1. ***Grading homework problems.*** 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.

   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.

   (a) *Ananya:* 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.

   Solution: The obvious problem that the base case is missing. More problematically, Ananya attempts to prove induction backwards by ***adding things onto the existing structure***.

   It is *not* true that any complete binary tree with $n+2$ nodes can be obtained from adding two leaves to a given n-node complete binary tree T. One might say, "Oh, in that case we should choose another tree T." Unfortunately, tree T is *given* to us, and we have no control in what it looks like. It is a better (and less confusing) idea to prove an induction statement by starting with an *arbitrary* object from the statement (say, an arbitrary complete binary tree T).

   > *Better answer from Ananya:* Let's prove the original statement by induction on the number of nodes in the tree. Let T be an *arbitrary* complete binary tree with n nodes. If T has exactly one leaf and no internal nodes, then the statement holds true. Otherwise, suppose T has at least one internal node. Pick an internal node x that is farthest away from the root of T. In this case, both children of x must be leaves, otherwise we should have chosen one of the children of x as x itself instead.
   > Now remove the two children of x and call the new tree T'. Since T' has fewer nodes than T, by induction hypothesis T' has more leaves than its internal nodes. By the above claim both children of x are leaves in T, and by removing them we turn x into a leaf (in T'); the difference between number of leaves and internal nodes remains unchanged from T to T'. This proves the statement.

   ■

   (b) *Brittany:* 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 $\ell > m$.

   Solution: The major mistake from Brittany's answer is that ***some variables are undefined***.

   This ranges from variable r, whose meaning (the *root* of T) is central to the correctness of the proof, to variables like T_1, T_2, n_1, n_2, ℓ, and m; we can mostly infer their meanings, but the proof became difficult to read without defining the variables explicitly.

Better answer from Brittany: Let's prove the statement by induction on the number of nodes in the tree. Let T_0 be an arbitrary complete binary tree with n_0 nodes. Denote the root of T_0 as r. If r is a leaf then there are no internal nodes in T_0 and we are done. If r is an internal node, remove r from T to obtain two complete binary trees T_1 and T_2. Let n_i be the number of nodes, ℓ_i be the number of leaves, and m_i be the number of internal nodes in T_i respectively, for each $i \in \{0,1,2\}$. Observe that $n_i = \ell_i + m_i$ for each i. Also,

$$n_0 = n_1 + n_2 + 1, \qquad m_0 = m_1 + m_2 + 1, \qquad l_0 = l_1 + l_2$$

hold because T_0 has all the nodes in T_1 and T_2, plus r which is an internal node. By induction hypothesis we have $\ell_i \geq m_i + 1$ for each i. Now

$$\ell = \ell_1 + \ell_2 \geq (m_1 + 1) + (m_2 + 1) = m + 1,$$

which proves the statement.

■

(c) *Charles:* 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.

Solution: The major flaw is that Charles' proof is ***sketchy and potentially unparsable***.

The charge operation is never formally defined, and it is unclear what "everyone" is referring to. Because the charge operation was undefined, the meaning of -1, $+1$, and $+2$ is completely unclear, as well as their connection to the main statement about leaves and internal nodes.

Better answer from Charles: Let T be an arbitrary complete binary tree. Set the charge of each node in T to be zero. Now for each node x (besides the root), subtract one-unit of charge from x and send it to the parent of x. After the charges are sent, every leaf has a -1 charge because the leaves have no children. Every internal node is left with a $+1$ charge because each internal node of a complete binary tree T has exactly two children and one parent. The root of T has $+2$ charges.
As the total number of charges in T remain zero, by pigeonhole principle there are more leaves than the internal nodes in T.

■

(d) *Daiwen:* 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.

Solution: Daiwen's answer suffers from the common mistake to ***prove by examples***.

To provide a formal proof, it is not sufficient to argue that your method works for a specific example; it must be shown that the method works for an *arbitrary* example. Also, in general sentence like "continue in the same fashion..." suggests that there is an induction/recursion hiding beneath, and one should spell it out explicitly by stating the induction hypothesis.

A better answer from Daiwen looks similar to the one from Ananya.

■

Note. The mistake made by Ananya is pretty common among students who are still learning induction. Another common mistake as a beginner is to focus on the *format* of an induction proof, by transcribing the text "$n = 1$", "$n = k$" and "$n = k + 1$" mindlessly onto the paper. It is more important to master the essence of induction proof: to reduce the current problem to the exact same problem one-size (or a-few-sizes) smaller. (Not to mention that the standard frameworks taught in many textbooks are confusing at best, and cultivate misleading intuition about inductions (or sometimes being outright wrong) at worst. The induction note by Erickson is a good resource to learn about better habits in writing induction proofs.

Undefined variables are the enemy of proof writers. Never, never, never write down a symbol/object without definition. On the other extreme, some people never use a symbol in their proofs; instead, they use *pronoun* when referring to an object. Using pronouns is an art that students need to pay extra caution to; improper uses of pronouns lead to ambiguity that sometimes changes the meaning of the sentence. We will point them out when we see one.

Charles's answer (and to some extent, Daiwen's) is what we called a *proof sketch*. You might hear from time to time that experts talk with each other in such fashion. Proof sketch provides a guideline for the experience readers to construct a proof on their own; however, proof sketches are *not* replacements for proofs. If you write proof sketches as answers, you are risking yourself to be misunderstood by the reader/grader (and rightfully so). It is crucial to think about the *audience* of your writings, oral presentations, and program comments. For example, I might write the official answers *differently* for a graduate-level course and consider Charles' answer to be correct.

Rubric: -1 point for missing the bold-text error from each student's answer.

For future homework problems, proofs using *backward induction* and *proof by examples* will receive an automatic ***zero***. Proofs containing *undefined variables*, *unparsable sentences* and *proof sketches* will trigger ***sudden-death***; you will receive a zero unless the proof is otherwise *perfect*. It is up to the graders to decide if they can understand your proof or not.

2. ***Neighborhoods in graphs.***

(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.

Solution: The problem is equivalent to asking whether the input graph is connected: there is a path from s to any vertex t in G if and only if the distance from s to t is finite, and thus t is contained in some k-th neighborhood of s.

Run an arbitrary graph traversal algorithm (say DFS) starting from s, and return *yes* if the graph is connected. ■

*(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.

Solution: Construct a (directed) *supergraph* H where the nodes of H are all possible vertex subsets of G. Create a directed edge from node S to node $N(S)$ in H for any vertex subset S. Now a directed path from $\{s\}$ to V exists in H if and only if there is a k-th neighborhood of s containing every single vertex in G.

Run an arbitrary graph traversal algorithm (say DFS) in H and check if node V is reachable from $\{s\}$. Return *yes* if so and *no* otherwise. ■

Solution (alternative): The problem is equivalent to asking whether the input graph is *not* bipartite.

First, check if the graph contains exactly one single vertex s; if so, return *yes*. Second, check if the graph is connected as in (a); if not, return *no*. Run the two-coloring algorithm starting from s and check if the graph is two-colorable. Return *no* if so and *yes* otherwise.

Proof sketch: To prove that the algorithm is correct, one has to argue that (1) if there is an odd cycle in G then eventually all vertices in the odd cycle will be in some k-th neighborhood of s simultaneously; from there all vertices will be in the $(k+n)$-th neighborhood after another n steps, where n is the number of vertices; and (2) if there are no odd cycles then the graph is bipartite (two-colorable), and any two vertices with different colors cannot show up in the same k-th neighborhood for any k (unless G has exactly one single vertex, in which case the answer is trivially *yes*). ■

Rubric: Standard 5-point grading scale. It is ***not*** required to prove the algorithms are correct.

- 5 means the answer is perfect — the solution is correct, concise, and convincing.
- 4 means the answer is basically correct, modulo some small minor issues.
- 3 means the answer is mostly correct, but there are some missing details or errors that does not affect the overall argument.
- 2 means the answer is heading in the right direction, but there are major gaps in the presentation before making the solution rigorous.
- 1 means the answer is incorrect, despite an honest attempt. The solution successfully convey the (while incorrect) ideas to the reader/grader.
- 0 means the answer is either missing, for the wrong question, or not readable/parsable.

3. ***Balanced parentheses.***

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

Solution: We prove the statement by induction on the length of the strings. Let z be an arbitrary balanced parentheses. We separate into three possibilities based on the recursive definition of balanced parentheses.

- If $z = \varepsilon$, there are no pairs [] in z and thus the statement is vacuously true.
- If $z = [w]$ for some balanced parenthesis w, either [] is chosen completely from w, or [] uses the leftmost [in z. In the first case, because w is shorter than z, the induction hypothesis impiles that one can remove the pair [] safely from w. In the second case, w must be ε because no nonempty balanced parentheses has its leftmost symbol to be] by the recursive construction. This implies that $z = []$, and we can safely remove the only pair [] and the result is another balanced parentheses.
- If $z = xy$ for some *nonempty* balanced parentheses x and y, there are two possible cases: either (1) [] lies completely inside x or y, or (2) [] uses symbols from both x and y. In the first case, because both x and y are shorter than z, the induction hypothesis applies and one can safely remove the pair [] from either x or y. The second case cannot happen because in such case the leftmost symbol in y must be], which is impossible by the recursive construction. ■

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

Solution: We prove the statement by induction on the length of the strings. Let z be an arbitrary balanced parentheses. We separate into three possibilities based on the recursive definition of balanced parentheses.

- If $z = \varepsilon$, there are no consecutive symbols][in z and thus the statement is vacuously true.
- If $z = [w]$ for some balanced parenthesis w, the symbols][must be chosen completely from w. Because w is shorter than z, the induction hypothesis implies that one can remove][safely from w.
- If $z = xy$ for some *nonempty* balanced parentheses x and y, there are two possible cases: either (1)][lies completely inside x or y, or (2)][uses symbols from both x and y. In the first case, because both x and y are shorter than z, the induction hypothesis applies and one can safely remove the pair][from either x or y. To prove the second case, one has to prove one additional claim:

  > Any nonempty balanced parentheses can be represented as either $u[v]$ (or $[u]v$) for some balanced parentheses u and v.

  Assuming the claim is true, we can finish the proof like this: since][uses symbols from both x and y, the rightmost symbol of x is] and the leftmost symbol of y is [. Write $x = u[v]$ and $y = [u']v'$ for some balanced parentheses u, v, u' and v' using the claim, we have $z = u[v][u']v'$. Removing the pair][gives us another balanced parentheses $u[vu']v'$. This concludes the proof to the main statement.

To prove the claim, it is sufficient for us to prove the case for $u[v]$; the case for $[u]v$ can be proved similarly. We again use induction on the length of the strings. Let z be an arbitrary nonempty balanced parentheses (and thus $z \neq \varepsilon$). Again we consider the two possible cases.

- If $z = [w]$ for some balanced parenthesis w, set $u = \varepsilon$ and $v = w$ turns z into $u[v]$.
- If $z = xy$ for some *nonempty* balanced parentheses x and y, by induction hypothesis $y = u'[v']$ for some balanced parentheses u' and v'; setting $u = xu'$ and $v = v'$ turns z into $u[v]$ and thus proving the claim. ■

Note. The claim in last sentence in the proof to Problem 3a (that no balanced parentheses has its leftmost symbol to be]), if being *completely rigorous*, requires yet another induction proof. The graders reserve the right to decide what facts about strings are basic and requires no proofs.

Rubric: Standard 5-point grading scale.