Divide and Conquer

Introduction

Suppose you're consulting for an IoT firm. The firm has placed multiple sensor devices in an organization where data is being collected at a rapid rate.

The IoT firm has designed a master-slave architecture where multiple sensors are responding to a single master node. It means that even though these devices have different ID and storage capacity, all the data collected by them is going to a specific master node. The master node can be considered as a simple raspberry pi device. All the sensors that are forwarding data to a particular raspberry pi will have its information stored for the data forwarding.

In the beginning the firm has not planned for load balancing. As the number of sensors/devices increased and the firm engineers kept on assigning the devices to a master/raspberry pi based on the maximum capacity, it created a unique problem where some of the raspberry pi or master nodes are overloaded and some of them are under loaded.

Housekeeping points

- This is a minimal example and may not follow some standard practices
- The focus is on the main flow, with minimal error handling. Errors in validation logic should be handled appropriately though.

Program Organization

This section will contain the information related to the different files available in the project.

1. *C03_P01_divide_and_conquer.py:* In this file we have designed a single class named ***System***. This class has methods that take data from csv files and store data of the sensor, master nodes and mapping of each of the sensors to its master nodes. Let's look at the different methods that are implemented;

a. *config_system:* This method is designed to take data from a csv file and push it to store it inside different list data structures. Different variables that are used to save the data are ***sensors_list, sensor_mapping_list, master_node_list****.* All these variables are declared in __init__ method. We are using a csv library to access each row and column to push data in the desired data structure.
b. *SensorAssignedCount:* This method is used to count the number between ***l and r*** variables by comparing against the *OverloadSensor.* After completing the count it is returning the count value from where it was called.
c. *getOverloadedNode*: This is the method that was called from the main function to find the overloaded node.

2. ***app_data.csv:*** This file contains the list of so that are available for you to push the data in the appropriate data structure. Please do not make any changes into this file. This file does not require any changes.
3. ***sample_output.txt:*** This is a sample output file. Once your code is in working state and all the methods are implemented properly, your output should appear like this.

4. Please understand that, for our use case in each of the test cases there will always be a situation where an overload node is present. It is possible that we do not have potential overloaded nodes but we will always have an overload node in our setup.

Problem Statement

The program structure is already set and there are specific methods that you are expected to implement. Please also read the comments in the code, especially in the methods to be implemented. You can modify main.py to add further demonstration calls.

Any master node will be considered as **overloaded** if it serves more than equal to **n//2** devices in a particular environment. Consider a situation where 10 master nodes were placed on a floor that consisted of more than 100 devices. These 100 devices are sending data to these master nodes and mapped accordingly. If any master node is serving more than or equal to 50 sensors then only it will be considered overloaded and all these nodes should be identified.

To keep the scope minimum we will focus on the nodes that are overloaded and potentially overloaded nodes and also we will always consider that there is a node available which is overloaded in every test case. .

As another requirement from the firm, they are also interested in the nodes which are potentially overloaded; meaning the nodes/raspberry pi's which are handling more than equal to n//3 sensors but less than n//2 sensors/devices.

1. ***OverloadNodeHelper:***
   This method is used to get the master node which is overloaded. In our case we are considering all the sensors which are handling more than n//2 sensors are treated as Overloaded nodes. This method will try to identify such nodes using a divide and conquer approach. It means the designed solution should solve this problem using divide and conquer approach and time complexity of this algorithm should not exceed more than ***n * log n***. Here n refers to the number of sensors.

2. **getPotentialOverloadNode:**

   This method is designed to find all the master nodes that are potentially overloaded. This means that all the master nodes that are handling more than or equal to n//3 devices and less than n//2 devices should be included here.

3. **recurrence_relation**: Write the recurrence relation for the first method and the time complexity for the implemented method. As mentioned in the problem statement description the time complexity can not be greater than **n* log n.** Please create a text file that shows the recurrence relation and time complexity. Zip the code file along with the implemented solution and send the complete folder.

Evaluation Rubric

Total Project Points: **20**

- Basic compilation without errors (**10%**) : **2 Points**
- Correctness:
  - Problem statement - 1 (**50%**) : **10 Points**
  - Problem statement - 2 (**20%**) : **4 Points**
  - Problem statement - 3 (**20%**) : **4 Points**

Program Instructions

1. Download the zipped folder named **C03_P01_divide_and_conquer.zip**, and unzip it on your local machine. Go into the directory named **C03_P01_divide_and_conquer**

2. Make sure that you have Python 3.6, or higher, installed. At your command prompt, run:

```
$ python --version
Python 3.7.3
```

If not installed, install the latest available version of Python 3.

3. To run the code in the source code folder, run the following command:
```
$ python3 C03_P01_divide_and_conquer.py (On many Linux/Mac platforms)
                    OR
$ python C03_P01_divide_and_conquer.py (On Windows/Mac platforms)
```

   In any case, one of these two commands should work.

4. Alternatively, you could install a popular Python IDE, such as PyCharm or Visual Studio Code, and select a command to build the project from there.

5. You will be making very frequent changes into the C03_P01_divide_and_conquer.py python files. The idea is to complete both the scripts so that it satisfies all the requirements. Once you have completed the changes in the code and it is executed without any error. Zip the folder as **C03_P01_divide_and_conquer.zip** & now it is ready for submission.