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
Johannes Lengler, Gleb Novikov, Chris Wendler

04. November 2019

Algorithms & Data Structures — Exercise sheet 7 — HS 19

Exercise Class (Room & TA): ______________________

Submitted by: ______________________

Peer Feedback by: ______________________

Points: _____

Submission: On Monday, 11 November 2019, 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 *Longest Ascending Subsequence.*

The longest ascending subsequence problem is concerned with finding a longest subsequence of a given array A of length n such that the subsequence is sorted in ascending order. The subsequence does not have to be contiguous and it may not be unique. For example if $A = [1, 5, 4, 2, 8]$, a longest ascending subsequence is $1, 5, 8$. Other solutions are $1, 4, 8$, and $1, 2, 8$.

Given is the array:

$$[19, 3, 7, 1, 4, 15, 18, 16, 14, 6, 5, 10, 12, 19, 13, 17, 20, 8, 14, 11]$$

Use the dynamic programming algorithm as described in class or the script to find the length of a longest ascending subsequence and the subsequence itself. Show all necessary tables and information you used to obtain the solution.

Exercise 7.2 *Longest Common Subsequence.*

Given are two arrays, A of length n, and B of length m, we want to find the their longest common subsequence and its length. The subsequence does not have to be contiguous. For example, if $A = [1, 8, 5, 2, 3, 4]$ and $B = [8, 2, 5, 1, 9, 3]$, a longest common subsequence is $8, 5, 3$ and its length is 3. Notice that $8, 2, 3$ is another longest common subsequence.

Given are the two arrays:

$$A = [7, 6, 3, 2, 8, 4, 5, 1]$$

and

$$B = [3, 9, 10, 8, 7, 1, 2, 6, 4, 5],$$

Use the dynamic programming algorithm as described in class or the script to find the length of a longest common subsequence and the subsequence itself. Show all necessary tables and information you used to obtain the solution.

Exercise 7.3 *Tinder Don*na Juan*a* **(1 Point)**.

You registered on Tinder and you got a lot of matches (you may assume that you have an endless amount of matches). Now, you would like to create a schedule for your dates. You don't date more than one person per day. Further, the day after having a date you always tell your best friend how it went and, thus, do not have time for a date on that day.

You tell your best friend about your success on Tinder and that you are trying to find a nice schedule for your dates. Your best friend challenges you to enumerate all possible date-schedules for the next T days. A schedule consists of T entries, where the i-th entry contains whether you have a date on this day or not.

Use dynamic programming to determine the number of different date-schedules under your constraints.

Hint: *In order to achieve full points your algorithm should solve this problem using* $\Theta(T)$ *time and memory.*

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. *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?

Specifically, you can use the following scheme:

Dimensions of the DP table:

Definition of the DP table:

Calculation of an entry:

Calculation order:

Reading the solution:

Running time:

Exercise 7.4 *Longest Snake* **(2 points).**

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 neighbours 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 neighbours 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 neighbours of each field. That is, to obtain the neighbours of a field F_i you may call $\mathcal{N}(F_i)$, which will return the neighbours $\mathcal{N}(F_i) = \{F_{j_1}, \dots, F_{j_6}\}$. 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.

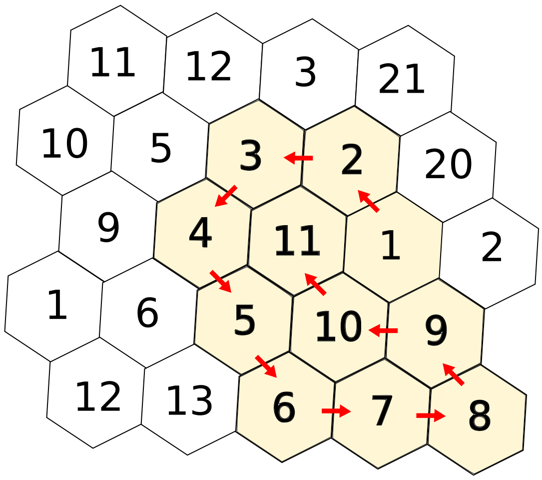

Figure 1: Example of a longest snake.

Hint: *In order to achieve full points 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:

(a) *Definition of the DP table: What are the dimensions of the table $DP[\dots]$? What is the meaning of each entry?*

(b) *Definition of the DP table*: What is the meaning of each entry?

(c) *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.

(d) *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?

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

(f) *Running time*: What is the running time of your solution?

Specifically, you can use the following scheme:

Dimensions of the table:

Meaning of a table entry (in words):

Computation of an entry (initialization and recursion):

Order of computation:

Computing the output:

Running time:

b) Provide an algorithm that takes as input $F_1, \ldots F_n$ 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 $F_1, \ldots, F_n$, outputs the longest snake (if there are more than one longest snake, your algorithm can output any of them).