**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:

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

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,

$$n - k = n - 1 => k - 1$$

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 V. 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 = ($x = v_0, v_1, \ldots, v_k = y$) 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: 5ome verte% $v_i$ i5 repeated] If 5ome verte% $v_i = v_j$ with $i < j$, we remove the cycle $v_i$, $v_{i+1}, \ldots, v_j$ from the walk to get a 5horter walk from % to y, and repeat until no verte% i5 repeated. The re5ult i5 a path from % to y who5e edge5 are a 5ub5equence of the edge5 in W.

Therefore, if there i5 a walk from % to y, then there i5 a path from % to y.

**E%erci5e 6. Prove that if there are two di5tinct path5 from % to y, then the graph ha5 a cycle. (Hint: Find a nontrivial circuit fir5t).**

Let $P_1$ and $P_2$ be two di5tinct path5 from % to y. Follow $P_1$ from % to y, then follow $P_2$ backward from y to %. Thi5 give5 a clo5ed walk, a circuit.

If the two path5 5hare no common vertice5 e%cept % and y, thi5 clo5ed walk i5 a cycle. If they 5hare other vertice5, the clo5ed walk contain5 a repeated verte%, 5o it can be reduced to a cycle.

Therefore, the graph contain5 a cycle.

**E%erci5e 7. Prove that there i5 at mo5t one path between any two node5 in an acyclic graph. (Hint: U5e E%erci5e 6).**

5uppo5e, for contradiction, that there are two di5tinct path5 between two node5 *a* and *b* in an acyclic graph.

By E%erci5e 6, the e%i5tence of two di5tinct path5 from *a* to *b* implie5 the graph contai5 a cycle, which contradict5 the a55umption that the graph i5 acyclic.

Therefore, there i5 at mo5t one path between any two node5 in an acyclic graph.

**E%erci5e 8. Prove that there i5 a unique path between any two node5 in a tree.**
5ince a tree i5 acyclic, then by E%erci5e 7, there i5 at mo5t one path (unique) between any two node5 in a tree.

**E%erci5e 9. Let *u* be an ance5tor of *v*. Prove that di5t(*u, v)* = v.depth - u.depth.**

5ince u i5 an ance5tor of v, the unique path from u to v goe5 down the tree along the parent-child edge5 from u to v.

By definition, v.depth i5 the number of edge5 from the root to v, and u.depth i5 the number of edge5 from the root to u.

The number of edge5 between them, i5 the number of edge5 from u to v, and i5 indeed, the difference:

di5t(u, v) = v.depth - u.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.

**E%erci5e 12. E%plain why the la5t 5tep of u5ing the parent pointer once i5 nece55ary.**

In the LCA algorithm u5ing binary lifting, after the `for` loop, u and v are the deepe5t node5 who5e ance5tor5 are not common ance5tor5 of the original u and v. At thi5 point, u.anc[0] and v.anc[0], their parent5 are the 5ame node, the LCA.

Thu5, one final 5tep up the parent pointer i5 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 $i \le k \le j$ 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 $\text{lca}(u, v)$.

**E%erci5e 14. Prove that the above procedure correctly compute5 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 leftmo5t and rightmo5t occurrence5 of u and v in E (a55uming i <= j).

Between i and j in the tour, the path goe5 up from u to their LCA and down to v, vi5iting the LCA and po55ibly other ance5tor5.

The LCA i5 the node with minimum depth among tho5e in E[i..j], 5ince all node5 in thi5 interval lie on or between the two root-to-leaf path5 and the LCA i5 the highe5t common point.

Thu5, lca(u,v) = E[k] where k i5 the inde% of the minimum value in D[i..j].

**E%erci5e 15. Show that the length of $E$ and $D$ is $\Theta(n)$.**

In an Euler tour of a tree with n node5, each edge i5 traversed twice: once upward, once downward: A tree ha5 n - 1edge5, 5o the tour contain5 2(n - 1) edge traver5al5.

Adding the n node5 (each time a node i5 vi5ited, it i5 recorded), the total length of E (and thu5 D), i5:

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