CSE 431/531: Algorithm Analysis and Design — Fall 2022

Homework 4

Instructor: Shi Li — **Deadline: 11/13/2022**

Your Name: Zheyuan Ma Your Student ID: 50321597

Problems	1	2	3	Total
Max. Score	20	30	30	80
Your Score				

The best way to solve Problems 2 and 3 is to use the following steps:

1. Give the definition of the cells in the dynamic programming table.
2. Show the recursions for computing the cells.
3. Give the order in which you compute the cells.
4. Briefly state why the algorithm achieves the desired running time.

Problem 1 Solve the matrix-chain-multiplication instance with the following sizes.

matrix	A_1	A_2	A_3	A_4	A_5
size	3×4	4×10	10×6	6×8	8×7

You need to fill the following two tables for the *opt* and π values, give the minimum cost of the instance (i.e., the number of multiplications), and describe the best way to multiply the matrices (using either a tree, or a formula with parenthesis).

$opt[i,j]$ (i \ j)	1	2	3	4	5
1	0	120	300	444	612
2		0	240	432	656
3			0	480	756
4				0	336
5					0

$\pi[i,j]$ (i \ j)	1	2	3	4	5
1		1	2	3	4
2			2	3	4
3				3	3
4					4

Table 1: *opt* and π values for the matrix chain multiplication instance.

The minimum cost for the instance is 612.
Describe the best way to multiple the matrices:

$(((A_1A_2)A_3)A_4)A_5$

Problem 2 An independent set of a graph $G = (V, E)$ is a set $U \subseteq V$ of vertices such that there are no edges between vertices in U. Given a graph with node weights, the maximum-weight independent set problem asks for the independent set of a given graph with the maximum total weight. In general, this problem is very hard. Here we want to solve the problem on trees: given a tree with node weights, find the independent set of the tree with the maximum total weight. For example, the maximum-weight independent set of the tree in Figure 1 has weight 47.

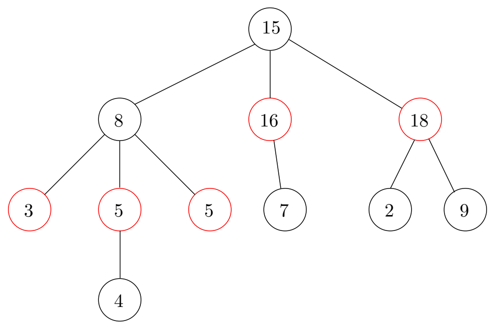

Figure 1: The maximum-weight indpendent set of the tree has weight 47. The red vertices give the independent set.

Design an $O(n)$-time algorithm for the problem, where n is the number of vertices in the tree. We assume that the nodes of the tree are $\{1, 2, 3, \cdots, n\}$. The tree is rooted at vertex 1, and for each vertex $i \in \{2, 3, \cdots, n\}$, the parent of i is a vertex $j < i$. In the input, we specify the weight w_i for each vertex $i \in \{1, 2, 3, \cdots, n\}$ and the parent of i for each $i \in \{2, 3, \cdots, n\}$.

Problem 3 Given a sequence $A = (a_1, a_2, \cdots, a_n)$ of n numbers, we need to find the longest increasing sub-sequence of A. That is, we want to find a maximum-length sequence $(i_1, i_2, \cdots, i_t)$ of integers such that $1 \leq i_1 < i_2 < i_3 < \cdots < i_t \leq n$ and $a_{i_1} < a_{i_2} < a_{i_3} < \cdots < a_{i_t}$.

For example, if the input $n = 11$, $A = (30, 60, 20, 25, 75, 40, 10, 50, 90, 70, 80)$, then the longest increasing sub-sequence of A is $(20, 25, 40, 50, 70, 80)$, which has length 6. The correspondent sequence of indices is $(3, 4, 6, 8, 10, 11)$.

Again, you only need to output the length of the longest increasing sub-sequence. Design an $O(n^2)$-time dynamic programming algorithm to solve the problem.

Problem 2

1) the table has 1 row and each cell is opt [j], maximum total weight of the independent set of the subtree rooted at vertex j

2) $$opt[j] = \max \begin{cases} \sum_{i \text{ childs of } j} opt[i] \\ \sum_{i \text{ grandchilds of } j} opt[i] + \omega_j \end{cases}$$
(mark the node j for retrival)

Tree is rooted at vertex 1
opt [1] is the maximum total weight of the independent set of tree

3) We need to compute the tree bottom-up. That is, we first compute opt[n], then opt[n-1] ... (post-order traversal)

4) There are n iterations (from n to 1), and a child value opt[i] is accessed only by its parent and grandparent. It takes O(n) time since we visit nodes in postorder and examine each edge exactly once.

Problem 3

1) for every $i \in [1, n]$, opt[i] is the length of the longest increasing subsequence of A that ends at A[i]

2) $$opt[i] = \max_{j<i,\ A[j]<A[i]} opt[j] + 1$$, and if there is no A[j]<A[i] for every j<i, opt[i] = 1
the final result is $\max_{i \in [1,n]} opt[i]$

3) We need to compute opt[i] from 1 to n

4) There are n iterations, and in each iteration, it takes O(n) time to traverse back to find the max opt[j] value, so total running time is O(n²)