Algorithms (2IL15): Homework Set A.2

Exercises for Lecture 3

A.2-1 ($\frac{1}{2} + \frac{1}{2}$ point) Someone claims that the matrix-multiplication problem (see the slides for Lecture 3) can also be solved by a greedy algorithm, as follows. Suppose we want to compute $A_1 \cdot A_2 \cdots A_n$, where A_i is a $p_{i-1} \times p_i$ matrix (for $1 \leqslant i \leqslant n$), in the cheapest manner. If $n = 1$ then we are done. Otherwise we find an index i (with $1 \leqslant i < n$) such that $p_{i-1}p_ip_{i+1}$ (which is the cost for computing $A_i \cdot A_{i+1}$) is minimized; we then compute $A_i \cdot A_{i+1}$, replace the two matrices A_i, A_{i+1} by the new matrix $A_i \cdot A_{i+1}$, and we recursively compute the product of the resulting $n - 1$ matrices.

(i) Suppose that this algorithm would be correct, that is, suppose it would always result in an optimal way to multiply the given matrices. Would you then prefer this algorithm over the dynamic-programming algorithm from Lecture 3, or would you prefer the dynamic-programming algorithm? Explain your answer.

(ii) Prove that this greedy algorithm is correct, or give a counterexample that shows that it does not always produce an optimal solution.

A.2-2 ($\frac{1}{2} + 1 + 1 + \frac{1}{2} + 1$ points) Let $X = \{x_1, \ldots, x_n\}$ be a sorted set with n numbers. A *cluster* is a subset $X_{i,j} = \{x_i, \ldots, x_j\}$ with $1 \leqslant i \leqslant j \leqslant n$. Each cluster has a (non-negative) cost, which we denote by $cost(X_{i,j})$. You may assume that these costs have already been computed and are given in an array Cost$[1..n, 1..n]$. (No other information about the costs is given.) A *cluster tree* is an ordered binary tree with n leaves, each representing a number in X, such that the leftmost leaf represents x_1, the next leaf represents x_2, and so on. Each internal node ν in a cluster tree represents a cluster $C(\nu)$, namely the cluster consisting of all numbers represented by the the leaves of the subtree rooted at the node ν. Fig. 1 shows an example of a cluster tree. The cost of a cluster tree $\mathcal{T}$ is defined as the sum of the costs of the clusters of its internal nodes:

$$cost(\mathcal{T}) := \sum_{\nu \text{ is an internal node of } \mathcal{T}} cost(C(\nu)).$$

Our goal is to develop a dynamic-programming algorithm that computes a cluster tree with minimum cost.

(i) Define a suitable subproblem in terms of a few parameters, and a corresponding variable that gives the value of optimal solution to the subproblem.

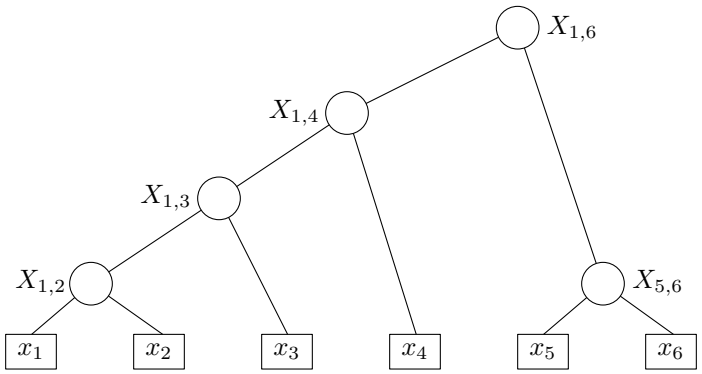

Figure 1: Example of a cluster tree on a set of six numbers. The clusters represented by the internal nodes are also indicated.

(ii) Give a recursive formula for the variable you defined in (i), and prove that your formula is correct.

(iii) Give pseudocode for a dynamic-programming algorithm that computes the cost of an optimal cluster tree for a given set of numbers. The input to your algorithm should be the array $\text{Cost}[1..n, 1..n]$ that specifies the costs of the possible clusters. Your algorithm should not use memoization, but fill in a table in a suitable order.

(iv) Analyze the running time of your algorithm.

(v) Extend the algorithm so that it not only computes the value of an optimal cluster tree, but an optimal tree itself.

Exercises for Lecture 4

A.2-3 ($\frac{1}{2} + 1 + \frac{1}{2}$ points) Suppose you must drive a distance D along a highway. You start your trip with a full tank of petrol, but on a full tank you can only drive distance $d_{\max}$ for some $d_{\max} < D$. Fortunately there are many petrol stations along the highway. The distance of petrol station i ($1 \leqslant i \leqslant n$) to the starting point of your trip is $d(i)$, where $d(1) < d(2) < \cdots < d(n)$. The petrol stations are quite busy, however, and when you stop at station i you have to wait $t(i)$ minutes before you can fill up your tank. Your goal is to select the petrol stations where you stop and fill up your tank, in such a way that the total waiting time is minimized. (Note that this problem is very similar to the problem in exercise A.1-5, except that now we take the waiting times at the petrol stations into account.) The problem can be stated more formally as follows.

A sequence $S = \langle i_1, \ldots, i_m \rangle$ of petrol stations, with $i_1 < i_2 < \cdots < i_m$, is *feasible* when

- $d(i_1) \leqslant d_{\max}$,
- $d(i_{j+1}) - d(i_j) \leqslant d_{\max}$ for all $1 \leqslant j < m$, and
- $D - d(i_m) \leqslant d_{\max}$.

You can assume that $d(1) \leqslant d_{\max}$, and $d(i+1) - d(i) \leqslant d_{\max}$ for all $1 \leqslant i < n$, and $D - d(n) \leqslant d_{\max}$, so that there is at least one feasible sequence. The *cost* of the sequence $S = \langle i_1, \ldots, i_m \rangle$, denoted $cost(S)$, is defined as $\sum_{j=1}^{m} t(i_j)$. The problem is now to find a feasible sequence S that minimizes $cost(S)$. In this exercise you have to develop a dynamic-programming algorithm for this problem. For simplicity you only have to compute the value of an optimal solution, and not an optimal solution itself. Your algorithm should run in $O(n^2)$ time.

(i) Define a suitable subproblem in terms of one or more parameters, and a corresponding variable that gives the value of optimal solution to the subproblem.

(ii) Give a recursive formula for the variable you defined in (i), and prove that your formula is correct.

(iii) Give pseudocode for a dynamic-programming algorithm that computes the value of an optimal solution to the problem.

A.2-4 ($\frac{1}{2} + 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{2}$ point) Let $P = \langle p_1, \ldots, p_n \rangle$ be a sequence of n points in the plane $\mathbb{R}^2$. No two points have the same x-coordinate, and the points are given from left to right. Any subsequence P' of P defines a path, which is given by visiting the points from P' in order (that is, from left to right). The cost of this path, $cost(P')$, is its total length. For example, if $P' = \langle p_2, p_4, p_5, p_8 \rangle$ then $cost(P') = |p_2p_4| + |p_4p_5| + |p_5p_8|$, where $|p_ip_j|$ denotes the length of the segment p_ip_j. We want to partition P into two subsets P_1, P_2 such that $cost(P_1) + cost(P_2)$ is minimized.

(i) Define a suitable subproblem in terms of one or more parameters, and a corresponding variable that gives the value of optimal solution to the subproblem.

(ii) Give a recursive formula for the variable you defined in (i), and prove that your formula is correct.

(iii) Give pseudocode for a dynamic-programming algorithm that computes the value of an optimal solution to the problem.

(iv) Analyze the running time of your algorithm.

(v) Extend the algorithm so that it not only computes the value of an optimal partition, but the partition itself.