1. ***Grading homework problems.*** You are a new teaching assistant of the upcoming exciting(!) course on COSC 39: Theory of Computation. The deadline for Homework 0 has just passed. After finding your most comfortable spot at home with snack on one side and pillows on the other, you are ready to grade. One of the homework problems asks:

   > A ***complete*** binary tree is a rooted binary tree where every node is either a *leaf* (with no children), or an *internal node* where both its children are presented.
   >
   > Prove that any complete binary tree has more leaves than the internal nodes.

   One student Ananya submitted the following answer:

   > Let's prove the statement by induction on the number of nodes in the tree. Given a complete binary tree T with n nodes, consider all the possible ways to add nodes to the tree. If we add two leaves to the same leaf node x in T, x becomes an internal node. By induction hypothesis T has more leaves than the internal nodes, and while we turn leaf x into an internal node, we also added two new leaves and thus the difference between number of leaves and internal nodes remains unchanged. Thus there are more leaves than the internals nodes.

   Another student Brittany submitted this answer:

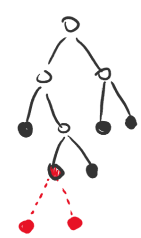
   > T has n nodes. If r is a leaf then we are done. If r is an internal node, remove r from T and now we have T_1 and T_2. Let ℓ_i be the number of leaves and m_i be the number of internal nodes in T_i. By induction, we have $\ell_i > m_i$ and $\ell_i + m_i = n_i$ for each i, and $n_1 + n_2 = n - 1$. Now adding r back, the leaves in T_i are still leaves and the internal nodes in T_i are still internal nodes, thus $\ell = \ell_1 + \ell_2 \geq (m_1 + 1) + (m_2 + 1)$ by the inequalities above. Because r itself is an internal node, one has $m = m_1 + m_2 + 1$, and thus $\ell > m$.

   A bright student Charles wrote this:

   > Charge everyone to their parent. Now every leaf will have a -1 and every internal node will have a $+1$, except for the root which has $+2$. This means that there are more leaves in the tree than the internal nodes.

   Student Daiwen's answer even has a picture:

   > Look at the picture. Removing the two red nodes will decrease the number of leaves by two, but at the same time turning their parent into a leaf, thus keeping the difference between the counts unchanged. Continue in the same fashion until all the nodes except root have been removed. Now the root is a leaf and there are no internal nodes, so the statement is proved.

   Give feedback to all the answers from the students by pointing out any false statements, logic flaws (where a true statement does not follow immediately from the previous true statement), and sentences that are not even wrong / not parsable.[1]

[1] Bonus points: Criticize your own solutions to the rest of the problems. Doing so will help you improve your scientific writing and communication skills. As a rule of thumb, your answers should at least pass your *own* criticism.

2. ***Neighborhoods in graphs.*** Let G be an undirected graph with *at most* one edge between any pair of vertices, but possibly with self-loops (an edge whose head and tail is the same vertex). Define V and E to be the vertex set and edge set of G, respectively.

   The ***neighborhood*** function $N : V \to 2^V$ takes a vertex v and maps it to the subset of vertices adjacent to v in G. We also define the neighborhood of any *subset* of vertices S as the union of each individual neighborhoods, one for each vertex in S. In notation,

   $$N(S) := \bigcup_{v \in S} N(v).$$

   Fix a starting vertex s in V. The ***k-th neighborhood*** of s is defined to be $N(N(\cdots N(s)\cdots))$, where $N(\cdot)$ is applied k times.

   (a) Describe an algorithm to decide if the following is true: for any vertex t in V, there is an integer k such that the k-th neighborhood of s contains t.

   *(b) Describe an algorithm to decide if the following is true: there is an integer k, such that for any vertex t in V the k-th neighborhood of s contains t.

3. ***Balanced parentheses.*** ***Balanced parentheses*** is a string over the two symbols `[` and `]`, defined *recursively* as one of the following:

   - an empty string ε;
   - string `[`w`]` for some balanced parenthesis w;
   - string xy for some *nonempty* balanced parentheses x and y.

   For example, `[[[][]]][][[[]]][]` is a balanced parentheses string of length 18.

   (a) Prove by induction that removing any pair of consecutive symbols `[]` (if exists) from any balanced parentheses results in another balanced parentheses.

   *(b) Prove by induction that removing any pair of consecutive symbols `][` (if exists) from any balanced parentheses results in another balanced parentheses.