Problem 4: CALICO's Corporate Conundrum
15+9+6=30 Points

Problem ID: `managers`
Rank: `2+3+3`

Introduction

Problem Summary

Given the managers of a company's $\mathbf{N}$ employees, find the maximum number of unique disputes a single employee is responsible for resolving. A dispute between two employees is resolved by the lowest-level manager that has authority over both of them, directly or indirectly.

You can find the original problem statement here:
https://calico.berkeley.edu/files/calsuco-22/contest/managers/managers.pdf

Big Ideas

It's been a while since we had a graph problem, so we hope that this one was a welcome addition to the contest!

Because every employee has exactly one manager, and the company must be connected, we know that the company must be a tree with the CEO at the root and manager-subordinate relations forming the parent-child edges. For the main test set and first bonus, the fact that each employee manages at most two others tells us we're working with a binary tree specifically.

Although a brute force approach was sufficient for the main test set, the step up to the first bonus required a clever change in approach for efficiency, and the step to the second bonus required a small formula optimization.

Main Test Set

Brute Force *Solution available in Python*

We can pass the main test set by identifying all possible disputes between pairs of employees. Then for each dispute, find the lowest-level manager that has authority over both of them and count the number of disputes that manager is responsible for resolving.

One way to find the lowest-level manager with authority over two employees is to:

1. Follow the chain of management upwards from one employee until you reach the CEO—in other words, find the manager of the current employee, then find their manager, etc. until you reach the CEO.
2. Track each employee you see in a set (including the original employee).
3. Then, do the same for the other employee, until you find an employee shared by the first employee's chain of management.
4. The first employee you find using this method will be the one to resolve their dispute.

After finding the resolving manager for all pairs of employees, we output the maximum number of disputes a single employee is responsible for resolving.

Since there are $N(N-1)$ pairs of employees in the company, the number of unique disputes within the company grows at a rate of $O(N^2 - N) = O(N^2)$. Each pair then requires the calculation of who will resolve the dispute. The complexity of the above implementation depends on the height of the management tree. For a typical, balanced binary tree, this grows at a rate of $O(log(N))$; however, in the worst case, the management tree could be a straight line—where every employee manages at most one other employee—so the height could grow at a rate of $O(N)$. Therefore, this brute force approach would have an overall runtime of $O(N^3)$.

Lowest Common Ancestor

Those of you more familiar with the subject may have recognized finding the lowest-level manager as the lowest common ancestor (LCA) problem. While algorithms to quickly compute the LCA exist, the bottleneck for this particular approach is the need to compute $O(N^2)$ different LCAs. That being said, it's possible to design a $O(N^2 log(N))$ or even $O(N^2)$ optimized brute force algorithm using these methods, although $O(N^3)$ was sufficient for this test set.

Bonus Test Set 1

Now that we have 10^5 employees, we need a faster way without looking at all employee pairs.

Paradigm Shift *Solutions available in Java, Python*

In the previous test set, we approached the problem by finding who's in charge of resolving any given dispute. Flipping our approach to this problem—finding which disputes can be resolved by any given employee—gives the potential to yield considerable efficiency advantages.

Let's define the branch of an employee as everyone they have authority over, including themselves. Every employee has a branch of the company in which they lead.

We need to recognize that a dispute will only be resolved by an employee if either:
1. the employee themself is one of the parties involved and the other party is a subordinate, or
2. both disputing parties are on different branches of the employee's direct subordinates.

The number of disputes that fall under the first condition is equal to the number of subordinates (direct or indirect) an employee has. In other words, the branch size of that employee minus 1. The number of disputes that fall under the second condition is equal to the size of the left branch multiplied by the size of the right branch—or zero, if the employee has less than two subordinate branches. Therefore, efficiently calculating employee branch sizes is key.

We can find these sizes recursively, where the size of an employee's branch is the size of its left and right subordinate branches added together (if present), plus one. Employees with no subordinates would have a branch size of one. While recursing, we can calculate both the number of subordinates an employee has, as well as the number of employees in each of their subordinate branches. With this information, we can calculate the number of disputes each employee needs to resolve, keeping track of the maximum along the way.

This recursive approach is known as depth-first search (DFS), and it lets us calculate the number of subordinates underneath each employee—alongside the number of disputes they're responsible for—in linear time $O(N)$.

Note that multiplying large branch sizes can result in values of up to $10^5 \bullet 10^5 = 10^{10} > 2^{32}$, so we should use a data type capable of handling larger numbers (such as a 64-bit integer).

Bonus Test Set 2

With employees able to have more than two subordinates, we need to generalize how we calculate the number of disputes under the second condition, where both disputing parties are on different subordinate branches.

With at most two branches, this used to mean multiplying the number of employees in each branch together. With more than two branches, we need to recognize this means (efficiently) finding the sum of all pairwise products between subtree sizes.

Optimized Summation *Solution available in C++*

Suppose we have subordinate branch sizes x_1, x_2, …, x_m for a given employee. Since we want to find the sum of pairwise products between different branches, we're essentially looking for:

$$\sum_{i \neq j}^{m} x_i x_j = \sum_{i=1}^{m} \sum_{j=i+1}^{m} x_i x_j$$

Computing this directly with a double loop would cause this calculation to have a runtime of $O(m^2)$, due to there being $m(m-1)$ pairs. This would cause our solution's runtime to increase to $O(N^2)$, which would be too large of an efficiency loss.

It's easier to see how to optimize this calculation if we unroll the sums. Doing so, we get the sum on the left. If we apply the distributive property to each row, we get the sum on the right:

$$\begin{aligned} x_1x_2 + x_1x_3 + x_1x_4 + x_1x_5 + \ldots + \\ x_2x_3 + x_2x_4 + x_2x_5 + \ldots + \\ x_3x_4 + x_3x_5 + \ldots + \\ x_4x_5 + \ldots + \\ \ldots \end{aligned} \qquad \begin{aligned} x_1(x_2 + x_3 + x_4 + x_5 + \ldots) + \\ x_2(x_3 + x_4 + x_5 + \ldots) + \\ x_3(x_4 + x_5 + \ldots) + \\ x_4(x_5 + \ldots) + \\ \ldots \end{aligned}$$

It's easy to see now that we can efficiently compute this sum by precomputing the right hand side of each row. Begin by setting RHS to $x_1 + x_2 + x_3 + \ldots + x_m$. Then, for each branch x_i, subtract x_i from RHS and add $x_i \cdot RHS$ to the total.

The calculation for each employee's disputes now takes $O(m)$ operations, but since the sum of all m never exceeds $\mathbf{N}$, our overall runtime remains $O(N)$.

Another way to approach this calculation is as follows: if we let S represent the sum of all branch sizes (in other words, $S = x_1 + x_2 + ... + x_m$), then we can recognize that each member of the ith branch will have $S - x_i$ possible disputes in other branches. We subtract x_i from S because we only want to include employees from other subtrees in the dispute; otherwise, this would cause a different employee to become responsible for resolving it.

One thing to note is that we want to avoid double counting pairs—a dispute between employees A and B should not be seen as distinct from a dispute between B and A. As a result, we divide by two to find the number of distinct disputes, giving the formula:

$$\frac{1}{2}\sum_{i=1}^{m} x_i(S - x_i).$$

This approach also has an overall runtime of $O(N)$, similar to the previous approach.

Negative Space

Yet another approach to computing the same sum is to recognize that it is equivalent to subtracting the total number of pairs within each subordinate branch from the total number of pairs in the employee's subordinate branches.

To find the total number of pairs in the employee's subordinate branches, we can start by finding S, the sum of each branch size. The total number of pairs then is given by $\frac{1}{2}S(S - 1)$. Similarly, the total number of pairs within each subordinate branch x_i, is given by $\frac{1}{2}x_i(x_i - 1)$. Like the last section, we divide both quantities by 2 to prevent double counting.

Thus, our final formula is:

$$\frac{1}{2}S(S - 1) - \sum_{i=1}^{m} \frac{1}{2}x_i(x_i - 1)$$

This approach also has an overall runtime of $O(N)$, similar to the previous approach.

Similarly to the last test set, we should use a data type capable of handling larger numbers (such as a 64-bit integer).