COMP 382: Reasoning about algorithms **Fall 2025**

October 30, 2025 *Instructors: Aliakbarpour, Chida, Mamouras*

Lab Worksheet 6: Minimum Spanning Trees (Solution)

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.

   Solution: A spanning tree T is connected and *acyclic*. When a new edge e is added to T, the two endpoints of e are already connected by a unique simple path in T. Adding e between these two endpoints completes this path into a cycle. Since T was acyclic, no other cycles are formed.

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.

   Solution: Note that T is a spanning tree, so if we remove one edge from T, it will be split into two connected components, each containing exactly one endpoint of e.

   Now, consider the path $P \coloneqq C \setminus \{e\}$, which connects the two endpoints of e through the cycle C.

   Clearly, this path will cross the cut formed by the connected components of $T \setminus \{e\}$ at some point, say at edge e'. If we add e' to $T \setminus \{e\}$, it will reconnect the two components since it is a cut edge.

   We can use e' to perform an edge exchange. Consider

$$T' \coloneqq T \setminus \{e\} \cup \{e'\}.$$

Since e' reconnects the two connected components created by removing e, T' is connected and has $|V| - 1$ edges. Thus, it is a spanning tree.

The weight of T' is

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

Since $w(e) > w(e')$, the term $w(e') - w(e)$ is strictly negative. Therefore, $W(T') < W(T)$, which contradicts the initial assumption that T was a *minimum spanning tree*.

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, $T' := T \setminus \{e'\} \cup \{e\}$ 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.

   Solution: Assume that e is an edge between two vertices u and v. Since $e \notin T$ and T is connected, there must exist a path P between u and v within T. Note that e does not belong to this path.

   Now, since S is a spanning tree, removing e from S will split it into two connected components: one containing u and the other containing v. This naturally defines a cut on G. The path P connects a vertex from one component to a vertex in the other, so it must cross this cut at some point. Hence, there exists a cut edge e' on this path.

Adding e' to $S \setminus \{e\}$ reconnects the two components, since e' crosses the cut. Therefore, the graph obtained by removing e and adding e' is indeed a spanning tree.

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

   Solution: By our construction, e' and e belong to cycle C, Removing any single edge from a cycle leaves the two endpoints of that edge still connected by the remainder of the cycle. Therefore, removing e' from T and adding e (the other path in C) keeps the graph connected. Since $|T'| = |T| = |V| - 1$, T' is 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 $W(T)$?

   Solution: We consider each case separately:

   - **Case 1:** $w(e') < w(e)$**.** In Since $w(e') < w(e)$, we have

     $$W(S') = W(S) + (w(e') - w(e)) < W(S)\,.$$

     Since S is a second best MST, $W(S)$ is the second smallest weight. S' is a spanning tree with a smaller weight than S, so it must be the case that $W(S')$ is an MST weight, i.e., $W(S') = W(T)$. Now, we can directly calculate the weight of T' and show it is equal to the weight of S'.

     $$W(T') = W(T) + w(e) - w(e') = W(S') + w(e) - w(e') = W(S)\,.$$

     Hence, T' is one a second best minimum spanning tree that is one-edge-flip away from T.

   - **Case 2:** $w(e') = w(e)$**.** Since $w(e') = w(e)$, we have $W(S') = W(S) - w(e) + w(e') = W(S)$. S' is also a second best MST. We compare the overlap:

     $$\begin{aligned} T \cap S' &= T \cap (S \setminus \{e\} \cup \{e'\}) \\ &= (T \cap S) \cup \{e'\} \quad \text{since } e \notin T \text{ and } e' \in T \setminus S \end{aligned}$$

     The overlap increases: $|T \cap S'| = |T \cap S| + 1$. This *contradicts* the initial assumption that S was the second best MST with the *maximum overlap* with T, because S' has the same weight as S but a strictly larger overlap.

   - **Case 3:** $w(e') > w(e)$**.** Note that T' is a spanning tree with weight $W(T') = W(T) - w(e') + w(e)$. However, if $w(e') > w(e)$, then $W(T') < W(T)$. This directly *contradicts* the fact that T is a minimum spanning tree. This argument is, in spirit, the same as the cycle property, which states that the heavier edge in a cycle must be excluded.

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?

   Solution: We can use a traversal algorithm, like BFS or DFS, starting at vertex u to find the path to v.

   - When performing the traversal, we record the path from u to v by keeping a pointer to parent.
   - Once the path is found, we iterate over all edges in this path and keep track of the edge with the largest weight below $w(e)$, setting this as our e'.

   Since the traversal is performed on the tree T, which has $|V|$ vertices and $|V| - 1$ edges, the time complexity is $O(|V|)$.

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 $e \in E \setminus T$?

   Solution: The algorithm involves:

   - Finding the initial MST T: $O(|E| \log |E|)$.
   - Iterating over all non-tree edges: There are $O(|E|)$ such edges.
   - For each non-tree edge, performing an $O(|V|)$ path traversal.

   The dominant time complexity is therefore $O(|E| \cdot |V|)$. Since $|E|$ can be up to $O(|V|^2)$ in a dense graph, this approach can be slow, with a worst-case complexity of $O(|V|^3)$.

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.

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

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

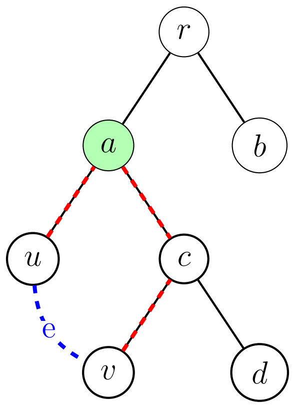

Figure 1: The path between u and v is shown by dashed red lines. The node a is the lowest common ancestor of u and v.

Solution: Note that when we add a non-tree edge $e = (u, v)$ to the MST T, it creates a unique cycle C. By the cycle property, the edge with the maximum weight in any cycle

cannot belong to the MST. Since all edges in C, except e, belong to an MST, e must be the edge with the maximum weight in the cycle C, meaning that $w(e) > w(e')$ for any $e' \in C \setminus \{e\}$. Now, to find e' (the optimal edge to remove), we have a simplified task: we only need to find the *maximum weight edge on the path between* u *and* v*, the endpoints of* e *in the tree.*

The main algorithmic challenge now is to find e' for all e efficiently. To solve this, we can root the MST T arbitrarily (say, at vertex r). The path between any two vertices u and v in T can be decomposed into the path from u to their *Lowest Common Ancestor* (LCA(u, v)) and the path from v to LCA(u, v):

$$P_{u,v} = P_{u,\mathrm{LCA}(u,v)} \cup P_{v,\mathrm{LCA}(u,v)} \,.$$

See Figure 1 for an example.

Therefore, the maximum weight edge on the path $P_{u,v}$ is simply:

$$\max_{e' \in P_{u,v}} \{w(e')\} = \max \left\{ \max_{e' \in P_{u,\mathrm{LCA}(u,v)}} \{w(e')\}, \quad \max_{e'' \in P_{v,\mathrm{LCA}(u,v)}} \{w(e'')\} \right\}$$

This insight allows us to break down the path problem into two simpler ancestor-descendant path problems.

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 (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 (max_e$[u][k]$): The edge that attains the maximum stored in 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.

Solution: A single traversal (DFS or BFS) starting at the root r is used to:

- Determine $d[u]$ for all u.

- Determine the base case values ($k = 0$): $p[u][0]$, $\text{max\_w}[u][0]$, and $\text{max\_e}[u][0]$ by determining the parent of u in the tree.

General Recurrence ($k \geq 1$): For $k \geq 1$, we compute the values by combining two jumps of size 2^{k-1}:

- Ancestor: $p[u][k] \coloneqq p[p[u][k-1]][k-1]$
- Max Weight: $\text{max\_w}[u][k] \coloneqq \max\{\text{max\_w}[u][k-1], \quad \text{max\_w}[p[u][k-1]][k-1]\}$
- Max Edge: $\text{max\_e}[u][k]$ is the edge (from the two 2^{k-1} segments) that corresponds to the $\text{max\_w}[u][k]$ value.

This preprocessing takes $O(|V|\log|V|)$ 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.

Solution:

1. **Equalize Depths:** If $d[u] \neq d[v]$, lift the deeper node (u) up by the difference $\Delta = |d[u] - d[v]|$ using binary jumps. The binary representation of Δ determines a series of jump values. For example, if $\Delta = 18 = 2^4 + 2^1$, we take the first jump to be $u \leftarrow p[u][1]$, and then $u \leftarrow p[u][4]$.

   For the rest of the algorithm, we may assume that u and v have the same depth.

2. **Check for Ancestor:** If $u = v$, then $\text{LCA}(u, v) = u$.

3. **Binary Search on Height of Common Ancestor:** We run a binary search on the height of the common ancestor in the range $[0, x]$. Initially, x can be $|V|$. We perform a simultaneous lift for both u and v of length equal to the power of two closest to $x/2$. Suppose 2^k is this closest power. Then, we check whether $p[u][k] = p[v][k]$.

   If yes, we know that the common ancestor of u and v lies in the range $[0, 2^k]$ (roughly $[0, x/2]$). We continue this until $x \leq 1$, where we can find the ancestor in $O(1)$. If no, then we set $u \leftarrow p[u][k]$ and $v \leftarrow p[v][k]$, and continue.

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

Solution: To find the maximum weight edge on $P_{u,v}$ (the path from u to v), we integrate a tracking variable into the lifting process that describe in the previous question using the max_e data structure we created earlier.

This entire query runs in $O(\log|V|)$ time.

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?

Solution:

- Step 1: MST Construction: $O(|E| \log |E|)$.
- Step 2: Binary Lifting Preprocessing: $O(|V| \log |V|)$.
- Step 3: Query all non-tree edges: For each of the $O(|E|)$ non-tree edges $e = (u, v)$, the LCA and path max-weight query takes $O(\log |V|)$. Total time for queries: $O(|E| \log |V|)$.

The overall time complexity is dominated by the slowest steps, resulting in an complexity of $O(|E| \log |V|)$.