Advanced Database Technology

IT University of Copenhagen

June 6, 2005

This examination assignment consists of 5 problems with a total of 15 subproblems. The weight of each problem is stated. You have 4 hours to answer all subproblems. The complete assignment consists of 6 numbered pages (including this page).

You may choose to write your answer in Danish or English. Your sheets should be numbered, and contain name, CPR number, and course code (ADBT). Write only on the front of the sheets, and sort them before numbering so that the problems appear in sorted order. When answering, remember to always **explain** (in reasonable detail) how you arrived at your answer.

GUW refers to *Database Systems – The Complete Book* by Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom, 2002

All written aids are allowed / Alle skriftlige hjælpemidler er tilladt.

1 Choosing an index (20 %)

In this problem we will investigate whether or not it can be beneficial to use two separate indexes on the same attribute for a relation, in a specific setting. The indexes are defined on a key for the relation, which has N records. We use B to denote the number of keys that fit in a disk block. We will consider two scenarios:

Scenario 1 Primary hash index on the key.

Scenario 2 Primary hash index **and** secondary B-tree index on the key.

Note that in both Scenario 1 and 2 the relation is only stored once, clustered according to the hash index. In the following we will assume for simplicity that the hash function used distributes the keys evenly, and that no overflow blocks are necessary, even after doing new insertions. Also, assume that the free space in B-tree nodes is negligible and can be ignored.

a) Suppose a key is 8 bytes and that a pointer is 4 bytes. What are the storage requirements of Scenario 1 and 2, besides the space used for storing the relation itself? (In giving the answer, you may ignore negligible terms.)

The following questions consider insertions of new records, point queries (to look up the record with a specific value for the key), and range queries (returning all records with key in a range of the form $[j, \ldots, j+r-1]$). We denote the size of the range by r, and the number of records returned by a range query by l.

b) Argue that it is possible to obtain the following I/O complexities for the operations insert, point query, and range query, in the two scenarios:

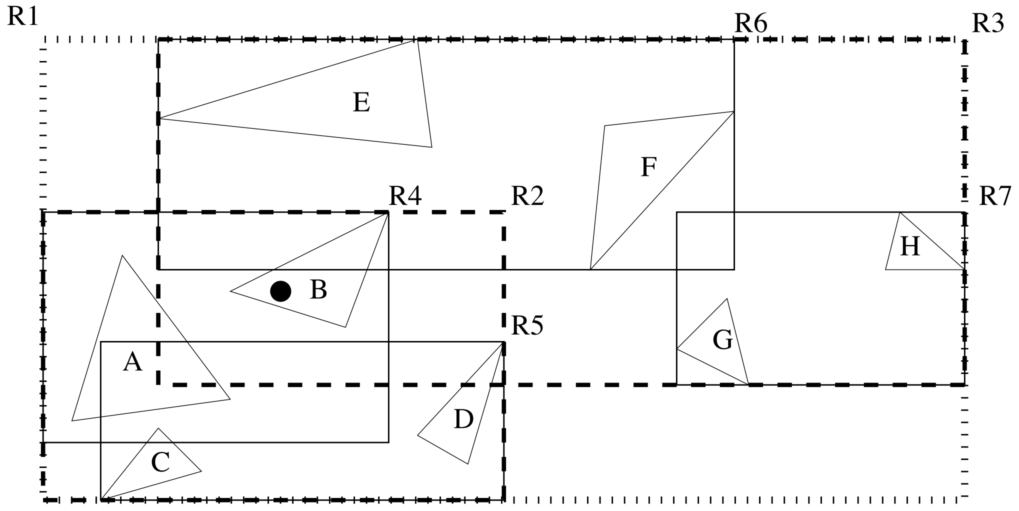
	Insert	**Point query**	**Range query**
Scenario 1	$O(1)$	$O(1)$	$O(r)$
Scenario 2	$O(\log_B N)$	$O(1)$	$O(\log_B N + l)$

In the following, ignore constant factors by assuming that all I/O bounds in **b)** hold with the constant 1 (i.e., ignore the big-O notation).

c) Consider a sequence of X insertions, Y point queries, and Z range queries, where range queries have range of average size r, and l records in the answer on average. When is scenario 1 and 2, respectively, the most efficient? Express your answer using an inequality involving (a subset of) the variables X, Y, Z, r, l, N, and B.

2 R-trees for triangles (20 %)

An R-tree is an index structure for geometric objects. In this problem we consider an R-tree where the geometric objects are non-overlapping triangles. We may draw a (binary) R-tree as follows:



Each rectangle corresponds to a node in the R-tree. In the drawing we have placed a name for each node at the upper right corner of its rectangle (except the root R1, whose name is at the top left corner). The leaf nodes containing the triangles are denoted A, B, C, D, E, F, G, H.

a) Consider a point query that asks for the triangle, if any, surrounding a point. In the example above, which nodes in the R-tree are visited when making the point query for the point shown as a black circle?

In worst case, many nodes in the tree may be visited when answering a point query. This makes it interesting to consider alternative, more specialized, data structures for the problem. We consider this question in external memory with block size B.

b) In the specific setting where we have N non-overlapping triangles as geometric objects, argue that there is an external memory data structure using $O(\log_B N)$ I/Os to perform a point query. (**Hint:** Make a reduction to a problem discussed in the course.)

c) How many nodes in an R-tree with N leaves may need to be visited while performing a single point query? Give the worst example you can come up with, involving N triangles and 1 point. The example should be valid no matter how the rectangles of the R-tree are chosen.

3 Query optimization (20 %)

Suppose we want to compute the natural join of relations $R_1(a, b)$, $R_2(b, c)$ and $R_3(c, d)$. All attributes are of type `int`. There are three possible join orders:

$(R_1 \bowtie R_2) \bowtie R_3$, $(R_1 \bowtie R_3) \bowtie R_2$, and $(R_2 \bowtie R_3) \bowtie R_1$.

We seek to find the join order that minimizes the size of the intermediate result. The database has kept "statistics", based on which we should make size estimates of each of the possible intermediate results. First of all, the relation sizes are:

$|R_1| = 10,000$, $|R_2| = 50,000$, $|R_3| = 2,000$.

Furthermore, the system maintains 3 independent tug-of-war signatures (as described in the paper *Tracking Join and Self-join Sizes in Limited Storage*) for each attribute in the relations. The signatures are shown in the following table:

	$R_1(a)$	$R_1(b)$	$R_2(b)$	$R_2(c)$	$R_3(c)$	$R_3(d)$
Signature 1	90	900	1000	500	1800	110
Signature 2	-80	-1000	-1100	700	1500	-100
Signature 3	70	700	900	-900	-2200	80

Finally, the system maintains random samples R'_1, R'_2, R'_3, consisting of 1% of the tuples in R_1, R_2, and R_3, respectively. Since the samples are small, and kept in internal memory, their join sizes can quickly be computed:

$|R'_1 \bowtie R'_2| = 100$, $|R'_2 \bowtie R'_3| = 200$, $|R'_1 \bowtie R'_3| = 2,000$.

a) Compute estimates for $|R_1 \bowtie R_2|$ and $|R_2 \bowtie R_3|$ based on each of the three tug-of-war signatures. Why are the tug-of-war signatures not useful for estimating the size of $|R_1 \bowtie R_3|$?

b) Compute estimates for $|R_1 \bowtie R_2|$, $|R_2 \bowtie R_3|$ and $|R_1 \bowtie R_3|$ based on the join sizes of the sampled relations. Supposing that an `int` is 4 bytes, what are the estimated sizes of the three possible intermediate results?

c) Suppose that all binary joins are computed using sort join, as described in GUW section 15.4.7, and that no pipelining is done.

- Argue that when joining any number of relations, it is always best to choose the join order that minimizes the total size of the intermediate results.
- Give an example where computing the join of 4 relations using a left-deep join tree does not minimize the total size of intermediate results.

4 Transaction processing (20 %)

In this problem we consider the following two transactions, consisting of 1 and 7 SQL statements, respectively.

Transaction 1:

```
1. UPDATE Seats SET reserved=1 WHERE seatID=42 AND reserved=0;
```

Transaction 2:

```
1. DELETE FROM FreeBusinessSeats;
2. DELETE FROM Upgrades;
3. INSERT INTO FreeBusinessSeats (SELECT seatID, reserved FROM Seats
                                  WHERE class='business' AND reserved=0);
4. INSERT INTO Upgrades (SELECT *
                         FROM (SELECT seatID FROM Seats
                               WHERE class='economy plus' AND reserved=1)
                         WHERE rownum<=(SELECT COUNT(*) FROM FreeBusinessSeats));
5. UPDATE FreeBusinessSeats SET reserved=1
   WHERE rownum<=(SELECT COUNT(*) FROM Upgrades);
6. UPDATE Seats SET reserved=0 WHERE seatID IN (SELECT * FROM Upgrades);
7. UPDATE Seats SET reserved=1
   WHERE (seatID,1) IN (SELECT seatID,reserved FROM FreebusinessSeats);
```

Suppose these transactions are run in a system using two-phase locking, with the exception that for transactions running at isolation level `READ COMMITTED` all read locks are released after every statement, and the necessary read locks are obtained immediately before each statement. Exclusive locks follow two-phase locking for all transactions. The system is designed to minimize the duration of locks.

a) State, for each of the relations accessed by Transaction 2, when shared and exclusive locks are obtained and released during the execution, if it is run at isolation level `SERIALIZABLE`.

b) Suppose Transaction 2 runs at isolation level `READ COMMITTED`. Argue that it is possible that `Seats` changes, because Transaction 1 runs, between statement 3 and statement 6.

Next we consider concurrency control using timestamps as described in GUW 18.8.

c) Explain what happens if the scheduler attempts to run Transaction 1 between statements 3 and 6 of Transaction 2:

- If the timestamp of Transaction 1 is smallest.
- If the timestamp of Transaction 2 is smallest.

5 Data mining (20 %)

Consider the following 8 "market baskets":

shampoo	beer	beer	shampoo	chips	milk	beer	diapers
milk	diapers	chips	diapers	coke	eggs	milk	milk
diapers	chips		eggs	beer	diapers	coke	
eggs			flour				

a) Suppose we run the Apriori algorithm on the above data to find pairs of items with support at least 3.

- Which items are found by the first pass?
- Which pairs of items are found by the second pass?

b) For each possible association rule with support at least 3, i.e., involving the pairs found in **a)**, state the confidence and interest of the association.

The Apriori algorithm runs in two passes under the assumption that there is sufficient internal memory to hold all occurring pairs of high support items in internal memory. Obviously, it would be nice to run the algorithm efficiently also in the case where the size, M, of internal memory is not sufficiently large.

c) Let N denote the total size of the data processed and stored by the Apriori algorithm, i.e., the market baskets, the item counts, and the pair counts. Show that both the first and the second pass of Apriori can be implemented to run in $O(\frac{N}{B}\log_{M/B}(N/M))$ I/Os without any assumptions about M. (**Hint:** Use an algorithm discussed in the course as a subroutine in your solution.)