Technische Universität München **Winter term 2019/20**
I7
Prof. J. Křetínský / M. Lazić / S. Sickert-Zehnter / C. Weil-Kennedy **21.11.2019**

Automata and Formal Languages — Programming Assignement Part 1

Due on 12.12.2019

Your task is to write a program in Java that searches for the shortest occurrence of a given pattern in a text, possibly with errors. This amounts to implementing the algorithms from the lecture notes and exercises and extending them.

Task

The final aim is to design an algorithm, with the following input and output:

Input: text $t \in \Sigma^+$, pattern p in the form of a regular expression, edit-distance $i \geqslant 0$
Output: the first shortest occurrence of a word w in t such that $w \in \Delta_{L(p),i}$, or $\perp$ if no such occurrence exists.

Recall that $\Delta_{L(p),i}$ is the language of all words with Levenstein-distance (or edit-distance) of at most i to some word in $L(p)$. See Exercise Sheet 6 for the formal definition. You can also find there a construction of an automaton accepting $\Delta_{L(p),i}$, starting from an automaton accepting $L(p)$. Note that this construction gives you an NFA-ϵ.

In order to achieve the whole task, here are four subtasks.

1. Implement the algorithm *PatternMatchingNFA* of the lecture notes, that is an algorithm with

   Input: pattern p in the form of a regular expression, text $t \in \Sigma^+$
   Output: the last position of the first occurrence of a word $w \in L(p)$ in t, or $\perp$ if no such occurrence exists

2. Modify this algorithm so that, for a text t, pattern p and distance $i \geqslant 0$, the algorithm returns the first occurrence *with edit-distance at most* i of pattern p.

   Input: pattern p, text $t \in \Sigma^+$, distance i
   Output: the last position of the first occurrence of a word $w \in \Delta_{L(p),i}$ in t, or $\perp$ if no such occurrence exists

3. Using the algorithm from the first subtask, implement an algorithm which searches for all the shortest occurrences of pattern p in text t, and returns the first one.

   Input: pattern p, text $t \in \Sigma^+$
   Output: the positions of the first and last letters of the first shortest occurence of a word $w \in L(p)$ in t, or $\perp$ if no such occurrence exists

4. Give an algorithm that searches for all the shortest occurrences *with edit-distance at most* i of pattern p in text t, and returns the first one.

   Input: pattern p, text $t \in \Sigma^+$, distance i
   Output: the positions of the first and last letters of the first shortest occurence of a word $w \in \Delta_{L(p),i}$ in t, or $\perp$ if no such occurrence exists

All four subtasks will be checked.

Example

Let $\Sigma = \{a, b, c\}$ and let p be the pattern given by the regular expression $ab(c)^*ba$. Given a text t, we number the letters from the left starting with 1, e.g. $t = a_1 a_2 \dots a_n$.

- Consider text $t_1 = abccba$. The first occurence of p in t_1 is $abccba$, and the last position of this first occurence is 6. There is no shorter occurrence, so this is at the same time the first shortest occurence of p in t_1. The first occurence of $\Delta_{L(p),1}$ is $abccb$, with last position 5 and there is no shorter occurrence.
- Consider text $t_2 = aacbcabccbaccbcabcabcc$. The first occurence of p in t_2 is $abccba$, and the last position of this first occurence is 11. There is no shorter occurrence. The first occurence of $\Delta_{L(p),1}$ is $abccb$, with last position 10. The first shortest occurence of $\Delta_{L(p),1}$ is the word $abca$ in position $16 - 19$. Notice that the first occurence is $6 - 10$ but it is not the shortest.
- Consider text $t_3 = ccccccccccc$. There is no occurence of words of $L(p)$ or $\Delta_{L(p),1}$ in t_3 so all the subtasks should answer $\perp$ for distance $i = 1$.

The expected outputs for these examples can be summarized in the following Table:

	$t_1 = abccba$	$t_2 = aacbcabccbaccbcabcabcc$	$t_3 = ccccccccccc$
Task 1	6	11	$\perp$
Task 2, $i = 1$	5	10	$\perp$
Task 3	1-6	6-11	$\perp$
Task 4, $i = 1$	1-5	16-19	$\perp$

Table 1: Table of outputs of the subtasks for the examples from above.

If the pattern contains the empty-word ε, then the answer for any text $t \in \Sigma^+$ is 0 for subtask 1 and 2 and $0 - 0$ for subtasks 3 and 4.

What to hand in

Download the zip file `afl-template.zip` provided on the website of the course. You should fill it out using Java, and the required JDK is JDK13 . The command to execute is:

```
./gradlew run --args='task_id example.regex lorem-ipsum.txt edit-distance'
```

Where `task_id / example.regex / lorem-ipsum.txt / edit-distance` need to be changed accordingly. The value of `task_id` can be 1,2,3 or 4 and corresponds to the numbering of the subtasks. You should print your solution to `stdout`. Print `not found` if the answer is $\perp$, `<number>-<number>` if the answer is the positions of first and last letter of an occurrence, and `<number>` if the answer is the position of the last letter of an occurrence. Note that there is a regular expression parser included in the template.

Send the modified folder `afl-template` in .zip format by email with subject header "afl project part 1" to chana.weilkennedy [at] in.tum.de. The deadline for this task is 12.12.2019.

Grading

Points will be awarded for correct solutions to examples. Point awards are subject to correctness of the result and the time performance (the output has to be correct and produced within a reasonable time).

Your source codes will be checked by the standard tools for plagiarism. This is an individual assignement.