[Solution] Homework 3: B+ Trees, Hashing, Bulk Loading
(due Feb 29th, 2016, 4:00pm, in class—hard-copy please)

Reminders:
  b. Rough time-estimates: 4–6 hours.
  c. Please type your answers. Illegible handwriting may get no points, at the discretion of the grader.
     Only drawings may be hand-drawn, as long as they are neat and legible.
  d. There could be more than one correct answer. We shall accept them all.
  e. Whenever you are making an assumption, please state it clearly.
  f. Lead TA for this HW: Sorour Amiri.

Q1. B+ Tree [24 points]
Assume the following B+ tree exists with d = 2, where left leaf page contains records below the key and right leaf page contains records equal or greater than the key.

![Figure 1. Initial B+Tree.](image)

Sketch the state of the B+ tree for each operation. For each part of the problem, disregard previous parts and apply the instruction on the original tree structure above. Maintain at least 50% occupancy at each step and overflow triggered split.

Note: Use the insertion and deletion algorithms given in the textbook section 10.5 (page 349) and 10.6 (page 353) respectively. Root node can have 1 to 2d keys. During deletion redistribute the leaf pages wherever possible. In case of merge, if you can borrow from both siblings, choose the one on the right.

Q1.1. (4 points) Insert 17*.

Solution:
Q1.2. (6 points) Starting from the B+Tree of Figure 1, Insert 31*.
Solution:

Q1.3. (4 points) Starting from the B+tree of Figure 1, Delete 7*.
Solution:

Q1.4. (4 points) Starting from the B+tree of Figure 1, Delete 24*.
Solution:

Q1.5. (6 points) Starting from the B+tree of Figure 1, Delete 14*.
Solution:
Q2. Bulk Loading a B+Tree [20 points]
Suppose we are bulk-loading an initially empty B+Tree. Pages have 28 bytes to store information. A key value takes 8 bytes, and a pointer to a tree node or row takes 4 bytes. Bulk load the B+ tree with data entries 1*, 2*, ..., 12* so that each leaf is at least half full using the algorithm outlined in Section 10.8.2 (Page 360) of the textbook.

Q2.1. (4 points) What is the order of the B+ tree? Also, how many keys and pointers per node can it hold? (Note: Recall that each node in a B+ tree is a page).
Solution:
The order of the B+ tree is 1.
Each node has 2 keys and 3 pointers. $2 \times 8 + 3 \times 4 = 16 + 12 = 28$.

Q2.2. (3 points) How many levels are in the resulting tree? (e.g. The B+Tree in Figure 1 has 2 levels).
Solution:
The resulting tree has 3 levels.

Q2.3. (5 points) Sketch the final B+ tree after bulk loading. No need to show each node, just enough for us to be convinced you have the right tree.
Solution:
Based on section 10.8.2 of the text book the algorithm of bulk loading has two steps:
(1) Sorting the data entries.
(2) Building the index from the sorted entries.

The final B+ tree
Q2.4. (3 points) Is this the densest possible (i.e. the most filled) B+-Tree tree with these keys? If not, sketch the densest possible tree.

Answer:

No, the densest possible tree with these keys is:

Q2.5. (5 points) What is the minimum number of keys that must be added so that the height of the tree increases by 1? List these keys. Note1: There may be more than one correct answer. Note2: Please use the algorithm outlined in section 10.6 (page 353) of the textbook.

Solution:
The minimum number of keys is two.
One possible list of these keys is: 13*, 14*.
This is based on pushing up the middle key (Lecture 8: Indexes and Hashing (Slide No. 57)).
Another answer is 3, if you push the first key up. We’ll accept both answers.
Q3. Linear Hashing [15 points]
Consider following linear hashing index to answer the questions. Assume that we split whenever a new key triggers the creation of a new overflow page.

Notes: Here h0 is (x mod 4). Currently, h0 is active, and no bucket is split. h1 is the next hash to be used (note that h1 would be (x mod 8)). Use the linear hashing algorithm outlined in Section 11.3 (page 379) of the textbook and Lecture 8: Indexes and Hashing (Linear Hashing).

<table>
<thead>
<tr>
<th>h0</th>
<th>PRIMARY PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Next=0 32 20 Bucket A</td>
</tr>
<tr>
<td>01</td>
<td>1 25 41 37 Bucket B</td>
</tr>
<tr>
<td>10</td>
<td>14 18 Bucket C</td>
</tr>
<tr>
<td>11</td>
<td>31 35 7 11 Bucket D</td>
</tr>
</tbody>
</table>

Figure 2. Linear Hashing

Q3.1. (2 points) What is the maximum number of data entries that can be inserted in the best-case scenario, before splitting a bucket?
Solution: 4

Q3.2. (2 points) Which bucket would 64 be inserted into (use above mentioned Figure 2 as a starting point)?
Solution: Bucket A

Q3.3. (4 points) Show the file after inserting smallest possible single data entry whose insertion causes a bucket split and which bucket will be split (use above mentioned Figure 2 as a starting point).
Solution:

This shows that the bucket A and bucket E (the fifth bucket) are based on hash h1 (as they will have the same h0 values). The rest of the buckets from the split pointer (B, C, D) are still using h0 (so their h1 values are meaningless right now).
Q3.4. (7 points) Show the file after inserting following entries 43, 29 (use above mentioned Figure 2 as a starting point). Assume that insertions are performed consecutively.

Solution:

Next=1

Next=2
Q4. Extendible Hashing [16 points]

Q4.1. (10 points) Consider we are using extendable hashing on a file that contains records with the following search-key values:

\[(3, 9, 12, 15, 16, 17, 18, 20, 21, 34, 44, 49)\]

Sketch the extendable hash structure (in the same order shown) for this file if the hash function is \(h(x) = x \mod 8\) and buckets can hold up to three records. Initially, the structure is empty, and global depth is 1.

Note: Use a directory of pointers to buckets. The directory structure doubles when a bucket overflows. Use the extendible hashing algorithm outlined in section 11.2 (page 373) of the textbook and Lecture 8: Indexes and Hashing (Extendible Hashing).

Solution:

In the following hash values are presented in binary. To solve this problem we use last digit at the beginning, next last two digits, and finally last three digits.

<table>
<thead>
<tr>
<th>Key</th>
<th>(x \mod 2)</th>
<th>(x \mod 4)</th>
<th>(x \mod 8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(3)</td>
<td>1</td>
<td>11</td>
<td>011</td>
</tr>
<tr>
<td>h(9)</td>
<td>1</td>
<td>01</td>
<td>001</td>
</tr>
<tr>
<td>h(12)</td>
<td>0</td>
<td>00</td>
<td>100</td>
</tr>
<tr>
<td>h(15)</td>
<td>1</td>
<td>11</td>
<td>111</td>
</tr>
<tr>
<td>h(16)</td>
<td>0</td>
<td>00</td>
<td>000</td>
</tr>
<tr>
<td>h(17)</td>
<td>1</td>
<td>01</td>
<td>001</td>
</tr>
<tr>
<td>h(18)</td>
<td>0</td>
<td>10</td>
<td>010</td>
</tr>
<tr>
<td>h(20)</td>
<td>0</td>
<td>00</td>
<td>100</td>
</tr>
<tr>
<td>h(21)</td>
<td>1</td>
<td>01</td>
<td>101</td>
</tr>
<tr>
<td>h(34)</td>
<td>0</td>
<td>10</td>
<td>010</td>
</tr>
<tr>
<td>h(44)</td>
<td>0</td>
<td>00</td>
<td>100</td>
</tr>
<tr>
<td>h(49)</td>
<td>1</td>
<td>01</td>
<td>001</td>
</tr>
</tbody>
</table>
Q4.2 (3 points) What are the final global and local depths of all the buckets in the hash structure?

Solution:
The final global depth is 3. That means maximum 3 bits are needed to tell which bucket an entry belongs to. The final local depth of buckets 1, 2, 5, 6 is 3 and the final local depth of buckets 3, 4 is 2. Here local depth means number of bits used to determine if an entry belongs to this bucket.

Q4.3 (3 points) Suppose following range query is given “select * from table where key> x and key<=y”. Do you think this hash structure will efficiently answer the query? Why or why not?

Solution:
No. All the hash buckets need to read to answer a range query, and key values in the range are not stored in consecutive locations in the buckets. Keys are distributed uniformly and randomly throughout all the buckets.

Q5. External Sorting [25 points]
Suppose you have a file with $2 \times 10^6$ pages and assume that a two-way external merge sort algorithm is used:

Q5.1. (2 points) How many runs will you produce in the first pass?

Solution:

$$2 \times 10^6$$

Q5.2. (3 points) How many passes will it take to sort the file completely?

Solution:

$$[\log_2 N] + 1 = [\log_2 2 \times 10^6 ] + 1 = [20.93] + 1 = 22$$

Q5.3. (5 points) What is the total I/O cost of sorting the file (with 3 buffer pages)?

Solution:

Based on section 13.2 in the textbook total cost is:

$$2 \times N \times [\log_2 N] + 1$$

Therefore:

Total I/O cost : $2 \times 2 \times 10^6 \times [\log_2 2 \times 10^6 ] + 1 = 4 \times 10^6 \times 22 = 88 \times 10^6$

Answer the following questions using the general external sorting algorithm outlined in section 13.3 of the textbook (page 424). Please write the formula you used in calculating the answers.
Q5.4. (5 points) How many buffer pages do you need to sort a file with 1010000 pages completely in just three passes? (Hint: ignore the ceiling function in the formula)

Solution:
Based on section 13.3 of the textbook number of passes is \( 1 + \left\lfloor \log_{B-1} \left( \frac{N}{B} \right) \right\rfloor \), where B is number of buffers and N is the size of file in number of pages. Therefor:

\[
3 = 1 + \left\lfloor \log_{B-1} \left( \frac{N}{B} \right) \right\rfloor \\
2 = \left\lfloor \log_{B-1} \left( \frac{N}{B} \right) \right\rfloor 
\]

if we ignore the ‘ceiling’ function.

\[
2 = \log_{B-1} \frac{N}{B} \Rightarrow (B - 1)^2 = \frac{N}{B} \Rightarrow B(B - 1)^2 = N \\
N = 1010000 \text{ Therefore:} \\
B(B - 1)^2 = 1010000 = 101 \times 100^2 \Rightarrow B = 101
\]

Q5.5. (3 points) Assume that the number of available buffer pages is the one you calculated in previous question (Q5.4.). What is the maximum size of the file (in number of pages) that we can sort with 2 passes?

Solution:
Based on section 13.3 of the textbook number of passes is \( 1 + \left\lfloor \log_{B-1} \left( \frac{N}{B} \right) \right\rfloor \), where B is number of buffers and N is the size of file in number of pages. Therefore:

\[
2 = 1 + \left\lfloor \log_{100} \left( \frac{N}{101} \right) \right\rfloor \\
1 = \left\lfloor \log_{100} \left( \frac{N}{101} \right) \right\rfloor \Rightarrow 100 = \left[ \frac{N}{101} \right]
\]

It means the maximum value of N can be 101 pages.

Q5.6. (5 points) What is the total I/O cost of sorting a file with \( 2 \times 10^6 \) pages with 17 available buffer page?

Solution:
Based on section 13.3 of the textbook the total cost is the total number of reads and writes:

\[
2 \times N \times \left( \left\lfloor \log_{B-1} \left( \frac{N}{B} \right) \right\rfloor + 1 \right)
\]

Therefore, total I/O cost is:

\[
2 \times 2 \times 10^6 \times \left( \left\lfloor \log_{16} \left( \frac{2 \times 10^6}{17} \right) \right\rfloor + 1 \right) = 4 \times 10^6 \times ([4.2] + 1) = 24 \times 10^6
\]
Q5.7. (2 points) Now assume that we have a disk with an average seek time of 12ms, average rotation delay of 6ms and a transfer time of 2ms for each page. Assuming the cost of reading/writing a page is the sum of those values (i.e. 20ms) and do not distinguish between sequential and random disk-access – any access is 20ms, what is the total running time to sort the file in Q5.6?

Solution:
The total running time to sort the file is:

\[
\text{Average access time } \times \text{ total I/O cost}
\]

Average access time includes seek time + rotational delay + transfer, which is 20ms

Base on Q5.6 total I/O cost to sort the file is \(24 \times 10^6\)

Therefore, the total running time to sort the file is:

\[
20 \text{ ms } \times 24 \times 10^6 = 48 \times 10^7 \text{ ms}
\]