JOHANNES KEPLER UNIVERSITÄT LINZ

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

Assignment 04

Deadline: **Tue. 21.12.2021, 23:59**

Graphs

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 4+1+1+1+3=10 points

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

k11942708

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, add an edge from d2 to d3 with edge weight d4 (overwriting existing edges if needed), insert d6 to d8 of your student ID as edge weights, **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:
   https://dreampuf.github.io/GraphvizOnline/#digraph%20G%20%7B%0A%20%20%20%201-%3E2%20%5Blabel%3D5%5D%0A%20%20%20%202-%3E3%20%5Blabel%3D10%5D%0A%20%20%20%20%23%20...%0A%7D

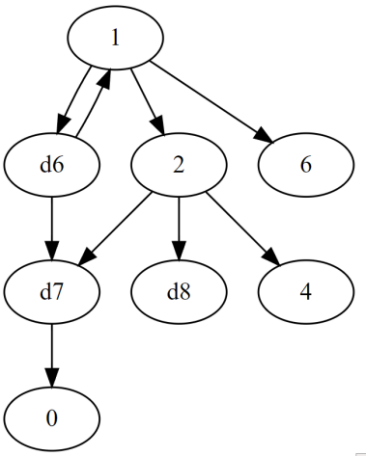
   Adjacency matrix:

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									

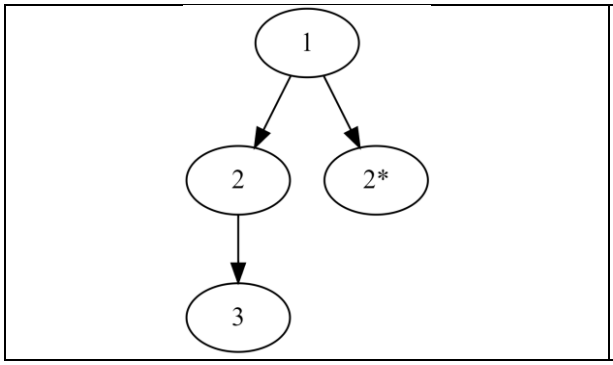
2. Draw a complete undirected graph with at least 5 vertices.
3. Draw an undirected graph with at least 3 components, where each vertex has at least a degree of 2.
4. Draw a cyclic directed graph with at least 4 vertices, where at least one vertex is not part of the cyclic path.
5. Perform a DFS on the following 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

Assignment 04

Deadline: **Tue. 21.12.2021, 23:59**



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*]
```

2. Graph implementation using an Edge List — 14 points

Implement an **undirected, weighted** graph without loops using an **edge list**. Instances of the class `Vertex` represent a vertex in a graph, instances of the class `Edge` represent undirected, weighted edges between two vertices. The graph itself has to be defined in the class `Graph` by you, don't change the given method declarations. **Make sure to provide a working implementation, since you will need a working graph implementation for assignment 5 (shortest path algorithms).**

```
class Vertex():
    def __init__(self, index, name=""):
        self.idx = index   # index at which the vertex has been added
        self.name = name   # name of the vertex
```

```
class Edge():
    def __init__(self, first_vertex, second_vertex, weight):
        self.first_vertex = first_vertex   # reference to a vertex
        self.second_vertex = second_vertex   # reference to a vertex
        self.weight = weight               # weight of the edge
```

```
from vertex import Vertex
from edge import Edge


class Graph():
    def __init__(self):
    self.vertices = []  # list of vertices in the graph
    self.edges = []  # list of edges in the graph
    self.num_vertices = 0
    self.num_edges = 0
    self.undirected_graph = True
```

ID: 11942708

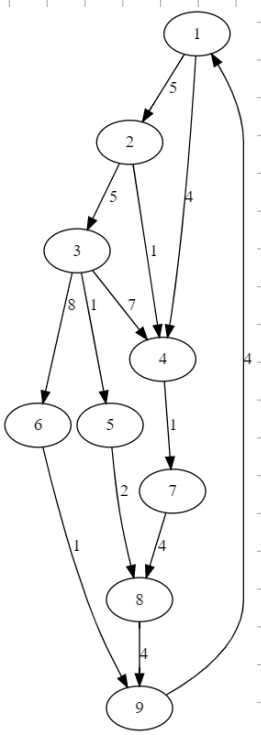
-> 1 1 9 4 2 7 0 8

d1 d2 d3 d4 d5 d6 d7 d8

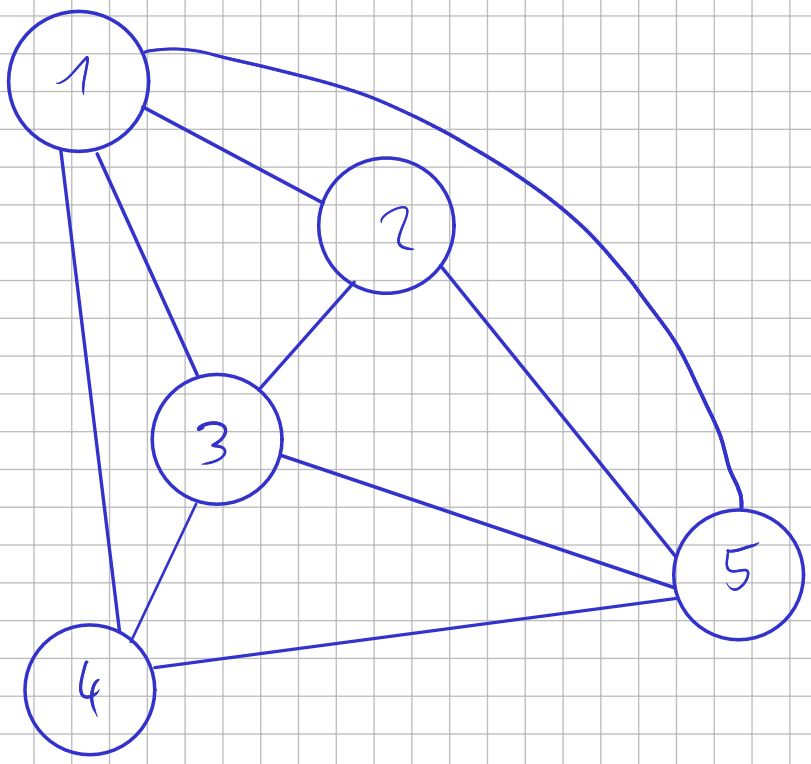
Adjacency matrix:

from/to	1	2	3	4	5	6	7	8	9
1		5		4					
2	5		5	1					
3				7	0	8			
4							1		
5								2	
6									1
7								4	
8									4
9	4								

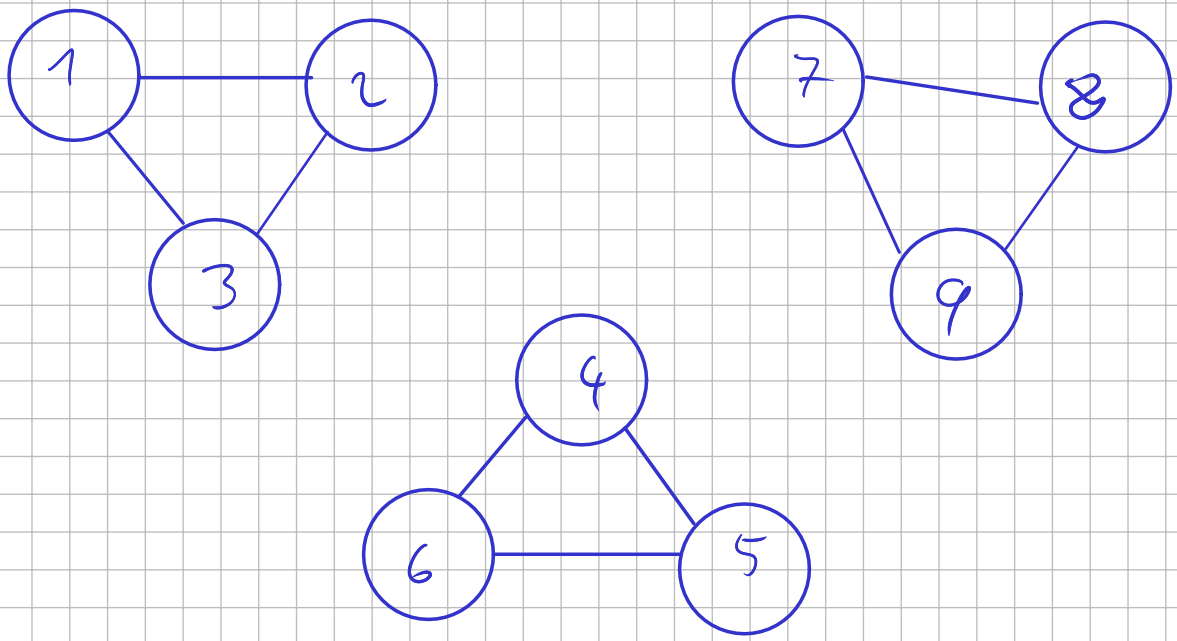
a. Is the graph weighted (yes / no)? ✓
b. Is it directed (yes / no)? ✓
c. Which vertex / vertices has / have the highest in-Degree (vertex id)?
d. Which vertex / vertices has / have the highest out-Degree (vertex id)?
e. What is the largest edge weight (number)? 8
f. Is the graph cyclic (cyclic / acyclic)?
g. How many loops are there (number)? 0
h. Is the graph connected (no / weakly / strongly)?
i. Is the graph a tree (yes / no)? no



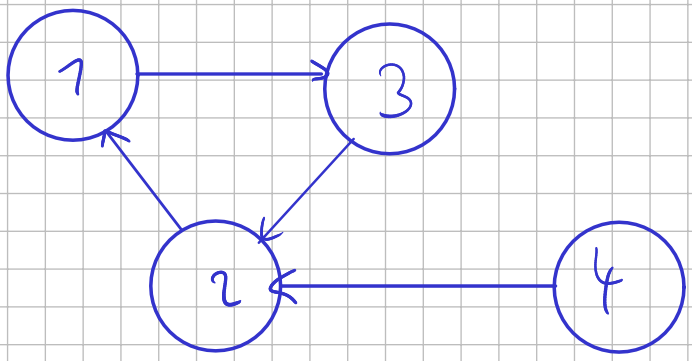
2. Draw a complete undirected graph with at least 5 vertices.



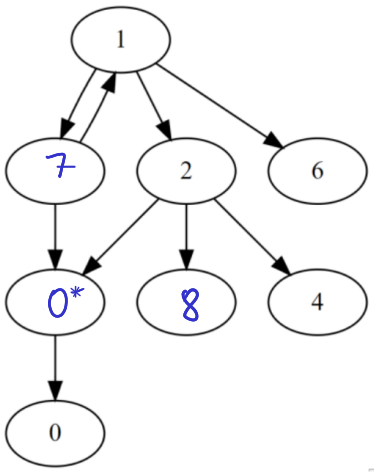
3. Draw an undirected graph with at least 3 components, where each vertex has at least a degree of 2.



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



5. Perform a DFS on the following 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



7 0 8
d6 d7 d8

1 - [2,6,7] - 2 - [1]
2 - [0*,4,8] - 0* - [1,2]
0* - [0] - 0 - [1,2,0*]
0 - [] - 0* (up) - [1,2,0*]
0* - [0] - 2 (up) [1,2,0*]
2 - [0*,4,8] - 4 - [1,2,0*,4]
4 - [] - 2 (up) - [1,2,0*,4]
2 - [0*,4,8] - 8 - [1,2,0*,4]
8 - [] - 2 (up) - [1,2,0*,4,8]
2 - [0*,4,8] - 1 (up) - [1,2,0*,4,8]
1 - [2,6,7] - 6 - [1,2,0*,4,8]
6 - [] - 1 (up) - [1,2,0*,4,8,6]
1 - [2,6,7] - 7 - [1,2,0*,4,8,6]
7 - [0*] - end - [1,2,0*,4,8,6,7]