Eidgenössische
Technische Hochschule
Zürich

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

Departement of Computer Science
Markus Püschel, David Steurer
Gleb Novikov, Tommaso d'Orsi, Ulysse Schaller, Rajai Nasser

6. December 2021

Algorithms & Data Structures Exercise sheet 11 HS 21

Exercise Class (Room & TA): ______________________

Submitted by: ______________________

Peer Feedback by: ______________________

Points: ____

Submission: On Monday, 13. December 2021, hand in your solution to your TA *before* the exercise class starts. Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

Exercise 11.1 *Shortest paths by hand* **(2 points)**.

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:

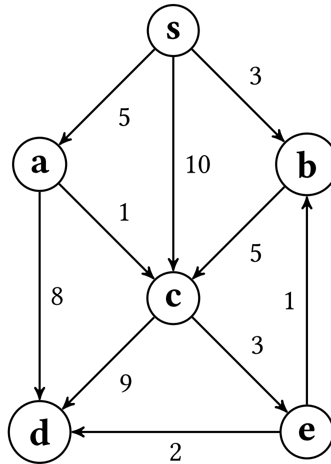
---

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 2. 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 node 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 node u computed so far,

   2) the set M of all nodes for which the minimal distance has been correctly computed so far,

   3) and the predecessor $p(u)$ for each node in M.

   Solution:

   When we choose **s**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = d[\mathbf{b}] = d[\mathbf{c}] = d[\mathbf{d}] = d[\mathbf{e}] = \infty$, $M = \{\mathbf{s}\}$, there is no $p(\mathbf{s})$.

   When we choose **b**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = 5$, $d[\mathbf{b}] = 3$, $d[\mathbf{c}] = 10$, $d[\mathbf{d}] = d[\mathbf{e}] = \infty$, $M = \{\mathbf{s}, \mathbf{a}, \mathbf{b}\}$, there is no $p(\mathbf{s})$, $p(\mathbf{a}) = p(\mathbf{b}) = \mathbf{s}$.

   When we choose **a**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = 5$, $d[\mathbf{b}] = 3$, $d[\mathbf{c}] = 8$, $d[\mathbf{d}] = d[\mathbf{e}] = \infty$, $M = \{\mathbf{s}, \mathbf{a}, \mathbf{b}\}$, there is no $p(\mathbf{s})$, $p(\mathbf{a}) = p(\mathbf{b}) = \mathbf{s}$.

   When we choose **c**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = 5$, $d[\mathbf{b}] = 3$, $d[\mathbf{c}] = 6$, $d[\mathbf{d}] = 13$, $d[\mathbf{e}] = \infty$, $M = \{\mathbf{s}, \mathbf{a}, \mathbf{b}, \mathbf{c}\}$, there is no $p(\mathbf{s})$, $p(\mathbf{a}) = p(\mathbf{b}) = \mathbf{s}, p(\mathbf{c}) = \mathbf{a}$.

   When we choose **e**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = 5$, $d[\mathbf{b}] = 3$, $d[\mathbf{c}] = 6$, $d[\mathbf{d}] = 13$, $d[\mathbf{e}] = 9$, $M = \{\mathbf{s}, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{e}\}$, there is no $p(\mathbf{s})$, $p(\mathbf{a}) = p(\mathbf{b}) = \mathbf{s}, p(\mathbf{c}) = \mathbf{a}, p(\mathbf{e}) = \mathbf{c}$.

   When we choose **d**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = 5$, $d[\mathbf{b}] = 3$, $d[\mathbf{c}] = 6$, $d[\mathbf{d}] = 11$, $d[\mathbf{e}] = 9$, $M = \{\mathbf{s}, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}\}$, there is no $p(\mathbf{s})$, $p(\mathbf{a}) = p(\mathbf{b}) = \mathbf{s}, p(\mathbf{c}) = \mathbf{a}, p(\mathbf{d}) = \mathbf{e}, p(\mathbf{e}) = \mathbf{c}$.

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?

   Solution: The algorithm works correctly.

   When we choose **s**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = d[\mathbf{b}] = d[\mathbf{c}] = d[\mathbf{d}] = d[\mathbf{e}] = \infty$.

   When we choose **b**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = 5$, $d[\mathbf{b}] = 3$, $d[\mathbf{c}] = 10$, $d[\mathbf{d}] = d[\mathbf{e}] = \infty$.

   When we choose **a**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = 5$, $d[\mathbf{b}] = 3$, $d[\mathbf{c}] = 8$, $d[\mathbf{d}] = d[\mathbf{e}] = \infty$.

   When we choose **c**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = 5$, $d[\mathbf{b}] = 3$, $d[\mathbf{c}] = 4$, $d[\mathbf{d}] = 13$, $d[\mathbf{e}] = \infty$.

   When we choose **e**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = 5$, $d[\mathbf{b}] = 3$, $d[\mathbf{c}] = 4$, $d[\mathbf{d}] = 13$, $d[\mathbf{e}] = 7$.

   When we choose **d**: $d[\mathbf{s}] = 0$, $d[\mathbf{a}] = 5$, $d[\mathbf{b}] = 3$, $d[\mathbf{c}] = 4$, $d[\mathbf{d}] = 9$, $d[\mathbf{e}] = 7$.

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 a minimal distance from $\mathbf{s}$ to u after the execution of the algorithm.

   Solution: The algorithm doesn't work correctly, for example, the distance from **s** to **b** is 1 (via the path **s**-**a**-**c**-**e**-**b**), but the algorithm computes exactly the same values of $d[\cdot]$ as in part b), so $d[\mathbf{b}] = 3$.

Exercise 11.2 *Arbitrage.*

When trading currencies, *arbitrage* means to exploit price differences in order to profit by exchanging currencies multiple times. For example, on June 2nd, 2009, 1 US Dollar could be exchanged for 95.729 Yen, 1 Yen for 0.00638 Pound sterling, and 1 Pound sterling for 1.65133 US Dollars. If a trader exchanged 1 US Dollar for Yen, exchanged the obtained amount for Pound sterling and finally exchanged this amount back to US Dollars, he would have obtained $95.729 \cdot 0.00638 \cdot 1.65133 \approx 1.0086$ US Dollars, corresponding to a gain of 0.86%.

a) You are given n currencies $\{1, \dots, n\}$ and an $(n \times n)$ exchange rate matrix R with positive rational number entries. For two currencies $i, j \in \{1, \dots, n\}$ one unit of currency i can be exchanged for $R(i, j) > 0$ units of currency j. The goal is to decide whether an arbitrage opportunity exists, i.e., if there exists a sequence of k different currencies $W_1, \dots, W_k \in \{1, \dots, n\}$ such that $R(W_1, W_2) \cdot R(W_2, W_3) \cdots R(W_{k-1}, W_k) \cdot R(W_k, W_1) > 1$ holds.

   Model the above problem as a graph problem. Show how the input can be transformed into a directed, weighted graph $G = (V, E, w)$ that contains a cycle with negative weight *if and only if* an arbitrage activity is possible. Justify why G contains a negative cycle if and only if an arbitrage opportunity exists.

   Hint: *Using logarithms might be beneficial because of the property* $\ln(a \cdot b) = \ln(a) + \ln(b)$.

   Solution: We create a complete graph $G = (V, E)$ with the vertices $V = \{1, \dots, n\}$. An edge $(i, j) \in E$ gets the weight $w(i, j) = -\log R(i, j)$. Then suppose an arbitrage opportunity with the sequence of currencies $W_1, \dots, W_k$ is possible. Then it must be the case that

$$
\begin{aligned}
& R(W_1, W_2) \cdot R(W_2, W_3) \cdot \ldots \cdot R(W_{k-1}, W_k) R(W_k, W_1) > 1 \\
\Leftrightarrow\; & \log \left(R(W_1, W_2) \cdot R(W_2, W_3) \cdot \ldots \cdot R(W_{k-1}, W_k) R(W_k, W_1) \right) > 0 \\
\Leftrightarrow\; & \log R(W_1, W_2) + \log R(W_2, W_3) + \ldots + \log R(W_{k-1}, W_k) + \log R(W_k, W_1) > 0 \\
\Leftrightarrow\; & -\log R(W_1, W_2) - \log R(W_2, W_3) - \ldots - \log R(W_{k-1}, W_k) - \log R(W_k, W_1) < 0 \\
\Leftrightarrow\; & w(W_1, W_2) + w(W_2, W_3) + \ldots + w(W_{k-1}, W_k) + w(W_k, W_1) < 0
\end{aligned}
$$

   consequently G contains a cycle of negative weight. Because we only used equivalence transformations, the argument applies in both directions.

b) The Bellman-Ford algorithm can be used to find out whether a graph contains negative cycles. After ℓ iterations of the Bellman-Ford loop $d[v]$ is equal to $d(s, v)^{\leq \ell}$, i.e., the minimum weight of a s-v walk with at most ℓ edges. A graph contains a negative cycle that can be reached from s if and only if there exists a vertex v such that $d(s, v)^{\leq n-1} \neq d(s, v)^{\leq n}$, where n is the number of vertices.

   Use the previous part of this exercise to design an algorithm that decides if an arbitrage opportunity exists. What is the best running time you can get (in terms of n)?

Solution: Since the graph is complete, any negatively-weighted cycle can be reached from any vertex, we can run Bellman-Ford to detect a cycle starting from any vertex in G. There are $|V| = n$ vertices and $|E| = n(n-1) \in \Theta(n^2)$ edges. The algorithm therefore has a running time of $\Theta(|V||E|) = \Theta(n^3)$.

Exercise 11.3 *Single-source shortest allowable paths* **(1 point)**.

Consider a weighted directed graph $G = (V, E)$ that is given in the adjacency-list representation. The graph G has n vertices and m edges. The weights $w : E \to \mathbb{R}$ of the edges are not necessarily positive, but the graph does not contain any negative weight cycle. Every vertex is colored either red or green, so that the vertex set is partitioned as $V = V_{red} \cup V_{green}$. A path is said to be *allowable* if it contains at most one red vertex.

We are given a source $s \in V$ and would like to find the weights of the *shortest allowable paths* from s to all vertices in V. That is, we are interested in $\left(\delta_a(s,v)\right)_{v \in V}$, where $\delta_a(s,v)$ is the weight of the shortest allowable path from s to v. If v is not reachable from s by an allowable path, then $\delta_a(s,v) = \infty$.

Describe an algorithm that can efficiently compute $\left(\delta_a(s,v)\right)_{v \in V}$. In order to get full points, the runtime of your algorithm should be $O(mn)$.

Hint: *Construct a weighted directed graph G' with $2n$ vertices and apply an algorithm that you learned in class on it.*

Solution: We construct the weighted graph $G' = (V', E')$ with the weight function $w' : E' \to \mathbb{R}$ as follows

- $V' = V^{(0)} \cup V^{(1)}$, where $V^{(0)}$ and $V^{(1)}$ are two disjoint copies of V. More precisely, for every $v \in V$, there is exactly one vertex $v^{(0)} \in V^{(0)}$ and another vertex $v^{(1)} \in V^{(1)}$. Clearly,
$$n' = |V'| = |V^{(0)}| + |V^{(1)}| = 2|V| = 2n.$$
- For every edge $(u, v) \in E$ in the original graph G, do the following:
  - If $u \in V_{green}$ and $v \in V_{green}$, add $(u^{(0)}, v^{(0)})$ and $(u^{(1)}, v^{(1)})$ to E' and set $w'(u^{(0)}, v^{(0)}) = w'(u^{(1)}, v^{(1)}) = w(u, v)$.
  - If $u \in V_{green}$ and $v \in V_{red}$, add $(u^{(0)}, v^{(1)})$ to E' and set $w'(u^{(0)}, v^{(1)}) = w(u, v)$.
  - If $u \in V_{red}$ and $v \in V_{green}$, add $(u^{(1)}, v^{(1)})$ to E' and set $w'(u^{(0)}, v^{(1)}) = w(u, v)$.
  - If $u \in V_{red}$ and $v \in V_{red}$, do not add any edge between $\{u^{(0)}, u^{(1)}\}$ and $\{v^{(0)}, v^{(1)}\}$.

  It is easy to see that $m' = |E'| \le 2|E| = 2m$: For every edge in E, we add at most two edges in E'.

The construction of G' takes $O(n' + m') = O(n + m)$ time. After constructing G', the algorithm proceeds as follows:

- If $s \in V_{green}$, we apply the Bellman-Ford algorithm on G' with $s^{(0)}$ being the source.
- If $s \in V_{red}$, we apply the Bellman-Ford algorithm on G' with $s^{(1)}$ being the source.

This will take $O(n'm') = O(nm)$ time.

Let $(d_{v'})_{v' \in V'}$ be the weights of the shortest paths that we obtain from applying the above procedure. For every $v \in V$, the shortest allowable path from s to v can then be computed as
$$\delta_a(s,v) = \min\left\{d_{v^{(0)}}, d_{v^{(1)}}\right\}.$$

This takes $O(n') = O(n)$ time.

Therefore, the total runtime of the algorithm is $O(nm)$.

For completeness, let us now explain why this is a correct algorithm (you did not have to do it since we had not explicitly asked for it in the task). The correctness of the described algorithm follows from the following observation:

- If $s \in V_{green}$, then for every $v \in V$, there is an allowable path from s to v in G with exactly $b \in \{0, 1\}$ red vertices if and only if there is a path from $s^{(0)}$ to $v^{(b)}$ in G'. In fact, there is a one-to-one correspondence between these paths and their weights are equal.
- If $s \in V_{red}$, then for every $v \in V$, there is an allowable path from s to v in G if and only if there is a path from $s^{(1)}$ to $v^{(1)}$ in G'. In fact, there is a one-to-one correspondence between these paths and their weights are equal.

Exercise 11.4 *Traveling in Examistan* ***(This exercise is from the August 2020 exam).***

Assume that there are n towns $T_1, \dots, T_n$ in the country Examistan. For each pair of distinct towns T_i and T_j, there is exactly one road from T_i to T_j. All of the roads in Examistan are one-way. This implies that there is always a road from T_i to T_j and another road from T_j to T_i. Each road has a nonnegative integer cost that you need to pay to use this road.

For simplicity you can assume that each town T_i is represented by its index i.

a) Model the n towns, the roads and their costs as a directed weighted graph: give a precise description of the vertices, edges and the weights of the edges of the graph $G = (V, E, w)$ involved (if possible, in words and not formal). What are $|V|$ and $|E|$ in terms of n?

   Solution: The towns are modeled as the vertices $V = \{1, \dots, n\}$ of the graph G. The roads are modeled as directed edges $E = \{(i, j) \mid i \neq j, i, j = 1, \dots, n\}$. The costs that you need to pay to use the roads are modeled as the weights w of the respective edges.

   The number of vertices is thus $|V| = n$ and the number of edges $|E| = n^2 - n$, since n^2 is the number of possible ordered pairs (i, j) and we have to subtract the n self-edges represented by (i, i) as they are not part of our graph.

   Alternative way to get the number of edges: you choose 2 out of n to get the number of unordered sets $\{i, j\}$ with $i \neq j$, resulting in $\binom{n}{2} = \frac{1}{2}(n-1)n$. But we care for the different directions so we have to multiply this number by 2 (for (i, j) and (j, i)) resulting in $|E| = n^2 - n$.

In the following subtask b), you can assume that the directed graph in a) is represented by a data structure that allows you to traverse the direct successors and direct predecessors of a vertex u in time $\mathcal{O}(\deg_+(u))$ and $\mathcal{O}(\deg_-(u))$ respectively, where $\deg_-(u)$ is the in-degree of vertex u and $\deg_+(u)$ is the out-degree of vertex u.

b) Due to the epidemiological situation in Examistan, the authorities decided to reduce the number of trips between different towns. Now the only way to get from one town to another is to use the roads. Moreover, if you want to travel from town T_i to the other town T_j, you must visit a test center during your trip (in T_i or T_j or elsewhere with a detour). Since test centers are expensive, there are only $k < n$ of them, and they are located only in the first k towns $T_1, \dots, T_k$ (i.e., one test center in each of these towns). To compensate for this, the authorities make the roads between all different T_i and T_j among $T_1, \dots, T_k$ free of charge (i.e. their cost is now 0).

Provide an as efficient as possible algorithm that takes as input a graph G from task a) and a number k, and outputs a table C such that $C[i][j]$ is the minimal total cost of roads that one should use to get from T_i to T_j with visiting a test center.

What is the running time of your algorithm in concise Θ-notation in terms of n and k ? Justify your answer.

Solution: Since the paths between the test centers are now free of charge (i.e $w_{i,j} = 0$), we can consider the test centers as a super-vertex (i.e. all the test centers merged to one vertex and connected to each other vertex with the minimal cost of all costs from the test centers). This graph has $n-(k-1)$ vertices. Then we run Dijsktra algorithm one time to get the shortest paths from the super-vertex to each vertex in an array $C_1[j]$. Then we reverse the edges and run Dijkstra another time to get the shortest paths from each vertex to the super-vertex in an array $C_2[i]$. The shortest path from vertex i to vertex j passing through one test center is then $C[i][j] = C_2[i] + C_1[j]$.

Creating the super-vertex can be done in time $\Theta(n \cdot k)$. The runtime for Dijkstra algorithm implemented with a Fibonacci heap on the modified graph with super-vertex is $\Theta((n-k)\log(n-k) + (n-k)^2 - (n-k))$, which does not change if we run it two times. Filling the final table is $\Theta((n-k)^2)$, so the overall runtime is $\Theta(nk + (n-k)^2) = \Theta(n^2)$.