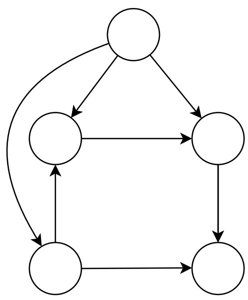
CS-E3190 Principles of Algorithmic Techniques

03. Dynamic Programming – Graded Exercise

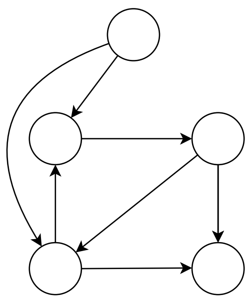
Please read the following **rules** very carefully.

- 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.
- Each week the second exercise is an **individual exercise**, and the teaching staff will not give hints or help with them. You are allowed to ask for hints for the first exercise.
- 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 reduced**.

1. **Basics of dynamic programming.**
   (a) (3p.) Consider the recursive algorithm that was found last week for computing binomial coefficients.
      i. Draw the recursion tree for Binom$(4, 2)$. Draw a DAG based on the recursion tree. Each function call with specific parameters can only appear once in the DAG.
      ii. Design a dynamic algorithm using memoization to improve the recursive algorithm.
      iii. Analyze the runtime and the memory complexity of your algorithm.
   (b) (2p.) For full points you need to justify your answers.
      i. Inspect the following graph. Is it a DAG?



      ii. How about the graph below?



2. **Individual exercise: Knapsack.** Consider the following problem called Knapsack problem. Given a knapsack with capacity $C \in \mathbb{N}$, and a set of items $I = \{1, 2, \ldots, n\}$ such that 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 total value without exceeding the weight capacity C. Denote the total value of the subset S by $V(S) = \sum_{i \in S} v_i$ and the weight by $W(S) = \sum_{i \in S} w_i$. If S is empty both of the 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 $V(0, w) = 0, \forall w$, because there are no items to pack. Denote the optimum value of the subproblem (I_k, w) by

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

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) (2p.) The optimal value $V(k, w) = V(S_k^*)$ satisfies

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

Using this fact, prove that

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

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.

(b) (1p.) Consider the following pseudocode for the recursive algorithm solving the problem. Draw the recursion tree for the problem for the following items and capacity 10.

item	value	weight
1	5	3
2	1	2
3	7	6
4	5	5

Algorithm 1: Knapsack$(I, (v_k, w_k)_{k \in I}, C)$

```
n ← |I|
if I = ∅ or C = 0 then
    return 0
end
else
    if w_n > C then
        /* Remove the last item, it can't fit.                    */
        return Knapsack(I \ {n}, C)
    else
        return max{Knapsack(I \ {n}, C), Knapsack(I \ {n}, C − w_n) + v_n}
    end
end
return V[n, C]
```

(c) (2p.) Based on the recursive algorithm, design a $O(nC)$ time dynamic algorithm using $O(C)$ memory. (*Hint: Because the weights are assumed to be integers, we can iterate through all the subproblems* (I_k, w), $k = 0, \ldots, n$ *and* $w = 0, \ldots, C$.)