JOHANNES KEPLER
UNIVERSITÄT LINZ

Algorithms and Data Structures 2
Winter term 2021
Martin Schobesberger, Stefan Grünberger
Dari Trendafilov, Markus Weninger

Deadline: **Tue. 11.01.2022, 23:59**

Assignment 05

Shortest paths in graphs

Please don't change any skeleton we provide. Of course, you are allowed to add code in your classes, as long as you don't break the skeleton and their interfaces (method names and variables' types) we need for the unit tests to work correctly. **Please remember to fill out the time log survey in Moodle.**

1. JKU navigation with Dijkstra — 24 points

Develop a simple walking distance information system for the JKU that determines the shortest distances between different parts of the JKU.
For this implement the class **`JKUMap`,** which is derived from the class **`Graph`** (from **your own** assignment 04) and additionally implements further methods declared in the skeleton (see below). Implement these methods, as well as a constructor that inserts all 17 vertices and 19 (undirected) edges in the graph from Figure 1.

Each vertex of this graph represents a POI (points of interests) at or around JKU. All POIs have a name. The edge weights represent the direct distance (in meter) between two POIs. Starting from any POI, the **`JKUMap`** provides three methods: (1) One returning the shortest distances to all other POIs (**`get_shortest_distances_from`**), (2) one returning the number of steps needed to reach each other POI via their shortest path (**`get_steps_for_shortest_path_from`**), and (3) one returning the shortest path between two POIs (**`get_shortest_path_from_to`**), all of them using the **shortest path algorithm of Dijkstra.**

```
    def get_shortest_path_from_to(self, from_vertex: Vertex, to_vertex: Vertex):
        """
        This method determines the shortest path between two POIs "from_vertex" and "to_vertex".
        It returns the list of intermediate steps of the route that have been found
        using the dijkstra algorithm.

        :param from_vertex: Start vertex
        :param to_vertex:   Destination vertex
        :return:
           The path, with all intermediate steps, returned as an list. This list
           sequentially contains each vertex along the shortest path, together with
           the already covered distance (see example on the assignment sheet).
           Returns None if there is no path between the two given vertices.
        :raises ValueError: If from_vertex or to_vertex is None, or if from_vertex equals to_vertex
        """

    def get_shortest_distances_from(self, from_vertex: Vertex):
        """
        This method determines the shortest paths from a given "from" vertex to all other vertices.
        The shortest distance (or -1 if no path exists) to each vertex is returned
        as a dictionary, using the vertex name as key and the distance as value.

        :param from_vertex: Start vertex
        :return
           A dictionary containing the shortest distance (or -1 if no path exists) to each vertex,
           using the vertex name as key and the distance as value.
        :raises ValueError: If from_vertex is None.
        """

    def get_steps_for_shortest_paths_from(self, from_vertex: Vertex):
        """
        This method determines the amount of "steps" needed on the shortest paths
        from a given "from" vertex to all other vertices.
        The number of steps (or -1 if no path exists) to each vertex is returned
        as a dictionary, using the vertex name as key and the number of steps as value.
        E.g., the "from" vertex has a step count of 0 to itself and 1 to all adjacent vertices.

        :param from_vertex: start vertex
        :return:
           A dictionary containing the number of steps (or -1 if no path exists) on the
           shortest path to each vertex, using the vertex name as key and the number of steps as
           value.
        :raises ValueError: If from_vertex is None.
        """
```

JOHANNES KEPLER UNIVERSITÄT LINZ

Algorithms and Data Structures 2
Winter term 2021
Martin Schobesberger, Stefan Grünberger
Dari Trendafilov, Markus Weninger

Assignment 05

Deadline: **Tue. 11.01.2022, 23:59**

The method **`get_shortest_path_from_to(self, from_vertex: Vertex, to_vertex: Vertex)`** determines the shortest path from one POI to another POI. An intermediate step on this path, with the intermediate distances up to that point, is represented by the class **`Step`**:

```
class Step():
    def __init__(self, point: Vertex, covered_distance: int):
        self.point = point                          # point of interest visited on this path
        self.covered_distance = covered_distance  # covered distance from the start to this point
```

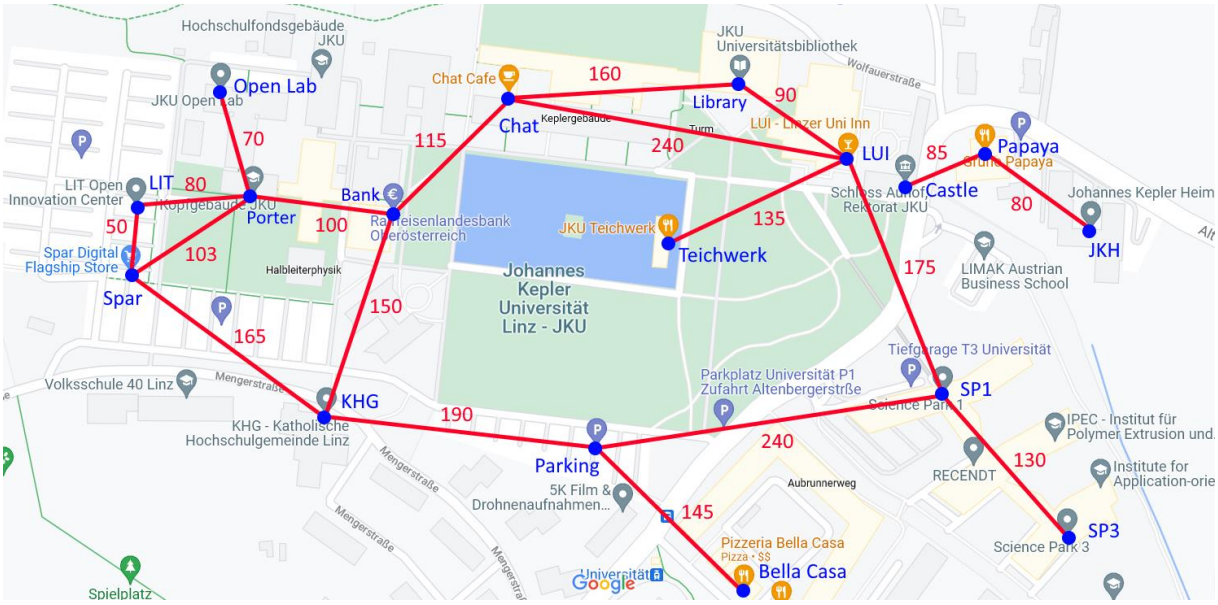

Figure 1: Map of JKU, the POIs used in this exercise, as well as the distances between these POIs.

In the following example (the graph is illustrated in Figure 1), you can see the returned list of **`get_shortest_path_from_to(SP3, Spar)`**:

Index 0	**1**	**2**	**3**	**4**
[SP3, 0]	[SP1, 130]	[Parking, 370]	[KHG, 560]	[Spar, 725]

Assignment 05

Deadline: **Tue. 11.01.2022, 23:59**

Additionally, the method **`get_shortest_distances_from`** determines the shortest distance from one station to **all other stations,** while the method **`get_steps_for_shortest_paths_from`** returns the number of steps on the respective shortest paths. Stations that cannot be reached shall have the distance and a step count of -1, the station where you start from has distance and a step count of 0. The results are both returned in form of a dictionary, where the POI's name is used as key and the distance / step is used as value.
In the following example, you see the returned values for both, **`get_shortest_path_from_to`** and **`get_steps_for_shortest_paths_from,`** applied on **LUI**.

POI	**`get_shortest_distances_from`**	**`get_steps_for_shortest_paths_from`**
Castle	-1	-1
Papaya	-1	-1
JKH	-1	-1
LUI	*0*	*0*
Library	90	1
Chat	240	1
Teichwerk	135	1
SP1	175	1
SP3	305	2
Parking	415	2
Bank	355	2
Bella Casa	560	3
KHG	505	3
Porter	455	3
Spar	558	4
LIT	535	4
Open Lab	525	4

Hints:
1. The examples above are also implemented in the provided unit tests.
2. For the implementation of this assignment a method like the following is recommended:

```
def _dijkstra(self, cur: Vertex, visited_list, distances: dict, paths: dict):
    """
    This method is expected to be called with correctly initialized data structures and recursively
calls itself.

    :param cur: Current vertex being processed
    :param visited_list: List which stores already visited vertices.
    :param distances: Dict (nVertices entries) which stores the min. distance to each vertex.
    :param paths: Dict (nVertices entries) which stores the shortest path to each vertex.
    """
```

3. You might also introduce further private methods, for example a helper method to initialize your data structures (such as initial distances or initial paths).

Submission

Please submit your **`jku_map.py`** as well as your graph**`.py`** in a ZIP archive and name the file **k12345678-assignment05.zip** where k12345678 should reflect your student ID.