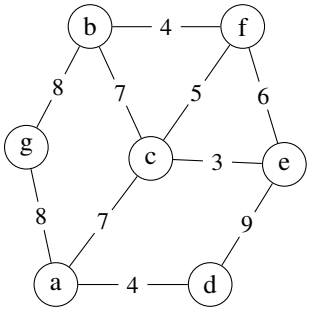
CMPSC 465 Data Structures & Algorithms
Fall 2025 Ke Chen and Yana Safonova

HW 8 Solution

1. (20 pts.) **Find MST.** Consider the following graph:



(a) Run Kruskal's algorithm on the graph given below: give the order of edges that are added to the MST (whenever you have a choice, always choose the smallest edge in *lexicographic* order).

(b) Run Prim's algorithm on the graph given below: give the order of vertices that are added to the MST (whenever you have a choice, always choose the smallest vertex in *lexicographic* order).

Solution:

(a) Edges are added in this order: $(c,e),(a,d),(b,f),(c,f),(a,c),(a,g)$.

(b) Vertices are added in this order: (a,d,c,e,f,b,g). The edges in the MST are: $(a,d),(a,c),(a,g),(c,e),(c,f),(b,f)$.

2. (20 pts.) **Max Subgraph.** Let $G=(V,E)$ be an undirected graph. A graph $G'=(V',E')$ is a *subgraph* of G if $V'\subseteq V$ and $E'\subseteq E$. The *degree* of a node in a graph is the number of edges incident to it. Give an algorithm to find a largest subgraph (i.e. a subgraph with as many vertices as possible) G' of G such that every node in G' has degree (in G') at least k. Your algorithm should run in time $O(|V|+|E|)$.

Solution:

If a node has degree less than k, then neither it nor its edges can be included in G'. Thus, if v has degree less than k, then $G'\subseteq G-v$. This implies the correctness of the following "greedy" algorithm: while G has a node v of degree less than k, set $G:=G-v$ and repeat.

To implement this, we can create and maintain a list of nodes with degree less than k, and also keep track of the degrees of every node (not just in that list). While the list is not empty, take an element from the list, remove it from the graph, subtract one from the degree of all of its neighbors, and if any of them now have degree less than k, add them to the list. When the list is empty, G' comprises the nodes remaining in the graph and the edges between them in E.

This algorithm takes $O(|V|+|E|)$ steps, since each vertex is examined at most once, and there may be a number of subtractions equal to the number of edges.

3. (20 pts.) **Strange MST.** Let $G=(V,E)$ be a connected, undirected graph with edge weights $w:E\to\mathbb{R}$. Each edge $e\in E$ is labeled as either *mandatory*, *forbidden*, or *optional*. We wish to determine whether there exists a spanning tree $T\subseteq E$ that includes all mandatory edges, excludes all forbidden edges, and connects all vertices without forming a cycle.

(a) Describe how the Disjoint Set Union (DSU) structure can be used to check whether *mandatory* edges form any cycle or not.

(b) Modify Kruskal's algorithm to construct the minimum-cost feasible spanning tree if it exists, and analyze the correctness and runtime.

Solution.

(a) To test for cycles, we initialize a DSU over V and process each mandatory edge $(u,v) \in E_M$. If $\text{Find}(u) = \text{Find}(v)$, the mandatory edges form a cycle and no feasible spanning tree exists.

(b) Initally we check for cycles based on (a). Then, if it does not contain any cycle among the mandatory vertices, we perform $\text{Union}(u,v)$, merging the components. After processing all mandatory edges, if no cycle is found, the partial forest induced by E_M is acyclic.

To construct the feasible minimum spanning tree, we run Kruskal's algorithm beginning from this partial forest. All forbidden edges E_F are ignored, and the remaining optional edges E_O are sorted by non-decreasing weight. Edges $(u,v,w) \in E_O$ are processed in order; each is added if and only if $\text{Find}(u) \neq \text{Find}(v)$. The algorithm halts once all vertices are connected.

Algorithm 1 STRANGEKRUSKAL($G = (V, E_M, E_O, E_F)$)

```
Initialize DSU on V
F = ∅
for each (u, v) ∈ E_M do
    if Find(u) = Find(v) then
        return Infeasible
    end if
    Union(u, v)
    F = F ∪ (u, v)
end for
Sort E_O by non-decreasing w
for each (u, v, w) ∈ E_O do
    if Find(u) ≠ Find(v) then
        Union(u, v)
        F = F ∪ (u, v)
    end if
end for
return F
```

The algorithm is correct because the DSU ensures acyclicity, and the cut and cycle properties of Kruskal's algorithm remain valid when initialized with an acyclic mandatory set. If E_M contains a cycle, any spanning tree containing all mandatory edges must also contain that cycle, violating feasibility. Otherwise, the algorithm extends E_M with the lightest feasible edges until full connectivity is achieved, yielding a minimum-cost feasible spanning tree. The runtime is dominated by sorting, giving $O(m \log m)$ overall, with DSU operations contributing only $O(m\alpha(n))$ amortized time.

4. (20 pts.) **Linear Transformation.** Let $G = (V,E)$ be a connected, undirected graph with edge weights $w : E \to \mathbb{R}_{>0}$, and let T be a minimum spanning tree of G. Suppose each edge weight $w(e)$ is modified according to a linear function

$$w'(e) = a \cdot w(e) + b$$

for constants $a > 0$ and $b \geq 0$.

(a) Prove or disprove: T remains a minimum spanning tree of G under the new weights w'.

(b) Generalize your argument to the case where $a > 0$ but b can be any real number (possibly negative).

Solution.

(a) Since $a > 0$ and $b \geq 0$, the transformation $w'(e) = aw(e) + b$ is *monotone increasing* in $w(e)$. Formally, for any two edges $e_1, e_2 \in E$:

$$w(e_1) < w(e_2) \implies w'(e_1) = aw(e_1) + b < aw(e_2) + b = w'(e_2).$$

Minimum spanning tree algorithms such as Kruskal's or Prim's select edges based on their relative weights. Because the relative order of edges is preserved under this linear transformation, the same edges will be chosen in the MST. Hence, T remains a minimum spanning tree under the new weights w'.

(b) If b can be any real number (possibly negative) but $a > 0$, the transformation is still monotone increasing:

$$w(e_1) < w(e_2) \implies w'(e_1) = aw(e_1) + b < aw(e_2) + b = w'(e_2).$$

The argument from part (a) still applies. Therefore, T remains a minimum spanning tree even if $b < 0$, as long as the slope a is positive. Any linear transformation with a positive slope ($a > 0$) preserves the MST, regardless of the value of b. Only if $a < 0$ could the MST potentially change.

5. (20 pts.) **Dynamic MST.** Let $G = (V, E)$ be a connected, undirected graph with edge weights $w : E \to \mathbb{R}$, and let T be a minimum spanning tree of G. Assume all edge weights are distinct.

(a) Suppose a new edge $e' = (u, v)$ with weight $w'(e')$ is added to G. Describe an efficient method to decide whether T remains the MST.

(b) Suppose the weight of an existing edge e is increased from $w(e)$ to $w'(e)$. How will the MST change?

Solution.

(a) Adding e' to T forms a unique cycle C in $T \cup \{e'\}$. Let f be the edge with maximum weight on C. If $w'(e') \geq w(f)$, then T remains the MST; otherwise, T is no longer minimal. This can be determined by finding the path from u to v in T and identifying the maximum-weight edge.

(b) Let $e = (x, y) \in E$ be an existing edge whose weight is changed from $w(e)$ to $w'(e)$. There are two cases depending on whether $e \in T$ or not. In each case we state whether T necessarily remains an MST, when it may cease to be one, and how to update T if needed.

Case 1: $e \notin T$ Making a non-tree edge heavier cannot create a lighter spanning tree than T; thus T remains an MST. No update is necessary.

Case 2: $e \in T$ Removing e from T splits T into two components A and B with $x \in A, y \in B$. Consider all edges in $E \setminus \{e\}$ that cross the cut (A, B). Let f be the lightest such crossing edge (if any). If $w'(e) \leq w(f)$ then T (with e now having weight $w'(e)$) is still an MST. If $w'(e) > w(f)$ then e should be replaced by f: the new MST is $T_{\text{new}} = T \setminus \{e\} \cup \{f\}$.

Rubric:

Problem 1, ? pts

?

Problem 2, ? pts

?

Problem 3, ? pts

?

Problem 4, ? pts

?

Problem 5, ? pts

?