Lab 3: Trees with AlgoBook

You are working for a promising new social media startup called "AlgoBook."

You have been tasked on a new internal budget tracking tool. The CEO would like to be able to input any individual at the company and figure out the budget of their organization.

The organization hierarchy is modeled as a tree where:

- individual contributors (IC) are external (or leaf) nodes
- organization managers (M) are internal nodes

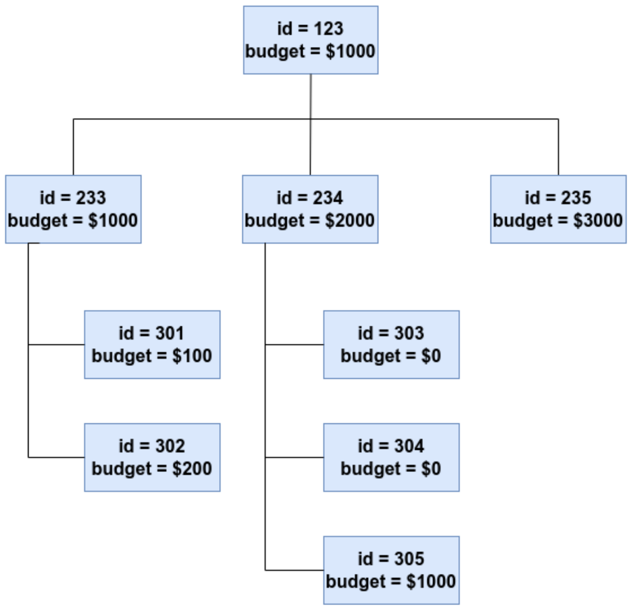
When figuring out an employee's organization, there are two cases to consider:

1. If the employee is an M (internal node), their organization is the subtree rooted at their corresponding node.
2. If the employee is an IC (external node), their organization is the subtree rooted at their manager's corresponding node.

An organization's budget is the sum of the budgets of every person in the organization (M and ICs). A node in the tree should have at least:

- The employee ID
- The budget of that employee
- A list of direct reports
- The employee's manager

In the below example, Employee 123 would have an organization that includes all of the shown employees. Employee 235 has the same organization. However, Employee 303 has an organization that starts at Employee 234 and additionally includes Employee 304 and 305.



Your program will need to have two components: a constructor and the getOrgBudget function.

The **constructor** for the organization class, which will take in a list of Employee objects, must convert the list into a tree data structure and store it as a class field. Each employee object has the following: employeeId, managerId (or null if it is the CEO/root), and budget.

As a hint, it will make things easier if you also store a mapping of employeeId to EmployeeTreeNodes as a second class field.

The **getOrgBudget** function will take in an employee ID and return the total budget of that employee's organization.

You are given the following in the starter code:

- The stubbed organization class
- An Employee class
- An EmployeeTreeNode class with suggested fields
- Unit tests

You may modify the EmployeeTreeNode class but may not change the Employee class.

You must use a Tree data structure to store the organization and must use a tree traversal algorithm to compute an organization's budget.

Deliverable Explanation

You must efficiently solve this problem in either Python or Java, using one of the attached starter code files. Several unit tests are provided to help you test your code.

Having the optimal runtime will grant you extra credit. On the other hand, you may also lose points if your code is substantially inefficient. You must also provide an explanation of the running time of your code in the provided placeholder in the starter code.

Your code will be evaluated against a set of visible and hidden test cases (the visible ones are also provided in the starter code for your convenience) to test correctness. The hidden test cases are intended to verify that you have a robust solution that considers possible edge cases and various inputs.

If you are using Java, your file's package must be set to "student".

Submission Instructions

Submit the following file on Gradescope:
AlgoBook.java to Lab3 (Java)
OR
algo_book.py to Lab3 (Python)

Grading Methodology

The lab is worth 100 points and is broken down into the following categories:

- Test Cases Passing (80 points)
- Big O explanation (15 points)
- Overall code performance and style (5 points)
- Having Optimal Running Time (5 points of Extra Credit)