FIT2004 Semester 2 2024: Assignment 2

DEADLINE:
Clayton cohort: Wednesday 16^{th} October 2024 23:55:00 AEDT.
Malaysia cohort: Wednesday 16^{th} October 2024 23:55:00 MYT.

LATE SUBMISSION PENALTY: 5% penalty per day. Submissions more than 7 calendar days late will receive 0. The number of days late is rounded up, e.g. 5 seconds late means 1 day late, 27 hours late is 2 days late.

For special consideration, please visit the following page and fill out the appropriate form: `https://forms.monash.edu/special-consideration`.

The deadlines in this unit are strict, last minute submissions are at your own risk.

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 containing all of the questions you have answered, `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. Even after the deadline, your solutions/approaches should not be shared before the grades and feedback are released by the teaching team. 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 for the completion of your assignment is not allowed in this unit!

Learning Outcomes

This assignment achieves the Learning Outcomes of:

- Analyse general problem solving strategies and algorithmic paradigms, and apply them to solving new problems;
- Prove correctness of programs, analyse their space and time complexities;
- Compare and contrast various abstract data types and use them appropriately;
- 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:
   :Time complexity analysis:
    :Space and auxiliary space complexity:
   :Space and auxiliary space complexity analysis:
    """
    # Write your codes here.
```

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

1 A Fun Weekend Away (10 marks, including 2 marks for documentation)

Given all the hard work during the semester, you and your friends decided that you should better get a weekend off after Week 12 before starting preparing for your final exams. Your plan is to do some fun activity (e.g., surfing, hiking, kayaking, mountain biking, etc.) during this weekend getaway. However, during your initial talks you quickly recognised that:

- There are a total of n participants that want to get a weekend getaway (where n is an integer greater than 1). The participants will be denoted by $\mathtt{P}_0, \ldots, \mathtt{P}_{n-1}$.
- People have different preferences regarding which activity they would prefer to do.
- While some people would prefer to do one activity that they already have experience on, others would prefer to try something new.
- It would be wise to have some people with previous experience to lead each activity.
- Everybody is fine with helping leading one activity if they have previous experience on that activity. But no one wants to be a leader for the whole weekend as everyone also wants to have some chill time.

After some discussion you settled on the following criteria:

- You shortlisted some m activities denoted by $\mathtt{A}_0, \ldots, \mathtt{A}_{m-1}$ that the participants would be assigned to.
- Each person should be assigned to one activity they are interested in, and that person would be doing that activity for the whole weekend.
- For each activity $\mathtt{A}_j$, you need one leader for Saturday and another **different** leader for the Sunday. Both leaders need to have previous experience in activity $\mathtt{A}_j$. You already filtered the list of activities to guarantee that for each activity $\mathtt{A}_j$ there are at least two participants with previous experience on it. You can assume that the number of activities is at most $n/2$.

Some of your friends had a bit more free time, so they went ahead and did the heavy work of surveying the preferences of all participants and their previous experiences, and also of trying to figure out accommodation, transportation, equipment renting, etc for each activity. They have put some reservations on hold, but they got stuck with the problem of assigning the participants to the specific activities. They need urgent help to confirm if the current plan for each activity is good in the sense of having some assignment from the participants to the activities that would satisfy all criteria (and thus they can proceed with the reservations), and, if such assignment exists, how to obtain one.

As the computer science expert in the group, you were tasked with writing a computer program to solve that task. You are given as input the following data:

- A list of lists `preferences`. For each person $\mathtt{P}_i$ and each activity $\mathtt{A}_j$, `preferences[i][j]` would be:

  - equal to 0 if the person is not interested in that activity.
  - equal to 1 if the person is interested in that activity but has no previous experience on it.
  - equal to 2 if the person is interested in that activity and has previous experience on it.
  - You can assume that every person is interested in at least one activity.

- A list `places`. For each activity $\texttt{A}_j$, `places[j]` indicates how many people should be going to do the activity $\texttt{A}_j$ if the reservations that are on hold were to be confirmed. You can assume that `places[j]` ≥ 2 for each j. And it holds that $\sum_{j=0}^{m-1}$ `places[j]` $= n$.

Your program should either find a way to partition the participants into the activities so that all the above constraints are satisfied; or correctly indicate it is impossible to partition the participants into the activities while satisfying all constraints.

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

- `None` (i.e., Python NoneType), if it is impossible to assign the participants into the activities while satisfying all constraints.
- Otherwise, it returns a list of lists `activities` in which, for $0 \leq j \leq m-1$, `activities[j]` is a list identifying the indices of the participants that will be going to activity $\texttt{A}_j$. If there are multiple valid options to partition the participants into the activities while satisfying all constraints, you can return any of those valid options (but should return exactly one of them).

1.1 Example

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

```
# Example
preferences = [[2, 1], [2, 2], [1, 1], [2, 1], [0, 2]]
places = [2, 3]

>>>  assign(preferences, places)
[[0, 3], [1, 4, 2]]
```

1.2 Complexity

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

2 Customised Spell Checker (10 marks, including 2 marks for documentation)

Allen likes the spell checking feature in his phone but finds that it occasionally fails to align with his messaging style, particularly when he uses social media acronyms. For example, Allen may type "DM me" to mean "direct message me," but the spell checker might suggest "AM" or "dem" for correcting "DM".

In this task, we wish to create a customised spell checker that can make correction suggestions according to users' preference on word choice. The customisation is based on the user's previous messages sent using the phone.

To solve this problem, you are required to create a class `SpellChecker`. The constructor for this class takes a file `Messages.txt` as a parameter. Additionally, this class has a method `check(input)` to check an input `word` and suggest possible corrections.

2.1 Input

You are provided with a file `Messages.txt` where each line consists of a message. A few example messages in the file are shown below.

```
Oh, LOL.
I do not understand. ELI5.
I know how to do this, AMA.
```

Although `Messages.txt` is provided on Moodle to make it easier for verifying the correctness of your implementation, you cannot assume that the file used for testing your algorithm will be similarly small. Your program will be tested on a different and potentially much **larger** file.

Each line in `Messages.txt` should be converted into a list of words before counting their frequency. For this assignment, we assume that a word contains only alphabetic characters ("a" to "z" and "A" to "Z") and numerical digits ("0" to "9"). Other symbols, such as spaces (" "), commas (","), periods ("."), or question marks ("?"), should not be included in a word. Note that the spell checker is case-sensitive.

2.2 Output

The `check` method accepts a single word `input`. If `input` matches any word in your previous messages, `check` returns an empty list. Otherwise, `check` returns a list of at most three words that have a non-empty common prefix with `input`. If there are no words sharing a common prefix with `input`, an empty list should be returned. If there are more than three words sharing a common prefix with `input`, all words are ranked:

- firstly by the number of characters in the common prefix shared with `input` (longer prefixes take precedence),
- secondly by the frequency of the word in `Messages.txt` (higher frequencies take precedence),

- thirdly by the *ASCII value ordering* following the characters from the left to right of the words (smaller ASCII value ordering take precedence).

Note that a word a has a smaller *ASCII value ordering* than a word b in the following cases:

1. When two words are of the same length, say $a = a_1a_2 \ldots, a_k$ and $b = b_1b_2 \ldots b_k$, then a_i has a smaller ASCII value than b_i for the first place i where the two words differ.

2. Otherwise, two words have different lengths. The shorter word is padded with the null character (with ASCII value 0) at the end until two words have the same length, and then two words are compared as in the first case.

The `check` method returns the **TOP 3** words according to the above ranking. However, the words in the output list do not need to follow any specific order.

2.3 Example

Suppose you are given a file `Messages.txt` as follows:

```
Oh, LOL.
I do not understand. ELI5.
IDK. Tell me more.
LMK if you want to go.
If you will not go, me neither.
I know how to do this, AMA.
Fine, IDC.
BTW, I will not come back home for dinner tonight.
IDK. Tell me more.
```

After initialisation, your `SpellChecker` will have the following outputs:

```
>>> myChecker = SpellChecker("Messages.txt")
>>> myChecker.check("IDK")
[]
>>> myChecker.check("zoo")
[]
>>> myChecker.check("LOK")
["LOL", "LMK"]
>>> myChecker.check("IDP")
["IDK", "IDC", "I"]
>>> myChecker.check("Ifc")
["If", "I", "IDK"]
```

2.4 Complexity Requirements

- The `__init__` method of `SpellChecker` must run in time complexity $O(T)$, where T is the total number of characters in `Messages.txt`.

- Let M be the length of the string `input` and U be the total number of characters in the correct output that should be returned by **check** on that input, then **check** must run in time complexity $O(M + U)$. Note that this is an **output dependent** complexity. For example, if there are no words match the string `input`, **check** should complete in $O(M)$.

Important: You are **NOT ALLOWED** to use hash functions/tables (e.g. **Python dictionary**) or linked lists in your implementation. The list data structure of fixed size in Python[1] is allowed, [ ].

[1] https://docs.python.org/3/tutorial/datastructures.html

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!**