Fall 17: CSci 5421—Advanced Algorithms and Data Structures

Homework 2

Please do all problems; we will grade a subset of the assigned problems (same subset for everyone). Please follow all of the instructions given in the handout for Homework 1.

1. (4+8 points)
   (a) Professor McMult claims that the following approach to the matrix-chain multiplication problem uses the fewest total number of scalar multiplications: Given the sequence $A_1, A_2, \ldots, A_n$ of matrices, first multiply the (adjacent) pair that requires the fewest scalar multiplications. Replace that pair by the resulting matrix and repeat until there is only one matrix left. Give a counterexample to the professor’s conjecture. You need only give the dimensions of the matrices (not the matrices themselves) and show that this strategy is sub-optimal.
   (b) Use the bottom-up (i.e., iterative) algorithm `Matrix-Chain-Order(p)` seen in class to determine the minimum number of multiplications needed to compute the product of a sequence of six matrices, whose dimensions are $p = (p_0, p_1, \ldots, p_6) = (10, 5, 15, 30, 1, 100, 20)$. You must show your work, i.e., the filled-in lookup table, the optimal parenthesization, and its cost.

2. (15 points) Let $A = a_1 a_2 \ldots a_n$ be a string of integers, not necessarily in sorted order. (For simplicity, assume that the integers in $A$ are distinct.) A subsequence, $B$, of $A$ is an increasing subsequence if the integers of $B$ are in increasing order. (For instance, 4,7,10 is an increasing subsequence of 1,4,2,7,5,9,10,8.) The goal is to find a longest increasing subsequence (LIS) of $A$. In the example above, one possible LIS is 1, 4, 7, 9, 10, which has length 5.

   Give a bottom-up dynamic programming algorithm to find an LIS of $A$ in $O(n^2)$ time.

   Your answer should include (i) a brief description of the main ideas, including the dynamic programming recurrence and its justification, (ii) pseudocode, and (iii) an analysis of the running time.

   Note: There is a simple approach that uses the LCS algorithm as a subroutine: Sort $A$ into a new string $A'$ and find the LCS of $A$ and $A'$. Do not use this method; instead, solve the problem from first principles.

   Hint: For $1 \leq i \leq n$, let $\ell_i$ be the length of an LIS of $A$ which terminates at $a_i$.

3. (10 points) Ex. 25.2-4, p. 699. Justify your answer carefully.

4. (12 points) Give a top-down, memoized version of the algorithm `Optimal-BST(p, q, n)`, based on Eq. 15.14 to compute an optimal binary search tree for a set of $n$ keys with search probabilities given by the arrays $p$ and $q$. (You do not have to construct the optimal tree itself; just compute its cost.) Give a careful analysis of the running time, which should be $\Theta(n^3)$. (Use the notion of “type 1” and “type 2” calls in your analysis.)
5. (15 points) Let $A$ be an $n \times n$ matrix of 0’s and 1’s. We would like to determine the largest square subblock of $A$ that consists entirely of 1’s. For instance, the largest square subblock in the matrix $A$ below has its upper left corner at $A(2,3)$ and has size 3.

$$
\begin{array}{cccccc}
0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
$$

Give a bottom-up dynamic programming algorithm for this problem that runs in $O(n^2)$ time. The output need only be the position of the upper left corner of the largest square subblock and its size.

**Hint:** Let $s(i, j)$ be the size of the largest square subblock with its upper-left corner at position $(i, j)$ in $A$.

6. (15 points) Let $P$ be a convex polygon in the plane, with vertices $v_1, v_2, \ldots, v_n$ numbered clockwise around $P$. A triangulation of $P$ is a partition of its interior into non-overlapping triangles, where the vertices of the triangles are vertices of $P$. (See figure.) The weight of a triangulation of $P$ is the sum of lengths of the perimeters of its triangles. For example, the triangulation in the figure has weight $(|v_1v_2| + |v_2v_3| + |v_3v_1|) + (|v_1v_3| + |v_3v_5| + |v_5v_1|) + (|v_3v_4| + |v_4v_5| + |v_5v_3|)$. (Here $|v_iv_j|$ denotes the length of edge $v_iv_j$.)

There are exponentially-many different triangulations of $P$ and different ones can have different weights. Give a bottom-up dynamic programming algorithm to compute a minimum-weight triangulation of $P$ in $O(n^3)$ time. The output should be the weight of the optimal triangulation and sufficient information to be able to construct it. (You do not have to write the routine to construct it though.) Assume the availability of a routine $Peri(i, j, k)$ which returns the perimeter of triangle $v_iv_jv_k$ in constant time.

Your answer should include (a) the key ideas behind your approach, including the dynamic programming recurrence and a brief justification for it, (b) pseudocode, and (c) an analysis of the running time.

**Hint:** In any triangulation of $P$, edge $v_1v_n$ belongs to some triangle $v_1v_nv_k$. Use this triangle to split the problem into subproblems. Let $c(i,j)$ be the weight of a minimum-weight triangulation of a convex polygon with vertices $v_i, \ldots, v_j$, $1 \leq i < j \leq n$. 