CSCI 270 Problem Set 7 Solutions

Fall 2025

Problem 1. *Let $G = (V, E)$ be a connected graph in which all edge weights are distinct. Prove that G has a unique MST.*

Solution. We proved the cut property in class: if $(S, \overline{S})$ is any cut of G and e_S is the unique cheapest edge crossing $(S, \overline{S})$, then *every* MST for G contains e_S. Now let

$$T = \{e_S : S \subseteq V, S \neq \varnothing\}.$$

Note that e_S exists (uniquely) for each $S \subseteq V$ owing to connectedness of G and distinctness of edge weights. Clearly (V, T) is connected; for any $A \subseteq V$, it cannot be that A is disconnected from $\overline{A}$, as $e_A \in T$. Furthermore, for any MST T', we have that $T \subseteq T'$ by the cut property. And it cannot be that $T \subsetneq T'$, otherwise T' has strictly greater cost than T. Thus T is the unique MST. □

Problem 2. *Given a graph in which all edge weights are distinct, design an algorithm to find the second minimum spanning tree. That is, a spanning tree whose total cost is greater than that of the minimum spanning tree but smaller or equal than that of any other spanning tree. Please provide your pseudocode, proof of correctness, and running time analysis.*

Solution. Let $G = (V, E)$ be the underlying graph, with $T \subseteq E$ the MST and $T' \subseteq E$ the second-minimal spanning tree. First note that $T' = T \cup \{e'\} \setminus \{e\}$ for some $e' \in T' \setminus T$ and $e \in T$. In particular, if T' disagreed with T by more than 1 "edge swap", it could have its edges repeatedly swapped with those of T until only 1 difference remains (by Problem 1 of HW 6), which would reduce the cost of T'.

Furthermore, for any fixed $e' \notin T$, $T \cup \{e'\}$ contains a unique cycle C containing e. Removing any edge $e \in C \cap T$ again produces a spanning tree $T \cup \{e'\} \setminus \{e\}$, and these account for all spanning trees of the form $T \cup \{e'\} \setminus \{e\}$ once e' is fixed. Clearly, the minimum-cost such spanning tree is the one which removes the edge $e \in C \cap T$ of greatest cost.

Then, in order to compute the second-minimum spanning tree, it suffices to: 1) Compute the MST, T e.g., via Kruskal's algorithm; 2) Iterate through all edges $e' \in E \setminus T$, find the cycle C in $T \cup \{e'\}$, and record the weight of $T \cup \{e'\} \setminus \{e\}$, where e is the most expensive edge in $C \cap T$; 3) Return the minimum weight observed in Step 2.

This is formalized by the following pseudocode.

```
def second_MST(G):
    T = Kruskal(G)
    w_T = sum(w(e) for e in T.edges)
    best = float('inf')

    for e in G.edges - T.edges:
        f = max_edge_in_cycle(T, e)
        best = min(best, w_T + w(e) - w(f))
    return best
```

Here `Kruskal` refers to Kruskal's algorithm and `max_edge_in_cycle` computes the highest-weight edge other than e in the unique cycle contained in $T \cup \{e\}$. This can be implemented using, e.g., DFS.

The runtime of the algorithm is $O(mn+m^2)$, as Kruskal's algorithm runs in time $O(m \log m) \subseteq O(m^2)$, and the `for` loop executes $O(m)$ times at a cost of $O(m+n)$ for each run of DFS.[1] □

Problem 3. *Decide whether you think the following statement is true or false. If true, provide a proof; if false, give a counterexample.*

Let G be a flow network that contains a path from s to t. For every maximum flow f on G, there exists an edge e such that $f(e) = c_e$.

Proof. The statement is true. Let f be a flow in G which does *not* saturate any edge; we will demonstrate that there exists a flow f' with $v(f') > v(f)$. To this end, let $P = (e_1, \ldots, e_\ell)$ be an $s \to t$ path in G. By assumption, $f(e_i) < c_{e_i}$ for all $i \in [\ell]$. Let $\delta = \min_{i \in [\ell]} c_{e_i} - f(e_i)$. As f does not saturate any edges, $\delta > 0$. Then consider the flow $f' = f + \delta \cdot P$. As f' does not exceed any capacity constraints, by definition of δ, it is indeed a valid flow. And

$$v(f') = v(f) + v(\delta \cdot P) = v(f) + \delta > v(f),$$

completing the argument. □

[1]It's actually possible to achieve a much better time of $O(m \log m)$ by being more clever in the `for` loop and avoiding a fresh DFS each iteration. In particular, each `max_edge_in_cycle` query can be solved in $O(\log n)$ time after a one-time $O(n \log n)$ preprocessing operation such as *binary lifting*. You don't need to know that for this class, but we want you to be aware that a faster solution exists!