**Distributed information system**

Submitting instructions:

* This code should be written in Python 3.7.x.
* Only vetex.py file will be submitted, please don’t create additional files or change the other attached files.
* The code should be organized, readable, and with meaningful names.

Goal:

In this project we would like to implement in Python a distributed algorithm executed by a vertex in the graph and from the vertex’s perspective.

Denote G=(V,E) is the input graph and n=|V| is the number of vertices in the graph. Every vertex has a unique id which is an integer between 1 and n. We will define the origin vertex to be with id=1. The goal is to build shortest paths tree (in terms of number of edges) from the origin vertex and to calculate for every vertex v the sum of *num* values in the sub-rooted tree. (the total sum includes the *num* values of v itself).

Message-sending:

Every vertex listens on port TCP, through this port it gets messages from the neighboring vertices in the graph. Message-sending between neighbors will be implemented **only** by the function send\_message(message,port,ips) in the file: *MessageSending.py*. You’ll find additional explanation about this function later on.

Task:

You should create a file named vetex.py. Within this file you should implement the function def vertex(ID). This function gets the parameters for the vertex from the file *input\_vetex\_[ID].txt*. The file contains lines, where in every line there is a certain parameter. The parameters of the input for vertex v are organized as follows:

1. Number of vertices in the graph
2. Listening port in TCP protocol
3. *n.num* value
4. The port on which a neighbor is listening in the TCP protocol
5. IP of a neighbor

Lines 4-5 repeated as the number of neighbors of the vertex. The input file ends with \*. Please view the example in the attached files.

The end of the algorithm:

In the end of the run, every vertex exports an output file named *output\_vertex\_[ID].txt* that includes 3 lines organized as follows (view an example in the attached files):

1. Distance from the origin vertex
2. Unique ID of the parent of v in the tree. (if v is an origin vertex this line should be *root*)
3. The sum of *num* values in the sub-rooted tree of v.

Requirements:

1. You are **only** allowed to import the following modules (exactly as written below):
   * *from socket import socket*
   * *from socket import error*
   * *from socket import AF\_INET*
   * *from socket import SOCK\_STREAM*
   * *from socket import SHUT\_RDWR*
   * *from socket import SHUT\_WR*
   * *from threading import Thread*
   * *from threading import Lock*
   * *from MessageSending import send\_message*
   * *import math*
2. You are not allowed to use global variables.
3. Communication between vertices can only be executed between neighbors in the graph.
4. In order for a vertex to process multiple messages simultaneously, after it gets a message-a new thread is opened for message treatment and the vertex immediately returns to listen on the port for additional messages. Notice that the only job of the listening thread is to receive the message and maybe execute short actions, it is not allowed to send and process information.
5. It is possible that two threads will try to access simultaneously to the resources of a vertex (for instance- the input files). For this reason, you should use a locking mechanism for the mutual resources. Notice that you should lock the resources for the minimal required duration.
6. The submitted code should work on every modern operating system. Therefore, make sure that you are not using specific operating system’s commands.
7. When a certain vertex finishes the algorithm, it stops listening on its port.

Attached files:

1. *simulate.py* - this file builds the input graph and simulates the environment on a single computer instead of on a net of computers. In addition, the file creates input files that are fitted for every vertex and finally checks the solution correctness. The file built as follows:
   * builds an input graph (random)
   * writes the information on the input files of the vertices.
   * Run every vertex in a separated thread.
   * Checks the input files and prints notes on the screen.
2. *MessageSending.py* – contains the *function send\_message(message,port,ip)*, which through this function you should send the messages. The function gets the message as a string, the port on which the message will be sent as an int, and the IP of the vertex as a string.
3. *files.zip* – the input and output files after running simulate.py with 8 vertices.

Additional notes:

1. Anything that wasn’t explicitly defined is for you to decide. (implementation of the algorithm, the data bases of the vertex etc.)
2. In *vertex.py* you are allowed to define functions and classes as you wish as long as the function *def vertex(ID)* will be defined.
3. It is possible that the file *simulate.py* and the function *send\_message* that you received will be changed in the check-up (number of vertices in the graph, the probability of an edge, message sending delays etc.)
4. Be strict about correct output names. The check-up will be automatically.

Suggestions:

* Before you start working on the code, try to create a TCP connection and communicate with other computer in order to understand the communication concept in Python.
* Algorithm wise, before you start writing the code make sure that the algorithm follows the instructions. Specifically, make sure that the algorithm adjusted to asynchronous environment and that every vertex knows when it finishes and make an output.

Goodluck!