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

2 December 2024

Algorithms & Data Structures — Exercise sheet 11 — HS 24

The solutions for this sheet are submitted on Moodle until 8 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.

The solutions are intended to help you understand how to solve the exercises and are thus more detailed than what would be expected at the exam. All parts that contain explanation that you would not need to include in an exam are in grey.

Exercise 11.1 *Multiplicatively shortest path* **(1 point)**.

Let $G = (V, E)$ be a directed graph, with weights $w_e \geq 1$ for $e \in E$. The *multiplicative weight* of a path $P \subseteq E$ is

$$\mathrm{mw}(P) := \prod_{e \in P} w_e \quad \text{(i.e., the } \textit{product} \text{ of the weights of the edges in } P).$$

Given $s, t \in V$, give an algorithm that computes the multiplicative weight of a shortest path from s to t (i.e, a path P from s to t for which $\mathrm{mw}(P)$ is minimized). You may assume that there is at least one path from s to t.

The runtime of your algorithm should be at most $O((n + m) \cdot \log n)$. You should argue briefly why your algorithm is correct and why it achieves the desired runtime.

Hint: You may use any algorithm you have seen in the lecture as a subroutine and use any runtime bounds you have seen for them without proof.

Hint: For all $a, b > 0$, we have $\log(a \cdot b) = \log(a) + \log(b)$.

Solution:

We define new weights $\widehat{w}_e := \log(w_e)$ for $e \in E$. Note that, since $w_e \geq 1$, we have $\widehat{w}_e \geq 0$ for all $e \in E$. Therefore, we can use Dijkstra's algorithm to find a shortest path from s to t in G (with weights $\widehat{w}$) in time $O((n + m) \log(n))$. Say this path has length ℓ. We then return 2^ℓ.

It remains to see why this algorithm is correct. For this, let $P \subseteq E$ be a path in G. Then,

$$\widehat{w}(P) := \sum_{e \in P} \widehat{w}_e = \sum_{e \in P} \log(w_e) = \log\left(\prod_{e \in P} w_e\right) = \log(\mathrm{mw}(P)),$$

and so $2^{\widehat{w}(P)} = \mathrm{mw}(P)$. Since 2^x is a monotonically increasing function in x (i.e., $2^a \geq 2^b$ if and only if $a \geq b$), it follows that a shortest path w.r.t. $\widehat{w}$ is a path of lowest multiplicative weight. Thus, the algorithm above is correct.

Guidelines for correction:

Two elements are important for this exercise:

- The observation that the weights $\widehat{w}$ are nonnegative;
- The observation that a shortest path w.r.t. $\widehat{w}$ is a path of lowest multiplicative weight;

Assuming a correct description of the algorithm, award 1 point if both elements are present and award 0.5 points if at least one is present.

Exercise 11.2 *Rotating weights* **(1 point)**.

Let $G = (V, E)$ be a complete graph with n vertices. Enumerate the vertices as $v_0, \ldots, v_{n-1}$. Let $c_0 : E \to \mathbb{R}$ be an edge weight function. When we are walking on this graph, the edge weights 'rotate', meaning at step t along a walk, we will have the edge weight function $c_t : E \to \mathbb{R}$ where $c_t(\{v_i, v_j\}) = c_0(\{v_{i+t}, v_{j+t}\})$ and $i + t$, $j + t$ are taken modulo n. Formally, given a walk $(v_{i_0}, v_{i_1}, ..., v_{i_k}, v_{i_{k+1}})$, its weight is $c_0(\{v_{i_0}, v_{i_1}\}) + c_1(\{v_{i_1}, v_{i_2}\}) + \cdots + c_k(\{v_{i_k}, v_{i_{k+1}}\})$

Suppose that there is no way to construct a cycle which has negative weight. Describe an algorithm that finds a shortest walk from v_0 to v_1 in time $O(n^5)$. Prove that your algorithm is correct, and achieves the desired runtime.

Remark. Recall that given two positive integers a and b, a modulo b is the remainder of the division of a by b. For example, assuming $n > 1$, we have $(n + 1)$ modulo n equal to 1.

Solution:

To solve this problem, we create a new directed graph. This graph has n layers, one for each 'rotation' state and so has vertex set equal to $V' = V \times \{0, 1, \ldots, n-1\}$. Then our new edge set E' we connect (u, i) to $(v, i + 1)$ with weight $c_i(u, v)$ for each $u, v \in V$ with $u \neq v$ and $i \in \{0, 1, \ldots, n-1\}$, where for $i = n-1$ we mean $(u, n-1)$ connects to $(v, n \mod n) = (v, 0)$. The new vertex set has $n \cdot n = n^2$ vertices and the new edge set can be upper bounded by $|E'| \leq n^2 \cdot n \leq O(n^3)$.

Now we run Bellman Ford on this new graph, starting at $(v_0, 0)$, since our weights are possibly negative. Then we iterate over all $i \in \{0, 1, \ldots, n-1\}$ and find the vertex of the form (v_1, i) which has the shortest distance to $(v_0, 0)$. There is a one-to-one correspondence between walks which start at v_0 and ending at v_1 in the original rotating graph and that of paths starting at $(v_0, 0)$ in the layered graph and ending at some (v_1, i) vertex. Thus the shortest path given by Bellman Ford will equal the shortest walk in the rotating original graph.

Bellman Ford runs in time equal to $O(|V'||E'|) \leq O(n^2 n^3) = O(n^5)$.

Guidelines for correction:

Award $1/2$ point for general idea of multiple layers describing different rotation states. Award 1 point if full construction is correct and uses analysis with Bellman-Ford properly.

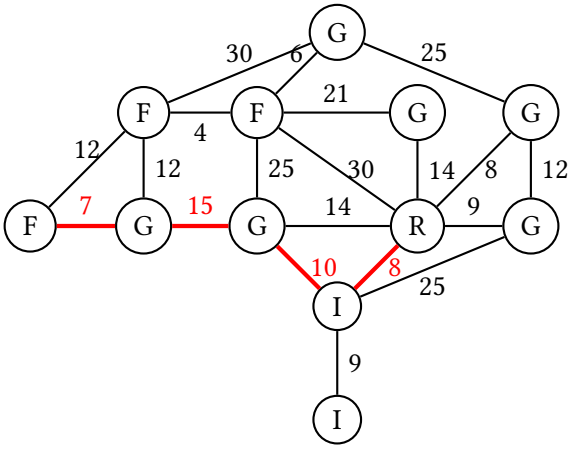
Exercise 11.3 *Language Hiking.*

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:

To solve this problem, we create a new directed graph. This graph contains multiple layers of our original graph, each containing all vertices of a specific language. Then we connect the vertices in and between each layer in a specific way, which allows us to simulate the hike.

[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 an $O(1)$. We obtain that the total runtime of the algorithm is $O((|V| + |E|)\log(|V|))$.

(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. 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:

We use a similar approach as before. As every combination is possible now, we use 4 variables to keep track of which languages where encountered, resulting in 2^4 layers. Each edge is used to update one boolean variable, e.g. for $(u,v) \in E$ and $l(v) = \mathrm{G}$, the boolean for G is set to 1. The sink can only be reached from vertices which satisfy all 4 booleans.

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

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

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

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 $(v, (b_\mathrm{G}, b_\mathrm{F}, b_\mathrm{I}, b_\mathrm{R}))$ such that only the boolean corresponding to the language of v (i.e., $\ell(v)$) is set to 1. 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 an $O(1)$. The total runtime of the algorithm is thus $O((|V| + |E|) \log(|V|))$.

Exercise 11.4 *Rail racer* **(1 point)**.

You are designing new routes for freight trains to ship cargo from Zurich to Zermatt. The railway network of Switzerland has many train stations (including Zurich and Zermatt). For simplicity, we assume there is at most one direct railway (i.e. not passing a third station) between any two stations and its length is a multiple of 10 km; also, the trains can travel on both directions of all railways. To guarantee safety, checkpoints are installed at every train station as well as at every 10 km on the railways. (For example, if there is railway of 40 km between station A and station B, then there will be 3 checkpoints on this railway in addition to the 2 checkpoints at station A and station B.) At each checkpoint, there is a speed limit: 50, 100, or 150 km/h. The trains should not exceed the speed limits at any checkpoint. The trains have three speed settings of 50, 100 and 150 km/h. The trains can increase or decrease its speed by 50 km/h linearly over a 10 km stretch of rail. The trains begin the journey from

Zurich at 50 km/h and must end in Zermatt at 50 km/h. (We ignore the acceleration phase from 0 to 50 km/h and the deceleration phase from 50 to 0 km/h.)

Calculate the shortest time the trains need to get from Zurich to Zermatt.

(a)* Calculating the acceleration can be done via the formula $a = (v_1^2 - v_0^2)/(2d)$ where v_0, v_1 are the initial and final speeds and d is the distance traveled. Using this formula, find the time it takes to traverse a 10km stretch of rails depending on your starting speed ($50, 100$ or 150 km/h) and also depending on whether you stay at a constant speed, or you accelerate/decelerate to the higher/lower speed.

Hint: The relationship between the time taken, acceleration and initial and final speeds is given by $v_1 = v_0 + a \cdot t$.

Solution:

For constant speed, we use $d = v_0 * t$ and solving for t we get $\frac{1}{5}$, $\frac{1}{10}$ and $\frac{1}{15}$hrs for speeds $50, 100$ and 150km/hr respectively.

For increasing speeds, we calculate the acceleration as $a = (v_1^2 - v_0^2)/(2d) = (v_1^2 - v_0^2)/(20)$. Then we use a change in speed formula of $v_1 = v_0 + a \cdot t$ and solve for t to get

$$t = \frac{v_1 - v_0}{a} = \frac{20(v_1 - v_0)}{v_1^2 - v_0^2} = \frac{20(v_1 - v_0)}{(v_1 - v_0)(v_1 + v_0)} = \frac{20}{v_1 + v_0}.$$

So for 50 to 100km/hr, we get $\frac{2}{15}$hrs, and for 100 to 150km/hr, we get $\frac{2}{25}$ hrs. Note it doesn't matter if we swap v_1 and v_0 so these times are also the same for decelerating.

(b) Model the problem as a graph problem such that you can directly apply one of the algorithms in the lecture, without modifications to the algorithm:

(1) Describe your graph. What are the vertices, what are the edges and the weights of the edges?

Solution:

The define the graph $G = (V, E, w)$ as follows: Let V_0 be the set of all checkpoints and stations, then we create 3 layers of V_0, one for each speed, so $V = V_0 \times \{50, 100, 150\}$. For each railway between two station/checkpoints, a and b, we connect a directed edge from (a, v_0) to (b, v_1) if two conditions are met:

i. v_0 respects a's speed limit and v_1 respects b's speed limit.

ii. v_0 and v_1 differ by at most 50.

The weights of edges will represent the time it takes to cross that 10km stretch. So within the same layer(meaning constant speed) will be $\frac{1}{5}$, $\frac{1}{10}$ and $\frac{1}{15}$ for layers $50, 100$ and 150 respectively. Between adjacent layers of 50 and 100 we have weight $\frac{2}{15}$ and between 100 and 150 we have weight $\frac{2}{25}$. Without doing part a, this will be $\frac{c}{150}$ and $\frac{c}{250}$.

Remark. Since the time it takes to accelerate/decelerate are the same, every directed has a $((a, v_0), (b, v_1))$ will have the same time to traverse as $((b, v_1), (a, v_0))$. Therefore using undirected edges is also fine.

(2) What is the graph problem that we are trying to solve?

Solution:

Since time taken to traverse a distance is nonnegative, we can assume that edge weights are positive. We are trying to solve the shortest time to get from $(\text{Zurich}, 50)$ to $(\text{Zermatt}, 50)$ since

we must start and stop at 50km/hr. So this is then the shortest path between the two vertices in the graph.

(3) Solve the problem using an algorithm discussed in the lecture (without modification).

Solution:

We apply Dijkstra's algorithm to $(\text{Zurich}, 50)$ to find the shortest path from $(\text{Zurich}, 50)$ to $(\text{Zermatt}, 50)$.

a

Note: If you didn't solve part (a)*, you can assume the time it takes to travel 10 km with starting speed v_0 and ending speed v_1 is $c/(v_0 + v_1)$ for some positive integer c.

Guidelines for correction:

Award $1/2$ point if general 3 layered graph is attempted and shortest path problem is recognized. Award 1 point for fully correct construction and correct algorithm.

Exercise 11.5 *Modernization of the Deutsche Bahn.*

The Deutsche Bahn (DB) has n train stations and m train tracks connecting them, so that from one station you can reach all other $n - 1$ stations. To save money, not all train tracks are actively maintained. However, the current state leads to a great number of problems, such as high delays and general unreliability, and this is very annoying for some ETH students traveling home for Christmas.

The DB wants to modernize some of the train tracks so that every two stations are connected by modernized tracks. Unfortunately, modernized tracks require frequent investments to maintain. For this exercise, we assume that the price for modernizing a track is negligible and we only consider the yearly maintenance cost. This cost is proportional to the length of the track. The DB modernizes a subset of the tracks such that the entire yearly maintenance cost is as small as possible.

After some time of operation, the DB notices that they also need a new signal at every modernized track to fully operate it. Again, we assume that the building cost is negligible and we don't consider it in this exercise. These signals also have a yearly maintenance cost k, which is the same for each signal.

Now, the DB asks you: in order to minimize the overall yearly maintenance cost (for both tracks and signals combined), whether they can just maintain the set of modernized tracks they already have and build a signal on each modernized track? Or do they have to modernize some new tracks (and potentially shut down some already modernized tracks) in order to be able to achieve the minimum yearly maintenance cost for both signals and tracks?

Decide whether they can keep the already maintained tracks or not. If so, explain your reasoning, otherwise provide a counterexample that shows that it might be necessary to modernize new tracks. Recall that we assume that the cost of modernizing a track is negligible and that we are only interested in minimizing the yearly maintenance cost.

Solution:

We rephrase this problem as a graph problem. Let us define the vertices as train stations and the edges as the tracks between them, i.e. $V = \{1, 2, \ldots, n\}$ and $E = \{\{i, j\} \mid \text{Station } i \text{ and } j \text{ are connected }\}$.

In order to minimize the overall maintenance price, the DB must modernize only train lines that form a minimum spanning tree T in $G = (V, E)$.

As a result, we can rephrase the problem as the following graph problem. Let T be a minimum spanning tree of a weighted graph G. Construct a new graph G' by increasing the weight of each edge in G by k. Do the edges of T form a minimum spanning tree of G'?

In a graph with n vertices, every spanning tree has $n - 1$ edges. Thus the weight of every spanning tree is increased by exactly $(n - 1) \cdot k$. Therefore, the minimum spanning trees remains the same.

In other words, the DB does not have to consider modernizing new tracks: Keeping the current modernized tracks and building a signal on each of them still guarantees minimal maintenance cost.