

# Advanced Database Technology

IT University of Copenhagen

June 12, 2006

This examination assignment consists of 7 problems with a total of 17 subproblems. The weight of each problem is stated. You have 4 hours to answer all subproblems. The complete assignment consists of 8 numbered pages (including this page).

You must write your answer in **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.

G UW 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 Representation of relations (20 %)

In this problem we consider a relation  $R(\mathbf{a}, \mathbf{b})$ , where  $\mathbf{a}$  and  $\mathbf{b}$  are integers (of type INT). We let  $B > 1$  denote the number of integers that fits in a disk block. Suppose that  $R$  consists of  $N$  tuples,  $\{(a_1, b_1), \dots, (a_N, b_N)\}$ , sorted such that  $a_1 < a_2 < a_3 < \dots < a_N$ . There are two natural ways of representing the relation on disk, ordered according to  $\mathbf{a}$ :

**Horizontal:**  $a_1, b_1, a_2, b_2, \dots, a_N, b_N$  (this is what we mainly considered in the course).

**Vertical:**  $a_1, a_2, \dots, a_N, b_1, b_2, \dots, b_N$ .

Some DBMSs allow the user to specify that vertical order should be used (this is an example of *vertical partitioning*).

a) How many I/Os are needed to read the  $K$  smallest values of  $\mathbf{a}$ , i.e.,  $a_1, \dots, a_K$ , in each of the two representations? State your answers as functions of  $K$  and  $B$  (exact numbers, no asymptotic notation).

Answer:

Horizontal:  $\lceil 2K/B \rceil$  I/Os. Vertical:  $\lceil K/B \rceil$  I/Os.

b) How many I/Os are needed to read the  $K$  smallest values of  $\mathbf{b}$  in each of the two representations? State your answers as functions of  $K$  and  $B$  (exact numbers, no asymptotic notation).

Answer:

Horizontal:  $\lceil 2N/B \rceil$  I/Os. Vertical:  $\lceil N/B \rceil$  I/Os.

c) Assume that there is no index on  $R$ . How many I/Os are needed to find the tuple with a particular value of  $\mathbf{a}$  in each of the two representations? State the worst case number of I/Os for the best algorithms you can think of (exact numbers, no asymptotic notation).

Answer:

In both cases, we can do a binary search for the block containing the right value of  $\mathbf{a}$ . In the vertical representation, an additional I/O is needed to read the corresponding value of  $\mathbf{b}$ . The complexities are:  $\lceil \log_2(2N/B + 1) \rceil$  I/Os (horizontal) and  $\lceil \log_2(N/B + 1) \rceil + 1$  I/Os (vertical).

We now consider a third alternative representation, the *multi-sorted* representation. Assume that the number of tuples in  $R$  is a perfect square, i.e., that  $\sqrt{N}$  is an integer. The idea is to change the horizontal representation by splitting it into  $\sqrt{N}$  intervals of  $\sqrt{N}$  tuples, and sorting each interval according to the value of  $\mathbf{b}$ . An example instance with  $N = 9$  is the following (we mark tuples by parentheses and intervals by square brackets for readability):

[(3, 2), (2, 3), (5, 5)], [(5, 4), (13, 9), (11, 10)], [(23, 1), (19, 6), (17, 14)]

**d)** Show that in the multi-sorted representation, it is possible to search for a particular value of **a**, as well as a particular value of **b**, in  $O(\sqrt{N} \log N)$  I/Os (without any index). You should describe search algorithms achieving this I/O bound (or better). Can you improve the representation, in terms of search time for particular values?

Answer:

To search for a particular value of **a**, first locate the correct interval (using linear or binary search). Then read the entire interval. This costs  $O(\sqrt{N})$  I/Os.

To search for a particular value of **b**, perform binary search in each interval, in  $O(\sqrt{N} \log N)$  I/Os.

An improved representation can be achieved, e.g., by:

- Changing the length of intervals to  $\sqrt{NB}$  (gives factor  $\sqrt{B}$ ).
- Indexing each interval (gives factor  $\log N$  or  $\log B$ , depending on the index structure).

## 2 Query optimization (15 %)

Consider relations  $R(a, b)$  and  $S(a, c)$ , where all attributes are of type integer. Assume that both relations have  $N$  tuples and are much larger than the capacity of main memory. The SQL query

```
SELECT R.a FROM R, S WHERE S.c > 10 AND R.a = S.a;
```

translates into the relational algebra expression

$$\pi_{R.a}(\sigma_{c>10}(R \bowtie S)) .$$

**a)** Suggest an alternative, equivalent relational algebra expression that is likely to result in a faster execution plan, and should never give a worse execution plan. Argue why this is the case.

Answer:

$\pi_{R.a}(R \bowtie \sigma_{c>10}(S))$ . This eliminates all tuples with  $c > 10$  before the join, saving the cost of reading/writing each data item three (or more) times in connection with the join. If pipelining is used (which it should be), the cost cannot be bigger than using the first expression.

Next, consider the following query:

```

SELECT R.a
FROM R
WHERE NOT EXISTS (SELECT S.a FROM S WHERE S.c = 10);

```

Since the subquery is correlated, a DBMS may execute it for every tuple of  $R$ . An alternative to the above query is the following, equivalent SQL statement:

```

(SELECT R.a FROM R) EXCEPT (SELECT S.a FROM S WHERE S.c = 10);

```

b) Compare the efficiency of the two SQL statements, assuming that the DBMS executes the former in the naïve way (computing the subquery for every tuple of  $R$ ), and uses no indexes. State the asymptotic (big- $O$ ) complexities in terms of  $N$  and  $B$ , assuming that all sorting steps and joins can be done using 2 passes.

Answer:

The first expression will require  $N$  subqueries, each taking  $O(N/B)$  I/Os, i.e.,  $O(N^2/B)$  I/Os in total.

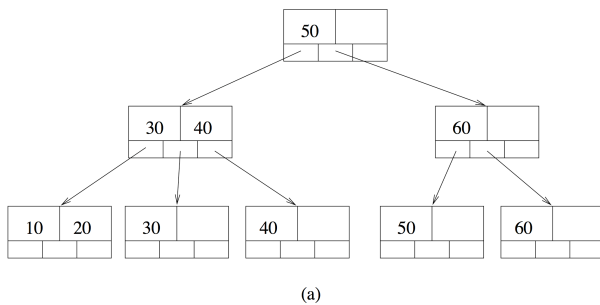
The second expression requires two selections and one set difference (implemented by sorting), each taking  $O(N/B)$  I/Os.

### 3 Height of B-trees (15%)

The purpose of this problem is to show that the height of a B-tree is sensitive to the order of insertion of keys into it. In the following, use the insertion algorithm described in G UW (Section 13.3.5) for insertions into B-tree nodes.

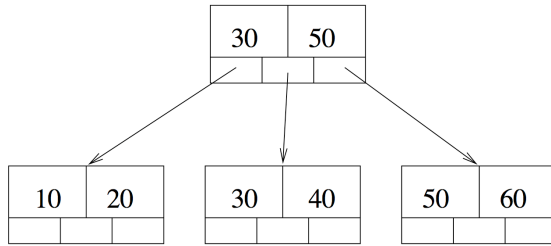
a) Insert the keys 60, 50, 40, 30, 20, 10 (in that order) into an empty B-tree with  $n = 2$  (i.e., at most two keys and three pointers can fit in a node). Show the result after inserting all the 6 keys.

Answer:



b) Construct a B-tree with  $n = 2$  for the keys 60, 50, 40, 30, 20, 10 having the smallest possible height.

Answer:



(b)

c) What are the minimum and maximum heights of a B-tree with  $n = 2$  and  $N$  keys? Argue that there exist sequences of insertions and deletions that result in the minimum and maximum height, respectively, for arbitrarily large values of  $N$ .

Answer:

$$\text{minimum height} = \lceil \log_3 N \rceil$$

$$\text{maximum height} = \lceil \log_2 N \rceil$$

Given a B-tree of any given height, we can maximize (or minimize) the number of keys by inserting (or deleting) keys in places where this will not give rise to changes in the height of the B-tree. This means that if  $N$  is a power of 2 (resp. 3) we can get to a situation in which the height is maximized (resp. minimized).

## 4 Choosing indexes (15%)

Suppose we are given a large employee database of a firm that stores the name of each employee together with the starting and ending dates of his employment. For the queries of subproblems a), b) and c) below, describe how you would store the database and any additional index structures to support the queries efficiently. You do not need to worry about efficient updates. Also, state the I/O complexities of the queries, assuming the database contains  $N$  records (each of constant size), and each block can hold at most  $B$  records.

a) Consider the above questions for the queries of the form: “Given a particular date, list all the employees of the firm on that date.”

Answer:

Consider the employment period of the employees as intervals and store them using a persistent B-tree to support the stabbing queries. Query time:  $O(\log_B N + \text{output}/B)$ .

b) Consider the above questions for the queries of the form: “Given an employee, list all the employees who joined the firm during his employment period.”

Answer:

Store the employee records in the increasing order of their starting dates in a B-tree. A query translates to a range query in this B-tree. Query time:  $O(\log_B N + output/B)$ .

Even if you did not answer the two previous subproblems, you may assume that answers are known when answering the final subproblem:

c) Consider the above questions for the queries of the form: “Given an employee, list all the employees who were working in the firm during some part of his employment period.”

Answer:

Combine (a) and (b). Query time:  $O(\log_B N + output/B)$

## 5 Concurrency control (15 %)

In the course we have considered concurrency control mechanisms based on two-phase locking (2PL). Transactions at lower isolation levels (e.g., `read committed`) may not use 2PL, but instead request and release locks before and after each statement.

a) Suppose that we have two concurrent transactions  $T_1$  and  $T_2$ , where  $T_1$  uses 2PL, and  $T_2$  is *read-only* and locks database elements only for the duration of single statements. Argue that the result of running  $T_1$  and  $T_2$  (considering database updates as well as results returned by transactions) may sometimes *not* be equivalent to a serial schedule.

Answer:

$T_2$  may read some database element, and release the lock on it, such that  $T_1$  can update it before  $T_2$  finishes. Then  $T_2$  can read the updated database element, and thus use two different snapshots of the database to form its answer.

Suppose that a DBMS implements *full locking*, meaning that all locks are obtained at the beginning of the transaction, and released at the end of the transaction.

b) Again, suppose that we have two concurrent transactions  $T_1$  and  $T_2$ . Now  $T_1$  uses full locking, and  $T_2$  is again *read-only* and locks database elements only for the duration of single statements. Argue that the result of running  $T_1$  and  $T_2$  is *always* equivalent to a serial schedule.

Answer:

[The question above was lacking information. It should have stated that  $T_1$  consists of a *single* statement.] With respect to  $T_2$ , the behavior of the DBMS will be as if  $T_1$  executed instantaneously at the moment it obtained its last lock. Thus  $T_2$  will appear to run either before or after  $T_1$ .

## 6 Undo/Redo Logging (10 %)

Consider the following transaction log from the start of the run of a database system that is using undo/redo logging with (nonquiescent) checkpointing for crash recovery:

- 1) <START T1>
- 2) <T1, X, 20, 10>
- 3) <T1, Y, 40, 0>
- 4) <START T2>
- 5) <T1, X, 50, 20>
- 6) <T2, Z, 30, 20>
- 7) <COMMIT T1>
- 8) <START T3>
- 9) <T3, U, 60, 30>
- 10) <T2, V, 50, 25>
- 11) <START CKPT (T2,T3)>
- 12) <T2, Z, 45, 30>
- 13) <COMMIT T2>
- 14) <START T4>
- 15) <T4, W, 80, 10>
- 16) <COMMIT T3>
- 17) <END CKPT>
- 18) <T4, W, 100, 80>
- 19) <COMMIT T4>

The log entries for database updates are in the format

⟨Transaction id, Variable, New value, Old value⟩

- a) What is the value of the data items X, Y, Z, U, V, and W on disk after recovery:
1. if the system crashes just before line 13 is written to disk?
  2. if the system crashes just before line 14 is written to disk?
  3. if the system crashes just before line 19 is written to disk?

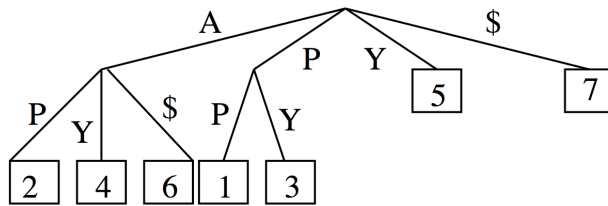
Answer:

1. X=50, Y=40, Z=20, U=30, V=25, W=10
2. X=50, Y=40, Z=45, U=30, V=50, W=10
3. X=50, Y=40, Z=45, U=60, V=50, W=10

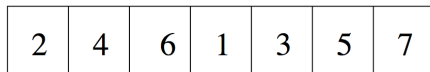
## 7 Text indexing (10 %)

a) Construct the suffix tree and suffix array for the string: PAPA YA

Answer:



Suffix Tree



Suffix Array

b) State one (or more) important advantage of suffix trees relative to suffix arrays, and vice versa.

Answer:

Suffix trees have faster searches (no logarithmic term). Suffix arrays use less space.