University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 7:

Implementation of Link State Algorithm

Submitted By:

1. Syed Mumtahin Mahmud, Roll: 50
2. Nazira Jesmin Lina, Roll: 55

Submitted On :

March 23, 2023

Submitted To :

Dr. Md. Abdur Razzaque

Md Mahmudur Rahman

Md. Ashraful Islam

Md. Fahim Arefin

1 Introduction

The Link State Algorithm is a routing algorithm used in computer networking to determine the shortest path between two nodes in a network. It is also known as the shortest path first algorithm. The algorithm works by having each router in the network maintain a complete map of the network topology, including information about the state of each link, such as its bandwidth and latency.

When a router receives information about the state of a link, it updates its map of the network and calculates the shortest path to all other nodes using Dijkstra's algorithm. Each router then broadcasts its updated map to all other routers in the network, allowing them to update their own maps and calculate new shortest paths.

The Link State Algorithm is commonly used in large-scale networks such as the Internet, as it can scale to handle networks with thousands or even millions of nodes. It is more efficient than other routing algorithms such as the Distance Vector Algorithm, as it does not require routers to broadcast their entire routing table to every other router in the network. Instead, only updates to the network topology are broadcast, reducing network traffic and improving performance.In this lab, you will design and implement a program that simulates the Link State Algorithm, demonstrating its functionality and gaining practical experience in using the algorithm for routing purposes.

1.1 Objectives

- To understand the Link State Algorithm and its role in computer networking.
- To learn how routers exchange information to build a network topology map and calculate the shortest path to a destination.
- To develop an understanding of the Link State Algorithm and its applications in computer networks by implementing the algorithm.

2 Theory

the Link State Algorithm is based on the concept of shortest path routing. In a network, there can be multiple paths to reach a destination, but the shortest path is the one that has the minimum number of hops or the lowest cost in terms of metrics such as bandwidth, delay, or congestion. The Link State Algorithm determines the shortest path by constructing a complete map of the network topology and calculating the shortest path to all other nodes using Dijkstra's algorithm.

The Link State Algorithm works by having each router in the network maintain a database of information about the state of each link, including its bandwidth, delay, and other metrics. This information is used to construct a map of the network topology, which is stored in the router's database. Each router then runs Dijkstra's algorithm to calculate the shortest path to all other nodes in the network.

When a link state change occurs, such as a link failure or a new link being added, the affected router updates its database and broadcasts the new information to all other routers in the network. Each router then updates its own database and recalculates the shortest path to all other nodes using Dijkstra's algorithm.

The Link State Algorithm has several advantages over other routing algorithms, such as the Distance Vector Algorithm. It can handle large-scale networks with thousands or even millions of nodes, and it is more efficient as it does not require routers to broadcast their entire routing table to every other router in the network. Instead, only updates to the network topology are broadcast, reducing network traffic and improving performance.

3 Methodology

1. **Design a network topology:**Determine the number of routers, their interconnections, and the link costs for each link.

2. **Implement a program that simulates the network:** Use a programming language to create a network simulator that includes the necessary components such as routers, links, packets, and routing algorithms.

3. **Implement the Link State Algorithm:** Add the necessary code to your program to implement the Link State Algorithm. This will include creating Link State Packets (LSPs) for each router and broadcasting them to all neighbors using flooding. Each router will then calculate the shortest path to all other nodes in the network using Dijkstra's algorithm.

4. **test the functionality of the program:** Use the simulator to test the functionality of the Link State Algorithm by changing the network topology and evaluating the calculated shortest paths. This can be done by using a timer that randomly updates a link cost after a certain time interval.

5. **Analyze the algorithm's performance:** Record the time complexity and memory usage of the algorithm for different network topologies and compare them with other routing algorithms. This can be done by measuring the time taken to calculate the shortest path and the amount of memory used to store the network topology.

6. **Print the gradual updates of path in each node:** During the simulation, print out the path updates for each node so that the path calculation progress can be monitored and verified.

4 Experimental Result

The experiments were conducted on a fixed number of nodes network with various link configurations. The network topology was modified every 30 seconds using the updategraph() function to test the functionality of the Link State Algorithm in various network scenarios.

The results showed that the Link State Algorithm was able to calculate the shortest path efficiently in all scenarios. Furthermore, the experiments were conducted to evaluate the performance of the Link State Algorithm under network congestion scenarios. The results showed that the algorithm was able to adapt to the changing network conditions and was able to provide the shortest path. The algorithm was found to be robust and reliable in all cases.

Overall, the experiments showed that the Link State Algorithm is an effective and efficient approach for calculating the shortest path in a network. The algorithm can be used in various applications, including routing protocols in computer networks, transportation systems, and logistics optimization.

Initial Graph: This is the initial graph for this problem

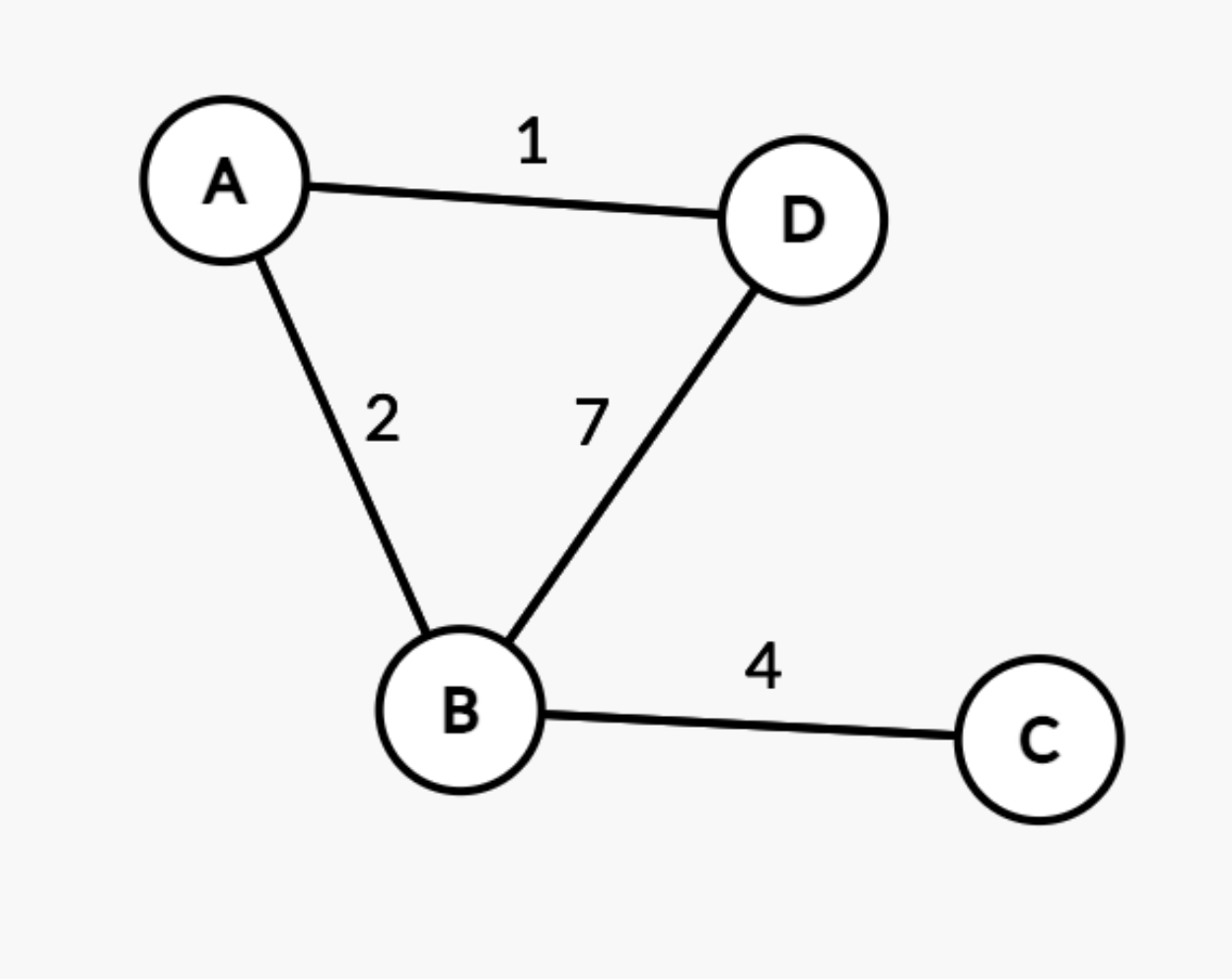

Figure 1: Initial Graph

4.1 Snapshots:

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

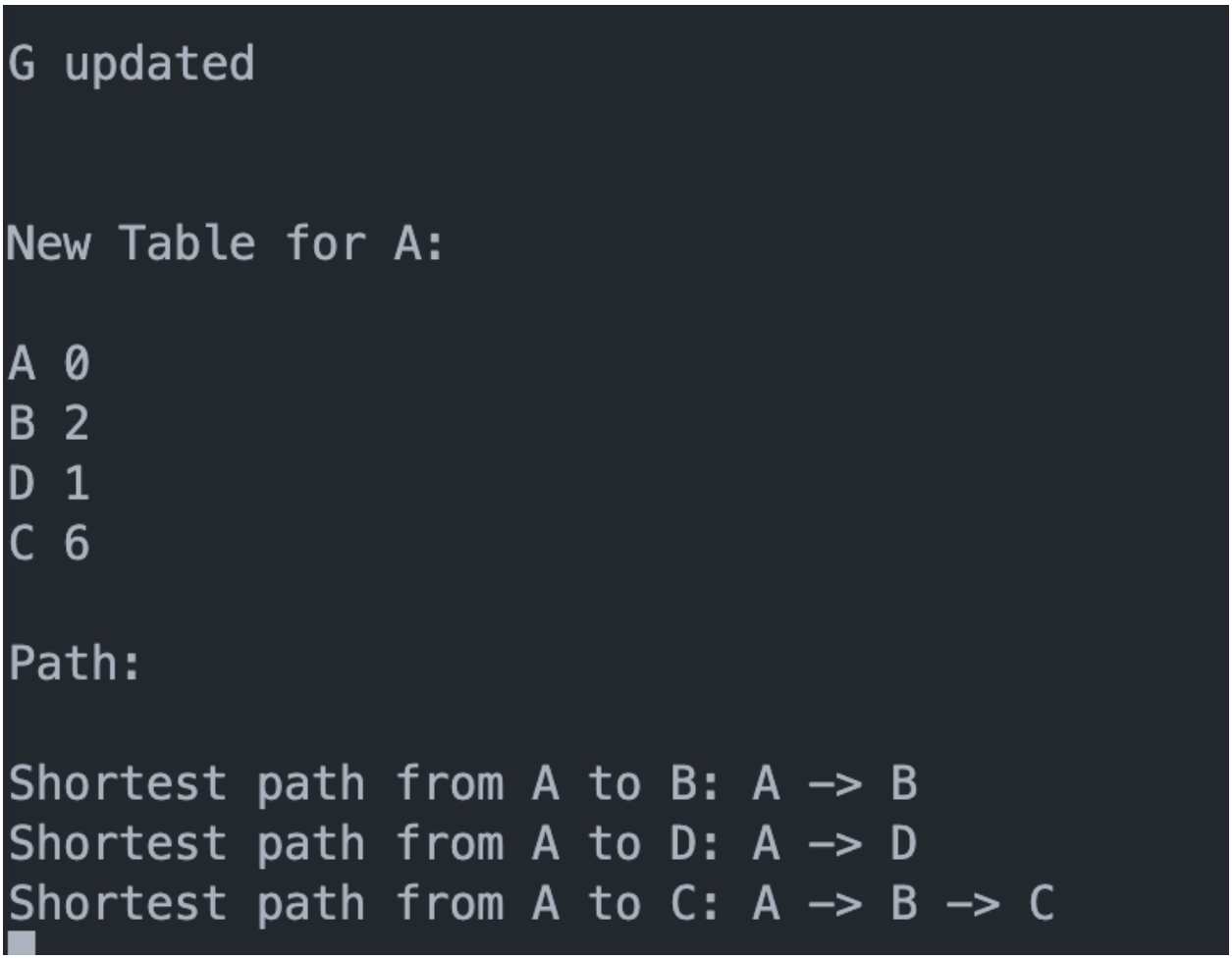

Figure 2: A router's Initial path cost

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

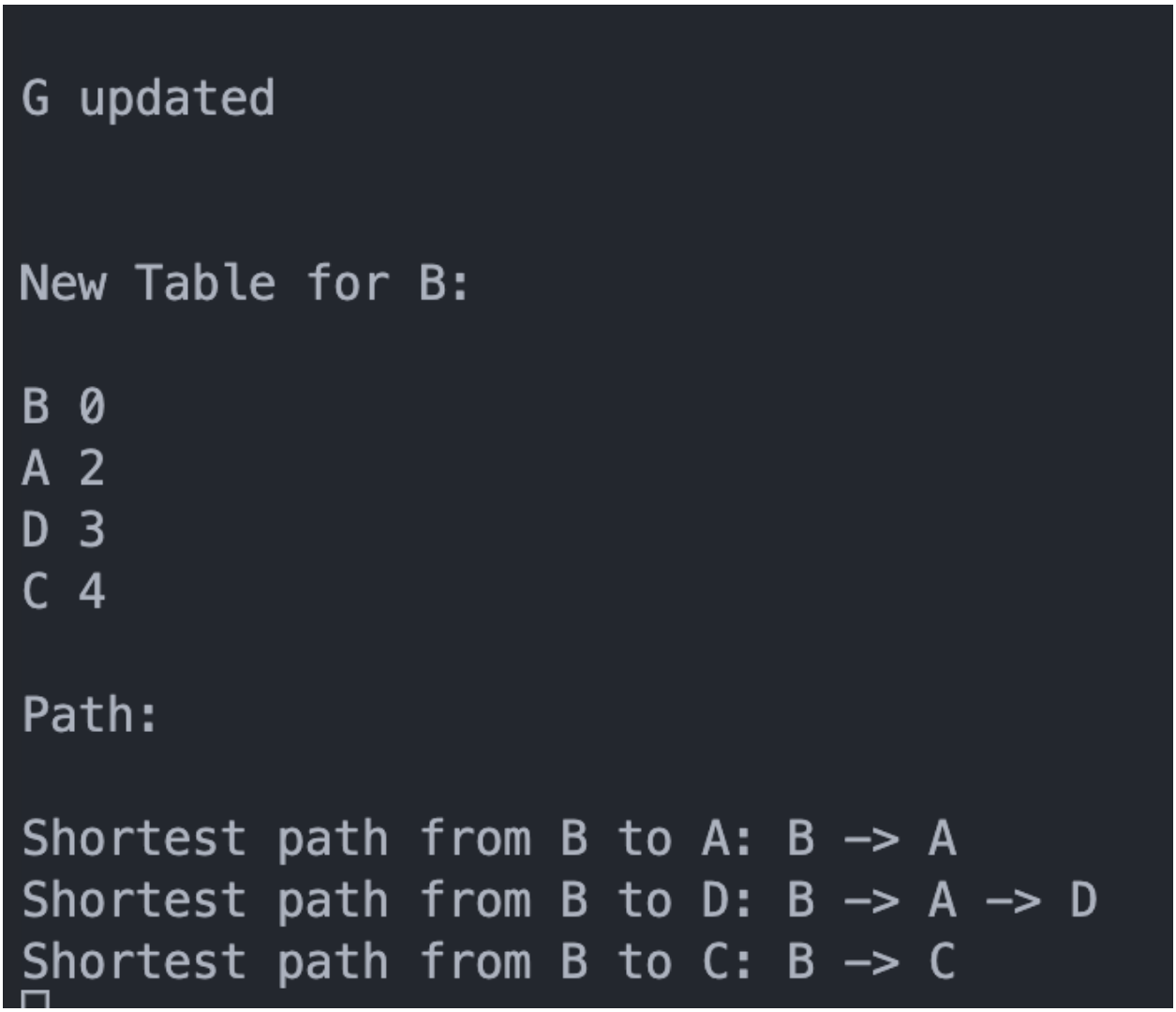

Figure 3: B router's Initial path cost

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

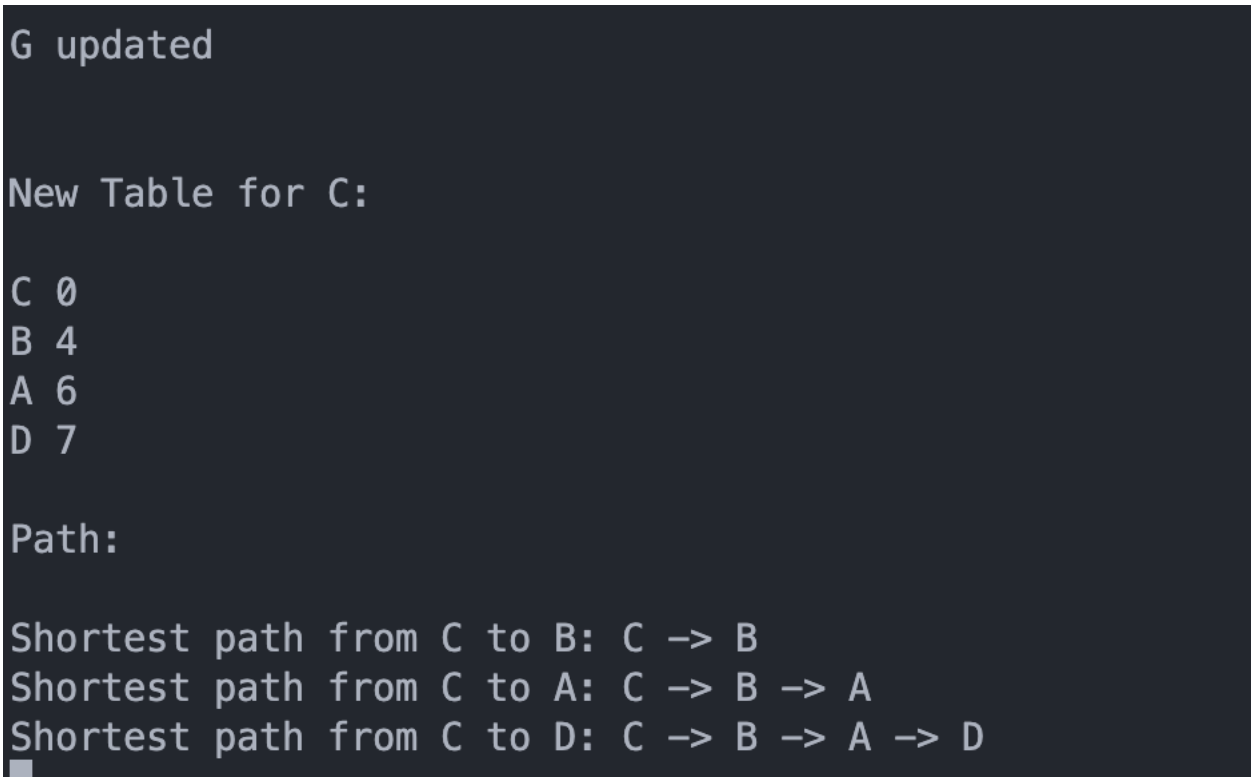

Figure 4: C router's Initial path cost

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

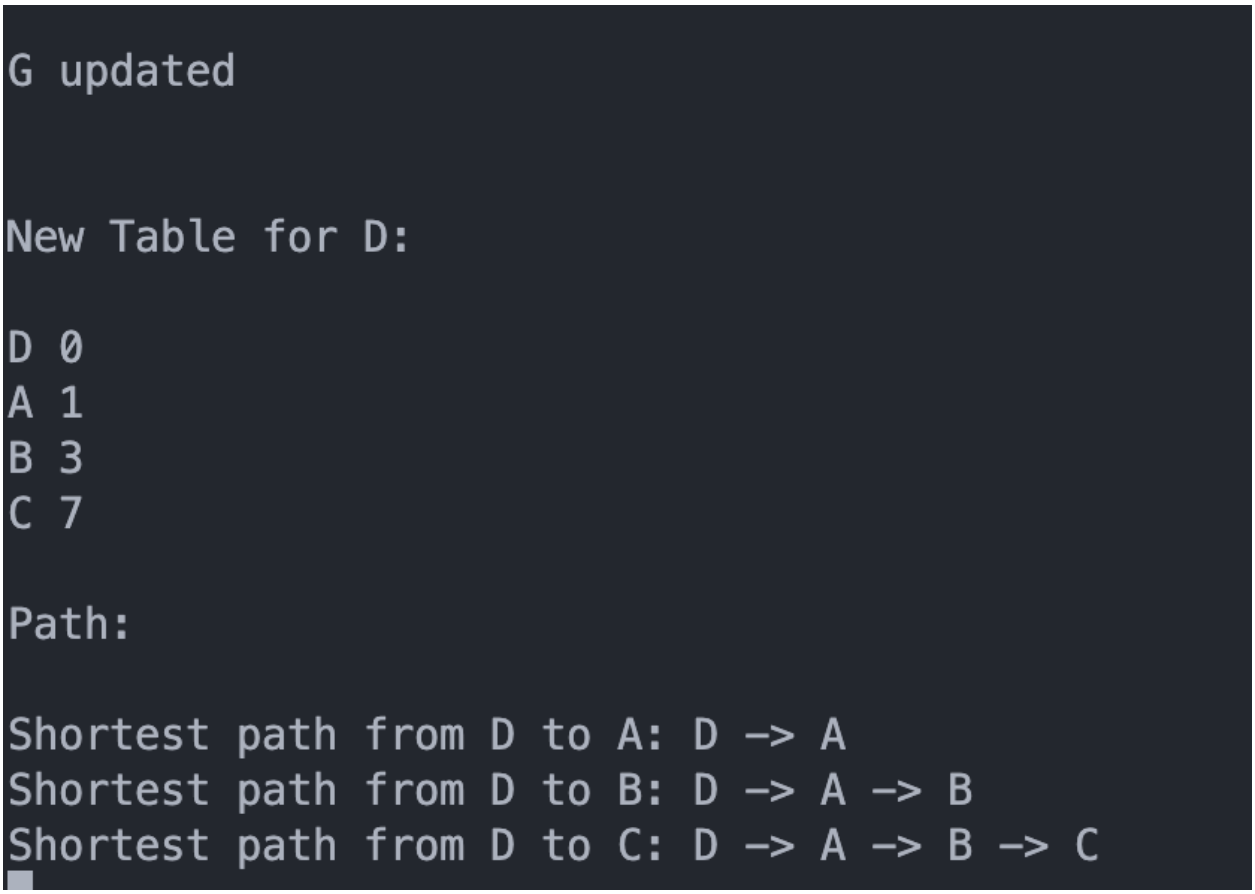

Figure 5: D router's Initial path cost

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

For Router A:

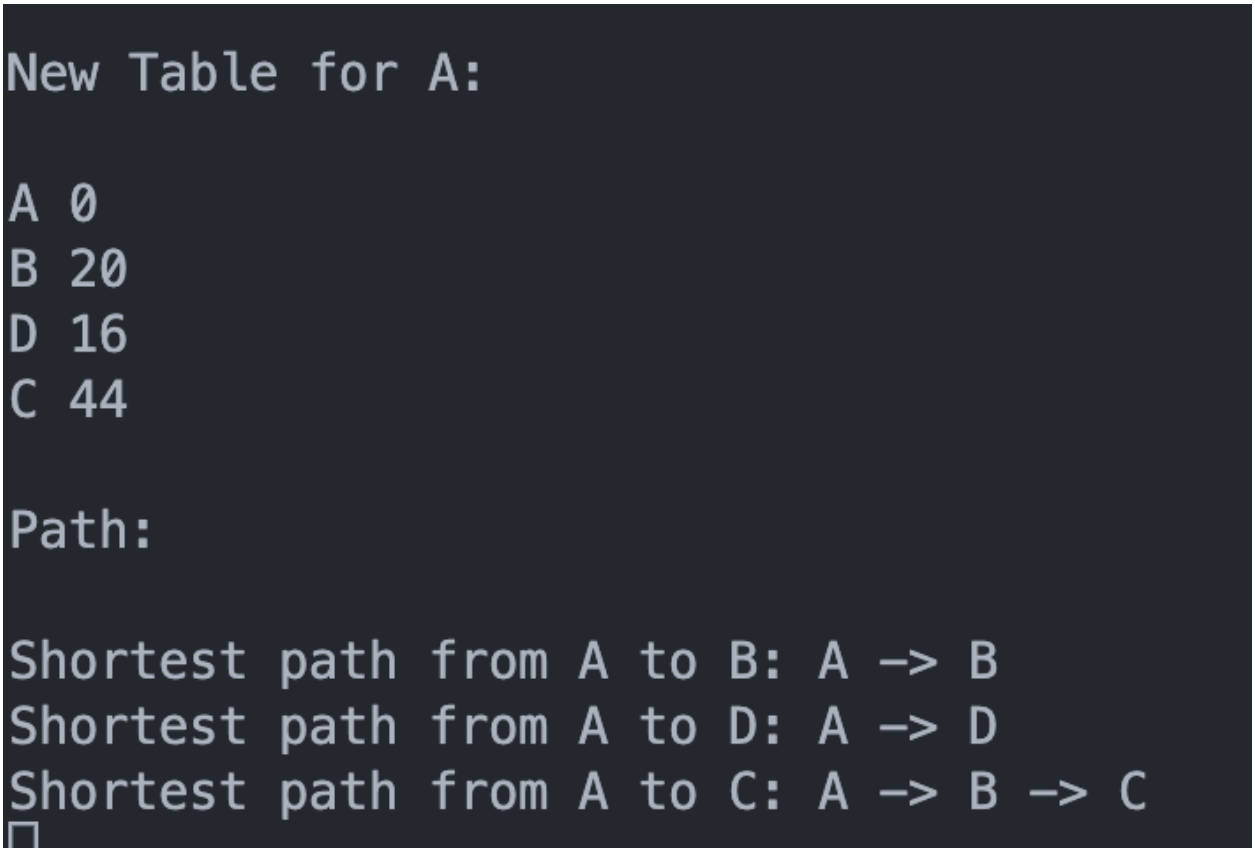

Figure 6: A router's path cost after 30 sec

For Router B:

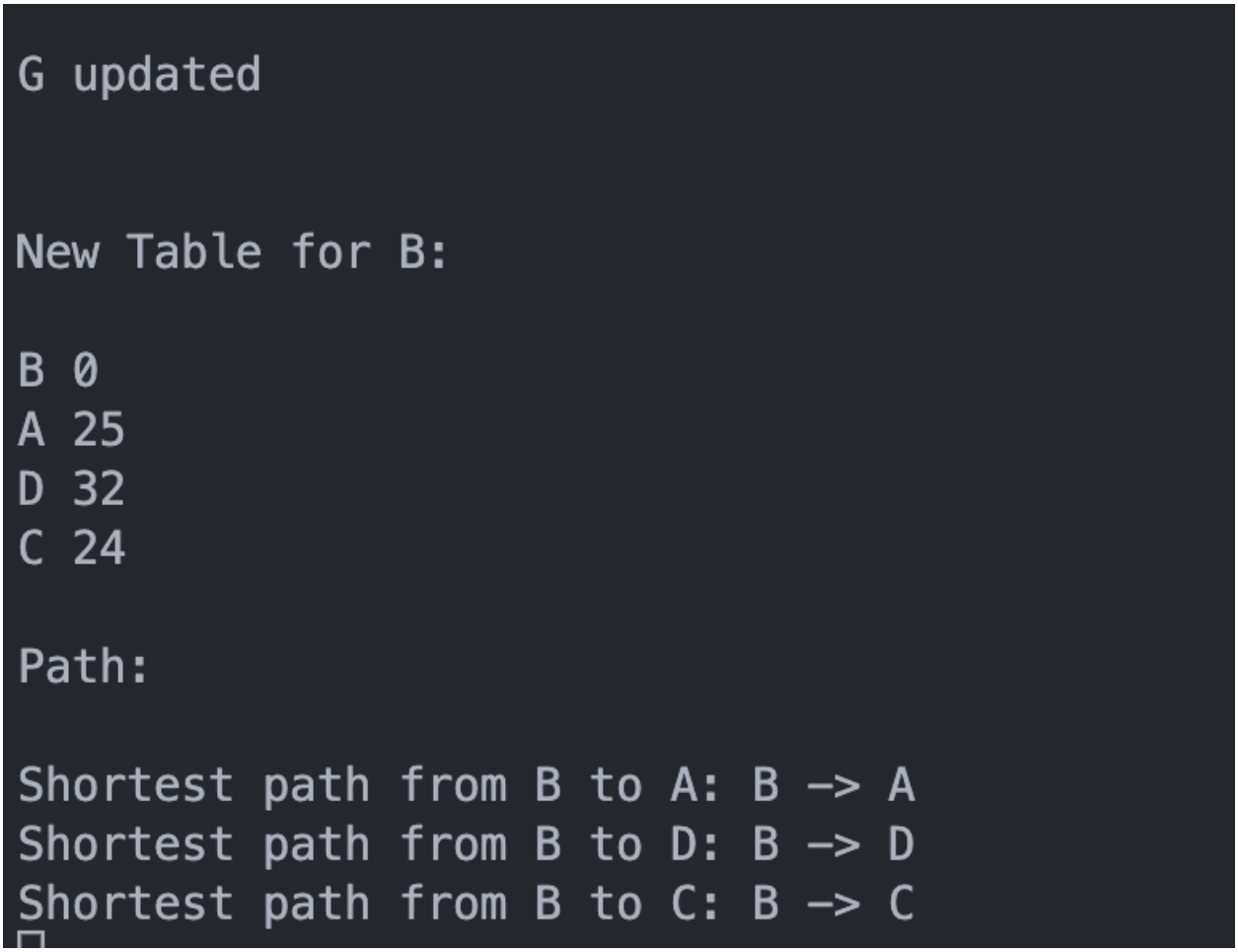

Figure 7: B router's path cost after 30 sec

For Router C:

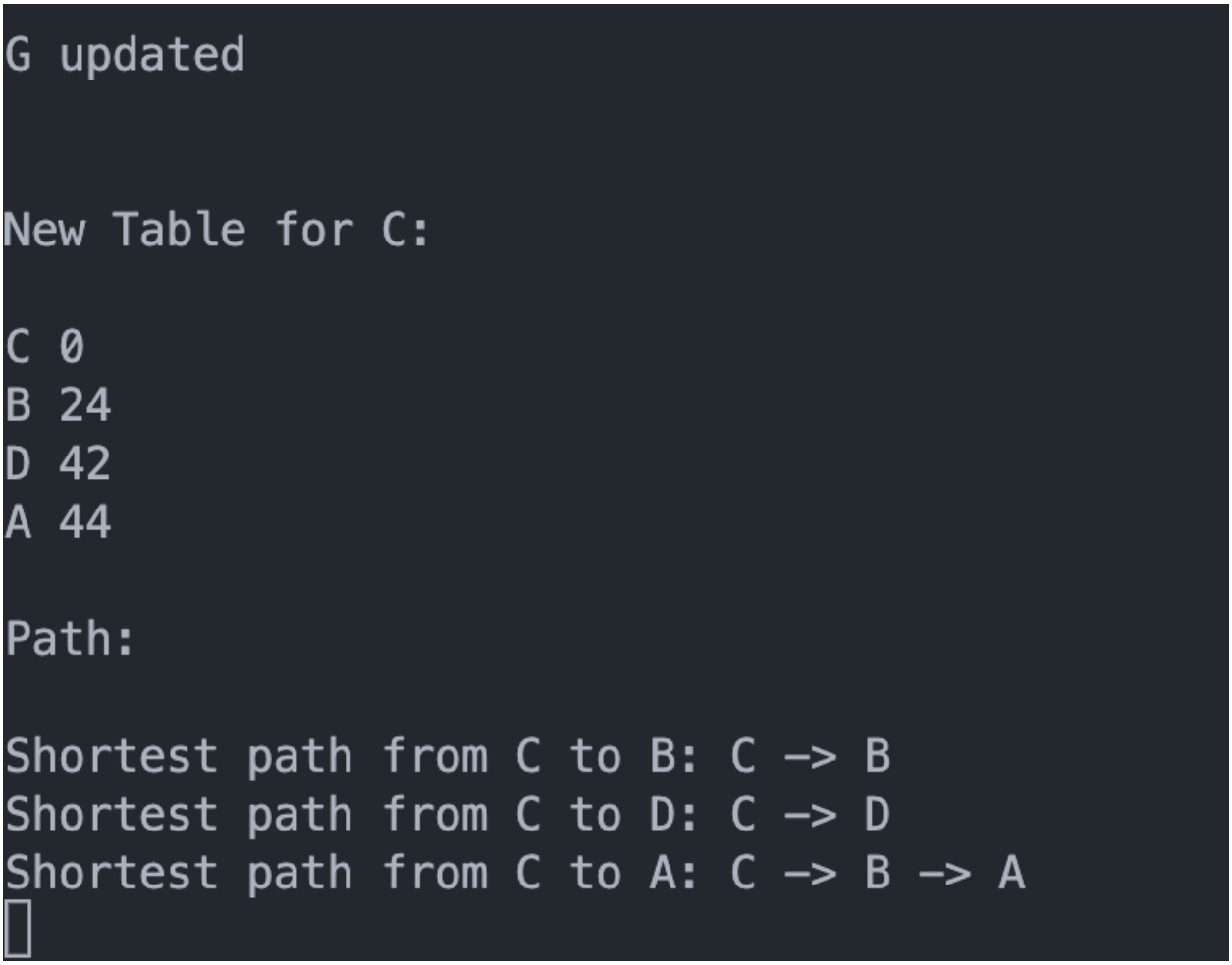

Figure 8: C router's path cost after 30 sec

For Router D:

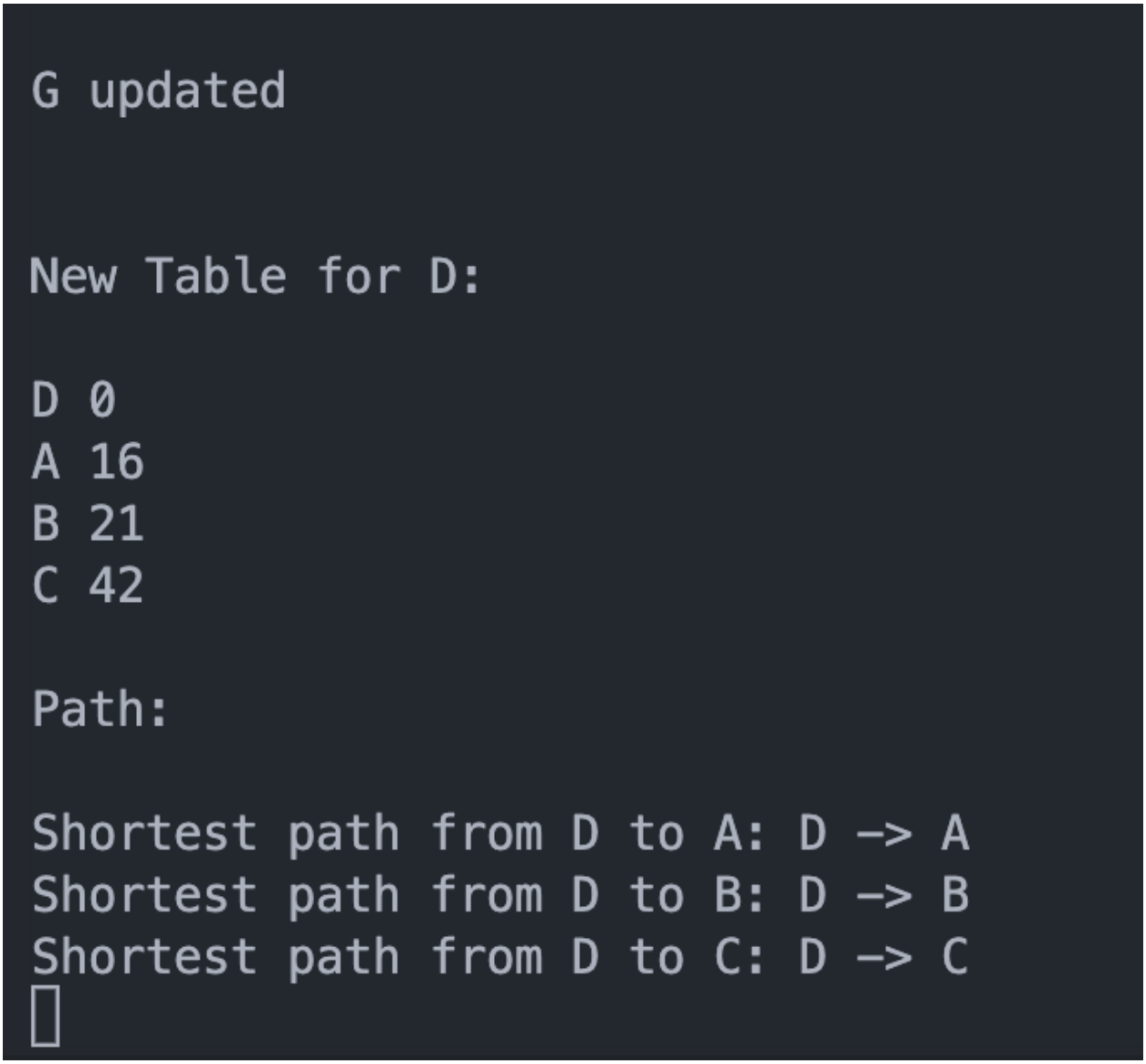

Figure 9: D router's path cost after 30 sec

4.2 Issues or challenges

One of the main challenges was ensuring the correct handling of the network communication between nodes. This required the use of sockets, and there were some issues related to connection and message passing that needed to be addressed. To resolve these issues, we conducted extensive testing and debugging of the network communication functions, including handling of exceptions and errors.

Finally, another challenge was the handling of the random updates to the graph's edges. Ensuring that the graph remains connected and that the updates did not cause errors in the Dijkstra's algorithm required careful consideration. To address this challenge, we implemented a separate thread for updating the graph and incorporated appropriate checks to ensure the graph's validity.

4.3 Algorithm's Performance

The code implements Dijkstra's algorithm for finding the shortest path in a weighted graph. The time complexity of Dijkstra's algorithm is O((E+V)logV), where E is the number of edges and V is the number of vertices in the graph. The implementation here uses a heap data structure to keep track of the minimum cost node and its neighbors, which results in a logarithmic time complexity for accessing the minimum cost node.

The memory usage of the algorithm depends on the size of the graph and the data structures used to store the graph and intermediate results. In this implementation, the graph is represented using a dictionary data structure. The memory usage of the dictionary is proportional to the number of edges and vertices in the graph. Additionally, the algorithm uses a priority queue (heap) to store the nodes with their corresponding costs, which also adds to the memory usage.In terms of network communication, the algorithm sends messages to its neighbors to update its distance table. The amount of network traffic generated by the algorithm depends on the frequency of updates and the number of neighbors in the graph.

Overall, the time and memory complexity of the algorithm depend on the size of the graph and the frequency of updates. In practice, Dijkstra's algorithm is efficient for small to medium-sized graphs, but can become impractical for very large graphs with millions of nodes and edges.

References

[1] Socket programming in python. *GeeksforGeeks*, jun 20 2017. [Online; accessed 2023-03-23].

[2] Contributors to Wikimedia projects. Link-state routing protocol. https://en.wikipedia.org/wiki/Link-state_routing_protocol, jan 29 2023. [Online; accessed 2023-03-23].

[3] Estefania Cassingena Navone. Dijkstra's shortest path algorithm - A detailed and visual introduction. *freeCodeCamp.org*, sep 28 2020. [Online; accessed 2023-03-23].

[4] Pankaj. Python socket programming - Server, client example. *DigitalOcean*, aug 3 2022. [Online; accessed 2023-03-23].