Advanced Database Technology

IT University of Copenhagen

June 10, 2004

This examination assignment consists of 5 problems with a total of 12 subproblems. The weight of each problem is stated. You have 4 hours to answer all 12 subproblems. You may choose to write your answer in Danish or English. Remember to write the page number, your name, and your CPR-number on each page of your written answer. The complete assignment consists of 5 numbered pages (including this page).

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

In problems that ask for efficient algorithms, the asymptotic complexity of your algorithm will be taken into account when grading. When answering, remember to always **explain** (in reasonable detail) how you arrived at your answer.

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

1 B-trees (25%)

Consider the following setting: We have a disk with block size 2404 bytes, and want to construct a B-tree index on an integer attribute of a relation R. An integer occupies 8 bytes of space, and a pointer uses 4 bytes of space. The size of a tuple in R is 100 bytes. The leaves of the B-tree contain pointers to the tuples of the relation, i.e., the index is dense. Each node in the B-tree is contained in 1 disk block.

a) What is the largest possible degree of an internal node in the B-tree?

b) In the above setting, what is the size of the largest relation that can be indexed by a B-tree with two levels of internal nodes?

In GUW it is described how keys can be deleted from a B-tree. A deletion may require several I/Os in addition to those needed for locating the key. An alternative strategy would be to use *tombstones* to mark keys as deleted. This could always be done using one I/O.

c) Discuss possible disadvantages of the tombstone approach. Consider:

- The space occupied by deleted keys.
- The time complexity of searching for a key in the B-tree.
- The time complexity of range queries.

Your discussion should be brief and to the point.

2 Storage systems (10%)

Suppose you are employed in a large corporation that has several regional headquarters, connected by a high-speed network. The company is about to launch a new joint database whose functionality is crucial to each headquarter. You are given the task of designing a robust disk-based storage system for the database. Since the database is expected to grow big, the redundancy should be as small as possible.

a) How should the storage system be designed if we desire that all data remains accessible in any regional headquarter, even if it is disconnected from the network? You need only consider how data is stored, not how it is updated.

b) How should the storage system be designed if we only desire that all data remains accessible to the other regional headquarters if the network connection to a *single* regional headquarter is lost? You need only consider how data is stored, not how it is updated.

3 Text data structures (20%)

Short Pat Arrays are parameterized by:

- p, the block size of the external memory suffix array.
- l, the number of characters from the text included in each Short Pat Array entry.

a) Draw the Short Pat Array with parameters $p = 3$ and $l = 2$ for all suffixes of the string `MISSISSIPPI`. Pointers to the suffix array need not be included.

DNA sequences are text strings over the four letter alphabet {`A,C,G,T`}. Consider a collection of a large number of DNA sequences, stored on disk. We want to be able to efficiently search for all sequences containing a given substring. For example, when searching for the substring `AGT` we want to return sequences such as `GAGTC` and `AGTTA`, but not `AGATA`.

b) We want to be able to efficiently find (pointers to) all sequences containing a given substring. Suggest an appropriate index data structure, and describe how to use it. Your solution should in particular be efficient in the case where many sequences must be reported. You need *not* consider how the index can be efficiently constructed and updated.

4 Relational operations and query optimization (30%)

Relations are often stored on disk sorted according to some attribute(s). For simplicity we assume that a relation is stored in a sequence of consecutive disk blocks. We now consider a 2-pass sort join, as described in GUW section 15.4.7, in the special case where one of the relations is already sorted according to the join attribute(s). Let R denote the unsorted relation, and S denote the sorted relation. As in GUW chapter 15 we denote by $B(R)$ and $B(S)$ the number of disk blocks occupied by R and S, respectively, and by M the number of disk blocks that fit in internal memory.

a) Which part of the 2-pass sort join algorithm may be skipped in this special case? Argue that the resulting I/O complexity is $3\,B(R) + B(S)$, not counting I/Os to write the output.

b) Describe a modification of the 2-pass sort join algorithm that uses $3\,B(R) + B(S)$ I/Os, not counting I/Os to write the output, and works in the above special case if $M > \sqrt{B(R)} + 2$. Argue for the the I/O complexity and memory requirements of the algorithm.

Now suppose we wish to compute the relational algebra expression $(R_1 \bowtie R_2) \bowtie R_3$. There is one attribute, A, appearing in all relations R_1, R_2, and R_3, but none of them is

stored in sorted order according to A. The order of the operations has been decided, i.e., we wish to first perform the join of R_1 and R_2, and then join the result with R_3.

Several algorithms may potentially be used to compute the join. We will consider four such algorithms, three from GUW and the algorithm asked for in b), called *Exam Join* (you may assume this algorithm exists, even if you did not answer b)). These algorithms are summarized below. All of them make certain assumptions, which may be assumed to hold for the above relations. In particular, it may be assumed that there are not too many tuples with the same value on the join attribute, and that the hash function used for Hash Join distributes the tuples evenly among the buckets. However, you must consider the requirements stated in the table.

Algorithm name	GUW	I/Os	Requirements
Simple Sort Join	15.4.5	$5\,(B(R)+B(S))$	$M > \sqrt{\max(B(R), B(S))}$
Improved Sort Join	15.4.7	$3\,(B(R)+B(S))$	$M > \sqrt{B(R)+B(S)}$
Hash Join	15.5.5	$3\,(B(R)+B(S))$	$M > \sqrt{\min(B(R), B(S))}$
Exam Join	-	$3\,B(R)+B(S)$	$M > \sqrt{B(R)}+2$ S sorted by join attribute(s)

The following is known about the intermediate results of the relational algebra expression (using precise estimates made by the optimizer):

$B(R_1) = 9,000 \quad B(R_2) = 3,000 \quad B(R_3) = 6,000$
$B(R_1 \bowtie R_2) = 13,000 \quad B(R_1 \bowtie R_2 \bowtie R_3) = 2,000.$

The computation is to be performed using internal memory $M = 100$.

c) Argue that if the first join is performed using a sort-based algorithm, the second join may be performed using *Exam Join*. Then determine the best join algorithms (among the above) that can be used, i.e., the ones resulting in the lowest total number of I/Os. Pipelining should be used if possible. Argue why your answer is correct.

5 Concurrency control (15%)

a) Determine which of the following schedules are conflict-serializable:

1. $r_2(A);\ w_1(A);\ w_2(A);\ r_1(A)$
2. $w_1(A);\ w_2(B);\ r_1(A);\ r_3(B);\ w_3(A);\ r_1(B)$

Justify your answer by drawing the corresponding precedence graphs, with indication of the actions behind each arc.

Several kinds of locks are discussed in GUW, e.g., shared locks and exclusive locks. *Increment locks* allow several transactions to simultaneously update the same database

elements, as long as the only kind of update done is incrementation of integers. Suppose we want a DBMS to allow several transactions to simultaneously update the same database elements, as long as the only kind of update done is resetting database elements to some default value (e.g., 0 for integers and the empty string for strings). To this end we may introduce a new kind of lock, a *reset lock* (denoted by R).

b) Give a compatibility matrix for shared, exclusive, and reset locks. Argue for your choice at each entry involving the reset lock.