NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CE/CZ4031
Database Systems Principles
Project 1 Report

Group Members:

Name	Matriculation Number
Goh Hong Xiang Bryan	U1920609E
Lee Cheng Han	U1920206L
Chong Jing Hong	U1922300B
Terry Joel Ee Wen Jie	U1922131D

Table of contents

1. Overview

This project is done using C# and the program is assumed to be running on a 64-bit OS.

Instructions:

1. Change Line 17 in Program.cs to the desired path. For instance, "C:\\Users\\User\\OneDrive\\Desktop\\DSP_New_2\\Project 1\\Project 1\\data.tsv".
2. Click on run project (For vscode users, type dotnet run in the terminal).
3. Enter any key to start project.
4. Enter size of block which can be either 100 or 500.
5. After that, experiment 1 to 5 will execute by itself.

2. Design of storage components

2.1. How each data is stored as a field

The data structure is as follows: tconst::string (the unique identifier for the movie), averageRating::double (weighted average of all the individual user ratings) and numVotes::int (number of votes the title has received). This is how our group decided to store each field:

- For the tconst string, we split the string into 10-character arrays to store it as a field. This is because after analyzing the data, we found that the maximum length of the tconst string is 10 characters (tt10001184). Hence, we used a character array of length 10 to ensure that this field is of fixed length for all data items. In any case if the string is less than the length of 10, the remaining character arrays will be instantiated with null values.

- For the averageRating double, we used a double to store the field as each averageRating field contains a decimal value up to 1 decimal place.

- For the numVotes int, the largest numVotes value we could find in the dataset has a length of 7 characters. Since in C#, the range for the *int* class is -2,147,483,648 to 2,147,483,647, we decided to use the *int* class to store the numVotes field.

- We have an additional attribute, blockID, to store the id of the block where the record is contained. This attribute is also of the class *int*.

2.2. How fields are packed into a record

Fields are packed into a record through Record class (Record.cs). Each record contains tConst, averageRating, numVotes and blockID with a fixed length / fixed number of bytes. Hence the records can be represented with a fixed number of bytes making it easier to interpret although some space is wasted in the process.

Below are the considerations of how the minimum number of bytes are determined for each field(s).

2.2.1. Tconst

In C#, the number of bytes for a character variable is 2 bytes. Hence with 10-character arrays, the total number of bytes represented by tconst is:

10 * 2 = 20 bytes

2.2.2. AverageRating

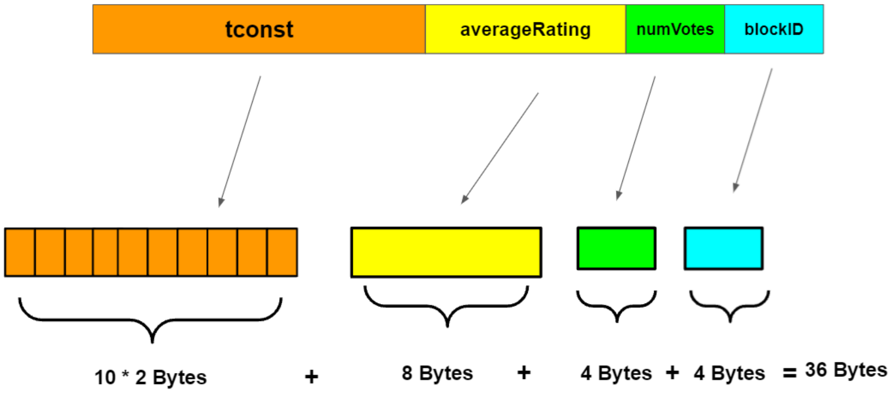
In C#, the number of bytes for a double variable is 8 bytes.

2.2.3. NumVotes

In C#, the number of bytes for an integer variable is 4 bytes.

2.2.4. BlockID

In C#, the number of bytes for an integer variable is 4 bytes.



20 + 8 + 4 + 4 = 36 bytes, hence the total number of bytes represented by each fixed length record is **36 bytes**.

Below illustrates an example of a record:

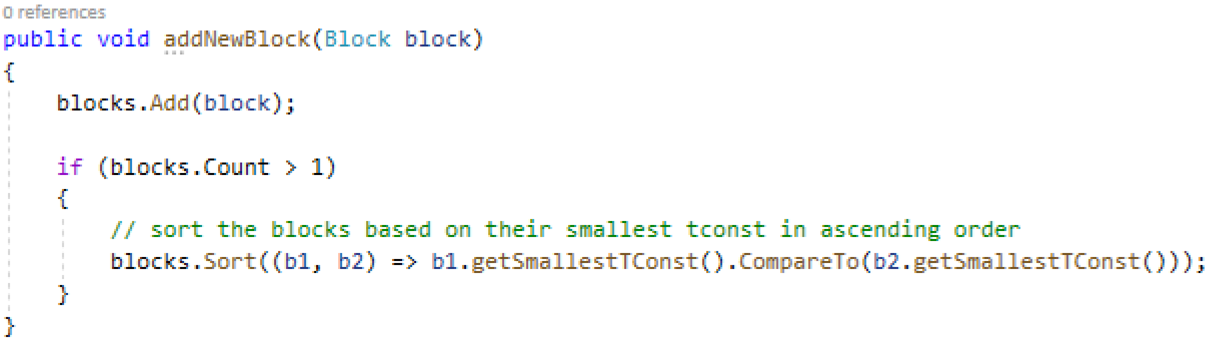
tConst	averageRating	numVotes	blockID
tt0000001	5.6	1645	1

2.3 How records are packed into a block

Since all records have a fixed length of 36 bytes, there is not a need to separate the records. The records will be stored sequentially on the block.

Using the addNewBlock function as shown below, a new block would be added whenever the block size limit has been reached.

For example, if the block size is 100B, the block can only hold up to 2 records and if the block size is 500B, the block can only hold up to 13 records.



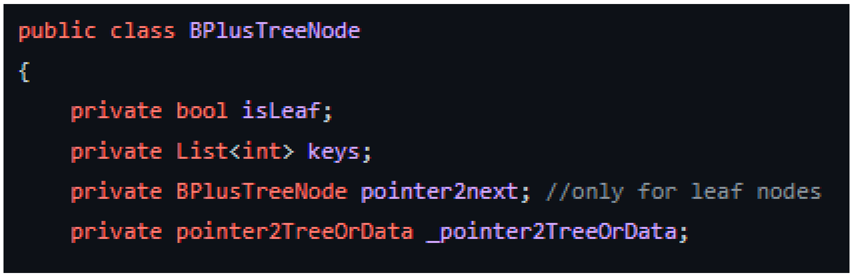
```
0 references
public void addNewBlock(Block block)
{
    blocks.Add(block);

    if (blocks.Count > 1)
    {
        // sort the blocks based on their smallest tconst in ascending order
        blocks.Sort((b1, b2) => b1.getSmallestTConst().CompareTo(b2.getSmallestTConst()));
    }
}
```

3. Design of B+ Tree

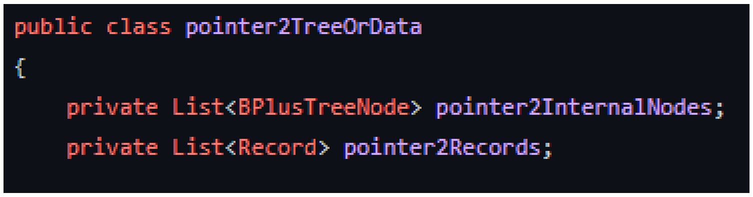
3.1. B+ Tree Node

There are two types of nodes in a B+ Tree, internal (non-leaf) nodes and leaf nodes. We considered building a base node class as well as two other derived classes for internal and leaf nodes. However, we decided on using a class that caters to both internal and leaf nodes because they share a majority of attributes and have similar behavior.

Object reference takes up 8 bytes (64 bits on a 64-bit OS) since the reference acts as a pointer. The object reference includes the boolean value isLeaf and the List<int> keys. Thus, each node can store a maximum of 12 object references in a 100 bytes block and 62 object references in 500 bytes block respectively.



```
public class BPlusTreeNode
{
    private bool isLeaf;
    private List<int> keys;
    private BPlusTreeNode pointer2next; //only for leaf nodes
    private pointer2TreeOrData _pointer2TreeOrData;
```



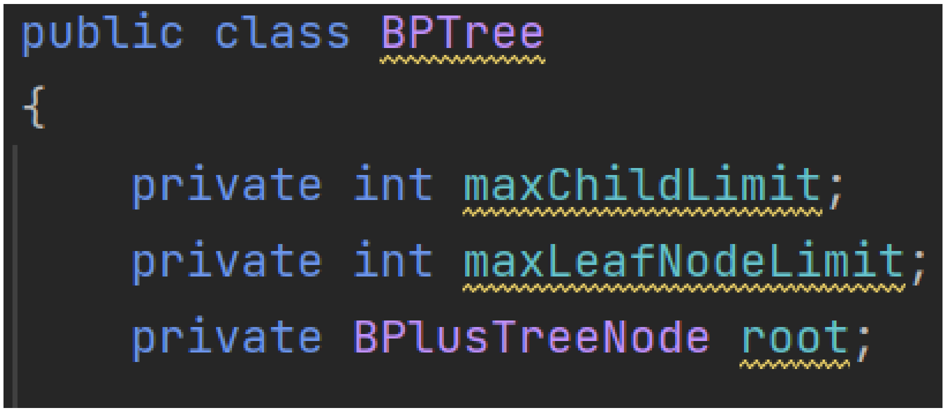
```
public class pointer2TreeOrData
{
    private List<BPlusTreeNode> pointer2InternalNodes;
    private List<Record> pointer2Records;
```

In our design, B+ Tree Node consists of the following attributes:

- Bool isLeaf
  - This allows us to determine if the node is an internal node or a leaf node.
- List<int> keys
  - Each node contains a set of keys. Since the key in our node is the number of votes that the movie received, we used a list of integers to store the various values.
- BPlusTreeNode pointer2Next
  - This attribute only applies to leaf nodes. It acts as a pointer to the right sibling of a leaf node. This forms a chain across the whole B+ Tree, such that all the leaf nodes are connected.

- pointer2TreeOrData _pointer2TreeOrData
    - Since a node can either point to another node or point to the record data, we created another class with two attributes to account for both scenarios.
    - pointer2InternalNodes contains a list of child nodes belonging to the initial node.
    - pointer2Records contains a list of records belonging to the initial node.

3.2. B+ Tree



```
public class BPTree
{
    private int maxChildLimit;
    private int maxLeafNodeLimit;
    private BPlusTreeNode root;
```

For the B+ Tree itself, we used the following attributes:

- int maxChildLimit
    - This attribute stores the maximum number of child nodes an internal node could have. This also refers to the number of pointers per internal node.
- int maxLeafNodeLimit
    - This attribute stores the maximum number of pointers, excluding the ones that point to sibling leaf nodes, a leaf node could have.
- BPlusTreeNode root
    - This attribute stores the root node of the tree.

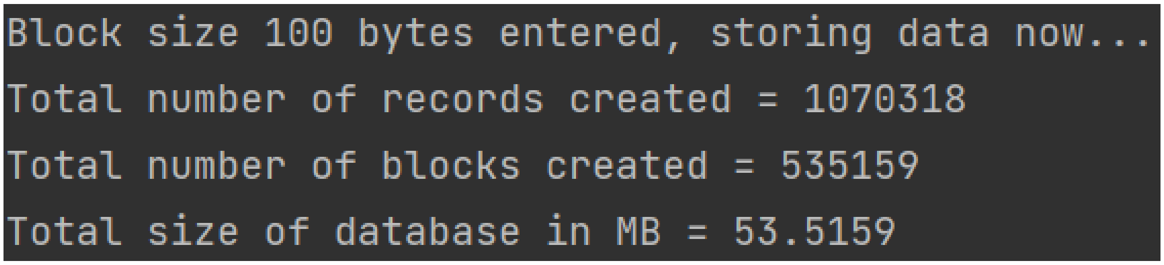
4. Results of the experiments

4.1. Experiment 1

Store the data (which is about IMDB movies and described in Part 4) on the disk (as specified in Part 1) and report the following statistics:

For block size = **100 bytes**:

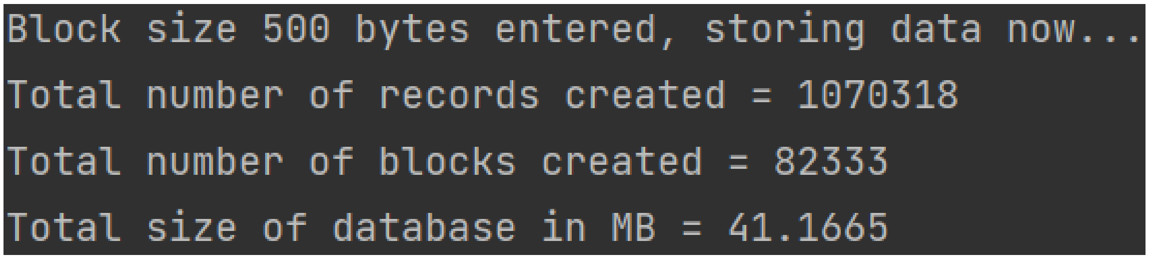
- the number of blocks: **535159**
- the size of the database (in terms of MB): **53.5159**



```
Block size 100 bytes entered, storing data now...
Total number of records created = 1070318
Total number of blocks created = 535159
Total size of database in MB = 53.5159
```

For block size = **500 bytes**:

- the number of blocks: **82333**
- the size of the database (in terms of MB): **41.1665**



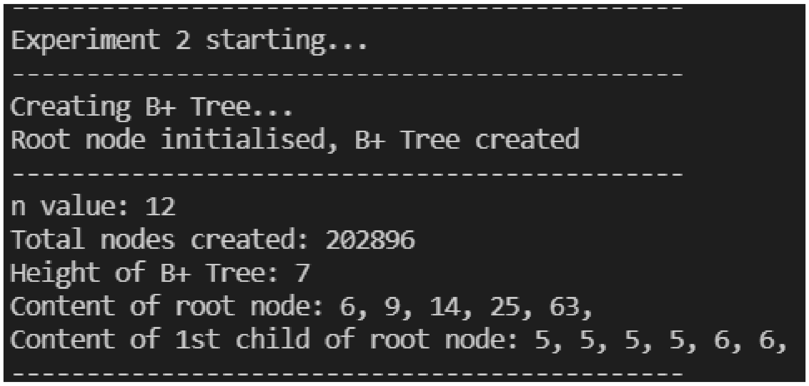
```
Block size 500 bytes entered, storing data now...
Total number of records created = 1070318
Total number of blocks created = 82333
Total size of database in MB = 41.1665
```

4.2 Experiment 2

Build a B+ Tree on the attribute “numVotes” by inserting the records sequentially and report on the following statistics:

For block size = **100 bytes**:

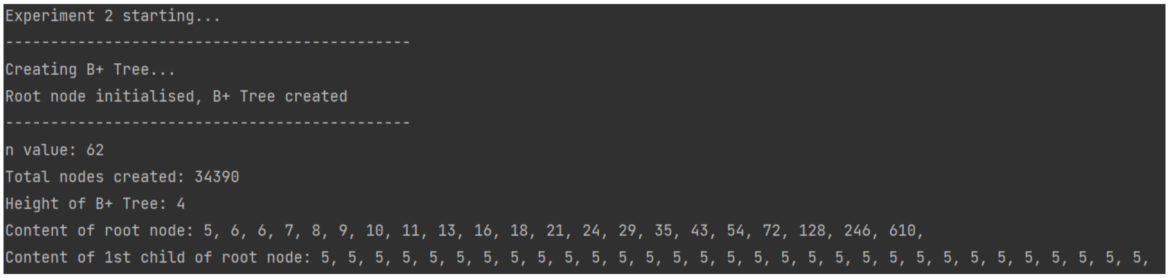
- the parameter n of the B+ Tree:
- Each object reference is 8 bytes in size, the parameter n = **12**
- the number nodes of the B+ Tree:
- the height of the B+ Tree, i.e. the number of levels of the B+ Tree:
- the content of the root node and its 1st child node:



```
---------------------------------------------
Experiment 2 starting...
---------------------------------------------
Creating B+ Tree...
Root node initialised, B+ Tree created
---------------------------------------------
n value: 12
Total nodes created: 202896
Height of B+ Tree: 7
Content of root node: 6, 9, 14, 25, 63,
Content of 1st child of root node: 5, 5, 5, 5, 6, 6,
---------------------------------------------
```

For block size = **500 bytes**:

- the parameter n of the B+ Tree:
- Each object reference is 8 bytes in size, the parameter n = **62**
- the number nodes of the B+ Tree:
- the height of the B+ Tree, i.e. the number of levels of the B+ Tree:
- the content of the root node and its 1st child node:



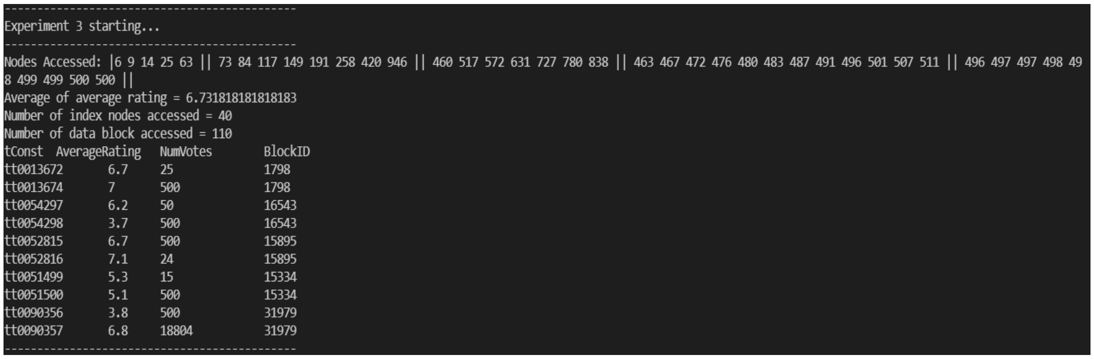
```
Experiment 2 starting...
---------------------------------------------
Creating B+ Tree...
Root node initialised, B+ Tree created
---------------------------------------------
n value: 62
Total nodes created: 34390
Height of B+ Tree: 4
Content of root node: 5, 6, 6, 7, 8, 9, 10, 11, 13, 16, 18, 21, 24, 29, 35, 43, 54, 72, 128, 246, 610,
Content of 1st child of root node: 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
```

4.3 Experiment 3

Retrieve those movies with the “numVotes” equal to 500 and report the following statistics:

For block size = **100 bytes**:

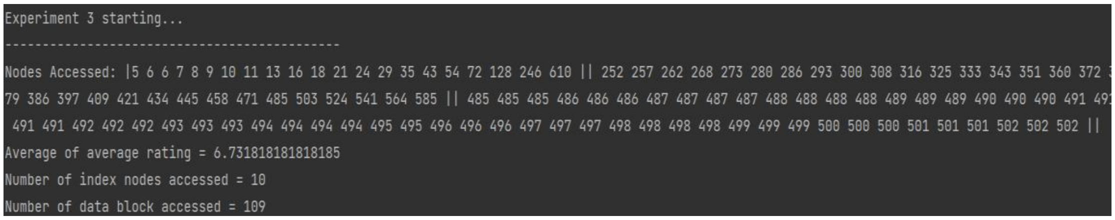
- the number and the content of index nodes the process accesses:
- the number and the content of data blocks the process accesses:
- the average of “averageRatings” of the records that are returned:



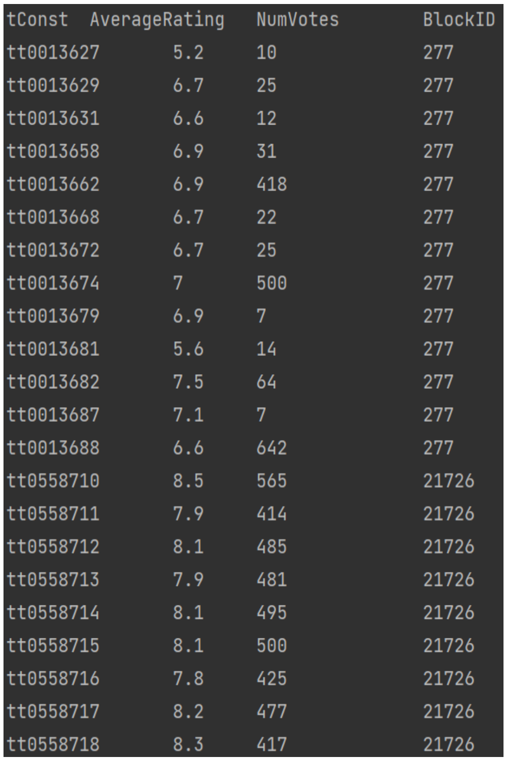
```
---------------------------------------------
Experiment 3 starting...
---------------------------------------------
Nodes Accessed: |6 9 14 25 63 || 73 84 117 149 191 258 420 946 || 460 517 572 631 727 780 838 || 463 467 472 476 480 483 487 491 496 501 507 511 || 496 497 497 498 49
8 499 499 500 500 ||
Average of average rating = 6.731818181818183
Number of index nodes accessed = 40
Number of data block accessed = 110
tConst  AverageRating   NumVotes        BlockID
tt0013672       6.7     25              1798
tt0013674       7       500             1798
tt0054297       6.2     50              16543
tt0054298       3.7     500             16543
tt0052815       6.7     500             15895
tt0052816       7.1     24              15895
tt0051499       5.3     15              15334
tt0051500       5.1     500             15334
tt0090356       3.8     500             31979
tt0090357       6.8     18804           31979
---------------------------------------------
```

For block size = **500 bytes**:

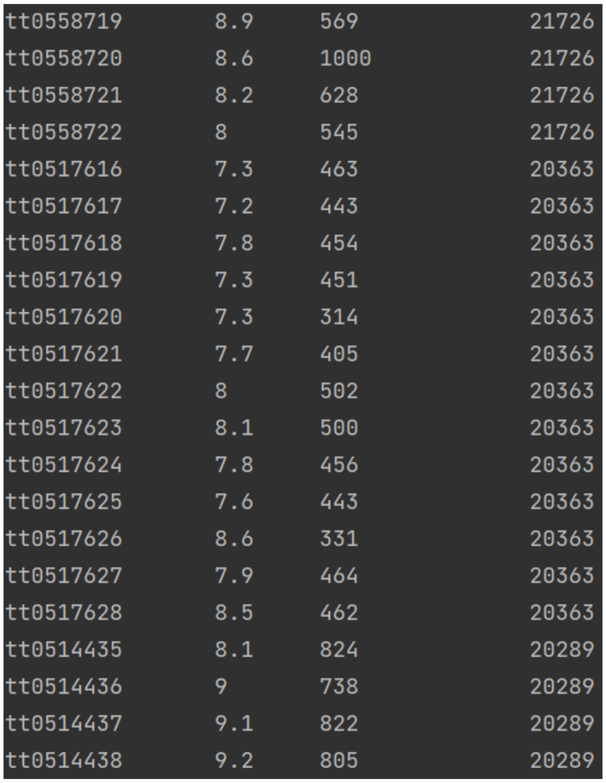
- the number and the content of index nodes the process accesses:
- the number and the content of data blocks the process accesses:
- the average of “averageRatings” of the records that are returned:



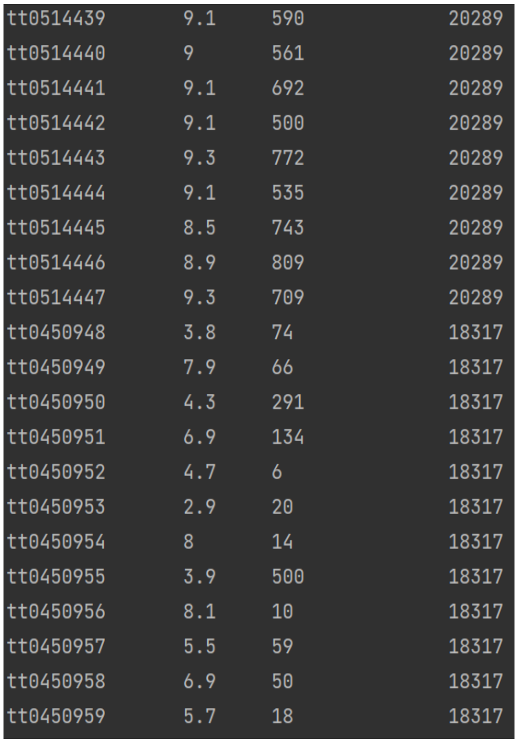
```
Experiment 3 starting...
---------------------------------------------
Nodes Accessed: |5 6 6 7 8 9 10 11 13 16 18 21 24 29 35 43 54 72 128 246 610 || 252 257 262 268 273 280 286 293 300 308 316 325 333 343 351 360 372 3
79 386 397 409 421 434 445 458 471 485 503 524 541 564 585 || 485 485 485 486 486 486 487 487 487 487 488 488 488 488 489 489 489 490 490 490 491 49
 491 491 492 492 492 493 493 493 494 494 494 494 495 495 496 496 496 497 497 497 498 498 498 498 499 499 499 500 500 500 501 501 501 502 502 502 ||
Average of average rating = 6.731818181818185
Number of index nodes accessed = 10
Number of data block accessed = 109
```



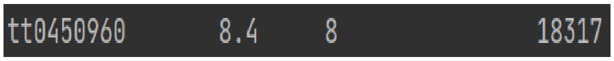
```
tConst  AverageRating   NumVotes        BlockID
tt0013627       5.2     10              277
tt0013629       6.7     25              277
tt0013631       6.6     12              277
tt0013658       6.9     31              277
tt0013662       6.9     418             277
tt0013668       6.7     22              277
tt0013672       6.7     25              277
tt0013674       7       500             277
tt0013679       6.9     7               277
tt0013681       5.6     14              277
tt0013682       7.5     64              277
tt0013687       7.1     7               277
tt0013688       6.6     642             277
tt0558710       8.5     565             21726
tt0558711       7.9     414             21726
tt0558712       8.1     485             21726
tt0558713       7.9     481             21726
tt0558714       8.1     495             21726
tt0558715       8.1     500             21726
tt0558716       7.8     425             21726
tt0558717       8.2     477             21726
tt0558718       8.3     417             21726
```



```
tt0558719       8.9     569             21726
tt0558720       8.6     1000            21726
tt0558721       8.2     628             21726
tt0558722       8       545             21726
tt0517616       7.3     463             20363
tt0517617       7.2     443             20363
tt0517618       7.8     454             20363
tt0517619       7.3     451             20363
tt0517620       7.3     314             20363
tt0517621       7.7     405             20363
tt0517622       8       502             20363
tt0517623       8.1     500             20363
tt0517624       7.8     456             20363
tt0517625       7.6     443             20363
tt0517626       8.6     331             20363
tt0517627       7.9     464             20363
tt0517628       8.5     462             20363
tt0514435       8.1     824             20289
tt0514436       9       738             20289
tt0514437       9.1     822             20289
tt0514438       9.2     805             20289
```



```
tt0514439       9.1     590             20289
tt0514440       9       561             20289
tt0514441       9.1     692             20289
tt0514442       9.1     500             20289
tt0514443       9.3     772             20289
tt0514444       9.1     535             20289
tt0514445       8.5     743             20289
tt0514446       8.9     809             20289
tt0514447       9.3     709             20289
tt0450948       3.8     74              18317
tt0450949       7.9     66              18317
tt0450950       4.3     291             18317
tt0450951       6.9     134             18317
tt0450952       4.7     6               18317
tt0450953       2.9     20              18317
tt0450954       8       14              18317
tt0450955       3.9     500             18317
tt0450956       8.1     10              18317
tt0450957       5.5     59              18317
tt0450958       6.9     50              18317
tt0450959       5.7     18              18317
```



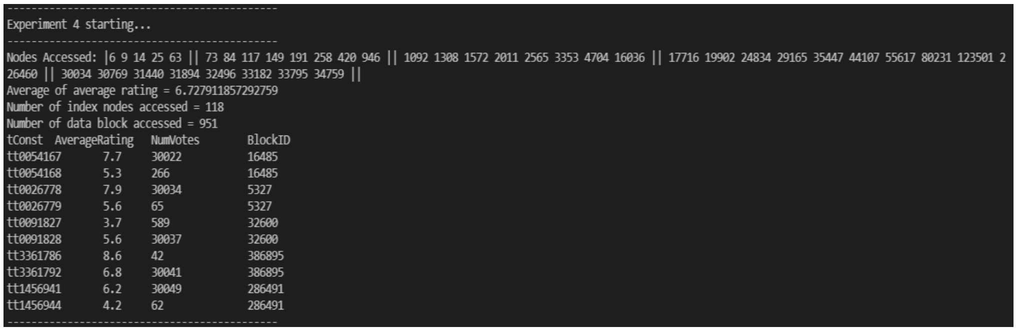
```
tt0450960       8.4     8               18317
```

4.4. Experiment 4

Retrieve those movies with the attribute “numVotes” from 30,000 to 40,000, both inclusively and report the following statistics:

For block size = **100 bytes**:

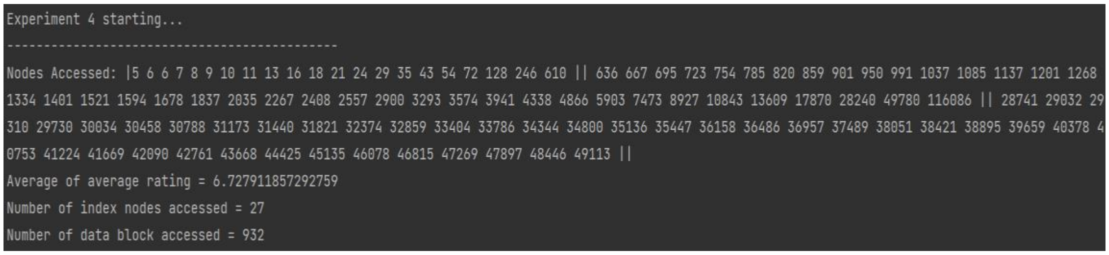
- the number and the content of index nodes the process accesses:
- the number and the content of data blocks the process accesses:
- the average of “averageRatings” of the records that are returned:



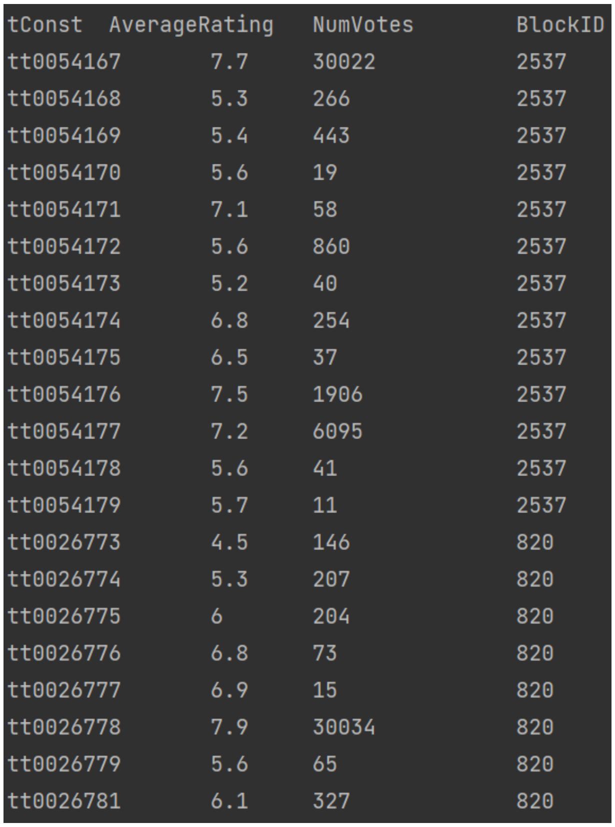
```
--------------------------------------------
Experiment 4 starting...
--------------------------------------------
Nodes Accessed: |6 9 14 25 63 || 73 84 117 149 191 258 420 946 || 1092 1308 1572 2011 2565 3353 4704 16036 || 17716 19902 24834 29165 35447 44107 55617 80231 123501 2
26460 || 30034 30769 31440 31894 32496 33182 33795 34759 ||
Average of average rating = 6.727911857292759
Number of index nodes accessed = 118
Number of data block accessed = 951
tConst  AverageRating   NumVotes        BlockID
tt0054167       7.7     30022           16485
tt0054168       5.3     266             16485
tt0026778       7.9     30034           5327
tt0026779       5.6     65              5327
tt0091827       3.7     589             32600
tt0091828       5.6     30037           32600
tt3361786       8.6     42              386895
tt3361792       6.8     30041           386895
tt1456941       6.2     30049           286491
tt1456944       4.2     62              286491
--------------------------------------------
```

For block size = **500 bytes**:

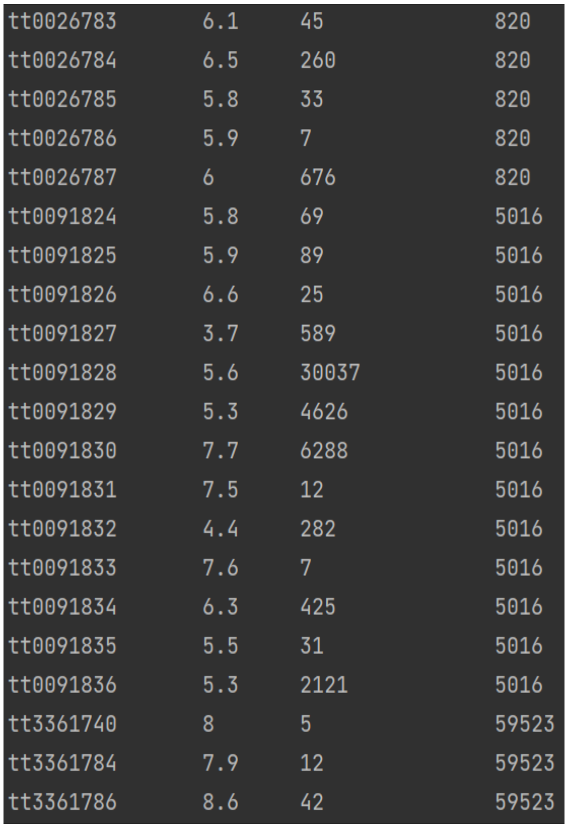
- the number and the content of index nodes the process accesses:
- the number and the content of data blocks the process accesses:
- the average of “averageRatings” of the records that are returned:



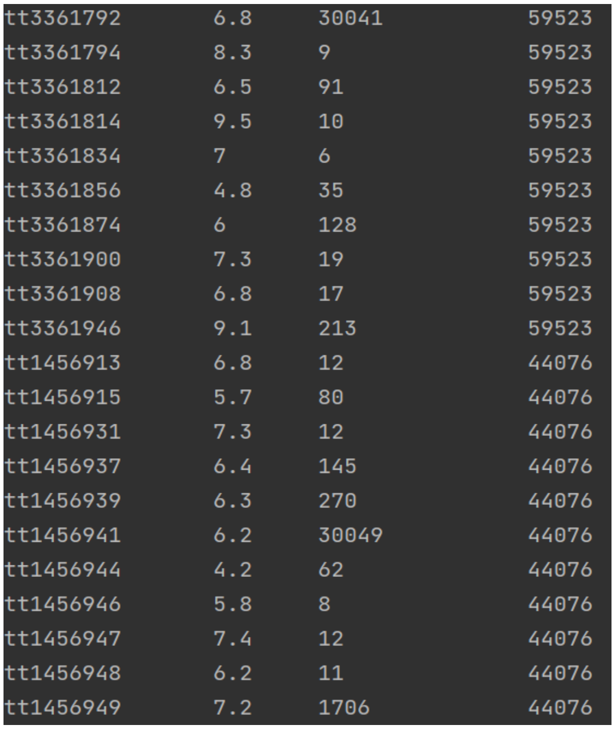
```
Experiment 4 starting...
--------------------------------------------
Nodes Accessed: |5 6 6 7 8 9 10 11 13 16 18 21 24 29 35 43 54 72 128 246 610 || 636 667 695 723 754 785 820 859 901 950 991 1037 1085 1137 1201 1268
1334 1401 1521 1594 1678 1837 2035 2267 2408 2557 2900 3293 3574 3941 4338 4866 5903 7473 8927 10843 13609 17870 28240 49780 116086 || 28741 29032 29
310 29730 30034 30458 30788 31173 31440 31821 32374 32859 33404 33786 34344 34800 35136 35447 36158 36486 36957 37489 38051 38421 38895 39659 40378 4
0753 41224 41669 42090 42761 43668 44425 45135 46078 46815 47269 47897 48446 49113 ||
Average of average rating = 6.727911857292759
Number of index nodes accessed = 27
Number of data block accessed = 932
```



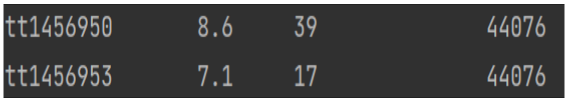
```
tConst  AverageRating   NumVotes        BlockID
tt0054167       7.7     30022           2537
tt0054168       5.3     266             2537
tt0054169       5.4     443             2537
tt0054170       5.6     19              2537
tt0054171       7.1     58              2537
tt0054172       5.6     860             2537
tt0054173       5.2     40              2537
tt0054174       6.8     254             2537
tt0054175       6.5     37              2537
tt0054176       7.5     1906            2537
tt0054177       7.2     6095            2537
tt0054178       5.6     41              2537
tt0054179       5.7     11              2537
tt0026773       4.5     146             820
tt0026774       5.3     207             820
tt0026775       6       204             820
tt0026776       6.8     73              820
tt0026777       6.9     15              820
tt0026778       7.9     30034           820
tt0026779       5.6     65              820
tt0026781       6.1     327             820
```



```
tt0026783       6.1     45              820
tt0026784       6.5     260             820
tt0026785       5.8     33              820
tt0026786       5.9     7               820
tt0026787       6       676             820
tt0091824       5.8     69              5016
tt0091825       5.9     89              5016
tt0091826       6.6     25              5016
tt0091827       3.7     589             5016
tt0091828       5.6     30037           5016
tt0091829       5.3     4626            5016
tt0091830       7.7     6288            5016
tt0091831       7.5     12              5016
tt0091832       4.4     282             5016
tt0091833       7.6     7               5016
tt0091834       6.3     425             5016
tt0091835       5.5     31              5016
tt0091836       5.3     2121            5016
tt3361740       8       5               59523
tt3361784       7.9     12              59523
tt3361786       8.6     42              59523
```



```
tt3361792       6.8     30041           59523
tt3361794       8.3     9               59523
tt3361812       6.5     91              59523
tt3361814       9.5     10              59523
tt3361834       7       6               59523
tt3361856       4.8     35              59523
tt3361874       6       128             59523
tt3361900       7.3     19              59523
tt3361908       6.8     17              59523
tt3361946       9.1     213             59523
tt1456913       6.8     12              44076
tt1456915       5.7     80              44076
tt1456931       7.3     12              44076
tt1456937       6.4     145             44076
tt1456939       6.3     270             44076
tt1456941       6.2     30049           44076
tt1456944       4.2     62              44076
tt1456946       5.8     8               44076
tt1456947       7.4     12              44076
tt1456948       6.2     11              44076
tt1456949       7.2     1706            44076
```



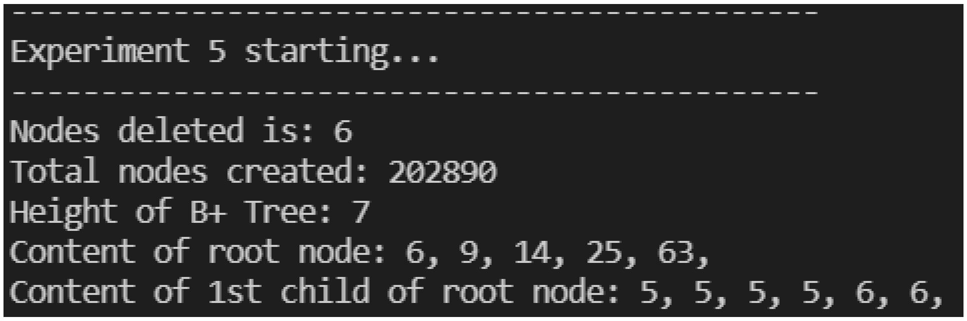
```
tt1456950       8.6     39              44076
tt1456953       7.1     17              44076
```

4.5 Experiment 5

Delete those movies with the attribute “numVotes” equal to 1,000, update the B+ tree accordingly, and report the following statistics:

For block size = **100 bytes**:

- the number of times that a node is deleted (or two nodes are merged) during the process of the updating the B+ tree:
- the number nodes of the updated B+ tree:
- the height of the updated B+ tree:
- the content of the root node and its 1st child node of the updated B+ tree:

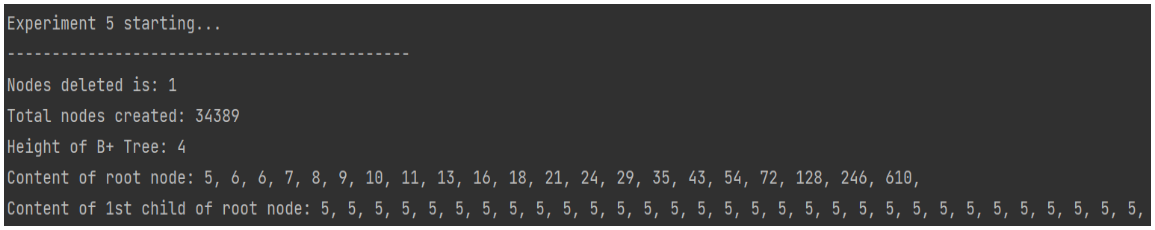


```
--------------------------------------------
Experiment 5 starting...
--------------------------------------------
Nodes deleted is: 6
Total nodes created: 202890
Height of B+ Tree: 7
Content of root node: 6, 9, 14, 25, 63,
Content of 1st child of root node: 5, 5, 5, 5, 6, 6,
```

For block size = **500 bytes**:

Delete those movies with the attribute “numVotes” equal to 1,000, update the B+ tree accordingly, and report the following statistics:

- the number of times that a node is deleted (or two nodes are merged) during the process of the updating the B+ tree:
- the number nodes of the updated B+ tree:
- the height of the updated B+ tree:
- the content of the root node and its 1st child node of the updated B+ tree:



```
Experiment 5 starting...
-------------------------------------------
Nodes deleted is: 1
Total nodes created: 34389
Height of B+ Tree: 4
Content of root node: 5, 6, 6, 7, 8, 9, 10, 11, 13, 16, 18, 21, 24, 29, 35, 43, 54, 72, 128, 246, 610,
Content of 1st child of root node: 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
```

5. Contributions of each group member

Name	Contributions
Goh Hong Xiang Bryan	Coding, report writing
Lee Cheng Han	Coding, report writing
Chong Jing Hong	Coding, report writing
Terry Joel Ee Wen Jie	Coding, report writing