Assignment 2
80 Points

Professor Yanlei Diao

Question 1 [28 points] Disks and Access Time

Consider a disk with a sector size of 512 bytes, 63 sectors per track, 16,383 tracks per surface, 10 double-sided platters (i.e., 20 surfaces). The disk platters rotate at 7,200 rpm (revolutions per minute). The average seek time is 9 msec, whereas the track-to-track seek time is 1 msec (use these numbers in appropriate places in your calculation).

Suppose that a page size of 4096 bytes is chosen, and a page can span sectors on difference tracks. Suppose that a file containing 1,000,000 records of 256 bytes each is to be stored on such a disk. No record is allowed to span two pages.

(1) What is the capacity of a track (in number of bytes)?
(2) What is the capacity of the disk (in number of bytes)?
(3) How many records fit in a page?
(4) How many records fit in a cylinder?
(5) If the file is arranged sequentially on the disk, how many sectors are needed? How many pages are needed? And how many cylinders are needed?
(6) How much time is required to read this file **sequentially**? Please show your calculation clearly for (a) seek time, where a random seek takes 9 msec and then the track-to-track seek takes 1 msec; (b) rotational delay, which occurs only in the first seek operation in sequential I/O; (c) transfer time, and (d) total time.
(7) How much time is needed to read 50% of the pages in the file **randomly**, that is, one random I/O per page as observed through index lookups? Please calculate the seek time, rotational delay and transfer time for each page first, and then calculate the total cost of 50% of the pages in the file.

To answer each question, please write a clean formula and your final answer.

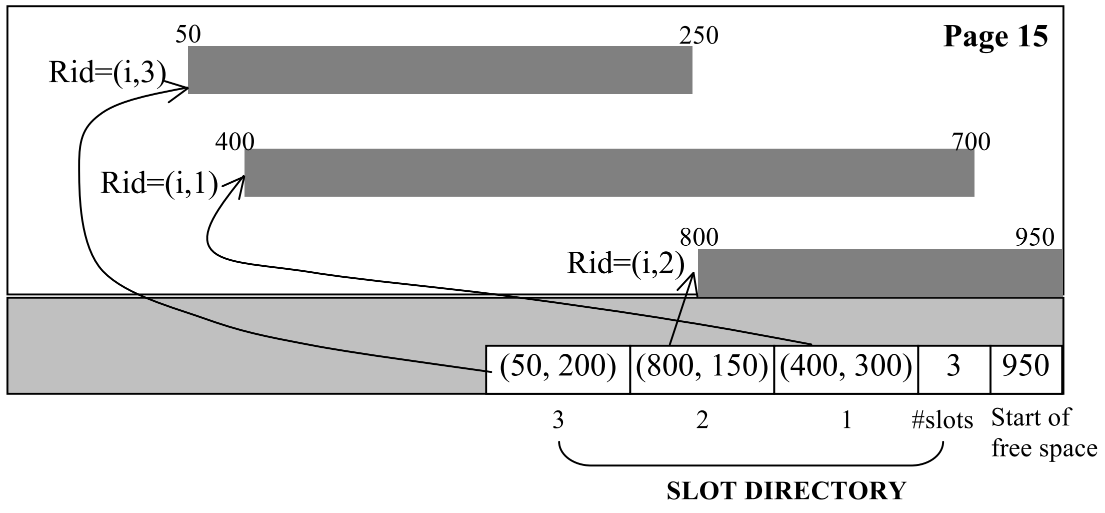
(1) What is the capacity of a track (in number of bytes)?	**Answer:** 512 * 63 = 32,256
(2) What is the capacity of the disk (in number of bytes)?	**Answer:** 512 * 63 * 16383 * 20 = 10,569,000,960
(3) How many records fit in a page?	**Answer:** 4096/256 = 16
(4) How many records fit in a cylinder?	**Answer:** (512/256) * 63 * 20 = 2,520
(5) If the file is arranged sequentially on the disk,	

how many sectors are needed? How many pages are needed? And how many cylinders are needed?	(a) Num. of sectors: ________________ (b) Num. of pages: ________________ (c) Num. of cylinders: ________________ **Answer:** 1,000,000/ 16 = 62,500 pages needed, or 62,500 * 8 = 500,000 sectors. Each cylinder has 63 * 20 = 1,260 sectors So we need ceiling(500,000/1,260) = 397 cylinders. If one considers the alignment of blocks (pages) to track boundaries, he may estimate the number of cylinders to be slightly different: ceiling(62,500 (pages) / (63/8=7) (pages/track)) / 20 = 447 cylinders
(6) How much time is needed to read this file **sequentially**?	(a) Seek time: ________________ (b) Rotational delay: ________________ (c) Transfer time: ________________ (d) Total: ________________ **Answer:** (a) *Seek time*: This access seeks the initial position of the file (whose cost can be approximated using the average seek time) and then seeks between adjacent tracks 496 times (whose cost is the track-to-track seek time). So the seek time is 0.009 + 396*0.001 = 0.405 seconds. (b) *Rotational delay*: The transfer time of one track of data is 1/ (7200/60) = 0.0083 seconds. For this question, we use 0.0083/2 = 0.00415 as an estimate of the rotational delay. Note: some student may consider rotational delay for each cylinder, which is not necessary, but we can take the answer as well. (c) *Transfer time*: It takes 0.0083*(500000/63) = 65.8730159 seconds to transfer data in 500,000 sectors. Therefore, total access time is 0.405 + 0.00415 +

	65.8730159 = 67.2821659 seconds.
(7) How much time is needed to read 50% of the pages in the file **randomly**, that is, one random I/O per page as observed through index lookups?	(a) Cost per page: ____________ (b) Total cost ____________ **Answer:** Number of pages = 5 * 6250 Time cost per page: 0.009 (seek) + 0.0083/2 (rotational delay) + 0.0083*8/63 (transfer) = 0.0142 seconds Total cost = 5 * 6250 * 0.0142 = 443.75 seconds Note: The students' answers may differ slightly due to the rounding issues, but those are still accepted as correct answers.

Question 2: Disk Page Layout [10 points]

The figure below shows a page containing variable length records. The page size is 1KB (1024 bytes). It contains 3 records, some free space, and a slot directory in that order. Each record has its record id, in the form of Rid=(page id, slot number), as well as its start and end addresses in the page, as shown in the figure.



(1) Now a new record of size 200 bytes needs to be inserted into this page. Apply the record insertion algorithm (with page compaction, if necessary) that we learned in class to this page. Show the **content of the slot directory after the new record is inserted**.

(2) The next question proceeds after the operation in Part (1). Now, the record with Rid =(15,3) needs to be deleted. Afterwards, another record of size 300 bytes needs to be inserted. Show the **content of the slot directory after the deletion and new insertion.**

Answer:
The insert algorithm is described as follows:
1) When there is enough free space:
   (1) In the free space, allocate space to accommodate the new record.
   (2) In the slot directory area, we have a list of slots, each of which is a pair (offset, length), indicating the start address and length of a data record. To create a slot for the newly insert record, (a) either reuse a slot with offset set to -1, or (b) if all are used, grow the slot area by 1.
2) If not enough space is available in the free space area, we perform page compaction:
   (1) Get all slots with offset not set to -1 from the slot directory,
   (2) Sort the slots by the offsets field,
   (3) Move the corresponding records to the beginning of the page in the sorted order of offset, so we can aggregate free space at the bottom of the page.

The deletion algorithm is simple. Given a record to delete, with its page id and slot number, we simply locate the slot for this record in the slot directory, and set the offset to -1.

(1) Content of the slot directory, from left to right, is:
[(650, 200), (0, 200), (500, 150), (200, 300)], 4, 850

(2) Content of the slot directory, from left to right, is:
[(450, 200), (650, 300), (300, 150), (0, 300)], 4, 950

Question 3 [26 points]: B+ Trees

(1) [8 points] Show the results of entering the keys 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 (in that order) to an initially empty B+ tree. Assume that every non-leaf node can hold up to 3 index entries and every leaf node can hold up to 3 data entries. In case of overflow, split the node (do not re-distribute keys to neighbors).

(2) [8 points] Now demonstrate a different insertion order that leads to a tree of different depth from the one in Part (a).

(3) [10 points] Assume that you have just built a B+ tree index using Alternative (2) on a heap file containing 1,000,000 records. The key field for this B+ tree index is a 40-byte string, and it is a candidate key of the associated relation. Pointers (i.e., record ids and page ids) are (at most) 10-byte values. The size of one disk page is 1024 bytes. The index was built in a bottom-up fashion (using the bulk-loading algorithm), and the nodes at each level were filled up as much as possible.

(a) How many levels does the resulting tree have? ________

(b) For each level of the tree, how many nodes are at that level?

Level 0: ________ , Level 1: ________ , Level 2: ________ , …

Answer:
(1) Resulting tree 1:
Root: (7)
Level 1 nodes: (3,5) and (9,11)
Level 2 nodes: (1,2) (3,4) (5,6) (7,8) (9,10) (11 12)

Resulting tree 2:
Root: (9)
Level 1 nodes: (3,5,7) and (11)
Level 2 nodes: (1,2) (3,4) (5,6) (7,8) (9,10) (11 12)

Note: When we split a leaf note, we have 4 entries so we split them evenly into two leaf nodes. In comparison, when we split a non-leaf node, we push one entry out of 4 to the parent node and split the remaining three entries into two nodes in either way.

(2) Insertion order: 1, 2, 4, 5, 3, 7, 8, 6, 10, 11, 9, 12

Resulting tree:
Root (4,7,10)
Level 2 nodes: (1,2,3) (4,5,6) (7,8,9) (10,11,12)

We also accept other answers if the student can justify well. The overall structure is simple: the leaf nodes must be full, so is the root.

(3) The answer to each question is given below.

(a) Since the search key of the index is a candidate key of the relation, there are as many data entries in the B+ tree as records in the heap file.

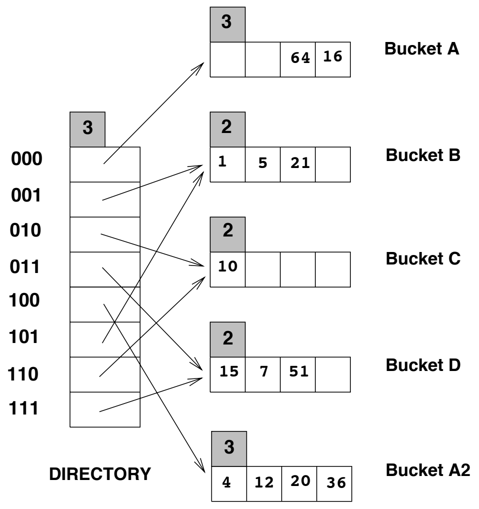
- An index page consists of at most *m* keys and *m*+1 pointers. So we have to maximize *m* under the condition that $m \cdot 40+(m+1)\cdot 10 \leq 1024$. The solution is $m = 20$, which means that we can have 20 keys and 21 pointers on an index page.
- A record on a leaf page consists of the key field and a pointer. Its size is 40+10=50 bytes. Therefore a leaf page has space for (1000/50)=20 data entries.

The resulting tree has $\lceil \log_{21}(1{,}000{,}000/20) \rceil + 1 = 5$ levels.

(b) Since the nodes at each level are filled as much as possible, there are 1,000,000/20 = 50, 000 leaf nodes on level 4. (A full index node has $2d+1 = 21$ children.) Therefore there are $\lceil 50{,}000/21 \rceil = 2381$ index pages on level 3, $\lceil 2381/21 \rceil = 114$ index pages on level 2, $\lceil 114/21 \rceil = 6$ index pages on level 1, and there is one index page on level 0 (the root of the tree).

Question 4 [16 points]: Extendible Hashing

Consider the Extendible Hashing index shown below. Answer the following questions about this index:

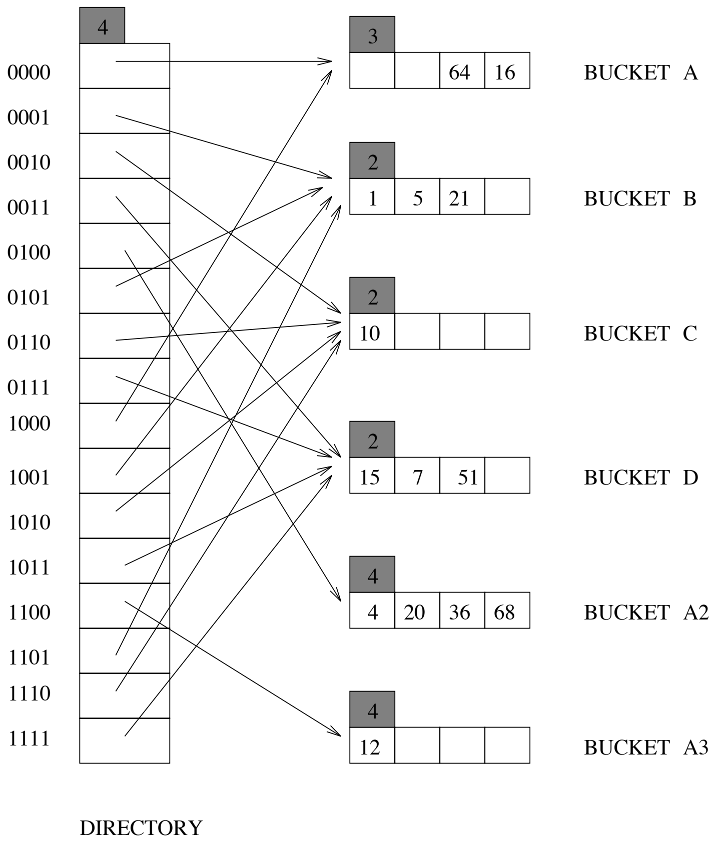


(1) Show the index after inserting an entry with hash value 68.

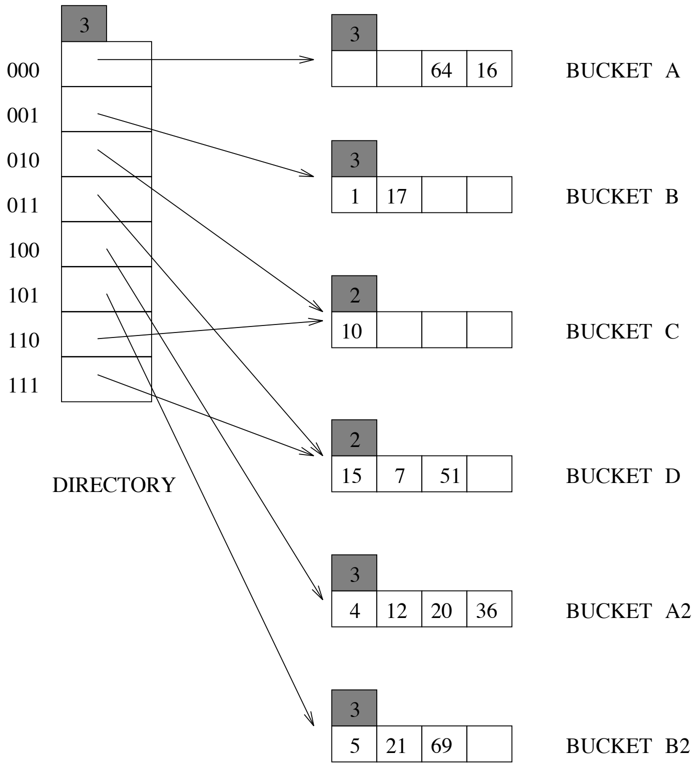
(2) Show the index after inserting entries with hash values 17 and 69 into the original index.

Answer:

(1)



(2)


3
000
001
010
011
100
101
110
111
DIRECTORY
3
64 16
BUCKET A
3
1 17
BUCKET B
2
10
BUCKET C
2
15 7 51
BUCKET D
3
4 12 20 36
BUCKET A2
3
5 21 69
BUCKET B2