CSE 431/531: Algorithm Analysis and Design Fall 2022

Homework 4 Solutions

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

Your Name: _____ Your Student ID: _____

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.

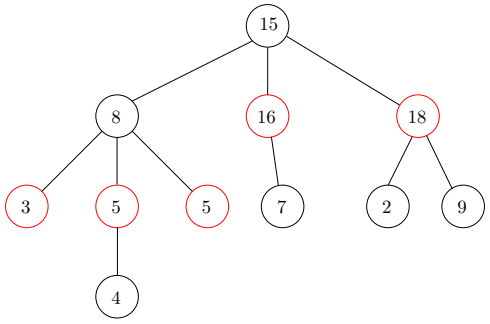

Figure 1: The maximum-weight indpendent set of the tree has weight 47. The red vertices give the independent set.

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 $i \in \{2, 3, \cdots, n\}$.

For every vertex $i \in \{1, 2, \cdots, n\}$, we let T_i be the sub-tree of T rooted at i.

1. We let $f[i, 0]$ to denote the weight of the maximum-weight independent set in T_i, that does not contain the vertex i. We use $f[i, 1]$ to denote the weight of the maximum-weight independent set in T_i, that may or may not contain vertex i.
2. For a leaf vertex i, we have $f[i, 0] = 0$ and $f[i, 1] = w_i$. For a non-leaf vertex i, we have

$$f[i, 0] = \sum_{j \text{ is a child of } i} f[j, 1],$$

$$f[i, 1] = \max \left\{ f[i, 0], w_i + \sum_{j \text{ is a child of } i} f[j, 0] \right\}.$$

The final answer is $f[1, 1]$.

(This part is the explanation of the recursion. To get a full score, you do not need this part.) That is, to compute $f[i, 0]$, we can not choose the vertex i. Then, for each child j of i, we need to choose the maximum weight independent set, which may or may not contain j; the weight of the set is exactly $f[j, 1]$. So we have the above formula to compute $f[i, 0]$. To compute $f[i, 1]$, we can choose to include or not include i in the independent set. If we do not include i, then the maximum weight we can achieve is exactly $f[i, 0]$. If we include i, then we get the weight w_i.

For every child j of i, we can not include j in the independent set for the sub-tree T_j. Thus, the maximum weight we can achieve in the subtree T_j is $f[j,0]$. So we have the above recursion for $f[i,1]$.

3. We compute $f[i,0]$ and $f[i,1]$ values in decreasing order of i. That is, we compute $f[n,0], f[n,1], f[n-1,0], f[n-1,1], \cdots, f[1,0], f[1,1]$ in this order.
4. The running time for computing $f[i,0]$ and $f[i,1]$ is $O(\text{degree of } i)$. As the total degree of all vertices is $2(n-1)$, the running time of the algorithm is $O(n)$.

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 $1 \le i_1 < i_2 < i_3 < \cdots < i_t \le n$ 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.

1. We use $f[i]$ to denote the length of the longest increasing sub-sequence with the last element being $A[i]$.
2. The formula for computing $f[i]$ is as follows:

$$f[i] = \max_{j<i: A[j]<A[i]} f[j] + 1,$$

where we assume max is 0 if there is no $j < i$ satisfying $A[j] < A[i]$.

The final answer is $\max_{i\in\{1,2,\cdots,n\}} f[i]$.

3. We compute $f[i]$ for $i = 1, 2, 3, \cdots, n$, in this order.
4. The running time for computing each $f[i]$ is at most $O(n)$, and we need to compute n values of $f[i]$. So overall the running time is $O(n^2)$.