FIT3155 S1/2020: Assignment 2
(Due midnight 11:59pm on Fri 22 May 2020)

[Weight: $10 = 4 + 2 + 2 + 2$ 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.
  - Place your suffix tree construction script under this directory.
  - Place the solution scripts to the 3 tasks, all of which will import your suffix tree construction script from above, into their respective subdirectories: `task1/`, `task2/`, and `task3/`.
- 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.

Assessable Tasks

It is a requirement of this assignment that all of the following three tasks should be addressed using a suffix tree of an input reference string. As always, marking will be based on the efficiency of your suffix tree construction, and those of the tasks below.

Mark distribution: 4 marks for suffix tree construction part. 2 marks × 3 tasks = 6 marks.

Task 1 For this task, you will revisit the pattern matching problem with wild card characters from your previous assignment.

Recall that for any given pattern `pat`$[1 \dots m]$ containing ≥ 0 wild card (‘`?`’) characters, the problem was to find all occurrences of `pat` in an input reference text, `txt`$[1 \dots n]$. As before the assumption is that the special wild card character ‘`?`’ in the pattern is considered a match with any character in the reference text. Also, the reference text does not contain any wild card characters.

Write an efficient program that constructs the suffix tree of any input `txt`, before searching on it for all occurrences of `pat` (potentially containing wild cards) within `txt`.

Strictly follow the following specification to address this task:

Program name: `wildcard_suffixtree_matching.py`

Arguments to your program: Two plain text files:

1. an input file containing `txt`$[1 \dots n]$. (For this assignment, this could be any standard ASCII text file that includes line breaks, white spaces etc.)
2. another input file containing `pat`$[1 \dots m]$ (potentially with ≥ 0 ‘`?`’ wild card characters).

Command line usage of your script:

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

As with the previous assignment, 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.

Example: If `pat`$[1 \dots 7]$ = `de??du?` and `txt`$[1 \dots 20]$ = `ddedadudadededududum`, the output should be:

```
2
10
12
```

Task 2 Given a string `str`$[1 \dots n]$, write a program that constructs its **suffix tree**, and using that suffix tree, outputs the Burrows-Wheeler Transform (BWT) of the string. Note: BWT was covered in FIT2004.
Strictly follow the following specification to address this task:

Program name: `suffixtree2bwt.py`

Argument to your program: An input file containing `str`$[1 \dots n]$.

- It is safe to assume that there are no line breaks in the input file, and that all characters `str`$[1 \dots n]$ are lexicographically larger than the terminal character, \$, which you will append to `str`$[1 \dots n]$ after reading it from the file.

Command line usage of your script:

```
suffixtree2bwt.py    <file containing str[1...n]>
```

Do not hard-code the filename in your program.

Output file name: `output_bwt.txt`

- Output format: The output file should contain (without line breaks) the BWT of `str`$[1 \dots n]$\$.

Example: If `str`$[1 \ldots n]\$ =$ `suffix_trees_and_bwt_are_related$`

the output would be:

```
dstdex__l_enrtrreuffeaat_e$wa_sbi
```

Task 3 For a string `str`$[1 \ldots n]$, define the values $L(i, j)$, $\forall 1 \leq i < j \leq n$, as the **longest prefix** that is common to the suffixes starting at indexes i and j in `str`.

For any input string `str`$[1 \ldots n]$ and an input list of (i, j) pairs, your task is to construct the suffix tree of `str`$[1 \ldots n]$, and using this tree output the $L(i, j)$ value for each (i, j) pair in that input list.

Strictly follow the following specification to address this task:

Program name: `lcps.py`

Arguments to your program: Two plain text files:

1. an input file containing `str`$[1...n]$ (without any line breaks).
2. another input file containing list of i and j pairs, one per line, where $1 \leq i < j \leq n$.

Command line usage of your script:

```
lcps.py    <string file>    <pairs file>
```

Do not hard-code the file names in your program.

Output file name: `output_lcps.txt`

- Each line will have three numbers (space separated) in the following format:
  $i \quad j \quad L(i, j)$.

Example: If the input file contained the string:

```
mississippi
```

and the input (i, j) pairs file contained,

```
8  11
2   5
1   5
4   7
```

then the output file will contain:

```
8  11   1
2   5   4
1   5   0
4   7   2
```

-=o0o=-
END
-=o0o=-