11. Problem Set on

Algorithms, Data Structures and Data Abstraction WS 18/19

Wolfgang Mulzer

Due on 18. January 2019, 10:00, in the respective TA's mailbox

Problem 1 Greedy Change *10 Points*

Consider the problem of making change at a cash register. We are supposed to return an amount of n cents, while using the minimum number of coins. Recall that there are Euro-coins for 1, 2, 5, 10, 20, 50, 100, and 200 cents.

(a) Describe a greedy algorithm for the problem, and implement it. What is the running time?

(b) Prove that your algorithm from (a) always gives an optimal solution.

Hint: Think about the structure of an optimal solution. How many 1-cent coins can it have at most? How many 2-cent coins? etc? What is the maximum partial amount in an optimal solution that can be represented by coins of value at most c, for $c \in \{2, 5, 10, 20, 50, 100\}$?

(c) Give a family of coin denominations for which the greedy algorithm fails to give an optimal solution. There should be a 1-cent coin, so that we can make change for every possible amount.

Problem 2 Edit Distance *10 Points*

The *edit distance* between two strings s and t is the minimum number of *edit operations* needed to convert s into t. There are three edit operations: (i) insert a character; (ii) delete a character; and (iii) replace a character by another one. For example, the edit distance between "APFEL" and "PFERD" is 3: delete A, replace L by R, insert D.

Give an algorithm that determines the edit distance between two strings s and t in $O(k\ell)$ time, where s has length k and t has length ℓ. Moreover, explain how to find an optimal sequence of edit operations.

Hint: Use dynamic programming similar to the LCS-problem: consider the respective last character of s and t and distinguish three possibilities: (a) convert s to t' and add a character; (b) convert s' to t and delete a character; or (c) convert s' to t' and replace a character, if necessary. Here, s' and t' denote s and t without the last character.

Problem 3 String Search *10 Points*

(a) Implement the naive algorithm for string search as well as the algorithm of Rabin-Karp. Then, determine how often the word *whale* appears in the novel *Moby Dick* (ignoring case). How do the two implementations compare?

Hint: You can find *Moby Dick* under
`https://algs4.cs.princeton.edu/63suffix/mobydick.txt`.

(b) The algorithm of Rabin-Karp can be extended easily to *several* search patterns. Given a string s and search patterns $t_1, \ldots, t_k$, determine the first position in s, where one of the patterns $t_1, \ldots, t_k$ occurs. Describe how to adapt the Rabin-Karp Algorithm to this situation. What is the heuristic running time of the algorithm (assuming that collisions are rare)?

Hint: First, assume that all search patterns have the same length. The general case can be reduced to this case, e.g., by truncating all search patterns to the same length.

(c) (*voluntary, 5 extra points*) Implement the algorithm from (b). Then, determine who occurs most often in Moby Dick: Ishmael, Queequeg or Ahab?