**Trees: Assignment**

Suppose you have a binary tree representing a command structure of the Starship USS Enterprise. The relationships between the officers can be organized using a BST.

Rank 1: 10 Captain Picard

/ \

Rank 2: 6 Commander Riker 11 Commander Data

/ \ \

Rank 3: 4 Lt. Cmdr. 7 Lt. Cmdr. 12 Lt. Cmdr.

Worf LaForge Crusher

\ \

Rank 4: 5 Lieutenant 13 Lieutenant

security-officer Selar

This tree is meant to represent who is in charge of lower-ranking officers. For example, Commander Riker is directly responsible for Worf and LaForge. People of the same rank are at the same level in the tree.

However, to distinguish between people of the same rank, those with more experience are on the right and those with less on the left (i.e., experience increases from left to right). The higher the officer's id, the more experience the officer has. You may assume **no** officers have the same id's or experience levels. In the diagram above, the number that precedes the officer's name represents each officer's id.

To succeed at this challenge, you'll need to demonstrate that you can create algorithms to insert, retrieve, and find information in the tree above.

**Instructions**

Your goal for this checkpoint is to get the tests to pass.

To do so, you will be modifying the existing StarshipEnterprise class to solve various problems.

**Existing files**

| **File path** | **Description** |
| --- | --- |
| src/StarshipEnterprise.js | Contains the definition of the StarshipEnterprise class. The constructor() method has already been completed for you. |

**Tasks**

Complete the following tasks to pass the tests and this assignment.

In src/StarshipEnterpriseOfficer.js, complete the following methods:

1. assignOfficer(): This method should accept an officer id and name as its arguments and assign a new officer to an existing officer based on experience level (as indicated by the officer's id). If there's no existing officer to assign the new officer to, then the new officer is the captain (e.g., Captain Picard)
2. findOfficersWithNoDirectReports(): This method should return an array of officers who do *not* have direct reports. For example, given the Starship USS Enterprise tree described above, the output would be: [ "Lieutenant Security-Officer", "Lt. Cmdr. LaForge", "Lieutenant Selar", ]
3. listOfficersByExperience(): This method should return an array of officers in increasing order of military experience. For example, given the Starship USS Enterprise tree described above, the output would be: [ "Lieutenant Selar", "Lt. Cmdr. Crusher", "Commander Data", "Captain Picard", "Lt. Cmdr. LaForge", "Commander Riker", "Lieutenant Security-Officer", "Lt. Cmdr. Worf", ]
4. listOfficersByRank() Suppose a fierce battle with an enemy ensues. This method should take the tree of commanding officers above and outline the ranking officers in their ranking order so that if officers start dropping like flies, we know who is the next person to take over command. Your method should return an object where each property represents the numerical rank and each value consists of an array of officers (in any experience order) of a particular rank.

{

1: [ 'Captain Picard' ],

2: [ 'Commander Riker', 'Commander Data' ],

3: [ 'Lt. Cmdr. Worf', 'Lt. Cmdr. LaForge', 'Lt. Cmdr. Crusher' ],

4: [ 'Lieutenant Security-Officer', 'Lieutenant Selar' ]

}

Once these tasks are complete, all tests should pass.