CS660 Fall 2024 – Written Assignment 2

Title: *Disks, Indexing (Tree, Hash, LSM), Ex. Sorting*

Due: 10/14 11:59 PM on Gradescope

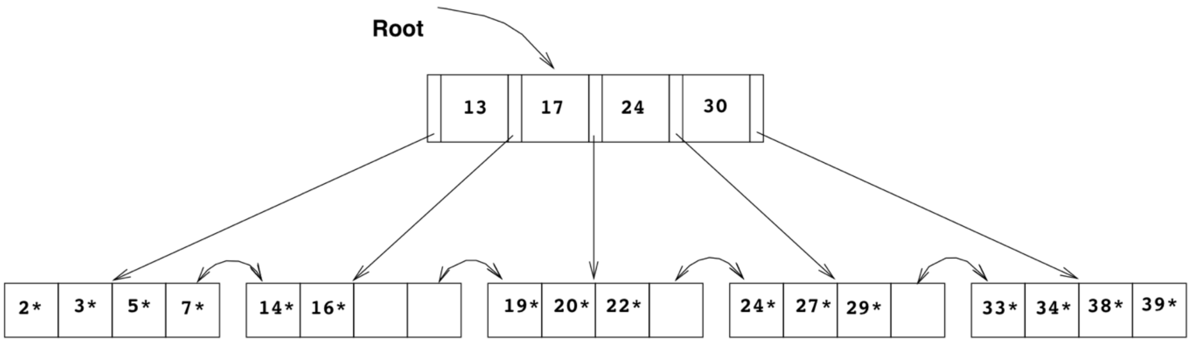
Problem 1 [25 pts]

Consider a hard disk drive that has 5 double-sided platters, each surface has 1000 tracks, each track has 256 sectors of size 512 bytes. Each block (disk page) comprises of 8 sectors. The seek time between adjacent tracks in 1ms and the average seek time between two random tracks is 25ms. The disk rotates at a speed of 9000 rpm (revolutions per minute).

Let's say, we have a file of size 1024 KB and it contains 2048 equal-sized records.

1. What is the size of a block? How many records fit in a block? How many blocks are required to store the entire file?
2. What is the capacity of each cylinder (in megabytes)?
3. What is maximum time (worst case) to read two blocks from the disk (the blocks to be read are part of the same read request and no external factors affect the read latency)?
4. If the file is stored "sequentially", how long will it take to read the whole file? Assume that for sequential writes data are written in adjacent tracks once a track is full.
5. If the blocks in the file are spread "randomly" across the disk, how long will it take to read the whole file?

Problem 2 [25 pts]



1. Based on the given B+ tree, Identify a list of five data entries such that:

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

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

2. 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?

3. 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?

Problem 3 [25 pts]

Suppose that we are using extendible hashing on a file that contains records with the following search-key values:

(449, 654, 135, 331, 615, 831, 1016, 176, 285, 468, 340, 124, 136, 668, 818, 117)

1. Load these values into a file in the given order using extendible hashing. Assume that every bucket (block) of the index can store up to four (4) data entries.

   Show the structure of the hash index after every 8 insertions, and the global and local depths.
   Use the hash function: h(K) = K mod 64 and then apply the extendible hashing technique.
   Using this function, every number is mapped first to a number between 0 and 63 and then we take its binary representation. The extendible hashing technique is then applied to the binary representation. Furthermore, initially, you start with a single bucket and a single pointer; the global and local depths are zero (0).

2. State one advantage and one disadvantage of Linear Hashing versus the Extendible Hashing.

Problem 4 [25 pts]

Suppose we want to store 2^{64} entries and that a disk page fits 2^{8} entries.
$n = 2^{64}; B = 2^{8}$

1. How many I/Os would a point query require? (Hint: assume a sorted file on disk)
2. Suppose we want to speed up our queries and we decide to build a B^{+} tree index. Compare the cost answering a point query with both approaches only considering the lookup cost, without calculating the cost to build the B^{+} tree.
3. Now assume that we use an LSM-tree to store all our entries. What is the cost of a point lookup if we employ an LSM-tree with size ratio 10 and merging policy tiering?
4. How does that cost change if we change the merging policy to leveling?