

# Distance Vector Routing Protocol

Name: Abhinav Konkal

Roll No: BT20CSE069

---

## 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 receive 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.

```

1  **** Iteration 0 ****
2
3
4  A => Router Object
5  Name: A
6  Neighbours: ['B', 'C']
7  Routing Table: {
8      A: 0          via: None
9      B: 5          via: B
10     C: 2          via: C
11  }
12
13  B => Router Object
14  Name: B
15  Neighbours: ['A']
16  Routing Table: {
17      A: 5          via: A
18      B: 0          via: None
19      C: inf        via: [no path]
20  }
21
22  C => Router Object
23  Name: C
24  Neighbours: ['A']
25  Routing Table: {
26      A: 2          via: A
27      B: inf        via: [no path]
28      C: 0          via: None
29  }
30
31

```

```

1
2
3
4  A => Router Object
5  Name: A
6  Neighbours: ['B', 'C']
7  Routing Table: {
8      A: 0          via: None
9      B: 5          via: B
10     C: 2          via: C
11  }
12
13  B => Router Object
14  Name: B
15  Neighbours: ['A']
16  Routing Table: {
17      A: 5          via: A
18      B: 0          via: None
19      C: 7          via: A
20  }
21
22  C => Router Object
23  Name: C
24  Neighbours: ['A']
25  Routing Table: {
26      A: 2          via: A
27      B: 7          via: A
28      C: 0          via: None
29  }
30
31

```

- `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`

The figure.txt file displays a network graph with five nodes labeled A, B, C, D, and E. Node A is at the top, connected to B (via a double slash), C (via a single slash), and D (via a single backslash). Node B is connected to A (via a double slash), C (via a single slash), and E (via a single backslash). Node C is connected to B (via a single slash) and A (via a double slash). Node D is connected to A (via a single backslash) and B (via a double slash). Node E is connected to B (via a single backslash) and C (via a single slash).

```
graph TD; A((A)) ---|>| B((B)); A ---|>| C((C)); A ---|>| D((D)); B ---|>| C; B ---|>| E((E)); C ---|>| A; C ---|>| B; D ---|>| A; D ---|>| B; E ---|>| C;
```

Code editor interface:

- File Edit Selection View Go Run Terminal Help
- EXPLORER
- BT20CSE069\_DVR
- dvr.py
- figure.txt
- input1.txt
- input2.txt
- output1.txt
- output2.txt
- Problem Statement.pdf
- Report.pdf

TERMINAL

```
PS C:\Users\HP\Desktop\BT20CSE069_DVR> python dvr.py input2.txt > output2.txt
PS C:\Users\HP\Desktop\BT20CSE069_DVR> [ ]
```

Bottom status bar:

- PROBLEMS
- OUTPUT
- DEBUG CONSOLE
- TERMINAL
- Ln 1, Col 1 Spaces: 4 UTF-16 LE CRLF Plain Text
- 32°C Haze
- Search
- File Explorer
- Task View
- PowerShell
- File
- Help
- 14:18 09-04-2023

The screenshot shows a Windows desktop environment with several open windows. In the center, there is a terminal window titled 'BT20CSE069\_DVR' containing the following command and its output:

```
PS C:\Users\HP\Desktop\BT20CSE069_DVR> python dvr.py input2.txt > output2.txt
```

The terminal output is as follows:

```
BT20CSE069_DVR
dvr.py
figure.txt
input2.txt
output2.txt
BT20CSE069_DVR.mp4
figure.txt
input1.txt
input2.txt
output1.txt
output2.txt
Problem Statement.pdf
Report.pdf

output2.txt
58
59
60 **** Iteration 1 ****
61
62 A => Router Object
63 Name: A
64 Neighbours: ['B', 'C', 'D']
65 Routing Table: {
66     A: 0          via: None
67     *B: 6         via: D
68     C: 2          via: C
69     D: 1          via: D
70     *E: 11        via: B
71 }
72
73 B => Router Object
74 Name: B
75 Neighbours: ['A', 'D', 'E']
76 Routing Table: {
77     *A: 6          via: D
78     B: 0          via: None
79     *C: 10         via: A
80     D: 5          via: D
81     E: 3          via: E
82 }
83
84 C => Router Object
85 Name: C
86 Neighbours: ['A']
87 Routing Table: {
88     A: 2          via: A
89 }

dvr.py
figure.txt
input2.txt
input2.txt
```

```
BT20CSE069_DVR
```

File Edit Selection View Go Run Terminal Help

EXPLORER

BT20CSE069\_DVR

- BT20CSE069\_DVR.mp4
- dvr.py
- figure.txt
- input1.txt
- input2.txt
- output1.txt
- output2.txt

Problem Statement.pdf

Report.pdf

output2.txt

```
231
232
233
234 **** Iteration 4 ****
235
236 A => Router Object
237 Name: A
238 Neighbours: ['B', 'C', 'D']
239 Routing Table: {
240     A: 0      via: None
241     B: 6      via: D
242     C: 2      via: C
243     D: 1      via: D
244     E: 9      via: D
245 }
246
247 B => Router Object
248 Name: B
249 Neighbours: ['A', 'D', 'E']
250 Routing Table: {
251     A: 6      via: D
252     B: 0      via: None
253     C: 8      via: D
254     D: 5      via: D
255     E: 3      via: E
256 }
257
258 C => Router Object
259 Name: C
260 Neighbours: ['A']
261 Routing Table: {
262     A: 2      via: None
263 }
```

dvr.py

figure.txt

input1.txt

input2.txt

output2.txt

```
1 5
2 A B C D E
3 A B 8
4 A C 2
5 A D 1
6 B D 5
7 B E 3
8 EOF
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

ps C:\Users\HP\Desktop\BT20CSE069\_DVR> python dvr.py input2.txt > output2.txt

ps C:\Users\HP\Desktop\BT20CSE069\_DVR>

OUTLINE

TIMELINE

Live Share KyLang Python

32°C Haze

powershell +

14:20 09-04-2023