NATIONAL UNIVERSITY
OF COMPUTER & EMERGING SCIENCES
PESHAWAR CAMPUS

Problem Set:	Assignment: A06	**Semester**:	Fall 2019
Points:	*10*		
Date Set:	*See Autograder*	**Due Date**:	*See Autograder*
Course:	CS218 - Data Structures	**Instructor**:	Dr. Nauman

1 Tree Case Study

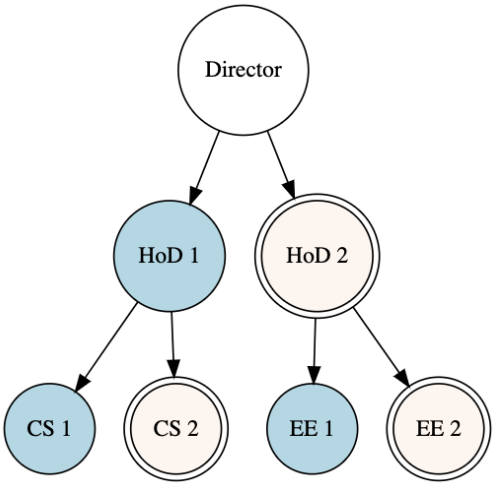
This is *not* a lengthy assignment. Just read through the statement carefully and complete the tasks as described. Make sure you finish one task before you move on to the next one.

1.1 Getting the Full Tree Code In

Bring the whole TreeNode code into your submission file. Make sure you have the following functions:
`dfs`, `dfs_inorder`, `dfs_postorder`, `bfs` and `dfs_apply`

1.2 Organization Hierarchy

We want to store an organization's hierarchy through our tree. An example looks like the following:



There are two HoDs that report to the Director and each HoD has two faculty members working in his or her department.

Now, we don't want to save just the name of the person in the tree because we might want to store other information about the employee too. To take care of this requirements, we first need to create a class called `Person`. For now, this class should have just two functions:

1. a constructor that takes in a name and saves it in a field
2. a dunder str function that returns the name so that we can easily output values of instances

Go ahead and create this class. Afterwards, you should be able to uncomment Section 1 and 2 at the end of a06.py to make sure your code works as expected.

1.3 Finding Values

We now come to the meat of this assignment. We want to add a function called `find_val` in the TreeNode class. This function will take in a string and try to find a node in the tree that matches with this string. If you do this recursively, it's pretty straight forward:

- If the required value matches the node's own value, just return self.
- If that is not the case, we can recursively call the find_val function of our left child (if it exists). If a match is found by that call, we return it. However, if this recursive called didn't find a match, we do another recursive call on our right child (again, if it exists).
- If the value is not matched with the current node and neither the left nor the right child can find it, we simply return `None`.

Section 3 of `a06.py` should be working fine now.

1.4 Collecting Node Values

Recall the `dfs_apply` function and `PerformSum` class. We need to something similar here. Instead of traversing the whole tree and summing the integers, we want to traverse the whole tree and append all node values to a list. For this purpose, create a class `Collector` which would be very similar to the `PerformSum` class. However, it should have a list as its field and on each `process` call, it should append the value of node to the list.

Make sure you have the equivalent `get_list` and `reset_list` functions just as we had in the `PerformSum` class. (Also make sure you are following the DFS pre-order for collecting items.)

Section 4 of `a06.py` should be working fine now.

1.5 Finding the Hiearchy of a Person

Finally, we need to add a function called `find_people_in_hierarchy` to TreeNode. This function would take in a string and return a list of all the people that are in the hierarchy for that node. For instance, if we pass in `'HoD 1'` to the above tree, it should return the list: `['HoD 1', 'CS 1', 'CS 2']`.

The process is quite simple: you simply try to find the node that matches with this string and then apply the collector logic to get the list. Just one caveat: if the string does not match any node, you need to raise an exception of type `ValueError` with any descriptive message.

Section 5 of `a06.py` should be working fine now and you can go ahead and submit your work.

2 Submission

Use `python run.py local` to ensure all tests are passing and then submit your assignment using `python run.py remote`

If you wish to request an extension, use the autograder UI to do so. Each student gets a maximum of 3 extension days per semester.