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

11 December 2023

Algorithms & Data Structures — Exercise sheet 12 — HS 23

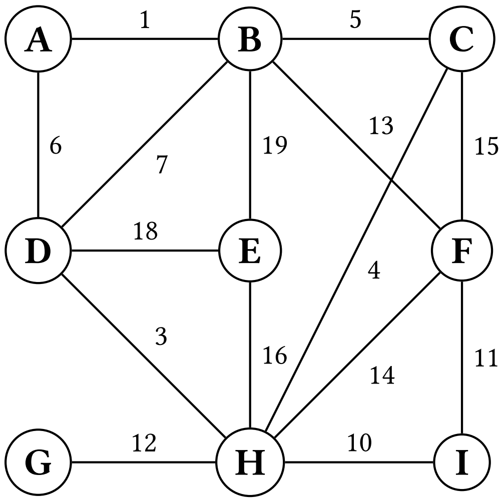
The solutions for this sheet are submitted at the beginning of the exercise class on 18 December 2023.

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 12.1 *MST practice* **(1 point)**.

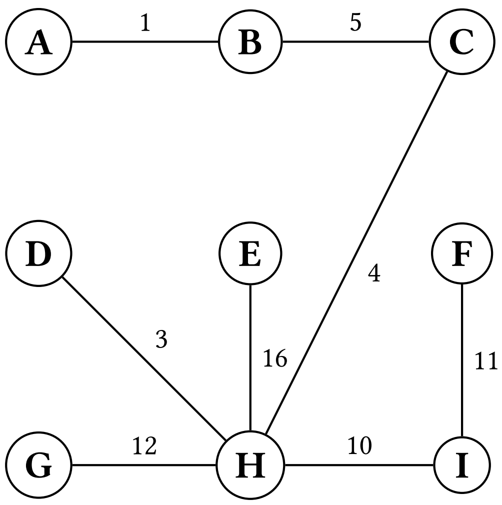
Consider the following graph.



(a) Compute the minimum spanning tree (MST) using Boruvka's algorithm. For each step, provide the set of edges that are added to the MST.

Solution:

In the first step we add the edges $\{A, B\}$ (for A and B), $\{C, H\}$ (for C), $\{D, H\}$ (for D and H), $\{E, H\}$ (for E), $\{F, I\}$ (for F), $\{G, H\}$ (for G) and $\{H, I\}$ (for I). This gives the components $\{A, B\}$ and $\{C, D, E, F, G, H, I\}$. In the second step, we add the edge $\{B, C\}$. Thus the minimum spanning tree consists of the following edges:



(b) Provide the order in which Kruskal's algorithm adds the edges to the MST.

Solution:

In Kruskal's algorithm the edges are added in the following order: $\{A, B\}$, $\{D, H\}$, $\{C, H\}$, $\{B, C\}$, $\{H, I\}$, $\{F, I\}$, $\{G, H\}$ and $\{E, H\}$.

(c) Provide the order in which Prim's algorithm (starting at vertex G) adds the edges to the MST.

Solution:

In Prim's algorithm the edges are added in the following order: $\{G, H\}$, $\{D, H\}$, $\{C, H\}$, $\{B, C\}$, $\{A, B\}$, $\{H, I\}$, $\{F, I\}$ and $\{E, H\}$.

Guidelines for correction:

If two out of the three subtasks are solved correctly, award $1/2$ point. If all subtasks are solved correctly, award 1 point.

Exercise 12.2 *Uniqueness of MSTs* **(1 point)**.

The goal of this exercise is to understand when a graph has a unique minimum spanning tree.

(a) Give an example of a graph for which the minimum spanning tree is not unique. Show how to get two different minimum spanning trees of this graph using Kruskal's or Prim's algorithm. When there is a choice because several edges have the same weight, the algorithms are allowed to pick any of those edges.

Solution:

Consider the following graph $G = (V, E)$ with $V = \{v_1, v_2, v_3\}$, $E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}\}$ and $w(\{v_1, v_2\}) = w(\{v_1, v_3\}) = w(\{v_2, v_3\}) = 1$.

If we run Kruskal's algorithm, we can first add the edge $\{v_1, v_2\}$ and then the edge $\{v_2, v_3\}$. Hence, $\{\{v_1, v_2\}, \{v_2, v_3\}\}$ is a minimum spanning tree.

If we run Prim's algorithm starting from v_3, we could first add $\{v_1, v_3\}$ and then $\{v_1, v_2\}$. Hence, $\{\{v_1, v_3\}, \{v_1, v_2\}\}$ is a minimum spanning tree as well. Thus, the minimum spanning tree of this graph G is not unique.

Remark: Note that we could have also gotten the second spanning tree with Kruskal's algorithm by different choices in the cases where we had edges with same weights. Similarly, we could have gotten the first spanning tree with Prim's algorithm by choosing differently in the cases of edges with the same weight.

It turns out that for a connected graph, if the weights of the edges are pairwise distinct, the minimum spanning tree is unique. To show this, let $G = (V, E)$ be a connected graph and $w : E \to \mathbb{R}_{\geq 0}$ be a weight function such that $w(e) \neq w(f)$ for $e, f \in E$ with $e \neq f$. We assume by contradiction that there are two different minimum spanning trees T_1 and T_2. Out of all edges that are in $T_1 \backslash T_2$ or $T_2 \backslash T_1$, let e be the edge of minimum weight (the edge of minimum weight is unique since by assumption the edge weights are pairwise distinct). By exchanging the roles of T_1 and T_2 if necessary, we can assume that $e \in T_1 \backslash T_2$.

(b) Show that $T_2 \cup \{e\}$ has a cycle and that there is an edge $f \in T_2 \backslash T_1$ on this cycle that has strictly larger weight than e.

Solution:

Since T_2 is a spanning tree, the graph $(V, T_2 \cup \{e\})$ is connected. By exercise 8.4b we know that a tree on $n = |V|$ edges has $n - 1$ edges. Thus, $|T_2 \cup \{e\}| = n - 1 + 1 = n > n - 1$ (note that $e \notin T_2$). Hence, $T_2 \cup \{e\}$ cannot be a tree and since it is connected, it has to have a cycle C (a tree is a connected graph without cycles).

Next, we want to show the existence of an edge f as claimed. For this, note that T_1 has no cycles, thus there is an edge f on the cycle C that is not in T_1. By definition of C, $f \in T_2 \cup \{e\}$ and since $e \in T_1$ we have $f \neq e$ and thus $f \in T_2$. We get that $f \in T_2 \backslash T_1$ and by the definition of e we have $w(e) \leq w(f)$ as wanted. Since the weights are pairwise distinct, we have $w(e) < w(f)$.

(c) Show that the minimum spanning tree of G with the weight function w is unique.

Hint: *Use part (b) to construct a spanning tree of smaller weight than T_2.*

Solution:

We claim that the following is a spanning tree: $(T_2 \cup \{e\}) \backslash \{f\}$. Since, $e \in T_1 \backslash T_2$ and $f \in T_2 \backslash T_1$, we have that $|(T_2 \cup \{e\}) \backslash \{f\}| = n - 1$. Also, the subgraph $(V, (T_2 \cup \{e\}) \backslash \{f\})$ is connected: Let $v, w \in V$. Since T_2 is spanning tree, there is a walk from v to w in T_2. If this walk does not contain f, then it is a walk in $(V, (T_2 \cup \{e\}) \backslash \{f\})$. Otherwise, we replace the edge f by $C \backslash \{f\}$. All these edges are in $(T_2 \cup \{e\}) \backslash \{f\}$. Thus, also in this case we get a walk from v to w in $(V, (T_2 \cup \{e\}) \backslash \{f\})$. Hence, $(T_2 \cup \{e\}) \backslash \{f\}$ is indeed spanning tree (see exercise 8.4c).

The weight of $(V, (T_2 \cup \{e\}) \backslash \{f\})$ is $w(T_2) + w(e) - w(f) < w(T_2)$ (by part (b) $w(e) < w(f)$). This is a contradiction since we assumed that T_2 is a minimum spanning tree. Thus, our assumption was wrong and we cannot have two different minimum spanning trees. Since the graph is connected, it has a (minimum) spanning tree, so G has a unique minimum spanning tree.

(d) Is the converse true as well? That is, if $G = (V, E)$ is a connected graph that has a unique minimum spanning tree, are the edge weights necessarily distinct? Give a proof or counterexample.

Solution:

This is not true, for example if G is a tree, then it has by construction only one spanning tree. Hence, no matter the weights, the minimum spanning tree will be unique.

As a concrete example we can take the graph $G = (V, E)$ with vertices $V = \{v_1, v_2, v_3\}$, edges $E = \{\{v_1, v_2\}, \{v_2, v_3\}\}$ and weights $w(\{v_1, v_2\}) = w(\{v_2, v_3\}) = 1$.

Note that the statement is still wrong if we exclude graphs that have just one spanning tree (i.e. trees). For example consider the graph $G' = (V', E')$ with vertices $V' = \{v_1, v_2, v_3\}$, edges $E' = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}\}$ and weights $w(\{v_1, v_2\}) = w(\{v_1, v_3\}) = 1$ and $w(\{v_2, v_3\}) = 2$. Then, G' has several spanning trees but the minimum spanning tree is unique: $\{\{v_1, v_2\}, \{v_1, v_3\}\}$.

Guidelines for correction:

The following 8 elements are important parts of this exercise. If at least 4 are solved correctly, $1/2$ point should be awarded. If at least 7 are solved correctly, 1 point should be awarded.

- Correct example in part (a).
- Showing that $T_2 \cup \{e\}$ contains a cycle (part (b)).
- Showing that there is an edge on this cycle that is in $T_2 \setminus T_1$ (part (b)).
- Concluding that this edge has strictly larger weight than e by definition of e (part (b)).
- Showing that $(T_2 \cup \{e\}) \setminus \{f\}$ is a spanning tree (part (c)).
- Computing the weight of $(T_2 \cup \{e\}) \setminus \{f\}$ and noting that it is smaller than the weight of T_2 (part (c)).
- Concluding that the minimum spanning tree in G is unique (part (c)).
- Correct counterexample in part (d).

Exercise 12.3 *Constructing a Fiber Optic Network.*

The government of Atlantis put you in charge of installing a fiber optic network that connects all its n cities. There are two technologies of fibre optic that you can use:

- Fibre 1.0: It is a good reliable technology that is relatively cheap. There is a list of pairs of cities between which it is possible to install a direct Fibre 1.0 link. Furthermore, for each such pair, there is a corresponding positive integer cost.
- Fibre 2.0: It is an emerging technology that it extremely good and can directly connect any two cities. However, its cost is too high and the government cannot afford a single Fibre 2.0 link.

Note that all direct links are two-directional. The installed network should connect all the cities of Atlantis: Between any two cities, there should be a connected path of direct links in the network that connects them.

A philanthropist volunteered to donate the cost of exactly $k < n$ direct Fibre 2.0 links, and you can use them to connect any k pairs of cities. Your goal is to minimize the cost that is paid by the government for the Fibre 1.0 links that are needed to construct a connected network. Describe an algorithm that finds the network that costs the government the minimum amount of money.

Note that it is possible to construct a network connecting all the cities of Atlantis using only Fibre 1.0 links, but we would like to benefit from the k Fibre 2.0 links that were donated by the philantropist in order to minimize the cost that is paid by the government.

Hint: *Modify Kruskal's algorithm.*

Solution:

Before stating the solution, we introduce some terminology: An undirected graph is said to be a forest if it does not contain any cycle. In other words, a graph is a forest if and only if its connected components are trees. We say that it is an ℓ-forest if it has ℓ connected components. Note that a graph is a 1-forest if and only if it is a tree.

Note that the removal of k edges from a tree yields a $(k+1)$-forest. Conversely, if we have an ℓ-forest, we can convert it into a tree by adding $\ell - 1$ edges connecting its connecting components without creating cycles.

Let $G = (V, E)$ be the undirected graph where the set of vertices V corresponds to the cities of Antlantis and the set of edges E corresponds to the pairs of cities between which we can install direct Fibre 1.0 links. The edges in E are weighted by the cost of installing direct Fibre 1.0 links.

Let T be the tree corresponding to the network to be installed. Let $e_1, \dots, e_k$ be the k edges corresponding to the Fibre 2.0 links that will be donated. Note that $e_1, \dots, e_k$ may or may not be in E.

Let $F = T \setminus \{e_1, \dots, e_k\}$ correspond to the Fibre 1.0 links for which the government has to pay. It is easy to see that F is a spanning $(k+1)$-forest of G: It is a subgraph of G (with the same set of vertices) which is also a $(k+1)$-forest.

We can now see that the problem is equivalent to finding a minimum spanning $(k+1)$-forest of G, i.e., one that has the minimum total weight. This can be done using a slight modification of Kruskal's algorithm. Let $n = |V|$ be the number of cities. Instead of completing Kruskal's procedure until adding $n - 1$ edges, we stop after adding $n - k - 1$ edges from E. The proof of correctness of this algorithm is very similar to that of Kruskal's algorithm for the minimum spanning tree problem.

Exercise 12.4 *TST and MST* (1 point).

Let $G = (V, E)$ be a connected, weighted graph, with weights $w : E \to \mathbb{R}_{\geq 0}$. A *travelling salesperson tour* (TST) in G is a closed walk which visits each vertex $v \in V$ at least once. We write $\mathrm{tst}(G)$ for the length of a shortest TST in G, that is:

$$\mathrm{tst}(G) = \min_{\substack{P=(v_1,\dots,v_\ell)\\ \text{is a TST in } G}} w(P), \qquad \text{where } w(P) := \sum_{i=1}^{\ell-1} w\big(\{v_i, v_{i+1}\}\big).$$

(a) Let $M \subseteq E$ be the edges of a minimum spanning tree of G, with weight $w(M) := \sum_{e \in M} w(e)$. Prove that $w(M) \leq \mathrm{tst}(G)$.

Solution:

Let P be a TST in G, and let $E(P)$ be the set of edges traversed by P. Then $E(P)$ spans V. Therefore, the graph $G' = (V, E(P))$ is connected, and thus it has a spanning tree T, whose weight is at most $w(P)$. But T is also a spanning tree for G, and so $w(M) \leq w(T) \leq w(P)$.

(b) Let $H = (V, M_{\text{double}})$ be the multigraph with vertex set V, and edge set M_{double} containing two copies of each edge $e \in M$. Prove that H has a Eulerian tour of length $2 \cdot w(M)$.

Hint: *See Exercise 10.1. What can you say about the degree of a vertex in H?*

Solution:

As we have doubled all edges in M to obtain M_{double}, each vertex $v \in V$ has even degree (in H). But this implies that H has a Eulerian tour. (To see this, we can use the construction of Exercise 10.1, which shows H has a Eulerian tour if and only if the (simple) graph H' obtained by subdivision of the edges of H has a Eulerian tour. The vertices of that graph all have even degree, and for simple graphs we know that this is equivalent to having a Eulerian tour). The length of a Eulerian tour in H is just $\sum_{e \in M_{\text{double}}} w(e) = 2 \sum_{e \in M} w(e) = 2 \cdot w(M)$.

(c) Describe an algorithm which outputs a TST in G of length at most $2 \cdot \text{mst}(G)$, where $\text{mst}(G)$ is the length of a minimum spanning tree of G. The runtime of your algorithm should be at most $O(|E| \log |E|)$. Prove that your algorithm is correct and achieves the desired runtime.

Hint: *For a connected graph with n vertices and m edges, you may use the fact that there exists an algorithm to find a minimum spanning tree in time $O(m \log m)$, and a Eulerian tour (if one exists) in time $O(m)$.*

Solution:

We start by finding a minimum spanning tree in G, with edges $M \subseteq E$, in time $O(|E| \log |E|)$. We then construct the multigraph H from part (b), and its (simple) subdivision graph H' in time $O(|V| + |E|)$. We find a Eulerian tour in H' (or H) (which exists by part (b)) in time $O(|E|)$. This Eulerian tour corresponds directly to a walk in G, which we output. This path visits each vertex $v \in V$ at least once. It has weight at most $2 \cdot w(M) = 2 \cdot \text{mst}(G)$. To see we have achieved the desired runtime, it remains to note that $|E| \geq \frac{1}{2}|V|$ as G is connected (unless $|V| = 1$).

Guidelines for correction:

The following elements are important in this exercise:

- In part (a), note that $(V, E(P))$ is connected, and thus has a spanning tree.
- In part (b), note that all vertices have even degree.
- In part (b), justify that we may conclude that H has a Eulerian tour (even though we have only seen the equivalence with even degree vertices for simple graphs).
- In part (c), correctly put together the previous parts to construct the algorithm.

If at least 2 of these items are present, award 1/2 point. If 4 of them are present, award 1 point. Do not subtract points in part (c) for failing to note that $|E| \geq \frac{1}{2}|V|$ only when $|V| > 1$.

Exercise 12.5 *Maximum Spanning Trees and Trucking.*

We start with a few questions about **maximum spanning trees**.

(a) How would you find the **maximum** spanning tree in a weighted graph G? Briefly explain an algorithm with runtime $O((|V| + |E|) \log |V|)$.

Solution:

We simply take any MST algorithm (e.g., Boruvka, Prim, or Kruskal) and replace all the $\min$s with $\max$s. Specifically: in Boruvka, we will find the maximum-weight outgoing edge from each connected component ("ZHK" from the lecture); in Prim, we will extract-max (instead of extract-min), use $\max$ to update weights, and use increase-key; in Kruskal, we will sort in decreasing order. The correctness arguments do not change (except for replacing "minimum" with "maximum"); the same $O((|V| + |E|) \log |V|)$ bound holds for runtime.

(b) Given a weighted graph $G = (V, E)$ with weights $w : E \to \mathbb{R}$, let $G_{\geq x} = (V, \{e \in E \mid w(e) \geq x\})$ be the subgraph where we only preserve edges of weight x or more. Prove that for every $s \in V$,

$t \in V$, $x \in \mathbb{R}$, if s and t are connected in $G_{\geq x}$ then they will also be connected in $T_{\geq x}$, where T is the maximum spanning tree of G.

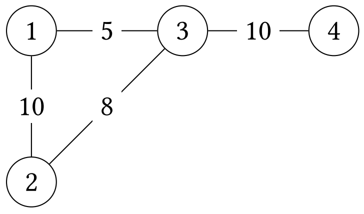
Hint: *Use Kruskal's algorithm as inspiration for the proof.*
Hint: *If it helps, you can assume all edges have distinct weight and only prove the claim for that case.*

Solution:

As argued in part (a), the maximum spanning tree is obtained by running Kruskal's algorithm that sorts the edges by decreasing weight, hence edges of $G_{\geq x}$ will be processed strictly before all of $G_{<x} := G \backslash G_{\geq x}$. Furthermore, Kruskal's algorithm only removes an edge if it would create a cycle, which does not affect connectivity. Hence, any pair $s, t \in V$ that was connected in $G_{\geq x}$ will still be connected in the maximum spanning tree using edges of weight at least x. In other words, s and t will be connected in $T_{\geq x}$, as needed.

Problem: You are starting a truck company in a graph $G = (V, E)$ with $V = \{1, 2, \dots, n\}$. Your headquarters are in vertex 1 and your goal is to deliver the maximum amount of cargo to a destination $t \in V$ in a single trip. Due to local laws, each road $e \in E$ has a maximum amount of cargo your truck can be loaded with while traversing e. Find the maximum amount of cargo you can deliver for each $t \in V$ with an algorithm that runs in $O((|V| + |E|) \log |V|)$ time. For the purpose of this exercise you can assume that your truck has unlimited capacity.

Example:



```
Output:
Max cargo to 1 is ∞
Max cargo to 2 is 10
Max cargo to 3 is 8
Max cargo to 4 is 8
```

Explanation:
The best path from the headquarters to 4 is $1 \to 2 \to 3 \to 4$, and the maximum cargo the truck can carry is $\min(10, 8, 10) = 8$.

(c) Prove that for every $t \in V$, the optimal route is to take the unique path in the **maximum** spanning tree of G.

Hint: *Suppose that the largest amount of cargo we can carry from 1 to t in G (i.e., the correct result) is OPT and let ALG be the largest amount of cargo from 1 to t in the maximum spanning tree. We need to prove two directions: $OPT \leq ALG$ and $OPT \geq ALG$.*
Hint: *One direction holds trivially as any spanning tree is a subgraph. For the other direction, use part (b).*

Solution:

Suppose that the largest amount of cargo we can carry from 1 to t in G (i.e., the correct result) is OPT and let ALG be the largest amount of cargo from 1 to t in the maximum spanning tree.

Direction $ALG \geq OPT$. By definition of OPT, there exists a path from 1 to t where all edges have weight $w(e) \geq OPT$. In other words, 1 and t are connected via $G_{\geq OPT}$. By part (b), they will also be connected in $T_{\geq OPT}$, where T is the maximum spanning tree of G. Hence, there is a path in T between 1 and t where all edges have weight $w(e) \geq OPT$. We conclude that $ALG \geq OPT$.

Direction $ALG \leq OPT$. Since any spanning tree is a subgraph of the original graph and no solution in a subgraph can be larger than in G, we conclude that $ALG \leq OPT$.

(d) Write the pseudocode of an algorithm that computes the output for all $t \in V$. The runtime of your algorithm should be $O((|V| + |E|) \log |V|)$. You can assume that you have access to a function that computes the maximum spanning tree from G and outputs it in any standard format. Briefly explain why the runtime bound holds.

Solution:

Algorithm 1

```
Input: graph $G$, given as $n \geq 1$ and an adjacency list $adj$ of $(\text{neighbor}, \text{weight})$ pairs.
Global variable: $marked[1 \dots n]$, initialized to $[False, False, \dots, False]$.

function $DFS(u, capacity)$                      ▷ we can reach $u$ with a truck of $capacity$
    Print("Max cargo to ", $u$, " is ", $capacity$)
    $marked[u] \leftarrow True$
    for each neighbor $(v, w) \in adj[u]$ do           ▷ edge $u \to v$ has weight $w$
        if not $marked[v]$ then
            $DFS(v, \min(capacity, w))$

$adj \leftarrow MaximumSpanningTree(G)$          ▷ We replace $G$ with its maximum spanning tree.
$DFS(1, \infty)$
```

The runtime of $MaximumSpanningTree(G)$ is $O((|V| + |E|) \log |V|)$ and the DFS runtime is $O(|V| + |E|)$. In total, we have a runtime of $O((|V| + |E|) \log |V|)$.