FIT2004 S1/2022: Assignment 2

DEADLINE: Thursday 14th April 2022 16:30:00 AEST

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 hours late means 1 day late, 27 hours late is 2 days late. For special consideration, please visit this page: `https://forms.monash.edu/special-consideration` and fill out the appropriate form.

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`.

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 and, as a result, many failed the unit. Helping others to solve the assignment is NOT ACCEPTED. Please do not share your solutions partially or completely to others. If someone asks you for help, ask them to visit a consultation for help.

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. Clarify these questions. You can go to a consultation, talk to your tutor, discuss the tasks with friends or ask in the forums.
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. 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 and the approach undertaken within the function.
- 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 Big-O 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!).

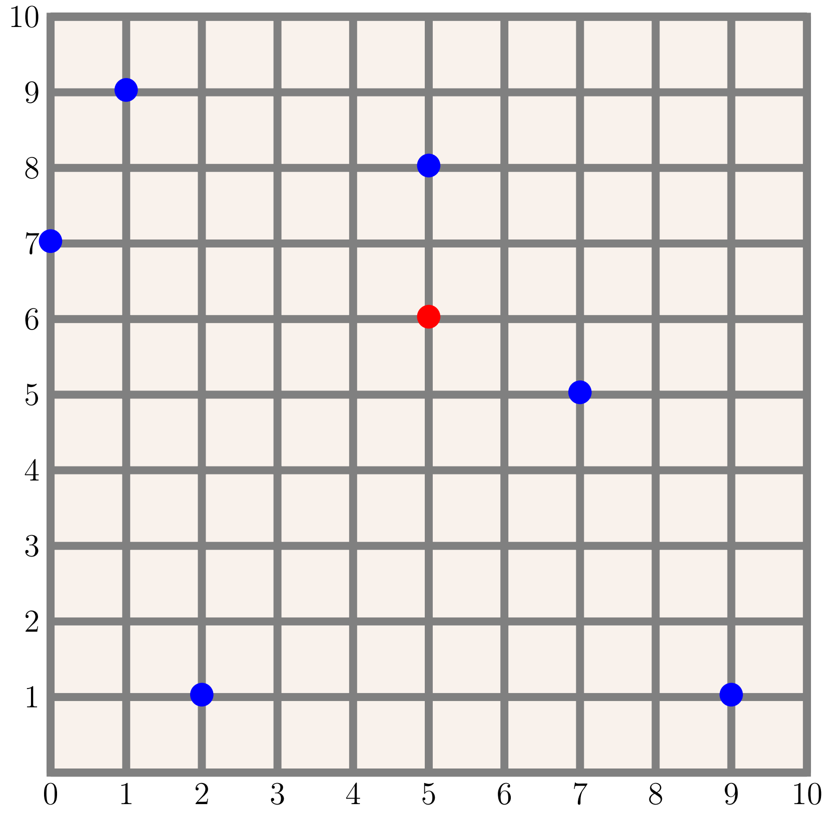
A suggested function documentation layout would be as follows:

```
def my_function(argv1, argv2):
    """
    High level description about the functiona and the approach you
    have undertaken.
    :Input:
        argv1:
        argv2:
    :Output, return or postcondition:
    :Time complexity:
    :Aux space complexity:
    """
    # Write your codes here.
```

1 Location Optimisation (4 marks + 1 mark for documentation)

Your company is planning to open a kiosk in a new city and want to find the perfect location for it. The streets of that city form a perfect grid and it is only possible to move along the streets. You are given a set of n relevant points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, which are all intersections in the grid. Your goal is to compute the coordinates (x, y) of an optimal intersection to open your kiosk such that $\sum_{i=1}^{n}(|x - x_i| + |y - y_i|)$ is minimal.

In the example below, if the relevant points are painted in blue, then the red point is one optimal solution, and the total distance from it to all relevant points is 35.



1.1 Input

Your ideal place finder is a Python function `ideal_place(relevant)`, where `relevant` = $[[x_1, y_1], [x_2, y_2], [x_3, y_3], \dots, [x_n, y_n]]$ contains the coordinates of the n relevant points.

1.2 Output

Your algorithm should output a single pair of coordinates x and y such that $\sum_{i=1}^{n}(|x - x_i| + |y - y_i|)$ is minimal. The output should be in the following format: `[x, y]`. If there are multiple optimal solutions, your algorithm can output anyone of them (but should output exactly one).

1.3 Examples

In the example depicted above the input/output behaviour of your algorithm would be the following:

```
# Example
# The relevant points given as input
relevant = [[5,8], [7,5], [9, 1], [0,7], [1,9], [2,1]]

>>> ideal_place(relevant)
[5, 6]
```

1.4 Complexity

You can assume addition, subtraction, modulus and comparison as basic operations. You cannot assume an upper bound on the coordinates. For a list of n relevant points, your algorithm for finding an ideal place should have a deterministic worst-case time complexity of $O(n)$ basic operations.

2 On The Way (4 marks + 1 mark for documentation)

You live life efficiently – no wasted movements, highly sustainable. If you are to head out to a destination, you would try to squeeze in a chore along the way:

- Going to campus? Let's grab a coffee along the way.
- Going back home? Let's grab dinner on the way back.
- Going to visit a friend? Let's fuel up my car along the route.

But of course, there could be multiple options for your chores. You could grab coffee from any local cafe or Starbucks. Likewise, dinner could be from McDonald's, Hungry Jack's or Subway. In the name of efficiency, your decision is made based on whichever is the closest to your direct route.

Given your FIT2004 study, you have come to the realization that it is possible to model your travels using the Graph data structures.

- The `RoadGraph` class implementation is as described in Section 2.1. An example of the graph structure is provided below.
- You can then calculate the optimal routes for your commute while doing a chore along the way using a function in the Graph class called `routing(self, start, end, chores_location)` detailed in Section 2.2.

```
class RoadGraph:
    def __init__(self, roads):
        # ToDo: Initialize the graph data structure here
    def routing(self, start, end, chores_location):
        # ToDo: Performs the operation needed to find the optimal route.
```

2.1 Graph Data Structure (1 mark)

You must write a class `RoadGraph` that represents the roads in the city. The `__init__` method of `RoadGraph` would take as an input a list of roads `roads` represted as a list of tuples (u, v, w) where:

- u is the starting location ID for a road, represented as a non-negative integer.
- v is the ending location ID for a road, represented as a non-negative integer.
- w is the distance along that road, represented as a non-negative integer.

- You cannot assume that the list of tuples are in any specific order.
- You cannot assume that the roads are 2-way roads.

- You can assume that the location IDs are continuous from 0 to $|V| - 1$ where $|V|$ is the total number of locations.
- You can assume that the maximum number of roads $|E|$ is significantly less than $|V|^2$.

Consider the following example for the roads in a city, are stored as a list of tuples:

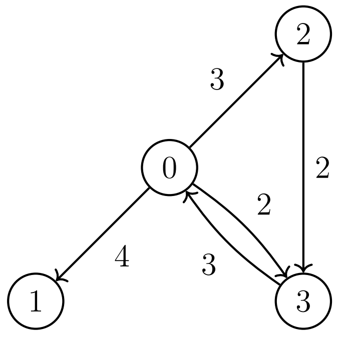
```
# The roads represented as a list of tuples
roads = [(0,1,4), (0,3,2), (0,2,3), (2,3,2), (3,0,3)]
```

- 4 locations in the city with the id of 0 to 3.
- 5 roads total in the city connecting the locations.

- There is a road from location 0 to location 1 with a distance of 4.
- There is a road from location 0 to location 3 with a distance of 2.
- There is a road from location 0 to location 2 with a distance of 3.
- There is a road from location 2 to location 3 with a distance of 2.
- There is a road from location 3 to location 0 with a distance of 3.
- Since you can't assume the roads are 2-way roads, we observe only location 0 and location 3 are connected through a 2-way road although there is a difference in the distance.

Running the following code would create a `RoadGraph` object called `mygraph`, which is then visualized below:

```
# Creating a RoadGraph object based on the given roads
mygraph = RoadGraph(roads)
```



Based on the information above, you would need to implement the `RoadGraph` class using either an adjacency matrix or adjacency list representation. Do note that one implementation is less efficient than the other in both time and space complexity. Thus, consider your implementation carefully in order to achieve full marks.

2.2 Optimal Route Function (3 marks)

You would now proceed to implement `routing(self, start, end, chores_location)` as a function within the `RoadGraph` class. The functions accepts 3 arguments:

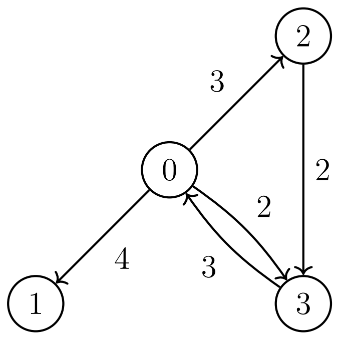
- `start` is a non-negative integer that represents the starting location of your journey. Your route must begin from this location.
- `end` is a non-negative integer that represents the ending location of your journey. Your route must end in this location.
- `chores_location` is a non-empty list of non-negative integers that stores all of the location where your chores could be performed.
- Do note that it is possible for the locations in `chores_location` to include the `start` and/or `end` as well.
- Do note that the locations in `chores_location` is not sorted.
- Do note that all locations in `chores_location` are valid locations in the `roads` list.

The function would then return the shortest route from the `start` location to the `end` location, going through at least 1 of the locations listed in `chores_location`.

- If such route exist, it would return the shortest route as a list of integers. If there are multiple such routes, return any one of these shortest routes.
- If no such route going from the `start` location to one of the locations in `chores_location` and proceeding next to the `end` location is possible, then the function would return `None`.

Several examples are provided in Section 2.3.

2.3 Examples



```
# Example 1
# The roads represented as a list of tuples
roads = [(0,1,4), (0,3,2), (0,2,3), (2,3,2), (3,0,3)]
# Creating a RoadGraph object based on the given roads
mygraph = RoadGraph(roads)
```

Starting from location 0, you are looking to end at location 1. Your chores can be completed at both location 2 and location 3. The optimal route for you is to travel going through location 3 to complete the chore, then going back to location 0 before ending at location 1. This route would only travel a total distance of 9. The alternative route of going through locations 0, 2, 3, 0 and then 1 would give you a total distance of 12 [1].

```
# Example 1.1
# the route inputs
start = 0
end = 1
chores_location = [2,3]

>>> mygraph.routing(start, end, chores_location)
[0, 3, 0, 1]
```

In another example below, you now start at location 0 but look to end in location 3. Your chore can only be completed in location 2 only. Thus the optimal route is to go from location 0 to location 2 to complete your chore and directly to location 3 that is your destination. Your total travel distance is 5.

```
# Example 1.2
# the route inputs
start = 0
end = 3
chores_location = [2]

>>> mygraph.routing(start, end, chores_location)
[0, 2, 3]
```

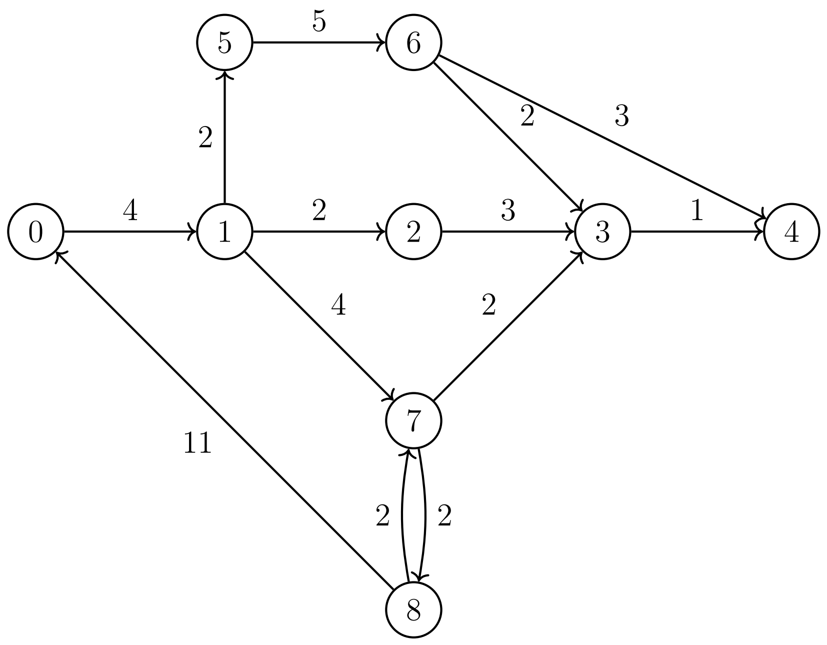
[1]Updated on 1st April 2022 13:28 Malaysia time; old value of 10 is wrong.

The next 2 examples below shows when routes are not possible; with the function returning `None`.

```
# Example 1.3 Chores can't be done
start = 2
end = 0
chores_location = [1]

>>> mygraph.routing(start, end, chores_location)
None
```

```
# Example 1.4 Can't reach end
start = 1
end = 2
chores_location = [0,3]

>>> mygraph.routing(start, end, chores_location)
None
```

In the following graph example, we look at a bigger, longer and more complex graph.



The graph above is generated through the following code.

```
# Example 2
# The roads represented as a list of tuples
roads = [(0, 1 ,4), (1, 2 ,2), (2, 3 ,3), (3, 4 ,1), (1, 5 ,2),
            (5, 6 ,5), (6, 3 ,2), (6, 4 ,3), (1, 7 ,4), (7, 8 ,2),
            (8, 7 ,2), (7, 3 ,2), (8, 0 ,11)]
# Creating a RoadGraph object based on the given roads
mygraph = RoadGraph(roads)
```

In the example below, we start at location 0, end at location 3, and your chores can be completed at locations 5, 7, 8 and 4. There are a few possible routes possible such as $0, 1, 5, 6, 3$ with a distance of 13 or route $0, 1, 7, 3$ with a distance of 10. We would choose the smallest distance.

```
# Example 2.1 Possible Route
start = 0
end = 3
chores_location = [5,7,8,4]

>>> mygraph.routing(start, end, chores_location)
[0, 1, 7, 3]
```

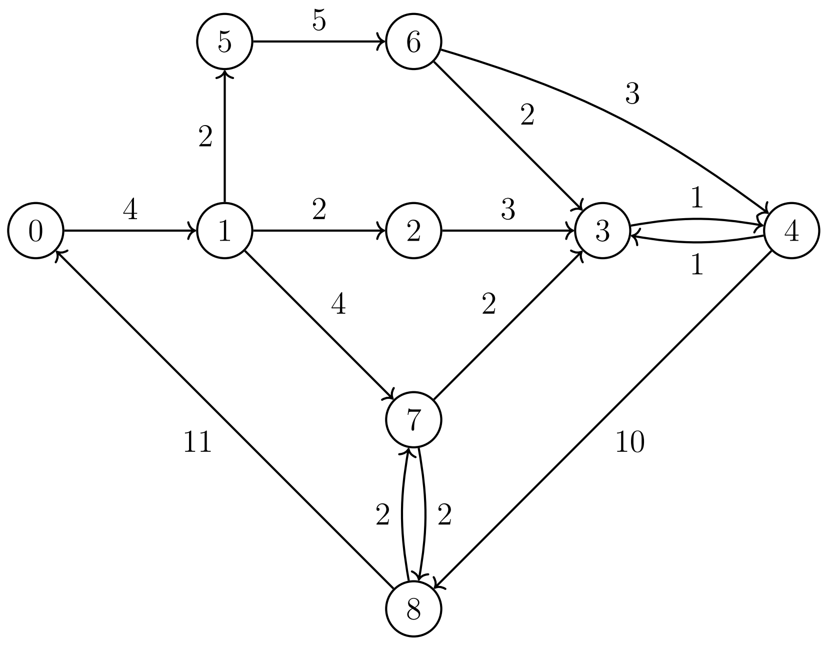
Similarly, if we were to start at location 1 we observe that despite the location 5 would be the closer location to us to perform the chore at a distance of 2 compared to location 7 with a distance of 4; the route after that would be further. Thus, we cannot make be short sighted in our path selection based on the nearest chore.

```
# Example 2.2 Possible Route, despite nearest chore
start = 1
end = 3
chores_location = [5,7,8,4]

>>> mygraph.routing(start, end, chores_location)
[1, 7, 3]
```

In the next example, we saw multiple routes with the same shortest distance – with route $0, 1, 5, 6, 4$ for a distance of 14 and route $0, 1, 5, 6, 3, 4$ for a distance of 14. Returning either would be accepted. Do note that route $0, 1, 7, 8, 7, 3, 4$ has a longer distance of 15 and should not be returned.

```
# Example 2.2 Possible Route, despite nearest chore
start = 0
end = 4
chores_location = [6,8]

>>> mygraph.routing(start, end, chores_location)
[0, 1, 5, 6, 4]
```

For the final example, we have an even more complicated network of roads.



```
# Example 3
# The roads represented as a list of tuples
roads = [(0, 1 ,4), (1, 2 ,2), (2, 3 ,3), (3, 4 ,1), (1, 5 ,2),
(5, 6 ,5), (6, 3 ,2), (6, 4 ,3), (1, 7 ,4), (7, 8 ,2),
(8, 7 ,2), (7, 3 ,2), (8, 0 ,11), (4, 3, 1), (4, 8, 10)]
# Creating a RoadGraph object based on the given roads
mygraph = RoadGraph(roads)
```

The following are 4 cases that were not highlighted by the earlier examples.

```
# Example 3.1 Best chore location is after end location
start = 1
end = 3
chores_location = [4,5,6,8]

>>> mygraph.routing(start, end, chores_location)
[1, 2, 3, 4, 3]
```

```
# Example 3.2 Best chore is before start location
start = 7
end = 4
chores_location = [5,6,8]

>>> mygraph.routing(start, end, chores_location)
[7, 8, 7, 3, 4]
```

```
# Example 3.3 Best chore is alone the shortest route already
start = 0
end = 4
chores_location = [1,3,4,5,6,8]

>>> mygraph.routing(start, end, chores_location)
[0, 1, 2, 3, 4]
```

```
# Example 3.4 There is chore at the start and/ or end location
start = 0
end = 4
chores_location = [0,4,5,6,8]

>>> mygraph.routing(start, end, chores_location)
[0, 1, 2, 3, 4]
```

There could be other cases not covered in the examples provided. Do have a think about what are the other possible scenarios related to the question.

2.4 Complexity

The complexity for this task is separated into 2 main components.

The `__init__(raods)` constructor of `RoadGraph` class would run in $O(|V| + |E|)$ time and space where:

- V is the set of unique locations in `roads`. You can assume that all locations are connected by roads (i.e a connected graph); and the location IDs are continuous from 0 to $|V| - 1$.
- E is the set `roads`.
- You can make an assumption that the number of roads $|E|$ is significantly less than $|V|^2$. Thus, you should not make the assumption that $|E| = \Theta(|V|^2)$.

The `routing(self, start, end, chores_location)` of the `RoadGraph` class would run in $O(|E| log|V|)$ time and $O(|V| + |E|)$ auxiliary space where:

- Note that the `chores_location` is not stated in the complexity. At worst, the size of `chores_location` is $|V|$.

2.5 Hint(s)

In order to do so, you would utilize algorithms and concepts that you have learnt from FIT2004, namely:

- Graph data structure.
- Graph algorithms.

Firstly, ensure a suitable graph representation is chosen to meet the complexity requirement for `__init__(roads)` constructor of `RoadGraph` class. A suitable representation would also ensure the complexity requirements for `routing(self, start, end, chores_location)` is met as well.

The complexity for `routing(self, start, end, chores_location)` can be met by running a certain algorithm once (approach 1) or twice (approach 2) at most [2]. Do take note that this complexity remains the same irregardless of the number of locations in `chores_location`; thus, the solution shouldn't scale in complexity to that.

You are allowed to process the graph data structure/representation in `__init__(roads)` as part of the pre-processing for `routing(self, start, end, chores_location)`. Doing so as you seen in some of the studio questions could greatly improve the complexity for `routing(self, start, end, chores_location)`.

[2] There can be other approaches that would meet the complexity requirements as well.

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.

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