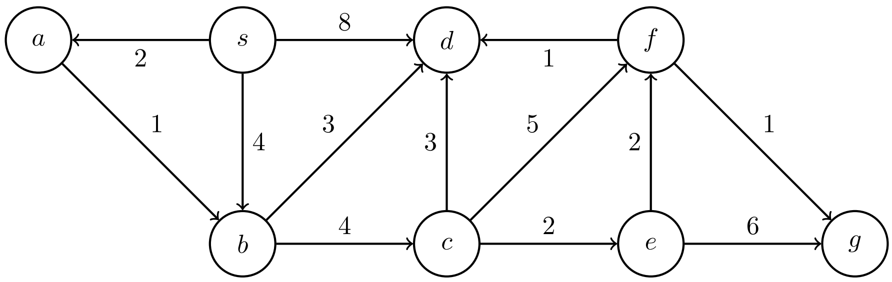
University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn

Algorithms and Data Structures
Summer Term 2021
Sample Solution Exercise Sheet 10

Exercise 1: Dijkstras' Algorithm

Execute Dijkstras' Algorithm on the following weighted, directed graph, starting at node s. Into the table further below, write the distances from each node to s that the algorithm stores in the priority queue after each iteration.



Initialization	s	a	b	c	d	e	f	g
$\delta(s,\cdot) =$	0	∞	∞	∞	∞	∞	∞	∞
1. Step $(u = s)$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
2. Step $(\boldsymbol{u =} \quad)$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
3. Step $(\boldsymbol{u =} \quad)$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
4. Step $(\boldsymbol{u =} \quad)$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
5. Step $(\boldsymbol{u =} \quad)$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
6. Step $(\boldsymbol{u =} \quad)$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
7. Step $(\boldsymbol{u =} \quad)$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
8. Step $(\boldsymbol{u =} \quad)$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								

Sample Solution

Initialisation	s	a	b	c	d	e	f	g
$\delta(s,\cdot) =$	0	∞	∞	∞	∞	∞	∞	∞
1. Step $(u = s)$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	4	∞	8	∞	∞	∞
2. Step $\boldsymbol{(u = a)}$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	∞	8	∞	∞	∞
3. Step $\boldsymbol{(u = b)}$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	∞	∞	∞
4. Step $\boldsymbol{(u = d)}$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	∞	∞	∞
5. Step $\boldsymbol{(u = c)}$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	9	12	∞
6. Step $\boldsymbol{(u = e)}$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	9	11	15
7. Step $\boldsymbol{(u = f)}$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	9	11	12
8. Step $\boldsymbol{(u = g)}$	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	9	11	12

Exercise 2: Currency Exchange

Consider n currencies $w_1, \dots, w_n$. The exchange rates are given in an $n \times n$-matrix A with entries a_{ij} ($i, j \in \{1, \dots, n\}$). Entry a_{ij} is the exchange rate from w_i to w_j, i.e., for one unit of w_i one gets a_{ij} units of w_j.
Given a currency w_{i_0}, we want to find out whether there is a sequence $i_0, i_1, \dots, i_k$ such that we make profit if we exchange one unit of w_{i_0} to w_{i_1}, then to w_{i_2} etc. until w_{i_k} and then back to w_{i_0}.

(a) Translate this problem to a graph problem. That is, define a graph and a property which the graph fulfills if and only if there is a sequence of currencies as described above.

(b) Give an algorithm that decides in $\mathcal{O}(n^3)$ time steps whether there is a sequence of currencies as described above. Explain the correctness and runtime.

Hint: $\log(a \cdot b) = \log a + \log b$.

Sample Solution

(a) We define a weighted graph $G = (V, E, w)$ with $V = \{1, \dots, n\}$, $E = V^2$ (i.e., the graph is directed and complete) and $w(i, j) = a_{ij}$ (i.e., A is the adjacency matrix). A sequence of currencies as described exists if and only if there is a cycle $(i_0, i_1, \dots, i_k, i_0)$ such that

$$\prod_{j=0}^{k-1} w(i_j, i_{j+1}) \cdot w(i_k, i_0) > 1 \ . \tag{1}$$

(b) In the adjacency matrix, we replace a_{ij} by $-\log a_{ij}$. That is, we define a graph $G = (V, E, w')$ with V and E as before and $w'(i, j) = -\log w(i, j)$. We run Bellman-Ford on G' with source i_0.

This algorithm checks if G' contains a negative cycle, i.e., nodes $i_0, \dots, i_k$ with

$$\begin{aligned}
& \sum_{j=0}^{k-1} w'(i_j, i_{j+1}) + w'(i_k, i_0) < 0 \\
\iff \quad & \sum_{j=0}^{k-1} - \log w(i_j, i_{j+1}) - \log w(i_k, i_0) < 0 \\
\iff \quad & \sum_{j=0}^{k-1} \log w(i_j, i_{j+1}) + \log w(i_k, i_0) > 0 \\
\iff \quad & \log \left(\prod_{j=0}^{k-1} w(i_j, i_{j+1}) \cdot w(i_k, i_0) \right) > 0 \\
\iff \quad & \prod_{j=0}^{k-1} w(i_j, i_{j+1}) \cdot w(i_k, i_0) > 1 \ .
\end{aligned}$$

So the algorithm checks property (1) from part (a). The runtime of Bellman-Ford is $\mathcal{O}(|V| \cdot |E|)$. With $|V| = n$ and $|E| = n^2$ we obtain a runtime of $\mathcal{O}(n^3)$.