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
François Hublet, Goran Zuzic, Tommaso d'Orsi, Jingqiu Ding

5 December 2022

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:

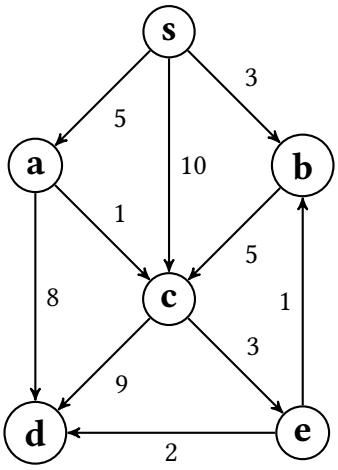
Algorithm 1

Input: a weighted graph, represeted 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.

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 **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 *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 $V = \{1, \ldots, n\}$ and that for $v \in V$ we have access to an adjacency list $adj[v]$.

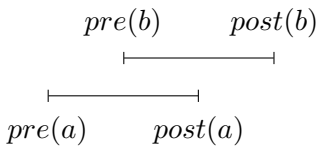
Algorithm 2

```
Input: graph G, given as adj and n ≥ 1.
Global variable: marked[1 ... n], initialized to [False, False, ..., False].
Global variable: T, initialized to T ← 1.
Global variable: pre[1 ... n].                    ▷ Pre-order number.
Global variable: post[1 ... n].                   ▷ Post-order number.

function DFS(v)
    marked[v] ← True
    pre[v] ← T
    T ← T + 1
    for each neighbor w ∈ adj[v] do
        if not marked[w] then
            DFS(w)
    post[v] ← T
    T ← T + 1

for v ∈ {1, ..., 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.



Solution:

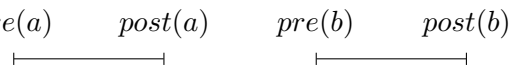
Assume, for the sake of contradiction that the pre/post relations can be achieved as in the figure. Hence, the DFS function is called on vertex a before being called on b. However, the figure stipulates that b starts before a completes. Now, since recursive calls stack (i.e., a parent call can only finish after all of its children's calls finish), it must be the case that b completes before a completes. Hence, interval b would be completely contained within interval a, which is contradicting the figure.

Grading: To get points, you need to mention(or imply) the observation that recursive calls stack.

(b) Give a short argument why undirected graphs cannot have any cross edges.

Solution:

A crossing edge is an edge between vertices a and b, where the intervals corresponding to those vertices are disjoint (see the figure).



Suppose this was the case. Then, in the recursive call corresponding to vertex a, we would discover an unmarked neighbor b. Hence, the call to b would necessarily be a child of a, and $\{a, b\}$ would be a back/forward edge. This completes the argument.

Grading: To get points, you need to recall the definition of crossing edge; and observe that the call to b would necessarily be a child of a, which contradicts the definition.

(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.*

Solution:

Direction ($\implies$). Assume for the sake of contradicting that there is a back edge $a \to b$. In other words, the interval a is nested inside interval b. Hence, by DFS properties, there exists a simple path p from b to a. Hence, p and $a \to b$ form a (directed) cycle. This contradiction completes the ($\implies$) direction.

Direction ($\impliedby$). It is sufficient to find a function $\pi : V \to \{1, \ldots, 2n\}$ (called the topological ordering) such that for every directed edge $a \to b$ we have $\pi(a) > \pi(b)$ (it can be easily shown that any graph that is consistent with π must be a DAG). Consider $\pi := post$, i.e., the post-order and consider an edge $a \to b$. As covered in the lecture, there are 4 possible relations between the intervals of a and b: (1) the intervals are disjoint and a is before b, (2) the intervals are disjoint and b is before a (3) a is nested within b, or (4) b is nested within a. Option (1) is impossible as then the call to a would trigger a nested call to b. In option (2), it holds that $post(a) = \pi(a) > \pi(b) = post(b)$. Option (3) is impossible since then $a \to b$ is a back edge and we assumed those don't exist. In option (4), it holds that $post(a) = \pi(a) > \pi(b) = post(b)$. Since all options satisfy $\pi(a) > \pi(b)$, we conclude that the π is a topological ordering, hence G is a DAG. This completes the ($\impliedby$) direction.

Grading: For direction ($\implies$), you need to observe that there is a path p from b to a which leads to

cycle with backedge. For direction ($\Longleftarrow$), you need to list all 4 different cases, and argue that for possible cases, there is a topological ordering. You also need to mention that existence of topological ordering implies DAG.

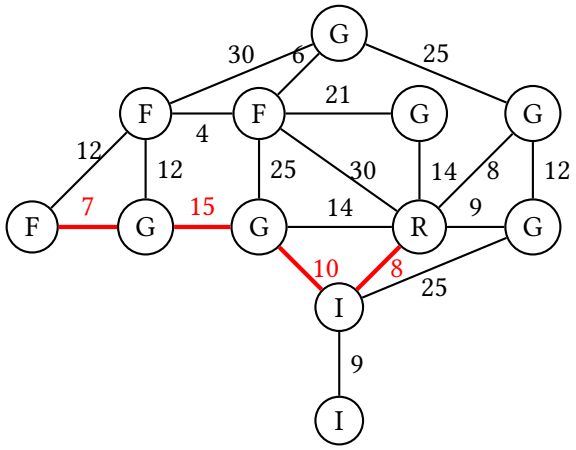
Exercise 11.3 *Language Hiking* **(2 points)**.

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 $w : E \to \mathbb{N}$ such that $w(e)$ corresponds to the number of hours needed to hike over road e, and a function $\ell : V \to \{\mathrm{G}, \mathrm{F}, \mathrm{I}, \mathrm{R}\}$ that maps each town to the language that is spoken there[1]. 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.*

Solution:

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

Consider the vertex set V' above, as well as the following edge set E' and weight function w':

$$\begin{aligned} E' = &\{\{v_s, (v,1)\} \mid \{u,v\} \in E, \ell(v) = \mathrm{R}\} \\ &\cup \{\{(u,1),(v,2)\} \mid \{u,v\} \in E, \ell(v) = \mathrm{I}\} \\ &\cup \{\{(u,2),(v,3)\} \mid \{u,v\} \in E, \ell(v) = \mathrm{G}\} \\ &\cup \{\{(u,3),(v,3)\} \mid \{u,v\} \in E, \ell(v) = \mathrm{G}\} \\ &\cup \{\{(u,3),(v,4)\} \mid \{u,v\} \in E, \ell(v) = \mathrm{F}\} \\ &\cup \{\{(v,4), v_d\} \mid v \in V\} \end{aligned}$$

$$w'(\{u',v'\}) = \begin{cases} 0 & \text{if } u' = v_s \text{ or } v' = v_d \\ w(\{u,v\}) & \text{if } u' = (u,i) \text{ and } v' = (v,j) \end{cases}$$

For each new vertex $(v,i) \in V'$, the first component $v \in V$ is a vertex in the original graph, while i is a counter which measures the progress over the path: if $i = 1$, only an R town has been visited; if $i = 2$, an R and an I town have been visited; if $i = 3$, an R, and I and at least one (or more) G towns have been visited; if $i = 4$, an R, an I, one or more G, and an F town have been visited. The weight of this edge remains the same as before. As an arbitrary number of G towns can be visited, we have transitions $(u,3) \to (v,3)$ (G to G) as well as $(u,3) \to (v,4)$ (G to F); since this is not the case for R, I, and F, we have only transitions $v_s \to (u,1)$, $(u,1) \to (v,2)$, and $(u,4) \to v_e$.

Moreover, a global source vertex v_s is connected to all R vertices. This corresponds to the choice of the first vertex (where Alice will start hiking). Similarly, a global destination vertex v_d is connected to all vertices with $i = 4$ with edges of weight 0, corresponding to the choice of the last vertex.

The length of the shortest path that follows the given pattern is exactly the length of the shortest path between v_s and v_d in $G' = (V', E')$ with weights w'. Since all weights are nonnegative, we can use Dijkstra's algorithm to find this shortest path.

The complexity of Dijkstra's algorithm is $O((|V'| + |E'|)\log(|V'|))$. Here, we have

$$\begin{aligned} |V'| &= |V| \cdot 4 + 2 \leq O(|V|) \\ |E'| &\leq |V| + |V| + |E| \cdot 2 \leq O(|V| + |E|), \end{aligned}$$

yielding $O((|V| + (|V| + |E|))\log(|V|)) = O((|V| + |E|)\log(|V|))$. Constructing the graph adds a cost $O(|V| + |E|)$ and extracting the result a $O(1)$. We obtain a total runtime in $O((|V| + |E|)\log(|V|))$.

Grading: To get points, you need to 1. construct the correct graph 2. argue that it corresponds to a shortest path problem in a nonnegative weight graph. 3. Use Dijkstra's algorithm and gives the right complexity.

(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.*

Solution:

Consider the vertex set V' above, as well as the following edge set E' and weight function w':

$$\begin{aligned}
E' = {} & \{\{v_s, (v, (1,0,0,0))\} \mid v \in V, \ell(v) = \mathrm{G}, \} \\
& \{\{v_s, (v, (0,1,0,0))\} \mid v \in V, \ell(v) = \mathrm{F}, \} \\
& \{\{v_s, (v, (0,0,1,0))\} \mid v \in V, \ell(v) = \mathrm{I}, \} \\
& \{\{v_s, (v, (0,0,0,1))\} \mid v \in V, \ell(v) = \mathrm{R}, \} \\
& \cup \{\{(v, (1,1,1,1)), v_d\} \mid v \in V\} \\
& \cup \{\{(u, (g,f,i,r)), (v, (1,f,i,r))\} \mid (g,f,i,r) \in \{0,1\}^4, \{u,v\} \in E, \ell(v) = \mathrm{G}\} \\
& \cup \{\{(u, (g,f,i,r)), (v, (g,1,i,r))\} \mid (g,f,i,r) \in \{0,1\}^4, \{u,v\} \in E, \ell(v) = \mathrm{F}\} \\
& \cup \{\{(u, (g,f,i,r)), (v, (g,f,1,r))\} \mid (g,f,i,r) \in \{0,1\}^4, \{u,v\} \in E, \ell(v) = \mathrm{I}\} \\
& \cup \{\{(u, (g,f,i,r)), (v, (g,f,i,1))\} \mid (g,f,i,r) \in \{0,1\}^4, \{u,v\} \in E, \ell(v) = \mathrm{R}\}
\end{aligned}$$

$$w'(\{u', v'\}) = \begin{cases} 0 & \text{if } u' = v_s \text{ or } v' = v_d \\ w(\{u,v\}) & \text{if } u' = (u, (g,f,i,r)) \text{ and } v' = (v, (g,f,i,r)) \end{cases}$$

For each new vertex $(v, (g,f,i,r)) \in V'$, the first component $v \in V$ is a vertex in the original graph, while g, f, i, and r are four Boolean variables that keep trace of whether a town with language G, F, I, or R has been visited already. Every edge $\{u,v\} \in E$ is replaced by a set of edges $\{(u, (g,f,i,r)), (v, (g',f',i',r'))\} \subseteq E'$ where the Boolean corresponding to language $\ell(v)$ is set to 1 and other Booleans are kept unchanged. The weight of this edge remains the same as before.

Moreover, a global source vertex v_s is connected to all vertices with $(0,0,0,0)$ Booleans with edges of weight 0. This corresponds to the choice of the first vertex (where Alice will start hiking). Similarly, a global destination vertex v_d is connected to all vertices with $(1,1,1,1)$ Booleans with edges of weight 0, corresponding to the choice of the last vertex.

The length of the shortest path that goes through all language regions is exactly the length of the shortest path between v_s and v_d in $G' = (V', E')$ with weights w'. Since all weights are nonnegative, we can use Dijkstra's algorithm to find this shortest path.

The complexity of Dijkstra's algorithm is $O((|V'| + |E'|) \log(|V'|))$. Here, we have

$$\begin{aligned}
|V'| &= |V| \cdot 2^4 + 2 \leq O(|V|) \\
|E'| &= |V| + |V| + |E| \cdot 2^4 \leq O(|V| + |E|),
\end{aligned}$$

yielding $O((|V| + (|V| + |E|)) \log(|V|)) = O((|V| + |E|) \log(|V|))$. Constructing the graph adds a cost $O(|V| + |E|)$ and extracting the result a $O(1)$. We obtain a total runtime in $O((|V| + |E|) \log(|V|))$.

Grading: To get points, you need to 1. construct the correct graph 2. argue that it corresponds to a shortest path problem in a nonnegative weight graph. 3. Use Dijkstra's algorithm and gives the right complexity.