FIT3155 S1/2023: Assignment 1
(Due midnight 11:55pm on Sun 02 April 2023)

[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. 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).

Assignment Questions

1. **Z-algorithm pattern matching allowing one transposition error**: In this exercise, you are asked to write a program using the Z-algorithm to find all occurrences of any given pattern `pat`$[1 \ldots m]$ within any given text `txt`$[1 \ldots n]$, while allowing for at most one *transposition* error.

   What is a transposition error? A fairly common typographical error is one where two successive characters of a word/string are interchnaged in their order of appearance (just as you see the pair of successive characters highlighted above in red).

Consider the following set of text and pattern, respectively:

- txt$[1 \ldots 15]$ = `babbababaabbaba`
- pat$[1 \ldots 4]$ = `abba`

At positions 2 and 10 in the text (counting from 1), the pattern `abba` appears exactly. On the other hand, at positions 4, 5, 6, and 12 this pattern appears with one transposition error.

Note, if at any position of an exact occurrence, if one could also redundantly derive an occurrence with a transposition error – this situation arises when transposing successive pairs of characters that are identical (eg. transposing '`bb`' in `abba` during the exact occurrences at position 2 and 10) – you only have to consider/report the position for its exact occurrence and can safely ignore the other redundant possibilities under a transposition.

Your program should identify all non-redundant (see note above) occurrences of the pattern with ≤ 1 transposition error within the text.

Strictly follow the specification below to address this question:

Program name: `q1.py`

Arguments to your program: Two filenames:

(a) the filename of an input file containing the text, txt$[1 \ldots n]$ (in a single line).
(b) the filename of another input file containing the pattern, pat$[1 \ldots m]$ (in a single line).

You are free to assume that both text and pattern are all lowercase letters from the 26-letter English Alphabet

Command line usage of your script:

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

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

Output filename: `output_q1.txt`

- First line should specify the number of occurrences of the pattern in text allowing for at most 1 transposition error.
- Each subsequent line should identify the position of each occurrence of the pattern in the text and (separated by a single space) the starting position (again in the text) of any transposition, when this is not an exact occurrence. See example format below.

Output format (example): For txt$[1 \ldots 15]$ = `babbababaabbaba` and pat$[1 \ldots 4]$ = `abba`, the output file should contain the following (all positions below are 1-indexed):

```
6
2
4 4
5 7
6 6
10
12 12
```

Comments to marker document: A PDF document, `comments_q1.pdf`, that describes at a high-level the logic of the approach you have implemented in your script. Further, you have to specify the worst-case time-complexity and space-complexity of your implementation as a function of the pattern length (m) and text length (n).

2. **Boyer-Moore pattern matching with a wildcard**: In week 2, you learnt the Boyer-Moore algorithm to find all exact occurrences of a pattern `pat`$[1 \dots m]$ in any given text `txt`$[1 \dots n]$.

   In this exercise, we will allow the pattern `pat`$[1 \dots m]$ to contain at most 1 wildcard character. Here we will use dot ($\cdot$) to represent this wildcard. Besides potentially one wildcard ($\cdot$) in the pattern, all other characters in the pattern and all characters of the text are drawn from the 26-letter lowercase English alphabet.

   What purpose does the wildcard serve? The wildcard character dot ($\cdot$) in the pattern can match any character in the alphabet.

   Your goal is to implement a version of the Boyer-Moore algorithm with all its shift rules and optimizations that can identify all occurrences in a given text of a given pattern (which can contain at most 1 wildcard (dot) in it).

   Strictly follow the specification below to address this question:

   Program name: `q2.py`

   Arguments to your program: Two plain text filenames:

   (a) name of an input file containing `txt`$[1 \dots n]$ (in a single line).
   (b) name of another input file containing `pat`$[1 \dots m]$ (in a single line).

   Command line usage of your script:

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

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

   Output filename: `output_q2.txt`

   - Each position where `pat` (with at most 1 wildcard) matches the `txt` should appear in a separate line. For example, when text = `bbbbbababbbbbabb`, and pattern = `bb.bb`, the output should be (all positions below are 1-indexed):

     ```
     1
     9
     12
     ```

   Comments to marker document: A PDF document, `comments_q2.pdf`, that describes at a high-level the logic of the approach you have implemented in your script.

```
-=o0o=-
  END
-=o0o=-
```