FIT3155 S1/2024: Assignment 3
(Due midnight 11:55pm on Sunday 26 May 2024)

[Weight: $10 = 5 + 5$ marks.]

Your assignment will be marked on the performance/efficiency of your programs. You must write all the code yourself, and should not use any external library routines that interferes with the assessable items, except those that are considered general/standard.

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.
- Bundle and upload your work as a ⟨studentid⟩.tar.gz or ⟨studentid⟩.zip archive.
  - Your archive should extract to a single directory which is your 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.
- Submit your zipped-archive 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 (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 restricts you from availing/using generative AI to answer these assessed questions.

Question 1: A simple coding exercise with a result that should make you think [5 marks]

This question is a pretty straightforward implementational exercise that any undergrad computer science student will be able to complete. But understanding the pattern of behaviour of the result/output from this exercise is something that will expand your learning. Rest assured, the 'thinking' part is not being assessed; only the implementation and its correctness will be assessed. Yet important you think about the pattern that emerges from your implementation and attempt to link it with something you have learnt in this unit.

Your program takes two arguments as input:

1. the size of a (small) alphabet ($|\mathcal{A}|$).
   - For practicality during testing, restrict $|\mathcal{A}|$ within the range $[1, 5]$.
   - Assume the characters in the alphabet start from the letter '`a`' onwards. That is, if $|\mathcal{A}| = 4$, the alphabet $\mathcal{A}$ becomes: {'`a`','`b`','`c`','`d`'}.
2. the length of the string, N.
   - For practicality during testing, restrict N within the range $[1, 10]$.

Using the two input parameters $|\mathcal{A}|$ and N write a program that can

- generate all possible strings of length N from the alphabet $\mathcal{A}$, and
- for each string, find the **number of distinct cyclic rotations** that are possible for that string. (See two examples given on the next page to understand what this means.)

Note: For testing, although the inputs $|\mathcal{A}|$ and N are restricted within a small range, your implementation however must be generalizable to any values (≥ 1) of those parameters.

Strictly follow the following specification to address this question:

Program name: `q1.py`

Arguments to your program: Two integers:

(i) alphabet size ≥ 1 (during testing, enter a value in the range $[1, 5]$)

(ii) string length ≥ 1 (during testing, enter a value in the range $[1, 10]$).

Command line usage of your script:

```
python q1.py <alphabet size> <string length>
```

Output to the terminal the following statistics in a single line, space separated

(i) The number of strings of length N from alphabet $\mathcal{A}$ with ≥ 2 distinct cyclic rotations.

(ii) The number of strings of length N from alphabet $\mathcal{A}$ with exactly N distinct cyclic rotations.

(iii) The number of strings of length N from alphabet $\mathcal{A}$ with exactly 1 distinct cyclic rotation.

(iv) Is the number of strings with ≥ 2 distinct cyclic rotations an integer multiple of N? If yes, print `true`, else print `false`.

Sample outputs (with supporting debug information, which you do not have to report)

```
(SAMPLE 1) Input values:
    2 4
    Output:
    14 12 2 false

    ------------------------------------------------------------------------------------------
    NOTE: YOU DO NO HAVE TO OUTPUT THIS PART. DETAILS HERE ONLY TO AID YOUR UNDERSTANDING/DEBUGGING.

    Debug information for |A| = 2 and N = 4:
    cyclicrotations(aaaa) | cyclicrotations(abaa) | cyclicrotations(baaa) | cyclicrotations(bbaa)
        aaaa              |     abaa              |     baaa              |     bbaa
        aaaa              |     baaa              |     aaab              |     baab
        aaaa              |     aaab              |     aaba              |     aabb
        aaaa              |     aaba              |     abaa              |     abba
            nDistinct = 1 |         nDistinct = 4 |         nDistinct = 4 |         nDistinct = 4
    cyclicrotations(aaab) | cyclicrotations(abab) | cyclicrotations(baab) | cyclicrotations(bbab)
        aaab              |     abab              |     baab              |     bbab
        aaba              |     baba              |     aabb              |     babb
        abaa              |     abab              |     abba              |     abbb
        baaa              |     baba              |     bbaa              |     bbba
            nDistinct = 4 |         nDistinct = 2 |         nDistinct = 4 |         nDistinct = 4
    cyclicrotations(aaba) | cyclicrotations(abba) | cyclicrotations(baba) | cyclicrotations(bbba)
        aaba              |     abba              |     baba              |     bbba
        abaa              |     bbaa              |     abab              |     bbab
        baaa              |     baab              |     baba              |     babb
        aaab              |     aabb              |     abab              |     abbb
            nDistinct = 4 |         nDistinct = 4 |         nDistinct = 2 |         nDistinct = 4
    cyclicrotations(aabb) | cyclicrotations(abbb) | cyclicrotations(babb) | cyclicrotations(bbbb)
        aabb              |     abbb              |     babb              |     bbbb
        abba              |     bbba              |     abbb              |     bbbb
        bbaa              |     bbab              |     bbba              |     bbbb
        baab              |     babb              |     bbab              |     bbbb
            nDistinct = 4 |         nDistinct = 4 |         nDistinct = 4 |         nDistinct = 1
    ------------------------------------------------------------------------------------------

(SAMPLE 2) Input values:
    3 3
    Output:
    24 24 3 true

    ------------------------------------------------------------------------------------------
    Debug information for |A| = 3 and N = 3:
    cyclicrotations(aaa)  | cyclicrotations(acb)    | cyclicrotations(bbc) | cyclicrotations(cba)
        aaa               |     acb                 |   bbc                |     cba
        aaa               |     cba                 |   bcb                |     bac
        aaa               |     bac                 |   cbb                |     acb
            nDistinct = 1 |         nDistinct = 3   |       nDistinct = 3  |         nDistinct = 3
    cyclicrotations(aab)  | cyclicrotations(acc)    | cyclicrotations(bca) | cyclicrotations(cbb)
        aab               |     acc                 |   bca                |     cbb
        aba               |     cca                 |   cab                |     bbc
        baa               |     cac                 |   abc                |     bcb
            nDistinct = 3 |         nDistinct = 3   |       nDistinct = 3  |         nDistinct = 3
    cyclicrotations(aac)  | cyclicrotations(baa)    | cyclicrotations(bcb) | cyclicrotations(cbc)
        aac               |     baa                 |   bcb                |     cbc
        aca               |     aab                 |   cbb                |     bcc
        caa               |     aba                 |   bbc                |     ccb
            nDistinct = 3 |         nDistinct = 3   |       nDistinct = 3  |         nDistinct = 3
    cyclicrotations(aba)  | cyclicrotations(bab)    | cyclicrotations(bcc) | cyclicrotations(cca)
        aba               |     bab                 |   bcc                |     cca
        baa               |     abb                 |   ccb                |     cac
        aab               |     bba                 |   cbc                |     acc
            nDistinct = 3 |         nDistinct = 3   |       nDistinct = 3  |         nDistinct = 3
    cyclicrotations(abb)  | cyclicrotations(bac)    | cyclicrotations(caa) | cyclicrotations(ccb)
        abb               |     bac                 |   caa                |     ccb
        bba               |     acb                 |   aac                |     cbc
        bab               |     cba                 |   aca                |     bcc
            nDistinct = 3 |         nDistinct = 3   |       nDistinct = 3  |         nDistinct = 3
```

```
cyclicrotations(abc)  | cyclicrotations(bba)    | cyclicrotations(cab) | cyclicrotations(ccc)
    abc               |     bba                |   cab                |     ccc
    bca               |     bab                |   abc                |     ccc
    cab               |     abb                |   bca                |     ccc
        nDistinct = 3 |         nDistinct = 3  |       nDistinct = 3  |         nDistinct = 1
cyclicrotations(aca)  | cyclicrotations(bbb)    | cyclicrotations(cac) |
    aca               |     bbb                |   cac                |
    caa               |     bbb                |   acc                |
    aac               |     bbb                |   cca                |
        nDistinct = 3 |         nDistinct = 1  |       nDistinct = 3  |
-------------------------------------------------------------------------------------------
```

Non-examinable component: Systematically generate output values by varying the input parameters $|\mathcal{A}|$ and N in the ranges specified above, and see if you can pick up some pattern that emerges whenever the output line reports '`true`'. More importantly, does that relate to anything we learnt in this unit? Advanced target: Can you think of a formal proof for this pattern of behaviour?

Question 2: B-tree of distinct words [5 Marks]

In this question you will be writing a program that implements an **in-memory**[1] construction of a B-tree data structure, with supporting operations that you have learnt in your lecture (Week 9). Your program takes three arguments:

1. Minimum degree parameter $t \geq 2$ of the B-tree. (Only exemption to this minimum degree is the root node.)

2. An input text file (`dictionary.txt`) containing a list of distinct words in some random order. All words in this file are strings of ASCII characters, in one-word-per-line format (see example below).

3. Another input file (`commands.txt`) specifying a sequence of either `insert <someword>` or `delete <someword>` commands, which need to be applied on the constructed B-tree. (See example below.)

    - That is, starting from the B-tree constructed on words in the input `dictionary.txt`, these commands should be carried out on the current state of the B-tree, executed one-by-one in sequence they appear in `commands.txt`.
    - Note, if any command asks you to either `insert <someword>` that is already in a B-tree or `delete <someword>` *not* in the B-tree, your program should recognize these events and ignore them, by searching for that word in the current state of the B-tree (before deciding to execute any specified command.)

After executing the commands, your program has to traverse the final state of the B-tree and output the sorted list of words contained in the B-tree (one word per line; see example below).

[1] By '*in-memory*', it is meant that you are free to store the whole B-tree in main memory during its construction and operations. That is, you do not have to worry about storing and manipulating this B-tree from a secondary storage device. However, note that in practice B-tree is a data structure driven from a secondary storage.

Strictly follow the specification below to address this question:

Program name: `q2.py`

Arguments to your program: As enumerated above, the inputs are:

1. t (minimum B-tree node degree parameter)
2. dictionary.txt (containing list of distinct words (ASCII strings) in some random order, one per line; see example below).
3. commands.txt (sequence of insert/delete commands, one command per line; see example below).

Command line usage of your script:

```
python q2.py <t> <dictionary.txt> <commands.txt>
```

Output file name: `output_q2.txt` (containing sorted words derived by traversing the B-tree)

- Important: The sorted order of words is based on ASCII values of characters within those words/strings. Internally, when constructing the B-tree, you will have to maintain the B-tree's search order on the same criterion.

Examples

dictionary.txt

```
schmaltzy
replica
Ascension
cascades
abscissas
replicate
Capitalize
capital
dairying
hearties
Winch
Libation
summarize
slicks
Addend
ventilated
synchs
```

commands.txt

```
delete Ascension
insert Abstemious
delete syzygy
insert synchs
delete slicks
```

output_q2.txt

```
Abstemious
Addend
Capitalize
Libation
Winch
abscissas
capital
cascades
dairying
hearties
replica
replicate
schmaltzy
summarize
synchs
ventilated
```

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