CS-E3190 Principles of Algorithmic Techniques

03. Dynamic Programming – Graded Exercise

Please read the following **rules** very carefully.

- Do not consciously search for the solution on the internet.
- You are allowed to discuss the problems with your classmates but you should **write the solutions yourself**.
- Be aware that **if plagiarism is suspected**, you could be asked to have an interview with teaching staff.
- The teaching staff can assist with understanding the problem statements, but will **not be giving any hints** on how to solve the exercises.
- In order to ease grading, we want the solution of each problem and subproblem to start on a **new page**. If this requirement is not met, **points will be deduced**.

1. **Knapsack.** Consider the KNAPSACK: given a knapsack with capacity $C \in \mathbb{N}$, and a set of items $I = \{1, 2, \ldots, n\}$ s.t. item i has value $v_i \in \mathbb{N}$ and weight $w_i \in \mathbb{N}$, select a subset $S \subseteq I$ of the items to pack that maximises value without exceeding the weight capacity C. Note that $\forall i \in I, v_i \leq C$ and $w_i \leq C$. For any selection $S \subseteq I$ let the value be $V(S) = \sum_{i \in S} v_i$ and weight be $W(S) = \sum_{i \in S} s_i$. If S is empty both sums are 0. The optimal value of the Knapsack-problem can be expressed as

$$OPT = \max_{S \subseteq I} \{V(S) \text{ subject to } W(S) \leq C\}$$

Sub-problems. Let $I_k = \{1, \ldots, k\}$ for $k \in \{1, \ldots, n\}$, and $I_0 = \emptyset$. Then (I_k, w) defines a sub-problem in which we maximise the value of a selection $S_k \subseteq I_k$ subject to $W(S_k) \leq w$. Note that $V(0, w) = 0, \forall w$. The optimum value is

$$V(k, w) = \max_{S_k \subseteq I_k} \{V(S_k) \text{ subject to } W(S_k) \leq w\}.$$

It follows from the definition that $V(n, C) = OPT$. Moreover $V(k+1, w) \geq V(k, w), \forall k \in \{0, 1, \ldots, n-1\}, \forall w \geq 0$, since having more items to choose from can only improve the value.

(a) Let $w_k \leq w, \forall k \geq 1$. Consider the sub-problem defined by (k, w) with optimum value $V(k, w)$, and let $S_k^* \subseteq I_k$ be the associated optimum solution.
Prove that the optimal value $V(k, w) = V(S_k^*)$ satisfies:

$$V(S_k^*) = \begin{cases} V(k-1, w) & \text{if } k \notin S_k^* \\ V(k-1, w - w_k) + v_k & \text{if } k \in S_k^*. \end{cases} \tag{1}$$

Hint: Consider the two cases separately, with a contradiction for both.

(b) Prove that, $\forall w \geq 1, \forall k \in \{1, \ldots, n\}$:

$$V(k, w) = \begin{cases} V(k-1, w) & \text{if } w_k > w \\ \max\{V(k-1, w), V(k-1, w - w_k) + v_k\} & \text{if } w_k \leq w \end{cases} \tag{2}$$

Hint: What decisions regarding $S \subseteq I$ do the various terms represent? If proving the claim is difficult, try filling in a table of $V(k, w)$-values on a toy instance.

(c) Give the pseudocode for an $O(nC)$-time dynamic programming algorithm that fills a table of $V(k, w)$-values. A full answer should:

   i. Give the pseudocode for how the table is filled and how the optimum is value returned;

   ii. Argue that the given algorithm runs in $O(nC)$ time and memory.

   You can assume that all integer values take $O(1)$ "units of memory", and that additions and comparisons take $O(1)$ "units of computation".

   Hint: If you get stuck, try filling in a table using the recursion (2) on a toy instance.

(d) Suppose there is a known constant U such that all instances satisfy $w_i \leq U, \forall i \in \{1, \dots, n\}$. Suggest a *simple* modification to your algorithm and *briefly* Prove that it guarantees a memory complexity of $O(nU)$ while maintaining correctness.