This exercise sheet is concerned with *dynamic programming*. If you have no experience, it can be very hard to apply dynamic programming directly. It may help to first design a recursive solution for understanding the problem, use memoization and then transform it finally into a dynamic program. An example of this transformation can be found in the example solution for programming part 1.

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  Mars mission.**

The rover *Curiosity* landed on Mars and is located at a starting position *S*. The goal is to move to a target position *Z*, and to collect rock samples that are as valuable as possible. To not use too much energy, the rover is only allowed to take a step to the east (right) and to the south (down). The value of each rock sample is stored in an \( m \times n \) matrix, e.g.

\[
\begin{array}{cccccc}
S & 9 & 2 & 5 & 11 & 8 \\
17 & 21 & 32 & 5 & 15 & 3 \\
2 & 2 & 3 & 8 & 1 & 5 \\
8 & 2 & 8 & 11 & 15 & 9 \\
0 & 5 & 3 & 10 & 4 & Z \\
\end{array}
\]

In the matrix above, an example of a south-east path from *S* to *Z* is shown where the value of the collected rock samples is maximum. This path can be described by enumerating the corresponding matrix positions: \((1, 1) \rightarrow (2, 1) \rightarrow (2, 2) \rightarrow (2, 3) \rightarrow (2, 4) \rightarrow \cdots \rightarrow (5, 6)\).
Provide a dynamic programming algorithm that takes an $m \times n$ matrix $A$ with $A[1, 1] = A[m, n] = 0$, and that computes a south-east path from $S = (1, 1)$ to $Z = (m, n)$ where the value of the collected rock samples is maximal. Note that we search for the path itself, and not just for the maximum value. Provide also the running time of your solution in dependency of $m$ and $n$.

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:**

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>27</th>
<th>42</th>
<th>41</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>39</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>49</td>
<td>2</td>
<td>38</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>47</td>
<td>29</td>
<td>28</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>25</td>
<td>33</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

**Hand-in:** until Wednesday, 9th April 2014.