Minimum Spanning Trees—Monday, November 8 / Tuesday, November 9

Readings

- Lecture Notes Chapter 21: Minimum Spanning Trees
- Lecture Notes Chapter 22: Union Find

Review: Minimum Spanning Trees (MSTs)

A spanning tree of a graph \( G = (V, E) \) is a subgraph that is a tree and which “spans” (or includes) all vertices in \( V \). Given an undirected, connected, and weighted graph \( G \), a minimum spanning tree (MST) of \( G \) is a spanning tree whose edges have a minimum sum out of all possible spanning trees of \( G \).

Kruskal’s Algorithm

At a high-level, running Kruskal’s Algorithm on an input graph \( G = (V, E) \) finds an MST by first starting on the graph with no edges. We sort the edges in increasing order of weight and then iterate through the edges in this order, adding an edge to the MST if it does not create a cycle since we want the minimum total sum. The pseudocode is as follows:

```plaintext
Kruskal(G)
    T = ∅
    Sort the edges in \( E \) in increasing order of weight.
    for each \( v \in V \) do
        MakeSet(\( v \))
    for each \( e = (u, v) \in E \) do
        if Find(\( u \)) ≠ Find(\( v \))
        then
            \( T = T \cup \{e\} \)
            Union(\( u, v \))
    return \( T \)
```

Since Kruskal’s relies on the ability to detect cycles efficiently, we use the Union-Find data structure, where the MAKESET operation runs in \( O(1) \) time and the UNION/FIND operations run in \( O(\log n) \) time if we just use Union by Rank and in \( O(\log^* n) \) — barely more than \( O(1) \) — time if we use Path Compression with Union by Rank. If we do not know anything about the edge weights, then the runtime of Kruskal’s will be the time it takes to sort the edges, which is \( O(m \log m) = O(m \log n) \), so it technically would not matter whether or not we optimize with Path Compression since both would not exceed the overall \( O(m \log n) \) runtime. However, for example, if the edges are given to us sorted, then the UF data structure can become the bottleneck for Kruskal’s runtime, so it is important to use Path Compression then.

Non-Distinct Edge Weights

Note that in lecture, we first proved the correctness of Kruskal’s in the case where edge weights were all distinct. However, we can still extend the proof of correctness to the case where edge weights are non-distinct: we arbitrarily break ties by modifying the edge weights so that they maintain their relative ordering but
are now distinct (full details on how to do this are in Section 21.4 of the lecture notes). As you will see in some of the problems below, any modifications to edge weights that maintain their relative ordering are valid transformations. Similarly, we can modify the cut and cycle property as follows:

**Cut Property:** Let $e = (u, v)$ be a minimum cost edge “crossing the cut” with one end in $S$ and the other in $V - S$. Then some MST of $G$ contains $e$.

**Cycle Property:** Let $C$ be any cycle in $G$ and let $e = (u, v)$ be a heaviest edge in $C$. Then $e$ does not belong to some MST of $G$.

**Problems**

**Problem 1**
Does Kruskal’s algorithm work on a graph with negative weights?

**Problem 2**
Suppose we have some MST, $T$, in a graph $G$ with positive edge weights. Construct a graph $G'$ where for any weight $w(e)$ for edge $e$ in $G$, we have weights $w(e)^2$ in $G'$. Is $T$ still a MST in $G'$? Prove your answer.

If $G$ also had some negative edge weights, would your answer change from above change? Prove your answer.

**Problem 3**
Suppose we have a connected graph $G$ where all edge weights are equal. Design an efficient algorithm find an MST of $G$. What is the running time of your algorithm?

**Problem 4**
Suppose that we have an MST $T$ of a graph $G$ but are told that an edge not in $T$ has a lower weight than originally specified and so $T$ is now an invalid MST. Is it guaranteed that we can fix our tree by removing an edge and adding a different one? If so, explain how. If not, provide a counterexample.