FIT2004 Data Structures and Algorithms
Assignment 2 - Semester 1 2025

DEADLINE:
Clayton cohort: 28th May 2025 23:55:00 AEST.
Malaysia cohort: 28th May 2025 23:55:00 MYT.

LATE SUBMISSION PENALTY: 5% penalty per day. Submissions more than 7 calendar days late will receive a score of 0. The lateness is measured in whole days, rounded up — for example, a submission that is 5 seconds late counts as 1 day late, and one that is 24 hours and 1 second late counts as 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.

ACADEMIC INTEGRITY: The assignments will be checked for plagiarism and collusion using advanced detector(s). In previous semesters, many students were detected 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! In fact they often hallucinate bad solutions.

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, 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, add 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 complexity:
   :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 code to the level required in the unit.

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!**

1 A Crowded Campus (9 marks)

The problem of class allocation is only becoming more and more difficult due to the current physical space constraints. There are many variables involved in the problem of allocating a unit's classes to specific classrooms and times, and allocating students to specific classes such as:

- the total number of students expected to enroll in a unit,
- which rooms have the necessary resources for the classes,
- how big the rooms are,
- the time availability of the students,
- and the time availability of the classrooms.

Given that physical space availability is currently the main bottleneck and that there are certain times of the day that are more preferred amongst students, the team responsible for managing the classroom spaces is considering placing stricter constraints on the usage of classroom space, based on the following general principles:

- During prime-time, a classroom would only be allocated to a specific unit if it is possible to allocate students to that class such that it reaches very close to the room capacity.
- During less popular times, the allocation of spaces is more flexible.
- For classroom spaces which are more popular with the units (as they have the best resources), the occupancy limits are stricter.

And, of course, the university also wants to make students as satisfied as possible by allocating as many students as they can to classes in their preferred times/days of the week.

The spaces admin team did a detailed analysis to set reasonable numbers for the minimum occupancy rate of specific classrooms during specific times of the day (based on the popularities of the classroom and the time slot). They have put a great effort in trying to come up with a draft allocation of classes to specific classroom spaces and times, but they have soon realised that verifying if it is possible to allocate the students accordingly to satisfy all the outlined constraints would be extremely hard to do manually. As they do not have a computer scientist in their team, they have asked for your help.

Particularly, they have asked you to help them verify the draft allocation of FIT2004 applied classes to specific classrooms and times. There are twenty time slots in which FIT2004 applied classes can run each week, as they are three hours long. These time slots will be numbered $0, 1, \ldots, 19$.

You are given as input the following data:

- A positive integer n denoting the number of students to be allocated to FIT2004 applied classes. The students will be numbered $0, 1, \ldots, n-1$.

- A positive integer m denoting the number of proposed FIT2004 applied classes in the draft. The proposed classes will be numbered $0, 1, \ldots, m-1$.
- A list of lists `timePreferences` of outer length n. For $i \in 0, 1, \ldots, n-1$, `timePreferences[i]` contains a permutation of the elements of set $\{0, 1, \ldots, 19\}$ to indicate the time slot preferences of student i. The time slots in `timePreferences[i]` appear in order of preference, with the time slot that student i likes the most appearing first.
- A list of lists `proposedClasses` of outer length m. For $j \in 0, 1, \ldots, m-1$:
  - `proposedClasses[j][0]` denotes the proposed time slot for the j-th class. Potentially, there can be multiple FIT2004 applied classes running in parallel.
  - `proposedClasses[j][1]` and `proposedClasses[j][2]` are positive integers that denote respectively, the minimum and maximum number of students that can be allocated to the j-th class to satisfy the space occupancy constraints.
- A positive integer `minimumSatisfaction`. It holds that `minimumSatisfaction` $\leq n$.

Your task is to write an algorithm that returns an allocation of each student to a proposed class. The returned allocation should satisfy the following requirements:

- Each student is allocated to exactly one class.
- Each proposed class satisfies its space occupancy constraints.
- At least `minimumSatisfaction` students get allocated to classes with class times that are within their top 5 preferred time slots.

To solve this problem, you should write a function `crowdedCampus(n, m, timePreferences, proposedClasses, minimumSatisfaction)`:

- In case no allocation satisfying all constraints exists, your algorithm should return `None` (i.e., Python NoneType).
- Otherwise, it should return a list `allocation` of length n containing exactly one possible allocation of the students to the classes that satisfies all constraints. For $i \in 0, 1, \ldots, n-1$, `allocation[i]` denotes the class number to which student i would be allocated.

Your algorithm should have worst-case time complexity $O(n^2)$ and worst-case auxiliary space complexity $O(n)$.

2 Typo (9 marks)

Levenshtein Distance or Edit Distance is a metric to measure the difference between two words – the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other [1].

You are an AI abuser and always ask an AI to do your assignment for you. However, the assignment forbids the use of AI and the AI is smart enough to ignore you no matter what you try to prompt it with. In fact, the AI is a troll and will provide wrong Python code and documentation with words that have Levenshtein distance exactly one from what it should be. Furthermore, it will only perform substitutions, and not insertions nor deletions.

You have learned that AI cannot be trusted and therefore are now coding a Python program to identify words that have Levenshtein distance exactly one from what they should be when only substitutions are considered. You are implementing a primary data structure that will help you efficiently compute this, as per the following signature:

```
class Bad_AI:
    def __init__(self, list_words):
        # ToDo: Create a data structure that stores list_words efficiently
        for the next task. Remember dictionary, including hashing, should
        not be used.

    def check_word(self, sus_word):
        # ToDo: This function identify words with Levenshtein distance
        value of exactly one (substitution).
```

2.1 Input

Based on the class signature given earlier, you have the following inputs:

- `list_words` is a list of N words, where the longest word has M characters and all of the characters in `list_words` add up to C. Thus, $O(C) \leq O(M * N)$.
- `sus_word` is a word with J characters.
- The characters in both lists are all lowercase, in the range of `a...z`. There are no special characters involved.
- All words in `list_words` are unique and you cannot assume they are in any particular order.

[1] From `https://en.wikipedia.org/wiki/Levenshtein_distance`

2.2 Output

The function `check_word(self, sus_word)` returns `result` - a list of the words from `list_words` whose Levenshtein distances to `sus_word` are equal to 1 when allowing only substitutions. If there are no such words, it would be an empty list `[]`.

Refer to the example(s) provided in Section 2.3.

2.3 Example

```
if __name__ == "__main__":
    list_words = ["aaa", "abc", "xyz", "aba", "aaaa"]
    list_sus = ["aaa", "axa", "ab", "xxx", "aaab"]
    my_ai = Bad_AI(list_words)
    for sus_word in list_sus:
        my_answer = my_ai.check_word(sus_word)
```

`my_answer` will contain the following for each iteration:

```
["aba"]
["aaa", "aba"]
[]
[]
["aaaa"]
```

2.4 Complexity

The class `Bad_AI` would have the following complexity:

- Function `__init__(self, list_words)` should have a worst-case time complexity of $O(C)$ with a worst-case auxiliary space complexity of $O(C)$.
- Function `check_word(self, sus_word)` should have a worst-case time complexity of $O(J * N) + O(X)$ with a worst-case auxiliary space complexity of $O(X)$. X is the total number of characters returned in the correct `result`.