Institut für Pervasive Computing

Algorithms and Data Structures 2
Winter term 2024

Assignment 04

Deadline: **Thu. 26.12.2024, 23:59**

Time log: Please remember the time you needed to solve this assignment and log it in the feedback form in Moodle! This information is fully anonymous.

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.

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.

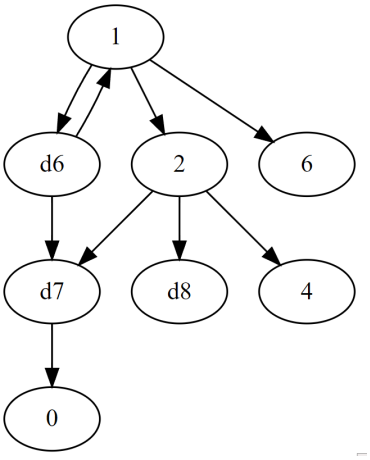
4. Draw a cyclic directed graph with at least 4 vertices, where at least one vertex is not part of the cyclic path.

Institut für Pervasive Computing

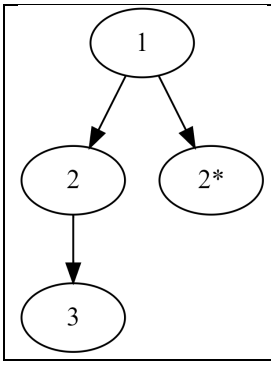
Assignment 04

Deadline: **Thu. 26.12.2024, 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: **Thu. 26.12.2024, 23:59**

2. JKU navigation with Dijkstra — 12 points

Develop a simple walking distance information system for the JKU that calculates the shortest distances between various parts of the campus.You will be provided with a skeleton file **(Dijkstra.py)** containing the implementation of several classes to model and manipulate a graph:

- **Node:** Represents a graph node with a name.
- **Edge:** Represents a weighted, undirected edge connecting two nodes.
- **Step:** Represents a step in a path, including the covered distance.
- **Graph:** A general graph structure with methods to manage nodes and edges.
- **JKUMap:** A specific graph implementation that models a map of locations and their connections.

The JKUMap class already includes a constructor that builds the graph shown in Figure 1. However, you need to implement additional methods as specified below.

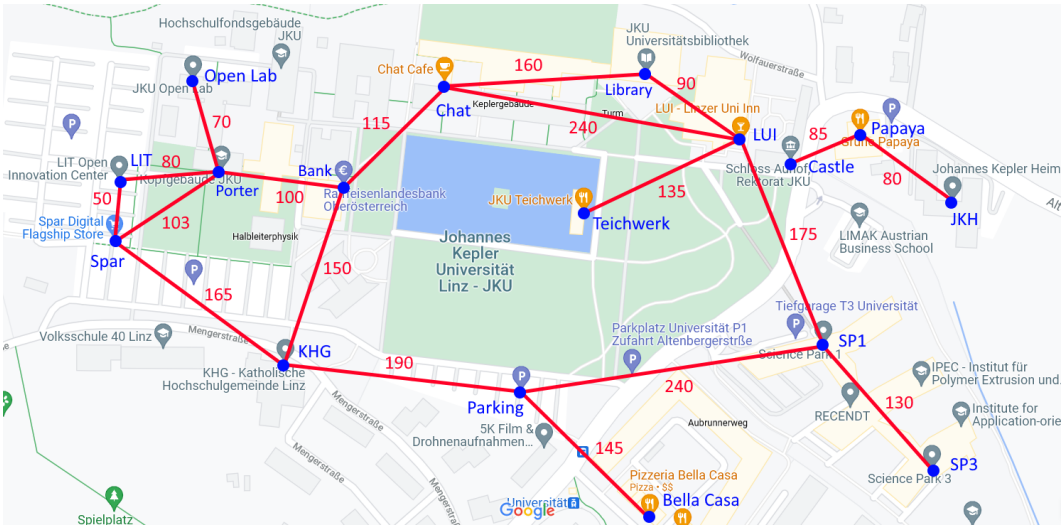


Figure 1: Map of JKU, the POIs used in this exercise, as well as the distances between these POIs.

Each node in this graph represents a POI (point of interest) at or around JKU. All POIs are assigned a name. The edge weights represent the direct distance (in meters) between two POIs. Starting from any POI, the JKUMap class includes three methods that you need to implement:

1. **get_shortest_path**: Returns the shortest path between two POIs using Dijkstra's shortest path algorithm.
2. **get_shortest_distances**: Returns the shortest distances from a given POI to all other POIs, also using Dijkstra's algorithm.
3. **reachable**: Checks whether there is a path between two points using either **Breadth-First Search (BFS)** or **Depth-First Search (DFS).**

```
def get_shortest_path(self, from_node, to_node):
    """
    This method determines the amount of "steps" needed on the shortest paths
    from a given "from" node to all other nodes.
    The number of steps (or -1 if no path exists) to each node is returned
    as a dictionary, using the node name as key and the distance as value.
    E.g., the "from" node has a step count of 0 to itself and 1 to all adjacent nodes.
    @:param from_node: start node
    @return:
    The path, with all intermediate steps, returned as an ArrayList. This list * sequentially contains each
    node along the shortest path, together with * the already covered distance. * Returns null if there is
    no path between the two given nodes.
    :raises ValueError: If from_node or to_node is None.
    """

def get_shortest_distances(self, from_node):
    """
    This method determines the shortest paths from a given "from" node to all other nodes.
    The shortest distance (or -1 if no path exists) to each node is returned
    as a dictionary, using the node name as key and the distance as value.
    :param from_node: Start node
    :return
    A dictionary containing the shortest distance (or -1 if no path exists) to each node,
    using the nodes name as key and the distance as value.
    :raises ValueError: If from_node is None.
    """
```

Institut für Pervasive Computing

Assignment 04

Deadline: **Thu. 26.12.2024, 23:59**

```
def reachable(self, from_node, to_node):
        """
       This method determines whether there exists a path between from_node and to_node.
       :param from Start node
       :param from Target node
       :return true if a path exists, false otherwise
       :throws ValueError If from or to is null.
        """
```

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

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

The method **`get_shortest_distances`** 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 **`get_shortest_distances`** applied on **LUI**.

POI	**`getShortestDistances(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, visited, distances, paths):
    """
    This method is expected to be called with correctly initialized data structures and recursively calls
    itself.

    :param cur: Current node being processed
    :param visited: Set which stores already visited nodes.
    :param distances: Dict (nNodes entries) which stores the min. distance to each node.
    :param paths: Dict (nNodes entries) which stores the shortest path to each node.
    """
```

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).

Assignment 04

Deadline: **Thu. 26.12.2024, 23:59**

4. If you want to use the Interactive Dijkstra Visualization at https://bachelorthesis.live/algorithms/dijkstra you can use the following DOT graph definition. Please fill out the survey on Moodle to give us feedback on this tool that is actively being developed, thank you!

```
graph {
splines=true;
node0 [label="Open Lab"]
node1 [label="LIT"]
node2 [label="Porter"]
node3 [label="Spar"]
node4 [label="Bank"]
node5 [label="KHG"]
node6 [label="Chat"]
node7 [label="Parking"]
node8 [label="Library"]
node9 [label="Teichwerk"]
node10 [label="Bella Casa"]
node11 [label="LUI"]
node12 [label="SP1"]
node13 [label="SP3"]
node14 [label="Castle"]
node15 [label="Papaya"]
node16 [label="JKH"]
node15 -- node14 [label=85]
node15 -- node16 [label=80]
node0 -- node2 [label=70]
node2 -- node1 [label=80]
node2 -- node3 [label=103]
node2 -- node4 [label=100]
node3 -- node1 [label=50]
node3 -- node5 [label=165]
node5 -- node4 [label=150]
node4 -- node6 [label=115]
node5 -- node7 [label=190]
node7 -- node10 [label=145]
node7 -- node12 [label=240]
node12 -- node13 [label=130]
node12 -- node11 [label=175]
node11 -- node9 [label=135]
node11 -- node8 [label=90]
node11 -- node6 [label=240]
node8 -- node6 [label=160]}
```

Submission

Please submit your **pen-and-paper** solution as **.pdf** and Dijkstra.py in a ZIP archive and name the file **k12345678-assignment04.zip** where k12345678 should reflect your student ID.