FIT3155 S2/2024: Assignment 1
(Due midnight 11:55pm on Sun 25 August 2024)

[Weight: $10 = 5 + 5$ 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. Standard data structures (e.g. list, dictionary, tuple, set etc.) that do not conflict with your assessment objectives are allowed, but ensure that their use is space/time efficient for the purpose you are using them. Also 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.
- Upload a .zip archive via Moodle with the filename of the form `<student_ID>.zip`.
  - Your archive should extract to a directory which is your Monash student ID.
  - This directory should contain a subdirectory for each of the two questions, named as: `q1/` and `q2/`.
  - Your corresponding scripts and work should be tucked within those subdirectories.

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 (click) to understand your responsibilities. As per FIT policy, all submissions will be scanned via MOSS (click) and/or JPlag (click).

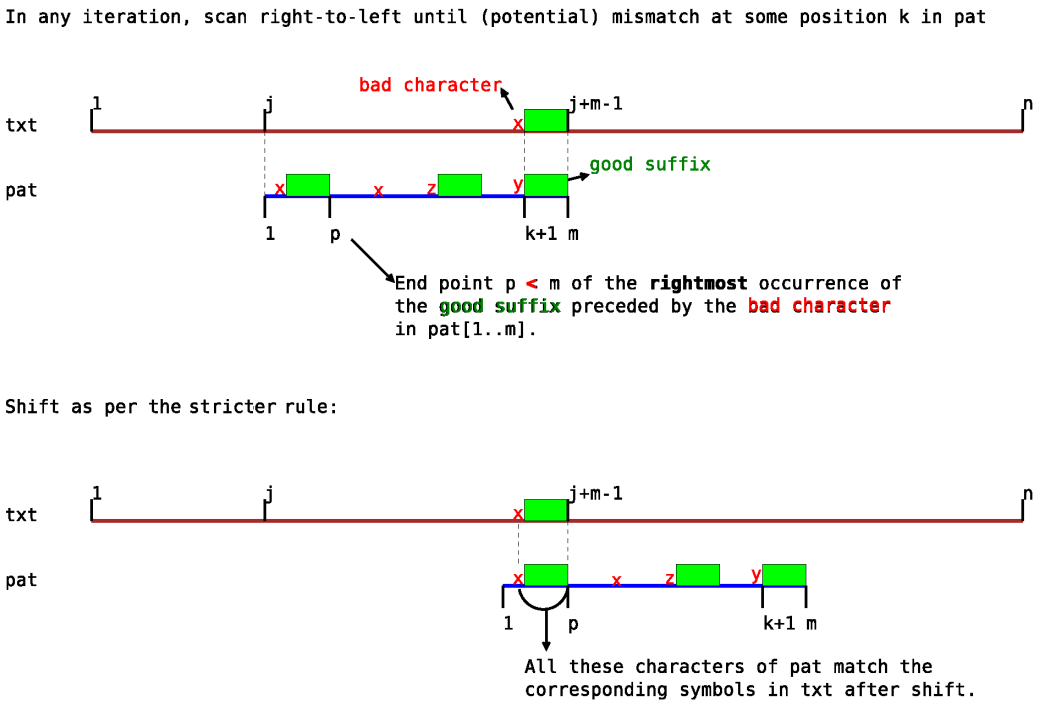
Generative AI not allowed!

This unit fully prohibits you from using generative AI to answer these assessed questions.

Note: For this assignment assume that the input text and pattern strings are always drawn from the a subset of printable ASCII characters: see [Link]. Specifically, these are printable characters whose ASCII values are in the range [37, 126] (both inclusive).

Assignment Questions

1. **Implementing Boyer-Moore with a stricter good suffix rule**: In this exercise you will be implementing the Boyer-Moore algorithm for exact pattern matching using a stricter version of the good suffix rule than the one introduced in your lecture in week 2. To understand the stricter good suffix rule, refer the following illustration.



Specifically, in any iteration of Boyer-Moore, after performing a right-to-left scan comparing characters in the pattern `pat`$[1 \ldots m]$ with their corresponding characters in the text `txt`$[1 \ldots n]$, if a mismatch is observed at some position k in `pat`, the stricter good suffix shift rule asks you to shift the pattern by $m - p$ positions, where $p < m$ is the **endpoint** in the pattern of the **rightmost** instance of the good suffix, **such that the bad character precedes that instance**. Formally, this implies

- `pat`$[p - m + k + 1 \ldots p] \equiv$ `pat`$[k + 1 \ldots .m]$, and
- the preceding character, `pat`$[p - m + k]$, **is identical** to the bad character in the text identified during the right-to-left scan.

Beyond this, all other specificiations of the Boyer-Moore algorithm discussed in Week 2 have to be considered and implemented where they ensure efficiency of your implementation.

Strictly follow the following specification to address this question:

Program/Function name: `q1.py`

Arguments to your program/function: Two filenames, containing:

(a) the string corresponding to `txt`$[1...n]$ (without any line breaks).

(b) the string corresponding to `pat`$[1..m]$ (without any line breaks).

Command line usage of your script:

```
python q1.py <text filename> <pattern filename>
```

Do not hard-code the filenames/input in your function. Ensure that we will be able to run your function from a terminal (command line) supplying any pair of text and pattern filenames as arguments. Give sufficient details in your inline comments for each logical block of your code. Uncommented code or code with vague/sparse/insufficient comments will automatically be flagged for a face-to-face interview with the CE before your understanding can be ascertained.

Output file name: `output_q1.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. **Matching a pattern containing unknown characters**: In this exercise you will attempt to implement pattern matching while allowing for unknown characters within the pattern. Here, we will use '`!`' to represent an unknown character. Importantly, during pattern matching an unknown character in the pattern is always considered as a match with any possible opposing character in the text.

   Specifically, let `pat`$[1 \ldots m]$ represent a pattern containing ≥ 0 unknown ('`!`') characters. Let `txt`$[1 \ldots n]$ denote some given text; it is safe to assume that `txt`$[1 \ldots n]$ does not contain any unknown characters for this exercise. Your goal is to write an *efficient* python program to detect all occurrences of `pat` in `txt`, while assuming that any unknown character in `pat`$[1 \ldots m]$ always matches any possible character in `txt`$[1 \ldots n]$.

   Strictly follow the following specification to address this question:

   Program/Function name: `q2.py`

   Arguments to your program/function: Two filenames, containing:

   (a) the string corresponding to `txt`$[1...n]$ (without any line breaks).

   (b) the string corresponding to `pat`$[1..m]$ (without any line breaks, and potentially containing zero or more '`!`' (i.e., unknown) characters).

   Command line usage of your script:

   ```
   python q2.py <text filename> <pattern filename>
   ```

   Do not hard-code the filenames/input in your function. Ensure that we will be able to run your function from a terminal (command line) supplying any pair of text and pattern filenames as arguments. Give sufficient details in your inline comments for each logical block of your code. Uncommented code or code with vague/sparse/insufficient comments will automatically be flagged for a face-to-face interview with the CE before your understanding can be ascertained.

   Output file name: `output_q2.txt`

   - Each position where `pat` matches the `txt` (while considering characters in text opposing unknown characters in the pattern as matches) should be reported 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

-=o0o=-
END
-=o0o=-