ASSIGNMENT 2: ORGANIZATIONAL HIERARCHY

Due Date: Sunday February 27th, 11:59 pm.

Goal: The goal of this assignment is to get some practice with trees and search trees. Search Trees are one of the most important data structures we will study in this class – they are regularly used in large database systems for storing indexes on the records.

Problem Statement: We want to maintain the list of employees in a company. We will be concerned with two quantities associated with each employee in the company -- ID of the employee (you can assume no two employees in the company have the same ID), and the level of the employee. The level denotes where the person stands in the hierarchy. Level 1 denotes the highest post in the company (say the owner), level 2 comes below level 1 and so on. There is only 1 person at level 1, but there can be several employees at level i > 1. Each level i employee works under a level i-1 employee, which is his/her immediate boss. Given an employee A, we can form a sequence of employees A',A'', A''', ... where A works under A', A' works under A'', and so on. We say that each employee in the sequence A',A'', A''',... is a boss of A. You can assume that the employee ids support the '<', '>' and '==' operations over them. We would like to implement a suitable tree-based data-structure so that we can implement the following operations :

```
public interface OrgHierarchy    {

        public OrgHierarchy();  /* Initializes an empty organization */

        public boolean isEmpty();  /* Returns true if the organization is empty. */

        public int size();  /* Returns the number of employees in the organization */

        public int level(int id) throws IllegalIDException;  /*
        Returns the level of the employee with ID=id */

        public void hireOwner(int id) throws NotEmptyException;  /* Adds the first
        employee of the organization, which we call the owner. There is only one owner
        in an org who cannot be deleted */

        public void hireEmployee(int id, int bossid) throws IllegalIDException;
        /* Adds a new employee whose ID is id. This employee  will work under an
        existing employee whose ID is bossid (note that this  automatically decides the
        level of id, it is one more than that of bossid). Your code should throw an
        exception if the id already exists in the OrgHierarcy */

        public void fireEmployee(int id) throws IllegalIDException;
        /* Deletes an employee who does not manage any other employees. */

        public void fireEmployee(int id, int manageid) throws IllegalIDException;
        /* Deletes an employee (id) who might manage other employees. Manageid is
        another employee who works at the same level as id. All employees working under
        id will now work under manageid */

        public int boss(int id) throws IllegalIDException;  /* Returns the immediate
        boss, the employee. Returns -1 if id is the owner's ID  */

        public int lowestCommonBoss(int id1, int id2) throws IllegalIDException;
        /* outputs the ID of the employee A who is a boss of both id1 and id2, and
        among all such persons has the largest level. In other words, we want to find
        the common boss who is lowest in the hierarchy in the company. If one of the
        input ids is the owner, output -1 */
        public String toString(int id) throws IllegalIDException; /* Returns the whole
        organization rooted at id in a String format. The return string should contain
        the employees (ids) level-wise. The return string should contain employee ids
        in a (single) space separated fashion. So the first input id (root) will
        appear, then all employees directly under that id, and then all employees
        directly under these employee ids and so on. Among employees at the same level,
        the output should be sorted in increasing order of the ids. */

}
```

Write a program which implements such a data-structure. You should choose your data-structures in a manner which allows for efficient implementation of various operations. Whenever possible, implement each method in no more than O(log(n)) time (and O(1) time wherever applicable) where n is the number of employees in the organization. If some methods cannot be implemented in O(log(n)), have a justification ready for why it is not possible.

What is being provided and what to submit?

1. We have provided code (will be uploaded on Piazza and Moodle) that can be used to interact with your OrgHierarchy implementation using a set of simple commands. We expect you to not alter that code, as it will be used to check the correctness of your code. You should **only modify the OrgHierarchy.java file** in the given set of files. You are free to add your own new java classes. The starter code includes
    a. makefile
    b. OrgHierarchyInterface.java - (contains the above given Interface)
    c. OrgHierarchy.java - (which you need to modify)
    d. NotEmptyException.java
    e. IllegalIDException.java
    f. Tester.java
2. You should create individual class files for each part as mentioned above. Submit your code in a .zip file named in the format **<EntryNo>.zip, e.g., 2018ME10000.zip.** Make sure that when we run "unzip yourfile.zip", there should be a directory <EntryNo> created which should contain all the files which are provided in the starter code, including modified OrgHierarchy.java file. You can create additional java class files as per your need/requirements. In addition to your code, "writeup.txt" also be present in the directory. Thus the directory structure should be as follows after unzipping:

    <EntryNo>

    - All the files provided in the Starter code (unmodified) - other than OrgHierarchy.java
    - Modified OrgHierarchy.java
    - Any other files needed for implementation
    - writeup.txt
3. You will be penalized for any submissions that do not conform to this requirement.
4. The writeup.txt should have a line that lists names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone say None. After this line, you are welcome to write something about your code, though this is not necessary.

What is allowed? What is not?

1. This is an individual assignment.
2. Your code must be your own. You can browse online resources for any general ideas/concepts, but you are supposed to search for/look at specific code meant to solve these or related problems.
3. You should develop your algorithm using your own efforts. You should not Google search for direct solutions to this assignment. However, you are welcome to Google search for generic Java-related syntax.

4. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class.** Please refer to the plagiarism related guidelines covered in the first lecture and follow them carefully. In case of any doubts, you are free to contact the TAs or the instructors.
5. You are not allowed to use built-in (or anyone else's) implementations of stacks, queues, vectors, growable arrays and/or other similar data structures - it is ok to use fixed size arrays as covered in the class. A key aspect of the course is to have you learn how to implement these data structures. You are free to use your own implementation of the List from Assignment 1.
6. Your submitted code will be automatically evaluated against another set of benchmark problems. You get a significant penalty if your output is not automatically parsable and does not follow input-guidelines.
7. We will run plagiarism detection software. Anyone found guilty will be awarded a suitable penalty as per IIT rules.

Evaluation Criteria

The assignment is worth 12 points. Your code will be autograded at the demo time against a series of tests. 8 points will be provided for correct output and 4 points will be provided for your explanations of your code and your answering demo questions appropriately. Your code should be as efficient as possible (primarily think about efficiency in terms of the time and memory complexity, and removing any obvious inefficiencies/redundant operations resulting in very slow implementations). Marks will be deducted for inefficient code/implementations.