Exercises and hand-ins

Advanced database technology

February 6, 2006

Consider the following classic internal sorting algorithm Selection Sort. It works as follows: For $i = 1, \ldots, N$ scan the whole input and output the ith smallest element (which, for $i > 1$, is the smallest element larger than the element just output). Some features of this algorithm:

- It does not change the input.
- Uses only memory to store a couple of elements.
- Outputs the sorted list sequentially.

This problem considers the performance of Selection Sort on external memory.

1. Suppose that we have a machine whose internal memory holds only the most recently read block of B elements (plus a few registers). Analyze the I/O complexity of Selection Sort.
2. How does the I/O complexity of Selection Sort change if we have internal memory for M elements, and the virtual memory system always keeps the M/B most recently accessed blocks in internal memory? Argue why your answer is correct.
3. Devise an external version of Selection Sort having the abovementioned features (i.e., it should not write anything other than the output to disk, but may make use of all internal memory). It should improve the I/O complexity of the internal version investigated in the first two questions by a factor of M.

There will be 5 mandatory hand-ins during the course to be solved individually or in groups of two students. Assignments are given in connection with the lectures (and put on the home page) and are normally due two weeks later. Every assigment is graded on a scale from A-E (A=very good, B=good, C=acceptable, D=not good enough, E=not handed in/late hand-in), and students must achieve A, B, or C on at least 4 out of 5 assignments. If you are not able to hand in in time, for some reason, inform the teacher as soon as possible. It is allowed to discuss assignments with other students, but the solutions must be prepared in groups of size at most 2. Hand-ins should be in English, and should normally be handed in on paper in connection with the lecture on that day (e-mail hand-in as a PDF file **before the lecture** *is possible in case of absence).*

Note: First hand-in (due February 20) is on the back of the sheet.

The following problem is to be handed in at the latest February 20.

We have argued that bags can be more efficient that sets (e.g., that adding tuples to a bag is more efficient than checking whether the tuple already exists). This problem considers cases where bags may be *less* efficient, and suggests a possible compromise between bags and sets.

We consider a relation where new tuples are steadily added (but no deletions are made). New tuples may be duplicates of existing ones, and these will not affect the queries we are interested in. That is, it is okay to ignore such an insertion. Assuming tuples have fixed size, let $B \geq 1$ denote the number of tuples per disk block. Let M be the number of tuples that fits in internal memory. There are two basic policies when handling new tuples:

1. Check whether the tuple exists, and insert it only if it does not. You may assume the existence of an index that requires 2 I/Os per lookup, and 2 I/Os for each insertion of a tuple.

2. Go ahead and insert any tuple (thus creating a bag). Using a buffer block, this requires $1/B$ I/Os per insertion.

Suppose that queries require every tuple of the set or bag to be scanned. That is, if there are N tuples (counting any duplicates several times), a query requires N/B I/Os. Answer the following questions:

1. Give an example of a situation (combination of certain insertions and queries) in which policy 1 is much better than policy 2.

2. Give an example of a situation (combination of certain insertions and queries) in which policy 2 is much better than policy 1.

3. Argue that a relation with duplicates may be turned into a set in "sorting complexity", and more specifically that it can be done in two passes if $M^2/B > N$, where , and N is the number of tuples in the relation.

4. Suppose that we use policy 2, but run the duplicate elimination procedure from question 3 (assuming $M^2/B > N$) whenever one of the following holds:

   (a) The number of tuples since the last duplicate elimination has doubled, or

   (b) Eight queries has been run since the last duplicate elimination.

   After the duplicate elimination, we then continue doing insertions and queries on the resulting relation. Argue that this would require a number of I/Os that, no matter what the sequence of insertions and queries, is at most a constant factor larger than following the best of policy 1 and policy 2 for that particular sequence.