FIT2004 S1/2023: Assignment 1

DEADLINE: Friday 28th April 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. 5 minutes 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` for Clayton students.
- `https://sors.monash.edu.my/` for Malaysian students.

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, `assignment1.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 eventual marks that are 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.
- Make sure you zip your files correctly (if required).

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 MEANS YOU GET 0 FOR THE ENTIRE ASSIGNMENT

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 Should I give a ride? (10 marks)

As a smart student you are always trying to optimise the driving time to your early morning algorithms lectures so that you can sleep more. You will be leveraging your algorithms skills to get an optimal solution.

Some of the roads in your city have carpool lanes that can only be used if there are at least 2 persons in the car, so you have to decide if you will be giving a ride to a fellow student or not. You have access to precise travel time information among key locations in your city, both with single car occupancy and with 2 or more persons in the car. And you have a list of locations in which there are students looking for a ride to the same destination you are going. You can assume that the potential passengers are always on time at the agreed location and there will be no additional time incurred for getting them into the car. You will either pickup passenger(s) with the same destination as you, or go alone the whole trip. Your absolute priority during those very early morning hours is maximising your sleeping time, so you are looking for the shortest total driving time and will not give a ride if that increases the total driving time.

You are a law-abiding citizen and would never drive in a carpool lane while you are alone in the car!

There are $|L|$ key location points represented by $0, 1, \ldots, |L|-1$. Your algorithm will take as input the following: a departure location `start` $\in \{0, 1, \ldots, |L|-1\}$, a destination location `end` $\in \{0, 1, \ldots, |L|-1\}$, a list `passengers` of locations where there are potential passengers, and a list of roads `roads` with the corresponding travel times. `passengers` is a list of integers such that each integer i in it is such that $i \in \{0, 1, \ldots, |L|-1\}$ and indicates that there are potential passengers at location i looking for a ride to your destination. The list of roads `roads` is represented as a list of tuples (a, b, c, d) where:

- $a \in \{0, 1, \ldots, |L|-1\}$ is the starting location of the road.
- $b \in \{0, 1, \ldots, |L|-1\}$ is the ending location of the road.
- c is a positive integer representing how many minutes you would spend to drive from a to b on that road if you are alone in the car.
- d is a positive integer representing how many minutes you would spend to drive from a to b on that road if you are there are 2 or more persons in the car.

Regarding those inputs:

- For any tuple in (a, b, c, d) in `roads`, you can assume that $d \leq c$ (as you can still use the non-carpool lanes when there are passengers in the car). For some tuple (a, b, c, d), it might be the case that $c = d$ (as some roads might not have carpool lanes, or the carpools lanes might not be improving the actual travel time on that road).
- You can assume that no roads will have only carpool lanes. I.e., if there is a road from a to b, it will always be possible to travel on it even if you are alone in the car. Therefore for any tuple (a, b, c, d) in `roads` the values c and d are well-defined.
- You can assume that for every location $\{0, 1, \ldots, |L|-1\}$ there is at least one road that begins or finishes there.
- You can assume that there is a route to go from `start` to `end` and that `start` $\neq$ `end`.
- The locations specified by `start` and `end` will not have potential passengers.
- You cannot assume that the list `passengers` is given to you in any specific order, but you can assume that there will be no repeated integers in `passengers`.

- You cannot assume that the list of tuples in `roads` are given to you in any specific order.
- You cannot assume that the roads are 2-way roads.
- The set of locations P specified in `passengers` can constitute any subset of $\{0, 1, \ldots, |L|-1\} \setminus \{\texttt{start}, \texttt{end}\}$, therefore you can neither assume $|P|$ is a constant nor assume that $P = \Theta(|L|)$.
- The number of roads $|R|$ might be significantly less than $|L|^2$, therefore you should not assume that $|R| = \Theta(|L|^2)$.

You should implement a function `optimalRoute(start, end, passengers, roads)` that returns one optimal route to go from `start` to `end` with the minimum possible total travel time. Your function should return the optimal route as a list of integers. If there are multiple route achieving the optimal time, you can return any one of them.

1.1 Complexity

Given an input with $|L|$ key locations and $|R|$ roads, your solution should have time complexity $O(|R| \log |L|)$ and auxiliary space complexity $O(|L| + |R|)$.

Note that the number $|P|$ of locations where there are potential passengers looking for a ride to your destination is not stated in the complexity. If your algorithm has time complexity $\Omega(|P||R| \log |L|)$, you will be losing a very significant amount of marks for this question.

1.2 Example

Consider the example below:

```
# Example
start = 0
end = 4
# The locations where there are potential passengers
passengers = [2, 1]
# The roads represented as a list of tuple
roads = [(0, 3, 5, 3), (3, 4, 35, 15), (3, 2, 2, 2), (4, 0, 15, 10),
(2, 4, 30, 25), (2, 0, 2, 2), (0, 1, 10, 10), (1, 4, 30, 20)]

# Your function should return the optimal route (which takes 27 minutes).
>>> optimalRoute(start, end, passengers, roads)
[0, 3, 2, 0, 3, 4]
```

2 Repurposing Underused Workspace (Dynamic Programming) (10 marks)

A large technology company owns a huge building and the director of the company wants to find the best area within the building to install the new high performance computing (HPC) facility which will arrive soon. Unfortunately, the company does not have any empty spaces to be utilised for this purpose. Instead, the director will have to use some of the its currently underused spaces. In fact, office areas have not been fully used since Covid-19.

Staff offices are placed in each level in a rectangular area with m aisles/columns. Each aisle has n rows where $n > m$. The HPC facility will require an area of size n connected sections. This means that only one section from each row has to be removed, so we end up with one empty aisle after some shuffling for the remaining sections within each row. Once we identify the location of those sections, the director and his team will decide what should be done with team members in these sections.

Occupancy data:

The director knows the occupancy probability for each section, which is an integer between 0 and 100 (inclusive) that represents the percent of working hours this section is usually occupied (on average).

Given a matrix of n rows and m aisles/columns $P[0...n-1][0...m-1]$, which contains the occupancy probability values for a total of $n \cdot m$ sections, the task is simply to identify the list of locations (i, j) for n sections which has the lowest total occupancy rate.

Selection conditions:

Specifically, we wish to remove only one section from each of the n rows (**Condition #1**).

To minimise the amount of shuffling work required before the installation, sections selected for removal in two adjacent rows must be in the same or adjacent columns (**Condition #2**). This means that any successive sections from the top row to the bottom row in the final selected sections should be adjacent vertically or diagonally.

The final solution would be the list of indices of n sections from top to down that has the total minimum occupancy rate (**Condition #3**).

To solve this problem, you will write a function `select_sections(occupancy_probability)`. If there are multiple solutions with same total minimum occupancy rate, you can return any one of them.

2.1 Input

occupancy_probability is a list of lists. There are n interior lists. All interior lists are length m (columns/aisles). Each interior list represents a different row of sections. occupancy_probability$[i][j]$ is an integer number between 0 and 100 (inclusive) which represents the occupancy probability for a section located at row i and column/aisle j.

2.2 Output

Your algorithm will return a list of 2 items:

- `minimum_total_occupancy` is an integer, which is the total occupancy for the selected n sections to be removed
- `sections_location` is a list of n tuples in the form of (i, j). Each tuple represents the location of one section selected for removal. i refers to the row index and can range from 0 to $n - 1$. j refers to the column index and can range from 0 to $m - 1$.

2.3 Example

```
# Example
occupancy_probability = [
[31, 54, 94, 34, 12],
[26, 25, 24, 16, 87],
[39, 74, 50, 13, 82],
[42, 20, 81, 21, 52],
[30, 43, 19,  5, 47],
[37, 59, 70, 28, 15],
[ 2, 16, 14, 57, 49],
[22, 38,  9, 19, 99]]

select_sections(occupancy_probability)
>>> [118, [(0, 4), (1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 2), (7, 2)]]
```

2.4 Complexity

`select_sections` should run in $O(nm)$ time and space, where n is the number of rows, and m is the number of columns/aisles. In the above example, $n = 8$, $m = 5$.

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.

Abiding by the complexity requirements is extremely important and failure to do so will be penalised heavily. It may well be worth it to submit a half-working solution within the complexity bounds than a completely working solution that is not within the complexity bounds.

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