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
François Hublet, Goran Zuzic, Tommaso d'Orsi, Jingqiu Ding

31 October 2022

Algorithms & Data Structures — Exercise sheet 6 — HS 22

The solutions for this sheet are submitted at the beginning of the exercise class on 07 November 2022.

Exercises that are marked by * are "challenge exercises". They do not count towards bonus points.

You can use results from previous parts without solving those parts.

Exercise 6.1 *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 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 6.2 *Coin Conversion* **(1 point)**.

Suppose you live in a country where the transactions between people are carried out by exchanging coins denominated in dollars. The country uses coins with k different values, where the smallest coin has value of $b_1 = 1$ dollar, while other coins have values of $b_2, b_3, \ldots, b_k$ dollars. You received a bill for n dollars and want to pay it *exactly* using the smallest number of coins. Assuming you have an unlimited supply of each type of coin, define OPT to be the minimum number of coins you need to pay exactly n dollars. Your task is to calculate OPT. All values $n, k, b_1, \ldots, b_k$ are positive integers.

Example: $n = 17, k = 3$ and $b = [1, 9, 6]$, then $\mathrm{OPT} = 4$ because 17 can be obtained via 4 coins as $1 + 1 + 9 + 6$. No way to obtain 17 with three or less coins exists. (A previous version had a typo "$k = 4$" that was corrected to "$k = 3$".)

(a) Consider the pseudocode of the following algorithm that "tries" to compute OPT.

Algorithm 1

```
1: Input: integers $n, k$ and an array $b = [1 = b_1, b_2, b_3, \dots, b_k]$.
2:
3: $counter \leftarrow 0$
4: while $n > 0$ do
5:     Let $b[i]$ be the value of the largest coin $b[i]$ such that $b[i] \leq n$.
6:     $n \leftarrow n - b[i]$.
7:     $counter \leftarrow counter + 1$
8: Print(“min. number of required coins = “, $counter$)
```

Algorithm 1 does not always produce the correct output. Show an example where the above algorithms fails, i.e., when the output does not match OPT. Specify what are the values of n, k, b, what is OPT and what does Algorithm 1 report.

(b) Consider the pseudocode below. Provide an upper bound in O notation that bounds the time it takes a compute $f[n]$ (it should be given in terms of n and k). Give a short high-level explanation of your answer. For full points your upper bound should be tight (but you do not have to prove its tightness).

Algorithm 2

```
1: Input: integers $n, k$. Array $b = [1 = b_1, b_2, b_3, \dots, b_k]$.
2:
3: Let $f[1 \dots n]$ be an array of integers.
4: $f[0] \leftarrow 0$                                   ▷ Terminating condition.
5: for $N \leftarrow 1 \dots n$ do
6:     $f[N] \leftarrow \infty$          ▷ At first, we need $\infty$ coins. We try to improve upon that.
7:     for $i \leftarrow 1 \dots k$ do
8:         if $b[i] \leq N$ then
9:             $val \leftarrow 1 + f[N - b[i]]$      ▷ Use coin $b[i]$, it remains to optimally pay $N - b[i]$.
10:            $f[N] \leftarrow \min(f[N], val)$
11: Print($f[n]$)
```

(c) Let $\mathrm{OPT}(N)$ be the answer (min. number of coins needed) when $n = N$. Algorithm 2 (correctly) computes a function $f[N]$ that is equal to $\mathrm{OPT}(N)$. Formally prove why this is the case, i.e., why $f[N] = OPT(N)$.

Hint: *Use induction to prove the invariant $f[n] = OPT(n)$. Assume the claim holds for all values of $n \in \{1, 2, \dots, N-1\}$. Then show the same holds for $n = N$.*

(d) Rewrite Algorithm 2 to be recursive and use memoization. The running time and correctness should not be affected.

Exercise 6.3 *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

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

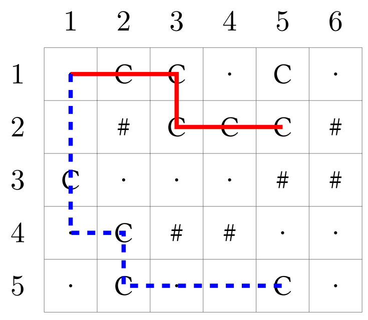
Use the dynamic programming algorithm from Section 3.3 of the script to find the length of a longest common subsequence and the subsequence itself. Show all necessary tables and information you used to obtain the solution.

Exercise 6.4 *Coin Collection* **(2 points)**.

Suppose you are playing a video game where your character's goal is to collect as many coins in a two-dimensional $m \times n$ grid world (m rows by n columns). The world is given to you as a table $A[1 \dots m][1 \dots n]$ where each cell is either a coin (denoted as "C"), impassible (denoted as "#"), or passable without coins (denoted as ".").

Your character starts at $(1, 1)$ (this cell will always be passable) and, in each turn, can move either right or down (up to your choice), or stop whenever (ending the game). Moving right corresponds to moving from $(x, y) \to (x, y + 1)$ and moving down is $(x, y) \to (x + 1, y)$. The goal is to determine the maximum number of coins the player can collect (by moving into a cell).

For example, on the $m \times n = 5 \times 6$ grid depicted right, the player can collect 5 coins by following the solid-red path. This is maximum possible and the answer is 5. A suboptimal path is depicted in dashed-blue, yielding 4 coins.



Remark: Be careful not to peek into an element of the table that is out-of-bounds (i.e., not within $[1, m] \times [1, n]$), as this can cause undefined behavior on a real computer.

(a) Write the pseudocode of a **recursive** function $f(x, y)$ which takes as argument a position of the character (x, y), and outputs the maximum number of coins that the character can collect if it started at (x, y) (ignoring all coins it might have previously collected). For example, in the grid above, $f(1, 1) = 5$, $f(2, 1) = 4$, $f(5, 5) = 1$, $f(5, 6) = 0$. The function does not need to be memoized for this subtask.

(b) Prove that your algorithm terminates in finite time (even if possibly exponential in the size of the input). Prove that the algorithm is correct.

Hint: *(This hint is assuming you implemented part (a) in the most natural recursive way.) To prove the algorithm completes in finite time, observe that $x + y$ only increases and is bounded, hence no infinite execution paths exist.*

Hint: *To prove the algorithm is correct, we simply need to prove the invariant which describes $f(x, y)$ (i.e., the first sentence of part (a)). Assume, by induction, the invariant holds for recursive calls $f(x, y)$ with strictly larger values of $x + y$, i.e., for those $f(x', y')$ such that $x' + y' > x + y$. Argue that then it also holds for $f(x, y)$ — we do this by considering the optimal path P^* that starts at (x, y) and* *consider three cases: if P^* ends immediately, if P^* initially goes to the right, or it goes down. Using the inductive hypothesis, argue that in each of those cases $f(x, y)$ becomes a value at least as large as the number of coins collected on P^*. Similarly, by considering the three cases, argue that the final value cannot be larger than that of P^* since otherwise we could find a better P^*. This, by induction, establishes that $f(x, y)$ is always equal to the number of coins on P^*.*

(c) Rewrite the pseudocode of the subtask (a), but apply memoization to the above f. Prove that calling $f(1, 1)$ will, in the worst-case, complete in $O(m \cdot n)$ time.

(d) Write the pseudocode for an algorithm that computes the solution in $O(m \cdot n)$ time, but does not use any recursion. Address the following aspects of your solution:

  (a) Definition of the DP table: What are the dimensions of the table DP? What is the meaning of each entry?

  (b) Computation of an entry: How can an entry be computed from the values of other entries?

  (c) 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?

  (g) Explicitly write out the pseudocode.