Programming Assignment 3: The Implementation of Automaton-based Pattern Matching and the Karp-Rabin Algorithm

CSCD501 Advanced Algorithms

Instructor: Dr. Bojian Xu
Eastern Washington University, Spokane, Washington

Please follow these rules strictly:

1. Verbal discussions with classmates are encouraged, but each student must independently write his/her own work, without referring to anybody else's solution.
2. No one should give his/her code to anyone else.
3. The deadline is sharp. Late submissions will **NOT** be accepted (it is set on the Canvas system). Send in whatever you have by the deadline.
4. Every source code file must have the author's name on the top.
5. All source code must be written in Java and commented reasonably well.
6. You are not allowed to use library-provided methods if you are asked to implement them.
7. Sharing any content of this assignment and its keys in any way with anyone who is not in this class of this quarter is NOT permitted.

The goal of this programming assignment is to have a faithful implementation of both the automaton-based algorithm and the Karp-Rabin algorithm for pattern matching. I suggest you start the work early as the work may cost you more time than what you thought.

1 Source code organization

For an easier source code review and grading, all of your Java source code of both implementations needs be in a single Java source code file named as: `PatternMatching.java`. This single Java source code file will contain your implementation of both the automaton-based algorithm and the Karp-Rabin algorithm for pattern matching.

- Please do NOT submit other auxiliary files or folders from the IDE that you use.
- Please do NOT use "package" in your submitted Java source code. Feel free to use package when you do the coding on your side, but please remove the package from your submitted Java source code.
- Make sure your submitted Java file can compile and run correctly using command line.

2 Alphabet

To make your coding work a bit easier, this implementation will only be for texts, whose alphabet Σ **is the extended ASCII coded characters**[1]. That is, Σ is the collection of all characters that can be represented by one byte (eight bits) using the extended ASCII code, whose values $\in [0, 255]$, i.e., $|\Sigma| = \sigma = 256$.

3 Input and Output

There are two inputs to your program. Each input is a text file of extended ASCII coded characters. The first text file content represents the long text T of size n. The second text file content represents the short pattern P of size m. The name of each input file will be supplied to your program by the user as a command line parameter. You are guaranteed that $0 < m \leq n$.

Please note that, in the case where an input file contains multiple lines of characters, each "end-of-line" or "newline" character is treated as a regular character and needs participate the pattern matching. In other words, the entire sequence of bytes starting from the very first byte up to the end of the file represent the input string. Please also note that the "end-of-file" character is NOT part of the input strings.

Your program is to find all the occurrences of the pattern P in the text T by printing out the index of the starting location of each occurrence. The printings of the indexes of the starting locations of all occurrences need be separated by commas. We use 0-based indexing.

[1] `https://en.wikipedia.org/wiki/Extended_ASCII`

Example 1. Suppose the user-provided first input file named `text.txt` has the following content

```
mississippi
```

Suppose the user-provided second input file named `pattern.txt` has the following content

```
ssi
```

Then, the command line to run your program will be:

```
java PatternMatching text.txt pattern.txt
```

Your program shall be printing the following results on the screen:

```
The automaton-based pattern matching search result:2,5
The Karp-Rabin pattern matching search result:2,5
```

because $T[2..4] = P$ and $T[5..7] = P$. Please note, however, the Karp-Rabin search may produce some additional false reports. What false positive matchings will be reported? We don't know. It depends on the input T and P and the choice of the random prime number p that is used for modulo operation as part of the Karp-Rabin algorithm's design. However, this program assignment require you hard code $\mathbf{p} = \mathbf{524287}$, such that, given T and P, the false positive reports produced by the Karp-Rabin algorithms shall be consistent among all correct implementations.

Example 2. For the same text T, suppose the user-provided second input file named `pattern.txt` has the following content

```
issi
```

Your program shall be printing the following results on the screen:

```
The automaton-based pattern matching search result:1,4
The Karp-Rabin pattern matching search result:1,4
```

Again, the Karp-Rabin search may produce some additional false reports.

Example 3. For the same text T, suppose the user-provided second input file named `pattern.txt` has the following content

```
isi
```

Your program shall be printing the following results on the screen:

```
The automaton-based pattern matching search result:null
The Karp-Rabin pattern matching search result:null
```

unless the Karp-Rabin search may produce some false reports.

4 Advice and Requirements

4.1 On the Automaton-based Pattern Matching

The work contains two main components. One is to create the 2-d table δ that represents the automaton for the given pattern P. The other is to use the created automaton to do the linear scanning of the long text T for searching the pattern P.

Since in this assignment we only consider extended ASCII code characters, the dimension size of the 2-d table δ is $(m+1) \times \sigma$, where $m = |P|$ and $\sigma = |\Sigma| = 256$. Refer to the lecture slides for the detailed algorithm for creating the δ table. Once the 2d table δ is created, we are having an automaton of $m+1$ states ready for use for pattern search: We start from the initial state 0; at any state s, if the next character we read from T is character c, we will jump to state $\delta[s][c]$; We detect one occurrence of the pattern P anytime we land on the acceptance state m.

4.2 On the Karp-Rabin's Algorithm

In this implementation, we require the value of the random prime number p to be hard coded as $\mathbf{p = 524287}$. The reason for this hardcoding is for an easier grading and results checking: Given the same value for p, all false postive reports from all correct implementations will be consistent for any given T and P. Your program has four main components:

1. Implement the $\Theta(m)$-time computation of the following Equation 1 using the bottom-up approach via an iterative structure.

$$\begin{aligned} H_p(\mathcal{P}) &= \left(\sum_{i=0}^{m-1} \sigma^i \mathcal{P}[m-i-1] \right) \mod p \\ &= \left(\sigma \left(\sum_{i=1}^{m-1} \sigma^{i-1} \mathcal{P}[m-i-1] \mod p \right) + \mathcal{P}[m-1] \right) \mod p \\ &= \ldots \end{aligned} \tag{1}$$

2. Implement the $\Theta(m)$-time computation of the following Equation 2 using the bottom-to-top fashion via an iterative structure.

$$\begin{aligned} \sigma^m \mod p &= (\sigma \cdot (\sigma^{m-1} \mod p)) \mod p \\ &= (\sigma \cdot ((\sigma \cdot (\sigma^{m-2} \mod p)) \mod p)) \mod p \\ &= \ldots \end{aligned} \tag{2}$$

3. Given $H_p(T_{r-1})$ and the value of $\sigma^m \mod p$ from the previous two components, and the next character $T[r+m-1]$, calculate $H_p(T_r)$ in $\Theta(1)$ time using the following Equation 3.

$$\begin{aligned} H_p(T_r) &= (\sigma H_p(T_{r-1}) - \sigma^m T[r-1] + T[r+m-1]) \mod p \\ &= (\sigma H_p(T_{r-1}) - (\sigma^m \mod p) T[r-1] + T[r+m-1]) \mod p \end{aligned} \tag{3}$$

4. Once the above three components are built, you will need an overall control of the scanning of the entire text T. At every step of the scanning $r = 0, 1, \ldots, n-m$: Compute the value of $H_p(T_r)$; report a pattern detection if $H_p(T_r) = H_p(\mathcal{P})$.

4.3 Suggestions for Coding

- Your chance of success is nearly zero if you don't understand the algorithms, so do not code by only trying to translate the pseudocode. Make sure you understand the work first.
- Come up with a good design of your program. Design and specify the inputs/outputs of each function well, and ensure the design is logically neat and clear. Implement functions one after another. Test one function well with many strange edge cases before you move to implement the next one.

5 Submission

- You submit one single Java source code file named `PatternMatching.java`.
- Save your files into a folder, named: **YourLastname_EWUID_prog3**.
- Compress the folder into a zip file, named **YourLastname_EWUID_prog3.zip**.
- Submit the zip file onto the Canvas system by the deadline.