Distance Vector Routing Protocol

Name: Kaushal Lodd

Roll No: BT19CSE052

Instructions to run the code

- Download the zip folder of the codebase and unzip.
- Open terminal and change directory to that of the codebase folder.
- type `python dvr.py <input_filename.txt>` in the terminal to run the code.
- There are two input files, namely `input1.txt` and `input2.txt`

Working of the code

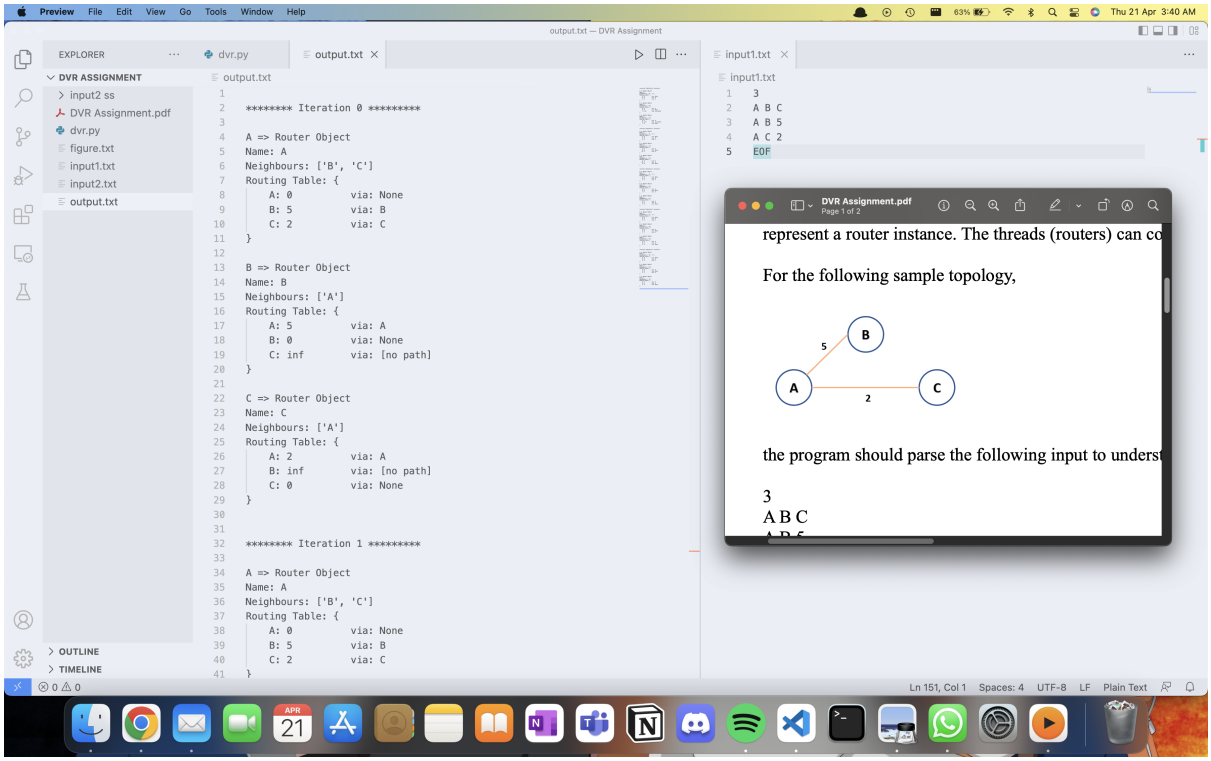
- Designed and implemented a `Router` class which contains attributes and methods like -
  - Name of the Router
  - List of Neighbouring Routers
  - Routing Table
    - The Routing Table of each router contains all the destination routers, the minimum cost required to route to the destination routers and the path via which it routes to the destination routers.
  - Shared queue (containing routing tables from other routers)
  - queue_lock to acquire and release the lock for the shared queue
- The `input_parser` function parses the input text file and creates Router instances and assigns cost to the links between the routers. It implements the Initialization phase of the DV (Distance Vector) Algorithm.
- The `threaded` function is the function invoked when a thread is created for each instance of a router. This method invokes the `add_to_queue` method which shares

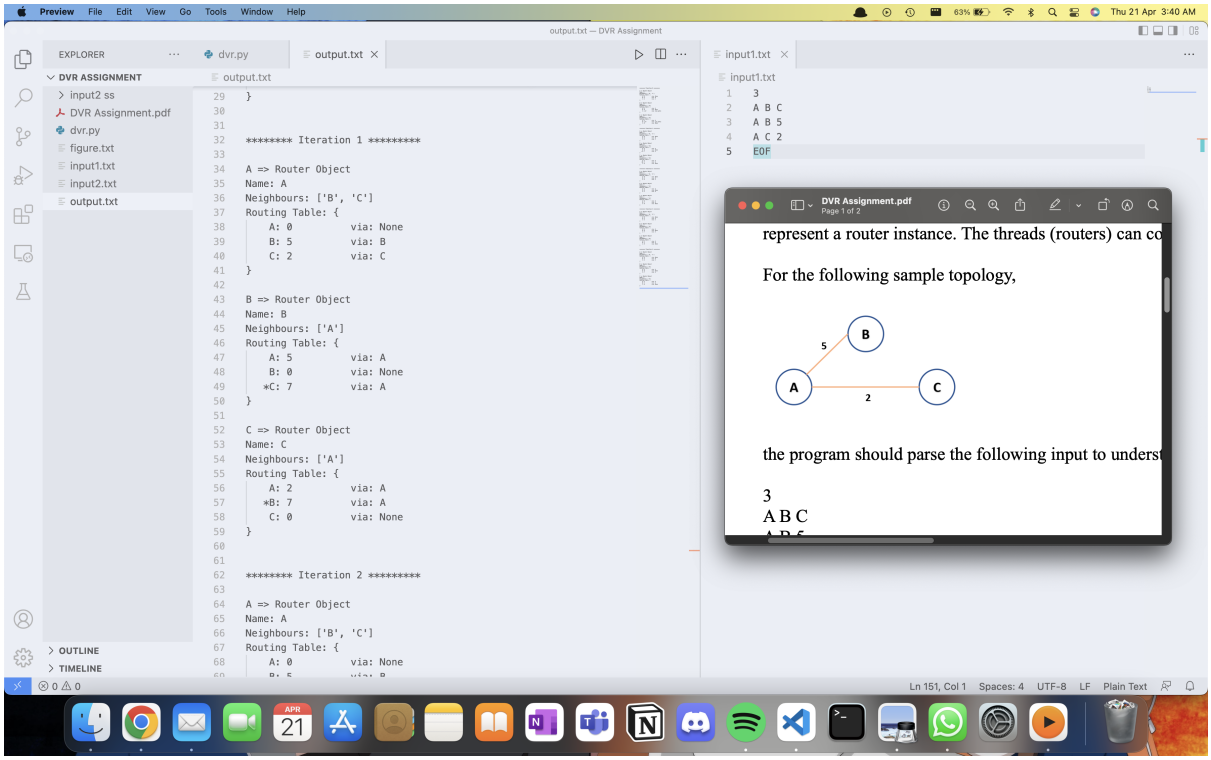
the current router's routing table to all other routers. The `threaded` function then waits for the current router to recieve the routing tables from other routers'.

- After storing the router tables of other routers in `Router.queue`, we will apply *Bellman-Ford's* equation to calculate least-costly path from the current router to all other routers in the network.
- We then update the current routing table according to *Bellman-Ford's* equation and wait for 2 seconds before running the next iteration of the algorithm on each router.
- **Note**: Each iteration of the algorithm is run on each router concurrently (parallelly) using multiple threads.
- Finally, over multiple iterations (in our case 4), the least-cost required to route to all the other routers is calculated and stored in the Routing Table.

Testing

- The DV Algorithm was tested on two input files, namely `input1.txt` and `input2.txt`
- `input1.txt` contains three routers with two links and is a fairly simple network. This was the example network provided to us in the Problem Statement. The algorithm calculates the least-cost path for all routers to all other routers in two iterations.
- The below screenshots show the Routing Table of each Router in various iterations.

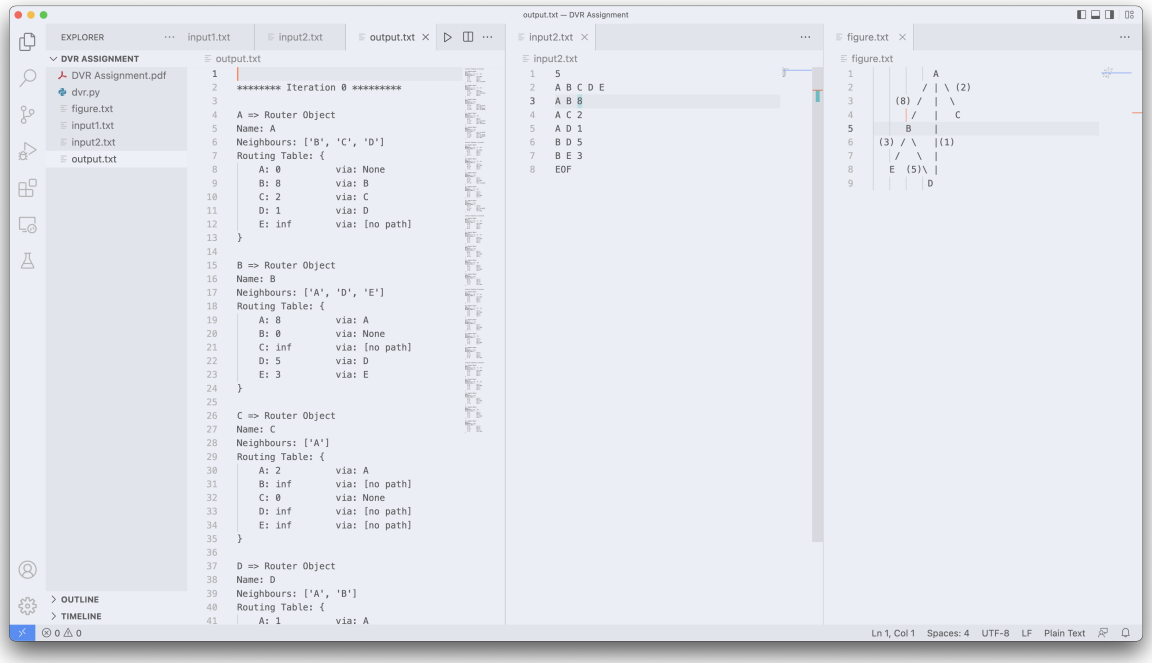


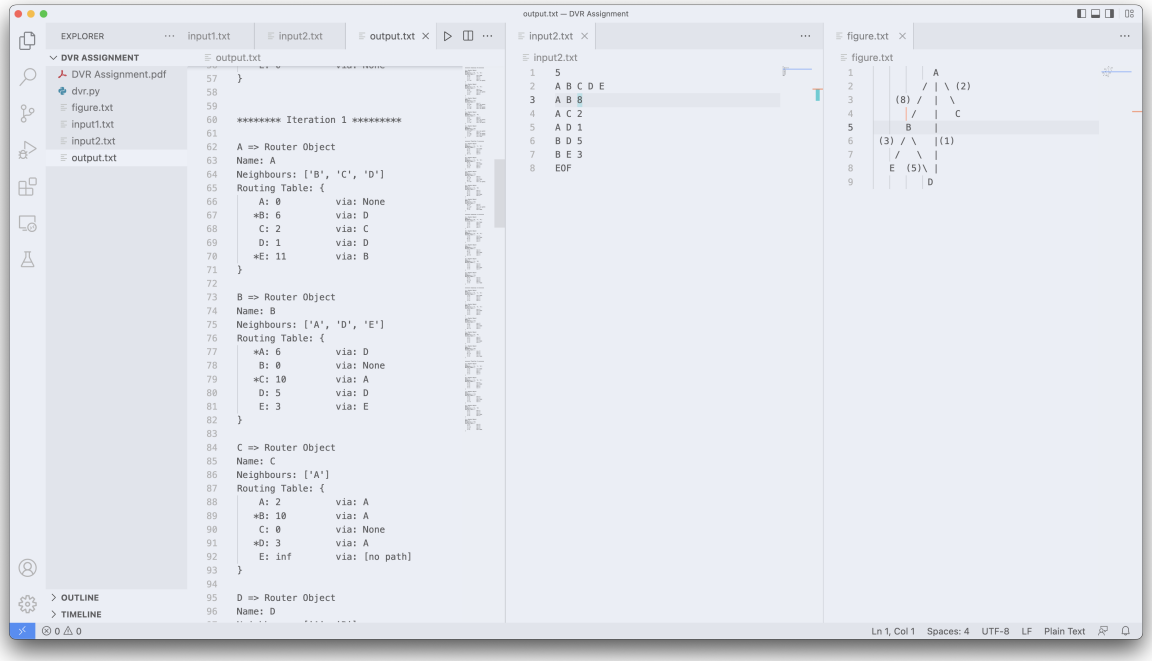


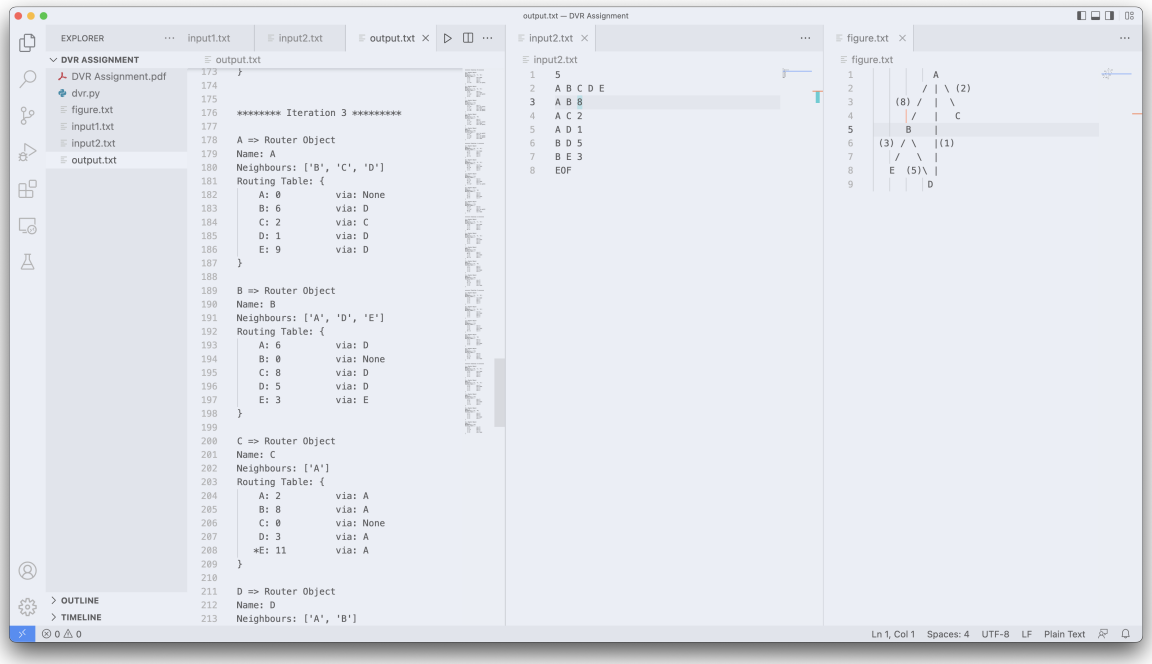
- `input2.txt` contains five routers and five links. This network is slightly more complicated and involves routing through multiple routers to get a least-cost

path. This network serves as a very good example to show how the shortest path isn’t necessarily the least-cost path.

- The graphical (pictorial) representation of the network in `input2.txt` can be found in `figure.txt`




output.txt — DVR Assignment
EXPLORER
input1.txt
input2.txt
output.txt
figure.txt
DVR ASSIGNMENT
DVR Assignment.pdf
dvr.py
figure.txt
input1.txt
input2.txt
output.txt
OUTLINE
TIMELINE
******** Iteration 1 *********
A => Router Object
Name: A
Neighbours: ['B', 'C', 'D']
Routing Table: {
A: 0 via: None
*B: 6 via: D
C: 2 via: C
D: 1 via: D
*E: 11 via: B
}
B => Router Object
Name: B
Neighbours: ['A', 'D', 'E']
Routing Table: {
*A: 6 via: D
B: 0 via: None
*C: 10 via: A
D: 5 via: D
E: 3 via: E
}
C => Router Object
Name: C
Neighbours: ['A']
Routing Table: {
A: 2 via: A
*B: 10 via: A
C: 0 via: None
*D: 3 via: A
E: inf via: [no path]
}
D => Router Object
Name: D
5
A B C D E
A B 8
A C 2
A D 1
B D 5
B E 3
EOF
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Plain Text


output.txt — DVR Assignment
EXPLORER
input1.txt
input2.txt
output.txt
figure.txt
DVR ASSIGNMENT
DVR Assignment.pdf
dvr.py
figure.txt
input1.txt
input2.txt
output.txt
OUTLINE
TIMELINE
******** Iteration 3 *********
A => Router Object
Name: A
Neighbours: ['B', 'C', 'D']
Routing Table: {
A: 0 via: None
B: 6 via: D
C: 2 via: C
D: 1 via: D
E: 9 via: D
}
B => Router Object
Name: B
Neighbours: ['A', 'D', 'E']
Routing Table: {
A: 6 via: D
B: 0 via: None
C: 8 via: D
D: 5 via: D
E: 3 via: E
}
C => Router Object
Name: C
Neighbours: ['A']
Routing Table: {
A: 2 via: A
B: 8 via: A
C: 0 via: None
D: 3 via: A
*E: 11 via: A
}
D => Router Object
Name: D
Neighbours: ['A', 'B']
5
A B C D E
A B 8
A C 2
A D 1
B D 5
B E 3
EOF
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Plain Text