Assignment Two - KMP string search

Completion requirements
Due: Wednesday, 16 April 2025, 11:59 PM

Overview: You are to implement the Knuth-Morris-Pratt (KMP) string search algorithm for plain-text (i.e. searching for plain character strings inside plain text documents, like searching for the string "whale" in our MobyDick.txt) using the method described in lectures. Your solution must be written in Java and run on the Linux machines in Lab 1 of R Block. You may work in pairs or by yourself, with slightly different specifications depending upon which you decide to do. You may wish to seek out a partner using the General Discussion Forum on Moodle, or ask your tutor for help finding someone.

Usage and overall behaviour: Your final program should have the following invocation from a Linux shell command line.

java KMPsearch "target" filename.txt

It assumes your code *KMPsearch.class* and the plaintext file called *filename.txt* are in the current working directory/folder. **For single-person submissions**, the program is to search each line of the file, one line at a time, for the substring *target* and each line that has at least one occurrence of that substring is to be printed out just once, preceded by the index into that line indicating where the target first occurs (first character of the line is at index 1). Any line that does not contain the target should not be printed out. Output is to standard output.

For pair submissions, each line containing the target substring should be printed out once for each occurrence of the substring, preceded by the index into the line of that occurrence (i.e. you have to continue your search in the same string after any success). That is, if the target appears three times, then the line is printed out three times, each time with a unique index indicating where the target string starts, and in order of those occurrences.

If your program is invoked with just the target string, and no name of a file to search through, then your program must just printout the KMP skip table as a plain-text comma-separated-version of a skip array similar to that described in lecture, with one row for the target string, then one row for each unique character in the target string, and finally one more row for all characters not found in the target string. Each cell, indexed by the row and column, gives an integer specifying how far to move the search forward when looking for the target character at the position corresponding to this column and finding the character corresponding to the matching row.

The rows for characters found in the string should be in alphabetical order (according to the result of a java character comparison), but the first row (corresponding to the pattern) and last row (corresponding to all characters not in the pattern) are prefixed

with an asterisk in the first column just so that every row has the same number of non-empty fields, separated by commas.

Example:

An example of usage for the program with just a target substring is as follows:

```
% java KMPsearch "kokako"
```

which computes the skip array for "kokako" and prints it as the following five lines to standard output:

*,k,o,k,a,k,o

a,1,2,3,0,5,6

k,0,1,0,3,0,5

o,1,0,3,2,5,0

*,1,2,3,4,5,6

Once again, the asterisks at the start of the first and last rows are just to have a uniform input format for each line. The first row is the pattern, and the last row has the skip distances for characters not in the pattern anywhere.

Note that putting double-quote characters around the target string on the command-line is not always necessary, but ensures the entire contents of that string makes its way into the String args of the java static main method. They are stripped off by the shell, but using the quotes allows you to include in your target pattern some characters that may otherwise have special meaning to the shell ... such as a plus sign or a bracket. And you can include a double-quote by preceding it with a backslash. However, we only plan to test with simple characters so do not worry if you struggle with special ones. Whatever string gets into your java program, then that is the one to search for.

A sample file and target string and sample output for that file will be posted Moodle shortly.

Submission:

Same as for the previous assignment. Place copies of your nicely formatted and well-documented source code into an other wise empty directory whose name is your student ID (or pair of SIDS for a team), and you may include an optional plaintext README to communicate anything to the user or marker of your program. Make sure your full name and student ID number are included in the source code header, and cite any external sources of help adequately enough that it can be verified by the marker. This includes use of AI, where you should give quite

full details as to how it was used. Using AI is fine, but I do want to know how it is being utilised.

Submit this folder via Moodle as per usual. Only one member of a pair need submit the solution, but the other partner should check that the right thing went in ..... possibly submitting your own plaintext README identifying your partner and any circumstances you want the marker to know about.