University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Schneider

Algorithms and Data Structures
Winter Term 2019/2020
Sample Solution Exercise Sheet 10

Remark: For this exercise, please watch the 14^{th} (final) video lecture.

Exercise 1: Topics for Final Lesson

In our final session (29th of January) we will have time to repeat some of the topics that you had difficulties with. For this purpose please send me an email (philipp.schneider@cs.uni-freiburg.de) with the topic "AD VOTE" and provide a list of three topics that you would like to repeat. From the topics most wished for I will compile the final, 11^{th} exercise sheet.

Exercise 2: Edit Distance

Let $A = a_1 \dots a_n, B = b_1 \dots b_m$ be two words. For $k \leq n, \ell \leq m$ let $A_k = a_1 \dots a_k, B_\ell = b_1 \dots b_\ell$ be the prefixes of A und B. Let $ED_{k,\ell} := ED(A_k, B_\ell)$ be the edit distance of A_k, B_ℓ. Use the dynamic programming algorithm from the lecture to compute $ED_{n,m}$ for the inputs $A = TORRENT$ und $B = RODENT$ by filling a table with values $ED_{k,\ell}$.

Sample Solution

We fill the following table according to the following recursion given in the lecture:

$$ED_{k,\ell} = \min\left(ED_{k,\ell-1} + 1, ED_{k-1,\ell} + 1, ED_{k-1,\ell-1} + \mathbb{1}_{a_k \neq b_\ell}\right)$$

$ED_{k,\ell}$	ε	T	O	R	R	E	N	T
ε	0	1	2	3	4	5	6	7
R	1	1	2	2	3	4	5	6
O	2	2	1	2	3	4	5	6
D	3	3	2	2	3	4	5	6
E	4	4	3	3	3	3	4	5
N	5	5	4	4	4	4	3	4
T	6	5	5	5	5	5	4	3

Exercise 3: Binomial Coefficient

Consider the following recursive definition of the binomial coefficient

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$$

with base cases $\binom{n}{0} = \binom{n}{n} = 1$. Give an algorithm that uses the principle of dynamic programming to compute $\binom{n}{k}$ in $\mathcal{O}(n \cdot k)$ time steps. Argue the running time of your algorithm

Sample Solution

Algorithm 1 BINOM(n, k) ▷ *global dictionary* `memo` *initialized with* `Null`

```
if k = 0 or k = n then return 1                          ▷ base cases
if memo[n, k] = Null then                                ▷ result not yet computed
    memo[n, k] ← Binom(n−1, k) + Binom(n−1, k−1)         ▷ compute partial results
return memo[n, k]
```

In the worst case, the routine $\textsc{Binom}(n, k)$ computes all partial results $\textsc{Binom}(m, l)$ for $m \leq n$ und $l \leq k$. However, each partial result is computed at most *once* before it is globally available in `memo`. There are at most $\mathcal{O}(n \cdot k)$ partial results, hence we call $\textsc{Binom}(\cdot, \cdot)$ at most $\mathcal{O}(n \cdot k)$ times when computing $\textsc{Binom}(n, k)$. Each call of $\textsc{Binom}(\cdot, \cdot)$ takes $\mathcal{O}(1)$ if we neglect the time required for sub-calls. Therefore the total time required is $\mathcal{O}(n \cdot k)$.

Exercise 4: Computing Minimum Change

Assume you are a vending machine and need to output an amount $N \in \mathbb{N}$ using coins with denominations $c_1, \ldots, c_n \in \mathbb{N}$ of which you have an unlimited supply. To make things simpler you do not actually have to compute the minimum cardinality set of coins that make up the amount N, but only the size of such a set (if it exists). Give an algorithm with runtime $\mathcal{O}(n \cdot N)$ that uses the principle of dynamic programming to compute the number of coins required to return an amount N, or ∞ if the amount can *not* be written as a weighted sum of $c_1, \ldots, c_n$. Argue the runtime of your algorithm.

Sample Solution

We compute the minimum number of coins $C(x)$ that sum up to x for all $x \in [N] := \{1, \ldots, \mathbb{N}\}$. If we know the solution for smaller amounts, then we can easily compute the amount of coins for N with the following recursion:

$$C(N) := \min_{i \in [n]} C(N - c_i) + 1.$$

As base cases we set $C(0) := 0$ and $C(x) = \infty$ for all $x < 0$.

Algorithm 2 CHANGE(N) ▷ *global dictionary* `memo` *initialized with* `Null`

```
if N = 0 then return 0                                   ▷ base cases
if N < 0 then return ∞
if memo[N] = Null then                                   ▷ result not yet computed
    memo[N] ← min_{i∈[n]} Change(N−c_i) + 1              ▷ compute partial results
return memo[N]
```

Each recursion takes at most $\mathcal{O}(n)$ time for computing the minimum over n values. Moreover we can have at most N recursions, since in each recursion we compute one result and after N recursions all results are computed and available in the dictionary and from then on (at the latest) we can look results up directly from the dictionary. Hence in total the algorithm takes $\mathcal{O}(n \cdot N)$ time.