CSE 431/531: Algorithm Analysis and Design Spring 2022

Homework 4

Instructor: Shi Li **Deadline: 4/20/2022**

Your Name: _____ Your Student ID: _____

Problems	1	2	3	4	Total
Max. Score	20	25	25	25	95
Your Score					

The total score for the 4 problems is 95, but your score will be truncated at 80. The best way to solve Problems 2, 3 and 4 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 Consider the following optimum binary search tree instance. We have 5 elements e_1, e_2, e_3, e_4 and e_5 with $e_1 < e_2 < e_3 < e_4 < e_5$ and their frequencies are $f_1 = 5, f_2 = 25, f_3 = 15, f_4 = 10$ and $f_5 = 30$. Recall that the goal is to find a binary search tree for the 5 elements so as to minimize $\sum_{i=1}^{5} \text{depth}(e_i) f_i$, where $\text{depth}(e_i)$ is the depth of the element e_i in the tree. You need to output the best tree as well as its cost. You can try to complete the following tables and show the steps. In the two tables, $opt[i, j]$ is the cost of the best tree for the instance containing $e_i, e_{i+1}, ..., e_j$ and $\pi[i, j]$ is the root of the best tree.

$opt(i,j)$ i \ j	1	2	3	4	5
1	5	35	65	95	170
2		25	55	85	155
3			15	35	90
4				10	50
5					30

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

Table 1: *opt* and π tables for the optimum binary search tree instance. For cleanness of the table, we assume $opt(i, j) = 0$ if $j < i$ and there are not shown in the left table.

$opt[1, 2] = \min\{0 + opt[2, 2], opt[1, 1] + 0\} + (f_1 + f_2) = \min\{0 + 25, 5 + 0\} + (5 + 25) = 35$
$\pi[1, 2] = 2$
$opt[2, 3] = \min\{0 + opt[3, 3], opt[2, 2] + 0\} + (f_2 + f_3) = \min\{0 + 15, 25 + 0\} + (25 + 15) = 55$

$$
\begin{aligned}
\pi[2,3] &= 2 \\
opt[3,4] &= \min\{0 + opt[4,4], opt[3,3] + 0\} + (f_3 + f_4) = \min\{0 + 10, 15 + 0\} + (15 + 10] = 35 \\
\pi[3,4] &= 3 \\
opt[4,5] &= \min\{0 + opt[5,5], opt[4,4] + 0\} + (f_4 + f_5) = \min\{0 + 30, 10 + 0\} + (10 + 30) = 50 \\
\pi[4,5] &= 5 \\
opt[1,3] &= \min\{0 + opt[2,3], opt[1,1] + opt[3,3], opt[1,2] + 0\} + (f_1 + f_2 + f_3) \\
&= \min\{0 + 55, 5 + 15, 35 + 0\} + 45 = 65 \\
\pi[1,3] &= 2 \\
opt[2,4] &= \min\{0 + opt[3,4], opt[2,2] + opt[4,4], opt[2,3] + 0\} + (f_2 + f_3 + f_4) \\
&= \min\{0 + 35, 25 + 10, 55 + 0\} + 50 = 85 \\
\pi[2,4] &= 2 \text{ or } 3 \\
opt[3,5] &= \min\{0 + opt[4,5], opt[3,3] + opt[5,5], opt[3,4] + 0\} + (f_3 + f_4 + f_5) \\
&= \min\{0 + 50, 15 + 30, 35 + 0\} + 55 = 90 \\
\pi[3,5] &= 5 \\
opt[1,4] &= \min\{0 + opt[2,4], opt[1,1] + opt[3,4], opt[1,2] + opt[4,4], opt[1,3] + 0\} \\
&\quad + (f_1 + f_2 + f_3 + f_4) \\
&= \min\{0 + 85, 5 + 35, 35 + 10, 65 + 0\} + 55 = 95 \\
\pi[1,4] &= 2 \\
opt[2,5] &= \min\{0 + opt[3,5], opt[2,2] + opt[4,5], opt[2,3] + opt[5,5], opt[2,4] + 0\} \\
&\quad + (f_2 + f_3 + f_4 + f_5) \\
&= \min\{0 + 90, 25 + 50, 55 + 30, 85 + 0\} + 80 = 155 \\
\pi[2,5] &= 3 \\
opt[1,5] &= \min\{0 + opt[2,5], opt[1,1] + opt[3,5], opt[1,2] + opt[4,5], opt[1,3] + opt[5,5], \\
&\quad opt[1,4] + 0\} + (f_1 + f_2 + f_3 + f_4 + f_5) \\
&= \min\{0 + 155, 5 + 90, 35 + 50, 65 + 30, 95\} + 85 = 170 \\
\pi[1,5] &= 3
\end{aligned}
$$

Finally, draw the optimum binary-search-tree (or describe the tree in words if it is inconvenient for you to draw).

The optimum tree has cost 170. It is rooted at e_3. The left child of e_3 is e_2 and the right child of e_3 is e_5. e_2 has left child being e_1, and e_5 has left child being e_4.

Problem 2 In this problem, we consider the weighted interval scheduling problem, with an additional constraint that the number of intervals we choose is at most k.

Formally, we are given n intervals $(s_1, f_1], (s_2, f_2], \cdots, (s_n, t_n]$, with positive integer weights $w_1, w_2, \cdots, w_n$, and an integer $k \in [n]$. The output of the problem is a set $S \subseteq [n]$ with $|S| \leq k$ such that for every two distinct elements $i, j \in S$, the intervals $(s_i, f_i]$ and $(s_j, f_j]$ are disjoint. Our goal is to maximize $\sum_{i \in S} w_i$.

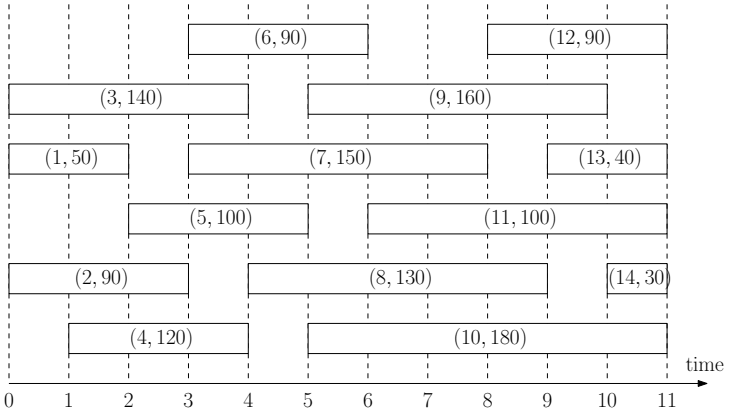

Figure 1: Weighted Job Scheduling Instance. Each rectangle denotes an interval. The first number on each rectangle is the index of the interval and the second number is its weight.

For example, consider the instance in Figure 1, with $k = 3$. Then the maximum weight we can obtain is 330, by choosing the set $\{2, 7, 12\}$ or $\{1, 5, 10\}$. If $k = 4$, then the maximum weight we can obtain is 340, with the set $\{1, 5, 9, 14\}$.

For simplicity, you only need to output the maximum weight that can be obtained. Design an $O(n \log n + nk)$-time dynamic programming algorithm to solve the problem.

We first sort all the jobs in non-decreasing order of f_j values. After renaming the jobs, we assume $f_1 \leq f_2 \leq \cdots \leq f_n$. As in the ordinary maximum weight interval scheduling problem, for every job j, let p_j be the maximum index i such that $f_i \leq s_i$; if no such i exists, then let $p_j = 0$.

In the dynamic programming, for every $j \in \{0, 1, 2, \cdots, n\}$ and $k' \in \{0, 1, 2, \cdots, k\}$, we let $opt[j, k']$ denote the maximum weight we can obtain by choosing at most k' disjoint intervals from the first j intervals.

$$opt[j, k'] = \begin{cases} 0 & \text{if } j = 0 \text{ or } k' = 0 \\ \max\{opt[j-1, k'], opt[p_j, k'-1] + w_j\} & \text{otherwise} \end{cases}$$

In the algorithm, we compute $f[j, k']$'s for every j from 0 to n and for every k' from 0 to k using the above formula. We output $f[n, k]$ in the end.

Each p_j can be computed in $O(\log n)$ time using binary search. Thus computing $p_1, p_2, \cdots, p_n$ takes $O(n \log n)$ time. The running time of the dynamic programming is $O(kn)$. So the total running time is $O(n \log n + kn)$.

Problem 3 Given a sequence $A = (a_1, a_2, \cdots, a_n)$ of n numbers, we need to find the longest increasing subsequence 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.

For every $i \in \{1, 2, 3, \cdots, n\}$, let $opt[i]$ be the longest increasing subsequence of A with the last element being $A[i]$. Then, we have $opt[i] = \max_{j<i:A[j]<A[i]} opt[j] + 1$, where we assume the maximum of an empty set is 0.

In the algorithm, we compute $opt[i]$ for every $i = 1, 2, 3, \cdots, n$ using the above formula. We output $\max_{i\in\{1,2,\cdots,n\}} opt[i]$. The running time for computing each $opt[i]$ is at most $O(n)$. So overall the running time is $O(n^2)$.

Problem 4 This problem asks for the maximum weighted independent set in a $2\times n$ size grid. Formally, the set of vertices in the input graph G is $V = \{1, 2\} \times \{1, 2, 3, \cdots, n\} = \big\{(r, c) : r \in \{1, 2\}, c \in \{1, 2, 3, \cdots, n\}\big\}$. Two different vertices (r, c) and (r', c') in V are adjacent in G if and only if $|r - r'| + |c - c'| = 1$. For every vertex $(r, c) \in V$, we are given the weight $w_{r,c} \geq 0$ of the vertex. The goal of the problem is to find an independent set of G with the maximum total weight. (Recall that $S \subseteq V$ is an independent set if there are no edges between any two vertices in S.) See Figure 2 for an example of an instance of the problem.

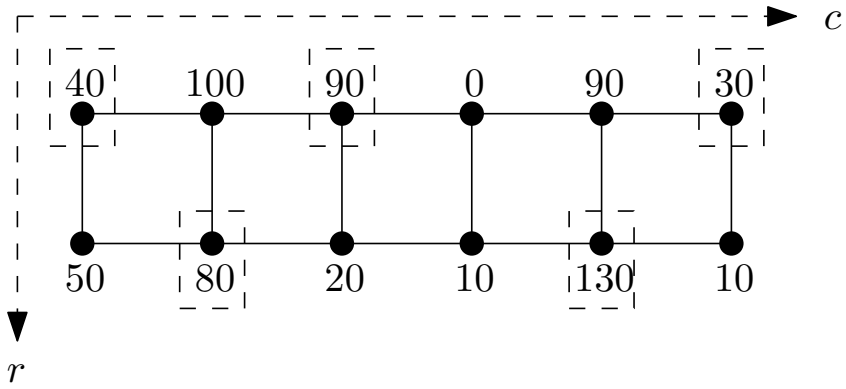

Figure 2: A maximum weighted independent set instance on a 2×6-grid. The weights of the vertices are given by the numbers. The vertices in rectangles form the maximum weighted independent set, with a total weight of 370.

Design an $O(n)$-time dynamic programming algorithm to solve the problem. For simplicity, you only need to output the *weight* of the maximum weighted independent set, not the actual set.

If you could not solve the above problem, you can try to solve the simpler problem when the grid size is $1 \times n$ instead of $2 \times n$ (that is, the input graph is a path on n vertices). If you solve the simpler case correctly, you will get 70% of the score.

For every $i \in \{1, 2, 3, \cdots, n\}$, let G_i be the graph of G induced by the first i columns of vertices. Then for every i, we define

- $opt[i, 0]$ to be the weight of the maximum weight independent set in G_i, that does not contain $(1, i)$ or $(2, i)$,
- $opt[i, 1]$ to be the weight of the maximum weight independent set in G_i, that contains $(1, i)$ (and thus does not contain $(2, i)$),

- $opt[i, 2]$ to be the weight of the maximum weight independent set in G_i, that contains $(2, i)$ (and thus does not contain $(1, i)$).

Then the formula for computing the f table is as follows: $opt[1, 0] = 0, opt[1, 1] = w_{1,1}, opt[1, 2] = w_{2,1}$, and for every $i \geq 2$,

$$opt[i, 0] = \max\{opt[i-1, 0], opt[i-1, 1], opt[i-1, 2]\}$$
$$opt[i, 1] = w_{1,i} + \max\{opt[i-1, 0], opt[i-1, 2]\}$$
$$opt[i, 2] = w_{2,i} + \max\{opt[i-1, 0], opt[i-1, 1]\}$$

For $opt[i, 0]$, we can not choose the two vertices in the i-th column. Thus, we only have the vertices on the first $i-1$ columns. We do not have restrictions on what vertices can be chosen on the $i-1$-th column. So, we have the above recursion.

For $opt[i, 1]$, we need to choose the vertex $(1, i)$ and get the weight $w_{1,i}$. Then for the $(i-1)$-th column, we can not choose the vertex $(1, i-1)$. We may or may not choose the vertex $(2, i-1]$. So we have the recursion for $opt[i, 1]$. The recursion for $opt[i, 2]$ can be obtained similarly.

The algorithm just computes $opt[i, 0], opt[i, 1], opt[i, 2]$ for every i from 1 to n using the above formula. The final output is $\max\{opt[n, 0], opt[n, 1], opt[n, 2]\}$. The running time of the algorithm is $O(n)$.

For the simpler algorithm: Let G_i be the graph induced by the first i vertices. For every $i \in \{0, 1, 2, \cdots, n\}$, let $opt[i]$ denote the maximum weight of an independent set in G_i. Then we have $opt[0] = 0, opt[1] = w_1$, and $opt[i] = \max\{opt[i-1], w_i + opt[i-2]\}$ for every $i = 2, 3, \cdots, n$. In the algorithm, we compute $opt[0], opt[1], \cdots, opt[n]$ using the formula in this order. We return $opt[n]$. The running time for computing each $opt[i]$ is $O(1)$ and the overall running time is $O(n)$.