Coursework 2: Tube Map (Version 2)

COMP70053 Python Programming (Autumn 2021)

Deadline: Thursday 28th Oct 2021 (7pm BST) on CATe

1 Overview

In this coursework, you will implement classes that form part of a TubeMap network. In particular, you will attempt to compute the shortest path between two London Tube stations.

The objectives of this coursework are:

- to give you hands-on experience in implementing an existing algorithm;
- to allow you to practise working with advanced Python constructs such as dictionaries, lists, and sets, and reading from JSON files;
- to enable you to use object-oriented programming in a practical application.

This coursework covers topics in Lessons 1 to 9 of the course materials.

You are limited to the **Python Standard Library** for this coursework. Do **NOT** use any external libraries such as `NumPy`.

2 Tasks

You are supplied with some skeleton code (see section 3).

Firstly, examine `tube/components.py`. You are provided with three classes: `Station`, `Line` and `Connection`. Some examples usage for these classes are provided at the end of the file. Figure 1 shows a class diagram with the relationship between the classes.

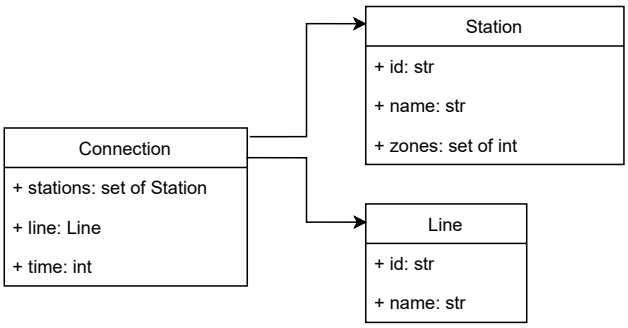

Figure 1: Relationships and dependencies between the classes `Connection`, `Station` and `Line`.

2.1 Task 1: Implementing the `TubeMap` class

You are also given a file `data/london.json` that contains data representing the London Tube map. You will import the data from this file into a useful `TubeMap` object to be used by other classes.

Your first task is to complete the `TubeMap` class in `tube/map.py`. The class comprises a collection of `Station`s, `Line`s, and `Connection`s between `Station`s on particular `Line`s.

As a minimum, the `TubeMap` class must contain these three member attributes (see Figure 2):

- `stations`: A dictionary that indexes `Station` instances by their `id`. When the zone of a station is not an integer in the JSON file, it means the station is in two zones at the same time. For example, if the zone of a station is "2.5", it means the station is in zones 2 and 3. The attribute `zones` should thus be the set `{2, 3}` in your code.
- `lines`: A dictionary that indexes `Line` instances by their `id`.
- `connections`: A list of `Connection` instances for the TubeMap

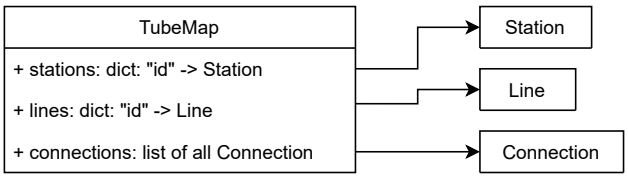

Figure 2: Relationships and dependencies between `TubeMap` and the classes `Station`, `Line` and `Connection`.

The `TubeMap` class must also contain at least the following method:

- `import_from_json(self, filepath)`: This method imports the tube map data from the JSON file indicated by `filepath` and updates the `TubeMap` attributes above. Note that if the `filepath` is invalid, none of those attributes should be updated (and no error should be raised).

A function `test_import()` is provided for you at the end of the file so that (1) you have an example usage of the `TubeMap` object and (2) you have a basic test for `import_from_json` using `data/london.json`.

2.2 Task 2: Implementing the `NeighbourGraphBuilder` class

Now that you have your `TubeMap` instance, you will be able to compute the shortest path from one station to another. For this, you will need to know which stations are connected to each other. You already have this information from the `connections` attribute in `TubeMap` from earlier. You should now encode and index this information into a graph data structure to make it easier for you to compute the shortest path later.

For your second task, complete the `NeighbourGraphBuilder` class in `network/graph.py`.

The class must have a `build()` method as a minimum. The method takes a `TubeMap` instance as input. It returns a nested `dict` representing the graph, or an empty `dict` if the input is invalid.

The nested `dict` is like a 2D grid, except that the indices are the station `id`s of two neighbouring stations (these are strings). The value of the `dict` is **a list of `Connection` instances** taken from `TubeMap.connections`. Examples are provided in the documentation in `network/graph.py`.

Important note: We assume that the connection between stations is bidirectional. For example, if *Baker Street* station (id `"11"`) and *Marylebone* station (id `"163"`) are connected, then you should set the value of both `graph["11"]["163"]` and `graph["163"]["11"]` to the **same list of `Connection` instances between the two stations**.

2.3 Task 3: Implementing the `PathFinder` class

Given that you can now generate a neighbour graph, you can use the graph to compute the shortest path between two stations. When we say 'shortest path', we mean the path that takes the least time.

For your third task, complete the `PathFinder` class in `network/path.py`. More specifically, complete the `get_shortest_path()` method that takes in the start and end station names as string inputs. It should compute the shortest path (in terms of duration) to get from the start station to the end station.

The method should return a list of `Station`s representing **one** shortest path between the start and end stations. If either of the provided station names does not exist, the method should return `None`.

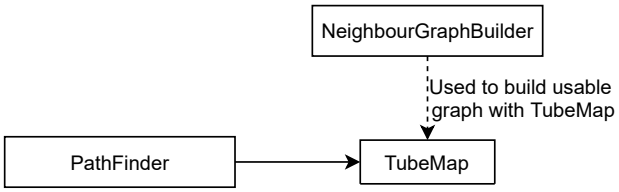

Figure 3: Relationships and dependencies between `PathFinder` and the classes `NeighbourGraphBuilder` and `TubeMap`. The class `PathFinder` does not need to possess a `NeighbourGraphBuilder` instance. However, `NeighbourGraphBuilder` can be used to preprocess the data in `TubeMap` to be used by `PathFinder`.

The dependencies between `PathFinder`, `NeighbourGraphBuilder` and `TubeMap` are briefly summarised in Figure 3.

A widely known algorithm to compute the shortest path on a graph is the Dijkstra algorithm. You can find relevant explanations, illustrations and pseudo-code on the following page: `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Pseudocode`.

3 Skeleton code

You are provided with a git repository containing the skeleton code. You will use `git` to push your code changes to the repository. To obtain the code, clone the repository using either SSH or HTTPS, replacing `<login>` with your College username.

- `git clone git@gitlab.doc.ic.ac.uk:lab2122_autumn/python_cw2_<login>.git`
- `git clone https://gitlab.doc.ic.ac.uk/lab2122_autumn/python_cw2_<login>.git`

Test your code with `LabTS` (`https://teaching.doc.ic.ac.uk/labts`) after pushing your changes to GitLab. Make sure your code runs successfully on LabTS without any syntax errors. You will lose marks otherwise. These automated tests are purely to ensure that your code runs correctly on the `LabTS` servers. **You are responsible for testing your own code more extensively**. The final assessment will be performed using a more extensive set of tests not made available to you.

It is also important to note that:

- The LabTS tests performed on `NeighbourGraphBuilder` rely on our own implementation of `TubeMap`.
- The LabTS tests performed on `PathFinder` rely on our own implementation of `TubeMap` and `NeighbourGraphBuilder`.

This way, if you make a tiny mistake in your implementation of TubeMap, that mistake may not propagate to all tests. In general, the different cases of propagation of errors will be taken into account in the marking scheme.

4 Handing in your coursework

Go to `LabTS` (`https://teaching.doc.ic.ac.uk/labts`), and click on the "`Submit to CATe`" button next to the specific commit you wish to submit. This will automatically submit your commit SHA1 hash to CATe.

5 Grading scheme

Criteria	**Max**	**Details**
Code correctness (Tasks 1+2+3)	4+3+7	Does your code run without any syntax errors? Does your code give the expected output for valid test cases, whether they be standard or edge cases? Does your program not end up in an infinite loop for certain inputs?
Code robustness	2	How well does your code pass our automated stress test with different kinds of input? Is it error free from invalid inputs?
Code readability	4	Did you use good coding style and naming conventions? Did you use meaningful names? Is your code highly abstracted, modular, and self-explanatory? Did you provide useful, meaningful comments where necessary?
TOTAL	**20**	Will be scaled to 6% of your final module grade.

Credits

Coursework designed and prepared by Luca Grillotti and Josiah Wang