Institute of
Pervasive Computing

Algorithms and Data Structures 1
*Summer term 2023*
Jäger, Beck, Anzengruber

# Assignment 7

Deadline CS: **Tue. 29.06.2023, 08:00**
Deadline AI: **Thu. 29.06.2023, 08:00**
Submission via: **Moodle**

### Elaboration time

*Remember the time you need for the elaboration of this assignment and document it in Moodle.*

# Strings & Pattern Matching

Please submit for example 1 a PDF of your solution and for example 2 your **rabin_karp.py** file.

## 1. KMP (Pen&Paper) 4+4 points

a) Create the failure function (P[j], f[j]) according to the Knuth-Morris-Pratt (KMP) algorithm for the pattern **"aabaabb"** and provide your result in the following table. In the blank space below the table insert your calculations step by step as presented in the exercise.

| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| P[0..j] | | | | | | | |
| f[j] | | | | | | | |

b) Apply the KMP algorithm with the pattern **"aabaabb"** (from example 1a) to the following text sequence **"baaabbaabaabaabb"**, and determine the number of required comparisons. For this, write down the characters of the pattern that you currently compare to the character in the text (one character per cell) and add in brackets "()" the number of comparisons (continuously increasing counter) as illustrated in the 2$^{nd}$ line of the table.

| b | a | a | a | b | b | a | a | b | a | a | b | a | a | b | b |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a(1) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |

Total number of comparisons: ____

# Assignment 7

## 2. Rabin-Karp algorithm 16 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):
```

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 „***AbCdExxx, Xxxxxke***“ and pattern „**xxx**“ the search should return `[5,11,12]`.

For the characters use the ASCII coding as presented in the exercise. For further information you can find an ASCII table here: *https://en.wikipedia.org/wiki/File:ASCII-Table-wide.svg*