DM808 I/O-Efficient Algorithms and Data Structures

Spring 2008

Project 3

Department of Mathematics and Computer Science
University of Southern Denmark

May 6, 2008

The goal of this project is to use some of the ideas, principles, and techniques studied in the course for designing and analyzing new (variants of existing) algorithms. In other words, the project is theoretical, not empirical.

The project is to be done in groups, preferably of size two. The report should describe your algorithms and prove their correctness and I/O-complexity.

Sorting

Design an I/O-efficient algorithm for removing duplicates from a multiset of N elements. The output should be a sorted set containing one copy of each distinct element, annotated with its multiplicity (i.e., its number of occurences in the original multiset).

The algoritm should run in

$$O(\frac{N}{B} + \max\{0, \frac{N}{B}\log_{M/B}\frac{N}{M} - \sum_{i=1}^{K}\frac{N_i}{B}\log_{M/B} N_i\})$$

I/Os, where K is the number of distinct elements, and N_i is the multiplicity of the ith distinct element.

(Hint: Use multiway mergesort, but keep only one copy when several copies of the same element meet during merging. In particular, any sorted stream created during the merge process can contain at most one copy of any element. Analyze the algorithm by considering how many of the N_i copies of an element can be present at a given height in the merge tree.)

Searching

Design an external data structure for the problem of maintaining a set of N intervals such that given a query point q, the *number* of intervals containing q can be reported. The I/O-complexity should be $O(\log_B^2 N)$ worstcase for queries and $O(\log_B^2 N)$ amortized for insertions and deletions of intervals. The space should be linear. Note that the output size is constant, and does not appear in the complexity bound, in contrast to the case of reporting all the actual intervals.

(Hint: Base your solution on ideas of the external interval tree, but use much simpler secondary structures.)

Graph Algorithms

Give an $O(\text{Sort}(N))$ external algorithm for computing a BFS-numbering of a tree (given as a list of edges oriented downwards). A BFS-numbering is a labeling of the nodes of the tree with numbers $1, 2, 3, \ldots$ such that

1. the root has label 1,
2. nodes in BFS-level i have smaller labels than nodes in BFS-level $i + 1$,
3. for any two nodes in BFS-level $i + 1$, their labels are in the same order as the labels of their parents.

The nodes in BFS-level i are the nodes whose path to the root contains i edges.

(Hint: Use a method similar to DFS on trees to annotate nodes with their BFS level. Then use a constant number of sorting step per level to propagate the BFS-labels downwards.)

Deadline

Hand in your report

Friday, May 23, 2008.

at the latest.