Introduction to Databases, Fall 2019

Homework #3 (50 Pts, Dec 9, 2019)

Student ID 2016311821

Name 한승하

Instruction: Please write your code to complete B-tree.
Compress 'main.c', 'BTREE.c', 'BTREE.h' and 'your report' (this current document file) and submit with the filename 'HW3_STUDENT ID.zip'.

NOTE: You should write your codes in 'WRITE YOUR CODE' signs. It is not recommended to edit other parts, but you can add/modify functions if you need.

(1) [**40 pts**] Implement **insertion** and **deletion** operations of B-tree and write the codes. In addition, for the given element sequences, show the results together. **(Insertion: 15pts, Deletion: 15 pts)**

- Definition of B-tree

  1. Every node has at most m children (m: order of B tree).

  2. Every non-leaf node (except root) has at least ⌈m/2⌉ children.

  3. A non-leaf node with k children contains k−1 keys.

  4. All leaves appear in the same level.

- Please refer to the following links to implement the B-tree.

  - https://en.wikipedia.org/wiki/B-tree#Terminology

  - https://www.cs.usfca.edu/~galles/visualization/BTree.html

(a) You should fill the implementation code in the B-tree template.

Answer: Submit your code to i-campus. Don't write your code here.

(b) Show the B-tree for each case.

Answer: Show your results. (Drawing or Snapshot)

For Insertion:

1) Max degree = 3

   Insert(1, 3, 7, 10, 11, 13, 14, 15, 18, 16, 19, 24, 25, 26)

```
[10 15 ]
[3 ] [13 ] [18 24 ]
[1 ] [7 ] [11 ] [14 ] [16 ] [19 ] [25 26 ]
```

2) Max degree = 4

   Insert(1, 3, 7, 10, 11, 13, 14, 15, 18, 16, 19, 24, 25, 26)

```
[10 15 ]
[3 ] [13 ] [18 24 ]
[1 ] [7 ] [11 ] [14 ] [16 ] [19 ] [25 26 ]
```

For Deletion:

1) Max degree = 3

   Insert(1, 3, 7, 10, 11, 13, 14, 15, 18, 16, 19, 24, 25, 26) Remove (13)

```
[10 18 ]
[3 ] [15 ] [24 ]
[1 ] [7 ] [11 14 ] [16 ] [19 ] [25 26 ]
```

2) Max degree = 4

   Insert(1, 3, 7, 10, 11, 13, 14, 15, 18, 16, 19, 24, 25, 26) Remove (13)

```
[10 18 ]
[3 ] [15 ] [24 ]
[1 ] [7 ] [11 14 ] [16 ] [19 ] [25 26 ]
```

(2) [10 pts] Compare the index scan and full table scan using SQL queries on MySQL. The selectivity of a predicate indicates how many rows from a row set will satisfy the predicate.

$$\text{selectivity} = \frac{Numbers\ of\ rows\ satisfying\ a\ predicate}{Total\ number\ of\ rows} \times 100\ (\%)$$

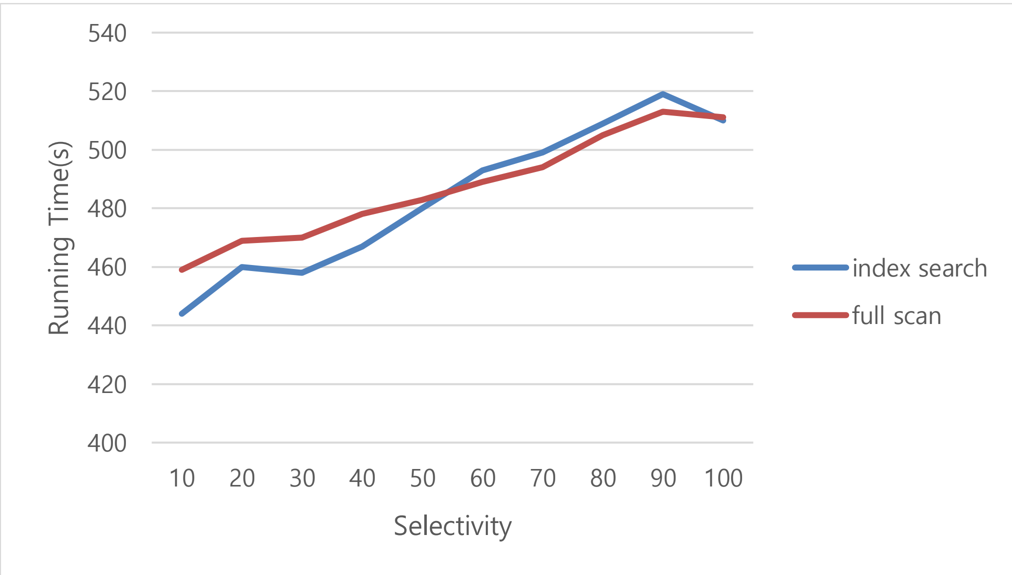
(a) Compare the running time between the index scan and the full table scan according to different data selectivity. Draw Comparison graph to compare two scan methods depending on the selectivity.

Please refer the 'example.sql' file (fix the total number of rows as 10,000,000). The figure below is only for an example. It is incorrect.

Answer: Show the correct comparison graph.

9	17:39:52	SELECT SUM(a) FROM TEST WHERE a > 9000000 LIMIT 0, 1000	1 row(s) returned	4.438 sec / 0.000 sec
10	17:39:57	SELECT SUM(a) FROM TEST WHERE a > 8000000 LIMIT 0, 1000	1 row(s) returned	4.594 sec / 0.000 sec
11	17:40:01	SELECT SUM(a) FROM TEST WHERE a > 7000000 LIMIT 0, 1000	1 row(s) returned	4.578 sec / 0.000 sec
12	17:40:06	SELECT SUM(a) FROM TEST WHERE a > 6000000 LIMIT 0, 1000	1 row(s) returned	4.672 sec / 0.000 sec
13	17:40:11	SELECT SUM(a) FROM TEST WHERE a > 5000000 LIMIT 0, 1000	1 row(s) returned	4.797 sec / 0.000 sec
14	17:40:15	SELECT SUM(a) FROM TEST WHERE a > 4000000 LIMIT 0, 1000	1 row(s) returned	4.938 sec / 0.000 sec
15	17:40:20	SELECT SUM(a) FROM TEST WHERE a > 3000000 LIMIT 0, 1000	1 row(s) returned	4.985 sec / 0.000 sec
16	17:40:25	SELECT SUM(a) FROM TEST WHERE a > 2000000 LIMIT 0, 1000	1 row(s) returned	5.094 sec / 0.000 sec
17	17:40:30	SELECT SUM(a) FROM TEST WHERE a > 1000000 LIMIT 0, 1000	1 row(s) returned	5.187 sec / 0.000 sec
18	17:40:36	SELECT SUM(a) FROM TEST LIMIT 0, 1000	1 row(s) returned	5.094 sec / 0.000 sec

31	17:46:18	SELECT SUM(b) FROM TEST WHERE b > 9000000 LIMIT 0, 1000	1 row(s) returned	4.594 sec / 0.000 sec
32	17:46:23	SELECT SUM(b) FROM TEST WHERE b > 8000000 LIMIT 0, 1000	1 row(s) returned	4.688 sec / 0.000 sec
33	17:46:28	SELECT SUM(b) FROM TEST WHERE b > 7000000 LIMIT 0, 1000	1 row(s) returned	4.704 sec / 0.000 sec
34	17:46:32	SELECT SUM(b) FROM TEST WHERE b > 6000000 LIMIT 0, 1000	1 row(s) returned	4.781 sec / 0.000 sec
35	17:46:37	SELECT SUM(b) FROM TEST WHERE b > 5000000 LIMIT 0, 1000	1 row(s) returned	4.828 sec / 0.000 sec
36	17:46:42	SELECT SUM(b) FROM TEST WHERE b > 4000000 LIMIT 0, 1000	1 row(s) returned	4.891 sec / 0.000 sec
37	17:46:47	SELECT SUM(b) FROM TEST WHERE b > 3000000 LIMIT 0, 1000	1 row(s) returned	4.938 sec / 0.000 sec
38	17:46:52	SELECT SUM(b) FROM TEST WHERE b > 2000000 LIMIT 0, 1000	1 row(s) returned	5.047 sec / 0.000 sec
39	17:46:57	SELECT SUM(b) FROM TEST WHERE b > 1000000 LIMIT 0, 1000	1 row(s) returned	5.125 sec / 0.000 sec
40	17:47:02	SELECT SUM(b) FROM TEST LIMIT 0, 1000	1 row(s) returned	5.109 sec / 0.000 sec



(b) Explain the comparison results.

Answer: Explain your comparison graph.

Index search, full scan 모두 selectivity 값에 따라 대체로 증가하는 모습을 보였다.

다만 selectivity 값이 높아 질수록 index search 가 full scan보다 더 좋은 효율을 보였다.

이는 일치하는 값이 많아질수록 index를 기준으로 검색하는 것이 더 빠른 방법이라는 것을 알 수 있다.

다만 이번 실험에서는 index값이 value당 하나씩 붙어있어 더 효율적인 검색이 불가능 했는데

Hash table 로 구성된 index search에선 index search가 훨씬 좋은 효율을 보일 것으로 예상된다