Challenge: The Burrows–Wheeler Transform (BWT)

You may have heard of the Pattern Matching problem, which has many applications in Computer Science and other fields such as biology and natural language processing. There are several approaches to solve this problem. In this challenge, we will focus on the Burrows–Wheeler Transform (BWT), which was invented in 1994 by Michael Burrows and David Wheeler.

1 Research (60 pts)

During this challenge, you need to research the Burrows–Wheeler Transform (BWT) and its applications. Please answer the following questions:

(1) How can we transform a string s into its BWT form? Give a step-by-step example.

(2) If you have a BWT form L of a string, how can you reverse the transform to get the original string? Give a step-by-step example, using the example from the previous question.

(3) What are the best and worst case complexities (both time and space) for building the BWT matrix from a string s with length n? Explain your answer.

(4) We often use suffix arrays or suffix trees to solve the Pattern Matching problem. Is there a connection between the BWT and these structures? Briefly explain the connection, if there is one. (Hint: *How does the BWT relate to the order of suffixes when they are sorted alphabetically?*)

(5) A special feature of the BWT in the Pattern Searching problem is that, if you have a BWT form L of a string and a pattern p, you can find where p appears in the original string s of L without needing to know s directly. How can we do this? Give a step-by-step example. (Hint: *Backward search with LF-Mapping*).

(6) Besides using it for pattern searching, are there other uses for the BWT? Explain the ideas behind them. (Hint: *BWT tends to group similar characters together. Can we use this to do something useful?*)

2 Coding (40 pts)

You are asked to implement the Burrows–Wheeler Transform (BWT) and its application in Pattern Matching. Your program should be able to:

(1) Convert strings to the BWT form and back: given a string s (where the end-of-string (EOS) symbol is the character $; the string only contains lowercase Latin characters), convert it to the BWT form. Conversely, given the BWT form L of a string, convert it back to the original form.

(2) Conduct pattern matching on a BWT-form of a paragraph: Given a paragraph d (where the end-of-string (EOS) symbol is the character $; the paragraph only contains spaces and lowercase Latin characters) in **normal form** and a string s. Using the BWT, find **all** occurrences of s in d.

Please read the following specifications for details.

2.1 Program Requirements (40 pts)

You will create a command-line application that reads input from text files and writes output to text files. There are 2 main commands:

Command 1 (20 pts) Convert strings between normal and BWT forms:

```
<yourprogram.exe> -c input.txt output.txt [--bwt]
```

Where:

- `input.txt`: This text file contains strings that will be converted to the BWT form. Each string is on a separate line in the file.
- `output.txt`: This text file will contain the converted form of each string from `input.txt`, with each converted string on a separate line.
- `--bwt`: If this parameter is included, the program will convert strings from normal form to BWT form. If it is not included, the program will convert from BWT form to normal form.

Command 2 (20 pts) Search for patterns in a paragraph.

```
<yourprogram>.exe -p paragraph.txt patterns.txt output.txt
```

Where:

- `paragraph.txt`: This text file contains the paragraph, all on a single line.
- `patterns.txt`: This text file contains the patterns to search for, with each pattern on a separate line.
- `output.txt`: This text file will contain the results of the pattern search, in the following format:

  ```
  pattern1: 0, 3, ..
  pattern2: 4, 9, ..
  ..
  ```

  Here, `pattern1, pattern2, ..` are the patterns from `patterns.txt`. After each pattern, there is a list of the starting positions (indices) where the pattern appears in the `paragraph`. The indices start from 0, where 0 is the position of the first character.

2.2 Extended Functionality (Optional – 20pts)

Based on Question 6 from Section 1, you can implement a simple feature that demonstrates an application of BWT besides Pattern Search. You are free to propose your own command-line parameters, but they must meet these requirements:

- All inputs and outputs must be loaded from and saved to text files.
- Your final output must be clear and consistent (e.g., if you are planning to implement BWT for string compression, then your outputs must be understandable from the input).
- All command-line parameters must be clearly explained in your Report, or at least, in a README.txt file that guides users on how to use them.

3 Regulations

3.1 Group Registration

This challenge is **optional**. Only the **first 10 groups** that submit their work will receive a grade.

- You will work in groups of 4 students. We will not accept work from individuals, or from groups with less than or more than 4 members.
- Your `GroupID` will be created using the last 4 digits of each student's ID, sorted in ascending order.
  - **For example:** if your group consists of 4 members with student IDs *23120999*, *23120998*, *23120987*, and *23120888*, then your `GroupID` will be *0888-0987-0998-0999*.

3.2 Report

You will write a report to summarize your project. In general, you will answer the questions in Section 1 and document your program from Section 2.

- The report should be **structured, logical, clear**, and **coherent**. The total length of your report should not be more than 15 pages.
- You must clearly show your group's information (names and student IDs) on the first page of your report. Your work progress (which part of this project you did and how much work you completed) should also be on this page.
- You should show all algorithms or steps of algorithms as pseudocode, diagrams, or examples. Do **not** copy and paste your source code into the report.
- You must properly cite all work that is not your own.

3.3 Submissions

- Your submission must be organized in a directory with the following structure:

```
[GroupID]/
    - Report.pdf
    - video.txt
    - code/
```

  where

  - `Report.pdf`: the report file (mentioned in section 3.2), in `.pdf` format.
  - `video.txt`: the URL of your demonstration video, which shows all features of your program. You must upload your video to YouTube (in **unlisted/public** mode) and paste the link in this file.
  - `code/`: the directory containing the source code of your project.
    - Your code should be clear, logical, and well-documented.
    - **Do not** submit executable files (`.exe`) to prevent your project from being mistaken as a virus by antivirus software.

  * Please make sure that you have removed all unnecessary files (`.vscode, .vs, x64`, etc.) to avoid making your submission too large for Moodle.

- The folder `[GroupID]` will then be compressed into a `[GroupID].zip` archive, and submitted to Moodle within the allowed time. We only accept submissions through the Moodle platform.

The submission link on Moodle will close when either enough submissions are received, or the deadline is reached. Submissions that violate these rules will receive a grade of 0 (zero). Plagiarism and cheating are strictly prohibited and will result in a grade of 0 (zero) for **the entire** course.