



Algorithms & Data Structures

Exercise sheet 11

HS 22

The solutions for this sheet are submitted at the beginning of the exercise class on 12 December 2022.

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 11.1 *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

Input: a weighted graph, represented via $c(\cdot, \cdot)$. Specifically, for two vertices u, v the value $c(u, v)$ represents the cost of an edge from u to v (or ∞ if no such edge exists).

function DIJKSTRA(G, s)

$d[s] \leftarrow 0$

 ▷ 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

 update upper bounds for all $v \in V \setminus S$:

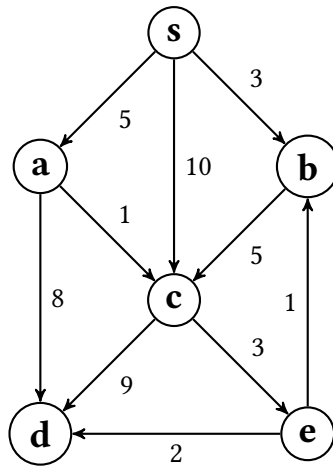
$d[v] \leftarrow \min_{\text{predecessor } u \in S \text{ of } v} d[u] + c(u, v)$

 (if v has no predecessors in S , this minimum is ∞)

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 saw in the lecture on December 1. 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 step (i.e. after each choice of v^*), write down:
 - 1) the upper bounds $d[u]$, for $u \in V$, between **s** and each vertex u computed so far,
 - 2) the set M of all vertices for which the minimal distance has been correctly computed so far,
 - 3) and the predecessor $p(u)$ for each vertex in M .
- b) Change the weight of the edge **(a, 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 **(e, b)** from 1 to -6 (so edges **(a, c)** and **(e, 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 a minimal distance from **s** to u after the execution of the algorithm.

Exercise 11.2 *Depth-First Search Revisited (1 point).*

In this exercise we examine the depth-first search in a graph $G = (V, E)$, printed here for convenience. For concreteness, you can assume that $V = \{1, \dots, n\}$ and that for $v \in V$ we have access to an adjacency list $adj[v]$.

Algorithm 2

Input: graph G , given as adj and $n \geq 1$.

Global variable: $marked[1 \dots n]$, initialized to $[False, False, \dots, False]$.

Global variable: T , initialized to $T \leftarrow 1$.

Global variable: $pre[1 \dots n]$.

▷ Pre-order number.

Global variable: $post[1 \dots n]$.

▷ Post-order number.

function $DFS(v)$

$marked[v] \leftarrow True$

$pre[v] \leftarrow T$

$T \leftarrow T + 1$

for each neighbor $w \in adj[v]$ **do**

if not $marked[w]$ **then**

$DFS(w)$

$post[v] \leftarrow T$

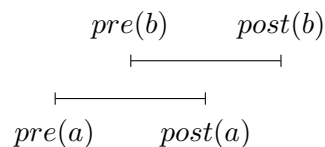
$T \leftarrow T + 1$

for $v \in \{1, \dots, n\}$ **do**

if not $marked[v]$ **then**

$DFS(v)$

- (a) Consider the graphical representation of the DFS order where a vertex v is represented as an interval $[pre(v), post(v)]$. Give a short argument why in directed or undirected graphs no two such intervals can intersect without one being fully contained in the other. Specifically, argue why the situation depicted in the figure below cannot happen.



- (b) Give a short argument why undirected graphs cannot have any cross edges.
- (c) Prove that a directed graph is acyclic (i.e., a DAG) if and only if it has no back edges. This was proven in the lecture, but the goal here is to explicitly write out the entire argument.
- Hint:** You need to prove both directions of the equivalence.
- Hint:** For the (\implies) direction, assume the opposite (there is a back edge), then simply find a cycle containing that back edge. If needed, you can use without proof the property that if the interval of a is contained within interval b , then there exists a simple path from b to a .
- Hint:** For the (\impliedby) direction, we need to prove the graph is a DAG (i.e., acyclic). It is sufficient to find a topological ordering such that all directed edges originate at vertices that are before their tail (according to the ordering). One specific order that works is the reverse post-order.

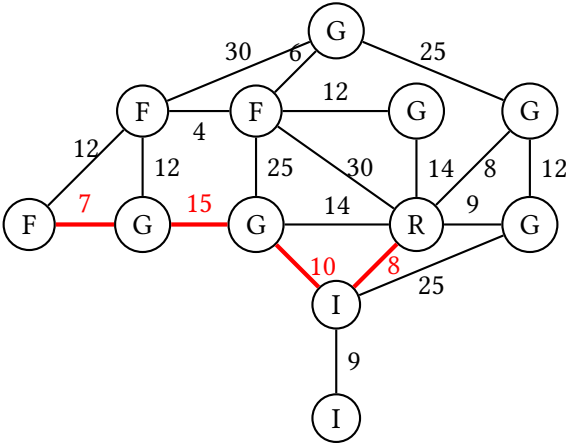
Exercise 11.3 Language Hiking (2 points).

Alice loves both hiking and learning new languages. Since she moved to Switzerland, she has always wanted to discover all four language regions of the country in a single hike – but she is not sure whether her week of vacation will be sufficient.

You are given a graph $G = (V, E)$ representing the towns of Switzerland. Each vertex V corresponds to a town, and there is an (undirected) edge $\{v_1, v_2\} \in E$ if and only if there exists a direct road going from town v_1 to town v_2 . Additionally, there is a function $w : E \rightarrow \mathbb{N}$ such that $w(e)$ corresponds to the number of hours needed to hike over road e , and a function $\ell : V \rightarrow \{G, F, I, R\}$ that maps each town to the language that is spoken there¹. For simplicity, we assume that only one language is spoken in each town.

Alice asks you to find an algorithm that returns the walking duration (in hours) of the shortest hike that goes through at least one town speaking each of the four languages.

For example, consider the following graph, where languages appear on vertices:



The shortest path satisfying the condition is marked in red. It goes through one R vertex, one I vertex, two G vertices and one F vertex. Your algorithm should return the cost of this path, i.e., 40.

- (a) Suppose we know the order of languages encountered in the shortest hike. It first goes from an R vertex to an I vertex, then immediately to a G vertex, and reaches an F vertex in the end, after going through zero, one or more additional G vertices. In other terms, the form of the path is RIGF or RIG...GF. In this case, describe an algorithm which finds the shortest path satisfying the condition, and explain its runtime complexity. Your algorithm must have complexity at most $O((|V| + |E|) \log |V|)$.

Hint: Consider the new vertex set $V' = V \times \{1, 2, 3, 4\} \cup \{v_s, v_d\}$, where v_s is a 'super source' and v_d a 'super destination' vertex.

- (b) Now we don't make the assumption in (a). Describe an algorithm which finds the shortest path satisfying the condition. Briefly explain your approach and the resulting runtime complexity. To obtain full points, your algorithm must have complexity at most $O((|V| + |E|) \log |V|)$.

Hint: Consider the new vertex set $V' = V \times \{0, 1\}^4 \cup \{v_s, v_d\}$, where v_s is a 'super source' and v_d a 'super destination' vertex.

¹G, F, I and R stand for German, French, Italian, and Romansh respectively.