Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Markus Püschel, David Steurer
Gleb Novikov, Tommaso d'Orsi, Ulysse Schaller, Rajai Nasser

08. November 2021

Algorithms & Data Structures — Exercise sheet 7 — HS 21

Exercise Class (Room & TA): ____________________

Submitted by: ____________________

Peer Feedback by: ____________________

Points: _____

Submission: On Monday, 15 November 2021, hand in your solution to your TA *before* the exercise class starts. Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

Exercise 7.1 *Subset sum for general integers* **(1 point)**.

Let $a_1, \dots, a_n, t$ be $n+1$ integers in $\mathbb{Z}$. We would like to check whether there is a subset $I \subseteq \{1, \dots, n\}$ such that $\sum_{i \in I} a_i = t$. Here, we adopt the convention that if I is empty, then $\sum_{i \in I} a_i = 0$.

We have seen in class that if $a_1, \dots, a_n, t$ are positive, then we can solve this problem in $O(nt)$ time using dynamic programming. In this exercise, we would like to handle the case where some of the integers $a_1, \dots, a_n, t$ could be negative or zero.

Provide a *dynamic programming* algorithm that solves the subset sum problem for general integers. The algorithm should have $O\left(n \cdot \sum_{i=1}^{n} |a_i|\right)$ runtime.

Hint: *The DP table is two-dimensional, and its size is $(n+1) \times (1 + \sum_{i=1}^{n} |a_i|)$. Furthermore, for $i > 0$, the entry $DP[i][j]$ can be computed from $DP[i-1][j]$ and $DP[i-1][j - a_i]$.*

Address the following aspects in your solution:

1. *Definition of the DP table*: What is the meaning of each entry?
2. *Computation of an entry*: How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries that do not depend on others.
3. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
4. *Extracting the solution*: How can the final solution be extracted once the table has been filled?
5. *Running time*: What is the running time of your solution?

Exercise 7.2 *Longest Snake.*

You are given a game-board consisting of hexagonal fields $F_1, \dots, F_n$. The fields contain natural numbers $v_1, \dots, v_n \in \mathbb{N}$. Two fields are neighbors if they share a border. We call a sequence of fields $(F_{i_1}, \dots, F_{i_k})$ a *snake* of length k if, for $j \in \{1, \dots, k-1\}$, F_{i_j} and $F_{i_{j+1}}$ are neighbors and their values satisfy $v_{i_{j+1}} = v_{i_j} + 1$. Figure 1 illustrates an example game board in which we highlighted the longest snake.

For simplicity you can assume that F_i are represented by their indices. Also you may assume that you know the neighbors of each field. That is, to obtain the neighbors of a field F_i you may call $\mathcal{N}(F_i)$, which will return the set of the neighbors of F_i. Each call of $\mathcal{N}$ takes unit time.

a) Provide a *dynamic programming* algorithm that, given a game-board $F_1, \dots, F_n$, computes the length of the longest snake.

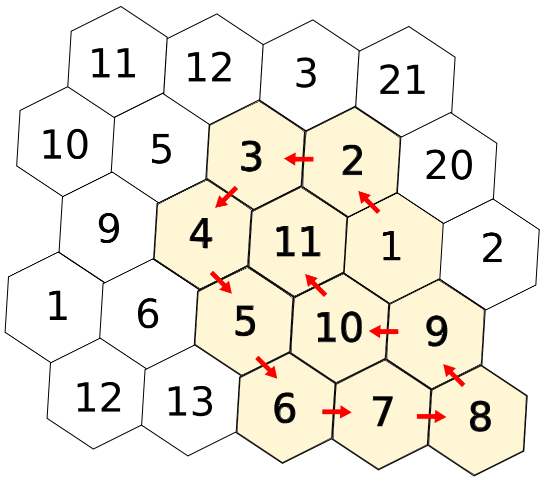

Figure 1: Example of a longest snake.

Hint: *Your algorithm should solve this problem using $\mathcal{O}(n \log n)$ time, where n is the number of hexagonal fields.*

Address the following aspects in your solution:

1. *Dimensions of the DP table*: What are the dimensions of the table $DP[\dots]$?
2. *Definition of the DP table*: What is the meaning of each entry?
3. *Computation of an entry*: How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
5. *Extracting the solution*: How can the final solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

b) Provide an algorithm that takes as input $F_1, \dots F_n$ and a DP table from part a) and outputs the longest snake. If there are more than one longest snake, your algorithm can output any of them. State the running time of your algorithm in Θ-notation in terms of n.

c)* Find a linear time algorithm that finds the longest snake. That is, provide an $\mathcal{O}(n)$ time algorithm that, given a game-board $F_1, \dots, F_n$, outputs the longest snake (if there are more than one longest snake, your algorithm can output any of them).

Exercise 7.3 *Road trip* **(1 point).**

You are planning a road trip for your summer holidays. You want to start from city C_0, and follow the only road that goes to city C_n from there. On this road from C_0 to C_n, there are $n-1$ other cities $C_1, \dots, C_{n-1}$ that you would be interested in visiting (all cities $C_1, \dots, C_{n-1}$ are right on the road from C_0 to C_n). For each $0 \le i \le n$, the city C_i is at kilometer k_i of the road for some given $0 = k_0 < k_1 < \dots < k_{n-1} < k_n$.

You want to decide in which cities among $C_1, \dots, C_{n-1}$ you will make an additional stop (you will stop in C_0 and C_n anyway). However, you do not want to drive more than d kilometers without making a stop in some city, for some given value $d > 0$ (we assume that $k_i < k_{i-1} + d$ for all $i \in [n]$ so that this is satisfiable), and you also don't want to travel backwards (so from some city C_i you can only go forward to cities C_j with $j > i$).

a) Provide a *dynamic programming* algorithm that computes the number of possible routes from C_0 to C_n that satisfies these conditions, i.e., the number of allowed subsets of stop-cities. In order to get full points, your algorithm should have $O(n^2)$ runtime.

   Address the following aspects in your solution:

   1. *Dimensions of the DP table*: What are the dimensions of the table $DP[\dots]$?
   2. *Definition of the DP table*: What is the meaning of each entry?
   3. *Computation of an entry*: How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries that do not depend on others.
   4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
   5. *Extracting the solution*: How can the final solution be extracted once the table has been filled?
   6. *Running time*: What is the running time of your solution?

b) If you know that $k_i > k_{i-1} + d/10$ for every $i \in [n]$, how can you turn the above algorithm into a linear time algorithm (i.e., an algorithm that has $O(n)$ runtime) ?

Exercise 7.4 *Animals in the zoo* **(1 point).**

A number n of animal species have been recently discovered in Africa. The zoo of Zürich is interested in acquiring as many animals from the new species as possible before a special exhibition that is taking place on December 1st, and you were put in charge of this task. Because of the time constraint, you can only organize one shipping of animals. The shipment can hold a maximum total weight of W. Furthermore, due to logistical constraints, you cannot isolate the animals during the shipment. Therefore, you cannot simultaneously bring two animals where one of them is a predator of the other.

Let $A_1, \dots, A_n$ be the $n > 4$ discovered species. You know that the species A_1, A_2 and A_3 are not predators, but for $4 \le i \le n$, the species A_i is a predator of only the species A_{i-1}, A_{i-2} and A_{i-3} (this means that, for example, A_i it is not a predator of species A_{i-4} or A_{i+1}).

For every $1 \leq i \leq n$, an animal from the species A_i has weight $w_i > 0$, and provides a value $v_i > 0$ to the zoo. You would like to figure out the collection of animals that you can bring to the zoo, and which provides the maximum total value to the zoo. We assume that $(w_i)_{1\leq i\leq n}$ and W are all positive integers. If you bring one animal from a species, then bringing another animal from the same species does not provide any additional value to the zoo. Therefore, there is no point in bringing two or more animals from the same species.

Provide a *dynamic programming* algorithm that solves this problem. The input to your algorithm are the weights $(w_i)_{1\leq i\leq n}$ and values $(v_i)_{1\leq i\leq n}$ of the animal species, and the maximum total weight W that is allowed in one shipping. In order to get full points, the runtime of your algorithm should be $O(nW)$.

Address the following aspects in your solution:

1. *Dimensions of the DP table*: What are the dimensions of the table $DP[\ldots]$?
2. *Definition of the DP table*: What is the meaning of each entry?
3. *Computation of an entry*: How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
5. *Extracting the solution*: How can the final solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Exercise 7.5 *Partitioning integers in three equal parts* ***(This exercise is from the January 2021 exam).***

You are given an array of n natural numbers $a_1, \ldots, a_n \in \mathbb{N}$ summing to $A := \sum_{i=1}^{n} a_i$, which is a multiple of 3. You want to determine whether it is possible to partition $\{1, \ldots, n\}$ into three disjoint subsets I, J, K such that the corresponding elements of the array yield the same sum, i.e.

$$\sum_{i\in I} a_i = \sum_{j\in J} a_j = \sum_{k\in K} a_k = \frac{A}{3}.$$

Note that I, J, K form a partition of $\{1, \ldots, n\}$ if and only if $I \cap J = I \cap K = J \cap K = \emptyset$ and $I \cup J \cup K = \{1, \ldots, n\}$.

For example, the answer for the input $[2, 4, 8, 1, 4, 5, 3]$ is *yes*, because there is the partition $\{3, 4\}$, $\{2, 6\}$, $\{1, 5, 7\}$ (corresponding to the subarrays $[8, 1]$, $[4, 5]$, $[2, 4, 3]$, which are all summing to 9). On the other hand, the answer for the input $[3, 2, 5, 2]$ is *no*.

Provide a *dynamic programming* algorithm that determines whether such a partition exists. Your algorithm should have an $\mathcal{O}(nA^2)$ runtime to get full points. Address the following aspects in your solution:

1) *Definition of the DP table*: What are the dimensions of the table $DP[\ldots]$? What is the meaning of each entry?
2) *Computation of an entry*: How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries that do not depend on others.
3) *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?

4) *Extracting the solution*: How can the final solution be extracted once the table has been filled?

5) *Running time*: What is the running time of your algorithm? Provide it in Θ-notation in terms of n and A, and justify your answer.