Lab Worksheet 6: Minimum Spanning Trees

The Cycle Property

A fundamental concept in MST algorithms is the *Cycle Property*.

Cycle Property: *For any cycle C in the graph, if the weight of an edge e of C is strictly greater than the weight of any other edge in C, then e cannot belong to any Minimum Spanning Tree (MST).*

Question 1: Understanding the Cycle Property. Suppose you have an MST T. If you add any edge $e \notin T$ to T, it creates a unique cycle C.

1. Explain why the new graph $T \cup \{e\}$ contains exactly one cycle.

2. Let us prove the cycle property by contradiction. Let $e \in C$ be the edge with the maximum weight in the cycle C; that is, $w(e) > w(e')$ for all $e' \in C \setminus \{e\}$. Suppose e belongs to a minimum spanning tree T. Explain how one can construct another spanning tree T' whose total weight is strictly smaller than that of T, thereby contradicting the assumption that T is an MST.

Second Best Minimum Spanning Tree

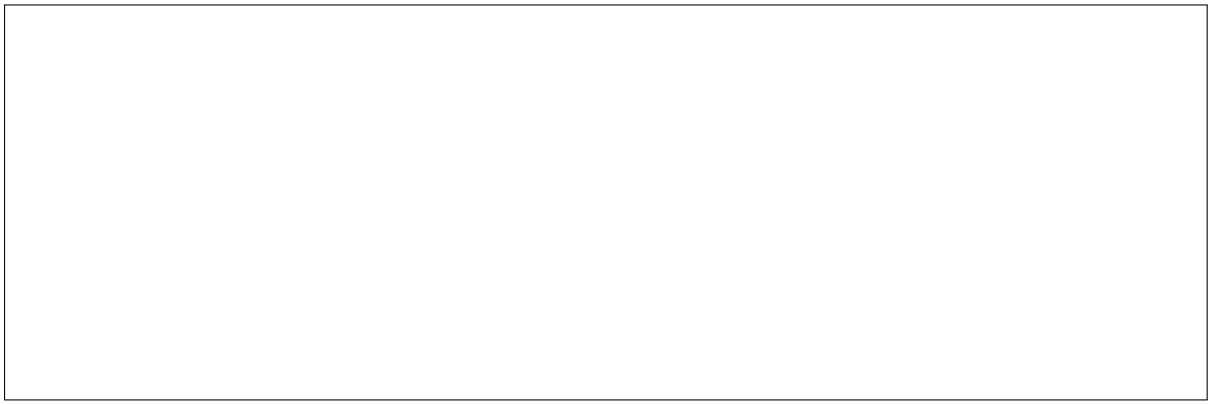
The goal of this problem is to find a spanning tree that has the second-smallest total edge weight. Before we begin, let $G = (V, E)$ be a connected, weighted, undirected graph, and let T be a minimum spanning tree (MST) of G. The weight of a spanning tree T is denoted $W(T)$. A *second best minimum spanning tree* is a spanning tree S such that $W(S) = \min\{W(T') \mid T' \text{ is a spanning tree and } W(T') > W(T)\}$. Note that even if there are multiple minimum spanning trees, none of them are considered a second best minimum spanning tree. The second best minimum spanning tree must have a weight that is strictly larger than $W(T)$.

The Second Best MST Is One-Edge-Flip Away

We now explore a crucial claim: A second best MST can always be found by taking an arbitrary MST, T, and swapping a single edge. That is, $T' \coloneqq T \setminus \{e'\} \cup \{e\}$ for some $e' \in T$ and $e \notin T$. Our goal here is to argue about the existence of these two edges e and e'.

Question 2: The Edge Exchange. Let S be a second best MST that has the **maximum overlap** with T, meaning the size of the set $T \cap S$ is maximized over all second best MSTs. Since $S \neq T$, and it has $|V| - 1$ edges, there must be an edge in S that does not belong to T. Let's take call this edge e. When you add e to the MST T, it creates a unique cycle C in $T \cup \{e\}$.

1. Prove that there must exist an edge $e' \in C$ such that $e' \in C$ such that $e' \notin S$ and adding e' to S and removing e from it creates another spanning tree.



2. Show that $T' \coloneqq T \setminus \{e'\} \cup \{e\}$ is also a spanning tree.

3. Now consider $S' \coloneqq S \setminus \{e\} \cup \{e'\}$ and $T' \coloneqq T \setminus \{e'\} \cup \{e\}$ we constructed. Our goal is to show that T' is a second best minimum spanning tree. In particular, it might be helpful to consider the following cases. Based on your answers to the earlier parts, analyze the weight of T' and S'.

   - **Case 1:** $w(e') < w(e)$**.** What is the relationship between $W(S')$ and $W(S)$? What does this say about $W(T')$?
   - **Case 2:** $w(e') = w(e)$**.** What is the relationship between $W(S')$ and $W(S)$?
   - **Case 3:** $w(e') > w(e)$**.** What is the relationship between $W(T')$ and $W(T)$?

From Edge-Flip to an Algorithm

The previous section established a crucial result: the second best minimum spanning tree is one edge-flip away from an arbitrary MST (say T). Specifically, for some edge $e = (u, v) \notin T$ and some edge $e' \in T$, the resulting tree $T' = T \setminus \{e'\} \cup \{e\}$ must be the second best MST.

There are two properties to point out about e and e'. By adding e to T, we will create a cycle C, to preserve the connectivity clearly e' must belong to this cycle. To get the second best minimum spanning tree, we need to make sure that the increase to the weight caused by this swap is minimal, but greater than zero. The weight of the new tree is:

$$W(T') = W(T) + w(e) - w(e')$$

Thus, we have $w(e') < w(e)$. And, for any given e, we will pick the edge e' with the maximum weight that is still smaller than $w(e)$.

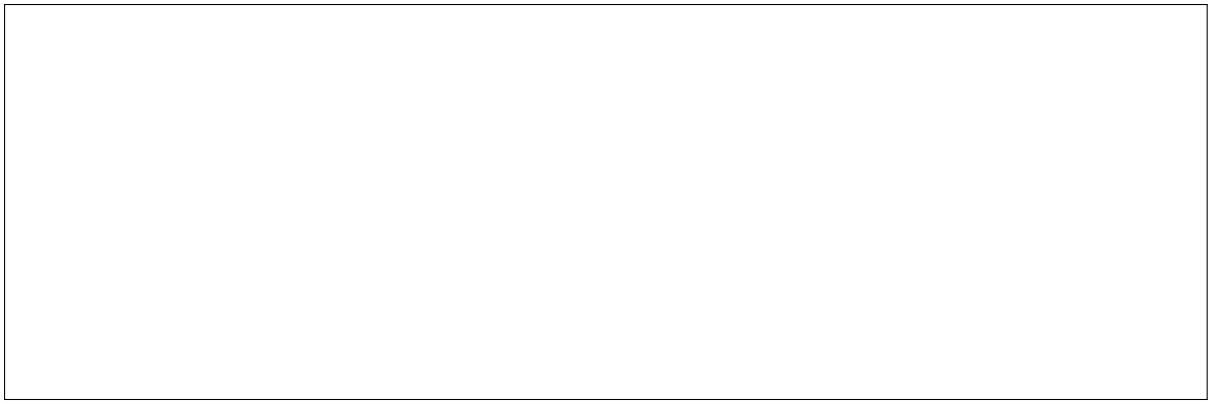
Since $W(T)$ is a constant, our objective becomes:

$$\min_{e \in E \setminus T} \left\{ w(e) - \max_{e' \in C, w(e') < w(e)} \{w(e')\} \right\}$$

where C is the unique cycle formed by $T \cup \{e\}$, and $e' \in C \setminus \{e\}$ with weight $w(e') < w(e)$.

Question 3: Finding the Optimal Edge to Remove (Brute-Force Approach). For a fixed non-tree edge $e = (u, v) \in E \setminus T$, we want to find the tree edge e' with the constrained mentioned above.

1. For a fixed input edge $e = (u, v)$, describe an algorithm that can be used to find the corresponding e'. What is the time complexity of this traversal?



2. What is the overall time complexity of finding the second best MST if we repeat the traversal described above for every non-tree edge $e \in E \setminus T$?

Path to the Max Edge: The LCA Connection

Assumption: For the remainder of this problem, let's assume all edges in the graph have *distinct weights*. This assumption simplifies the analysis by ensuring $w(e') \neq w(e)$. In fact, we can show that finding the desired e' as stated before can be replaced by finding the first (or lowest) common ancestor of the endpoints of $e = (u, v)$ and taking the maximum edge on the paths from u and v to this ancestor.

Question 4: Simplified Goal The maximum weight edge on the path $P_{u,v}$ is equal to the maximum of the maximum weight edges on the two paths leading from u and v up to their Lowest Common Ancestor.

$$\underset{e' \in P_{u,v}, w(e') < w(e)}{\arg\max} \{w(e')\} = \underset{e' \in P_{u,\mathrm{LCA}(u,v)} \cup P_{v,\mathrm{LCA}(u,v)}}{\arg\max} \{w(e')\}.$$

Figure 1 illustrates the LCA and $P_{u,v}$.

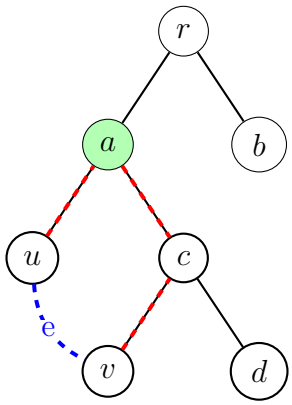


Figure 1: The path between u and v is shown by dashed red lines. The node a is the lowest common ancestor of u and v.

Binary Lifting for Efficiency

The Binary Lifting technique allows us to answer LCA and path-max queries efficiently. We precalculate information for "jumps" of size 2^k in the tree. Let $L = \lceil \log_2 |V| \rceil$. We compute the following metrics for every node u and every power k from 0 to L:

1. Depth ($d[u]$): The distance (number of edges) from the root r to u.
2. Ancestor Array ($p[u][k]$): The 2^k-th ancestor of u.
3. Maximum Weight Array ($\text{max\_w}[u][k]$): The maximum edge weight on the path from u up to its 2^k-th ancestor, $p[u][k]$.
4. Maximum Weight Edge Array ($\text{max\_e}[u][k]$): The edge that attains the maximum stored in $\text{max\_w}[u][k]$.

Question 5: Preprocessing and Initial Calculation. Describe a dynamic programming algorithm that can compute the above values. Your algorithm should run in time $O(|V| \log |V|)$ time.

Question 6: Finding the Lowest Common Ancestor (LCA). Given the values defined earlier, design an algorithm that finds $\text{LCA}(u, v)$ in $O(\log |V|)$ time.

Question 7: Efficiently Finding the Maximum Edge. Describe a procedure to find the maximum edge to the common ancestor in $O(\log |V|)$.

Question 8: Time Complexity. Given the efficiency of Binary Lifting, what is the overall time complexity of the final algorithm for finding the second best MST?