Homework 3

Course: Algorithm Design and Analysis **Semester**: Spring 2024

Instructor: Shi Li **Due Date**: **2024/4/21**

Student Name: _________ Student ID: _________

Problems	1	2	3	4	5	Total
Max. Score	20	15	20	15	30	100
Your Score						

Remarks: All the problems are for dynamic programming.

- To analyze the correctness of dynamic-programming based algorithms, you need to provide three elements:
  (1) the definition of cells (sub-problems) in the dynamic-programming table,
  (2) the formula for computing the values of the cells, including both the boundary cases and the recursive cases,
  (3) and in which order you compute the cells.

  The three elements are sufficient for the proof of the correctness. If you use a pseudo-code instead, it will only cover (2) and (3), but not (1).
- If additionally, some problems ask for the solution that achieves the optimum value, a pseudo-code for outputting the solution is sufficient.
- If the running time of your algorithm is obvious, one sentence describing it is sufficient.

Problem 1. You need to use dynamic programming to solve the Knapsack problem with unlimited copies of each item.

We are given n items indexed by $[n] = \{1, 2, 3, \cdots, n\}$. Each item $i \in [n]$ has an integer value $v_i \geq 1$ and an integer weight $w_i \geq 1$. Each item i has unlimited number of copies. You have a budget of W. The goal of the problem is to buy a set of items with budget W so as to maximize the value. Now we can buy many copies of each item. Formally we need to find a vector $(c_1, c_2, \cdots, c_n)$, where each c_i is a non-negative integer and $\sum_{i=1}^{n} c_i w_i \leq W$, so as to maximize $\sum_{i=1}^{n} c_i v_i$.

For example, suppose we have $n = 3$ items, with values $v_1 = 16, v_2 = 30$ and $v_3 = 60$ and weights $w_1 = 30, w_2 = 50$ and $w_3 = 80$, and the budget is $W = 200$. Then consider three different solutions:

- Buy 4 copies of item 2. The weight is $50 \times 4 \leq 200$. The value is $30 \times 4 = 120$.
- Buy 2 copies of item 1, 1 copy of item 2, and 1 copy of item 3. The weight is $30 \times 2 + 50 + 80 \leq 20$. The value is $16 \times 2 + 30 + 60 = 122$.
- Buy 1 copy of item 1 and 2 copies of item 3. The weight is $30 + 80 \times 2 \leq 200$. The value is $16 + 60 \times 2 = 136$.

The third solution gives the maximum value. Indeed, it is the best solution for the instance. So, for the instance, the maximum value is 136, and the c_i values are $c_1 = 1, c_2 = 0$ and $c_3 = 2$.

Design an $O(nW)$-time algorithm to solve the problem. You need to output both the maximum value and the c_i's that achieve the value.

Problem 2. 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.

Problem 3. 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.

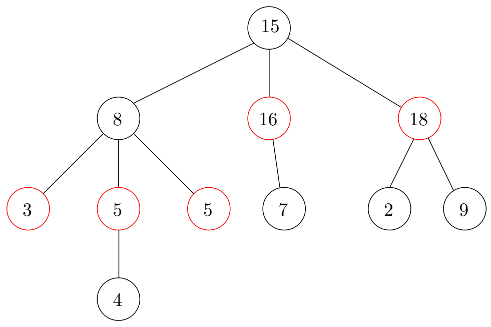

Figure 1: The maximum-weight indpendent set of the tree has weight 47. The red vertices give the independent set.

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

Your algorithm needs to output both the maximum weight, and the indices of the vertices in the independent set that achieves the maximum weight.

Problem 4. Consider the following job-selection problem. Suppose you can undertake one job in each of the following n weeks. The set of possible jobs is divided into those that are low-stress and that are high-stress. The basic question, each week, is whether to take on a low-stress job or a high-stress job.

If you select a low-stress job in week i, then you get a revenue of $l_i > 0$ dollars; if you select a high-stress job, you get a revenue of $h_i > 0$ dollars. The catch, however, is that in order for you to take on a high-stresss job in week i, it is required that you do no job (of either type) in week $i - 1$. On the other hand, it's OK if you take a low-stress job in week i even if you have done a job (of either type) in week $i - 1$.

So, given a sequence of n weeks, a plan is specified by a choice of "low", "high", or "none" for each of the n weeks, with the property that if "high" is chosen for week $i > 1$, then "none" has to be chosen for week $i - 1$. (You can choose "high" for week 1.) The value of the plan is determined in the natural way: for each i, you add l_i to the value if you choose "low" in week i, and you add h_i to the value if you choose "high" in week i. (You add 0 if you choose "none".)

For example, suppose $n = 5$ and the h_i and l_i values are given in the following table:

i	1	2	3	4	5
h_i	50	20	20	10	60
l_i	30	15	10	5	50

Then the maximum revenue you can achieve is 135, achieved by the plan "high, low, low, none, high". In this plan, the revenue you get in the each of the 5 days are 50, 15, 10, 0 and 60.

Design an efficient (i.e, polynomial-time) dynamic-programming algorithm to solve the problem. For convenience you only need to output the maximum revenue you can achieve.

Problem 5. Given an array A of n numbers, we say that a 5-tuple $(i_1, i_2, \cdots, i_{10})$ of integers is inverted if $1 \leq i_1 < i_2 < i_3 < \cdots < i_5 \leq n$ and $A[i_1] > A[i_2] > A[i_3] > \cdots > A[i_5]$.

(a) Give an $O(n^2)$-time algorithm to count the number of inverted 5-tuples w.r.t A.

(b) Improve the running time for (a) to $O(n \log n)$.