CSC373 Worksheet 3

July 28, 2020

Source: link

1. **CLRS 15.2-1:** Find an optimal parenthesization of a matrix-chain product whose sequence of dimension is $< 5, 10, 3, 12, 5, 50, 6 >$

2. **CLRS 15.2-2:** Give a recursive algorithm $MATRIX-CHAIN-MULTIPLY(A, s, i, j)$ that actually performs the optimal matrix-chain multiplication, given the sequence o matrices $< A_1, A_3, ..., A_n >$, the s table computed by $MATRIX - CHAIN - ORDER$, and the indices i and j. (The initial call would be $MATRIX-CHAIN-MULTIPLY(A, s, 1, n)$).

3. **CLRS 15.2-5:** Let $R(i, j)$ be the number of times that table entry $m[i, j]$ is referenced while computing other table entries in call of $MATRIX - CHAIN - ORDER$. Show that the total number of references for the entire table is

$$\sum_{i=1}^{n}\sum_{j=1} R(i, j) = \frac{n^3 - n}{3}$$

4. **CLRS 15.3-2:** Draw the recursion tree for $MERGE - SORT$ procedure from Section 2.3.1 on an array of 16 elements. Explain why memoization fails to speed up a good divide-and-conquer algorithm such as $MERGE - SORT$

5. **CLRS 15.3-3:** Consider a variant of the matrix-chain multiplication problem in which the goal is to parenthesize the seqnce of matrices so as to maximize, rather than minimize the number of scalar multiplications. Does this problem exhibit optimal substructure?

6. **CLRS 15.3-4:** As stated, in dynamic programming we first solve hte subproblems and then choose which of them to use in an optimal solution to the problem. Professor Capulet claims that we do not always need to solve all the subproblems in order to find an optimal solution. She suggests that we can find an optimal solution to the matrix chain multiplication problem by always choosing the matrix A_k at which to split the subproduct $A_i A_{i+1} ... A_j$

7. **CLRS 15.4-1:** Determine an LCS of $< 1, 0, 0, 1, 0, 1, 0, 1 >$ and $< 0, 1, 0, 1, 1, 0, 1, 1, 0 >$

8. **CLRS 15.4-2:** Give pseudocode to reconstruct an LCS from the completed c table and the original sequences $X =< x_1, x_2, ..., x_m >$ and $Y =< y_1, y_2, ..., Y_n >$ in $O(m+n)$ time, without using the b table.

9. **CLRS 15.4-6:** Give an $O(n \lg n)$-time algorithm to find the longest monotonically increasing subsequence of a seuence of n numbers. (Hint: Observe that the last element of a candidate subseqneuce of length i is at least as large as the last element of a candidate subseqneunce of length $i-1$. Maintain candidate subsequences by linking them through the input sequence)

10. **DPV 6.7:** A sequence is *palindromic* if it is the same whether read left to right or right to left. For instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsequences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is not palindromic). Devise an algorithm that takes a sequence $x[1...n]$ and returns the (length of the) longest palindromic subsequence. It's running time should be $O(n^2)$.

11. **DPV 6.21:** A *vertac cover* of a graph $G = (V, E)$ is a subset of verticies $S \subseteq V$ that includes at least one endpoint of every edge in E. Give a linear-time algorithm for the following task

Input: An undirected tree $T = (V, E)$

Output: The size of the smallest vertex cover of T

For instance, in the following tree, possible vertex covers include $\{A, B, C, D, E, F, G\}$ and $\{A, C, D, F\}$ but not $\{C, E, F\}$. The smallest vertex cover has size 3: $\{B, E, G\}$.