DM207 I/O-Efficient Algorithms and Data Structures

Fall 2011

Project 3

Department of Mathematics and Computer Science
University of Southern Denmark

December 13, 2011

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 (group size one is allowed).

Searching

For the external interval tree (from Section 6 in the lecture notes *External Memory Geometric Data Structures* by L. Arge), show how to perform the following query:

> IntervalStabbingQuery(s,t): return all stored intervals $[x, y]$ which completely contain the query interval $[s, t]$ (i.e., all intervals $[x, y]$ for which $x \leq s$ and $t \leq y$).

The I/O-complexity should be $O(\log_B N + T/B)$, where T is the number of intervals returned.

[Hint: This is a slight variation on the standard stabbing query algorithm.]

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 labelling 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: You may be inspired by the algorithm for DFS on trees. Also, the fact that $\sum_i \text{Sort}(N_i)) \leq \text{Sort}(\sum_i N_i)$ may be useful (but should be proved if used).]

Sorting

Design an I/O-efficient algorithm for removing duplicates from a multi-set 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 occurrences in the original multi-set).

The algorithm should run in

$$O(\sum_{i=1}^{K} \frac{N_i}{B} \log_{M/B}(\frac{N}{MN_i}))$$

I/Os, where K is the number of distinct elements, and N_i is the multiplicity of the ith distinct element (i.e., the number of times it occurs in the input). Here, $\log_y(x)$ is shorthand for $\max\{\log_y(x), 1\}$.

[Hint: Use multi-way 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 in the worst case be present at a given height in the merge tree, and by using that one copy on one level represents $O(1/B)$ I/Os from the scanning of it during a merge process.]

Formalities

The report should describe your algorithms and argue for their correctness and their I/O-complexity. Your report should contain 5–12 pages.

The project is evaluated by pass/fail grading. The grading will be based on:

- The clarity, precision, conciseness, and comprehensiveness of your description of your algorithms.
- The correctness of your algorithms.
- The clarity, precision, conciseness, and comprehensiveness of your arguments for correctness and I/O-efficiency of your algorithms.
- The correctness of your arguments.

You should hand in your report in pdf using the assignment hand-in at the Blackboard page of the course (under menu item "Tools").

For groups of size more than one: For formal reasons, you will need to designate who wrote which part of the report.

Hand in your report

Friday, January 6, 2012, at 13:00,

at the latest.