COMP3234B Computer and Communication Networks

Lab 5: Bellman-Ford Routing Algorithm

Introduction

In this lab, we will use Python multithreading to implement a routing simulator with the Bellman-Ford algorithm. We have introduced in lectures that Bellman-Ford algorithm is an iterative, asynchronous and distributed routing algorithm executed by all nodes together to compute the least-cost paths from each node to all other nodes. In this lab exercise, we are going to implement the Bellman-Ford algorithm, and to try out good news travels fast, count to infinity, and poisoned reverse by adjusting the link costs.

Lab Exercise: Simple routing simulator with Bellman-Ford algorithm

We now implement a multi-threading routing simulator where we use a thread to simulate a router (aka node), which computes the least-cost paths from the node to all other nodes using the Bellman-Ford algorithm.

Step 1: Download lab5_materials.zip from Moodle. Unzip it and you will find 5 files provided: routing.py, topology1.txt, topology2.txt, topology3.txt, topology4.txt.

Step 2: Open routing.py using a text editor, which contains the implementation of the routing simulator (you need to implement some functions to make it work). The simulator reads a file (e.g. topology1.txt), which contains the link costs between nodes in a network, in the format of N rows with N elements in each row (i.e., a NXN matrix), where N is the number of nodes in the network and the element in the (a,b) position represents the cost of the link between node a and node b (the cost is "inf" if no link between the two nodes). In topology1.txt, we provide the link costs of the simple network on page 18 of 11_Network_III_COMP3234_s2024.pdf. In topology2.txt, we provide link costs in a network with 3 nodes, which resembles the network on page 20 of 11_Network_III_COMP3234_s2024.pdf. In topology3.txt, we provide link costs in a network with 4 nodes. In topology4.txt, we provide link costs in a network with 10 nodes. The simulator is to be started by "python3 routing.py <topology_file_name>".

Step 3: Check out the provided implementation in routing.py. Explanations of some key code steps are as follows.

a. In the main function, we call function import_topology to import the link costs from the topology file into cost_matrix, which calls loadtxt function in the numpy module to load the text file (https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html).

You can install the numpy module using the following command on both Windows and macOS/Linux.

```
pip3 install numpy
```

b. After reading the cost matrix from the topology file, we obtain the total number of nodes and create threads to simulate the nodes. We also create a few global data structures:
   - routers. It is a dictionary in which the key is the router id and the value is an instance of the Router class. The dictionary is a mapping of router's id to the corresponding router thread. It can be used for simulating exchange of distance vectors between routers. It is worth noting that a lock should be acquired before any read/write from/into any router's distance_vector_table (The function send_distvec in the RouterThread class is a good example to illustrate how to use dictionary routers to send distance vectors).
   - global_mutex, which is a lock on cost_matrix. The lock should be acquired before updating/checking updates of cost_matrix.

c. In the init function of RouterThread class, we use deepcopy function in the copy module (https://www.python-course.eu/python3_deep_copy.php) to maintain a local copy of the link costs between the node and its neighbors. The node also initializes the neighbor list, successor list (containing successor on the current least-cost path to each destination). In the initialize_distance_vector function, the node initializes its distance vector list (including its own distance vector and distance vecors of neighbors). Then it sends its distance vector to neighbors, i.e., by appending its distance vector into the target neighbor's distance_vector_table.
   - TODO: You should complete initiation of the distance vector list at each router where we marked two "TODO"s in the init function, following the Bellman-Ford algorithm's initialization steps on page 17 of 11_Network_III_COMP3234_s2024.pdf.

d. In the run function of RouterThread class, we check two events every 1 second: (1) whether the router has received distance vector(s) from its neighbors, by calling is_neighbor_distvect_received function; (2) whether any link cost between itself and its neighbors has been changed, by calling is_neighbor_link_changed function. If either of the two events happens, we call update_distvec function to update the distance vector of this router as well as its successor list.

e. TODO: You should complete the function update_distvec to update the distance vector of this router and its successor list where we marked "TODO" in the update_distvec function, following the Bellman-Ford algorithm's steps on page 17 of 11_Network_III_COMP3234_s2024.pdf (you should add the successor update step yourself).

Step 4: In the main function, we also provide code to support the following commands for checking the status of the threads, change link cost and enable poisoned reverse:

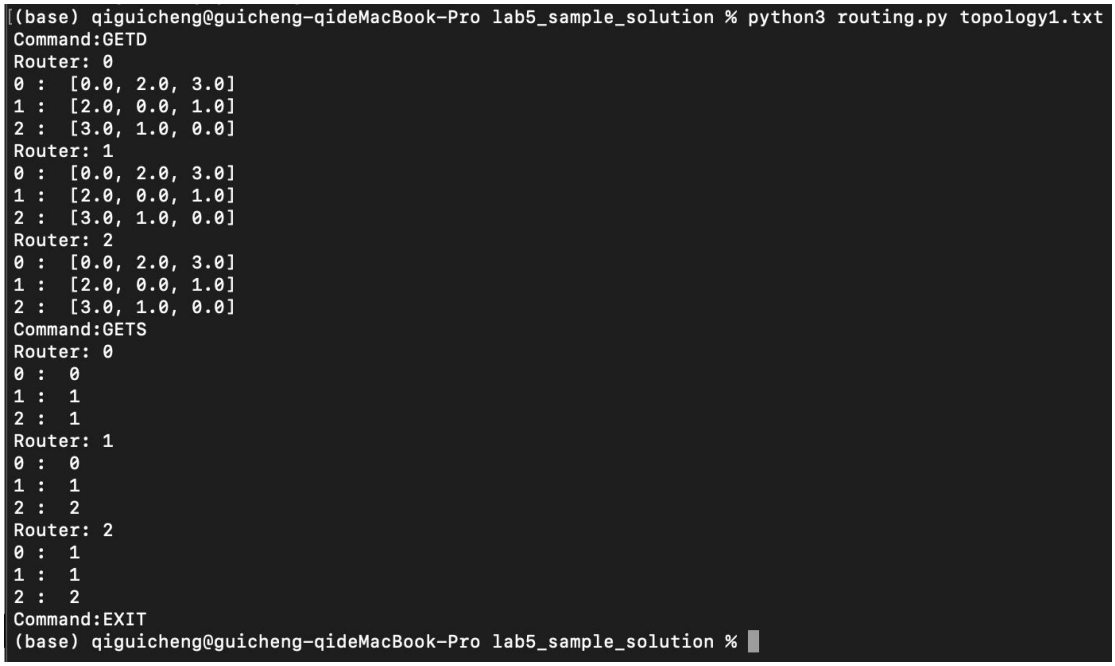
a. GETD: it calls the auxiliary function get_d in each thread to print out distance vectors each router maintains.
b. GETS: it calls the auxiliary function get_s in each thread to print out the successor list of each router towards all destinations.
c. CHANGECOST: it enables changing the link cost between a specific node pair, by asking the user to input indices of the start node and the end node, and then specify the new link cost between the two nodes.
d. POISONEDREVERSE: it enables poisoned reverse algorithm in the router threads. By default, poisoned_reverse flag is set to False in each thread. By typing this command, we set poisoned_reverse to True, and our provided poisoned reverse algorithm in update_distvec function will be activated, that tells a neighbor an infinity cost to a destination if the router uses that neighbor as next hop towards the destination.
e. EXIT: the program exits after receiving this command.

Step 5: Test your program as follows:

a. Launch a terminal and switch to the directory of lab5. Run the routing program as follows:

```
python3 routing.py topology1.txt
```

When the program runs, it will compute the least-cost paths on the topology included in topology1.txt. You can use commands GETD and GETS to check the results. Sample output can be seen in the following screenshot:



b. Run the routing program as follows:

```
python3 routing.py topology2.txt
```

When the program runs, it will compute the least-cost paths on the topology included in topology2.txt. Similarly, you can use commands GETD and GETS to check the results. Next you can type CHANGECOST command, input two nodes you choose and a new cost along the link between the two nodes (e.g., change the cost between node 0 and node 1 to 1 following “link cost change: scenario 1” on page 20 of 11_Network_III_COMP3234_s2024.pdf); then you can use GETD and GETS to check the new results. Sample output as follows:

```
(base) qiguicheng@guicheng-qideMacBook-Pro lab5_sample_solution % python3 routing.py topology2.txt
Command:GETD
Router: 0
0 :  [0.0, 3.0, 4.0]
1 :  [3.0, 0.0, 1.0]
2 :  [4.0, 1.0, 0.0]
Router: 1
0 :  [0.0, 3.0, 4.0]
1 :  [3.0, 0.0, 1.0]
2 :  [4.0, 1.0, 0.0]
Router: 2
0 :  [0.0, 3.0, 4.0]
1 :  [3.0, 0.0, 1.0]
2 :  [4.0, 1.0, 0.0]
Command:GETS
Router: 0
0 :  0
1 :  1
2 :  1
Router: 1
0 :  0
1 :  1
2 :  2
Router: 2
0 :  1
1 :  1
2 :  2
Command:CHANGECOST
start node:0
end node:1
updated cost:1
Command:GETD
Router: 0
0 :  [0.0, 1.0, 2.0]
1 :  [1.0, 0.0, 1.0]
2 :  [2.0, 1.0, 0.0]
Router: 1
0 :  [0.0, 1.0, 2.0]
1 :  [1.0, 0.0, 1.0]
2 :  [2.0, 1.0, 0.0]
Router: 2
0 :  [0.0, 1.0, 2.0]
1 :  [1.0, 0.0, 1.0]
2 :  [2.0, 1.0, 0.0]
Command:GETS
Router: 0
0 :  0
1 :  1
2 :  1
Router: 1
0 :  0
1 :  1
2 :  2
Router: 2
0 :  1
1 :  1
2 :  2
Command:EXIT
(base) qiguicheng@guicheng-qideMacBook-Pro lab5_sample_solution %
```

Note that if you type GETD or GETS too soon after the program starts or after CHANGECOST, the algorithm may not have converged and you may observe intermedia results; if so, you can wait a few more seconds and type GETD or GETS again, and then you will observe the final results. Especially, in this “good news travels fast” scenario, the algorithm converges fast.

c. Run the routing program as follows:

```
python3 routing.py topology2.txt
```

Again, we compute the least-cost paths on the topology included in topology2.txt. This time we are going to use CHANGECOST command to increase a link cost (e.g., change the cost between node 0 and node 1 to 60 following “link cost change: scenario 2” on page 25 of 11_Network_III_COMP3234_s2024.pdf); then you can use GETD and GETS to check the new results. In this “bad news travels slowly” or “count to infinity” scenario, you can easily observe intermedia results when typing GETD or GETS and need to wait for a while before obtaining the final results, due to many iterations the algorithm runs before convergence. Sample output as follows:

```
(base) qiguicheng@guicheng-qideMacBook-Pro lab5_sample_solution % python3 routing.py topology2.txt
Command:CHANGECOST
start node:0
end node:1
updated cost:60
Command:GETD
Router: 0
0 :  [0.0, 51.0, 50.0]
1 :  [5.0, 0.0, 1.0]
2 :  [4.0, 1.0, 0.0]
Router: 1
0 :  [0.0, 51.0, 50.0]
1 :  [7.0, 0.0, 1.0]
2 :  [6.0, 1.0, 0.0]
Router: 2
0 :  [0.0, 51.0, 50.0]
1 :  [5.0, 0.0, 1.0]
2 :  [6.0, 1.0, 0.0]
Command:GETS
Router: 0
0 :  0
1 :  2
2 :  2
Router: 1
0 :  2
1 :  1
2 :  2
Router: 2
0 :  1
1 :  1
2 :  2
Command:GETD
Router: 0
0 :  [0.0, 51.0, 50.0]
1 :  [21.0, 0.0, 1.0]
2 :  [22.0, 1.0, 0.0]
Router: 1
0 :  [0.0, 51.0, 50.0]
1 :  [23.0, 0.0, 1.0]
2 :  [22.0, 1.0, 0.0]
Router: 2
0 :  [0.0, 51.0, 50.0]
1 :  [21.0, 0.0, 1.0]
2 :  [22.0, 1.0, 0.0]
Command:GETS
Router: 0
0 :  0
1 :  2
2 :  2
Router: 1
0 :  2
1 :  1
2 :  2
Router: 2
0 :  1
1 :  1
2 :  2
Command:GETD
Router: 0
0 :  [0.0, 51.0, 50.0]
1 :  [51.0, 0.0, 1.0]
2 :  [50.0, 1.0, 0.0]
Router: 1
0 :  [0.0, 51.0, 50.0]
1 :  [51.0, 0.0, 1.0]
2 :  [50.0, 1.0, 0.0]
Router: 2
0 :  [0.0, 51.0, 50.0]
1 :  [51.0, 0.0, 1.0]
2 :  [50.0, 1.0, 0.0]
Command:GETS
Router: 0
0 :  0
1 :  2
2 :  2
Router: 1
0 :  2
1 :  1
2 :  2
Router: 2
0 :  0
1 :  1
2 :  2
Command:EXIT
```

d. Run the routing program as follows:

```
python3 routing.py topology2.txt
```

This time we are going to enable poisoned reverse algorithm by typing POISONEDREVERSE command before using CHANGECOST to increase the link cost. We will observe much faster convergence, as shown in the screenshot below:

```
(base) qiguicheng@guicheng-qideMacBook-Pro lab5_sample_solution % python3 routing.py topology2.txt
Command:POISONEDREVERSE
Command:CHANGECOST
start node:0
end node:1
updated cost:60
Command:GETD
Router: 0
0 :  [0.0, 51.0, 50.0]
1 :  [51.0, 0.0, 1.0]
2 :  [50.0, 1.0, 0.0]
Router: 1
0 :  [0.0, 51.0, 50.0]
1 :  [51.0, 0.0, 1.0]
2 :  [50.0, 1.0, 0.0]
Router: 2
0 :  [0.0, inf, 50.0]
1 :  [inf, 0.0, 1.0]
2 :  [50.0, 1.0, 0.0]
Command:GETS
Router: 0
0 :  0
1 :  2
2 :  2
Router: 1
0 :  2
1 :  1
2 :  2
Router: 2
0 :  0
1 :  1
2 :  2
Command:EXIT
(base) qiguicheng@guicheng-qideMacBook-Pro lab5_sample_solution %
```

e. Run the routing program as follows:

```
python3 routing.py topology3.txt
```

You can try various combination of commands to evaluate your program on the topology included in topology3.txt. Some sample outputs are as follows.

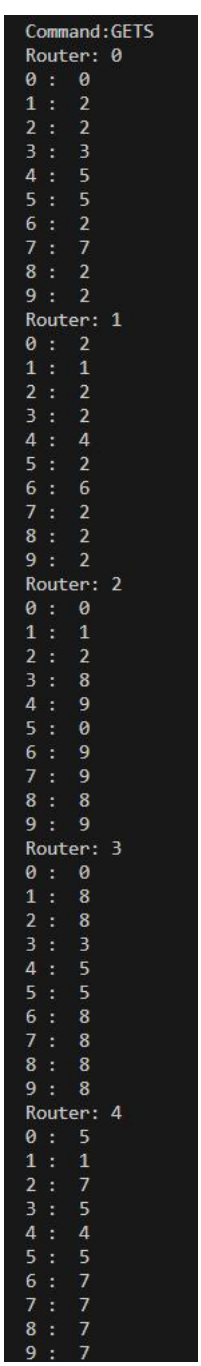
```
(base) qiguicheng@guicheng-qideMacBook-Pro lab5_sample_solution % python routing.py topology3.txt
Command:GETD
Router: 0
0 :  [0.0, 1.0, 6.0, 3.0]
1 :  [1.0, 0.0, 5.0, 2.0]
Router: 1
0 :  [0.0, 1.0, 6.0, 3.0]
1 :  [1.0, 0.0, 5.0, 2.0]
2 :  [6.0, 5.0, 0.0, 3.0]
3 :  [3.0, 2.0, 3.0, 0.0]
Router: 2
1 :  [1.0, 0.0, 5.0, 2.0]
2 :  [6.0, 5.0, 0.0, 3.0]
3 :  [3.0, 2.0, 3.0, 0.0]
Router: 3
1 :  [1.0, 0.0, 5.0, 2.0]
2 :  [6.0, 5.0, 0.0, 3.0]
3 :  [3.0, 2.0, 3.0, 0.0]
Command:GETS
Router: 0
0 :  0
1 :  1
2 :  1
3 :  1
Router: 1
0 :  0
1 :  1
2 :  3
3 :  3
Router: 2
0 :  3
1 :  3
2 :  2
3 :  3
Router: 3
0 :  1
1 :  1
2 :  2
3 :  3
Command:EXIT
```

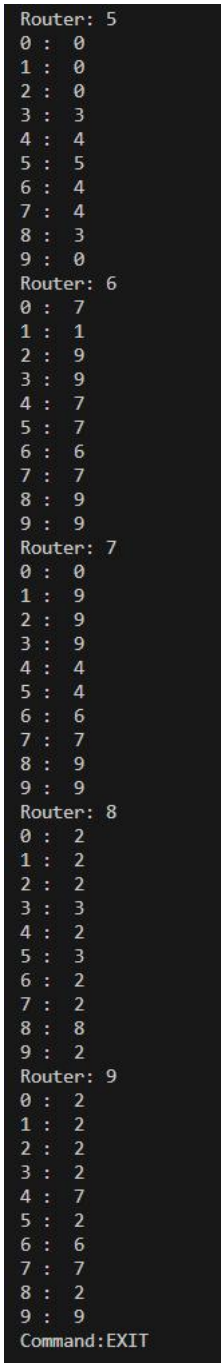
f. Run the routing program as follows:

```
python3 routing.py topology4.txt
```

Similarly, you can try various combination of commands to evaluate your program on the topology included in topology4.txt. Some sample outputs are as follows.

```
(base) C:\Users\qgc19\Desktop>python routing.py topology4.txt
Command:GETD
Router: 0
0 :  [0.0, 11.0, 4.0, 5.0, 23.0, 12.0, 17.0, 12.0, 5.0, 10.0]
2 :  [4.0, 7.0, 0.0, 3.0, 21.0, 16.0, 13.0, 10.0, 1.0, 6.0]
3 :  [5.0, 10.0, 3.0, 0.0, 24.0, 13.0, 16.0, 13.0, 2.0, 9.0]
5 :  [12.0, 23.0, 16.0, 13.0, 11.0, 0.0, 27.0, 22.0, 15.0, 22.0]
7 :  [12.0, 17.0, 10.0, 13.0, 11.0, 22.0, 5.0, 0.0, 11.0, 4.0]
8 :  [5.0, 8.0, 1.0, 2.0, 22.0, 15.0, 14.0, 11.0, 0.0, 7.0]
Router: 1
1 :  [11.0, 0.0, 7.0, 10.0, 17.0, 23.0, 18.0, 17.0, 8.0, 13.0]
2 :  [4.0, 7.0, 0.0, 3.0, 21.0, 16.0, 13.0, 10.0, 1.0, 6.0]
4 :  [23.0, 17.0, 21.0, 24.0, 0.0, 11.0, 16.0, 11.0, 22.0, 15.0]
6 :  [17.0, 18.0, 13.0, 16.0, 16.0, 27.0, 0.0, 5.0, 14.0, 7.0]
8 :  [5.0, 8.0, 1.0, 2.0, 22.0, 15.0, 14.0, 11.0, 0.0, 7.0]
9 :  [10.0, 13.0, 6.0, 9.0, 15.0, 22.0, 7.0, 4.0, 7.0, 0.0]
Router: 2
0 :  [0.0, 11.0, 4.0, 5.0, 23.0, 12.0, 17.0, 12.0, 5.0, 10.0]
1 :  [11.0, 0.0, 7.0, 10.0, 17.0, 23.0, 18.0, 17.0, 8.0, 13.0]
2 :  [4.0, 7.0, 0.0, 3.0, 21.0, 16.0, 13.0, 10.0, 1.0, 6.0]
3 :  [5.0, 10.0, 3.0, 0.0, 24.0, 13.0, 16.0, 13.0, 2.0, 9.0]
5 :  [12.0, 23.0, 16.0, 13.0, 11.0, 0.0, 27.0, 22.0, 15.0, 22.0]
8 :  [5.0, 8.0, 1.0, 2.0, 22.0, 15.0, 14.0, 11.0, 0.0, 7.0]
9 :  [10.0, 13.0, 6.0, 9.0, 15.0, 22.0, 7.0, 4.0, 7.0, 0.0]
Router: 3
0 :  [0.0, 11.0, 4.0, 5.0, 23.0, 12.0, 17.0, 12.0, 5.0, 10.0]
2 :  [4.0, 7.0, 0.0, 3.0, 21.0, 16.0, 13.0, 10.0, 1.0, 6.0]
3 :  [5.0, 10.0, 3.0, 0.0, 24.0, 13.0, 16.0, 13.0, 2.0, 9.0]
5 :  [12.0, 23.0, 16.0, 13.0, 11.0, 0.0, 27.0, 22.0, 15.0, 22.0]
8 :  [5.0, 8.0, 1.0, 2.0, 22.0, 15.0, 14.0, 11.0, 0.0, 7.0]
9 :  [10.0, 13.0, 6.0, 9.0, 15.0, 22.0, 7.0, 4.0, 7.0, 0.0]
Router: 4
1 :  [11.0, 0.0, 7.0, 10.0, 17.0, 23.0, 18.0, 17.0, 8.0, 13.0]
4 :  [23.0, 17.0, 21.0, 24.0, 0.0, 11.0, 16.0, 11.0, 22.0, 15.0]
5 :  [12.0, 23.0, 16.0, 13.0, 11.0, 0.0, 27.0, 22.0, 15.0, 22.0]
7 :  [12.0, 17.0, 10.0, 13.0, 11.0, 22.0, 5.0, 0.0, 11.0, 4.0]
Router: 5
0 :  [0.0, 11.0, 4.0, 5.0, 23.0, 12.0, 17.0, 12.0, 5.0, 10.0]
2 :  [4.0, 7.0, 0.0, 3.0, 21.0, 16.0, 13.0, 10.0, 1.0, 6.0]
3 :  [5.0, 10.0, 3.0, 0.0, 24.0, 13.0, 16.0, 13.0, 2.0, 9.0]
4 :  [23.0, 17.0, 21.0, 24.0, 0.0, 11.0, 16.0, 11.0, 22.0, 15.0]
5 :  [12.0, 23.0, 16.0, 13.0, 11.0, 0.0, 27.0, 22.0, 15.0, 22.0]
Router: 6
1 :  [11.0, 0.0, 7.0, 10.0, 17.0, 23.0, 18.0, 17.0, 8.0, 13.0]
6 :  [17.0, 18.0, 13.0, 16.0, 16.0, 27.0, 0.0, 5.0, 14.0, 7.0]
7 :  [12.0, 17.0, 10.0, 13.0, 11.0, 22.0, 5.0, 0.0, 11.0, 4.0]
9 :  [10.0, 13.0, 6.0, 9.0, 15.0, 22.0, 7.0, 4.0, 7.0, 0.0]
Router: 7
0 :  [0.0, 11.0, 4.0, 5.0, 23.0, 12.0, 17.0, 12.0, 5.0, 10.0]
4 :  [23.0, 17.0, 21.0, 24.0, 0.0, 11.0, 16.0, 11.0, 22.0, 15.0]
6 :  [17.0, 18.0, 13.0, 16.0, 16.0, 27.0, 0.0, 5.0, 14.0, 7.0]
7 :  [12.0, 17.0, 10.0, 13.0, 11.0, 22.0, 5.0, 0.0, 11.0, 4.0]
9 :  [10.0, 13.0, 6.0, 9.0, 15.0, 22.0, 7.0, 4.0, 7.0, 0.0]
Router: 8
0 :  [0.0, 11.0, 4.0, 5.0, 23.0, 12.0, 17.0, 12.0, 5.0, 10.0]
1 :  [11.0, 0.0, 7.0, 10.0, 17.0, 23.0, 18.0, 17.0, 8.0, 13.0]
2 :  [4.0, 7.0, 0.0, 3.0, 21.0, 16.0, 13.0, 10.0, 1.0, 6.0]
3 :  [5.0, 10.0, 3.0, 0.0, 24.0, 13.0, 16.0, 13.0, 2.0, 9.0]
8 :  [5.0, 8.0, 1.0, 2.0, 22.0, 15.0, 14.0, 11.0, 0.0, 7.0]
Router: 9
1 :  [11.0, 0.0, 7.0, 10.0, 17.0, 23.0, 18.0, 17.0, 8.0, 13.0]
2 :  [4.0, 7.0, 0.0, 3.0, 21.0, 16.0, 13.0, 10.0, 1.0, 6.0]
3 :  [5.0, 10.0, 3.0, 0.0, 24.0, 13.0, 16.0, 13.0, 2.0, 9.0]
6 :  [17.0, 18.0, 13.0, 16.0, 16.0, 27.0, 0.0, 5.0, 14.0, 7.0]
7 :  [12.0, 17.0, 10.0, 13.0, 11.0, 22.0, 5.0, 0.0, 11.0, 4.0]
9 :  [10.0, 13.0, 6.0, 9.0, 15.0, 22.0, 7.0, 4.0, 7.0, 0.0]
```





Submission:

You should submit the following files

(1) routing.py
(2) topology1.txt
(3) topology2.txt
(4) topology3.txt
(5) topology4.txt

by compressing them into a lab5-yourUID.zip file and submit it on Moddle before 23:59 Wednesday April 3, 2024:

(1) Login Moodle.
(2) Find “Labs” in the left column and click “Lab 5”.
(3) Click “Add submission”, browse your .zip file and save it. Done.
(4) You will receive an automatic confirmation email, if the submission was successful.
(5) You can “Edit submission” to your already submitted file, but ONLY before the deadline.