unspanned records, which means records must be within one block. It is much simpler to manage the data in this way. The blocks are organized on disks sequentially since all blocks have a fixed length, separation is not needed. The current block is physically contiguous with the next block, so on and so forth.



# B+ Tree Overview




A B+ tree is a data structure often used in the implementation of database indexes. The B+ tree consists of a single node at the top also known as root node. The root node points to two or more blocks called child nodes which have further child nodes and so on. The B+-Tree is called a balanced tree because every path from the root node to a leaf node is the same length.

The B+ tree is similar to a B tree except:

- B+ trees don't store data pointers in interior nodes, they are ONLY stored in leaf nodes. This means that interior nodes can fit more keys on the block of memory.
- The leaf nodes of B+ trees are linked, so doing a linear scan of all keys will require just one pass through all the leaf nodes. This property can be utilized for efficient search as well since data is stored only in leafs.

The primary value of B+ Tree is that it offers significant value by providing efficient data retrieval in block-oriented storage applications like file systems and databases.

B+ tree satisfies the following conditions:

- An internal node (nodes that aren't the root or leaves) must have a number of children d such that $\lceil m/2 \rceil \le d \le m$
- The root node may have at least 2 children and up to m children
- The number of keys k that an internal node may carry is $\lceil m/2 \rceil - 1 \le k \le m - 1$
- Leaf nodes have no children, but the number of dictionary pairs k that a single leaf node may carry is $\lceil m/2 \rceil - 1 \le k \le m - 1$

## Code Implementation

The B+ Tree program consists of the following:

1. ***Key.java*** - The Key class contains key value and the list of values (data pointer).
    - *Attributes*

    ```
    private float key; // Key Value

    List<Record> values; // Data Value: Array List of Record
    ```

    - *Methods:*
        - *get/set* methods
        - *toString()* - prints the information of key

```
import java.util.ArrayList;

public class Key {

    private float key; // Key Value

    List<Record> values; // Data Value: Array List of Record

    // Initialize Key
    public Key(float key, Record value) {
        this.key = key;
        if (null == this.values) {
            values = new ArrayList<>();
        }
        this.values.add(value);
    }

    // Initialize Key
    public Key(float key) {
        this.key = key;
        this.values = new ArrayList<>();
    }

    // Return Key Value
    public float getKey() {
        return key;
    }

    // Set Key Value
    public void setKey(float key) {
        this.key = key;
    }

    // Get Key Data Values
    public List<Record> getValues() {
        return values;
    }

    // Set Key Data Values
    public void setValues(List<Record> values) {
        this.values = values;
    }

    // Print Key Information
    public String toString() {
        //return "<KEY>[Key= " + key + ", DataPointer= " + values + "] ";
        return "<KEY>[Key= " + key +  "] ";
    }
}
```

2. ***Node.java*** - The Node class contains the ArrayList of *Key class (Key.java) and ArrayList of Node Class* which is used as a pointer to children.
    - *Attributes:*

```java
public class Node {
    private List<Key> keys; //Pointer to Keys

    private List<Node> children; //Pointer to Children

    private Node prev; //Pointer to Previous Node

    private Node next; //Pointer to Next Node

    private Node parent; //Pointer to Parent Node

    public boolean isLeaf; //Indicate if it's a leaf node
```

- *Methods:*
    - *get/set* methods
    - *toString()* - prints the information of node keys

```java
    // Initialize Node
    public Node() {
        this.keys = new ArrayList<>();
        this.children = new ArrayList<>();
        this.prev = null;
        this.next = null;
    }

    // Return Array List of Keys
    public List<Key> getKeys() {
        return keys;
    }

    // Set Keys of the Node
    public void setKeys(List<Key> keys) {
        Iterator<Key> iter = keys.iterator();
        while (iter.hasNext()) {
            this.keys.add(iter.next());
        }
    }

    // Get List of Children Node
    public List<Node> getChildren() {
        return children;
    }

    // Set Children of Node
    public void setChildren(List<Node> children) {
        this.children = children;
    }

    // Get Next Node
    public Node getNext() {
        return next;
    }

    // Set Next Node
    public void setNext(Node next) {
        this.next = next;
    }

    // Get Parent Node
    public Node getParent() {
        return parent;
    }

    // Set Parent Node
    public void setParent(Node parent) {
        this.parent = parent;
    }

    // Get Previous Node
    public Node getPrev() {
        return prev;
    }

    // Set Previous Node
    public void setPrev(Node prev) {
        this.prev = prev;
    }

    // Return Node Information
    @Override
    public String toString() {
        return "Keys: " + keys.toString();
    }
```

3. ***BPTree.java*** - The BPTree class contains methods in implementing the B+ tree.

- *Attributes:*

```java
// B+ Tree Info
private int m;
private Node root;
private int height = 0;

// For Experiment Purposes
private int numNodes = 0;
private int numDeleted = 0;
private int numMerged = 0;
private int indexNodesAccess = 0;
private int dataBlocksAccess = 0;
private int uniqueKeysCount = 1;
private int recordsCountInANode = 0;
private int recordsCountTotal = 0;
private int numOfNodes = 0;
```

- *Methods:*
  - *get/set* methods
  - *Insert, Search, Delete*
  - *merge, split* - merge/split nodes
  - *print* - prints the information of nodes and trees

```java
public BPTree(int order) {
    this.m = order; // Set B+ Tree M value
    this.root = null; // Set Root Node to null on default
}

// 1: Insertion Functions
public void insertKey(float key, Record value) {
    // 1: Empty B+ Tree => Create New Root Node
    if (null == this.root) {
        Node newNode = new Node();
        newNode.getKeys().add(new Key(key, value));
        this.root = newNode;
        this.root.isLeaf = true;
        this.root.setParent(null); // Since the root has no parent, parent set to null
    } else if (this.root.getChildren().isEmpty() && this.root.getKeys().size() < (this.m
        // 2: Node is Not Full
        this.root.isLeaf = false;
        insertExternalNode(key, value, this.root);
    } else {
        // 3: Normal insert
        Node curr = this.root;

        // Traverse to the last leaf node
        while (!curr.getChildren().isEmpty()) {
            curr = curr.getChildren().get(searchInternalNode(key, curr.getKeys()));
        }

        insertExternalNode(key, value, curr);

        // External node is full => Split node
        if (curr.getKeys().size() == this.m) {
            splitExternalNode(curr, this.m);
        }
    }

    // Increase number of nodes value
    numNodes++;

}

private void insertExternalNode(float key, Record value, Node node) {
    // Find index of key to be inserted
    int index = searchInternalNode(key, node.getKeys());

    if (index != 0 && node.getKeys().get(index - 1).getKey() == key) {
        // Add the new value to the list
        node.getKeys().get(index - 1).getValues().add(value);
    } else {
        // Key is null => Add key and data
```

*(see full code in BPTree.java)*

4. ***Block.java*** - The Block class adds records into blocks.
    - *Attributes:*

```
public class Block {

    private List<Record> records; //Pointer to Records
```

    - *Methods:*
        - *get/set* methods

```
    public Block ()
    {
        this.records = new ArrayList<>();
    }

    public List<Record> getRecords() {
        return records;
    }

    public void setRecords(List<Record> records) {
        Iterator<Record> iter = records.iterator();
        while (iter.hasNext()) {
            this.records.add(iter.next());
        }
    }
}
```

5. ***Record.java*** - The Record class contains the records of the database.
    - *Attributes:*

```
  private String tconst;
  private float averageRating;
  private int numVotes;
```

    - *Methods:*
        - *get* methods

```
public Record(String tconst, float averageRating, int numVotes) {
    this.tconst = tconst;
    this.averageRating = averageRating;
    this.numVotes = numVotes;
}

public String getTConst() {
    return tconst;
}

public float getAverageRating() {
    return averageRating;
}

public int getNumVotes() {
    return numVotes;
}
```

6. ***Database.java*** - The Database class contains methods to allocate/deallocate blocks, check database size and print records.
    - *Attributes:*

```java
public static int recordSize; // Record Size

private int poolSize; // Memory Pool Size
private int blockSize; // Block Size
private int freeSize; // Free Size
private int sizeUsed; // Size Used
private int allocated; // Allocated Size
private int remaining; // Number of Blocks Remaining
private int block; // Number of Blocks
public int recordsPerBlock; // Number of Records Per Block
public int totalNoOfRecords; // Total Number of Records
public int totalBlockSize; // Total Block Size
public int totalRecordSize; // Total Record Size
public int recordCounter = 0; // Number of Record
public int databaseSize = 0; // Database Size

private List<Block> listBlk; // Pointer to Data Block
private Block blk; // Block
```

    - *Methods:*
        - *get/set* methods
        - *allocate/deallocate* - allocating/deallocating of blocks
        - *print* - prints the information about the disk space size

```java
    public void printDatabaseInfo() {
        System.out.println("Memory Size: " + poolSize + " bytes");
        System.out.println("Block Size: " + blockSize + " bytes");
        System.out.println("Record Size: " + recordSize + " bytes");
        System.out.println("Free Size: " + freeSize + " bytes");
        System.out.println("Size Used: " + sizeUsed + " bytes");
        System.out.println("Available: " + remaining + " Blocks");
        System.out.println("Allocated: " + allocated + " Blocks");
        System.out.println("Total number of records: " + totalNoOfRecords
        System.out.println("Number of records per block: " + recordsPerBl
        System.out.println("Number of Blocks: " + block);
        System.out.println("Total record size: " + totalRecordSize);
        System.out.println("Total block size: " + totalBlockSize);

        databaseSize = totalRecordSize + totalBlockSize;
        System.out.println("The Size of database: " + databaseSize);
    }

    public void setRecord(int totalNoOfRecords) {
        this.totalNoOfRecords = totalNoOfRecords;

        // round up to the nearest whole number to get minimum number of
        this.block = (int) Math.ceil((double) this.totalNoOfRecords / (do
    }

    public int getRecord() {
        return totalNoOfRecords;
    }

    public void printDataRecords() {

        System.out.println("List blk contents size: " + listBlk.size());

        for (int i = 0; i < listBlk.size(); i++) {
            // System.out.println("Each blk contents size: " +
            // listBlk.get(i).getRecords().size());

            for (int j = 0; j < listBlk.get(i).getRecords().size(); j++)
                /*
                 * System.out.println("Blk record tconst: " +
                 * listBlk.get(i).getRecords().get(j).getTConst());
```

(see full code in Database.java)

7. ***Main.java*** - The main class reads the data.tsv file from the folder and creates the B+ Tree in the program. As it reads from the file, it creates a node and inserts into the tree. While making sure it follows the rules of B+ Tree, it will split or merge other nodes while it inserts in each line of the file. The main class outputs the experiment's details.
    - *Attributes:*

```java
// 1: Load Data File
String workingDir = System.getProperty("user.dir");
// String fileName = workingDir + "\\" + "Project-BPlusT
// + "\\" + "data.tsv";
String fileName = workingDir + "\\" + "data.tsv";
File inputFile = new File(fileName);

// 2: Set Database and B+ Tree Size
BPTree tree = null;
Database db = null;

boolean exit1 = false;
boolean selected = false;
```

    - *Methods:*
        - *Scanner* - reads the data file
        - Creates B+ tree
        - *Experiment options* - carry out experiments

```java
db.setRecord(recordCounter);
sc.close();
System.out.println("[Done Loading]");

// 4: Main Menu
boolean exit = false;
do {

    System.out.println("============================Experiments============================");
    System.out.println("1: Database Info");
    System.out.println("2: B+ Tree Info");
    System.out.println("3: Search Key with average rating of 8");
    System.out.println("4: Search Key Range with average rating of 7 to 9");
    System.out.println("5: Delete Key with average rating of 7");
    System.out.println("6: Quit");
    Scanner scan = new Scanner(System.in);
    int choice = scan.nextInt();

    switch (choice) {
    case 1:
        // Experiment 1
        System.out.println("============================Experiment 1============================");

        db.printDatabaseInfo();
        db.printDataRecords();
        break;
    case 2:
        // Experiment 2
        System.out.println("============================Experiment 2============================");
        tree.displayTreeInfo();
        tree.displayHeightInfo();
        System.out.print("\n");
        break;

    case 3:
        // Experiment 3
        System.out.println("============================Experiment 3============================");
        List<Record> searchValues = tree.searchKey(8);

        System.out.println("List of tconst: ");
        for (int j = 0; j < searchValues.size(); j++) {

            System.out.print(searchValues.get(j).getTConst() + " ");

            if (j % 100 == 0 && j != 0) {
                System.out.print("\n");
```

*(see full code in main.java)*

# Experiments

## Experiment 1

Store the data (which is about IMDb movies and described in Part 4) on the disk and report the following statistics:

**a) The number of blocks:**

**1- Block Size = 100 bytes**
Memory Size: 500000000 bytes (5mb)
Available : 500000000/100 = 5000000 blocks
Record Size: 18 bytes (tconst 10bytes(string), averageRating 4bytes(float), numVotes 4bytes(int))
Total number of records: 1070318
Number of record per block : 100(block size) / 18(record size)= 5 records(rounded down)
Number of blocks: 1070318/5 = 214064 blocks(rounded up)
Total record size: 19265724 bytes
Total block size: 214064*100 = 21406400 bytes
Remaining: 5000000 - 214064 = 4785936 blocks

**2- Block Size = 500 bytes**
Memory Size: 500000000 bytes (5mb)
Available: 500000000/500 = 1000000
Record Size: 18 bytes (tconst 10bytes(string), averageRating 4bytes(float), numVotes 4bytes(int))
Total number of records: 1070318
Number of record per block : 500(block size) / 18(record size)= 27(rounded down)
Number of blocks: 1070318/27 = 39642(rounded up)
Total record size: 19265724 bytes
Total block size: 39642*500 = 19821000 bytes
Remaining: 1000000 - 39642 = 960358 blocks

**b) The size of database:**

The size of database = sum of the size of the relational data (total record size) + sum of the size of index nodes of the B Plus tree (total index nodes * block size)

**1- Block Size = 100 bytes**

Size of database: 19265724 + 25 * 100 = 19268224

**2- Block Size = 500 bytes**

Size of database: 19265724 + 5 * 500 = 19268224

# Experiment 2

Build a B+ tree on the attribute "averageRating" by inserting the records sequentially and report the following statistics:

Each node of a B+ tree is stored as a block. An order for a B+ tree is the maximum branching factor which is defined as the maximum number of children a node may have. In this case, the maximum number of children corresponds to the maximum number of pointers in a node.

**a) The parameter n of the B+ tree**

Each key in a B+ tree node is a float which occupies 4 bytes.
As the program is running Java HotSpot(TM) 64-Bit Server Virtual machine (VM), each pointer (to a node of a B+ tree or to a data block) occupies 8 bytes.

**1- Block Size = 100 bytes**
Let k be the number of keys
4k + 8 * (k + 1) <= 100
k <= 7.67
k = 7

Hence, the maximum number of keys is 7 and the maximum number of pointers is 7+1=8 that could be stored in a node of the B+ tree.

**2- Block Size = 500 bytes**
Let k be the number of keys
4k + 8 * (k + 1) <= 500
k <= 41
k = 41

Hence, the maximum number of keys is 41 and the maximum number of pointers is 41+1=42 that could be stored in a node of the B+ tree.

**b) The number of nodes of the B+ tree**

**1- Block Size = 100 bytes**

| | **Nodes** |
|---|---|
| **Root (Level 1):** | 1 |
| **Internal (Level 2):** | 4 |
| **Leaf (Level 3):** | 20 |

Total number of nodes of the B+ tree: 25

**2- Block Size = 500 bytes**

| | Nodes |
|---|---|
| **Root (Level 1):** | 1 |
| **Leaf (Level 2):** | 4 |

Total number of nodes of the B+ tree: 5

**c) The height of the B+ tree, i.e., the number of levels of the B+ tree**

The leaf nodes must have at least 1,070,318 record pointers (excluding sequential pointers). Sequential pointers are the last pointers of each leaf node, their last pointers point to the first pointers of the adjacently connected next leaf node. Dense index is used for the lowest level at leaf nodes, which signifies that 1 pointer points to 1 record.

There are 91 unique keys for the float attribute "averageRating". The data structure of our B+ tree is built in such a way that all of the duplicate keys of attribute "averageRating" are stored contiguously together in the leaf nodes as an array list.

**1- Block Size = 100 bytes**
The maximum number of keys of an internal node: 7
The maximum number of keys of a leaf node: 7

Let k be the number of keys
Let n be the number of levels of the B+ tree

91 <= k * (k + 1) ^ (n-1)
91 <= 7 * (8) ^ (n-1)
13 <= (8) ^ (n-1)
log(13) <= log((8) ^ (n-1))
log(13) <= (n-1) * log(8)
(n-1) >= log(13) / log(8)
n >= 2.233479906
n = 3, hence the B+ tree has 3 levels.

**2- Block Size = 500 bytes**
The maximum number of keys of an internal node: 41
The maximum number of keys of a leaf node: 41

Let k be the number of keys
Let n be the number of levels of the B+ tree

91 <= k * (k + 1) ^ (n-1)
91 <= 41 * (42) ^ (n-1)
2.219512195 <= (42) ^ (n-1)
log(2.219512195) <= log((42) ^ (n-1))
log(2.219512195) <= (n-1) * log(42)
(n-1) >= log(2.219512195) / log(42)
n >= 1.213311373
n = 2, hence the B+ tree has 2 levels.

**d) The root node and its child nodes (actual content)**

**The format of the output in the root and internal node is:**
Key:(Total Number of Records);
Ex) 3.2:(Empty);

As the root and internal node only contains the keys and pointers, the number of records are empty. The semicolon (;) denotes a pointer.

**The format of the output in the leaf node is:**
Key:(Total Number of Records);
Ex) 9.6:(2833);

As the leaf node contains the keys, pointers and duplicate records, the number of records are shown clearly. The semicolon (;) denotes a pointer.

**<u>1- Block Size = 100 bytes</u>**
**Console Output:**
**Printing level 1 (Root)**
3.2:();5.4:();7.5:();||
**Printing level 2**
1.6:();2.0:();2.4:();2.8:();||3.6:();4.0:();4.4:();5.0:();||5.9:();6.6:();7.1:();||8.0:();8.4:();8.8:();9.2:();9.6:();||
**Printing level 3**
1.0:(946);1.1:(250);1.2:(294);1.3:(276);1.4:(322);1.5:(318);||1.6:(473);1.7:(397);1.8:(560);1.9:(457);||2.0:(726);2.1:(697);2.2:(862);2.3:(858);||2.4:(1108);2.5:(1039);2.6:(1185);2.7:(1111);||2.8:(1631);2.9:(1286);3.0:(1799);3.1:(1563);||3.2:(2087);3.3:(1877);3.4:(2353);3.5:(2392);||3.6:(2956);3.7:(2760);3.8:(3570);3.9:(2993);||4.0:(4463);4.1:(3623);4.2:(5180);4.3:(4595);||4.4:(5632);4.5:(5399);4.6:(7037);4.7:(6511);4.8:(8765);4.9:(7131);||5.0:(10461);5.1:(8891);5.2:(12277);5.3:(10910);||5.4:(13305);5.5:(12514);5.6:(15400);5.7:(14646);5.8:(19331);||5.9:(16077);6.0:(22363);6.1:(19410);6.2:(25968);6.3:(22788);6.4:(26847);6.5:(24969);||6.6:(29200);6.7:(26913);6.8:(34628);6.9:(28134);7.0:(36626);||7.1:(30461);7.2:(38190);7.3:(32150);7.4:(35764);||7.5:(32012);7.6:(36460);7.7:(32329);7.8:(36607);7.9:(28376);||8.0:(33429);8.1:(26115);8.2:(29087);8.3:(21064);||8.4:(20196);8.5:(16799);8.6:(16522);8.7:(13177);||8.8:(12902);8.9:(8572);9.0:(9057);9.1:(5637);||9.2:(6390);9.3:(3828);9.4:(3724);9.5:(2312);||9.6:(2833);9.7:(1651);9.8:(1885);9.9:(623);10.0:(3026);||

Total number of nodes in B+ tree is: 25
Total number of records in B+ tree is: 1070318

**2- Block Size = 500 bytes**
**Console Output:**

**Printing level 1 (Root)**
3.4:();5.5:();7.6:();||
**Printing level 2**
1.0:(946);1.1:(250);1.2:(294);1.3:(276);1.4:(322);1.5:(318);1.6:(473);1.7:(397);1.8:(560);1.9:(457);2.0:(726);2.1:(697);2.2:(862);2.3:(858);2.4:(1108);2.5:(1039);2.6:(1185);2.7:(1111);2.8:(1631);2.9:(1286);3.0:(1799);3.1:(1563);3.2:(2087);3.3:(1877);||3.4:(2353);3.5:(2392);3.6:(2956);3.7:(2760);3.8:(3570);3.9:(2993);4.0:(4463);4.1:(3623);4.2:(5180);4.3:(4595);4.4:(5632);4.5:(5399);4.6:(7037);4.7:(6511);4.8:(8765);4.9:(7131);5.0:(10461);5.1:(8891);5.2:(12277);5.3:(10910);5.4:(13305);||5.5:(12514);5.6:(15400);5.7:(14646);5.8:(19331);5.9:(16077);6.0:(22363);6.1:(19410);6.2:(25968);6.3:(22788);6.4:(26847);6.5:(24969);6.6:(29200);6.7:(26913);6.8:(34628);6.9:(28134);7.0:(36626);7.1:(30461);7.2:(38190);7.3:(32150);7.4:(35764);7.5:(32012);||7.6:(36460);7.7:(32329);7.8:(36607);7.9:(28376);8.0:(33429);8.1:(26115);8.2:(29087);8.3:(21064);8.4:(20196);8.5:(16799);8.6:(16522);8.7:(13177);8.8:(12902);8.9:(8572);9.0:(9057);9.1:(5637);9.2:(6390);9.3:(3828);9.4:(3724);9.5:(2312);9.6:(2833);9.7:(1651);9.8:(1885);9.9:(623);10.0:(3026);||

Total number of nodes in B+ tree is: 5
Total number of records in B+ tree is: 1070318

# Experiment 3

Retrieve the attribute “tconst” of those movies with the
“averageRating” equal to 8 and report the following statistics:

**a) The number and the content of index nodes the process accesses**

**1- Block Size = 100 bytes**

The total number of Index Nodes Access: 3
The content are:
Index Node Access: Node= [<KEY>[Key= 3.2] , <KEY>[Key= 5.4] , <KEY>[Key= 7.5] ]
Index Node Access: Node= [<KEY>[Key= 8.0] , <KEY>[Key= 8.4] , <KEY>[Key= 8.8] , <KEY>[Key= 9.2] , <KEY>[Key= 9.6] ]
Index Node Access: Node= [<KEY>[Key= 8.0] , <KEY>[Key= 8.1] , <KEY>[Key= 8.2] , <KEY>[Key= 8.3] ]

**2- Block Size = 500 bytes**

The total number of Index Nodes Access: 2
The content are:
Index Node Access: Node= [<KEY>[Key= 3.4] , <KEY>[Key= 5.5] , <KEY>[Key= 7.6] ]
Index Node Access: Node= [<KEY>[Key= 7.6] , <KEY>[Key= 7.7] , <KEY>[Key= 7.8] , <KEY>[Key= 7.9] , <KEY>[Key= 8.0] , <KEY>[Key= 8.1] , <KEY>[Key= 8.2] , <KEY>[Key= 8.3] , <KEY>[Key= 8.4] , <KEY>[Key= 8.5] , <KEY>[Key= 8.6] , <KEY>[Key= 8.7] , <KEY>[Key= 8.8] , <KEY>[Key= 8.9] , <KEY>[Key= 9.0] , <KEY>[Key= 9.1] , <KEY>[Key= 9.2] , <KEY>[Key= 9.3] , <KEY>[Key= 9.4] , <KEY>[Key= 9.5] , <KEY>[Key= 9.6] , <KEY>[Key= 9.7] , <KEY>[Key= 9.8] , <KEY>[Key= 9.9] , <KEY>[Key= 10.0] ]

**b) The number and the content of data blocks the process accesses**

**1- Block Size = 100 bytes**

The total number of Data Blocks Access: 1
The content are:

```
Data Block Access: Key=8.0
Value Size=33429 Records
Value (0)=Record@6615435c
```

**2- Block Size = 500 bytes**

The total number of Data Blocks Access: 1
The content are:

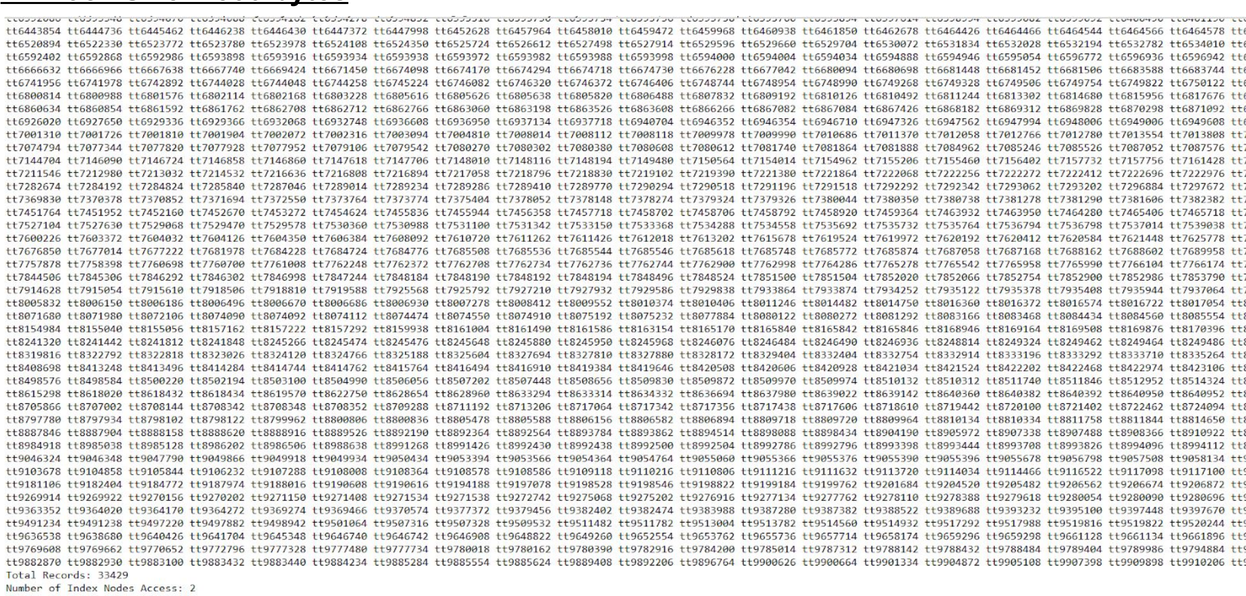
```
Data Block Access: Key=8.0
Value Size=33429 Records
Value (0)=Record@1936f0f5
```

### c) The attribute “tconst” of the records that are returned

The total number of records that contain averageRating equal to 8 is 33429 records.

**<u>1- Block Size = 100 bytes</u>**


**<u>2- Block Size = 500 bytes</u>**



# <u>Experiment 4</u>

Retrieve the attribute “tconst” of those movies with the
attribute “averageRating” from 7 to 9, both inclusively and report the
following statistics:

**a) The number and the content of index nodes the process accesses**

**<u>1- Block Size = 100 bytes</u>**

The total number of Index Nodes Access: 3
The content are:
Index Node Access: Node= [<KEY>[Key= 3.2] , <KEY>[Key= 5.4] , <KEY>[Key= 7.5] ]
Index Node Access: Node= [<KEY>[Key= 3.2] , <KEY>[Key= 5.4] , <KEY>[Key= 7.5] ]
Index Node Access: Node= [<KEY>[Key= 5.9] , <KEY>[Key= 6.6] , <KEY>[Key= 7.1] ]

**<u>2- Block Size = 500 bytes</u>**

The total number of Index Nodes Access: 2
The content are:
Index Node Access: Node= [<KEY>[Key= 3.4] , <KEY>[Key= 5.5] , <KEY>[Key= 7.6] ]
Index Node Access: Node= [<KEY>[Key= 3.4] , <KEY>[Key= 5.5] , <KEY>[Key= 7.6] ]

**b) The number and the content of data blocks the process accesses**

**1- Block Size = 100 bytes**

The total number of Data Block Access: 26
The content are:

```
Value (0)= Record@3cbbc1e0
Data Block Access: Key= 7.7
Value Size= 32329 Records
Value (0)= Record@35fb3008
Data Block Access: Key= 7.8
Value Size= 36607 Records
Value (0)= Record@7225790e
Data Block Access: Key= 7.9
Value Size= 28376 Records
Value (0)= Record@54a097cc
Data Block Access: Key= 8.0
Value Size= 33429 Records
Value (0)= Record@36f6e879
Data Block Access: Key= 8.1
Value Size= 26115 Records
Value (0)= Record@5a61f5df
Data Block Access: Key= 8.2
Value Size= 29087 Records
Value (0)= Record@3551a94
Data Block Access: Key= 8.3
Value Size= 21064 Records
Value (0)= Record@531be3c5
Data Block Access: Key= 8.4
Value Size= 20196 Records
Value (0)= Record@52af6cff
Data Block Access: Key= 8.5
Value Size= 16799 Records
Value (0)= Record@735b478
Data Block Access: Key= 8.6
Value Size= 16522 Records
Value (0)= Record@2c9f9fb0
Data Block Access: Key= 8.7
Value Size= 13177 Records
Value (0)= Record@2096442d
Data Block Access: Key= 8.8
Value Size= 12902 Records
Value (0)= Record@9f70c54
Data Block Access: Key= 8.9
Value Size= 8572 Records
Value (0)= Record@234bef66
Data Block Access: Key= 9.0
Value Size= 9057 Records
Value (0)= Record@737996a0
Data Block Access: Key= 9.1
Value Size= 5637 Records
Value (0)= Record@61dc03ce
```

**2- Block Size = 500 bytes**

The total number of Data Block Access: 46
The content are:

```
Value Size= 13177 Records
Value (0)= Record@97e1986
Data Block Access: Key= 8.8
Value Size= 12902 Records
Value (0)= Record@26f67b76
Data Block Access: Key= 8.9
Value Size= 8572 Records
Value (0)= Record@153f5a29
Data Block Access: Key= 9.0
Value Size= 9057 Records
Value (0)= Record@7f560810
Data Block Access: Key= 9.1
Value Size= 5637 Records
Value (0)= Record@69d9c55
Data Block Access: Key= 9.2
Value Size= 6390 Records
Value (0)= Record@13a57a3b
Data Block Access: Key= 9.3
Value Size= 3828 Records
Value (0)= Record@7ca48474
Data Block Access: Key= 9.4
Value Size= 3724 Records
Value (0)= Record@337d0578
Data Block Access: Key= 9.5
Value Size= 2312 Records
Value (0)= Record@59e84876
Data Block Access: Key= 9.6
Value Size= 2833 Records
Value (0)= Record@61a485d2
Data Block Access: Key= 9.7
Value Size= 1651 Records
Value (0)= Record@39fb3ab6
Data Block Access: Key= 9.8
Value Size= 1885 Records
Value (0)= Record@6276ae34
Data Block Access: Key= 9.9
Value Size= 623 Records
Value (0)= Record@7946e1f4
Data Block Access: Key= 10.0
Value Size= 3026 Records
Value (0)= Record@3c09711b
```

### c) The attribute “tconst” of the records that are returned

Total Records with the attribute “averageRating” from 7 to 9: 545,895

**1- Block Size = 100 bytes**


**2- Block Size = 500 bytes**

# Experiment 5

**a) The number of times that a node is deleted (or two nodes are merged) during the process of the updating the B+ tree**

**1- Block Size = 100 bytes**

The total number of deleted nodes is 0
The total number of merged nodes is 0

**2- Block Size = 500 bytes**

The total number of deleted nodes is 0
The total number of merged nodes is 0

**b) The height of the updated B+ tree**

**1- Block Size = 100 bytes**

The tree height is 3.

**2- Block Size = 500 bytes**

The tree height is 2.

**c) The root node and its child nodes of the updated B+ tree**

**1- Block Size = 100 bytes**
**Console Output:**
**Printing level 1 (Root)**
3.2:();5.4:();7.5:();||
**Printing level 2**
1.6:();2.0:();2.4:();2.8:();||3.6:();4.0:();4.4:();5.0:();||5.9:();6.6:();7.1:();||8.0:();8.4:();8.8:();9.2:();9.6:();||
**Printing level 3**
1.0:(946);1.1:(250);1.2:(294);1.3:(276);1.4:(322);1.5:(318);||1.6:(473);1.7:(397);1.8:(560);1.9:(457);||2.0:(726);2.1:(697);2.2:(862);2.3:(858);||2.4:(1108);2.5:(1039);2.6:(1185);2.7:(1111);||2.8:(1631);2.9:(1286);3.0:(1799);3.1:(1563);||3.2:(2087);3.3:(1877);3.4:(2353);3.5:(2392);||3.6:(2956);3.7:(2760);3.8:(3570);3.9:(2993);||4.0:(4463);4.1:(3623);4.2:(5180);4.3:(4595);||4.4:(5632);4.5:(5399);4.6:(7037);4.7:(6511);4.8:(8765);4.9:(7131);||5.0:(10461);5.1:(8891);5.2:(12277);5.3:(10910);||5.4:(13305);5.5:(12514);5.6:(15400);5.7:(14646);5.8:(19331);||5.9:(16077);6.0:(22363);6.1:(19410);6.2:(25968);6.3:(22788);6.4:(26847);6.5:(24969);||6.6:(29200);6.7:(26913);6.8:(34628);6.9:(28

134);||7.1:(30461);7.2:(38190);7.3:(32150);7.4:(35764);||7.5:(32012);7.6:(36460);7.7:(32329);7.8:(36607);7.9:(28376);||8.0:(33429);8.1:(26115);8.2:(29087);8.3:(21064);||8.4:(20196);8.5:(16799);8.6:(16522);8.7:(13177);||8.8:(12902);8.9:(8572);9.0:(9057);9.1:(5637);||9.2:(6390);9.3:(3828);9.4:(3724);9.5:(2312);||9.6:(2833);9.7:(1651);9.8:(1885);9.9:(623);10.0:(3026);||

Total number of nodes in B+ tree is: 25
Total number of records in B+ tree is: 1033692

**<u>2- Block Size = 500 bytes</u>**

**Console Output:**

**Printing level 1 (Root)**

3.4:();5.5:();7.6:();||

**Printing level 2**

1.0:(946);1.1:(250);1.2:(294);1.3:(276);1.4:(322);1.5:(318);1.6:(473);1.7:(397);1.8:(560);1.9:(457);2.0:(726);2.1:(697);2.2:(862);2.3:(858);2.4:(1108);2.5:(1039);2.6:(1185);2.7:(1111);2.8:(1631);2.9:(1286);3.0:(1799);3.1:(1563);3.2:(2087);3.3:(1877);||3.4:(2353);3.5:(2392);3.6:(2956);3.7:(2760);3.8:(3570);3.9:(2993);4.0:(4463);4.1:(3623);4.2:(5180);4.3:(4595);4.4:(5632);4.5:(5399);4.6:(7037);4.7:(6511);4.8:(8765);4.9:(7131);5.0:(10461);5.1:(8891);5.2:(12277);5.3:(10910);5.4:(13305);||5.5:(12514);5.6:(15400);5.7:(14646);5.8:(19331);5.9:(16077);6.0:(22363);6.1:(19410);6.2:(25968);6.3:(22788);6.4:(26847);6.5:(24969);6.6:(29200);6.7:(26913);6.8:(34628);6.9:(28134);7.1:(30461);7.2:(38190);7.3:(32150);7.4:(35764);7.5:(32012);||7.6:(36460);7.7:(32329);7.8:(36607);7.9:(28376);8.0:(33429);8.1:(26115);8.2:(29087);8.3:(21064);8.4:(20196);8.5:(16799);8.6:(16522);8.7:(13177);8.8:(12902);8.9:(8572);9.0:(9057);9.1:(5637);9.2:(6390);9.3:(3828);9.4:(3724);9.5:(2312);9.6:(2833);9.7:(1651);9.8:(1885);9.9:(623);10.0:(3026);||

Total number of nodes in B+ tree is: 5
Total number of records in B+ tree is: 1033692