CSCI3160: Regular Exercise Set 7

Prepared by Yufei Tao

Problem 1. Let x be a string of length n, and y a string of length m. Define $opt(i,j)$ to be the length of an LCS of $x[1:i]$ and $y[1:j]$ for $i \in [0,n]$ and $j \in [0,m]$. In the lecture, we already discussed how to calculate $opt(i,j)$ for all possible (i,j) pairs. Based on that discussion, explain an algorithm that can output an LCS of x and y in $O(nm)$ time.

Problem 2 (Matrix-Chain Multiplication). The goal in this problem to calculate $\boldsymbol{A}_1\boldsymbol{A}_2...\boldsymbol{A}_n$ where $\boldsymbol{A}_i$ is an $a_i \times b_i$ matrix for $i \in [1,n]$. This implies that $b_{i-1} = a_i$ for $i \in [2,n]$, and the final result is an $a_1 \times b_n$ matrix. You are given an algorithm $\mathcal{A}$ that, given an $a \times b$ matrix $\boldsymbol{A}$ and a $b \times c$ matrix $\boldsymbol{B}$, can calculate $\boldsymbol{AB}$ in $O(abc)$ time. To calculate $\boldsymbol{A}_1\boldsymbol{A}_2...\boldsymbol{A}_n$, you can apply *parenthesization*, namely, convert the expression to $(\boldsymbol{A}_1...\boldsymbol{A}_i)(\boldsymbol{A}_{i+1}...\boldsymbol{A}_n)$ for some $i \in [1,n-1]$, and then parenthesize each of $\boldsymbol{A}_1...\boldsymbol{A}_i$ and $\boldsymbol{A}_{i+1}...\boldsymbol{A}_n$ recursively. A *fully parenthesized product* is

- either a single matrix or
- the product of two fully parenthesized products.

For example, if $n = 4$, then $(\boldsymbol{A}_1\boldsymbol{A}_2)(\boldsymbol{A}_3\boldsymbol{A}_4)$ and $((\boldsymbol{A}_1\boldsymbol{A}_2)\boldsymbol{A}_3)\boldsymbol{A}_4$ are fully parenthesized, but $\boldsymbol{A}_1(\boldsymbol{A}_2\boldsymbol{A}_3\boldsymbol{A}_4)$ is not. Each fully parenthesized product has a computation cost under $\mathcal{A}$; e.g., given $(\boldsymbol{A}_1\boldsymbol{A}_2)(\boldsymbol{A}_3\boldsymbol{A}_4)$, you first calculate $\boldsymbol{B}_1 = \boldsymbol{A}_1\boldsymbol{A}_2$ and $\boldsymbol{B}_2 = \boldsymbol{A}_3\boldsymbol{A}_4$, and then calculate $\boldsymbol{B}_1\boldsymbol{B}_2$, all using $\mathcal{A}$. The cost of the fully parenthesized product is the total cost of the three pairwise matrix multiplications.

Design an algorithm to find in $O(n^3)$ time a fully parenthesized product with the smallest cost.

Problem 3 (Longest Ascending Subsequence). Let A be a sequence of n distinct integers. A sequence B of integers is a *subsequence* of A if it satisfies one of the following conditions:

- $A = B$ or
- we can convert A to B by repeatedly deleting integers.

The subsequence B is *ascending* if its integers are arranged in ascending order. Design an algorithm to find an ascending subsequence of A with the maximum length. Your algorithm should run in $O(n^2)$ time. For example, if $A = (10, 5, 20, 17, 3, 30, 25, 40, 50, 60, 24, 55, 70, 58, 80, 44)$, then a longest ascending sequence is $(10, 20, 30, 40, 50, 60, 70, 80)$.

Problem 4*. In this problem, we will revisit a regular exercise discussed before and derive a faster algorithm using dynamic programming.

Let A be an array of n integers (A is not necessarily sorted). Each integer in A may be positive or negative. Given i, j satisfying $1 \le i \le j \le n$, define *subarray* $A[i:j]$ as the sequence $(A[i], A[i+1], ..., A[j])$, and the *weight* of $A[i:j]$ as $A[i] + A[i+1] + ... + A[j]$. For example, consider $A = (13, -3, -25, 20, -3, -16, -23, 18)$; $A[1:4]$ has weight 5, while $A[2:4]$ has weight -8. Design an algorithm to find a subarray of A with the largest weight in $O(n)$ time.

Remark: We solved the problem using divide-and-conquer in $O(n \log n)$ time before.