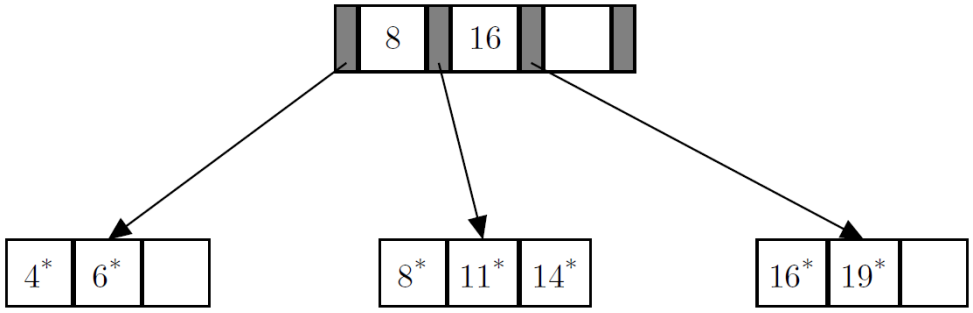
NYU Tandon School of Engineering **November 30, 2018**
Computer Science and Engineering
CS6083, Fall 2018

Problem Set 4 (due December 11)

Note: For simplicity, you may assume that KB, MB, and GB refer to 10^3, 10^6, and 10^9 bytes, respectively.

Problem 1

Assume the following simple B^+-tree with n=4:



This tree consists of only a root node and three leaf nodes. Recall that the root node must have between 2 and *n* child pointers (basically RIDs) and between 1 and *n*-1 key values that separate the subtrees. In this case, the values 8 and 16 mean that to search for a value strictly less than 8 you visit the left-most child, for at least 8 and strictly less than 16, you visit the second child, and otherwise the third child. Each internal node (none in this figure) has between 2 and 4 child pointers and between 1 and 3 key values (always one less than the number of pointers), and each leaf node has between 2 and 3 key values, each with an associated RID (to its left) pointing to a record with that key value in the underlying indexed table. Sketch the state of the tree after each step in the following sequence of insertions and deletions:

Insert 3, Insert 5, Insert 12, Delete 16, Delete 4, Insert 4, Insert 3

Note that for insertions, there are two algorithms, one that splits a full node without trying to off-load data to a direct neighbor, and one that first tries to balance with a direct neighbor in the case of a full node. Please use the first algorithm! For deletion, first try to merge, and if not possible, rebalance with a neighboring sibling.

Problem 2

You are given a sequence of 8 key values and their 8-bit hash values that need to be inserted into an extendible hash table where each hash bucket holds at most two entries. The sequence is presented in Table 1 below. (You do not need to know what function was used to compute the hashes, since the hashes are already given.) In Figure 1 you can see the state of the hash table after inserting the first two keys, where we only use the first (leftmost) bit of each hash to organize the buckets. Now insert the remaining six keys (k_2 to k_7) in the order given. Sketch the bucket address table and buckets after each insertion.

Keys	Hash values
k0	10001101

k1	01010011
k2	00000110
k3	01011011
k4	00101110
k5	01100001
k6	01011010
k7	00011001

Table 1: Sequence of 8 keys and their corresponding 8-bit hashes.

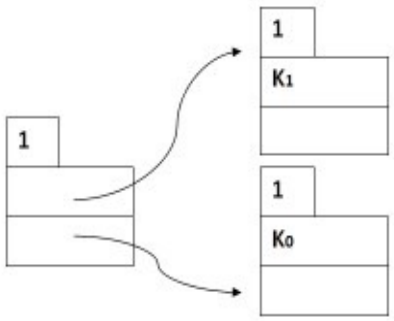


Figure 1: Hash Table state after insertion of the first two keys

Problem 3

In the following, assume the latency/transfer-rate model of disk performance, where we estimate disk access times by allowing blocks that are consecutive on disk to be fetched with a single seek time and rotational latency cost (as shown in class). Assume that a child pointer, or a record ID in an index entry, costs 8 bytes.

You are given the following very simple database schema for a check-in mechanism similar to FourSquare, where users, identified by ID, name, and hometown, can check into different places, such as bars, restaurants, museums, etc. Each place is identified by an ID, a name, coordinates, and a city. Whenever a user checks into a place, we store the user ID, the place ID, and the time and date of the check-in. The details of the schema are shown below:

User (uid, uname, ucity)
Place (pid, pname, pxcoord, pycoord, pcity)
CheckIn (uid, pid, cdate, ctime) **// uid references User and pid references Place**

Assume there are 10 million users, 2 million places, and 2 billion check-ins. Each CheckIn tuple is of size 50 bytes, all other tuples are of size 100 bytes, and each ID requires 16 bytes. Consider the following queries (where JOIN refers to a standard natural join on common attributes):

SELECT C.uid
FROM CheckIn C JOIN Place P
WHERE C.cdate = "January 12, 2014" and P.pname = "Madison Square Garden"

SELECT C.uid, C.pid, C.cdate

FROM User U JOIN CheckIn C Join Place P
WHERE ucity = "Pittsburgh" and pcity = "Austin"

(a) For each query, describe in one sentence what it does. (That is, what task does it perform?)

In the following questions, to describe how a query could be best executed, draw a query plan tree and state what algorithms should be used for the various selections and joins. Also provide estimates of the running times, assuming these are dominated by disk accesses.

(b) Assume that there are no indexes on any of the relations, and that all relations are unclustered (not sorted in any way). Describe how a database system would best execute the two queries in this case, given that 400MB of main memory are available for query processing, and assuming a hard disk with 10 ms for seek time plus rotational latency (i.e., a random access requires 10 ms to find the right position on disk) and a maximum transfer rate of 100 MB/s.

Assume that 2% of all users live in Chicago and that these users are also responsible for 2% of all check-ins. On an average day, 5000 people check into Madison Square Garden. There were a total of one million check-ins overall on January 12, 2014. One out of 200 users lives in Pittsburgh, and one out of 200 places is in Austin, and these users and places can be assumed to display fairly average behavior (i.e., they are not significantly more or less active than the average user and place).

(c) Consider a sparse clustered B+-tree index on uid in the User table, and a dense unclustered B+-tree index on pid in the Check-In table. For each index, what is the height and the size of the tree? How long does it take to fetch a single record with a particular key value using these indexes? How long would it take to fetch all, say, 50 records matching a particular pid in the case of the second index? (You may assume 80% occupancy ratio for the index nodes, and 100% for the underlying table.)

(d) Suppose that for each query, you could create up to two index structures to make the query faster. What index structures would you create, and how would this change the evaluation plans and running times? (In other words, redo (b) for each query using your best choice of up to two indexes for that query.)

Problem 4

(a) Consider a hard disk with 15000 RPM and 2 single-sided platters. Each surface has 400,000 tracks and 1000 sectors per track. (For simplicity, we assume that the number of sectors per track does not vary between the outer and inner area of the disk.) Each sector has 1024 bytes. What is the capacity of the disk? What is the maximum rate at which data can be read from disk, assuming that we can only read data from one surface at a time? What is the average rotational latency?

(b) Suppose the same disk as in (a), where the average seek time (time for moving the read-write arm) is 3ms. How long does it take to read a file of size 100 KB? How about a file of 100 MB? Use both the block model (4KB per block) and the latency/transfer-rate model, and compare. You can assume that KB, MB, GB are 10^3, 10^6, and 10^9 bytes, respectively.

(c) Suppose you have a file of size 250 GB that must be sorted, and you have only 10 GB of main memory to do the sort (plus unlimited disk space). Estimate the running time of the I/O-efficient merge sort algorithm from the class on this data, using the hard disk from part (b). Use the latency/transfer-rate model of disk performance, and ignore CPU performance. Assume that in the merge phase, all sorted runs from the initial phase are merged together in a single merge pass.

(d) Suppose you use two (instead of one) merge phases in the scenario in (c). What degree would you choose for the merges? What is the running time now?