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, Ulysse Schaller

26. October 2020

Algorithms & Data Structures — Exercise sheet 6 — HS 20

Exercise Class (Room & TA): _____

Submitted by: _____

Peer Feedback by: _____

Points: _____

Submission: On Monday, 2. November 2020, 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 6.1 *Minimum edit distance: finding an invariant* **(1 point)**.

Let $\Sigma = \{a, b, c, \ldots, z\}$ denote the alphabet. The minimum edit distance is equal to the minimal amount of insertions, deletions and substitutions required to change the string α to the string β. E.g., `editDistance`((t,i,g,e,r), (z,i,e,g,e)) = 3. The following algorithm takes two strings $\alpha = (\alpha_1, \ldots, \alpha_m) \in \Sigma^m$ and $\beta = (\beta_1, \ldots, \beta_n) \in \Sigma^n$, with $m, n > 1$, as input and computes the minimum edit distance.

Algorithm 1 `editDistance`(α, β)

```
d ← 0^{(m+1)×(n+1)} an (m + 1) × (n + 1) matrix of zeros
for i = 1, ..., m do
    d_{i,0} = i
for j = 1, ..., n do
    d_{0,j} = j
for i = 1, ..., m do
    for j = 1, ..., n do
        if α_i = β_j then
            c = 0
        else
            c = 1
        d_{i,j} = min(d_{i−1,j} + 1, d_{i,j−1} + 1, d_{i−1,j−1} + c)
        // Your invariant from a) must hold here.
return d_{m,n}
```

Note that for d we started our indexing from 0 and for α and β from 1.

a) Formulate an invariant $INV(i, j)$ that holds after the (i, j)-th iteration of the for loops, i.e., after the computation of $d_{i,j}$ in the pseudo code.

b) Prove the correctness of the algorithm `editDistance` by induction over i and j using your invariant.

Hint: You may ignore the exact order of computation and use the following induction hypothesis of the form: '$INV(i, j)$ holds for all i, j with $i + j \leq k$'. Then, you do an induction step from k to $k + 1$.

Exercise 6.2 *Introduction to dynamic programming* **(1 point)**.

Consider the recurrence

$$\begin{aligned} F_1 &= 1 \\ F_2 &= 1 \\ F_3 &= 1 \\ F_i &= F_{i-1} + F_{i-2} + F_{i-3}. \end{aligned}$$

a) Provide a recursive function (using pseudo code) that computes F_i for $i \in \mathbb{N}$.

b) Lower bound the running time of your recursion from a) using Ω notation.

c) Improve the running time of your algorithm using the memoization. Provide pseudo code of the improved algorithm and analyze its running time.

d) Compute F_i using dynamic programming and state the running time of your algorithm: Address the following aspects in your solution:

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

ii) *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.

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

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

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

Exercise 6.3 *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 from section 3.2. of the script to find the length of a longest ascending subsequence and the subsequence itself. Provide the intermediate steps, i.e., DP-table updates, of your computation.

Exercise 6.4 *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 from Section 3.3 of 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 6.5 *Dynamic programming marathon* **(1 point)**.

A	1	3	2	1
3	2	1	1	B

Figure 1: Runner problem for a cost array of size 2×5.

Imagine, a runner wants to run from A to B in Fig. 1. There are two lanes available. One is represented by the first row and the other by the second row. On some sections, the first lane is faster than the second lane, and vice versa. The runner can change lanes at any time, but this costs 1 minute every time. In this exercise, you are supposed to provide a dynamic programming algorithm that computes the optimal track.

Formally, the problem is defined in terms of a cost array $c \in \mathbb{N}^{2 \times n}$. In Fig. 1 $n = 5$. Now, the runner starts at position $(1, 1)$ and wants to run to $(2, n)$. Running along a lane from field (i, j) to the field $(i, j + 1)$ requires $c_{i,j+1}$ minutes. Changing lanes from field $(1, j)$ to $(2, j)$ requires $1 + c_{2,j}$ minutes, and from field $(2, j)$ to $(1, j)$ requires $1 + c_{1,j}$ minutes.

Provide an algorithm using dynamic programming that computes the optimal track from A to B. Your algorithm should compute the optimal sequence $(1, 1), (i_1, j_1), \ldots, (i_k, j_k), (2, n)$ and its cost (= the time required by the runner to run the sequence).

Address the following aspects in your solution:

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

ii) *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.

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

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

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