CSE 431/531: Algorithm Analysis and Design — Fall 2021

Homework 4

Instructor: Shi Li — **Deadline: 11/18/2021**

Your Name: ______________________ Your Student ID: ________________

Problems	1	2	3	Total
Max. Score	25	25	30	80
Your Score				

Problem 1 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 $\sum_{i \in S} w_i$.

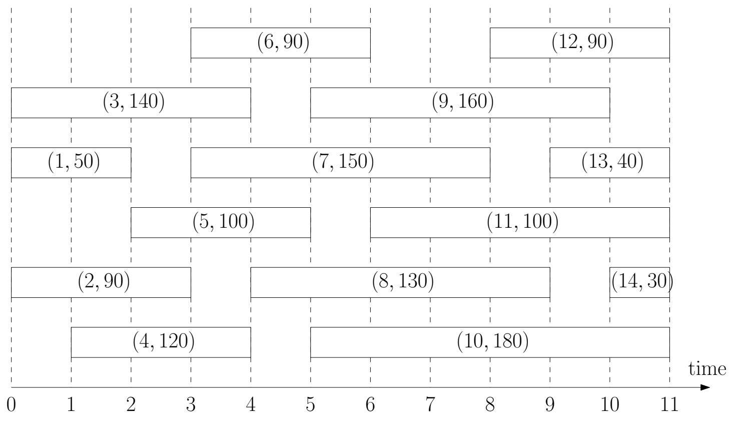

Figure 1: Weighted Job Scheduling Instance. Each rectangle denotes an interval. The first number on each rectangle is the index of the interval and the second number is its weight.

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 $\{1, 5, 9, 14\}$.

Design an $O(n \log n + nk)$-time dynamic programming algorithm to solve the problem. For simplicity, you only need to output the maximum weight that can be obtained. You need to give the definition of the cells and show how to compute the cells.

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 $f_i \le s_j$; if no such i exists, then let $p_j = 0$. 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.

For simplicity of description, we say a subset $S \subseteq [n]$ is an independent set if for every two distinct elements $i, j \in S$, we have that $(s_i, f_i]$ and $(s_j, f_j]$ are disjoint. So, the goal of the problem is to find the maximum weight independent set of size at most k.

In the dynamic programming, for every $j \in \{0, 1, 2, \cdots, n\}$ and $k' \in \{0, 1, 2, \cdots, k\}$, we let $f[j, k']$ denote the weight of the maximum weight independent set S such that $S \subseteq [j]$ and $|S| \le k'$. Then, $f[j, k']$'s can be computed as follows:

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

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. The running time of the dynamic programming is $O(kn)$. Taking the running time for sorting the jobs and computing p_j values into account, the total running time is $O(n \log n + kn)$.

Problem 2 (From leetcode.com) There are n balloons in a row. Each balloon is painted with a positive integer number on it. You are asked to burst all the balloons.

If you burst the i-th remaining balloon in the row, you will get $nums[i-1] \times nums[i] \times nums[i+1]$ coins, where $nums[t]$ for any t is the number on the t-th remaining balloon in the row. If $i - 1 = 0$ or $i + 1$ is more than the number of balloons, then treat it as if there is a balloon with a 1 painted on it.

Return the maximum coins you can collect by bursting the balloons wisely. For example, if there are initially 4 balloons in the row with numbers 3,1,5,8 on them. Then the maximum coins you can get is 167. This is how the array of numbers change when you burst the balloons: $(3, 1, 5, 8) \to (3, 5, 8) \to (3, 8) \to (8) \to ()$. The coins you get is $3 \times 1 \times 5 + 3 \times 5 \times 8 + 1 \times 3 \times 8 + 1 \times 8 \times 1 = 167$.

Design an $O(n^3)$-time algorithm to solve the problem. For convenience, you only need to output the maximum number of coins you can get.

For simplicity, assume the balloons 0 and $n + 1$ have number 1 on them but they can not be burst (so, $nums[0] = nums[n + 1] = 1$). Then in the problem, we need to burst all balloons except for balloons 0 and $n + 1$, and get the maximum coins.

Definition of the cells: for every pair of integers i, j with $0 \le i < j \le n + 1$, we define $f[i, j]$ to be the maximum coins we can get by bursting (and only bursting) all the balloons from $i + 1$ to $j - 1$,

Recursion for computing the cells: for every i, j with $0 \le i < j \le n + 1$

$$f[i,j] = \begin{cases} 0 & \text{if } j - i = 1 \\ \max_{k: i<k<j} (opt[i,k] + opt[k,j] + nums[i] \times nums[k] \times nums[j]) & \text{if } j - i \ge 2 \end{cases}$$

In the algorithm, we computing all the cells $f[i, j]$ in non-decreasing order of $j - i$, and output $f[1, n]$. The algorithm runs in $O(n^3)$ time since there are $O(n^2)$ cells and each cell takes $O(n)$ time to compute.

Problem 3 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.

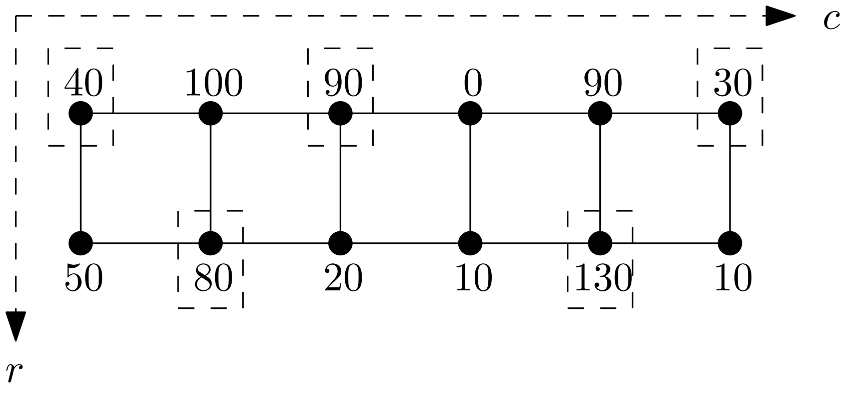

Figure 2: A maximum weighted independent set instance on a 2×6-grid. The weights of the vertices are given by the numbers. The vertices in rectangles form the maximum weighted independent set, with a total weight of 370.

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 verticies). 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

- $f[i, 0]$ to be the weight of the maximum weight independent set in G_i, that does not contain $(1, i)$ or $(2, i)$,
- $f[i, 1]$ to be the weight of the maximum weight independent set in G_i, that contains $(1, i)$ (and thus does not contain $(2, i)$),
- $f[i, 2]$ 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: $f[1, 0] = 0$, $f[1, 1] = w_{1,1}$, $f[1, 2] = w_{2,1}$, and for every $i \geq 2$,

$$\begin{aligned} f[i, 0] &= \max\{f[i-1, 0], f[i-1, 1], f[i-1, 2]\} \\ f[i, 1] &= w_{1,i} + \max\{f[i-1, 0], f[i-1, 2]\} \\ f[i, 2] &= w_{2,i} + \max\{f[i-1, 0], f[i-1, 1]\} \end{aligned}$$

For $f[i,0]$, 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 $f[i,1]$, 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 $f[i,1]$. The recursion for $f[i,2]$ can be obtained similarly.

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