Exercise Sheet 13

Handout: December 12th — Deadline: December 19th, 4pm

Question 13.1 (0.5 marks)

Consider a directed graph $G(V, E)$.

1. With an adjacency list representation, how long does it take to compute the in-degree of a vertex $v \in V$?
2. With an adjacency list representation, how long does it take to compute the in-degrees of all vertices $v \in V$?
3. With an adjacency matrix representation, how long does it take to compute the in-degree of a vertex $v \in V$?
4. With an adjacency matrix representation, how long does it take to compute the in-degree of all vertices $v \in V$?

Question 13.2 (0.5 marks)

A mayor of a city decides to monitor every road in the city with 360° video surveillance cameras. Imagine the road network as an undirected graph where edges represent roads and vertices represent junctions. When a video camera is placed on a junction, it can monitor all incident roads.

In graph terms, an edge is called *monitored* if there is a camera on at least one of its vertices. The goal is to identify on which vertices to put cameras in order to monitor every edge with a minimum number of cameras.

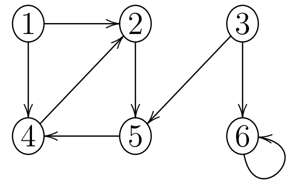
The mayor decides to use the following strategy: *While there is an unmonitored edge, put video cameras on **both** of its vertices.*

How good is this strategy? Does it always produce an optimal solution? Does it come close? Justify your answer.

Can you think of a greedy strategy for this problem?

Question 13.3 (0.25 marks)

Perform a breadth-first search on the following graph with vertex 3 as source. Show the d and π values of each node.



Question 13.4 (0.25 marks)

State what happens if BFS uses a single bit to store the colour of each vertex (0 for white and 1 for gray) and thus the last line of the algorithm is removed.

Question 13.5 (0.25 marks)

What is the running time of BFS if an adjacency matrix representation is used instead of an adjacency list?

Question 13.6 (1 mark) Implement BFS(G, s) for a given undirected graph $G(V, E)$ and the procedure PRINT-PATH(G, s, v). The input will be:

- first line: N M (the number of vertices and edges).
- second line: the source node s.
- M lines each containing a pair $v_i v_j$ meaning there is an edge between these two nodes.
- the final two lines contain one node each v_y and v_z to be given input to PRINT-PATH(G, s, v).

You have to first build the adjacency list representing the graph with the required attributes (colour, .d, .π). And then run the two algorithms on the graph G.

The output will be the two paths from s to v_y and from s to v_z.

You can use library functions for Enqueue and Dequeue if you prefer.