Eidgenössische
Technische Hochschule
Zürich

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

Department Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

3rd November 2016

# Datenstrukturen & Algorithmen Exercise Sheet 7 AS 16

**Hand-in:** Thursday, 10th November 2016 before the start of the lecture at 10:00 in the entrance area of ML D28. Please staple all sheets together and use this sheet as the cover page. Fill out the first two fields of the form below.

Exercise class (Room & Day): _____

Submitted by: _____

Corrected by: _____

Bonus points: ____

**Hint:** This exercise sheet is concerned with *dynamic programming*. A complete description of a dynamic program **always** consists of the following aspects (interesting also for the exam!):

1) *Definition of the DP table*: What are the dimensions of the 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? Which entries 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?

The running time of a dynamic program is usually easy to calculate by multiplying the size of the table with the time required to compute each entry. Sometimes, however, the time to extract the solution dominates the time to compute the entries.

### Exercise 7.1 *Enumerating Palindromes.*

A *palindrome* is a word whose meaning may be interpreted the same way in either forward or reverse direction, e.g. the word RACECAR. Formally, a palindrome is a sequence $\langle a_1, \ldots, a_n \rangle$ where either $n = 1$, or $a_1 = a_n$ and $\langle a_2, \ldots, a_{n-1} \rangle$ is a palindrome (for $n = 2$ we only require $a_1 = a_2$). Let $A[1..n]$ be an array storing a string of length $n$. A subarray $A[i..j]$, $1 \leq i \leq j \leq n$, is called *palindrome in* $A$ if $\langle A[i], \ldots, A[j] \rangle$ is a palindrome.

*Example:* The array [L, A, R, A] contains the palindromes A, R, L and ARA (the palindrome A occurs twice). The array [A, N, N, A] contains the palindromes A, N, NN and ANNA (the palindromes A and N occur twice).

a) Let $A$ be an array containing a string of length $n$. Describe a dynamic programming algorithm that outputs all pairs $(i, j)$ where $\langle A[i], \ldots, A[j] \rangle$ is a palindrome. Provide also the running time of your solution.

*Example:* For the input $[\texttt{L}, \texttt{A}, \texttt{R}, \texttt{A}]$, the output consists of the pairs $(1, 1)$, $(2, 2)$, $(3, 3)$, $(4, 4)$, $(2, 4)$. For the input $[\texttt{A}, \texttt{N}, \texttt{N}, \texttt{A}]$, the output consists of the pairs $(1, 1)$, $(2, 2)$, $(3, 3)$, $(4, 4)$, $(2, 3)$, $(1, 4)$. In the output, no special order of the pairs is required.

*Hint*: Notice that the exercise can easily be solved without dynamic programming by trivial enumeration of all palindromes in time $\mathcal{O}(n^3)$. We search for a more efficient algorithm.

b) Suppose that the algorithm of a) computed the DP table already. Describe in detail, how a longest palindrome in $A$ can be extracted from the DP table. Provide also the required running time.

**Exercise 7.2** *Ascending Sequences.*

In this exercise, we consider a two-dimensional array $A$ with $n$ rows and $m$ columns. The element $A[i][j]$ is *adjacent* to the elements $A[i-1][j]$, $A[i][j-1]$, $A[i+1][j]$ and $A[i][j+1]$, if these elements exist (elements at the borders of the array are adjacent to correspondingly fewer elements).

A sequence $x_1, x_2, \ldots, x_k$ of elements in the array is called *ascending sequence* if it satisfies the following conditions:

- the elements in the sequence are sorted in ascending order, and
- for every $i \in \{1, \ldots, k-1\}$, the elements $x_i$ and $x_{i+1}$ are adjacent in the array.

We search for a longest ascending sequence in a given two-dimensional array. In the example below, a possible sequence would be $4, 6, 28, 29, 47, 49$. Design the most efficient algorithm that finds such longest ascending sequence using dynamic programming. Describe the algorithm, and specify its running time.

**Example array**:

| | | | | |
|---|---|---|---|---|
| 9 | 27 | 42 | 41 | 48 |
| 35 | 39 | 8 | 3 | 5 |
| 12 | 49 | 2 | 38 | 4 |
| 15 | 47 | 29 | 28 | 6 |
| 19 | 1 | 25 | 33 | 10 |