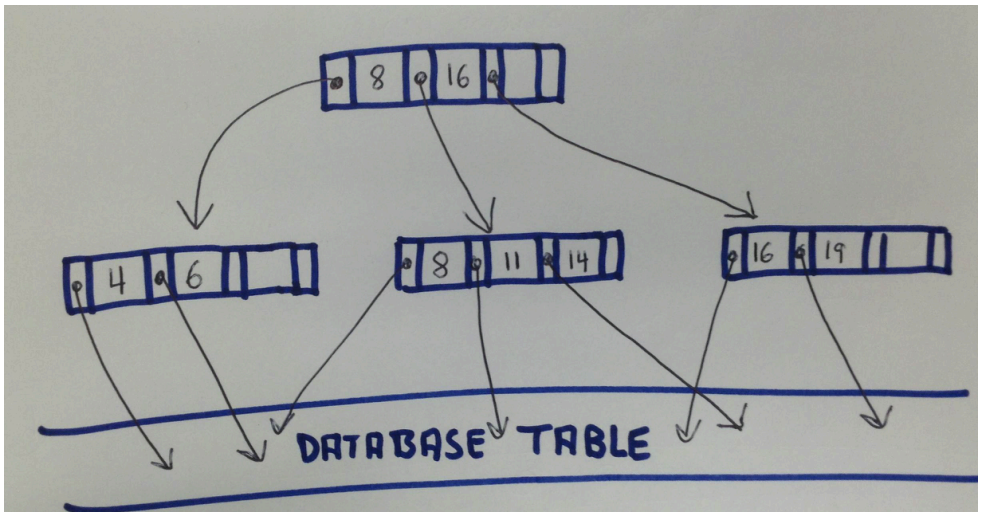
NYU Tandon School of Engineering
Computer Science and Engineering
CS6083, Fall 2015

December 3, 2015

Problem Set 4 (due 12/14/2015)

Problem 1

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

Delete 4, Delete 8, Insert 12, Insert 17, Insert 21, Insert 7, Insert 9, Delete 14

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!

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 resulting 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 the buckets after each insertion.

keys	Hash values
k_0	10001100
k_1	00001111
k_2	10000101
k_3	11001000
k_4	10101100
k_5	01010001
k_6	00101010
k_7	10010101

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

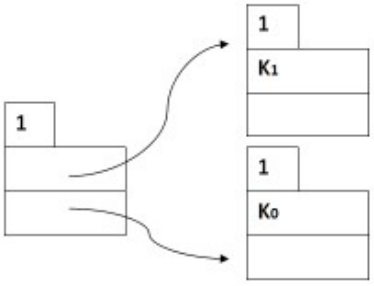


Figure 1: Hash Table state after the insertion of the first two keys

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). Also, we use the term RID (Record ID) to refer to an 8-byte "logical pointer" that can be used to locate a record (tuple) in a table.

You are given the following very simple database schema that models an online service similar to Twitter, where users can send tweets of up to 130 characters (attribute ttext) and can follow other users. We store the city where a user lives and the time and date when she joined, and we also store for each tweet the city where the user was, and the time and date. The details of the schema are shown below:

User (uid, uname, ucity, utimedate)
Follows(uid, followerid) // uid and followerid reference uid in User
Tweet(tid, uid, ttext, ttimedate, tcity) // uid references User

Assume there are 500 million users, 5 billion following relationships, and 500 billion tweets over a total period of 1000 days. Each Tweet tuple is of size 200 bytes, and all other tuples are of 100 bytes.

The uid, tid, utimedate, and ttimedate attributes require 16 bytes. Consider the following queries:

```
SELECT U.uid, U.uname
FROM User U1 JOIN Follows F, User U2
WHERE F.followerid = U2.uiddate AND U2.ucity = “Dingleboro”
```

```
SELECT U.uid
FROM User U JOIN Tweet T
WHERE date(t.ttimedate) = “January 12, 2014” and U.ucity = “New York”
```

```
SELECT U.uid
FROM User U JOIN Tweet T
WHERE U.ucity = T.tcity
```

```
SELECT U.uid, count(*)
FROM User U JOIN Tweets
WHERE U.ucity = “Miami”
GROUP BY U.uid
HAVING count(*) > 10000
```

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

In the following, 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. 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 all four queries in this case, given that 10GB 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 1% of all users live in New York and 0.1% in Miami, and 100 users live in Dingleboro. Otherwise, assume that data is evenly and independently distributed; e.g., 500 million tweets were done on January 12, 2014 as there were 500 billion tweets over 1000 days.

(c) Consider a sparse clustered B+-tree index on uid in the User table, and a dense unclustered B+-tree index on (uid, tcity) in the Tweets 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 value using these indexes? How long would it take to fetch all 50 records matching a particular key value in the case of the second index?

(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 indexes for that query.)

Problem 4:

(a) Consider a hard disk with 6000 RPM and 3 double-sided platters. Each surface has 200,000 tracks and 2000 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 512 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 we have the same disk as in (a), where the average seek time (time for moving the read-write arm) is 5ms. How long does it take to read a file of size 20 KB? How about a file of 200 KB? How about a file of 20 MB? Use both the block model (4KB per block) and the latency/transfer-rate model, and compare.

(c) Suppose you have a file of size 80 GB that must be sorted, and you have only 1 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 is better, a 8-way merge followed by a 10-way merge, or a 5-way merge followed by a 16-way merge? Analyze the running times.