University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 7: **Implementation of Link State Routing Algorithm**

Submitted By:

Joty Saha (Roll-51)
Ahona Rahman (Roll-59)

Submitted To:

Dr. Md. Abdur Razzaque
Md. Mahmudur Rahman
Md. Ashraful Islam
Md. Fahim Arefin

Submitted On: February 14, 2024

Contents

1 Introduction

TIn computer networking, efficient routing of data packets is crucial for the optimal performance of communication systems. The Link State Routing Algorithm stands as one of the fundamental methods used for this purpose. Unlike distance vector algorithms, which rely on iterative exchanges of routing tables, the Link State Algorithm takes a different approach. It involves each router in the network maintaining a detailed map of the entire network topology, which it then uses to compute the shortest path to each destination. This proactive approach to routing ensures faster convergence and better scalability in large networks.

1.1 Objectives

- **Understand Link State Algorithm Principles:**
  - Grasp the theoretical foundations of the Link State Routing Algorithm.
  - Comprehend the exchange of link state information and database construction.
- **Implement Network Topology Simulation:**
  - Gain hands-on experience in simulating network topologies.
  - Design a program to mimic router behavior and information exchange.
- **Demonstrate Link State Algorithm Functionality:**
  - Showcase the algorithm's ability to compute optimal paths.
  - Illustrate router behavior in constructing routing tables and forwarding decisions.
- **Evaluate Algorithm Performance:**
  - Analyze the Link State Algorithm's performance in terms of convergence time.
  - Assess scalability and robustness against network failures through experimentation.

2 Theory

The Link State Routing Algorithm is a key method used in computer networks for determining the shortest path between nodes. Unlike distance vector algorithms, which rely on iterative updates of routing tables, the Link State Algorithm operates by having each router maintain a map of the entire network topology. This map, known as the Link State Database (LSDB), contains information about the network's nodes and the links between them.

-

- Link State Advertisement (LSA)

  In the Link State Algorithm, routers periodically exchange Link State Advertisements (LSAs) to inform each other about changes in the network topology. Each LSA contains information about a router's neighboring routers and the cost to reach them. When a router receives an LSA, it updates its LSDB accordingly.

- Shortest Path Calculation

  Once routers have built their LSDBs, they use algorithms like Dijkstra's algorithm to calculate the shortest path to each destination node. Dijkstra's algorithm iteratively explores the network graph, updating the shortest path estimates until it finds the shortest path to every node.

- Forwarding Decision

  After calculating the shortest paths, each router constructs its routing table, which determines how packets are forwarded towards their destinations. When a router receives a packet, it consults its routing table to determine the next hop along the shortest path to the packet's destination.

- Experiment Implementation

  To implement the Link State Routing Algorithm experimentally, participants will simulate a network environment where routers exchange LSAs and compute shortest paths. They will develop a program to mimic router behavior, including LSDB construction, LSA exchange, shortest path calculation, and packet forwarding. Through this simulation, participants will observe the algorithm's functionality and analyze its performance in various network scenarios.

3 Methodology

3.1 Design and Implementation of Router Simulation Program:

- Design a program that simulates a network of routers, representing each router as a graph node and each link as an edge with a given cost.
- Implement data structures to represent routers, edges, and the network topology.
- Include functionalities for each router to maintain a mapping of ports and router names.

3.2 Implementation of Link State Algorithm:

3.2.1 Creation of a network topology:

- Read the list of edges for each router from a configurable file.
- Create Link State Packet (LSP) containing ID, Time-To-Live (TTL), cost of immediate edges, and send to all neighbors.
- Implement flooding mechanism: neighbors broadcast LSPs to their neighbors recursively.
- Upon receiving a new message, each node runs Dijkstra's algorithm to calculate the current shortest path to all other nodes.

3.2.2 Maintenance and updating of the network topology:

- Implement functionality to randomly change the cost of edges after a certain time interval.
- Update LSPs accordingly and propagate changes to neighbors through flooding.

3.3 Output All Pair Shortest Paths:

- Develop a function to output the shortest paths for each node based on the computed routing tables.

3.4 Performance Analysis:

- Measure time complexity: Analyze the time taken for running Dijkstra's algorithm and updating the network topology.
- Count the number of messages sent during LSP flooding and topology updates.
- Monitor memory usage throughout the simulation.

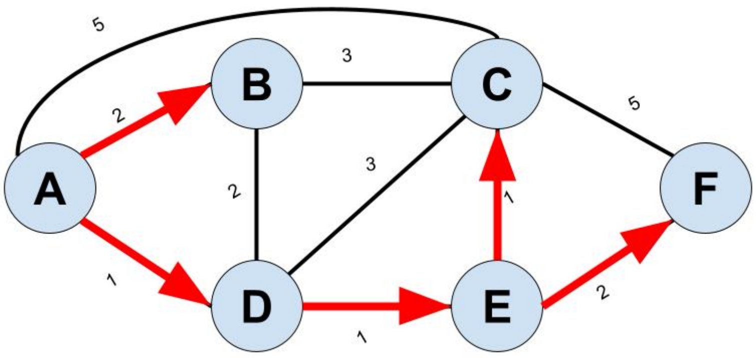
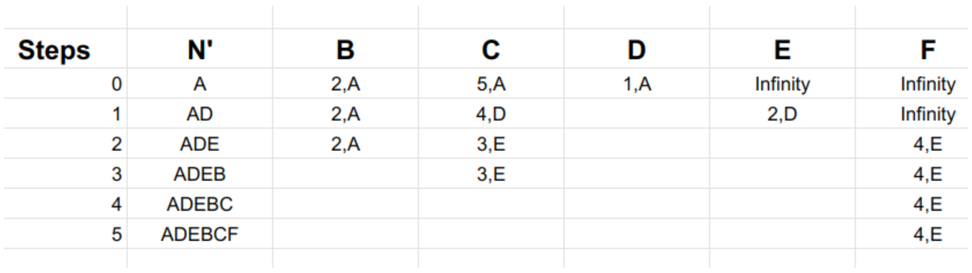
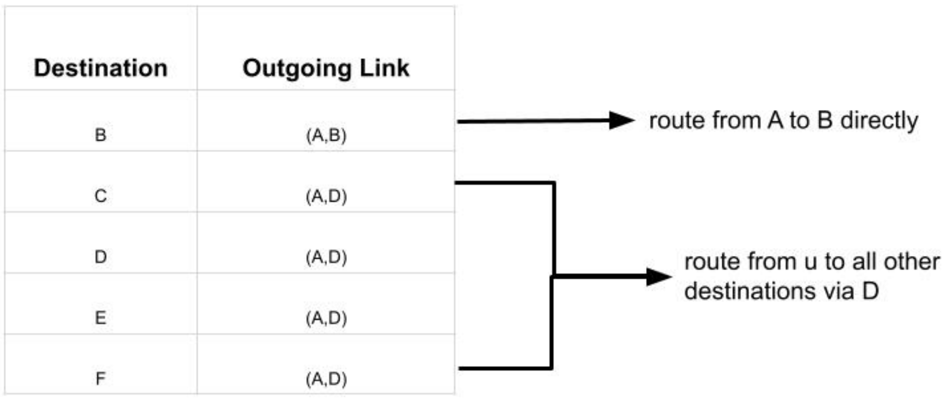


Figure 1: Resulting least-cost-path tree from A



Steps	N'	B	C	D	E	F
0	A	2,A	5,A	1,A	Infinity	Infinity
1	AD	2,A	4,D		2,D	Infinity
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					4,E

Figure 2: Dijkstra's algorithm table(for source A)



Destination	Outgoing Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

route from A to B directly
route from u to all other destinations via D

Figure 3: Rresulting forwarding table in A

3.5 Testing and Evaluation:

- Configure the initial network topology from a file to allow flexibility in testing.
- Set up a timer to randomly update an edge every 30 seconds to simulate network changes.
- Evaluate the functionality of the Link State Algorithm by analyzing the calculated shortest paths under various network scenarios.

3.6 Documentation:

- Document the design, implementation, and functionality of the program.
- Record experimental results including performance metrics and test outcomes.
- Provide insights and analysis based on the collected data.

By following these steps, the methodology ensures a systematic approach to implementing, testing, and evaluating the Link State Algorithm in the simulated network environment.

4 Experimental result

4.1 Link State Routing Graph

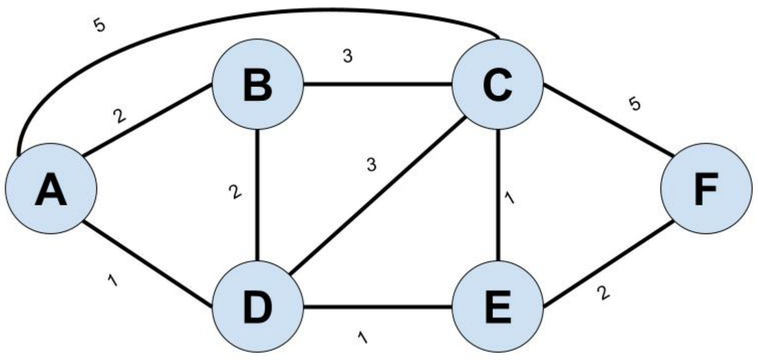
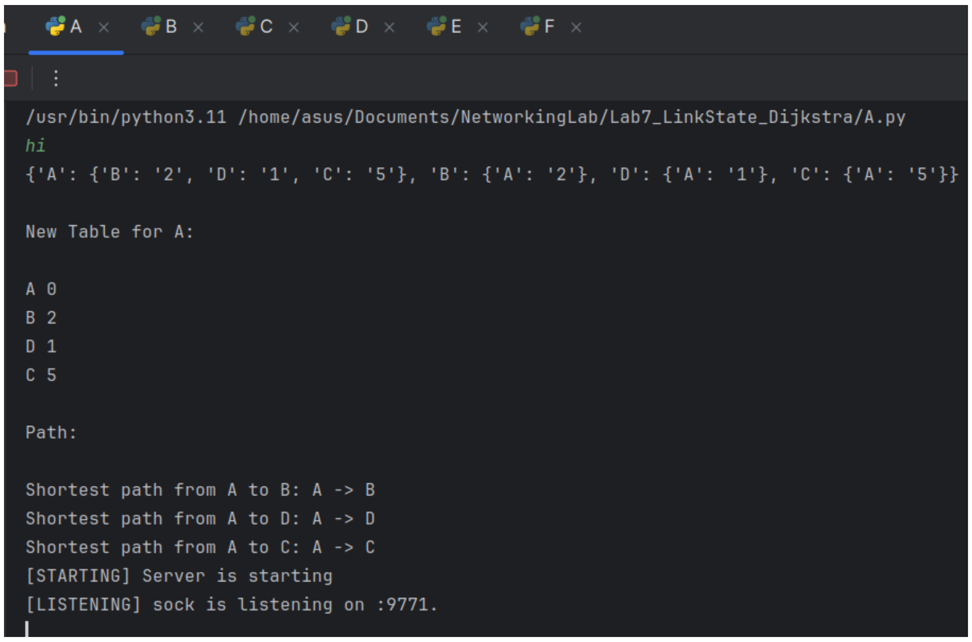


Figure 4: Link State Graph

4.2 Snapshots

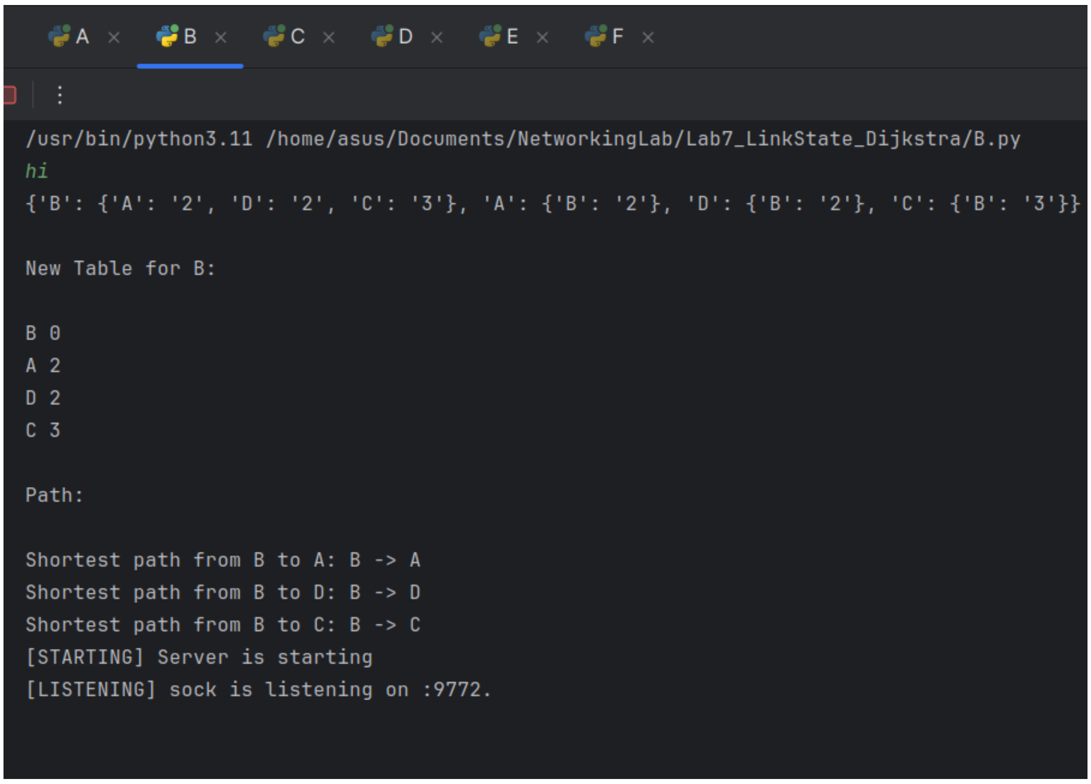
- **For Router A:** This the experimental result(path cost) of A router for the initial graph



```
/usr/bin/python3.11 /home/asus/Documents/NetworkingLab/Lab7_LinkState_Dijkstra/A.py
hi
{'A': {'B': '2', 'D': '1', 'C': '5'}, 'B': {'A': '2'}, 'D': {'A': '1'}, 'C': {'A': '5'}}

New Table for A:

A 0
B 2
D 1
C 5

Path:

Shortest path from A to B: A -> B
Shortest path from A to D: A -> D
Shortest path from A to C: A -> C
[STARTING] Server is starting
[LISTENING] sock is listening on :9771.
```

Figure 5: A router's Initial path cost

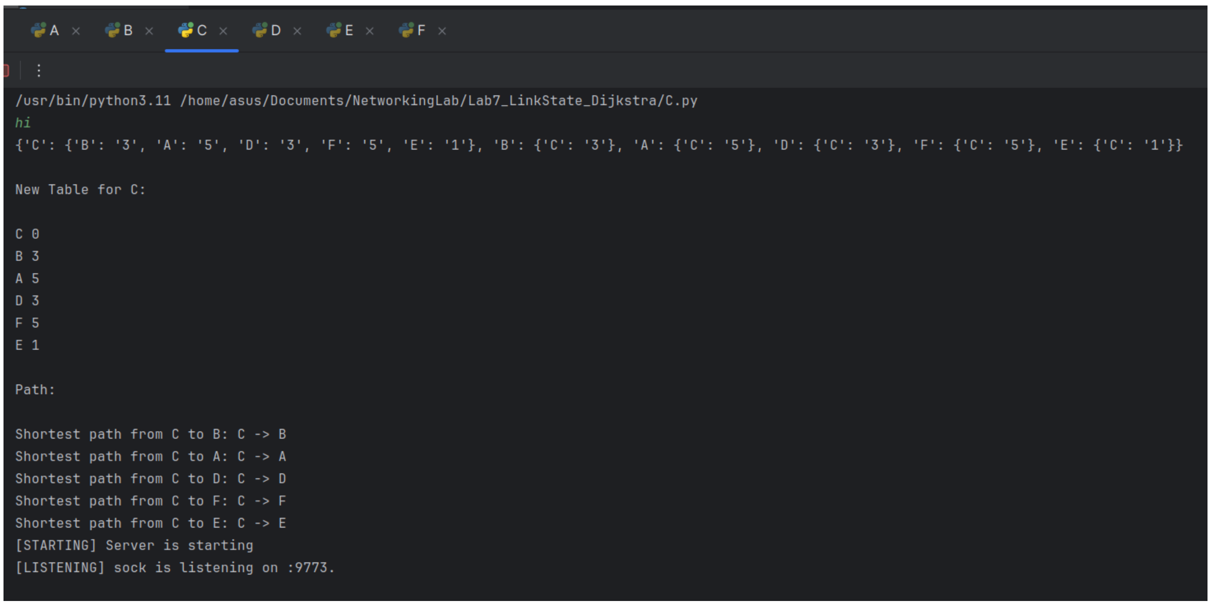
- **For Router B:** This the experimental result(path cost) of B router for the initial graph



```
/usr/bin/python3.11 /home/asus/Documents/NetworkingLab/Lab7_LinkState_Dijkstra/B.py
hi
{'B': {'A': '2', 'D': '2', 'C': '3'}, 'A': {'B': '2'}, 'D': {'B': '2'}, 'C': {'B': '3'}}

New Table for B:

B 0
A 2
D 2
C 3

Path:

Shortest path from B to A: B -> A
Shortest path from B to D: B -> D
Shortest path from B to C: B -> C
[STARTING] Server is starting
[LISTENING] sock is listening on :9772.
```

Figure 6: B router's Initial path cost

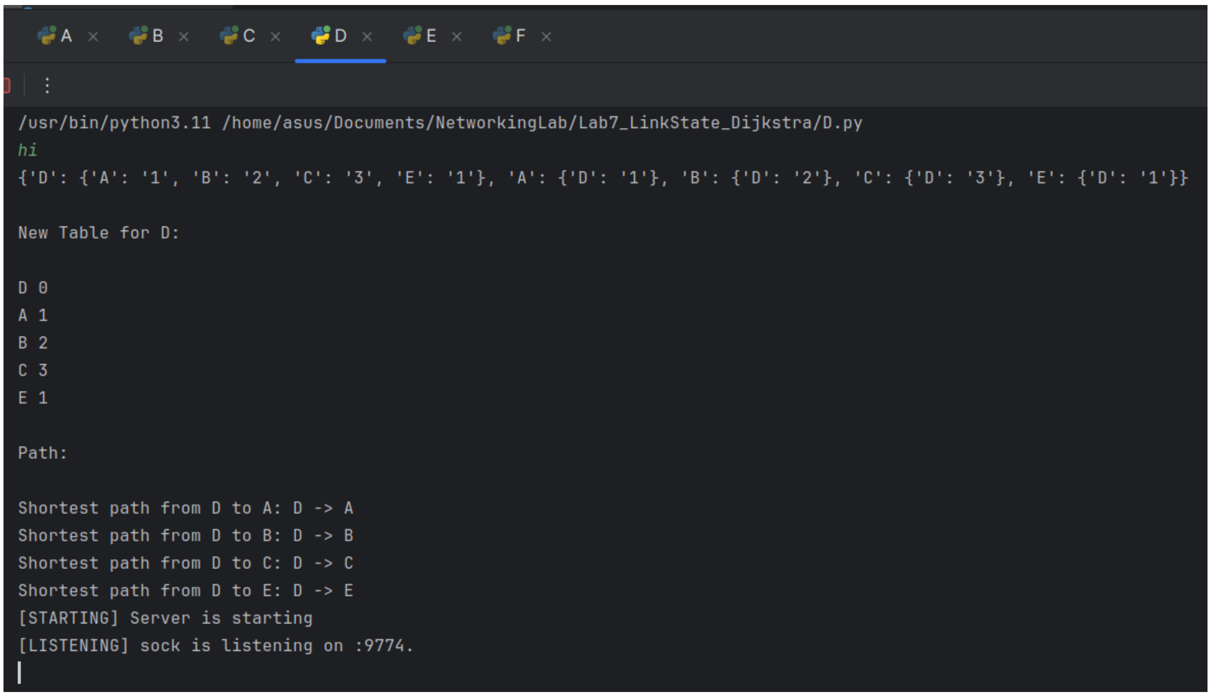
- **For Router C:** This the experimental result(path cost) of C router for the initial graph



```
/usr/bin/python3.11 /home/asus/Documents/NetworkingLab/Lab7_LinkState_Dijkstra/C.py
hi
{'C': {'B': '3', 'A': '5', 'D': '3', 'F': '5', 'E': '1'}, 'B': {'C': '3'}, 'A': {'C': '5'}, 'D': {'C': '3'}, 'F': {'C': '5'}, 'E': {'C': '1'}}

New Table for C:

C 0
B 3
A 5
D 3
F 5
E 1

Path:

Shortest path from C to B: C -> B
Shortest path from C to A: C -> A
Shortest path from C to D: C -> D
Shortest path from C to F: C -> F
Shortest path from C to E: C -> E
[STARTING] Server is starting
[LISTENING] sock is listening on :9773.
```

Figure 7: C router's Initial path cost

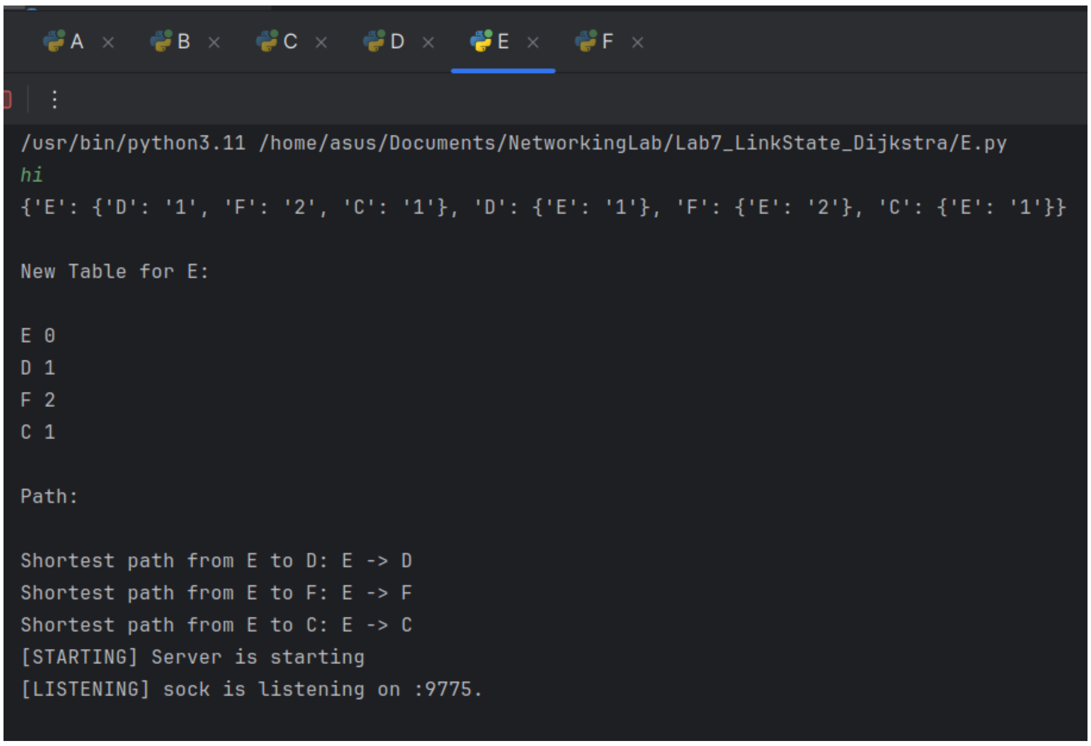
- **For Router D:** This the experimental result(path cost) of D router for the initial graph



```
/usr/bin/python3.11 /home/asus/Documents/NetworkingLab/Lab7_LinkState_Dijkstra/D.py
hi
{'D': {'A': '1', 'B': '2', 'C': '3', 'E': '1'}, 'A': {'D': '1'}, 'B': {'D': '2'}, 'C': {'D': '3'}, 'E': {'D': '1'}}

New Table for D:

D 0
A 1
B 2
C 3
E 1

Path:

Shortest path from D to A: D -> A
Shortest path from D to B: D -> B
Shortest path from D to C: D -> C
Shortest path from D to E: D -> E
[STARTING] Server is starting
[LISTENING] sock is listening on :9774.
```

Figure 8: D router's Initial path cost

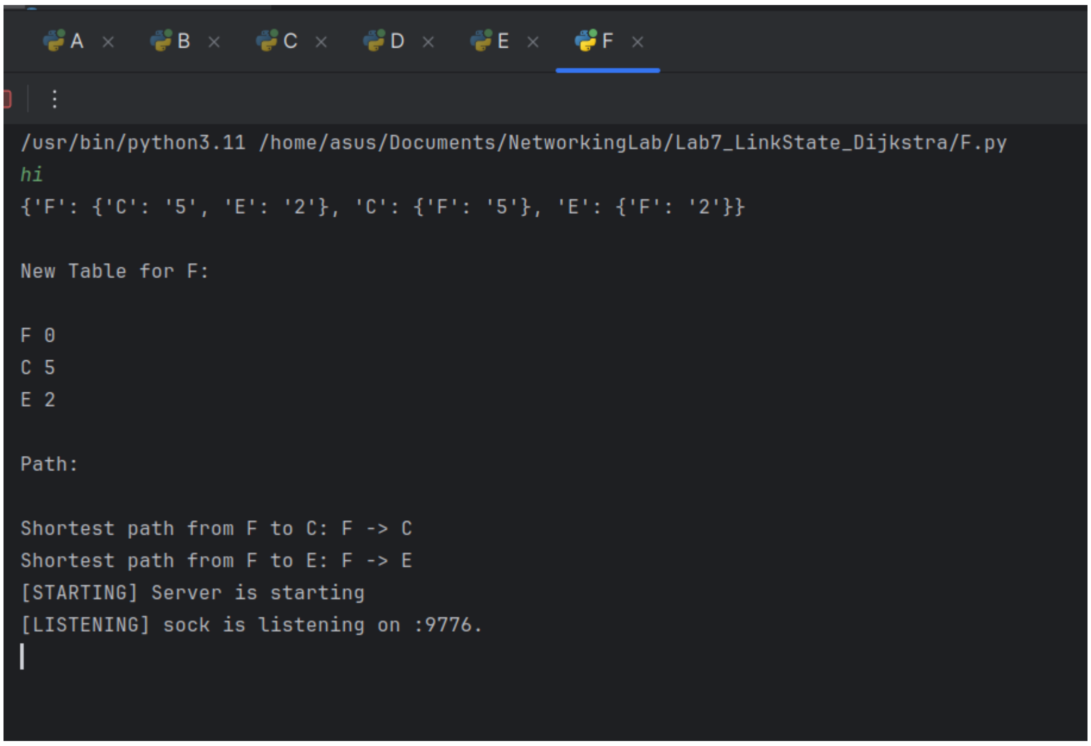
- **For Router E:** This the experimental result(path cost) of E router for the initial graph



```
/usr/bin/python3.11 /home/asus/Documents/NetworkingLab/Lab7_LinkState_Dijkstra/E.py
hi
{'E': {'D': '1', 'F': '2', 'C': '1'}, 'D': {'E': '1'}, 'F': {'E': '2'}, 'C': {'E': '1'}}

New Table for E:

E 0
D 1
F 2
C 1

Path:

Shortest path from E to D: E -> D
Shortest path from E to F: E -> F
Shortest path from E to C: E -> C
[STARTING] Server is starting
[LISTENING] sock is listening on :9775.
```

Figure 9: E router's Initial path cost

- **For Router F:** This the experimental result(path cost) of F router for the initial graph

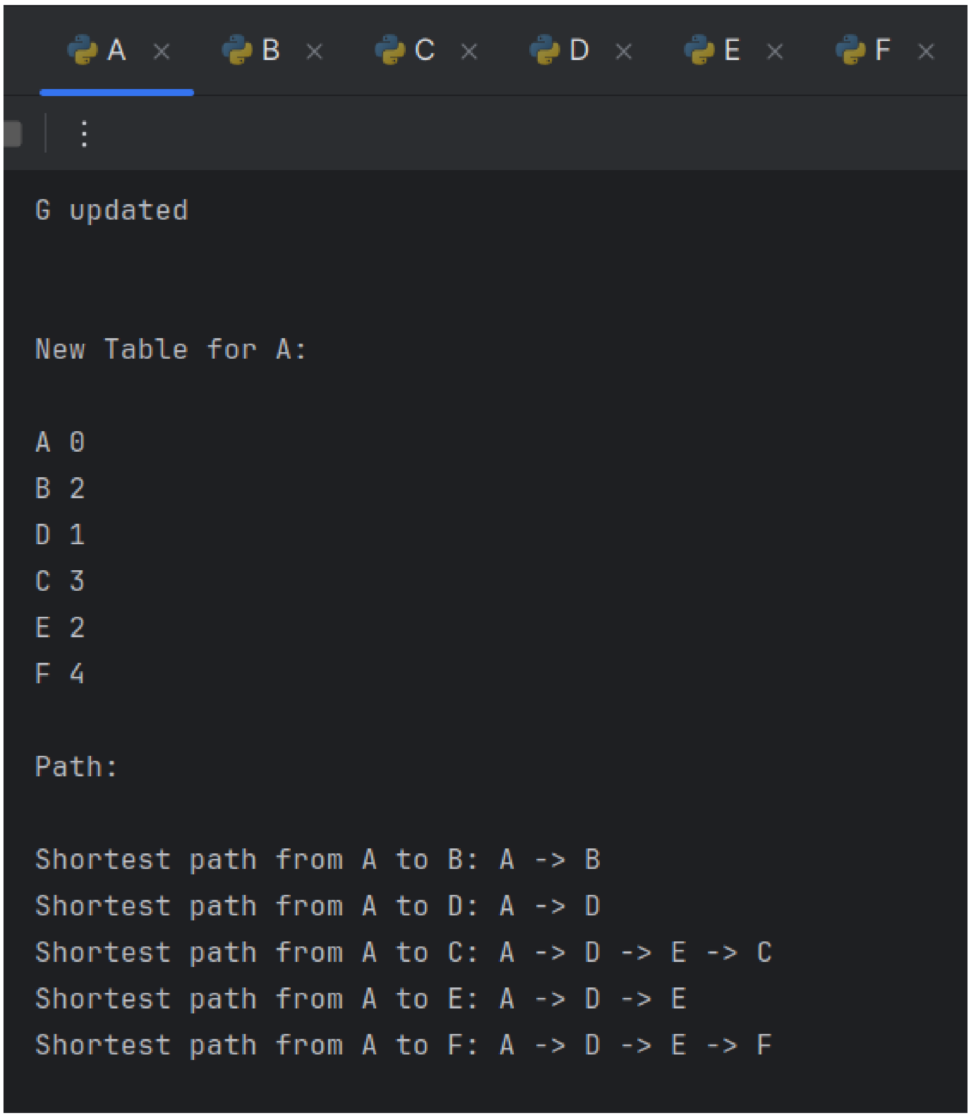


```
/usr/bin/python3.11 /home/asus/Documents/NetworkingLab/Lab7_LinkState_Dijkstra/F.py
hi
{'F': {'C': '5', 'E': '2'}, 'C': {'F': '5'}, 'E': {'F': '2'}}

New Table for F:

F 0
C 5
E 2

Path:

Shortest path from F to C: F -> C
Shortest path from F to E: F -> E
[STARTING] Server is starting
[LISTENING] sock is listening on :9776.
```

Figure 10: F router's Initial path cost

After 30 seconds, the path costs are randomly updated for each router or the new edges' values are added to the graph.

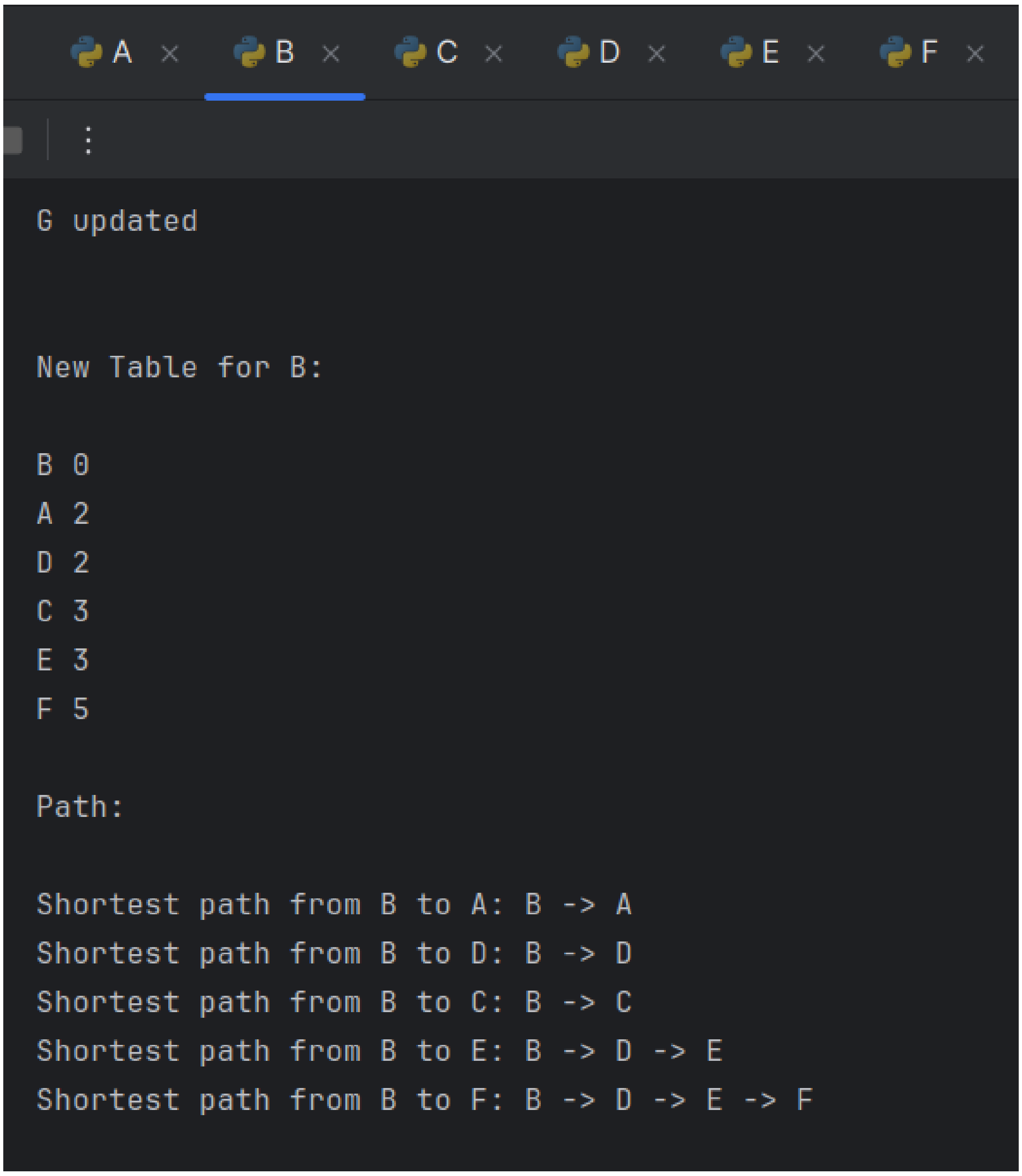
- **For Router A:** This the experimental result(path cost) of A router after 30 seconds



```
G updated


New Table for A:

A 0
B 2
D 1
C 3
E 2
F 4

Path:

Shortest path from A to B: A -> B
Shortest path from A to D: A -> D
Shortest path from A to C: A -> D -> E -> C
Shortest path from A to E: A -> D -> E
Shortest path from A to F: A -> D -> E -> F
```

Figure 11: A router's path cost after 30 sec

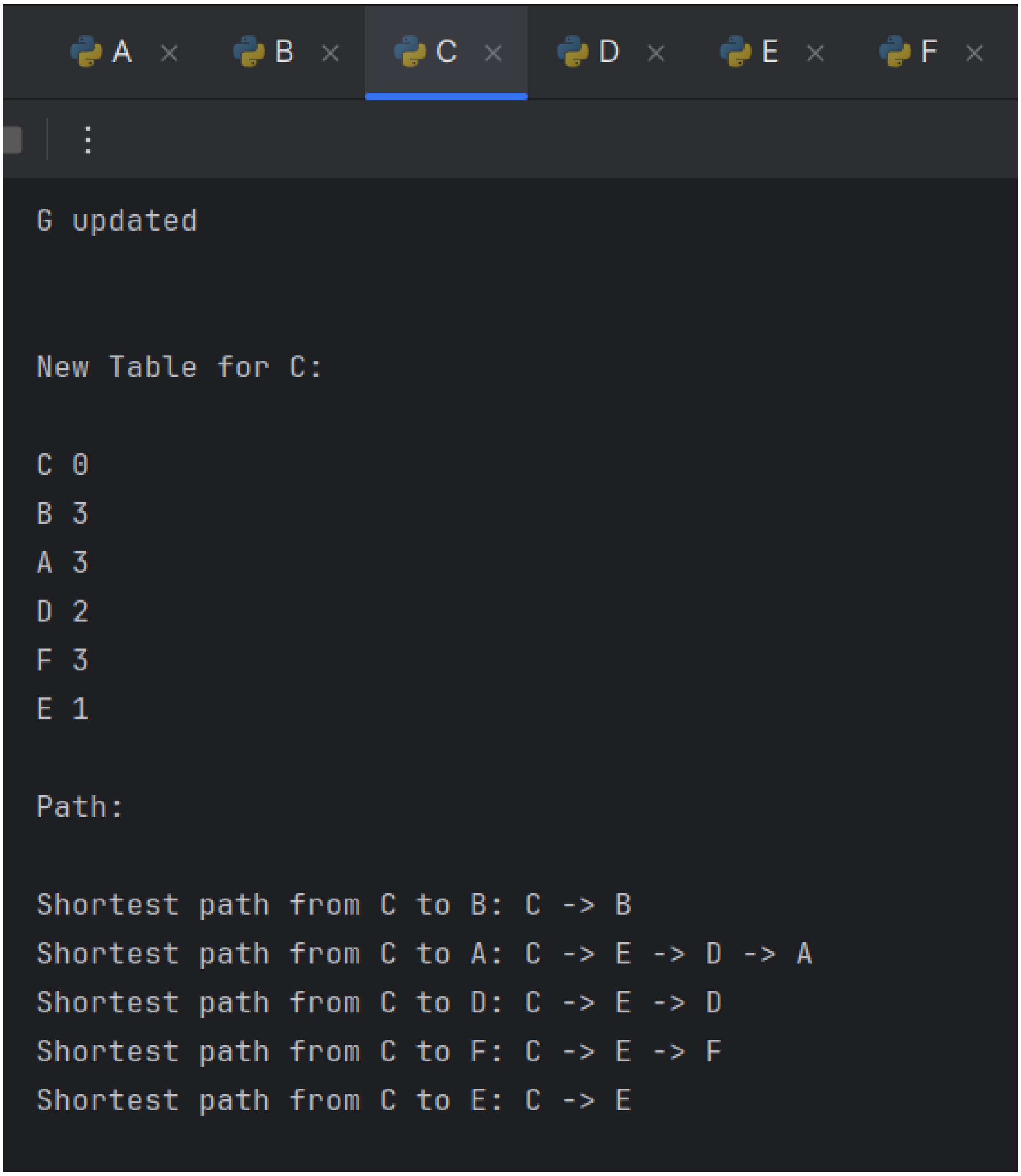
- **For Router B:** This the experimental result(path cost) of B router after 30 seconds



```
G updated


New Table for B:

B 0
A 2
D 2
C 3
E 3
F 5

Path:

Shortest path from B to A: B -> A
Shortest path from B to D: B -> D
Shortest path from B to C: B -> C
Shortest path from B to E: B -> D -> E
Shortest path from B to F: B -> D -> E -> F
```

Figure 12: B router's path cost after 30 sec

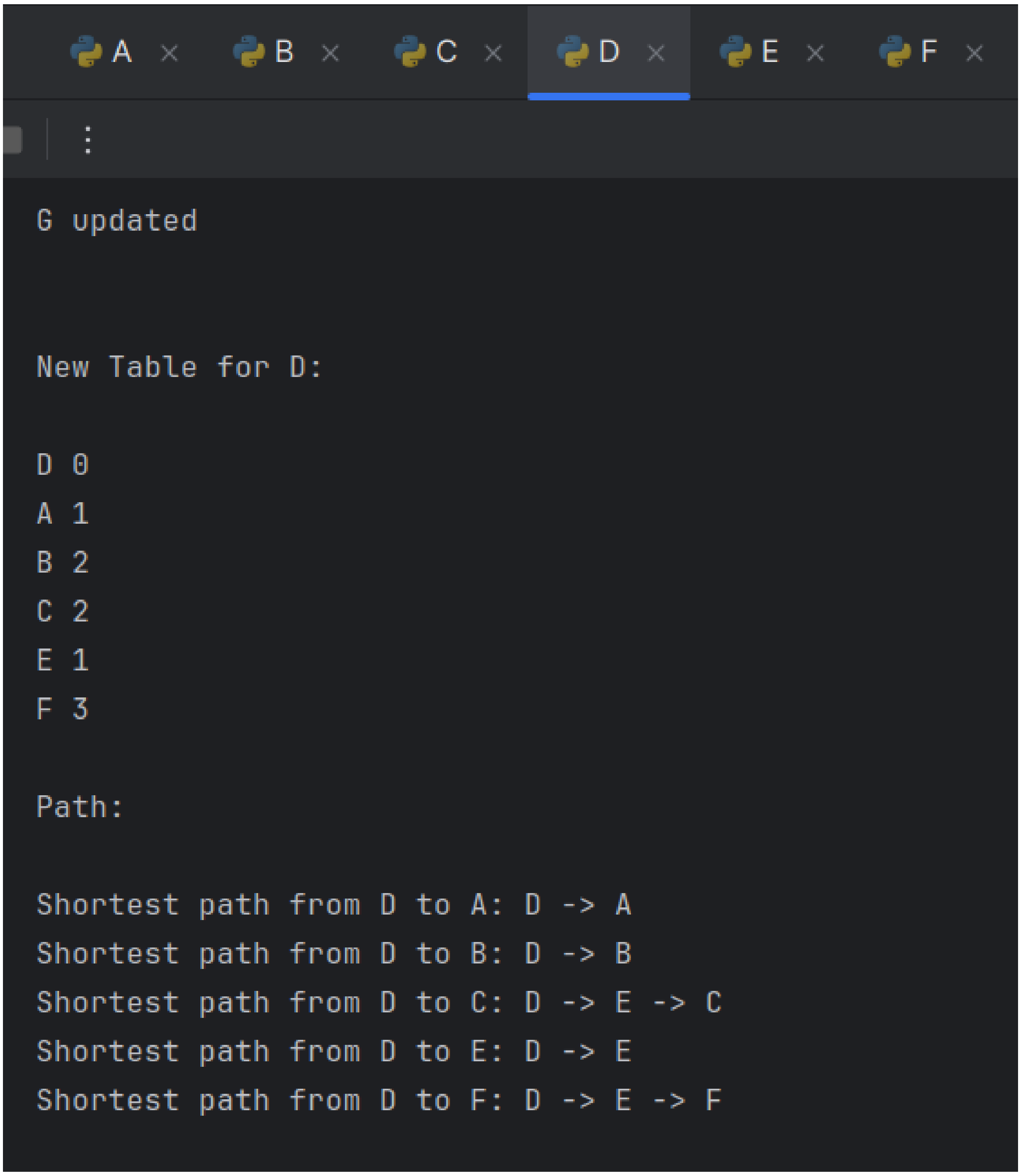
- **For Router C:** This the experimental result(path cost) of C router after 30 seconds



```
G updated


New Table for C:

C 0
B 3
A 3
D 2
F 3
E 1

Path:

Shortest path from C to B: C -> B
Shortest path from C to A: C -> E -> D -> A
Shortest path from C to D: C -> E -> D
Shortest path from C to F: C -> E -> F
Shortest path from C to E: C -> E
```

Figure 13: C router's path cost after 30 sec

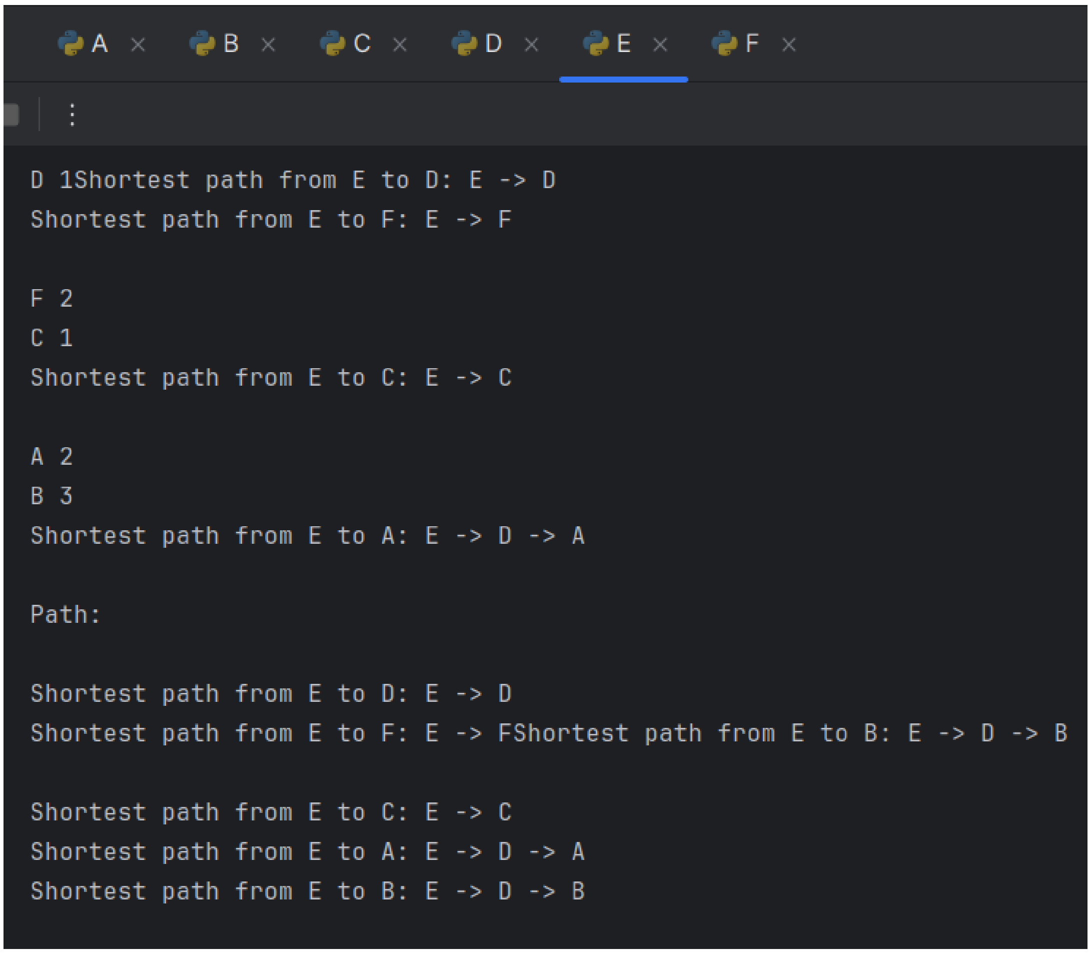
- **For Router D:** This the experimental result(path cost) of D router after 30 seconds



```
G updated


New Table for D:

D 0
A 1
B 2
C 2
E 1
F 3

Path:

Shortest path from D to A: D -> A
Shortest path from D to B: D -> B
Shortest path from D to C: D -> E -> C
Shortest path from D to E: D -> E
Shortest path from D to F: D -> E -> F
```

Figure 14: D router’s path cost after 30 sec

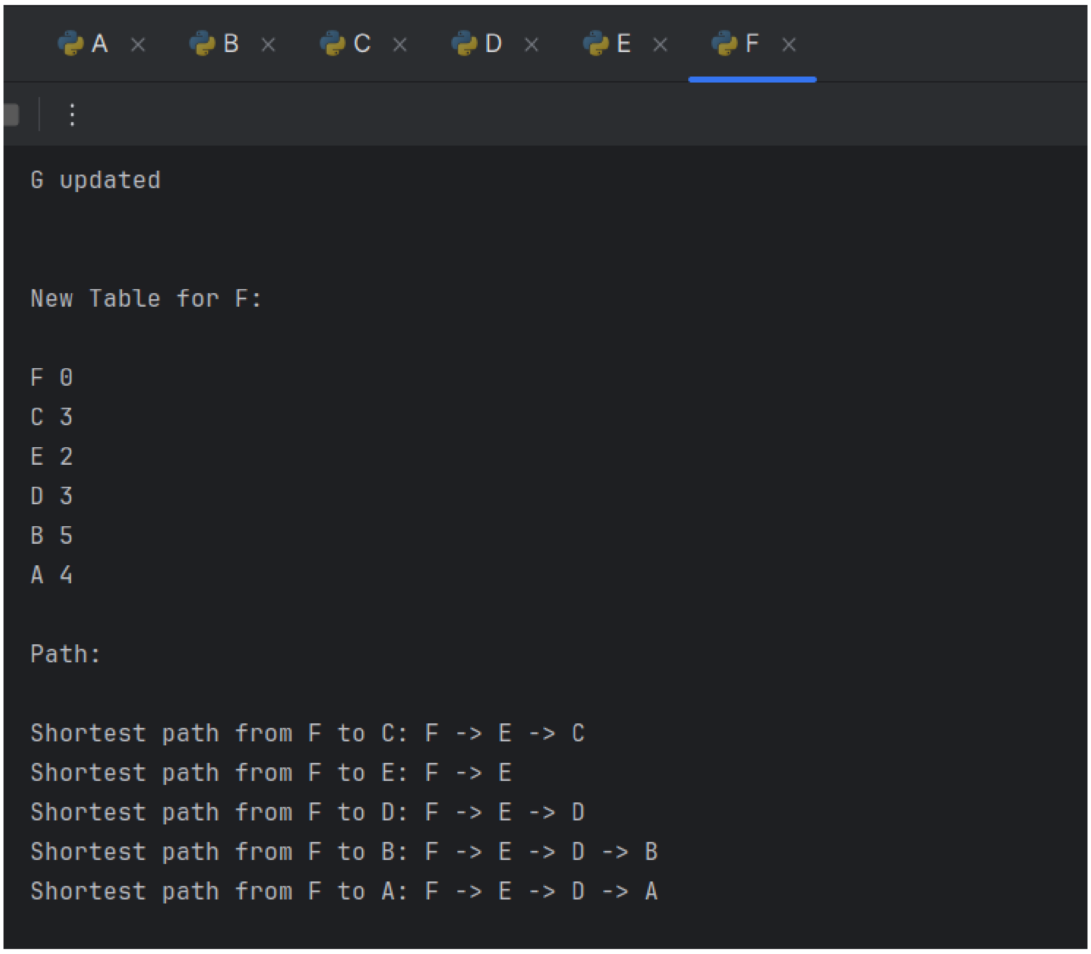
- **For Router E:** This the experimental result(path cost) of E router after 30 seconds



```
D 1Shortest path from E to D: E -> D
Shortest path from E to F: E -> F

F 2
C 1
Shortest path from E to C: E -> C

A 2
B 3
Shortest path from E to A: E -> D -> A

Path:

Shortest path from E to D: E -> D
Shortest path from E to F: E -> FShortest path from E to B: E -> D -> B

Shortest path from E to C: E -> C
Shortest path from E to A: E -> D -> A
Shortest path from E to B: E -> D -> B
```

Figure 15: E router's path cost after 30 sec

- **For Router F:** This the experimental result(path cost) of F router after 30 seconds



```
G updated


New Table for F:

F 0
C 3
E 2
D 3
B 5
A 4

Path:

Shortest path from F to C: F -> E -> C
Shortest path from F to E: F -> E
Shortest path from F to D: F -> E -> D
Shortest path from F to B: F -> E -> D -> B
Shortest path from F to A: F -> E -> D -> A
```

Figure 16: F router's path cost after 30 sec

4.3 Issues and Challenges

- Graph Representation One initial challenge was to decide on an efficient way to represent the network topology as a graph in the program. While a simple adjacency matrix could have been used, it would have resulted in high memory consumption for larger networks. To address this, an adjacency list representation was chosen, which efficiently stores only the edges and their weights for each vertex.
- LSP Distribution Implementing the mechanism for distributing Link State Packets (LSPs) posed another challenge. Initially, a straightforward approach was considered where each router sends LSPs directly to its neighbors. However, this approach resulted in inefficient flooding and redundant transmissions. To optimize LSP distribution, a more refined flooding algorithm was implemented, where routers keep track of previously received LSPs and only forward new or updated LSPs to their neighbors. This optimization significantly reduced unnecessary message propagation and improved the efficiency of the algorithm.
- Topology Updates Updating the network topology dynamically presented a challenge, particularly in maintaining consistency across routers. One approach considered was to periodically broadcast updated edge costs to all routers. However, this approach introduced the risk of inconsistency if multiple routers attempted to update the topology simultaneously. To address this, a synchronized update mechanism was implemented, where routers coordinate with each other to ensure consistency before applying topology changes. This approach minimized the risk of inconsistency and maintained network stability during topology updates.
- Error Handling and Robustness Ensuring the program's robustness against errors and unexpected conditions was crucial. Comprehensive error handling mechanisms were implemented to detect and handle various exceptional scenarios, such as invalid input data, network partitioning, or router failures. Additionally, extensive testing and validation were conducted to verify the program's behavior under different network conditions and edge cases.

4.4 Algorithm's performance

4.4.1 Time Complexity

1. **Link State Routing Algorithm** The time complexity of the Link State Routing Algorithm mainly depends on the underlying graph traversal and shortest path calculation algorithms used.

    - The initialization of the algorithm involves initializing data structures such as priority queues or arrays, which typically takes $O(V)$ time, where V is the number of vertices.
    - The main iteration of the algorithm involves updating shortest path distances and predecessors, which typically takes $O(V^2)$ time for each iteration. Since there can be at most V iterations in the worst case, the overall time complexity of the algorithm is $O(V^3)$.
    - However, if optimized shortest path algorithms like Dijkstra's algorithm or Floyd-Warshall algorithm are used, the time complexity can be reduced accordingly. Dijkstra's algorithm typically has a time complexity of $O((V + E) \log V)$, while Floyd-Warshall algorithm has a time complexity of $O(V^3)$.

2. **LSP Distribution and Topology Updates** The time complexity of simulating LSP distribution and topology updates depends on the mechanisms employed for message propagation and topology maintenance.

    - If a naive flooding mechanism is used, where each router forwards messages to all neighbors, the time complexity can be high, possibly $O(V^2)$ for each message propagation.
    - Optimized mechanisms, such as keeping track of previously received LSPs to avoid redundant transmissions, can reduce the time complexity significantly.

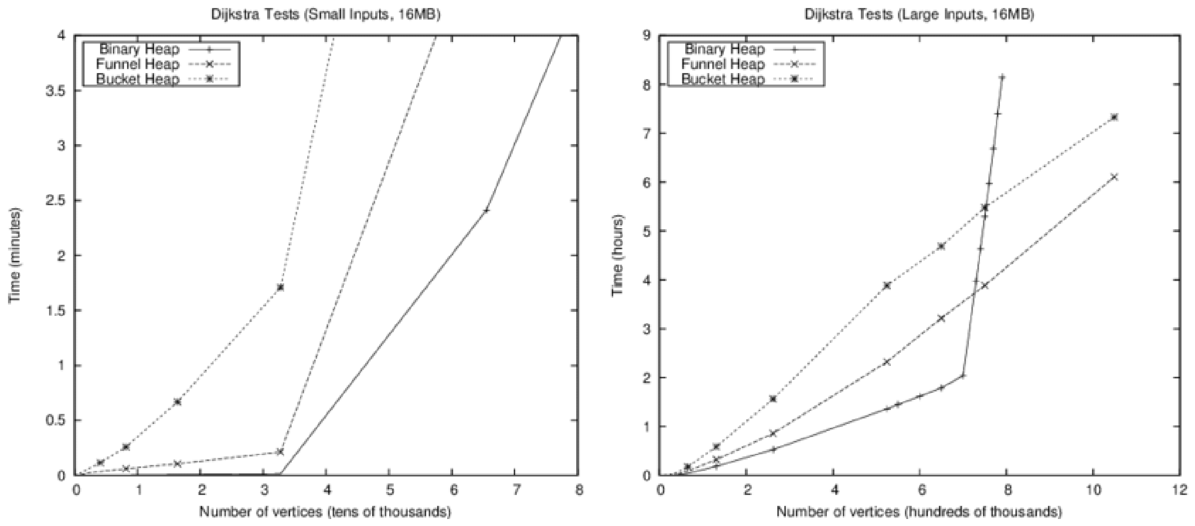


Figure 17: time complexity vs number of vertices

4.4.2 Memory Usage

1. **Graph Representation** The memory usage for representing the network topology as a graph depends on the chosen data structure.

   - Using an adjacency matrix requires $O(V^2)$ memory, where V is the number of vertices, which can be impractical for large networks.
   - Using an adjacency list representation is more memory-efficient, typically requiring $O(V + E)$ memory, where E is the number of edges.

2. **LSP Distribution and Topology Updates** The memory usage for simulating LSP distribution and topology updates depends on the data structures used for message queues, routing tables, and maintaining network state.

   - Optimized data structures and memory management techniques can help reduce memory usage and improve efficiency.

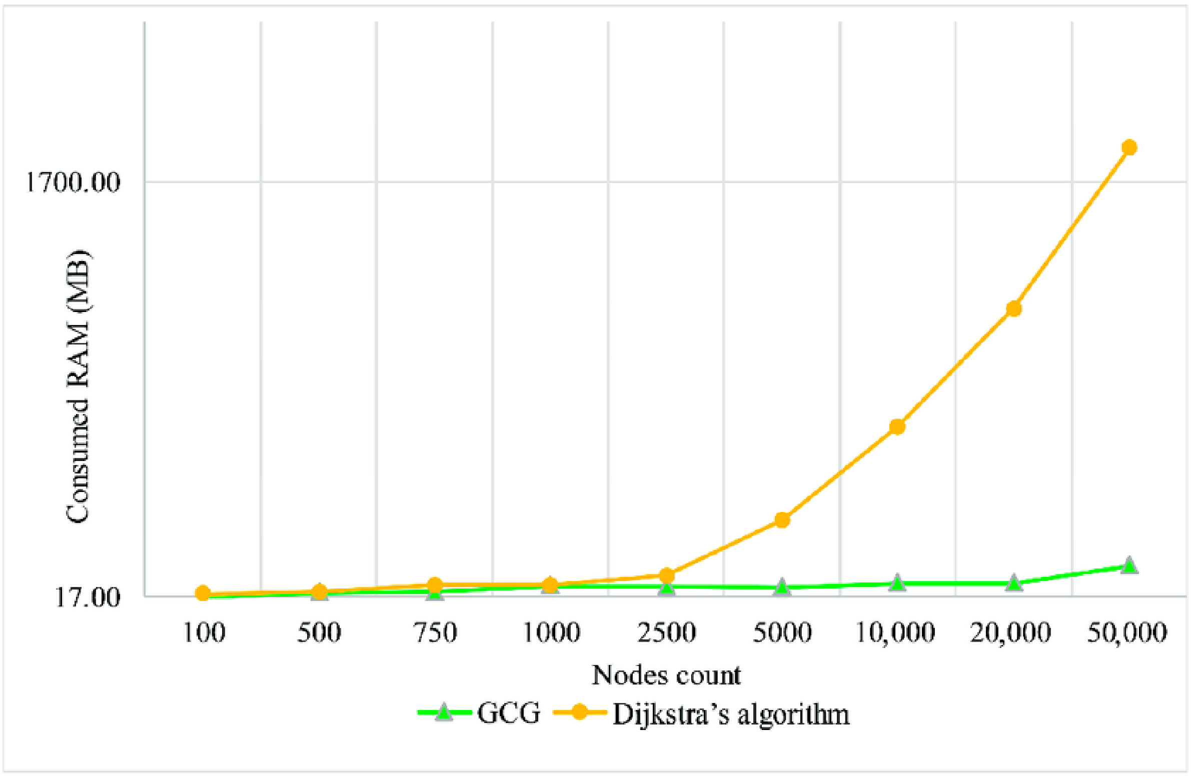


Figure 18: memory vs number of vertices

5 Experience

- Initial Design and Setup:
  - We began by conceptualizing and designing a program to simulate a network of routers.
  - Meticulously crafted data structures were created to accurately represent routers, edges, and the network topology.
  - Consideration was given to how routers would communicate and exchange information within the simulated network.
- Algorithm Implementation:
  - The implementation of the Link State Algorithm was a multi-faceted endeavor.
  - Steps were taken to create a network topology, including reading edge lists from a configurable file.
  - Link State Packets (LSPs) were generated, containing essential information such as ID, Time-To-Live (TTL), and the cost of immediate edges.
  - A flooding mechanism was implemented to ensure LSPs were disseminated to all routers in the network.
  - Dijkstra's algorithm was integrated to calculate the shortest paths based on the received LSPs.
- Testing and Evaluation:
  - Extensive testing was conducted to verify the correctness and performance of the implemented algorithm.
  - Initial network topologies were configured from files to provide a baseline for testing.
  - Dynamic changes were introduced to edge costs at regular intervals to simulate network dynamics.
  - The algorithm's functionality was evaluated under various network scenarios to gauge its adaptability and robustness.
- Performance Analysis:
  - Performance analysis played a crucial role in understanding the algorithm's efficiency and scalability.
  - Time complexity was measured by analyzing the execution time of Dijkstra's algorithm and network topology updates.
  - The number of messages sent during LSP flooding and topology updates was counted to assess communication overhead.
  - Memory usage was monitored throughout the simulation to identify potential resource constraints.
- Documentation and Analysis:
  - Thorough documentation of the experiment was essential for capturing design decisions, implementation details, and experimental results.
  - Insights gained from analyzing the collected data were documented to provide a comprehensive understanding of the experiment's outcomes.
  - Analysis of performance metrics and observations facilitated the extraction of meaningful conclusions regarding the algorithm's strengths and limitations.

References

[1] https://www.javatpoint.com/link-state-routing-algorithm

[2] https://www.geeksforgeeks.org/unicast-routing-link-state-routing/

[3] https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

[4] https://www.youtube.com/watch?v=kW6zV-040SY

[5] https://www.youtube.com/watch?v=ZItmPVX_WIw