Introduction to Database Systems

Problem Set 3

Due: February 25, 2016 at 11:59 PM

1. We have studied five basic file organizations for a table:

   - heap file (randomly ordered)
   - sorted file
   - clustered B+ Tree
   - unclustered B+ Tree
   - unclustered hash index

   What main conclusions can you draw from our discussion of these options? Which of the five organizations would you choose for a file where the most frequent operations are as follows?

   - Search for records based on a range of field values.
   - Perform inserts and scans, where the order of records does not matter.
   - Search for a record based on a particular field value.

2. Consider a disk with a sector size of 512 bytes, a block size of 1024 bytes, 2000 tracks per surface, 50 sectors per track, five double-sided platters, and average seek time of 10 msec. Its platters spin at 5400 rpm. Suppose that a file containing 100,000 records of 100 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

   (a) How many records fit onto a block?

   (b) How many blocks are required to store the entire file? If the file is arranged sequentially on the disk, how many surfaces are needed?

   (c) How many records of 100 bytes each can be stored using this disk?

   (d) If pages are stored sequentially on disk, with page 1 on block 1 of track 1, what page is stored on block 1 of track 1 on the next disk surface? How would your answer change if the disk were capable of reading and writing from all heads in parallel?

   (e) What time is required to read a file containing 100,000 records of 100 bytes each sequentially? Again, how would your answer change if the disk were capable of reading/writing from all heads in parallel (and the data was arranged optimally)?

   (f) What is the time required to read a file containing 100,000 records of 100 bytes each in a random order? To read a record, the block containing the record has to be fetched from disk. Assume that each block request incurs the average seek time and rotational delay.

3. Consider a buffer pool with 3 pages that receives requests for the following page numbers:

   $$2, 4, 4, 2, 5, 2, 1, 1, 3, 1$$

   Calculate the hit rate (# of hits / all page requests) for the following replacement policies:

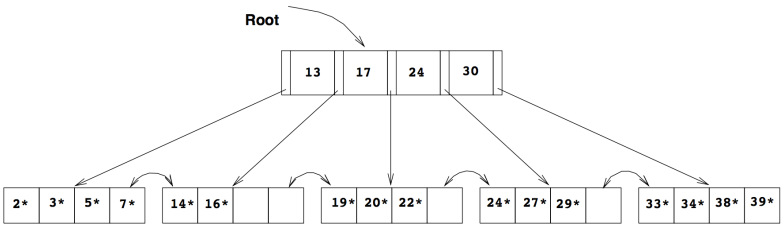
   (a) least recently used

   (b) clock replacement

   (c) most recently used

   Which strategy do you recommend for this workload?

4. Consider the following B+ tree:



   (a) Identify a list of five data entries such that:

      i. Inserting the entries in the order shown and then deleting them in the opposite order (e.g., insert a, insert b, delete b, delete a) results in the original tree.

      ii. Inserting the entries in the order shown and then deleting them in the opposite order (e.g., insert a, insert b, delete b, delete a) results in a different tree.

   (b) What is the minimum number of insertions of data entries with distinct keys that will cause the height of the (original) tree to change from its current value (of 1) to 3?

   (c) Would the minimum number of insertions that will cause the original tree to increase to height 3 change if you were allowed to insert duplicates (multiple data entries with the same key), assuming that overflow pages are not used for handling duplicates?

5. The algorithms for insertion and deletion into a B+ tree are presented as recursive algorithms. In the code for insert, for instance, a call is made at the parent of a node N to insert into (the subtree rooted at) node N, and when this call returns, the current node is the parent of N. Thus, we do not maintain any parent pointers in nodes of B+ tree. Such pointers are not part of the B+ tree structure for a good reason, as this exercise demonstrates. An alternative approach that uses parent pointers–again, remember that such pointers are not part of the standard B+ tree structure!–in each node appears to be simpler:

   "Search to the appropriate leaf using the search algorithm; then insert the entry and split if necessary, with splits propagated to parents if necessary (using the parent pointers to find the parents)."

   Consider this (unsatisfactory) alternative approach:

   (a) Suppose that an internal node N is split into nodes N and $N2$. What can you say about the parent pointers in the children of the original node N?

   (b) Suggest two ways of dealing with the inconsistent parent pointers in the children of node N.

   (c) For each of these suggestions, identify a potential (major) disadvantage.

   (d) What conclusions can you draw from this exercise?