FIT3155 S2/2018: Assignment 1
(Due midnight 11:59pm on Sunday 16 September 2018)

[Weight: $20 = 5 + 5 + 5 + 5$ 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.
- 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 four questions, named as `q1/, q2/, q3/, and q4/`.
  - 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

1. Given some text `txt`$[1..n]$ and a pattern `pat`$[1..m]$, write a program to identify all positions within `txt`$[1..n]$ that matches the `pat`$[1..m]$ within a **Hamming distance** ≤ 1.

   The Hamming distance between any two strings of equal length is the number of positions in which their corresponding symbols are different.

Strictly follow the following specification to address this question:

Program name: `search_hammingdist.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:

```
search_hammingdist.py <textfile> <patternfile>
```

Output file name: `output_hammingdist.txt`

- Output format of each line of the output:

```
<position_in_txt>  <hamming_distance_with_pat>
```

- Example output for text = `bbcaefadcabcpqr`, and pattern = `abc`:

```
 1     1
 7     1
10     0
```

2. As a minor variation to the above question, given some text `txt`[1..n] and a pattern `pat`[1..m], write a program to identify all positions within `txt`[1..n] that matches the `pat`[1..m] within an **Edit distance** ≤ 1.

   The edit distance between two strings is the minimum number of edit operations (insertions, deletions, substitutions) required to transform one string into the other.

   Note: It is possible that a substring starting at a position in the text has many possible ways it can match the pattern under this edit distance threshold. For such positions, you are required to report any one of the possibilities.

   Strictly follow the following specification to address this question:

   Program name: `search_editdist.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:

```
search_editdist.py <textfile> <patternfile>
```

   Output file name: `output_editdist.txt`

   - Output format of each line of the output:

```
<position_in_txt>  <edit_distance_with_pat>
```

   - Example output for text = `abdyabxdcyabcdz`, and pattern = `abcd`

```
 1     1
 5     1
11     0
```

3. Given a string `str`[1...n]\$, write a program that constructs its **suffix tree**, and strictly using that suffix tree outputs the Burrows-Wheeler Transform (BWT) of that string. (Note: BWT was covered in FIT2004)

   Strictly follow the following specification to address this question:

   Program name: `suffixtree2bwt.py`

   Argument to your program: An input file containing `str`[1...n] (without line breaks).

   Command line usage of your script:

   ```
   suffixtree2bwt.py <stringfile>
   ```

   Output file name: `output_bwt.txt`

   - Output format: The output file should contain (without line breaks) the BWT of `str`[1...n]\$.
   - Example output for string = `suffix_trees_and_bwt_are_related$`
     `dstdex__l_enrtrreuffeaat_e$wa_sbi`

4. Write a program that implements Kruskal's Minimum-weight Spanning Tree (MST) algorithm, using the union-by-rank, and find-using-path-compression operations.

   Strictly follow the following specification to address this question:

   Program name: `kruskal.py`

   Argument to your program: An input file containing the weighted undirected graph information (space-separated three columns format, see below):

   - Input format: Each line of the input graph information file contains an edge and its corresponding weight in the following format:
     `<vertex_u>  <vertex_v>  <weight_of_edge_from_u_to_v>`
   - Example Input graph:

     ```
     1 3 3
     1 5 6
     2 4 9
     2 4 8
     4 5 7
     ```

   Command line usage of your script:

   ```
   kruskal.py <inputgraphfile>
   ```

   Output file name: `output_kruskal.txt`

   - Output format: Same format as the input (shown above). However, the output should be listed in the sorted (ascending) order of the edge weights of the MST.
   - Example output for the above example input graph:

     ```
     1 3 3
     1 5 6
     4 5 7
     2 4 8
     ```

-=o0o=-
END
-=o0o=-