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, David Steurer
Kasper Lindberg, Lucas Slot, Hongjie Chen, Manuel Wiedmer

25 November 2024

Algorithms & Data Structures — Exercise sheet 10 — HS 24

The solutions for this sheet are submitted on Moodle until 1 December 2024, 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 *Shortcutting closed Eulerian walks* **(1 point)**.

Let $G = (V, E)$ be the complete graph meaning we have an edge set E equal to all possible edges between distinct vertices. Furthermore, there is a weight function $w : E \to \mathbb{R}_{\geq 0}$ on the edges which satisfies a 'metric' property: for any distinct $u, v, w \in V$ we have $w(\{u, w\}) \leq w(\{u, v\}) + w(\{v, w\})$. Given a sequence of edges $C = (e_1, \ldots, e_k)$ which form a cycle in G, the weight of this cycle is defined as $w(C) = \sum_{i=1}^{k} w(e_i)$.

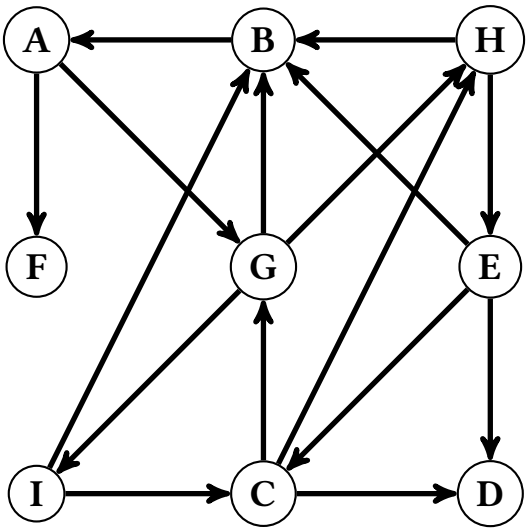
Now consider another graph $H = (V, F)$ which has the same vertices as G. Furthermore, we can create a weight function $w' : F \to \mathbb{R}_{\geq 0}$ compatible with w meaning for each $f \in F$, if f's endpoints are u and v then $w'(f) = w(\{u, v\})$.

Suppose that H is connected and has a closed Eulerian walk denoted by a sequence of edges $T = (e_1, \ldots, e_k)$. The weight of the walk is defined as $w'(T) = \sum_{i=1}^{k} w'(e_i)$. Describe an algorithm that takes a closed Eulerian walk T of H and produces a Hamiltonian cycle C in G with weight at most that of the closed Eulerian walk, meaning $w(C) \leq w'(T)$. Your algorithm should have runtime $O(|F|)$. Argue why your algorithm is correct and why it satisfies the runtime bound.

Exercise 10.2 *Breadth-first search* **(1 point)**.

Execute a breadth-first search (BFS) on the following graph $G = (V, E)$ starting at vertex A using the algorithm you have seen in the lecture, which updates a queue Q.

You should always enqueue the neighbours of a vertex in alphabetical order.



(a) Write down all operations that are executed on the queue Q during the BFS (in order), and write down the contents of Q after each operation is executed (in order). For example, the first operation is $\mathrm{enqueue}(A)$ and after this operation, $Q = [A]$. The second operation is $\mathrm{dequeue}(A)$, and after this operation, $Q = [\,]$.

(b) Give a *shortest path tree* for G (with root A).

(c) Let $S_k := \{v \in V : \mathrm{dist}(A, v) = k\}$. Write down S_k for $k = 0, 1, 2, 3, 4, 5$.

(d) An edge $e \in E$ is called *critical* if there is a vertex $v \in V$ to which the distance from A increases after removing e. Write down all critical edges of G.

(e) Write down the enter-number and leave-number of each vertex of the graph. (These numbers indicate the number of queue-operations that have been executed as a vertex enters (respectively, leaves) the queue. For example, $\mathrm{enter}(A) = 0$ and $\mathrm{leave}(A) = 1$.)

(f) Let $t_k := \min_{v \in V}\{\mathrm{leave}(v) : \mathrm{dist}(A, v) \geq k\}$. Write down t_k for $k = 0, 1, 2, 3, 4, 5$. (We define $\min \emptyset := \infty$.)

(g) Let $R_k := \{v \in V : t_k \leq \mathrm{leave}(v) < t_{k+1}\}$. Write down R_k for $k = 0, 1, 2, 3, 4, 5$.

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 *Number of minimal paths* **(1 point)**.

Let $G = (V, E)$ be an undirected graph with n vertices and m edges. Let $v, v' \in V$ be two distinct vertices and suppose that the distance between the two is k.

Describe an algorithm which counts the number of paths from v to v' of length k. 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. Argue why your algorithm is correct and why it satisfies the runtime bound.

Hint: *Modify BFS.*

Exercise 10.5 *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:

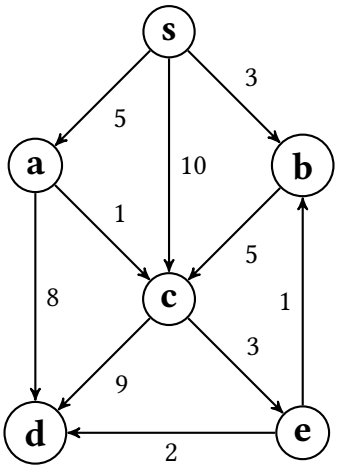
Algorithm 1 Dijkstra(G, s)

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.