CS411 Database Systems

Fall 2009

HW#3

Due: 3:15pm CST, November 3, 2009

Note: Print your name and NetID in the upper right corner of every page of your submission. Hand in your stapled homework to Donna Coleman in 2106 SC. In case Donna is not in office, slide your homework under the door.

To grade homeworks faster, the homework is partitioned into two parts. **Please, submit each part separately.** For each part, make sure to write down your name and NetID.
Handwritten submissions will be graded but they will take longer to grade. For clarity, machine formatted text is preferable: Expect to lose points if your handwritten answer is unclear or misread by the grader.

This homework is partitioned into two parts as follows:

- Part 1: Problem 1 - Problem 3
- Part 2: Problem 4 - Problem 5

Part 1

Problem 1 Representing Block and Record Addresses (18 points)

Suppose that we have 4096-byte blocks in which we store records of 100 bytes. The block header consists of an offset table, as in the figure below, using 2-byte pointers to records within the block.

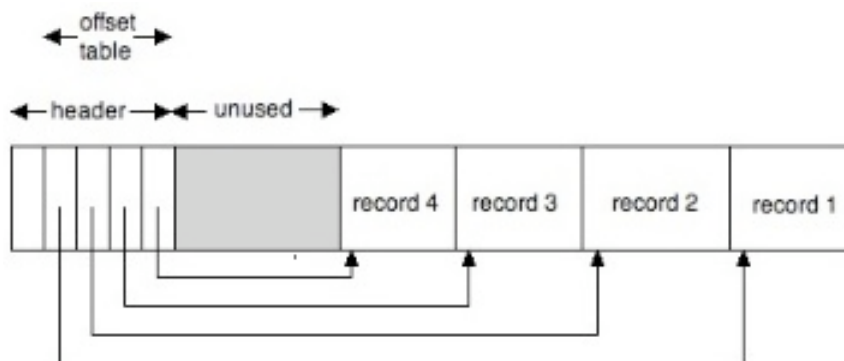

Figure 1: Block Header

a) What kind of information is stored in the block header in Figure 1 besides the offset table? (2 points)
Block header also contains information such as schema, length, timestamp.

b) Explain why we prefer to have unused area in the middle of the block? (2 points)
To accommodate growing records (records are placed starting at the end of the block). If records are placed just after the offset table, we have to move all existing records to make space for a new entry in the offset table, every time we insert a new record.

c) On an average day, two records per block are inserted, and one record is deleted. A deleted record must have its pointer replaced by a "tombstone", because there may be dangling pointers to it. For specificity, assume the deletion on any day always occurs before the insertions. If the block is initially empty, after how many days will there be no room to insert any more records? (10 points)
Each additional record requires 100 bytes + 2-byte pointer. When a record is deleted the 2-byte pointer remains.

First record inserted: 204 bytes
next day:
delete followed by 2 records inserting: 204-100+2(102) = 308
third day:
delete followed by 2 records inserting: 308-100+2(102) = 412
fourth day:
delete followed by 2 records inserting: 412-100+2(102) = 516
Each day, after the first record insertion, the effective record growth is 104 bytes. 204+n(104) $\leq$ 4096 where n is 1 less than the total number of days.
n = 37
Answer: After 38 days there will be not enough space to insert new record.

d) Redesign the layout of a block to store records more efficiently considering the fact the block stores $fixed - length$ records. (4 points)
Given that it stores fixed-length records, the offset table and record headers are not needed, consisting only of the block header and actual records.

Figure 2: Fixed length records

<u>**Problem 2**</u> Variable-length Data and Records (15 points)

Suppose blocks have 1000 bytes available for the storage of records, and we wish to store on them fixed-length records of length r, where $500 < r \leq 1000$. The value of r includes the record header, but a record fragment requires an additional 16 bytes for the fragment header. For what values of r can we improve space utilization by spanning records?

Every record and record fragment requires a fragment header to support spanned records.
Since $500 < r \leq 1000$, we can store only a single record when we don't support spanned records. Therefore, we don't use 1000-r bytes in this case. If we support them, we can store one record and one record fragment and we need to use 32 bytes for fragment headers. Therefore, if $r < 1000 - 32 = 968$, then we can improve space usage.

Problem 3 Index structure basics (20 points)

Consider an indexed sequential file consisting of 10,000 blocks. Each block contains 10 fixed sized records. Each key value found in the file is unique. For this problem, assume that:
- Pointers to blocks are 10 bytes long.
- Pointers to records are 20 bytes long.
- Index blocks are 5000 bytes (in addition to the header).
- Search keys for file records are 10 bytes long.

(a) How many blocks do we need to hold a sparse one-level, primary index? (5 points).

ceil(10000*(10+10) / 5000) = 40

Explanation: We need to store search key per block (10 bytes per block), and one block pointer (10 bytes) that points to the first block. Block pointers per block are not required since the blocks are contiguous, and so a block pointer can be computed using the offset from the first block pointer.
Number of blocks: **40**

(b) In (a), how many disk I/Os do we need to find and retrieve a record with a given key at the worst case? (5 points)

Sparse index has pointers for each block, not records. In part a, 21 blocks result. In order to search for the correct index, $\log_2 40$ I/O plus a final I/O to retrieve data;
$\log_2 40 + 1 = 6.32$

(c) Suppose you now construct a one-level, dense secondary index. Compute its minimum size in blocks. (5 points)

ceil(10000 * 10 / floor(5000 / (10+20))) = 603

Note: Taking floor as above is necessary to make sure that records do not span blocks. However, students who assumed that records span blocks, and computed number of blocks as 10000 * 10 / (5000 / (10+20)) = 600 is considered correct.
Number of blocks: **603**

(d) Suppose that we introduce an additional second level index on the sparse index in (a). How many disk I/Os do we need to find and retrieve a record with a given key at the worst case? (5 points)

We only need one second-level index. Thus, we need one I/O to read the second-level index, one I/O to read the first-level index, and one I/O to read the actual data block. In total, we need three I/O's.

Part 2

Problem 4 B-Tree (28 points, each part 7 points)

Consider a B-tree of degree $d = 2$, shown in Figure 2. Remember that each block has space for $2d$ keys and $2d+1$ pointers. The textbook uses a different parameter n in Chapter 14.2.1, which is equal to $d/2$. Please consider the execution of each operation in the following questions.

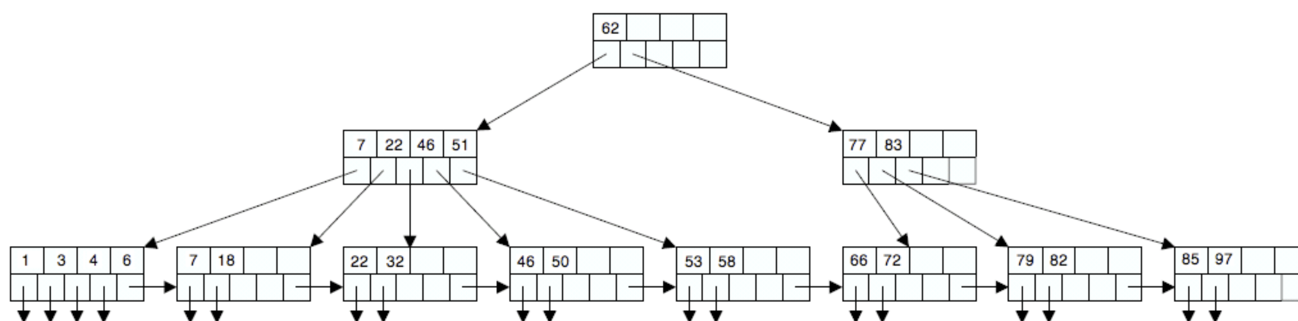

Figure 3: Block Header

(a) Look up the record with key 82. Please describe how to traverse the tree in detail.
Check 82 against the key stored in the root node - since 82 is bigger, take the right edge to the second node containing 77 and 83. Again, compare 82 against these keys linearly traversing them; since 82 is bigger than 77 but less than 83, take the edge from the second pointer to the leaf node containing 79 and 82, compare against these keys traversing from left to right. The key is found since the second key in the node matches the value 82.

(b) Look up the records with key in the range of 51 and 80. Please describe how to traverse the tree in detail.
Similarly, compare the key in the root node against 51 and traverse down to the 4th root node. From that node, linearly traverse to the next leaf node, until a key with 51 is found, or the first key with a value greater than 51 is found. In this case, it is the key with value 53. Once this key is found, traverse further until a key with value equal to 80 is found or the last key with a value less than 80 is found, which in this case is 79.

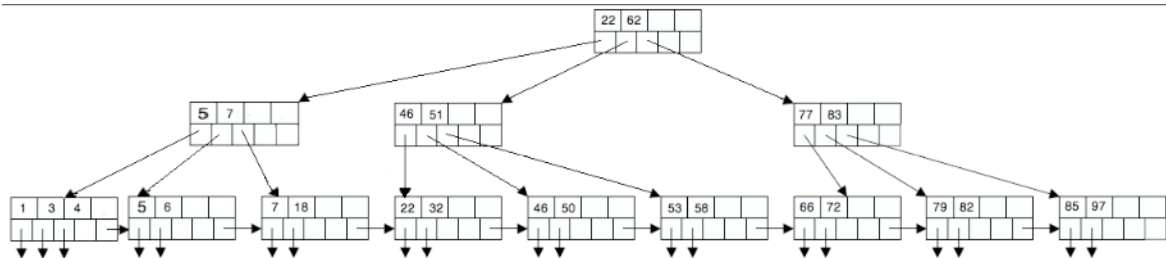

Figure 4: c. after adding 5

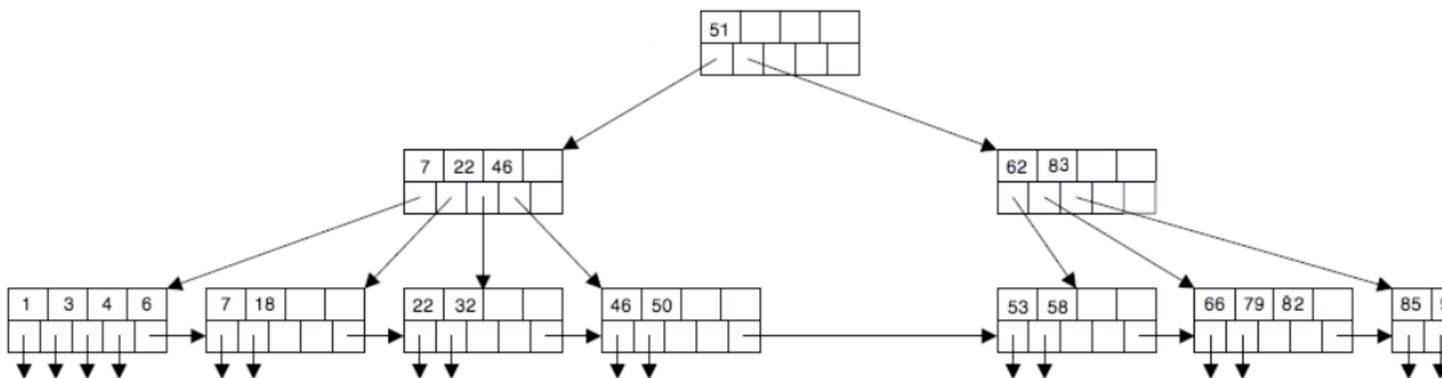

Figure 5: d. after deleting 72

<u>Problem 5</u> Hash Table (15 points)

Consider indexing the following key values using an extensible hash table. Keys are inserted in the following order:
34, 60, 51, 73, 49, 84, 25

The hash function h(n) for key n is h(n) = n mod 16, that is , the hash function is the reminder after the key value is divided by 16, giving the hash a 4-bit value. Assume that each bucket can hold 2 data items.

a) **34 % 16 = 2 → 0010**
60 % 16 = 12 →1100
51 % 16 = 3 → 0011
73 % 16 = 9 → 1001
49 % 16 = 1 → 0001
84 % 16 = 4 → 0100
25 % 16 = 9 → 1001

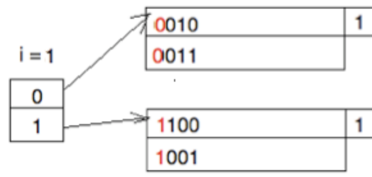

Figure 6: a). hash table

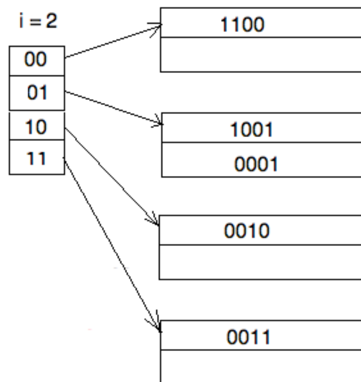

Figure 7: b). linear hash table