CS466 — Fall 1999

Problem Set 1

Assigned problems

These problems are due October 5 in class. Do your own work.

1. When using a recursive algorithm one must select an optimal problem size n_0 at which to terminate the recursion and to solve subproblems directly. This problem investigates the selection of optimal size. For simplicity, it concerns only one algorithm—Strassen's algorithm for multiplication of matrices (BB, pp. 242-3; CLR, pp. 739–743).

   Let $M_S(n)$ be the number of scalar multiplications and $A_S(n)$ the number of scalar additions required to multiply two $n \times n$ matrices using Strassen's algorithm. Let $M_d(n)$ be the number of scalar multiplications and $A_d(n)$ the number of scalar additions required to multiply two $n \times n$ matrices directly from the definition of matrix product.

   Strassen's algorithm has the recurrence relations

$$
\begin{aligned}
M_S(n) &= \begin{cases} M_d(n) & \text{if } n \le n_0, \\ 7M_S(\frac{n}{2}) & \text{if } n > n_0, \end{cases} \\
A_S(n) &= \begin{cases} A_d(n) & \text{if } n \le n_0 \\ 7A_S(\frac{n}{2}) + 18(\frac{n}{2})^2 & \text{if } n > n_0. \end{cases}
\end{aligned}
$$

   (The BB version replaces the 18 by a 24. Use the 18.)

   (a) Give the exact solution for $M_S(n)$ and $A_S(n)$ for inputs of the form $n = 2^i n_0$.

   (b) Find the value of n_0 that minimizes the number of scalar operations.

   (c) The number of scalar operations does not tell the whole story, of course. Choose *one* of the following, and discuss its effects. You do not need exact calculations but rather thoughtful consideration.

      i. Suppose that the cost of scalar operations depends on the size of the operands. Present a simple model of this situation, and discuss how the best value of n_0 changes. (You do not need an exact formula for n_0.)

      ii. Strassen's algorithm requires some overhead to control the recursion and to form the intermediate results. Discuss how to incorporate this overhead into the above formulas, and explain how it will affect the final result.

      iii. Since Strassen's algorithm involves many subtractions, it can introduce large round-off errors when used for floating-point calculations. Suggest some alternative representations of numbers (i.e., substitutes for floating point) and discuss how use of these representations will affect the trade-offs for Strassen's algorithm.

2. A *shuffle* of s sequences is any sequence that contains each element of each sequence in the same order—that is, a sequence that can be partitioned into s sorted subsequences. Define

$$\mathit{Shuf}(X) = \min_s \{X \text{ is the shuffle of } s \text{ sequences}\}.$$

   (a) Find the worst-case time required by Odd-Even-Straight Mergesort on sequences X with $\mathit{Shuf}(X) \leq s$.

   (b) Show that $\Omega(n \log s)$ comparisons are necessary.

   (c) Since the bounds in (a) and in (b) don't match, one wants an improvement.

   Give an efficient algorithm that can unshuffle a shuffle of s sequences, given s in advance. That is, the algorithm receives input (X, s) and should either partition X into s sorted subsequences or determine that $\mathit{Shuf}(X) > s$. Use $O(n \log s)$ steps.

   (d) (Extra credit) Show how to sort a shuffle of s sequences in $O(n \log s)$ time given only the shuffled sequence, not the value of s.

3. (CLR Problem 18-2, p. 376. Binary search with fast amortized insertions.)

   To store a set of up to n items, keep $\lceil \log n \rceil$ arrays of sizes 2^i for i from 0 to $\log n$. At any one time, array i will either be empty or contain 2^i items in sorted order. The items in two different arrays have no necessary relation between them.

   (a) Explain how to perform a search in this structure. Analyze the worst-case time for a search.

   (b) Explain how to insert a new element into the structure, assuming that less than n were present. Analyze the worst-case and amortized running times for a sequence of searches and insertions.

   (c) Explain how to delete an element from the structure. Analyze the worst-case and amortized running times for a sequence of searches, insertions and deletions.

   Suggest a way to modify the requirements of the data structure to improve the time in the case that deletions may occur. (You do not need a precise analysis.)

4. Inputs:

   undirected graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$

   weight function $w[u, v]$

   Output:

   minimum-weight spanning tree, as a set T of edges.

   Algorithm:

   $T \leftarrow \emptyset$

   For $i \leftarrow 1$ to n

   $\quad V_i \leftarrow \{i\}$

   $\quad E_i \leftarrow \{\text{edges incident to } i\}$

   [*Invariant: Sets V_i are disjoint; E_i contains all edges incident to members of V_i.*]

   While more than one set V_i remains

   $\quad$ choose any set V_i

   $\quad (u, v) \leftarrow$ minimum weight edge of (E_i), removing it

   $\quad V_i \leftarrow$ set containing u; $V_j \leftarrow$ set containing v;

   $\quad$ **If** $i \neq j$

   $\quad\quad$ add (u, v) to T

   $\quad\quad V_i \leftarrow V_i \cup V_j$; destroy V_j.

   $\quad\quad E_i \leftarrow E_i \cup E_j$

   (a) Prove that the algorithm is correct.

   (b) Explain how to implement it using mergeable heaps.

   (c) Compare the asymptotic running times using binomial heaps and using Fibonacci heaps. Which do you expect will be faster in practice?