Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Johannes Lengler, Markus Püschel, David Steurer
Kasper Lindberg, Kostas Lakis, Lucas Pesenti, Manuel Wiedmer

24 November 2025

Algorithms & Data Structures — Exercise sheet 10 — HS 25

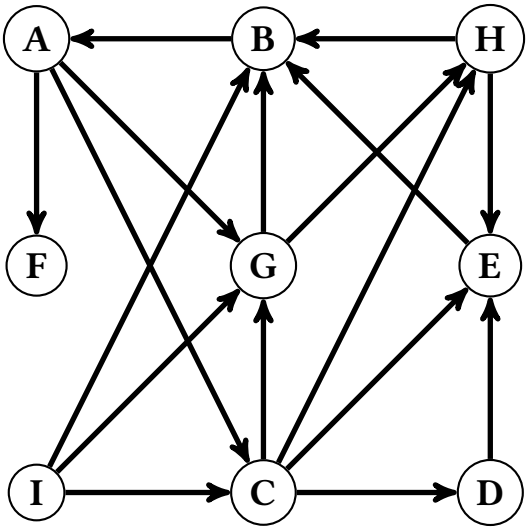
The solutions for this sheet are submitted on Moodle until 30 November 2025, 23:59.

Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

You can use results from previous parts without solving those parts.

Exercise 10.1 *Depth-first search* **(1 point)**.

Execute a depth-first search (*Tiefensuche*) on the following graph. Use the algorithm presented in the lecture. Always do the calls to the function "visit" in alphabetical order, i.e. start the depth-first search from A and once "visit(A)" is finished, process the next unmarked vertex in alphabetical order. When processing the neighbors of a vertex, also process them in alphabetical order.



(a) Mark the edges that belong to the depth-first forest (*Tiefensuchwald*) with a "T" (for tree edge).

(b) For each vertex in the depth-first forest, give its *pre*- and *post*-number.

(c) Give the vertex ordering that results from sorting the vertices by pre-number. Give the vertex ordering that results from sorting the vertices by post-number.

(d) Mark every forward edge (*Vorwärtskante*) with an "F", every backward edge (*Rückwärtskante*) with a "B", and every cross edge (*Querkante*) with a "C".

(e) Does the above graph have a topological ordering? If yes, write down the topological ordering we get from the above execution of depth-first search; if no, argue how we can use the above execution of depth-first search to find a directed cycle.

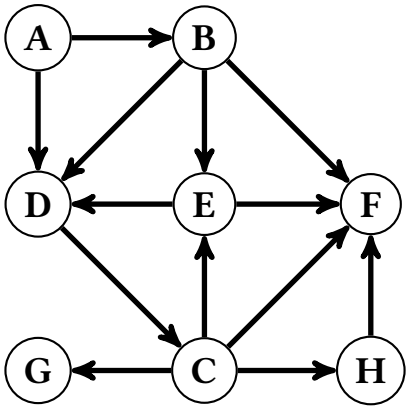
(f) Draw a scale from 1 to 18, and mark for every vertex v the interval I_v from pre-number to post-number of v. What does it mean if $I_u \subset I_v$ for two different vertices u and v?

(g) Consider the graph above where the edge from B to A is removed and an edge from F to I is added. How does the execution of depth-first search change? Does the graph have a topological ordering? If yes, write down the topological ordering we get from the execution of depth-first search; if no, argue how we can use the execution of depth-first search to find a directed cycle. If you sort the vertices by *pre-number*, does this give a topological sorting?

Exercise 10.2 *Breadth-First Search* **(1 point)**.

Execute a breadth-first search (Breitensuche) on the following (directed) graph starting from vertex A. Use the algorithm presented in the lecture.

When processing the neighbors of a vertex, process them in alphabetical order.



a) Provide the BFS order of visit of the nodes.

b) Provide the enter and leave time of each node.

c) Indicate the shortest-path-tree that is obtained by BFS.

d) Determine the distance from A to every node in the graph.

Exercise 10.3 *Driving on highways.*

In order to encourage the use of train for long-distance traveling, the Swiss government has decided to make all the m highways between the n major cities of Switzerland one-way only. In other words, for any two of these major cities C_1 and C_2, if there is a highway connecting them it is either from C_1 to C_2 or from C_2 to C_1, but not both. The government claims that it is however still possible to drive from any major city to any other major city using highways only, despite these one-way restrictions.

(a) Model the problem as a graph problem. Describe the set of vertices V and the set of edges E in words. Reformulate the problem description as a graph problem on the resulting graph.

(b) Describe an algorithm that checks the correctness of the government's claim in time $O(n + m)$. Argue why your algorithm is correct and why it satisfies the runtime bound.

Hint: *You can make use of an algorithm from the lecture. However, you might need to modify the graph described in part (a) and run the algorithm on some modified graph.*

Exercise 10.4 *Shortest paths by hand.*

Dijkstra's algorithm allows to find shortest paths in a directed graph when all edge costs are nonnegative. Here is a pseudo-code for that algorithm:

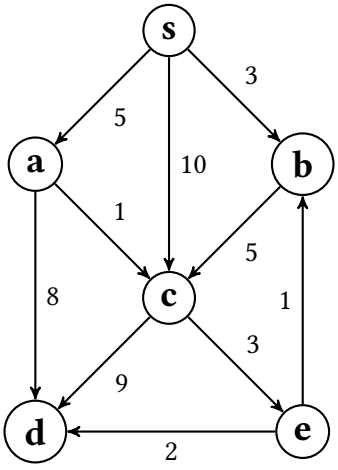
Algorithm 1 Dijkstra(G, s)

Input: A starting vertex s and a weighted graph G represented via $c(\cdot, \cdot)$. Specifically, for two vertices u, v the value $c(u, v)$ represents the cost of the edge from u to v (or ∞ if no such edge exists).
Operations:
$d[s] \leftarrow 0$ ▷ upper bounds on distances from s
$d[v] \leftarrow \infty$ for all $v \neq s$
$S \leftarrow \emptyset$ ▷ set of vertices with known distances
while $S \neq V$ **do**
  choose $v^* \in V \setminus S$ with minimum upper bound $d[v^*]$
  add v^* to S
  for each $v \in V \setminus S$ **do**
    for each $u \in S$ **do**
      if $c(u, v) < \infty$ **then**
        $d[v] \leftarrow \min\{d[v], d[u] + c(u, v)\}$

We remark that this version of Dijkstra's algorithm focuses on illustrating how the algorithm explores the graph and why it correctly computes all distances from s. You can use this version of Dijkstra's algorithm to solve this exercise.

In order to achieve the best possible running time, it is important to use an appropriate data structure for efficiently maintaining the upper bounds $d[v]$ with $v \in V \setminus S$ as you will see in the next lecture. In the other exercises/sheets and in the exam you should use the running time of the efficient version of the algorithm (and not the running time of the pseudocode described above).

Consider the following weighted directed graph:



(a) Execute the Dijkstra's algorithm described above by hand to find a shortest path from **s** to each vertex in the graph. After each iteration of the while-loop, write down:

1) $d[u]$ for all $u \in V$ (which are upper bounds on the distances from **s** to u computed so far),

2) the set S (which contains vertices for which the distance has been correctly computed so far),

3) and the predecessors for each vertex $u \in S \setminus \{s\}$. (A predecessor of a vertex $u \in S \setminus \{s\}$ is a vertex $v \in S$ which satisfies $d[u] = d[v] + c(v, u)$.)

(b) Change the weight of the edge $(\mathbf{a}, \mathbf{c})$ from 1 to -1 and execute Dijkstra's algorithm on the new graph. Does the algorithm work correctly (are all distances computed correctly) ? In case it breaks, where does it break?

(c) Now, additionally change the weight of the edge $(\mathbf{e}, \mathbf{b})$ from 1 to -6 (so edges $(\mathbf{a}, \mathbf{c})$ and $(\mathbf{e}, \mathbf{b})$ now have negative weights). Show that in this case the algorithm doesn't work correctly, i.e. there exists some $u \in V$ such that $d[u]$ is not equal to the minimum distance from **s** to u after the execution of the algorithm.

Exercise 10.5 *Strongly connected vertices* **(1 point)**.

Let $G = (V, E)$ be a directed graph with n vertices and m edges. We say two distinct vertices $v, w \in V$ are *strongly connected* if there exists both a directed path from v to w, and from w to v.

Describe an algorithm which finds a pair $v, w \in V$ of strongly connected vertices in G, or decides that no such pair exists. The runtime of your algorithm should be at most $O(n + m)$. You are provided with the number of vertices n, and the adjacency list Adj of G.

Hint: *Use DFS as a subroutine.*