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

9 December 2024

Algorithms & Data Structures — Exercise sheet 12 — HS 24

The solutions for this sheet are submitted on Moodle until 15 December 2024, 23:59.

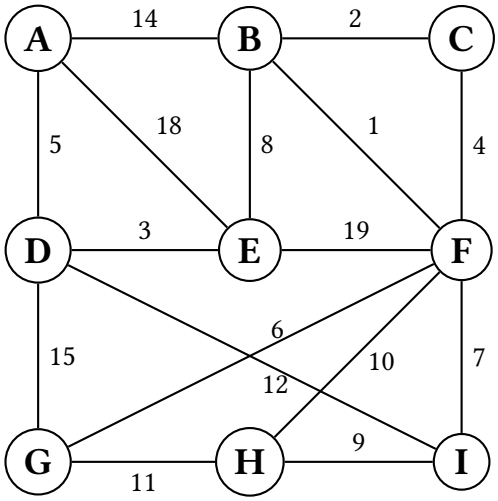
Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

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

Consider the following graph.

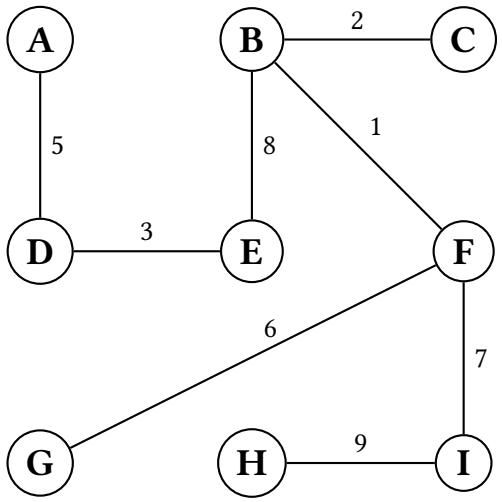


(a) Compute the minimum spanning tree (MST) using Boruvka's algorithm. For each step, provide the set of edges that are added to the MST.

Solution:

In the first step we add the edges $\{A, D\}$ (for A), $\{B, F\}$ (for B and F), $\{B, C\}$ (for C), $\{D, E\}$ (for D and E), $\{F, G\}$ (for F and G), $\{H, I\}$ (for H) and $\{F, I\}$ (for I). This gives components $\{A, D, E\}$ and $\{B, C, F, G, H, I\}$. In the second step, we add the edge $\{B, E\}$.

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:
$\{B, F\}, \{B, C\}, \{D, E\}, \{A, D\}, \{F, G\}, \{F, I\}, \{B, E\}$ and $\{H, I\}$.

(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:
$\{G, F\}, \{B, F\}, \{B, C\}, \{F, I\}, \{B, E\}, \{D, E\}, \{A, D\}$ and $\{H, I\}$.

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 *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 is 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 direct Fibre 2.0 links ($k < n$), 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 $e_1, \ldots, e_k$ be the k edges corresponding to the Fibre 2.0 links that will be donated. Note that $e_1, \ldots, e_k$ may or may not be in E.

Let $F = T \setminus \{e_1, \ldots, e_k\}$ 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.

We briefly sketch how this proof of correctness works. To this end, recall that correctness of the classical Kruskal's algorithm can be shown by proving the invariant that the current solution is always a subset of some MST. In our case we can instead use the invariant that the current solution is always a subset of some minimum spanning $(k+1)$-forest. We can then (as usual) prove this invariant by induction: At the beginning (i.e. when our solution contains no edges), the invariant is met because the empty set is a subset of every minimum spanning $(k+1)$-forest. For the inductive step we note that by induction hypothesis there is some minimum spanning $(k+1)$-forest F that contains our current solution S and we wish to show that this remains true when adding a minimum weight edge that does not induce a cycle. For this distinguish two cases.

1. The added edge e is part of F. In this case we are done since F still contains our current solution.
2. The added edge e is not part of F. In this case we consider the graph F' resulting from adding e to F. Again there are two subcases: Either F' is still a forest, in this case we can delete some arbitrary edge in $F \setminus S$ (i.e. one of the edges in F outside of the components of S) of which we know that it has weight at most as large as e since e is a minimum weight edge. Otherwise F' contains a cycle, then this cycle contains some other edge connecting two components of S which we can delete and we again know that its weight is at most as large as that of e since e is a minimum weight edge. Using this procedure we constructed a minimum weight $(k+1)$-forest containing $S \cup \{e\}$.

Exercise 12.3 *Exploring connectivity of MSTs* **(1 point)**.

In this exercise, we explore connectivity properties of the set of spanning trees and MSTs of a graph using only 'local' changes. First we prove whats called the symmetric basis exchange property. Let $G = (V, E)$ be a connected graph and $w : E \to \mathbb{R}_{\geq 0}$ be a weight function.

(a) Let T_1 and T_2 be two different spanning trees of G and let $e \in T_1 \setminus T_2$. Show that there exists an edge $f \in T_2 \setminus T_1$ such that $(T_1 \setminus \{e\}) \cup \{f\}$ and $(T_2 \setminus \{f\}) \cup \{e\}$ are both spanning trees.

Solution:

Removing $e = (u, v)$ from T_1 splits the tree into two disconnected subtrees, one connected to u and one connected to v. This follows because paths between vertices in trees are unique. So for a vertex x, consider the unique path to u and the unique path to v, at least one of them must not use e in the path and this is will be the vertex its still connected to. So let S be all vertices connected to u and $\bar{S}$ be all vertices connected to v.

Adding e to T_2 creates a cycle C since the graph is still connected and trees on the same graph must have the same number of edges and we have added an edge. Furthermore C includes the edge e. Say the cycle looks like $C = (v, u, \ldots, v)$, so we start and end at v. Since we go from $\bar{S}$ to S in the first edge e and the cycle must end in $\bar{S}$, this means that at some point along the cycle we take an edge $f = (w, x)$ which has $w \in S$ and $x \in \bar{S}$.

So $T_1' = (T_1 \setminus \{e\}) \cup \{f\}$ is connected because S and $\bar{S}$ are both connected subtrees and f has both endpoints on either side, so the whole is connected. Therefore as $|T_1'| = |T_1|$ and we are connected, T_1' is a spanning tree. For $T_2' = (T_2 \setminus \{f\}) \cup \{e\}$, since f was part of the cycle when we included e, any path between two vertices in T_2 previously can replace any use of f with the rest of the cycle C. This then creates a valid path between any two vertices in T_2' and so its connected. Since $|T_2'| = |T_2|$ and its connected, T_2' is also a spanning tree.

Now consider the graph $H = (\mathcal{B}, \mathcal{E})$ where each vertex of H corresponds to a spanning tree of G and we assign an edge between two vertices of H if their corresponding spanning trees differ by exactly two edges.

(b) Show that the graph H is connected.

Hint: *Repeatedly apply part (a).*

Solution:

We show that for any two spanning trees T_1 and T_2 of G that there is a path that connects them in H. We do this by induction over the size of the difference, we define as $|T_1 \setminus T_2|$.

Note that since $|T_1| = |T_2|$, we have that $|T_1 \setminus T_2| = |T_2 \setminus T_1|$ so the ordering of the set difference doesn't matter for this problem and therefore we use this notion of 'difference' for the solution.

Base case: For the base case $k = 0$, it must be that $T_1 = T_2$ and thus the trivial path $P = (T_1)$ works.

Induction hypothesis: Suppose that there is a path in H for between any two spanning trees with difference k.

Induction step: We now show for two spanning trees T_1 and T_2 of difference $k+1$, there is a path between them in H. Pick an edge $e \in T_1 \setminus T_2$. By part (a), we know there exists an edge $f \in T_2 \setminus T_1$ such that $T_1' = (T_1 \setminus \{e\}) \cup \{f\}$ is a spanning tree. Since the difference of T_1 and T_1' is one element, $(T_1, T_1') \in \mathcal{E}$.

Also we have $|T_1' \setminus T_2| = k$ since we removed e, so we have $|(T_1 \setminus \{e\}) \setminus T_2| = k$, and because f is in both sets, adding f to $T_1 \setminus \{e\}$ while also performing $\cdots \setminus T_2$ keeps the difference at k elements. By the inductive hypothesis there is a path $P = (T_1', \ldots, T_2)$ in H, and appending T_1 to the start of this list we get a full path $P' = (T_1, T_1', \ldots, T_2)$ in H. Thus T_1 and T_2 are connected by a path.

By induction there is a path between any two spanning trees where their difference is any integer $k \geq 0$, and therefore it holds for any two spanning trees. Thus H is connected.

(c) Consider the subgraph of H, H_{MST}, whose vertices are all MSTs of G and we keep an edge between two vertices if, again, the corresponding MSTs differ by two edges. Show that H_{MST} is connected.

Hint: *Reuse the proof for (b) but also analyze the weights of the new spanning trees produced by (a).*

Solution:

We repeat the proof of part (b) but replace any spanning trees with MSTs and use the full power of part (a).

Base case: For the base case again, any two MSTs with difference $k = 0$ must be the same MST hence the trivial path works.

Induction hypothesis: Suppose there is a path in H_{MST} between any two MSTs with difference k.

Induction step: We now show for two MSTs T_1 and T_2 of difference $k+1$, there is a path between them in H. Pick an edge $e \in T_1 \setminus T_2$. Then there exists an edge $f \in T_2 \setminus T_1$ such that $T_1' = (T_1 \setminus \{e\}) \cup \{f\}$ and $T_2' = (T_2 \setminus \{f\}) \cup \{e\}$ are both spanning trees. By definition of MST, we have that $w(T_1) \leq w(T_1') = w(T_1) - w(e) + w(f)$ so rearranging we get $w(e) \leq w(f)$. Similarly $w(T_2) \leq w(T_2') = w(T_2) - w(f) + w(e)$ and hence $w(f) \leq w(e)$.

Combining these two inequalities means that $w(e) = w(f)$ and therefore $w(T_1') = w(T_1)$ and we conclude that T_1' is an MST as well. Again, since $|T_1' \setminus T_2| = k$, there is a path $P = (T_1', \ldots, T_2)$ in H_{MST} and thus appending on the edge (T_1, T_1'), we get a path from $(T_1, \ldots, T_2)$.

By induction there is a path between any two MSTs where their difference is any integer $k \geq 0$, and therefore it holds for any two MSTs. Thus H_{MST} is connected.

Guidelines for correction:

Award $1/2$ point if part a is solved correctly. Award another $1/2$ point if parts b and c are correct.

Exercise 12.4 *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 $G = (V, E)$? Describe 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.

Another valid solution would be to note that all our MST algorithms work just as well if our graph contains negative edge weights, so we can multiply all edge weights in our graph by -1 and then run any MST algorithm (Kruskal, Prim, Boruvka). It is easy to see that any maximum spanning tree in the original graph is now a minimum spanning tree in the modified graph.

An important note is further that inverting edge weights and working with negative edge weights in general is fine when it comes to computing minimum (or maximum) spanning trees, but this is not true for shortest path algorithms as e.g. Dijkstra's algorithm requires the assumption of non-negative edge weights and stops being correct if this assumption is violated.

(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:

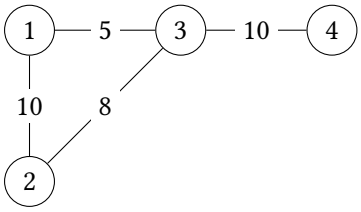
As argued in part (a), the maximum spanning tree is obtained by running Kruskal's algorithm that sorts the edges by decreasing weight, hence edges of $G_{\geq x}$ will be processed strictly before all of $G_{<x} := G \setminus G_{\geq x}$. Furthermore, Kruskal's algorithm only removes an edge if it would 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.

For the sake of completeness, we give a second, alternative proof that does not rely on Kruskal's (or any other) algorithm. Our proof works by contradiction. To this end, we assume that s, t are not connected in $T_{\geq x}$ but they are connected in $G_{\geq x}$. Since T as a tree is a connected graph, there exists some path P in T connecting s and t. This path contains at least one edge e of weight strictly less than x as otherwise this would contradict our assumption. Removing e from T splits the tree into two components C_1, C_2 such that s is in C_1 and t is in C_2.

However, since s, t are connected in $G_{\geq x}$, there is some path P' from s to t in G where all edges have weight at least x. This path contains at least one edge e' with one endpoint in C_1 and the other in C_2. Adding e' to $T \setminus \{e\}$ thus creates a tree T' which has strictly larger weight than T since $w(e) < x$ but $w(e') \geq x$. This contradicts our assumption that T was a maximum spanning tree and thus finishes our proof.

Problem: You are starting a truck company in a graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$. 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 $ALG \geq OPT$.

Note how this proof exploits the statement from task (b). It implies the (on a first glance surprising) result that for any pair of vertices s, t the unique path between s and t in a maximum spanning tree maximises the minimum edge weight among all paths that connect s to t in G. Analogously, in a minimum spanning tree, we minimise the maximum edge weight among all paths from s to t.

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 (neighbor, 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] ← True
    for each neighbor (v, w) ∈ adj[u] do     ▷ edge u → v has weight w
        if not marked[v] then
            DFS(v, min(capacity, w))

adj ← MaximumSpanningTree(G)                 ▷ We replace G with its maximum spanning tree.
DFS(1, ∞)
```

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 $O((|V| + |E|) \log |V|)$.

This algorithm applies what we learned from the previous sub-tasks: to find the maximum amount of concrete that can be moved from 1 to t, it suffices to consider the unique path from 1 to t in a maximum spanning tree. The algorithm therefore computes this tree T and then uses a DFS to compute the minimum edge weight in the unique path connecting s to t in T.

Exercise 12.5 *Heavy and light edges* **(1 point)**.

Let $G = (V, E)$ be a connected, undirected, weighted graph with positive weights $w_e > 0$ for $e \in E$. We say an edge $e \in E$ is *heavy* if there exists a cycle $C \subseteq E$ so that $e \in C$ is the (strictly) heaviest edge in C, i.e.,

$$w_e > w_f \text{ for all } f \in C \text{ with } f \neq e.$$

We say an edge is *light* if there exists a minimum spanning tree $T \subseteq E$ of G which contains e.

(a) Show that a heavy edge cannot be light.

Hint: Assume for a contradiction that $T \subseteq E$ is an MST of G and that T contains a heavy edge e. Say e is the heaviest edge in a cycle $C \subseteq E$. Construct a strictly cheaper spanning tree of G by removing e from T, and replacing it by a different edge $f \in C$.

Solution:

Let $T \subseteq E$, $e \in E$, $C \subseteq E$ be as in the hint. Let $T' = T \setminus \{e\}$. The graph $H = (V, T')$ has two connected components, say with vertex sets $V_1, V_2 \subseteq V$. Note that one endpoint of e lies in V_1 and the other lies in V_2. As e is part of the cycle C, this means that there must be an edge $f \in C$ with $f \neq e$ which also has one endpoint in V_1 and the other in V_2. Therefore, the set $T^* = T' \cup \{f\}$ is a spanning tree for G (it is connected, and it contains exactly $|V| - 1$ edges). But, since e is heavy, we have

$$w(T^*) = w(T) - w_e + w_f < w(T),$$

contradicting the fact that T was a *minimum* spanning tree.

(b*) Show that an edge which is not heavy, must be light. Conclude that an edge is heavy if and only if it is not light.

Hint: You may use without proof that Kruskal's algorithm is correct regardless of the order in which edges of equal weight are processed.

Solution:

Kruskal's algorithm relies on a sorting of the edges $E = \{e_1, e_2, \ldots, e_m\}$, where $w_{e_i} \leq w_{e_j}$ for all $1 \leq i \leq j \leq m$. It is correct for *any* sorting satisfying this condition (i.e., we are allowed to put edges of equal weight in any order). Now let $e \in E$ be an edge which is not heavy, and let $E = \{e_1, e_2, \ldots, e_m\}$ be a sorting of E with $e = e_j$ for some $1 \leq j \leq m$, and such that $e_j > e_k$ for all $1 \leq k < j$ (i.e., a sorting where e comes first among all edges with the same weight). During Kruskal's algorithm we loop through the edges in E in order, adding each edge to a set T if it does not introduce a cycle. When the algorithm terminates, T is a minimum spanning tree for G. We claim e must be in T. Let T_{j-1} be the (partial) spanning tree constructed by Kruskal's after the first $j - 1$ edges have been considered. By assumption on the order, T_{j-1} contains only edges of weight strictly less than w_e. For this reason, $T_{j-1} \cup \{e\}$ cannot contain a cycle (as this would mean e is heavy). Therefore, e is added to the (partial) spanning tree when it is considered, and in particular is present in the final tree T.

Guidelines for correction:

For this exercise (part a), four elements are important:

- Observation that $H = (V, T')$ consists of two components;
- Observation that there must be an $f \in C$ different from e which connects these two components;
- Observation that $T' \cup \{f\}$ is a spanning tree for G;
- Observation that $T' \cup \{f\}$ has strictly smaller weight than T.

Award 1 point if all elements are present, award 1/2 point if at least 3 are present.