Institut for Matematik og Datalogi
Syddansk Universitet

March 7, 2008
JFB

DM508 – Algorithms and Complexity – F08
Lecture 11

Announcement

The exam questions for this year will be the same as last year. See the course's homepage.

Lecture, March 5

We analyzed the running time of the KMP algorithm. We introduced approximation algorithms from chapter 5, and covered the randomized algorithms for MAX-SAT from the textbook and the notes (by Motwani and Raghavan).

Lecture, March 10

The students will have an opportunity to ask questions. I may present some research on on-line algorithms.

Problems to be discussed on March 14

1. A dynamic hash table could be implemented as follows: The hash table initially has size 1. When an insert occurs, if the table is not full, the item is hashed into the table. If the table is full, a new table which is twice as large is created, and all of the current elements are re-hashed into the new table.

   Assume that hashing one item takes constant time. Let T_i be the data structure after operation number i. Let n_i be the number of items in the table after operation number i. (Some operations might be searches, rather than inserts, so it is possible that $n_i \neq i$.) Let s_i be the size of the table after operation number i (always a power of 2). Define a potential function $\Phi(T_i) = 2n_i - s_i$. Note that for all tables of size at least two, the table is always at least one item more than half full. What is the amortized cost of an insertion? Consider both the case where the table is not full just before the insertion, and the case where it is full. (Do the amortized analysis.)

   Suppose n items are inserted into the structure. Give a good worst case bound on the total cost of doing all of these n insertions.

2. A main ingredient in the analysis of Fibonacci heaps is that the degree of a node must small relative to the size of the subtree of which it is a root. Unlike many other efficient data structures, there is no logarithmic bound on the depth of a tree produced by operations on n items. Show this by describing a sequence of Fibonacci heap operations on n items that produces a heap-ordered tree of depth $\Omega(n)$ in a Fibonacci heap.

3. In the *half 4-CNF satisfiability* problem, a 4-CNF formula (CNF form, with exactly 4 literals per clause) F is given. One knows that at least half of the clauses are satisfiable by any truth assignment. The problem is to determine if there exists a truth assignment to the variables of F which satisfies the entire formula. Prove that the half 4-CNF satisfiability problem is NP-complete.

4. In the *Partition* problem, a finite set A is given, along with a positive integer size $s(a)$, for each $a \in A$. The problem is to determine if there exists a subset $A' \subset A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$ (i.e. can you partition the set into two subsets so the sizes of the items add together to exactly the same amount?). In the *Bin Packing* problem, a positive integer bin capacity B, a positive integer K, and a finite set A is given, along with a positive integer size $s(a)$, for each $a \in A$. The problem is to determine if there exists a partition of A into disjoint sets $A_1, A_2, ..., A_K$ suchthat the sum of the sizes of the items in each A_i is B or less. (This partition into disjoint sets gives a packing into K bins.)

   - Partition is known to be NP-complete. Using this fact, prove that Bin Packing is also NP-complete.
   - Show that if there is an algorithm for Bin Packing which runs in time $f(n)$ for some function f, then there is an algorithm for the cost version of Bin Packing (find the cost of the packing which uses fewest bins, where the cost is the number of bins used) which runs in time $O(p(f(n))$ for some polynomial p.
   - Suppose that you know that in the optimal packing there are at most 10 items per bin and that there is an algorithm for Bin Packing which runs in time $f(n)$ for some function f. Show that then there is an algorithm for finding an optimal packing (one that used the smallest possible number of bins) which runs in time $O(p(f(n))$ for some polynomial p.