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
Lucas Slot, Manuel Wiedmer, Hongjie Chen, Ding Jingqiu

4 December 2023

Algorithms & Data Structures Exercise sheet 11 HS 23

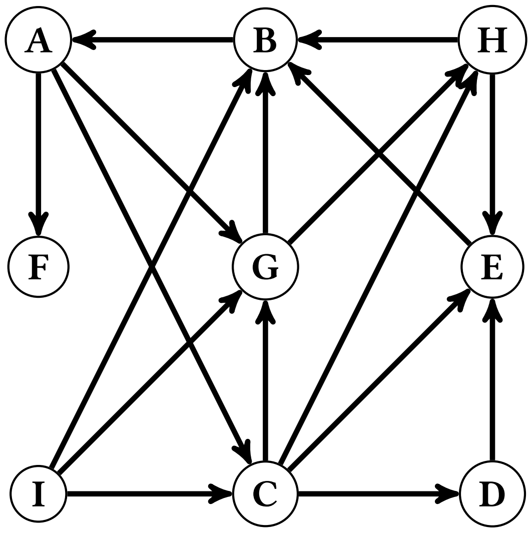
The solutions for this sheet are submitted at the beginning of the exercise class on 11 December 2023.

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 *Breath-first search* **(1 point)**.

Execute a breadth-first search (Breitensuche) on the following graph[1] starting from vertex A. Use the algorithm presented in the lecture. When processing the neighbors of a vertex, process them in alphabetical order.



(a) For each vertex, give its enter- and leave-number.

Solution:

$A : (0, 1), B : (11, 15), C : (2, 5), D : (6, 12), E : (7, 13), F : (3, 9), G : (4, 10), H : (8, 14).$

Vertex I is not reachable from A, so it does not occur in the breath-first search and does not get an enter- or leave-number.

(b) Give the distances from A to any vertex.

Solution:

[1]This is the same graph that was already used in exercise 10.2 as an example how depth-first search works.

$d(A, A) = 0$, $d(A, B) = 2$, $d(A, C) = 1$, $d(A, D) = 2$, $d(A, E) = 2$, $d(A, F) = 1$, $d(A, G) = 1$, $d(A, H) = 2$.

The distance $d(A, I)$ has not been assigned since I cannot be reached from A.

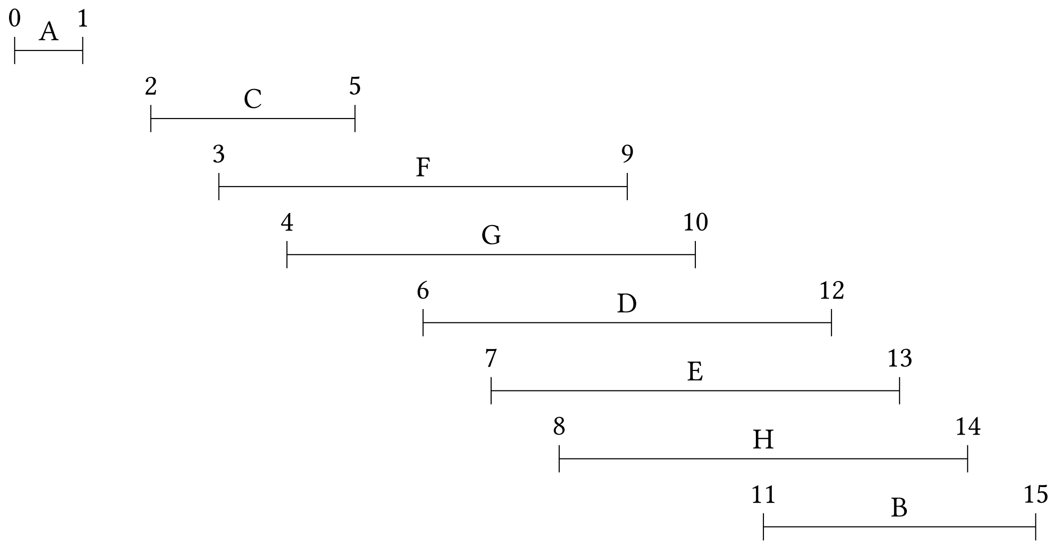
(c) Give the ordering of the vertices that results from sorting them by enter-number (or equivalently by leave-number).

Solution:

The order of the vertices according to enter-number is (A, C, F, G, D, E, H, B). Note that I does not occur in this ordering since it has not been assigned an enter-number.

(d) Draw a scale from 0 to 18 and mark for every vertex v the interval I_v we get from the above execution of breadth-first search. Is it possible that for some vertices $v \neq w$ we have $I_v \subseteq I_w$ (for a general graph G)? Why/Why not?

Solution:



Let v and w be two distinct vertices of G. We have that $I_v \subseteq I_w$ is equivalent to enter$[v] <$ enter$[w]$ and leave$[w] <$ leave$[v]$ (note that the inequalities are strict since $v \neq w$). We know from the lecture that if enter$[v] <$ enter$[w]$, then also leave$[v] <$ leave$[w]$ (the enter- and leave- order are the same). Hence, we cannot have $I_v \subseteq I_w$ for any two distinct vertices v and w.

Guidelines for correction:

The following 3 elements are important in this exercise. If all of them are solved correctly, award 1 point. If at least 2 are solved correctly, award 1/2 point.

- Correctly determining the enter- and leave-numbers in part (a) and the ordering in (c).
- Correctly determining the distances from A (it is ok to set $d(A, I) = \infty$) in part (b).
- Arguing that we can never have $I_v \subseteq I_w$ for distinct vertices v and w in part (d).

Exercise 11.2 *Shortest paths with cheating* **(1 point).**

Let $G = (V, E)$ be a weighted, directed graph with weights $c : E \to \mathbb{R}_{\geq 0}$. We consider a variation of the shortest path problem in G, where we are allowed to 'cheat' by setting a certain number of weights to 0. Formally, for $k \in \mathbb{N}$, we write C_k for the set of all weight functions $\gamma : E \to \mathbb{R}_{\geq 0}$ on G with $\gamma(e) \neq c(e)$ for at most k edges $e \in E$.[2]

Given $s, t \in V$, we wish to find a path $P = (v_1 = s, v_2, \ldots, v_\ell = t)$ in G which minimizes:

$$c_k(P) := \min_{\gamma \in C_k} \gamma(P), \text{ where } \gamma(P) := \sum_{i=1}^{\ell-1} \gamma\big((v_i, v_{i+1})\big).$$

We call such a path a 'shortest path from s to t with k cheats.'

Recall that a naive implementation of Dijkstra's algorithm finds the length of a shortest path in a weighted graph (without cheating) in time $O(|V|^2)$.

(a) Describe an algorithm which finds the length of a shortest path from s to t with k cheats in time $O(|E|^k \cdot |V|^2)$. Prove that your algorithm is correct, and achieves the desired runtime.

Hint: *Apply Dijkstra's algorithm to G several times, for different weight functions.*

Solution:

Note that we can assume that we set exactly k weights to 0 (instead of at most k). The reason for this is that when setting a weight to 0, the length of any path can only decrease. So setting additional edges to 0 will not increase the length of a shortest path.

We iterate over all possible ways to set k weights to 0. In each iteration, we apply Dijkstra's algorithm to find a shortest path for that particular choice of cheat-edges. We output the smallest length over all iterations.

To see that this algorithm achieves the desired runtime, note that there are $\binom{|E|}{k} \leq |E|^k$ ways to select the k cheat-edges. So we have at most $|E|^k$ iterations, and each iteration takes $O(|V|^2)$ time, leading to $O(|E|^k \cdot |V|^2)$ time in total.

(b) Describe an algorithm which finds the length of a shortest path from s to t with k cheats in time $O\big((k|V|)^2\big)$. Prove that your algorithm is correct, and achieves the desired runtime.

Hint: *Construct a new graph $G' = (V', E')$ whose vertex set V' consists of $k + 1$ copies of V. Choose the edges E' and weights c' in a clever way, and apply Dijkstra's algorithm to G'.*

Solution:

We set $V' = \{v^{(\ell)} : v \in V,\ \ell \in \{0, 1, 2, \ldots, k\}\}$, which has size $|V'| = (k + 1) \cdot |V|$. We define E' and $c' : E' \to \mathbb{R}_{\geq 0}$ by:

$$\forall\, (v, w) \in E,\ 0 \leq \ell \leq k : \quad (v^{(\ell)}, w^{(\ell)}) \in E' \text{ with } c'((v^{(\ell)}, w^{(\ell)})) = c((v, w)), \tag{T1}$$

$$\forall\, (v, w) \in E,\ 0 \leq \ell \leq k-1 : \quad (v^{(\ell)}, w^{(\ell+1)}) \in E' \text{ with } c'((v^{(\ell)}, w^{(\ell+1)})) = 0, \tag{T2}$$

$$\forall\, 0 \leq \ell \leq k-1 : \quad (t^{(\ell)}, t^{(\ell+1)}) \in E' \text{ with } c'((t^{(\ell)}, t^{(\ell+1)})) = 0. \tag{T3}$$

The first set of edges (T1) represents normal edge-traversal in G, the second set (T2) represents cheating, and the final set (T3) ensures that there is always a path from $s^{(0)}$ to $t^{(k)}$ in G' if there is a path from s to t in G.

[2]We assume that $|E| \geq k$.

We run Dijkstra's algorithm on (G', c') to find the length of a shortest path between $s^{(0)}$ and $t^{(k)}$ in time $O(|V'|^2) \le O\big((k|V|)^2\big)$.

It remains to show our algorithm is correct. Write L for the length of a shortest path from s to t in (G, c) with k cheats (or $L = \infty$ if no such path exists). Write L' for the length of a shortest path from $s^{(0)}$ to $t^{(k)}$ in (G', c') (or $L' = \infty$ if not such path exists).

We show first that $L' \ge L$. If there is no path from $s^{(0)}$ to $t^{(k)}$ in (G'), then surely $L' \ge L$, so we may assume there is a shortest path $P' = (v_1^{(a_1)}, \dots, v_\ell^{(a_\ell)})$ from $s^{(0)}$ to $t^{(k)}$ in (G', c'). By definition of E', we know that $P = (v_1, v_2, \dots, v_\ell)$ is a path in G from s to t (after possibly removing some subsequent copies of t from the path, corresponding to edges of type (T3), which have weight 0). Since there are at most k edges of type (T2) in P', we conclude that there is a weight function $\gamma \in C_k$ such that $\gamma(P) = c'(P')$. But that means $L \le c_k(P) \le L'$.

Next we show $L \ge L'$. Let $P = (v_1, v_2, \dots, v_\ell)$ be a shortest path from s to t in G with k cheats. Again, we may assume such a path exists. Let $\gamma \in C_k$ be the weight function for which $\gamma(P) = c_k(P) = L$. Now, consider the path $P' = (v_1^{(a_1)}, \dots, v_\ell^{(a_\ell)})$ in G', where $a_1, \dots, a_\ell$ are defined by $a_1 = 0$, and, for $1 \le i \le \ell$,

$$a_i = \begin{cases} a_{i-1} + 1 & \text{if } c((v_{i-1}, v_i)) \ne \gamma((v_{i-1}, v_i)), \\ a_{i-1} & \text{if } c((v_{i-1}, v_i)) = \gamma((v_{i-1}, v_i)). \end{cases}$$

This path is well-defined because $c((v_{i-1}, v_i)) \ne \gamma((v_{i-1}, v_i))$ for at most k different i. By definition of c', the length of P' in (G', c') is at most L. We know that $v_1^{(a_1)} = s^{(0)}$, and $v_\ell^{(a_\ell)} = t^{(i)}$, for some $0 \le i \le k$. We can thus extend P' to be a path P'' from $s^{(0)}$ to $t^{(k)}$ using edges of type (T3) (whose cost is zero). We conclude that $L' \le c'(P'') \le \gamma(P) = L$.

Guidelines for correction:

Award 1/2 point for each part. In part (b), it is important to show the correspondence between shortest paths with cheats in G, and shortest paths in G', which includes:

- Correct definition of G', γ.
- Correct proof that $L' \ge L$.
- Correct proof that $L \ge L'$.

Award 1/2 points if at least 2 of these points are present. If the only missing element of a solution is the edges of type (T3), award 1 point.

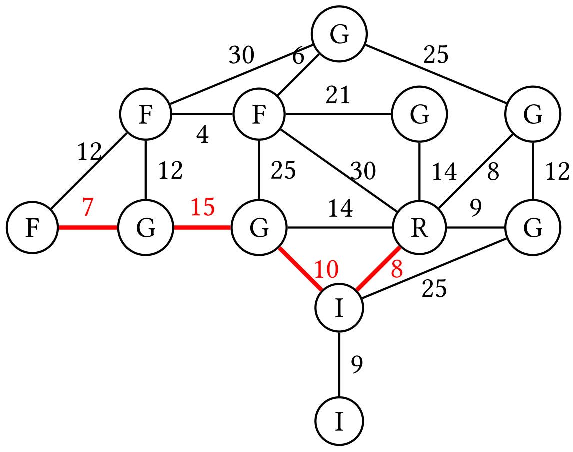
Exercise 11.3 *Language Hiking.*

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[3]. 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:

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

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

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

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

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

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. 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':

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

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

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 *Driving from Zurich to Geneva* **(1 point)**.

Bob is currently in Zurich and wants to visit his friend that lives in Geneva. He wants to travel there by car and wants to use only highways. His goal is to get to Geneva as cheap as possible. He has a map of the cities in Europe and which ones are connected by highways (in both directions). For each highway connecting two cities he knows how much fuel he will need for this part (depending on the length, condition of the road, speed limit, etc.) and how much this will cost him. This cost might be different depending on the direction in which he travels. Furthermore, for some connections between two cities, he has the option to take a passenger with him that will pay him a certain amount of money. Again this might be different depending on the direction he travels. We assume that this option is only availabe to him between cities directly connected by a highway and that the passengers want to travel the direct road and would not agree to making a detour. Also Bob has a small car, so he can only take at most one passenger with him. It is possible that he gains more money from this than he has to pay for the fuel between two given cities but we assume that he has no way to gain an infinite amount of money, i.e. there is no round-trip from any city that earns him money.

(a) 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 graph $G = (V, E, w)$ is defined as follows: V is the set of cities on his map of Europe. There are directed edges between the cities (in both directions) if they are connected by a highway. The weight of any edge is the difference of the cost he needs to pay for the fuel for this highway and the money that a passenger would pay him (if available, otherwise it is just the cost he needs pay).

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

Solution:

We are trying to find the shortest path from the vertex v_{Zurich} corresponding to Zurich and the vertex v_{Geneva} corresponding to Geneva in the graph G. Note that since from no city he has a roundtrip that gains him money, the graph G does not have negative cycles, so we are indeed looking for a path and not a walk.

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

Solution:

We can apply the Bellman-Ford algorithm to (G, v_{Zurich}) to find the shortest path from v_{Zurich} to v_{Geneva} (we have no negative cycles, so Bellman-Ford works). Note that the edge weights might be negative, so we cannot apply Dijkstra's algorithm.

(b) Now we change the problem slightly. Bob got a list from his friend of certain highways that are in a bad condition. To not damage his car, he decided that he want to use at most one of these highways. Again, 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:

We define the graph $G' = (V', E', w')$ as follows: The set of vertices is $V' = V \times \{0, 1\}$. For every edge $e = (u, v) \in E$, if it is a normal highway, we have the edges $((u, 0), (v, 0))$ and $((u, 1), (v, 1))$ in E'. If it is a highway in bad condition, then we have the edge $((u, 0), (v, 1))$ in E'. The weight function w' is as before: The weight of any edge is the difference of the cost he needs to pay for the fuel for this highway and the money that a passenger would pay him (if available, otherwise it is just the cost he needs pay).

This construction can be interpreted as follows: We have two levels in the graph G', namely $V \times \{0\}$ and $V \times \{1\}$ and they are connected in such a way that it matches the problem description. Highways in bad condition connect layer 0 to layer 1 (and only in this direction). All other highways are present in both layer 0 and layer 1 (and do not cross between layers). Consider a path in this modified graph G'. Since we have no edges from layer 1 to layer 0 and whenever a highway in bad condition is used, the path moves from layer 0 to layer 1, this path can use at most one highway in bad condition.

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

Solution:

We are trying to find the shortest path between $(v_{\text{Zurich}}, 0)$ and $(v_{\text{Geneva}}, 0)$ or $(v_{\text{Geneva}}, 1)$ (whichever is shorter). Again, since the graph has no negative cycles, we are indeed looking for a path and not just a walk. Since we argued above that a path in G' uses at most one highway in bad condition, this is indeed what we are looking for (paths between $(v_{\text{Zurich}}, 0)$ and $(v_{\text{Geneva}}, 0)$ use no highway in bad condition and paths between $(v_{\text{Zurich}}, 0)$ and $(v_{\text{Geneva}}, 1)$ use exactly one).

Remark: If we wanted to model the problem as finding a single shortest path, we could add a vertex t to the graph and add the two edges $((v_{Geneva}, 0), t)$ and $((v_{Geneva}, 1), t)$. The weight of these two edges can be set to 0. Then the goal would to find a shortest path between $(v_{Zurich}, 0)$ and t. The shortest path from Zurich to Geneva would then be this shortest path without the last edge $((v_{Geneva}, i), t)$.

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

Solution:

We can again apply the Bellman-Ford algorithm to the modified graph $(G', (v_{\text{Zurich}}, 0))$ to find the shortest paths from $(v_{\text{Zurich}}, 0)$ to $(v_{\text{Geneva}}, 0)$ and $(v_{\text{Geneva}}, 1)$ (we again have no negative cycles, so Bellman-Ford works).

Guidelines for correction:

$1/2$ point should be awarded for correctly solving part (a) and $1/2$ point should be awarded for correctly solving part (b). However, if in both part (a) and (b) everything is correct except the algorithm to use (e.g. if Dijkstra is proposed to solve the problem), still $1/2$ point should be awarded.

Exercise 11.5 *Ancient Kingdom of Macedon.*

The ancient Kingdom of Macedon had n cities and m roads connecting them, such that from one city, you can reach all other $n - 1$ cities. All roads were *roman roads*, i.e. stone-paved roads that did not require any maintenance, and no two roads were of the same length. With the technological developments in the Roman Kingdom, a new type of carriage was developed, called the *Tesla Carriage*, which was much faster than all the alternatives in the Ancient Macedon Kingdom. However, the Tesla Carriage required *asphalt roads* to operate, and such roads had to be maintained every year, or otherwise the asphalt would wear off, rendering the road unusable as if it was a roman road.

With the effort to modernize the kingdom, Phillip II promised the Ancient Macedonians that he will provide them with asphalt roads by paving some of the existing roman roads, such that every two cities can be reached with a Tesla Carriage. The price to pave a roman road or maintain an asphalt road is equal, and is proportional to the length of the road. To save money Phillip II decided to pave sufficient roman roads to fulfill his promise, while minimizing the overall yearly maintenance price.

Even in the first years, the new Tesla Carriages improved the lives of the average Ancient Macedonians, but at the same time, they also provided means for robbers to commit crimes and escape to another city. To resolve this, Phillip II decided to create checkpoints the second year, one at each asphalt road. Each of the checkpoint will have a fixed cost for both building and maintenance.

Assuming a fixed price k for each checkpoint, does Phillip II have to consider paving new roman roads, or he can maintain the same set of roads in order to make sure that the overall maintenance price of the roads and the checkpoints is still minimal? Prove your reasoning, or provide a counterexample.

Note: For simplicity, assume that the roman roads were paved all at once, on the first day of the year, and maintenance will be done the same day next year, again all at once. Also assume that checkpoints can also be built at once for all roads, as well a they can be maintained all at once in a day.

Solution:

Let's think of all cities in the Ancient Macedon Kingdom as vertices in a graph G and all roads as edges. In order to minimize the overall maintenance price, Philip II must pave only roman roads that form a minimum spanning tree T in G. 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, Phillip II does not have to consider paving new roman roads: Keeping the current asphalt roads and building a checkpoint on each of them still guarantees minimal maintenance cost.