Report for project 2

Xunjie Liu 1613204

For question 1:

'get_graph(file)' function will read txt file to return a dictionary describing this graph. The format of the dictionary is showed below:

```
graph = dict() # {'1': [['2', 4], ['3', 2]], '2': [['3', 5]]}
```

{Node_name: [[neighbor_name1, cost], [neighbor_name2, cost]] and so on.

For question 2:

'dijkstra(graph, source='1')' function will read graph generated from question 1, and the source is default to '1', and it mainly follow the psudocode showed below:

```
1  function Dijkstra(Graph, source):
2      dist[source]  := 0                    // Distance from source to source
3      for each vertex v in Graph:           // Initializations
4          if v ≠ source
5              dist[v]  := infinity          // Unknown distance function from source to v
6              previous[v]  := undefined     // Previous node in optimal path from source
7          end if
8          add v to Q                        // All nodes initially in Q (unvisited nodes)
9      end for
10
11     while Q is not empty:                 // The main loop
12         u := vertex in Q with min dist[u] // Source node in first case
13         remove u from Q
14
15         for each neighbor v of u:          // where v has not yet been removed from Q.
16             alt := dist[u] + length(u, v)
17             if alt < dist[v]:              // A shorter path to v has been found
18                 dist[v]  := alt
19                 previous[v]  := u
20             end if
21         end for
22     end while
23     return dist[], previous[]
24 end function
```

In order to realize priority queue, I used heapq module in python. This function will return two dictionaries: distance and previous.

For 'distance', its keys are destination nodes and values are corresponding minimum cost for the source. For 'previous', its keys are destination nodes and values are previous nodes in the shortest path from the source. I generated shortest paths from source to destination nodes using 'distance' and 'previous', then write them into Route_Map.json file. In JSON file, it is a dictionary whose format is "route_map[node] = [path, cost]"

```
From 1, To 2, Path: 1->3->2, Minimum Cost: 5
From 1, To 3, Path: 1->3, Minimum Cost: 3
From 1, To 4, Path: 1->3->4, Minimum Cost: 6
From 1, To 5, Path: 1->3->5, Minimum Cost: 7
From 1, To 6, Path: 1->3->4->6, Minimum Cost: 9

Read route information into JSON file, with name as 'Route_Map.json'
```

For question 3:

Protocol for question 3:

1. Port number of each process (router) is equal to 12000 + node_name.
2. Routers/Processes should only actively start a TCP connection to Routers/Processes with larger number node name.
3. Once a Router/Process receive a TCP connection, save data into public container and send its own routing table.
4. For each process, the input parameter is a Router object which has the same attributes and relax (update the routing table) as the real router, and this process will generate two threads, one for listening, another for actively start TCP connections.
5. Divide neighbours of a Router into two arrays: connect_to and connect_in. Connect_to stores the neighbours which has larger number node names. Connect_in stores the neighbours with smaller number node names.
6. For Listener thread in each process, once it confirms that every neighbour in Connect_in has actively sent TCP connection to it, this thread will end.
7. For Messager thread who actively starts TCP connections, after it sends TCP connections to every neighbour in Connect_to, this thread will end.
8. For Messager, if it cannot connect to the neighbours, sleep for 2 seconds and retry.
9. Every time Listener or Messager thread receives a routing table from neighbours, immediately do the update operation in Router.
10. In order to initialize a Router object, write a blank routing table into a JSON file with a file name as its node name.

The final result in JSON file should be similar to this:

```
{'1': [0, '1'], '2': [5, '3'], '3': [3, '3'], '4': [6, '3'], '5': [7, '3'], '6': [9, '3']}
{'1': [5, '3'], '2': [0, '2'], '3': [2, '3'], '4': [5, '4'], '5': [6, '3'], '6': [8, '4']}
{'1': [3, '1'], '2': [2, '2'], '3': [0, '3'], '4': [3, '4'], '5': [4, '5'], '6': [6, '4']}
{'1': [6, '3'], '2': [5, '2'], '3': [3, '3'], '4': [0, '4'], '5': [2, '5'], '6': [3, '6']}
{'1': [7, '3'], '2': [6, '3'], '3': [4, '3'], '4': [2, '4'], '5': [0, '5'], '6': [5, '6']}
{'1': [9, '4'], '2': [8, '4'], '3': [6, '4'], '4': [3, '4'], '5': [5, '5'], '6': [0, '6']}
```

Which means: Destination: [cost, next_node]

I wrote both parts in one python file, so you can just input “project2.py nodeExample.txt” in command line.

```
D:\Study\Year 3\CSE205\Project 2>project2.py nodeExample.txt
```

```
if __name__ == '__main__':
    file_name = sys.argv[1]
    main_1(file_name) # program for question 1 and question 2

    main_2(file_name) # program for question 2
```

Here is the result, ‘Route_Map.json’ is for part 1, others are node file for part 2 (read nodeExample.txt).