Institut für Pervasive Computing

Algorithms and Data Structures 2
Winter term 2023

Assignment 04

Deadline: **Wed. 10.01.2024, 23:59**

Graphs + Shortest Path

Please don't change any skeleton we provide. Of course, you are allowed to add code (methods and global variables), as long as you don't break the skeleton (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. Graphs and their terminology — 2.5+1+1+1+2.5=8 points

Please solve the following exercises using **pen & paper**.

For the parts where you have to use digits of your student ID: **k12345678, d1 = 1, d2 = 2, …, d8 = 8 (if one of the digits is 0, use 1 instead)**

1. Given the following adjacency matrix, insert d6 to d8 of your student ID as edge weights, add an edge from d2 to d3 with edge weight d4 (overwriting existing edges if needed), **draw the corresponding graph** (incl. edge weight – if you want, you can use a graph drawing engine such as DOT/Graphviz*) and **answer the following questions**:
    a. Is the graph weighted (yes / no)?
    b. Is it directed (yes / no)?
    c. Which vertex / vertices has / have the highest in-Degree (vertex id)?
    d. Which vertex / vertices has / have the highest out-Degree (vertex id)?
    e. What is the largest edge weight (number)?
    f. Is the graph cyclic (cyclic / acyclic)?
    g. How many loops are there (number)?
    h. Is the graph connected (no / weakly / strongly)?
    i. Is the graph a tree (yes / no)?

    * DOT/Graphviz visualization template: link

Adjacency matrix:

from/to	1	2	3	4	5	6	7	8	9
1		5		4					
2	5		5	1					
3				d6	d7	d8			
4							1		
5								2	
6									1
7								4	
8									4
9									

2. Draw a complete undirected graph with at least 5 vertices.

3. Draw an undirected graph with at least 3 components, where each vertex has at least a degree of 2.

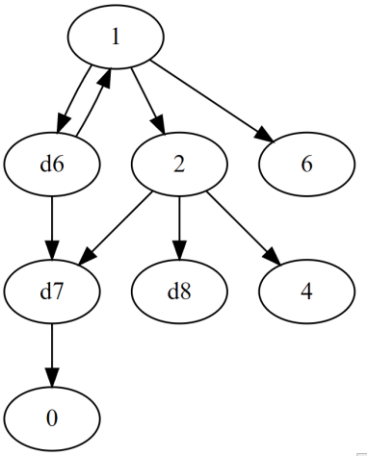
4. Draw a cyclic directed graph with at least 4 vertices, where at least one vertex is not part of the cyclic path.

Institut für Pervasive Computing

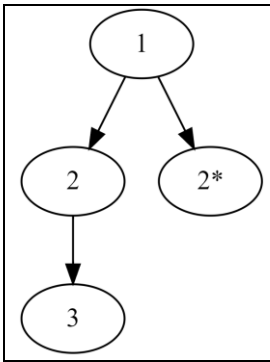
Assignment 04

Deadline: **Wed. 10.01.2024, 23:59**

5. Perform a DFS on the following (right) graph. If one of your student ID digits would cause a duplicate in the graph, append an asterisk to the digit (i.e., d6 = 2 and would be a duplicate, then d6 = 2*). Asterisk numbers come after their respective non-asterisk numbers (i.e., 2* comes after 2 but before 3 in ascending order). Start DFS at 1 and take notes on each step: current node - adjacent nodes (ascending) - next node (smallest adjacent or "up") - visited nodes



Example




```
1 -     [2,2*] - 2      - [1]
2 -     [3]    - 3      - [1,2]
3 -     []     - 2(up)  - [1,2,3]
2 -     [3]    - 1(up)  - [1,2,3]
1 -     [2,2*] - 2*     - [1,2,3]
2* -    []     - 1(up)  - [1,2,3,2*]
1 -     [2,2*] - end    - [1,2,3,2*]
```

Institut für Pervasive Computing

Assignment 04

Deadline: **Wed. 10.01.2024, 23:59**

2. JKU navigation with Dijkstra — 12 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`** (a ready-to-use graph implementation provided as skeleton for this exercise) and additionally has to implement further methods (see below).

`JKUMap` already contains a constructor that builds the graph shown in Figure 1.

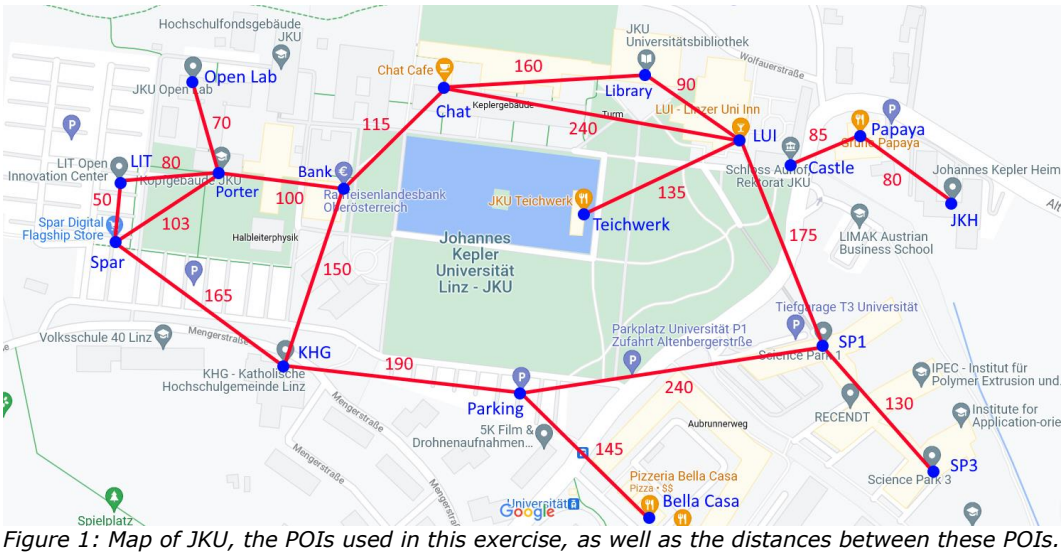


Figure 1: Map of JKU, the POIs used in this exercise, as well as the distances between these POIs.

Each vertex of this graph represents a POI (point of interest) 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 two methods that you have to implement: One returning the shortest path between two POIs (**`get_shortest_path_from_to`**) and one returning the shortest distances to all other POIs (**`get_shortest_distances_from`**), both 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.
    """
```

Institut für Pervasive Computing

Assignment 04

Deadline: **Wed. 10.01.2024, 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
```

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]

The method **`getShortestDistancesFrom`** determines the shortest distance from one POI to **all other POIs**. Stations that cannot be reached shall have the distance of -1, the station where you start from has distance of 0. The result is returned in form of a map, where the POI's name is used as key and the distance is used as value.
In the following example, you see the returned values for **`getShortestDistancesFrom`** applied on **LUI**.

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

Hints:

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

```
def _dijkstra(self, cur: Vertex, visited_set, 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_set: Set 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 **pen-and-paper** solution as **`.pdf`** and **`jku_map.py`** in a ZIP archive and name the file **k12345678-assignment04.zip** where k12345678 should reflect your student ID.