Problem 1. Define a *tree* as a connected graph without cycles. Prove:

(i) Every tree has at least one leaf node.

(ii) Every tree with n nodes has exactly $n-1$ edges.

Proof. **Proof of the first statement.** Start from an arbitrary node u and construct a path that is as long as possible. As any tree is acyclic, such path exists and it has finite length. The endpoint must have degree 1, as for the other case we can continue to extend the path from the endpoint for a longer one, or we have encountered a loop.

Proof of the second statement. By induction, consider the predicate

$$P(n): \text{ if } G \text{ is an } n\text{-vertex tree, then it has } n-1 \text{ edges.}$$

Base Case: The case of $n=1$ is vacuously true: a tree with 1 node has no edge.

Inductive Step: Suppose $P(k)$ for any $k<n$ is true, then consider an n-vertex tree. From the first statement, there is at least one leaf node. Therefore, by removing this node u and its incident edge, we have obtained an (i) acyclic, (ii) connected graph G'. This is because removing an edge does not create cycles, and any path that starts and end from the vertices in G' is fully contained in G' due to the degree of u. Therefore, the resulting graph G' is a tree and it has $n-2$ edges. Placing the removed node and its incident edge reverts G' back to G and we finish the inductive step by arguing that G is confirmed to have $n-1$ edges. □

Problem 2. Let G be a simple graph with n vertices and $n-1$ edges. Prove: if G is connected (i.e., a path exists between any two vertices in G), then G must be a tree.

Proof. By induction, consider the predicate

$$P(n): \text{ if } G \text{ is a connected graph with } n \text{ vertices and } n-1 \text{ edges, then it is a tree.}$$

Base Case: The case of $n=1$ is vacuously true: a graph with one single node and no edge is a tree itself.

Inductive Step: Let $n \geq 2$ and suppose $P(n-1)$ is true, then consider a connected graph G with n vertices and $n-1$ edges. First, all vertices have degree at least one since the graph is connected, and there is at least one node with degree 1, otherwise we would have

$$\#\text{edges} = \frac{1}{2}\sum_{v\in V_G} d(v) \geq \frac{1}{2}\cdot 2n = n.$$

Now, we pick any node v with degree 1. The graph $G-\{v\}$ is connected and acyclic, since no path in G can pass through v without having v as one of the endpoints. Thus, $G-\{v\}$ is a tree.

By adding v and $\{v,u\}$ back, we cannot create a cycle. (Such a cycle would have to contain v, forcing it to have degree at least 2. However, v has degree 1.) Also, G is connected, so G must be a tree. □

Problem 3. Let T be a tree. Add a new edge between two vertices in T, forming a cycle *cyc*. Now, remove from G an arbitrary edge e' of *cyc*; let G' be the graph thus obtained. Prove: G' is a tree.

Proof. Assume that T has n vertices. Clearly, T and G' both have $n-1$ edges. We show that G' is connected.

Let u and v be two arbitrary vertices. Suppose that in the original tree T, u and v is connected by a path without passing across e', then the removal of e' does not affect the connectness between u and v. Otherwise, suppose that $e = \{u', v'\}$ and u goes to u', crosses e' to v', and continues to go to v. Then, we just need to show that u' is connected to v' in G. Since the edge $\{u', v'\}$ is in *cyc*, this implies that in *cyc* we can find a path from u' to v' from the other side of $\{u', v'\}$. This path must remain in G'.

We conclude that G' is a connected graph and has $n-1$ edges, which is a tree. $\square$

Problem 4. Let S be a set of integer pairs of the form (id, v). We will refer to the first field as the *id* of the pair, and the second as the *key* of the pair. Design a data structure that supports the following operations:

- `Insert`: add a new pair (id, v) to S (you can assume that S does not already have a pair with the same id).
- `Delete`: given an integer t, delete the pair (id, v) from S where $t = id$, if such a pair exists.
- `DeleteMin`: remove from S the pair with the smallest *key*, and return it.

Your structure must consume $O(n)$ space, and support all operations in $O(\log n)$ time where $n = |S|$.

Solution. Maintain two binary search trees T_1, T_2 storing the pairs indexed on id and v, respectively.

- `Insert`: Insert the pair (id, v) to both T_1, T_2.
- `Delete`: Search in T_1 in $O(\log n)$ time to remove the pair with $t = id$. If such pair (t, v) exists, then search in T_2 to remove the pair with v as its key in $O(\log n)$ time.
- `DeleteMin`: Search in T_2 in $O(\log n)$ time to find the pair with smallest key v. Report the pair and delete this pair using its id from T_1 in $O(\log n)$ time.

Problem 5. Prove: in a weighted undirected graph $G = (V, E)$ where all the edges have distinct weights, the minimum spanning tree (MST) is unique.

Proof. Let the MST generated by our algorithm T be a spanning tree with the edge set $e_1, e_2, \dots, e_{n-1}$. Suppose there is another optimal MST T' with edges $e_1^*, e_2^*, \dots, e_{n-1}^*$:

- Case 1: $e_1 \neq e_1^*$, it must be that $w_{e_1^*} > w_{e_1}$ due to the distinct weights of the edges in the graph and the minimality of e_1. Adding the edge e_1 to T' and removing an edge from the cycle created gives us a tree T'' with a smaller weight. This is a contradiction.
- Case 2: There exists an $t \geq 2$ such that $e_1 = e_1^*, \dots, e_{t-1} = e_{t-1}^*$ and $e_t \neq e_t^*$. Recall that the edges $e_1, e_2, \dots, e_{t-1}$ forms a tree, let the set of vertices in this tree be S. Then, without loss of generality, suppose that $u \in S$ and $v \notin S$. Adding the edge e_t to T' forms a cycle. Start a walk on the cycle from u and exit S by crossing e_t, reaching v, then we must enter S again by some edge $\{u', v'\}$ with $u' \notin S$ and $v' \in S$. Since the Prim's algorithm must have discovered this edge before opting for e_t, we must have $w_{e_t} \leq w_{e_t^*}$. Noting that the weights are distinct values gives us a contradiction.

□

Problem 6. Describe how to implement the Prim's algorithm on a graph $G = (V, E)$ in $O((|V| + |E|) \cdot \log|V|)$ time.

Solution. Maintain a data structure storing the node as the key, with an additional field storing an edge, which supports pushing a node to the structure, decreasing the value of a node, and popping a node in $O(\log|V|)$ time per operation. Besides, we maintain an array with length $|V|$ to check whether v is in S currently.

Step 1. Identify an edge $\{u, v\}$ with minimum weight in $O(|V|)$ time. Let $S = \{u, v\}$, $T = \{\{u, v\}\}$. For every $w \in V$, identify any edge e incident to u or v and apply `Insert`(w, e). If no such edge exists, we do `Insert`(u, nil), and consider nil as an edge with weight of ∞.

Step 2. While $V \neq S$ (or, equivalently $|V| \neq n$), we perform the following operations: (i) pop a node w that is stored with an edge e from the structure, let $S \leftarrow S \cup e$ and $T \leftarrow T \cup \{e\}$. Check if there is any cross edge $e' = \{w, x\}$ incident with e, apply the `DecreaseKey`(x, e') operation for considering the replacement of a cross edge associated with $x \notin S$.

Now we describe the implementation of the data structure. Similar to Problem 4, we utilize two binary search trees for insertion, find, deletion, and report min operations, with operation cost associated with the number of nodes in the binary search tree. The `DecreaseKey` operation with node u and an updating edge e' can be implemented by first finding the node u for its associated edge e. Then we compare the weights of the two edges: if $w_{e'} < w_e$ then we delete node u from the data structure and again, insert u to the data structure while storing e' in the additional field instead.

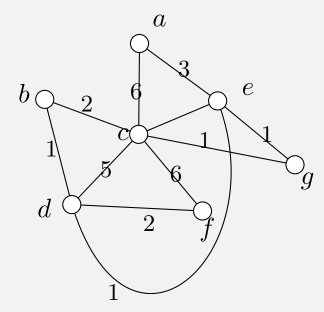
Charge the storage cost of the data structure on the number of vertices, $|V|$, in G. Clearly, all the nodes are inserted before Step 2 and thus the cost of querying is $O(\log|V|)$.

Problem 7. Let T be a tree. Prove: for any two distinct nodes u, v in the tree, there exists one and exactly one simple path from u to v (a simple path is a path where no vertex appears twice).

Proof.
(Existence:) Since T is connected, there must exist a path from u to v.

(Uniqueness:) Let the paths from u to v be $\pi_1 = (u, w_1, w_2, \ldots, w_m, v)$ and $\pi_2 = (u, z_1, z_2, \ldots, z_n, v)$. There must exist $w_i \neq z_j$, or else the two paths would be the same. Denote the first node they diverge to w_i and z_j as x, and the first node they reconverge as y. Then, we can recognize a cycle by traversing from x to y following the path in π_1 and from y to x following the path π_2 in reverse. □

Problem 8. Consider the weighted undirected graph below.



Suppose that we run Prim's algorithm to find a minimum spanning tree (MST) of this graph. Explain the order of edges picked by the algorithm.
Indicate how the cross edges change as Prim's algorithm runs.

Solution.
Step 1. Edges $\{b, d\}, \{d, e\}, \{e, g\}, \{c, g\}$ are the lightest of weight 1. We can take any of these, suppose without loss of generality we take the edge $\{b, d\}$. Now we have the cross edges $\{d, e\}, \{d, c\}, \{d, f\}, \{b, c\}$.

Step 2. We take the lightest edge available in the cross-edge set, and add it to construct a tree. Update the set of cross-edges accordingly.

Step 2(a). $\{d, e\}$ is the lightest cross-edge available. The cross edges contain $\{b, c\}, \{d, c\}, \{d, f\}, \{e, c\}, \{e, g\}, \{e, a\}$.

Step 2(b). $\{e, g\}$ is the lightest cross-edge available. The cross edges contain $\{b, c\}, \{d, c\}, \{d, f\}, \{e, c\}, \{e, a\}, \{c, g\}$.

Step 2(c). $\{c, g\}$ is the lightest cross-edge available. The cross edges contain $\{d, f\}, \{e, a\}$.

Step 2(d). $\{d, f\}$ is the lightest cross-edge available. The cross edge contains $\{e, a\}$.

Step 2(e). $\{e, a\}$ is the lightest cross-edge available. No more cross edge is available.

Problem 9. Let $G = (V, E)$ be an undirected connected graph where each edge in E is associated with a positive weight. Consider any non-empty subset $S \subset V$. An edge $\{u, v\}$ in E is an S-cross edge if $u \in S$ but $v \notin S$. Prove: if e is an S-cross edge that has the minimum weight among all S-cross edges, e must belong to some MST of G.

Proof. Take an arbitrary MST T. We just need to deal with the case that $e = \{u', v'\}$ is not an edge of T, there must be an alternative S-cross edge $\{u, v\}$ with $u \in S$ and $v \notin S$. Otherwise, S is disconnected with $V \backslash S$.

Add e to T and this forms a cycle *cyc*. Clearly, $\{u, v\}$ is on the cycle $(u - \ldots - u' - v' - \ldots - v - u)$. Thus, removing $\{u, v\}$ from the cycle results in a tree T'. As e and $\{u, v\}$ are both S-cross edges, T' must be an MST since the weight of e is the minimum among all S-cross edges. $\square$