Exercise 1. Prove that any connected graph with n >= 1 node has >= n - 1 edges. (Hint: Adding an edge can only decrease the number of connected components by 1.)

We prove by induction on the number of edges.

[Base Case] A connected graph with n = 1 has 0 edges, and 0 >= 1 - 1 = 0, so the statement holds.

[Inductive Case] Assume any connected graph with n nodes and m edges satisfies m >= n - 1. We take a connected graph G with n nodes and m + 1 edges. Remove any edge e from G.

[Case 1: G remains connected] Then, by the induction hypothesis, m >= n - 1, so m + 1 >= n.

[Case 2: G becomes disconnected] Then, G - e now has two connected components. Let them have $n_1 + n_2 = n$ nodes.

By the inductive hypothesis, the total edges in G - e is at least $(n_1 - 1) + (n_2 - 1) = n - 2$. Then, G has at least (n - 2) + 1 = n - 1edges.

Thus, any connected graph with n nodes has at least n - 1 edges.

Exercise 2. Prove that any acyclic graph with n >= 1 nodes has <= n - 1 edges. (Hint: Adding a bridge decreases the number of connected components by 1.)

We prove by induction on the number of nodes *n*.

[Base Case] A graph with n = 1 has at most 0 edges, and 0 <= n - 1 = 1 - 1 = 0, so the statement holds.

[Inductive Case] Assume any acyclic graph with fewer than n nodes has at most (n - 1) - 1 = n - 2 edges. Take an acyclic graph G with *n* nodes and *m* edges.

since G is acyclic, it has at least one leaf (a node of degree 1). Remove that leaf and its incident edge. The remaining graph has *n - 1* nodes, remains acyclic, and has *m - 1* edges.

By the inductive hypothesis, m - 1 <= (n - 1) <= n - 2

Thus, m <= n - 1.

Hence, any acyclic graph with *n* nodes has at most *n - 1* edges.

Exercise 3. Prove that a tree with $n \geq 1$ nodes has exactly $n - 1$ edges.

A tree is defined a a connected and acyclic graph. By Exercise 1, any connected graph with n nodes has at least n - 1 edges. By Exercise 2, any acyclic graph with n nodes, has at most n - 1 edges.

since a tree is both connected and acyclic, it must satisfy both inequalities simultaneously:

$$n - 1 \leq (\text{number of edges}) \leq n - 1$$

Therefore, a tree with n nodes has exactly n - 1 edges.

Exercise 4. Prove that a graph is a tree if and only if it satisfies at least two of the following:

1. **It is connected.**
2. **It is acyclic.**
3. **It has n - 1 edges.**

Note that a tree satisfies all three.

[=>] “If it satisfies two of the three properties, then the graph is a tree.”

We consider the following cases:

[Case 1: Connected and Acyclic] Then by definition, the graph is a tree, and by Exercises 1 and 2, it also has n - 1 edges.

[Case 2: Connected and n - 1 edges] suppose for contradiction, that the graph has a cycle. Removing an edge from the cycle keeps it connected, but leaves n - 2 edges, contradicting Exercise 1, which states that for a graph to be connected, it has to have at least n - 1 edges. Hence, it’s also acyclic and is also a tree.

[Case 3: Acyclic and n - 1 edges] suppose the graph is disconnected with k >= 2 components, each a tree. Let component i have n_i nodes, so the sum of n_i = n and total edges, are the sum of (n_i - 1) = n - k. But given that the graph has n - 1 edges,

$$n - k = n - 1 \Rightarrow k - 1$$

Therefore, it is actually just one component, the graph is connected.

Thus, any of the two imply the third, and a tree, which is connected and acyclic, satisfies all three.

[<=] “If the graph is a tree, then it satiefies at least two of the three properties.” By definition, a tree is a connected, acyclic graph with n nodes and n - 1 edges.

The following exercises show that there is a unique path from one node to another node in a tree.

Exercise 5. Prove that if there is a walk from x to y, then there is a path from x to y using only a subsequence of the edges of that walk.

Let W = (x = v_0, v_1, …, v_k = y) be a walk from x to y. We consider the following cases, regarding the occurrences of vertices:

[Case 1: No vertex repeated in W] If no vertex is repeated in W, then W is already a path.

[Case 2: some vertex v_i is repeated] If some vertex $v_i = v_j$ with $i < j$, we remove the cycle v_i, v_{i+1}, …, v_j from the walk to get a shorter walk from x to y, and repeat until no vertex is repeated. The result is a path from x to y whose edges are a subsequence of the edges in W.

Therefore, if there is a walk from x to y, then there is a path from x to y.

Exercise 6. Prove that if there are two distinct paths from x to y, then the graph has a cycle. (Hint: Find a nontrivial circuit first).

Let P_1 and P_2 be two distinct paths from x to y. Follow P_1 from x to y, then follow P_2 backward from y to x. This gives a closed walk, a circuit.

If the two paths share no common vertices except x and y, this closed walk is a cycle. If they share other vertices, the closed walk contains a repeated vertex, so it can be reduced to a cycle.

Therefore, the graph contains a cycle.

Exercise 7. Prove that there is at most one path between any two nodes in an acyclic graph. (Hint: Use Exercise 6).

suppose, for contradiction, that there are two distinct paths between two nodes *a* and *b* in an acyclic graph.

By Exercise 6, the existence of two distinct paths from *a* to *b* implies the graph contais a cycle, which contradicts the assumption that the graph is acyclic.

Therefore, there is at most one path between any two nodes in an acyclic graph.

Exercise 8. Prove that there is a unique path between any two nodes in a tree.
since a tree is acyclic, then by Exercise 7, there is at most one path (unique) between any two nodes in a tree.

Exercise 9. Let *u* be an ancestor of *v*. Prove that dist(*u, v)* = v.depth - u.depth.

since u is an ancestor of v, the unique path from u to v goes down the tree along the parent-child edges from u to v.

By definition, v.depth is the number of edges from the root to v, and u.depth is the number of edges from the root to u.

The number of edges between them, is the number of edges from u to v, and is indeed, the difference:

$$\text{dist}(u, v) = v.\text{depth} - u.\text{depth}$$

This works because each integer c can be written in binary, so the code above eventually removes each bit of c from MSB to LSB. Similarly, we can optimize lca by again going through these pointers in decreasing size while we can:

```
def lca(u, v):
    u = climb(u, u.depth - v.depth)
    v = climb(v, v.depth - u.depth)

    for k from h to 0:
        if u.anc[k] != v.anc[k]:
            u = u.anc[k]
            v = v.anc[k]

    # still need to go up once
    if u != v:
        u = u.parent
        v = v.parent

    return u
```

Exercise 12. Explain why the last step of using the parent pointer once is necessary.

Exercise 12. Explain why the last step of using the parent pointer once is necessary.

In the LCA algorithm using binary lifting, after the `for` loop, u and v are the deepest nodes whose ancestors are not common ancestors of the original u and v. At this point, u.anc[0] and v.anc[0], their parents are the same node, the LCA.

Thus, one final step up the parent pointer is needed to reach the LCA from both u and v.

Note that "Euler tour" is not quite the same as an "Eulerian circuit", because each edge is traversed twice—once upward, and once downward.[4] Performing an Euler tour is straightforward, and afterward, we get a list of nodes in the Euler tour.

For example, for the tree above, the Euler tour is:

$$E = [0, 1, 0, 2, 0, 3, 4, 3, 5, 3, 0].$$

As it turns out, the LCA of u and v can be obtained as follows:

1. Let i be the leftmost index of any occurrence of u or v in E.
2. Let j be the rightmost index of any occurrence of u or v in E.
3. The LCA is $E[k]$, where $i \leq k \leq j$ and $E[k]$ is the node with the minimum *depth* in $E[i...j]$.

The first two steps can be done in $\mathcal{O}(1)$ after an $\mathcal{O}(n)$ preprocessing; we can simply store the leftmost and rightmost occurrence of every node in the Euler tour. The last step is a range minimum query on the *depth sequence* of the Euler tour.

Exercise 14. Prove that the above procedure correctly computes the $\text{lca}(u, v)$.

Exercise 14. Prove that the above procedure correctly computes the lca(u,v).

Let E be the Euler tour of the rooted tree and D the corresponding depth array. Let i and j be the leftmost and rightmost occurrences of u and v in E (assuming i <= j).

Between i and j in the tour, the path goes up from u to their LCA and down to v, visiting the LCA and possibly other ancestors.

The LCA is the node with minimum depth among those in E[i..j], since all nodes in this interval lie on or between the two root-to-leaf paths and the LCA is the highest common point.

Thus, lca(u,v) = E[k] where k is the index of the minimum value in D[i..j].

Exercise 15. Show that the length of E and D is $\Theta(n)$.

In an Euler tour of a tree with n nodes, each edge is traversed twice: once upward, once downward: A tree has n - 1edges, so the tour contains 2(n - 1) edge traversals.

Adding the n nodes (each time a node is visited, it is recorded), the total length of E (and thus D), is:

$$n + 2(n - 1) = 3n - 2 = \mathbf{\Theta(\mathit{n}).}$$