FIT2004 S2/2023: Assignment 2

DEADLINE: Friday 20^{th} October 2023 16:30:00 AEDT.

LATE SUBMISSION PENALTY: 10% penalty per day. Submissions more than 7 calendar days late will receive 0. The number of days late is rounded up, e.g. 1 second late means 1 day late, 24 hours and 1 minute late means 2 days late. The deadlines are strict, last minute submissions are at your own risk. For special consideration, please visit the following page and fill out the appropriate form:

- `https://forms.monash.edu/special-consideration` for Clayton students.
- `https://sors.monash.edu.my/` for Malaysian students.

PROGRAMMING CRITERIA: It is required that you implement this exercise strictly using the **Python programming language** (version should not be earlier than 3.5). This practical work will be marked on the time complexity, space complexity and functionality of your program, and your documentation.

Your program will be tested using automated test scripts. It is therefore critically important that you name your files and functions as specified in this document. If you do not, it will make your submission difficult to mark, and you will be penalised.

SUBMISSION REQUIREMENT: You will submit a single python file, `assignment2.py`. Moodle will not accept submissions of other file types.

PLAGIARISM: The assignments will be checked for plagiarism using an advanced plagiarism detector. In previous semesters, many students were detected by the plagiarism detector and almost all got zero mark for the assignment (or even zero marks for the unit as penalty) and, as a result, the large majority of those students failed the unit. Helping others to solve the assignment is NOT ACCEPTED. Please do not share your solutions partially or completely to others. Using contents from the Internet, books etc without citing is plagiarism (if you use such content as part of your solution and properly cite it, it is not plagiarism; but you wouldn't be getting any marks that are possibly assigned for that part of the task as it is not your own work).

The use of generative AI and similar tools is not allowed in this unit!

Learning Outcomes

This assignment achieves the Learning Outcomes of:

- 1) Analyse general problem solving strategies and algorithmic paradigms, and apply them to solving new problems;
- 2) Prove correctness of programs, analyse their space and time complexities;
- 3) Compare and contrast various abstract data types and use them appropriately;
- 4) Develop and implement algorithms to solve computational problems.

In addition, you will develop the following employability skills:

- Text comprehension.
- Designing test cases.
- Ability to follow specifications precisely.

Assignment timeline

In order to be successful in this assessment, the following steps are provided as a **suggestion**. This is an approach which will be useful to you both in future units, and in industry.

Planning

1. Read the assignment specification as soon as possible and write out a list of questions you have about it.
2. Try to resolve these questions by viewing the FAQ on Ed, or by thinking through the problems over time.
3. As soon as possible, start thinking about the problems in the assignment.
    - It is strongly recommended that you **do not** write code until you have a solid feeling for how the problem works and how you will solve it.
4. Writing down small examples and solving them by hand is an excellent tool for coming to a better understanding of the problem.
    - As you are doing this, you will also get a feel for the kinds of edge cases your code will have to deal with.
5. Write down a high-level description of the algorithm you will use.
6. Determine the complexity of your algorithm idea, ensuring it meets the requirements.

Implementing

1. Think of test cases that you can use to check if your algorithm works.
    - Use the edge cases you found during the previous phase to inspire your test cases.
    - It is also a good idea to generate large random test cases.
    - Sharing test cases **is** allowed, as it is not helping solve the assignment.
2. Code up your algorithm (remember decomposition and comments), and test it on the tests you have thought of.
3. Try to break your code. Think of what kinds of inputs you could be presented with which your code might not be able to handle.
    - Large inputs
    - Small inputs
    - Inputs with strange properties
    - What if everything is the same?
    - What if everything is different?
    - etc...

Before submission

- Make sure that the input/output format of your code matches the specification.
- Make sure your filenames match the specification.
- Make sure your functions are named correctly and take the correct inputs.
- Remove print statements and test code from the file you are going to submit.

Documentation

For this assignment (and all assignments in this unit) you are required to document and comment your code appropriately. Whilst part of the marks of each question are for documentation, there is a baseline level of documentation you must have in order for your code to receive marks. In other words:

Insufficient documentation might result in you getting 0 for the entire question for which it is insufficient.

This documentation/commenting must consist of (but is not limited to):

- For each function, high-level description of that function. This should be a two or three sentence explanation of what this function does.
- Your main function in the assignment should contain a generalised description of the approach your solution uses to solve the assignment task.
- For each function, specify what the input to the function is, and what output the function produces or returns (if appropriate).
- For each function, the appropriate Big-O or Big-Θ time and space complexity of that function, in terms of the input size. Make sure you specify what the variables involved in your complexity refer to. Remember that the complexity of a function includes the complexity of any function calls it makes.
- Within functions, comments where appropriate. Generally speaking, you would comment complicated lines of code (which you should try to minimise) or a large block of code which performs a clear and distinct task (often blocks like this are good candidates to be their own functions!).

A suggested function documentation layout would be as follows:

```
def my_function(argv1, argv2):
    """
    Function description:

    Approach description (if main function):

    :Input:
        argv1:
        argv2:
    :Output, return or postcondition:
    :Time complexity:
    :Aux space complexity:
    """
    # Write your codes here.
```

There is a documentation guide available on Moodle in the Assignment section, which contains a demonstration of how to document code to the level required in the unit.

1 Customized Auto-complete (10 marks)

Alice used to love the auto-complete feature in her phone until the day she sent her thesis advisor an email saying "Thank you for being on my adversary panel". She soon received a reply "I hope you meant the advisory panel ;)". Needless to say, she was embarrassed and wrote a long email (this time disabling the auto-complete feature) explaining that the mistake was caused due to the auto-complete feature. She decided to do something about it and now wants to modify the auto-complete feature in her phone.

She has come to you for help and asks, "Can you please help me implement a customized auto-complete feature for my phone?"

"Well, I would love to help but I am quite busy with studies and ...", you murmur. Alice says, "Oh come on! I will provide you all the data, and all you have to do is to implement an algorithm". You desperately try to explain your situation, "Alice! I need to start revising all the previous lectures and applied classes for FIT2004 so that I am well prepared for the final exam, and I ... ".

Alice does not give up so easily and interrupts you, "Firstly, you should have started revising these much earlier, not in week 10. Secondly, implementing the algorithm will strengthen your understanding which will help you in the final exam. Finally, I know you are such a good friend!!! Pleaseeeeeeeee :)". And before you could say something, she starts explaining the customized auto-complete feature.

The standard auto-complete function in her phone displays up to three words that have the user's entered word as a prefix. E.g., if you are typing "adv", the auto-complete feature shows "adversary", "advisory" and "adventure". She wants to modify the auto-complete feature such that it only shows one word (instead of three) but also shows the definition of the word so that she can make sure that she is using the right word. She also wants to know how many words are there in the dictionary that have this prefix. E.g., if there are 43 words that have "adv" as a prefix, the auto-complete feature must display 43.

She has downloaded a dictionary from the internet that contains English words and their definitions. She has also downloaded the text of all the emails she has ever sent. She has processed the text and has assigned each word in the dictionary a frequency which corresponds to the number of times she ever used the word in her emails. She wants the auto-complete feature to display the word in the dictionary that has the highest frequency among the words that have her entered word as a prefix. For example, assume that the frequencies of "adversary", "advisory", and "adventure" are 100, 200 and 150, respectively. If she types "adv", the modified auto-complete system should display "advisory" with its definition and the number 43 that corresponds to the number of words that have "adv" as a prefix . She implemented a sorting based algorithm to do the prefix matching but it is too slow. You need to help her in efficient implementation of this feature.

1.1 Input

You are provided with a file `Dictionary.txt` that contains 3,000 words, their frequencies and their definitions (see Moodle). First few words in the dictionary along with their definitions and frequencies are shown below.

```
word: abaca
frequency: 554
definition: The Manila-hemp plant (Musa textilis); also, its fiber. See
Manila hemp under Manila.

word: abacinate
frequency: 1149
definition: To blind by a red-hot metal plate held before the eyes. [R.]

word: abacination
frequency: 342
definition: The act of abacinating. [R.]

word: abaciscus
frequency: 740
definition: One of the tiles or squares of a tessellated pavement;
an abaculus.

word: abacist
frequency: 336
definition: One who uses an abacus in casting accounts; a calculator.
```

Although `Dictionary.txt` provided on Moodle is sorted in alphabetical order of words to make it easier for you to manually verify the correctness of your implementation (and contains only the words starting with letter a), you cannot assume that the file on which your algorithm will be tested will also be sorted and will be similarly small. Your program may be tested on a different (and possibly a much larger) Dictionary. You can assume that each word in the dictionary consists of only lowercase English letters (e.g., they do not contain white spaces, hyphens or other symbols).
You have been provided a helper function called `load_dictionary(filename)` in the file `assignment2.py`. You must use this function to read the input file `Dictionary.txt`.

```
Dictionary = load_dictionary("Dictionary.txt")
```

The function `load_dictionary` returns a list of lists where the i-th element in the list corresponds to i-th word in `Dictionary.txt` represented as a list `[word, definition, frequency]`. The following shows the output of `print(Dictionary[0])` after you have loaded the dictionary as above.

```
['abaca', 'The Manila-hemp plant (Musa textilis); also, its fiber. See Manila
hemp under Manila.', 554]
```

1.2 Output

You must write a class called `Trie` as follows:

```
class Trie:
    def __init__(self, Dictionary):
        # ToDo: create a Trie based on the Dictionary which is a list
        of lists produced by load_dictionary function as described above.

    def prefix_search(self, prefix):
        # ToDo: this function must take a prefix as input and return a list
        [word, definition, num_matches] containing three elements where
        the first element "word" is the prefix matched word with the highest
        frequency, "definition" is its definition from the dictionary and
        num_matches is the number of words in the dictionary that have
        the input prefix as their prefix.
```

You will need to create a trie as follows.

```
if __name__ == "__main__":
    Dictionary = load_dictionary("Dictionary.txt")
    myTrie = Trie(Dictionary)
```

You will also need to implement the function `prefix_search`. This function must return a list containing three elements `[word, definition, num_matches]` containing the prefix matched word with the highest frequency, its definition and the number of words that have the input prefix as the prefix of the word. If there are multiple words with the highest frequency, you must display the alphabetically smallest word (e.g., if prefix is "be", and "best" and "beast" both have the same frequency - and highest among all words matching the prefix - your program must display "beast" because it is alphabetically smaller than "best"). If there is no word in the dictionary that matches the input prefix, you must return a list containing `[None,None,0]`. Below is a sample execution of the program for different inputs for the file `Dictionary.txt` provided on Moodle.

```
>>> myTrie.prefix_search("a")
['aberr', 'To wander; to stray. [Obs.] Sir T. Browne.', 3000]

>>> myTrie.prefix_search("an")
['andantino', 'Rather quicker than andante; between that allegretto.', 205]

>>> myTrie.prefix_search("ana")
['analeptic', 'Restorative; giving strength after disease. -- n.', 161]

>>> myTrie.prefix_search("anac")
['anaclastic', 'Produced by the refraction of light, as seen through water;
as, anaclastic curves.', 24]

>>> myTrie.prefix_search("anace")
[None, None, 0]
```

```
>>> myTrie.prefix_search("")
['aberr', 'To wander; to stray. [Obs.] Sir T. Browne.', 3000]
```

Note that for the prefix `"anac"`, there are two words in the dictionary `anaclastic` and `anacoenosis` with frequency `1462` and there is no other word in the dictionary that has `anac` as their prefix and has frequency higher than `1462`. The algorithm returns `anaclastic` because it is alphabetically smaller than `anacoenosis`.
The last prefix in the above sample output is an empty string in which case the most frequent word in the whole dictionary is returned.
Important: Your program will be tested on a variety of test cases using an automated script. Therefore, it is critical that your program follows the output format as shown above.
You only need to submit `assignment2.py`. Do not submit `Dictionary.txt` or any other file.

1.3 Complexity/implementation requirement

Your program will have two components. First, you will construct a Trie. Then, for each of call of the function `prefix_search`, you will need to return the relevant results. The worst-case time complexity to construct the Trie must be $O(T)$ where T is the total number of characters in `Dictionary.txt`. The worst-case space complexity of your algorithm must also be $O(T)$. The complexity requirements for the function `prefix_search` is $O(M + N)$ where M is the length of the prefix entered by the user and N is the total number of characters in the word with the highest frequency and its definition. Note that this is optimal because the size of the output string is $O(M + N)$ and no algorithm can achieve a better complexity. Note that N may be equal to M (i.e., the word is the same as the prefix and the definition is empty). Also, note that, in this assignment, we are assuming that each string comparison takes $O(L)$ where L is the number of characters in the strings.
Important: You must implement your own Trie and are not allowed to use publicly available implementations. Also, you are **NOT ALLOWED** to use Python dictionary (i.e., hash tables) or linked lists for implementing Trie. You must use arrays (called lists in Python) to implement the nodes of the Trie (as explained in the lecture slides).

2 A Weekend Getaway (10 marks)

With the end of the semester rapidly approaching, you and most of your friends realised that a weekend getaway after Week 12 would be a great idea so that you relax a bit before starting studying for your final exams. The ones that were not initially sure if taking 2 days off would be best idea eventually decided to go with the flow and also plan a weekend getaway.

You all got excited and came up with great ideas for destinations nearby, but soon realised that there are some constraints:

1. You want to save money and therefore you want to use the least amount of cars possible.
2. Not all of you have driver licences.
3. People have different preferences regarding destinations, and everyone should go to one of the locations they want.
4. You are all very tired after Week 12 and therefore agreed that no one should be the driver on both the outbound and inbound journeys.

After some discussions you come up with the following agreement:

- You realised that each car available can transport up to 5 people (including the driver). As you are n persons, you decided to use $\lceil n/5 \rceil$ cars in order to reduce the costs (and you have $\lceil n/5 \rceil$ cars available).
- You shortlisted $\lceil n/5 \rceil$ different destinations. For each of those destinations, one car will be going there with up to 5 persons on it, and the car will stay on that destination for the whole weekend (i.e., it is not possible to reuse the car to bring more people).
- Everyone will indicate the destinations they are interested in and if they have a driver licence or not.
- Everyone should be going to one of the destinations they are interested in (and only the $\lceil n/5 \rceil$ cars can be used as means of transport).
- In every car there should be at least two persons with driver licences traveling on it, so that the driver on the inbound journey is not the same as the driver in the outbound journey.

As the computer scientist in the group, you were tasked with writing a computer program to: either find a way to divide the persons into the cars/destinations so that all the above constraints are satisfied; or correctly indicate it is impossible to divide the persons into the cars/destinations while satisfying all constraints.

The persons will be numbered $0, 1, \ldots, n-1$, while the destinations/cars will be numbered $0, 1, \ldots, \lceil n/5 \rceil - 1$. Car number j will go to the destination number j. You will get as input a list of lists `preferences` and a list `licences` such that:

- For each $0 \leq i \leq n-1$, the elements of the list `preferences[i]` will form a subset of $\{0, 1, \ldots, \lceil n/5 \rceil - 1\}$ indicating the destinations in which person i is interested. You cannot assume that the elements in `preferences[i]` are listed in any specific order.

- The elements of the list `licences` will form a subset of $\{0, 1, \ldots, n-1\}$ indicating which persons have driver licences. You cannot assume that the elements in `licences` are listed in any specific order.

To solve this problem, you should write a function `allocate(preferences, licences)` that returns:

- `None` (i.e., Python NoneType), if it is impossible to allocate the persons into the cars/destinations while satisfying all constraints.
- Otherwise, it returns a list of lists `cars` in which, for $0 \leq j \leq \lceil n/5 \rceil - 1$, `cars[j]` is a list identifying the persons that will be traveling on car j to destination j. If there are multiple valid allocations satisfying all constraints, you can return any of those valid allocations (but should return exactly one of them).

2.1 Example

Consider the following example in which the function returns one valid allocation for the specified input.

```
# Example
preferences = [[0], [1], [0,1], [0, 1], [1, 0], [1], [1, 0], [0, 1], [1]]
licences = [1, 4, 0, 5, 8]

>>>  allocate(preferences, licences)
[[0, 4, 3, 2], [1, 5, 6, 7, 8]]
```

Note that there are other possible allocations for this specific input. It is fine to return any of the allocations that satisfy all constraints!

2.2 Complexity

Your solution should have a worst-case time complexity of $O(n^3)$.

Warning

For all assignments in this unit, you may **not** use python **dictionaries** or **sets**. This is because the complexity requirements for the assignment are all deterministic worst-case requirements, and dictionaries/sets are based on hash tables, for which it is difficult to determine the deterministic worst-case behaviour.

Please ensure that you carefully check the complexity of each in-built python function and data structure that you use, as many of them make the complexities of your algorithms worse. Common examples which cause students to lose marks are **list slicing**, inserting or deleting elements **in the middle or front of a list** (linear time), using the `in` keyword to **check for membership** of an iterable (linear time), or building a string using **repeated concatenation** of characters. Note that use of these functions/techniques is **not forbidden**, however you should exercise care when using them.

Please be reasonable with your submissions and follow the coding practices you've been taught in prior units (for example, modularising functions, type hinting, appropriate spacing). While not an otherwise stated requirement, extremely inefficient or convoluted code will result in mark deductions.

These are just a few examples, so be careful. **Remember that you are responsible for the complexity of every line of code you write!**