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

1. November 2021

Algorithms & Data Structures — Exercise sheet 6 — HS 21

Exercise Class (Room & TA): ____________________

Submitted by: ____________________

Peer Feedback by: ____________________

Points: _____

Submission: On Monday, 8. 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 6.1 *Introduction to dynamic programming* **(1 point)**.

Consider the recurrence

$$F_1 = 1$$
$$F_n = \left(\min_{1\le i<n} F_i^2 + F_{n-i}^2\right) \bmod 3n \quad \text{for } n \ge 2,$$

where $a \bmod b$ is the remainder of dividing a by b.

a) Consider the following algorithm that computes F top-down

Algorithm 1 Computing $F(n)$

```
function F(n)
    if $n = 1$ then
        return 1
    else
        $x \leftarrow F(1)^2 + F(n-1)^2$
        for $i = 2 \ldots \lfloor \frac{n}{2} \rfloor$ do
            $x \leftarrow \min(x, F(i)^2 + F(n-i)^2)$
        return $x \bmod 3n$
```

Lower bound the running time $T(n)$ of the above algorithm (i.e., give a simple function $g(n)$ such that $T(n) \ge \Omega(g(n))$) and show that it has an exponential running time.

Hint: *Prove by induction that* $T(n) \ge (3/2)^{n-1}$.

b) Improve the running time of the algorithm in (a) using memoization. Provide pseudo code of the improved algorithm.

c) Compute $F(n)$ bottom-up 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 $DP[\dots]$?* 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?

Remark. The "naive" algorithm from part (a) is called a top-down approach, while the DP algorithm from part (c) is called a bottom-up approach.

Exercise 6.2 *Longest common substring: finding an invariant* **(1 point)**.

Let $\Sigma = \{a, b, c, \dots, z\}$ denote the alphabet. Given two strings $\alpha = (\alpha_1, \dots, \alpha_m) \in \Sigma^m$ and $\beta = (\beta_1, \dots, \beta_n) \in \Sigma^n$, we are interested in the length of their longest common *substring*, which is the largest integer k such that there are indices i and j with $(\alpha_i, \alpha_{i+1}, \dots, \alpha_{i+k-1}) = (\beta_j, \beta_{j+1}, \dots, \beta_{j+k-1})$. Note that this problem is different from the longest common subsequence problem that you saw in the lecture. For example, the longest common substring of $\alpha = (a, a, b, c, b, a)$ and $\beta = (a, b, a, b, c, a)$ is (a, b, c), which is of length 3.

Below is the pseudo-code of an algorithm that computes the length of the longest common substring of two strings $\alpha \in \Sigma^m$ and $\beta \in \Sigma^n$ using $\Theta(mn)$ elementary operations:

Algorithm 2 `LongestCommonSubstring`(α, β)

```
$L \leftarrow \mathbf{0}^{m \times n}$ an $m \times n$ matrix of zeros.
for $i = 1, \dots, m$ do
    for $j = 1, \dots, n$ do
        if $\alpha_i = \beta_j$ then
            if $i = 1$ or $j = 1$ then
                $L_{i,j} = 1$
            else
                $L_{i,j} = L_{i-1,j-1} + 1$
        else
            $L_{i,j} = 0$
        // Your invariant from a) must hold here.
return $\max\{L_{i,j} : 1 \leq i \leq m, 1 \leq j \leq n\}$
```

a) Execute the algorithm on the strings $\alpha = (a, b, c)$ and $\beta = (c, a, b)$. Write down the value of the matrix L after each pass of the inner for-loop.

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

Hint: Consider the subproblem of finding the longest common substring that ends at some given indices of α and β.

c) Prove by induction that $INV(i,j)$ holds for all $1 \leq i \leq m$ and $1 \leq j \leq n$. Deduce that the algorithm `LongestCommonSubstring` is correct.

Hint: *You can perform induction over the minimum index* $k := \min(i,j)$.

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 *Optimizing Starduck's profit* **(1 point)**.

The coffeeshop chain Starduck's is planning to open several cafés in Bahnhofstrasse Zürich. There are n possible locations $1,\ldots,n$ for their shops on Bahnhofstrasse, ordered by their distance to Zürich main station $m_1 < \ldots < m_n$. Opening a shop at location i would yield Starduck's a profit of $p_i > 0$. However, they are not allowed to open cafés that are too close to each other, namely any two cafés should have distance at least d from each other, for some given value $d > 0$.

a) Provide an algorithm using dynamic programming that computes the maximum total profit that Starduck's can make on Bahnhofstrasse. In order to get full points, your algorithm should have $O(n \log n)$ runtime.

Hint: *Consider the subproblem of finding the maximum total profit that Starduck's can make if only locations* $1,\ldots,i$ *are available*

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?

b)* You now would like to recover not only the maximum total profit, but the corresponding locations where shops should be opened in order to achieve this profit. How can you get this out of your DP table in time $O(n)$?

Remark. There might be multiple optimal opening strategies, and it is enough if you can recover just one of them from the DP table.