Report

Python version is 3.7.3

How to implement

In order to implement LSR protocol. I divide the program in the following parts

1) process text

   in this part, the program will read the txt file and extract some information (e.g. nearby nodes & the distance away from the nodes), and the information will be stored in a tuple list.

2) send link-state information

   after extracting the txt file, it will send the processed information (the tuple list) to all directly nearby nodes. This is the first time to send and the content the original information without any updates.

3) Receive information

   Receive information from all nearby nodes and store them, wait for updating the existed content.

4) Update link-state information

   Update existed content from received information. As it is the UDP connection, unreliable transmission, if it times out, the program will ignore the received content and wait for next receiving.

5) Send the updated link-state information

   After updating, the program will send the updated information in order to make all nodes know the whole topology

6) Clean fail nodes

   The topology is dynamic, which means some nodes will fail. In order to ensure the following Dijkstra algorithm can run successfully, the tracks of failed nodes will be cleaned.

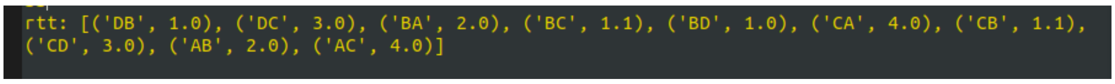
7) Dijkstra algorithm

   Use verified information to run Dijkstra algorithm, and find the shortest node to all nodes.

Data structure

In this program, I use tuple and list to simulate a graph.

E.g.



```
rtt: [('DB', 1.0), ('DC', 3.0), ('BA', 2.0), ('BC', 1.1), ('BD', 1.0), ('CA', 4.0), ('CB', 1.1),
('CD', 3.0), ('AB', 2.0), ('AC', 4.0)]
```

Each tuple in the list means one edge,

the first element in tuple means nodes, the second element means the distance from this information, I can make a graph.

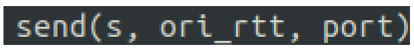
Deal with failed nodes

In order to check one node whether is failed or not, I design a mechanism. For example, we have node A & B, from the specification, the node A program knows the relation 'AB', but it doesn't know the relation 'BA' (Node B knows that). So once the node A program knows the relation 'BA', which means the Node B is in work.

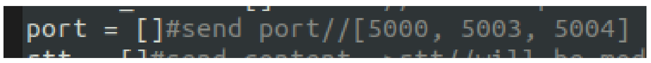
e.g. The content includes both `('BA', 2.0)` and `('AB', 2.0)`, it means the AB nodes are all in work. The program will iterate each tuple in the content list and check whether they appear in pair, after iterating and analyzing I can find which nodes are failed and clean all edges with them.

Restrict link-state broadcast

To avoid excessive link-state broadcast, a flooding routing algorithm is used in this program, once the host receive the broadcast link-state, it will only send the updated link-state information to the nodes which are directly linked with it, and the port information is provided by txt file.



```
send(s, ori_rtt, port)
```

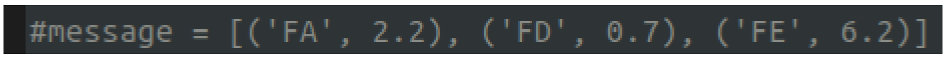


```
port = []#send port//[5000, 5003, 5004]
```

What's more, a filter mechanism is also introduced in my program, once the program receives the link-state broadcast from other nodes, it will process the received content and then send it instead of simply forward without any processes. It only sends the contents that are not included in the received content. In this way, there won't be duplicate content in the whole broadcast.

This is the example of one broadcast content.

The explanation of the content I have mention in data structure part.



```
#message = [('FA', 2.2), ('FD', 0.7), ('FE', 6.2)]
```

Extension & improvements

According to the network topology, each node has different amounts of nearby nodes, and the number of nearby nodes is closely related with the receive times. To ensure each node won’t miss the receiving content. The receiving times is designed to have relationships with its number of nearby ports, and each node will have different receiving times, which means it can not only ensure receive all contents but also avoid excessive receiving operations.