JOHANNES KEPLER
UNIVERSITÄT LINZ

Algorithms and Data Structures 1
Summer term 2021
Dari Trendafilov,
Stefan Grünberger

Assignment 07

Deadline: **Thu. 24.06.2021, 23:59**
Submission via: **Moodle**

Elaboration time

*Remember the time you need for the elaboration of this assignment and document it in the file **time.txt** according to the structure illustrated in the right box. Please do not pack this file into an archive but upload it as a **separate file**.*

```
#Student ID
K12345678
#Assignment number
07
#Time in minutes
190
```

Strings & Pattern Matching

For both algorithms the **search method** should return a **list** with the starting indices of the positions where the pattern was found in the input text, or **None** if not found.

The **alphabet** of the input text and pattern is **letters** (upper- and lower case), **spaces** (' '), **periods** ('.') and **commas** (','), and the match must be **case sensitive**. None or empty strings in the pattern or text should trigger a ***ValueError*** exception. In case of partially **overlapping** matches, as in the example below (see index 11 and 12), all of them should be counted.
Example: For sequence „***abcdexxxunbxxxxke***" and pattern „**xxx**" the search should return `[5,11,12]`.

1. KMP algorithm — 9 points

Implement the **KMP** search algorithm as presented in the exercise and provide the **failure** table using the following **skeleton**:

```
class KMP:

    # param pattern - The pattern that is searched in the text.
    # param text - The text in which the pattern is searched.
    # return a list with the starting indices of pattern occurrences in the text, or None if not found.
    def search(self, pattern, text):

    # param pattern - The pattern for which the failure table shall be calculated.
    # return a list with the failure table values for the given pattern.
    def get_failure_table(self, pattern):
```

2. Rabin-Karp algorithm — 15 points

Implement the **Rabin-Karp** search algorithm as presented in the exercise. A hash value is calculated for the pattern (of length *m*), and for a partial sequence from the text (with the same length *m)*. If the two hash values are equal, the brute-force method is used to verify character by character if the pattern and the sequence are identical. Implement the **rolling-hash function** for computing the **hash values** for **base b=29** and **modulo** (if provided in the constructor)**,** using the following **skeleton**:

```
class RabinKarp:

        # initialise with the provided modulo value or None if omitted.
    def __init__(self, mod_val = None):

        # param pattern - The pattern that is searched in the text.
        # param text - The text in which the pattern is searched.
        # return a list with the starting indices of pattern occurrences in the text, or None if not found.
    def search(self, pattern, text):

        # param sequence - The sequence for which the (rolling) hash shall be computed.
        # param lastCharacter - The character to be removed from the hash when a new character is added.
        # param previousHash - The most recent hash value to be reused in the new hash value.
        # return hash value for the given character sequence using base 29.
    def get_rolling_hash_value(self, sequence, last_character, previous_hash):
```

Use **ASCII** coding for the letters as presented in the exercise (https://en.wikipedia.org/wiki/File:ASCII-Table-wide.svg).