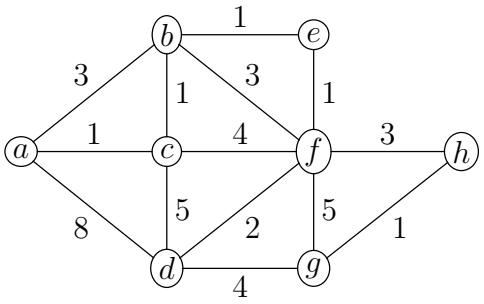
Exercise Sheet 15

Handout: December 21st — Deadline: December 28th, 4pm

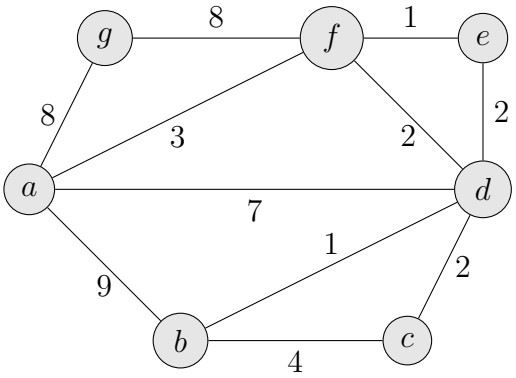
Question 15.1 (0.25 marks)

Compute the minimum spanning tree of the following weighted graph both with Prim's and Kruskal's algorithm. List the edges in the order considered, draw the tree and calculate its weight.



Question 15.2 (0.25 marks)

Execute Dijkstra's algorithm on the following weighted graph to find a shortest path from vertex a to c. Show for each iteration of the while loop which vertex is added to the set S and how the distance estimates of adjacent vertices are being refined.



Question 15.3 (0.5 marks)

A precondition for Dijkstra's algorithm is that all edges of the directed graph under consideration have non-negative weight.

Somebody on the internet claims that the algorithm works for graphs with negative edge weights as well: just add an appropriate constant c to each edge weight to make all weights positive, then run Dijksta's algorithm, and finally remove the constants from the shortest paths computed.

Give a directed acyclic graph as a counterexample to falsify this claim. Explain in your own words what goes wrong.

Question 15.4 (3 marks)

Implement DIJKSTRA(G, w, s) for a given directed graph $G(V, E)$ with weight vector w. The algorithm should return the shortest paths from the source s to all other nodes. The input will be:

- first line: N M (the number of vertices and edges).
- M lines each containing a pair $v_i v_j$ meaning there is an edge $v_i \rightarrow v_j$ followed by the weight of the edge.
- the final two lines contain one node each v_y and v_z to be given input to PRINT-PATH(G, s, v) which will print the shortest path from the source to the respective node.

You have to first implement a MIN-PRIORITY-QUEUE and the respective procedures (EXTRACT-MIN(Q), DECREASE-KEY(Q, x, k)) - You should NOT USE libraries for this.

Remember that you need a "handle" to keep track of where each node is in the heap - see Lecture Slides Week 8 to recap Priority Queues.

Then you have to build the adjacency list representing the graph.

Finally you can implement Dijkstra.