Advanced Algorithms, Fall 2012

Prof. Bernard Moret

Homework Assignment #8

due Sunday night, Dec. 1

Write your solutions in LaTeX using the template provided on the Moodle and web sites and upload your PDF file through Moodle by 4:00 am Monday morning, Dec. 2.

Question 1.
Perhaps the simplest dynamic program is one that finds all pairwise shortest paths between the vertices of an undirected graph. You are given a graph by its adjacency matrix, in which entry (i, j) is either ∞ (say the maximum integer representable in your machine) if there is no edge between i and i, or the length of the edge $\{i, j\}$ (a positive integer) otherwise. The output is a pairwise distance matrix giving the length of the shortest path between any two vertices. Show how to formulate this algorithm with three nested loops and verify that it runs in cubic time.

Question 2.
You are given a free tree T with n nodes. (A free tree is not rooted, its edges are undirected, and the degree of a node is not bounded—in other words, it is just a connected acyclic undirected graph.) Each edge is assigned a weight, which may be positive, zero, or negative. Devise a linear-time dynamic program to find a path in T such that the sum of the edge weights on this path is minimized.

Question 3.
You are given a convex polygon represented by the list of its vertices, say in counterclockwise order along the perimeter, p_1, p_2, $\ldots$, p_n. Devise an $\Theta(n^3)$ dynamic program to find a triangulation of this polygon such that the sum of the (Euclidean) lengths of the $(n-3)$ added edges is minimized. (A triangulation is a collection of *chords*, that is, edges between polygon vertices that cross the interior of the polygon, that partition the interior of the polygon into triangles. A simple polygon of n vertices always needs exactly $n-3$ chords for a triangulation.)

Question 4.
We saw how to use divide-and-conquer to reduce the cost of multiplying two square matrices. Now consider using dynamic programming to reduce the cost of multiplying a chain of rectangular matrices. You are given matrices A_i, $I = 1, \ldots, n$; these matrices are rectangular (i.e., not necessarily square), but the number of columns of matrix A_i always equals the number of rows of matrix A_{i+1}, for all i, $1 \leq i < n$. You want to compute the product $\Pi_{i=1}^{n} A_i$ as efficiently as possible, using plain matrix multiplication. That is, multiplying an $k \times l$ matrix by a $l \times m$ matrix to produce a $k \times m$ matrix will cost $\Theta(klm)$ time. Since matrix multiplication is associative, you can parenthesize the product in any properly nested way that you want—you can choose any pair of adjacent matrices to multiply first, replacing them by their product, and so on. Different parenthesizations will give different amounts of work, so you are to devise a dynamic programming algorithm to find the optimal parenthesization.