FIT3155 S1/2020: Assignment 1
(Due midnight 11:59pm on Fri 24 April 2020)

[Weight: $10 = 4 + 3 + 3$ marks.]

Your assignment will be marked on the *performance/efficiency* of your program. You must write all the code yourself, and should not use any external library routines, except those that are considered standard. The usual input/output and other unavoidable routines are exempted.

Follow these procedures while submitting this assignment:

The assignment should be submitted online via moodle strictly as follows:

- All your scripts MUST contain your name and student ID.
- Use `gzip` or `Winzip` to bundle your work into an archive which uses your student ID as the file name. (STRICTLY AVOID UPLOADING `.rar` ARCHIVES!)
  - Your archive should extract to a directory which is your student ID.
  - This directory should contain a subdirectory for each of the three questions, named as: `q1/, q2/ and q3/`.
  - Your corresponding scripts and work should be tucked within those subdirectories.
- Submit your zipped file electronically via Moodle.

Academic integrity, plagiarism and collusion

Monash University is committed to upholding high standards of honesty and academic integrity. As a Monash student your responsibilities include developing the knowledge and skills to avoid plagiarism and collusion. Read carefully the material available at https://www.monash.edu/students/academic/policies/academic-integrity to understand your responsibilities. As per FIT policy, all submissions will be scanned via MOSS.

Assignment Questions

For the questions below, assume the alphabet is composed of printable ASCII characters.

1. **Mirrored Boyer-Moore algorithm**: In week 2, you learnt the Boyer-Moore's algorithm to find all exact occurrences of a pattern `pat`$[1 \ldots m]$ in any given text `txt`$[1 \ldots n]$. Recall that, in each iteration, the algorithm you learnt involved scanning aligned characters

from right to left, while the pattern was shifted from left to right under the text between iterations.

In this exercise, you will be implementing a *mirrored* version of Boyer-Moore that achieves the same task. In the mirrored algorithm, the pattern is to be shifted **leftwards** under the text between iterations, while scanning in each iteration proceeds from **left to right**. Specifically, in the first iteration, pat$[1 \dots m]$ and txt$[n - m + 1 \dots n]$ are aligned, and the scanning is done left-to-right: pat$[1]$ with txt$[n - m + 1]$, pat$[2]$ with txt$[n - m + 2]$ and so on.

Your mirrored implementation should employ all the corresponding *mirrored* rules you have learnt for the regular Boyer-Moore implementation. Additionally, your implementation should include the optimization that avoids all unnecessary character comparisons (i.e., those you know from the previous iteration are already identical).

Strictly follow the following specification to address this question:

Program name: `mirrored_boyermoore.py`

Arguments to your program: Two plain text files:

(a) an input file containing txt$[1...n]$ (without any line breaks).

(b) another input file containing pat$[1..m]$ (without any line breaks).

Command line usage of your script:

`mirrored_boyermoore.py <text file> <pattern file>`

Do not hard-code the file names/input in your program. The pattern and text should be specified as arguments. Penalties apply if you do.

Output file name: `output_mirrored_boyermoore.txt`

- Each position where pat matches the txt should appear in a separate line. For example, when text = abcdabcdabcd, and pattern = abc, the output should be:

```
1
5
9
```

2. **Pattern matching with wild card characters**: Let us extend the pattern matching problem to allow for wild card characters in the pattern. Let '?' represent a special wild card character that a pattern can employ to match any character in the text.

Let pat$[1 \dots m]$ represent a pattern containing ≥ 0 wild card ('?') characters. Let txt$[1 \dots n]$ denote some given text; assume txt does not have wild card characters in this exercise.

Write an efficient program that finds all occurrences of pat in txt, where pat allows for wild card characters.

Strictly follow the following specification to address this question:

Program name: `wildcard_matching.py`

Arguments to your program: Two plain text files:

(a) an input file containing txt$[1...n]$ (without any line breaks).

(b) another input file containing pat$[1..m]$ (without any line breaks, and potentially with ≥ 0 '?' wild card characters).

Command line usage of your script:

`wildcard_matching.py <text file> <pattern file>`

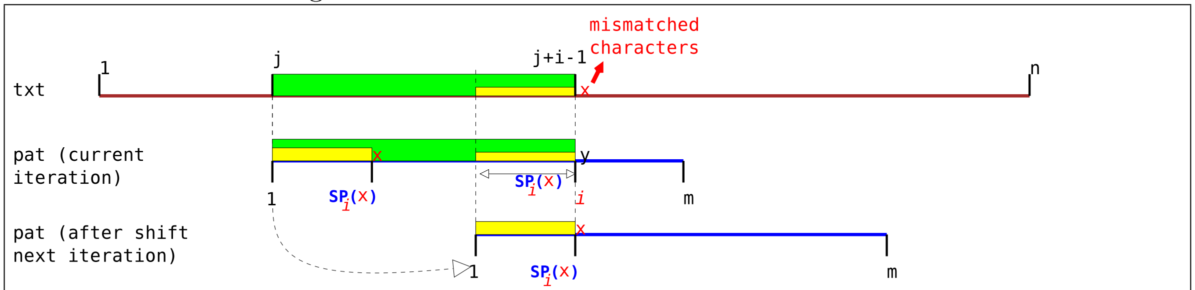
Do not hard-code the file names/input in your program. The pattern and text should be specified as arguments. Penalties apply if you do.

Output file name: `output_wildcard_matching.txt`

- Each position where `pat` matches the `txt` (after ignoring characters in text opposing wild card characters in the pattern) should appear in a separate line. For example, when `pat`$[1 \ldots 7]$ = `de??du?` and `txt`$[1 \ldots 20]$ = `ddedadudadededududum`, the output should be:

```
2
10
12
```

3. **Modified Knuth-Morris-Pratt**: This question deals with a minor modification of Knuth-Morris-Pratt (KMP) algorithm. Refer to the definition of SP_i on slide #43 of your lecture slides from week 2 material on Moodle. This task modifies SP_i to $\text{SP}_i(\texttt{x})$, which has the following definition:



Assume that some text `txt`$[1 \ldots n]$ and a pattern `pat`$[1 \ldots m]$ were derived from a fixed alphabet $\aleph$. In some iteration of KMP, let `pat` be compared with some position in `txt`:

- let the characters in the text, `txt`$[j \ldots j+i-1]$, match exactly with the corresponding characters in the prefix of the pattern, `pat`$[1 \ldots i]$, and
- let the next character in the text, `txt`$[j+i] \equiv \texttt{x}$, be a mismatch with the corresponding next character in the pattern, `pat`$[i+1] \equiv \texttt{y}$.

Then, $\text{SP}_i(\texttt{x})$ is defined as the length of the **longest proper suffix** of `pat`$[1 \ldots i]$ that matches the prefix of `pat`, with the extra condition that `pat`$[\text{SP}_i(\texttt{x})+1] = \texttt{x}$. That is, `pat`$[i - \text{SP}_i(\texttt{x})+1 \ldots i]$ = `pat`$[1 \ldots \text{SP}_i(\texttt{x})]$, and `pat`$[\text{SP}_i(\texttt{x})+1] \equiv \texttt{x}$ while `pat`$[i+1] \equiv \texttt{y}$.

Clearly, $\text{SP}_i(\texttt{x})$ yields a modified (and a more stringent) shift rule for the KMP algorithm than the one given in your lecture slides.

Your task is to implement a **space and time efficient** KMP implementation (to find all exact occurrences of `pat` in `txt`) using $\text{SP}_i(\texttt{x})$ values. The pre-computation of $\text{SP}_i(\texttt{x})$ values should be done using the Z-algorithm.

Strictly follow the following specification to address this question:

Program name: `modified_kmp.py`

Arguments to your program: Two plain text files:

(a) an input file containing `txt`$[1 \ldots n]$ (without any line breaks).

(b) another input file containing `pat`$[1..m]$ (without any line breaks).

Command line usage of your script:

`modified_kmp.py <text file> <pattern file>`

Do not hard-code the file names/input in your program. The pattern and text should be specified as arguments.

Output file name: `output_kmp.txt`

- Same as q1: each position where `pat` matches the `txt` should appear in a separate line. For example, when text = `abcdabcdabcd`, and pattern = `abc`, the output should be:

```
1
5
9
```

-=o0o=-
END
-=o0o=-