PRN: 21070126039   Name: Jainil Patel   Batch: AI/ML A2

CN LAB 8

Implementation of Shortest Path Algorithm (Dijkstra's Algorithm)

Learning Outcome

After Completion of this experiment, students will be able to understand and implement Shortest Path Algorithm (Dijkstra's Algorithm)

Theory

Dijkstra's Algorithm is a graph-based, single-source, shortest path algorithm used in computer science and network routing. It finds the shortest path from a starting node to all other nodes by iteratively selecting the node with the smallest cumulative distance and updating distances to neighboring nodes. It ensures optimality for non-negative edge weights, making it valuable in routing, navigation, and network design. Dijkstra's Algorithm is widely applied in routing packets across computer networks, calculating directions in mapping applications, and optimizing transportation routes.

Steps

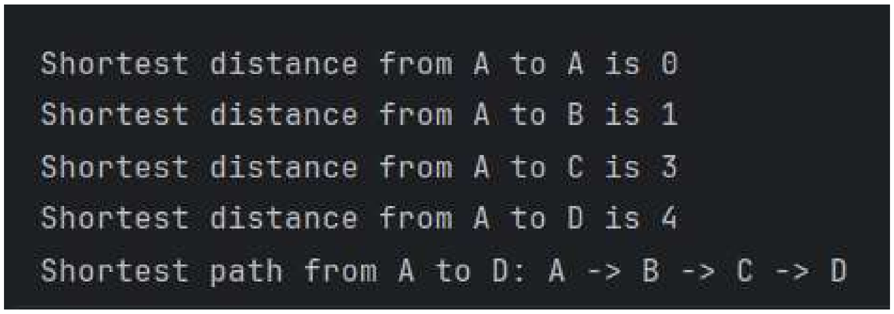
Steps for implementing Dijkstra's Algorithm to find the shortest path in a graph:

1. **Initialization:**
   - Create a list of unvisited nodes and set the distance to the source node as 0. Set the distance to all other nodes as infinity.
   - Create an empty set for the set of visited nodes.
2. **Select Node:**
   - Choose the unvisited node with the smallest distance as the current node.
   - Initially, this would be the source node.
3. **Explore Neighbours:**
   - For the current node, examine all unvisited neighbors.
   - Calculate their tentative distances through the current node. Update their distances if a shorter path is found.
4. **Mark as Visited:**
   - Once you have considered all of the unvisited neighbors of the current node, mark the current node as visited.
   - A visited node will not be checked again.
5. **Select the Next Node:**
   - Select the unvisited node with the smallest tentative distance and set it as the new current node.
   - Go back to step 3.
6. **Repeat:**
   - Continue this process until you have visited all nodes or until the destination node is marked as visited.
7. **Path Reconstruction:**
   - Once the destination node has been marked as visited, you can backtrack from the destination node to the source node to determine the shortest path.

Code

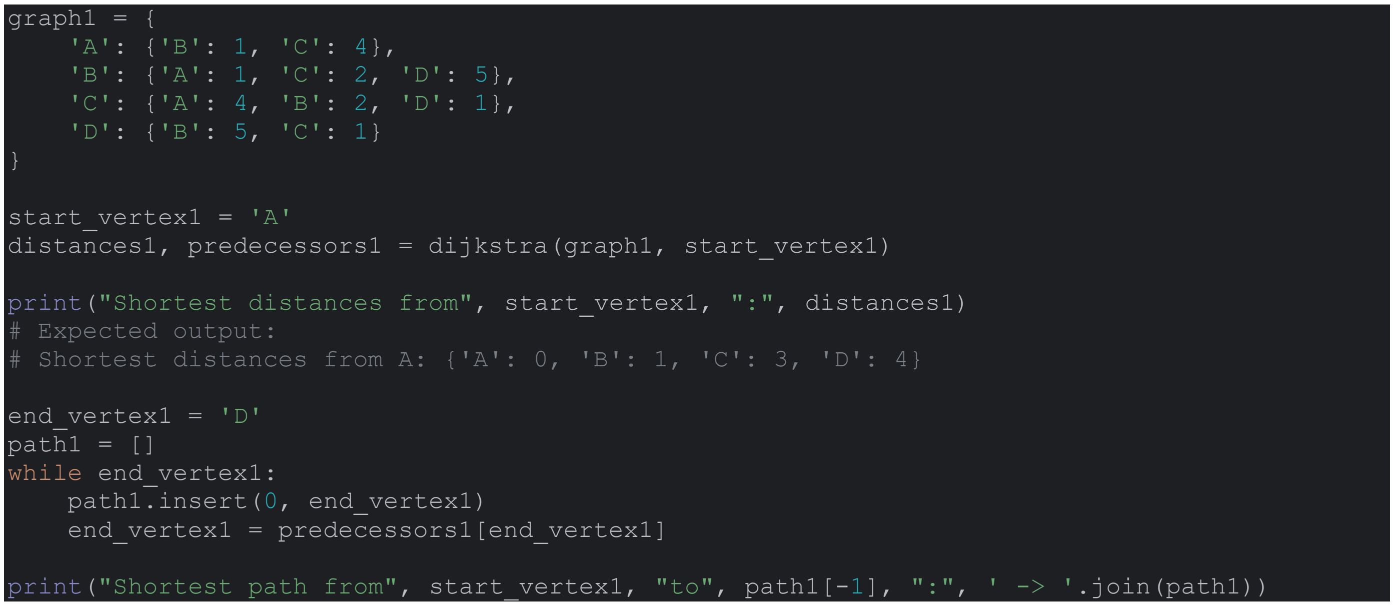
```python
import heapq

def dijkstra(graph, start):
    # Initialize distance and predecessor dictionaries
    distance = {vertex: float('infinity') for vertex in graph}
    previous = {vertex: None for vertex in graph}

    # Set the distance to the starting node to 0
    distance[start] = 0

    # Create a priority queue to store vertices and their distances
    priority_queue = [(0, start)]

    while priority_queue:
        # Get the vertex with the smallest distance
        current_distance, current_vertex = heapq.heappop(priority_queue)

        # If the current distance is greater than the recorded distance, skip
        if current_distance > distance[current_vertex]:
            continue

        # Explore the neighbors of the current vertex
        for neighbor, weight in graph[current_vertex].items():
            distance_to_neighbor = distance[current_vertex] + weight

            # If the new distance is shorter, update the distance and predecessor
            if distance_to_neighbor < distance[neighbor]:
                distance[neighbor] = distance_to_neighbor
                previous[neighbor] = current_vertex
                heapq.heappush(priority_queue, (distance_to_neighbor, neighbor))

    return distance, previous

# Example usage:
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 5},
    'C': {'A': 4, 'B': 2, 'D': 1},
    'D': {'B': 5, 'C': 1}
}

start_vertex = 'A'
distances, predecessors = dijkstra(graph, start_vertex)

# Print the shortest distances from the starting vertex
for vertex, distance in distances.items():
    print(f"Shortest distance from {start_vertex} to {vertex} is {distance}")

# Print the shortest path from the starting vertex to a specific vertex
end_vertex = 'D'
path = []
while end_vertex:
    path.insert(0, end_vertex)
    end_vertex = predecessors[end_vertex]

print(f"Shortest path from {start_vertex} to {path[-1]}: {' -> '.join(path)}")
```

Output:



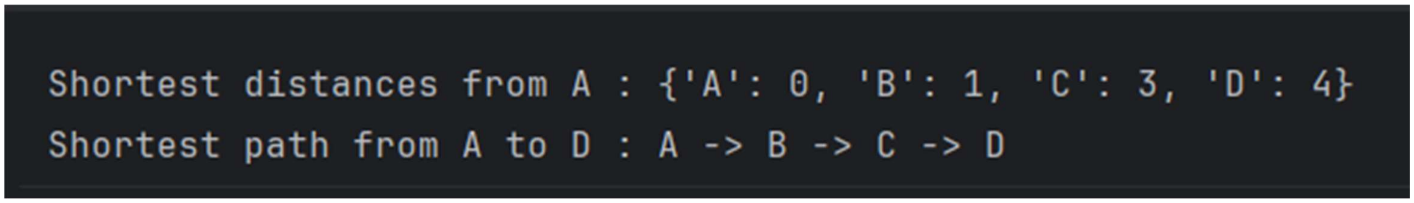
```
Shortest distance from A to A is 0
Shortest distance from A to B is 1
Shortest distance from A to C is 3
Shortest distance from A to D is 4
Shortest path from A to D: A -> B -> C -> D
```

Graph 1:



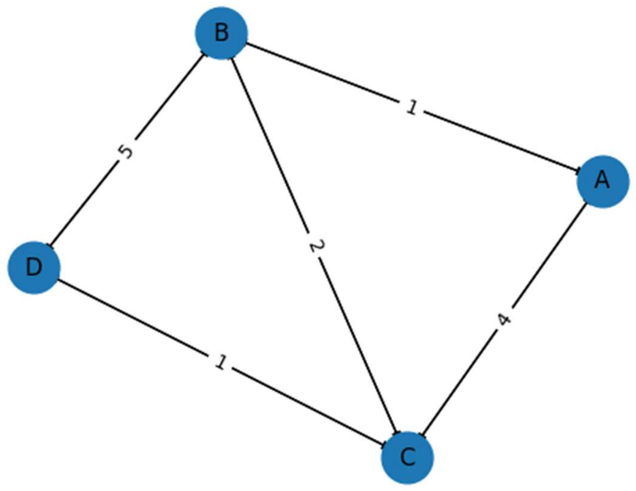
```
graph1 = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 5},
    'C': {'A': 4, 'B': 2, 'D': 1},
    'D': {'B': 5, 'C': 1}
}

start_vertex1 = 'A'
distances1, predecessors1 = dijkstra(graph1, start_vertex1)

print("Shortest distances from", start_vertex1, ":", distances1)
# Expected output:
# Shortest distances from A: {'A': 0, 'B': 1, 'C': 3, 'D': 4}

end_vertex1 = 'D'
path1 = []
while end_vertex1:
    path1.insert(0, end_vertex1)
    end_vertex1 = predecessors1[end_vertex1]

print("Shortest path from", start_vertex1, "to", path1[-1], ":", ' -> '.join(path1))
```

Output:

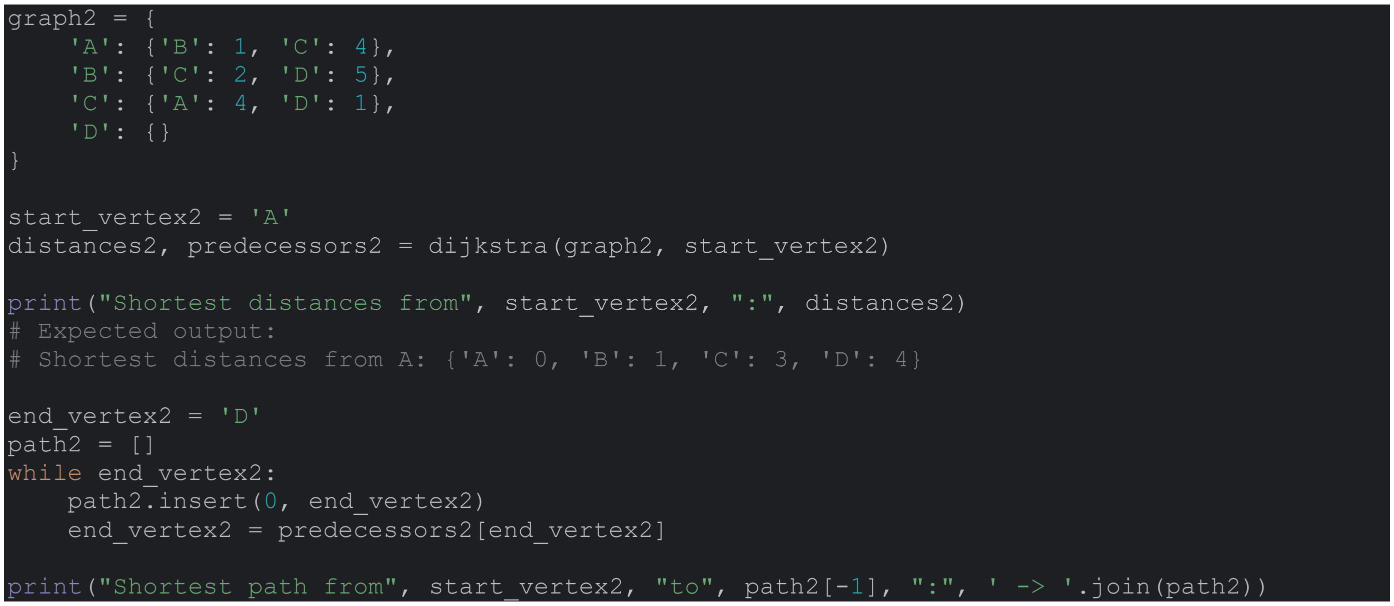


```
Shortest distances from A : {'A': 0, 'B': 1, 'C': 3, 'D': 4}
Shortest path from A to D : A -> B -> C -> D
```

Test Case 1: Basic Example

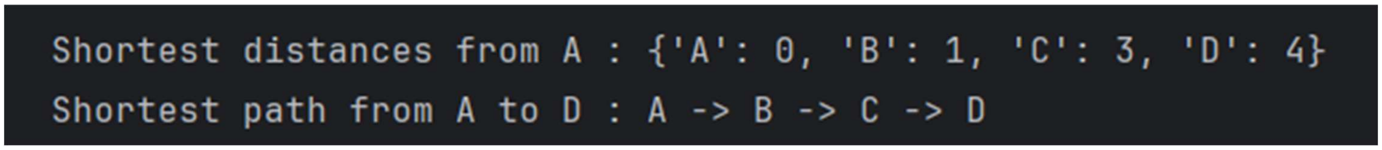


Graph 2:



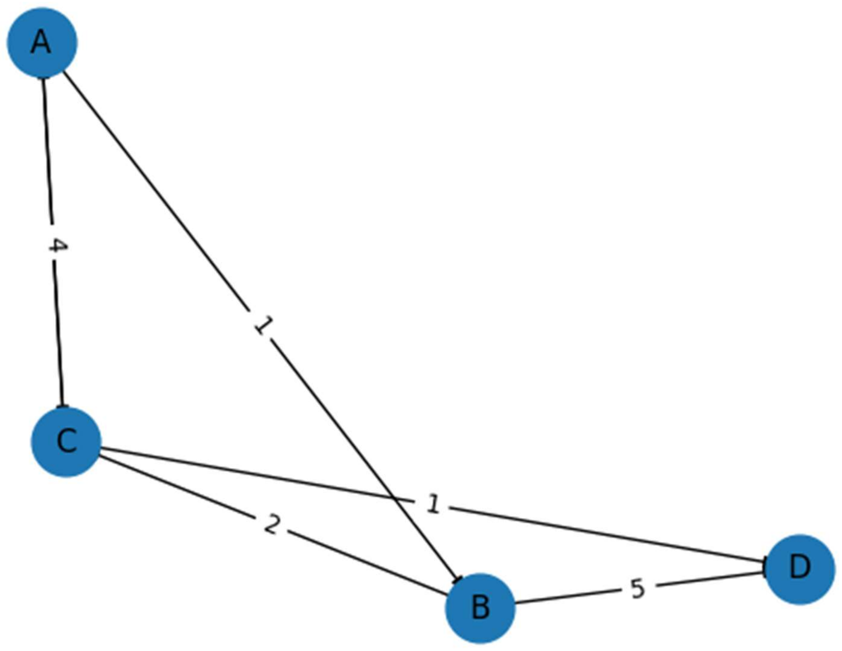
```
graph2 = {
    'A': {'B': 1, 'C': 4},
    'B': {'C': 2, 'D': 5},
    'C': {'A': 4, 'D': 1},
    'D': {}
}

start_vertex2 = 'A'
distances2, predecessors2 = dijkstra(graph2, start_vertex2)

print("Shortest distances from", start_vertex2, ":", distances2)
# Expected output:
# Shortest distances from A: {'A': 0, 'B': 1, 'C': 3, 'D': 4}

end_vertex2 = 'D'
path2 = []
while end_vertex2:
    path2.insert(0, end_vertex2)
    end_vertex2 = predecessors2[end_vertex2]

print("Shortest path from", start_vertex2, "to", path2[-1], ":", ' -> '.join(path2))
```

Output:



```
Shortest distances from A : {'A': 0, 'B': 1, 'C': 3, 'D': 4}
Shortest path from A to D : A -> B -> C -> D
```

Test Case 2: Directed Graph



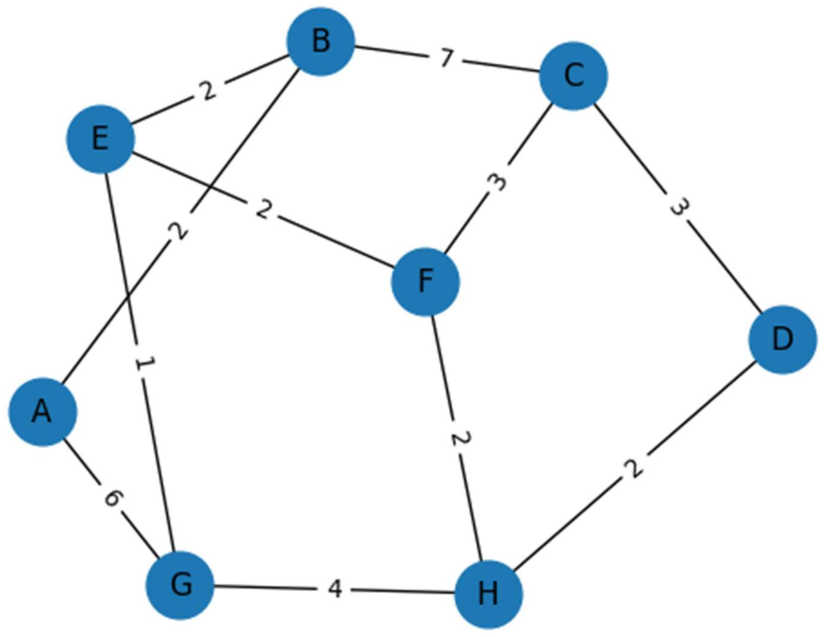
Graph 3:

```
graph3 = {
    'A': {'B': 2, 'G': 6},
    'B': {'A': 2 ,'C': 7 , 'E': 2},
    'C': {'B': 7 ,'D': 3 , 'F' : 3},
    'D': {'C': 3 , 'H': 2},
    'E': {'B': 2 ,'G': 1,'F': 2},
    'F': {'C': 3 , 'H': 2 , 'E' : 2},
    'G': {'A': 6 , 'H': 4 , 'E' : 1},
    'H': {'G': 4 , 'F': 2 , 'D' : 2}

}

start_vertex3 = 'A'
distances3, predecessors3 = dijkstra(graph3, start_vertex3)

print("Shortest distances from", start_vertex3, ":", distances3)

end_vertex3 = 'D'
path3 = []
while end_vertex3:
    path3.insert(0, end_vertex3)
    end_vertex3 = predecessors3[end_vertex3]

print("Shortest path from", start_vertex3, "to", path3[-1], ":", ' -> '.join(path3))
```

Output:

```
Shortest distances from A : {'A': 0, 'B': 2, 'C': 9, 'D': 10, 'E': 4, 'F': 6, 'G': 5, 'H': 8}
Shortest path from A to D : A -> B -> E -> F -> H -> D
```

Test Case 3: Disconnected Graph



Self-Assessment

1. Enlist types of routing algorithms in Computer network?

Types of Routing Algorithms in Computer Networks: In computer networks, there are several routing algorithms used to determine the best path for data to travel from the source to the destination. Some common types of routing algorithms include:

- Static Routing: This involves manually configuring routing tables to define the path for data to follow. It's simple but not very adaptive.
- Dynamic Routing: Dynamic routing algorithms, like OSPF (Open Shortest Path First) and RIP (Routing Information Protocol), automatically adjust routing tables based on network changes.
- Shortest Path Routing: These algorithms find the shortest path to a destination, such as Dijkstra's algorithm and Bellman-Ford algorithm.
- Distance Vector Routing: Routing tables are updated based on the number of hops to a destination, with algorithms like RIP.
- Link-State Routing: Algorithms like OSPF consider the state and cost of individual links in the network to determine the best path.

2. Explain shortest path routing in Computer network?

In this algorithm, to select a route, the algorithm discovers the shortest path between two nodes. It can use multiple hops, the geographical area in kilometres or labelling of arcs for measuring path length.
The labelling of arcs can be done with mean queuing, transmission delay for a standard test packet on an hourly basis, or computed as a function of bandwidth, average distance traffic, communication cost, mean queue length, measured delay, or some other factors.
In shortest path routing, the topology communication network is defined using a directed weighted graph. The nodes in the graph define switching components and the directed arcs in the graph define communication connection between switching components. Each arc has a weight that defines the cost of sharing a packet between two nodes in a specific direction.
This cost is usually a positive value that can denote such factors as delay, throughput, error rate, financial costs, etc. A path between two nodes can go through various intermediary nodes and arcs. The goal of shortest path routing is to find a path between two nodes that has the lowest total cost, where the total cost of a path is the sum of arc costs in that path.
For example, Dijkstra uses the nodes labelling with its distance from the source node along the better-known route. Initially, all nodes are labelled with infinity, and as the algorithm proceeds, the label may change.

3. What is static routing and dynamic routing?

- **Static Routing**

It is also known as "non-adaptive routing". The network administrator contains the routes in the routing table while using this routing. As a result, the router transfers data from the source to the destination using the administrator-defined route. Routing decisions don't depend on factors like network traffic or topology.
Furthermore, static routing doesn't need much bandwidth between routers. The network is also more secure because the network administrator conducts the necessary routing activities. Furthermore, the overall cost of static routing is less. Static routing is also unsuitable for big networks with significant traffic due to the difficulties of manually adding routes to routing tables. Furthermore, the administrator must be familiar with the network to add routes to the routing table manually.

- **Dynamic Routing**

Dynamic routing is also known as "adaptive routing", and it is a method of automatic routing. In other words, when new routers are introduced to the network, the routing tables change. When a router fails, the routing table automatically modifies to get the destination. As a result, dynamic routing responds to network and traffic changes.

This routing method employs dynamic routing algorithms to find new routes to the destination. As a result, all network routers should use dynamic routing protocols that are consistent.
Dynamic routing needs fewer routes. Moreover, it offers more accurate results in determining the optimum path based on network changes. However, this routing method needs more bandwidth and offers less security.

4. What is spanning tree algorithm?

A Spanning Tree Algorithm, particularly the Spanning Tree Protocol (STP) and its fast variant, the Rapid Spanning Tree Protocol (RSTP), is a crucial network technology used to eliminate loops in Ethernet networks. It ensures network reliability and stability by creating a loop-free topology. Below, I'll explain this concept and provide an example of its application.

Spanning Tree Algorithm:

Ethernet networks are susceptible to loops, which can lead to broadcast storms and network congestion. A Spanning Tree Algorithm addresses this problem by creating a loop-free network topology. The algorithm selects one switch (usually the root bridge) as the "root" and then disables certain links to form a spanning tree while ensuring redundancy for fault tolerance.

Example Application:

Imagine a large office building with multiple switches interconnected to provide network connectivity. Without a Spanning Tree Algorithm, there's a risk of network loops. For instance, consider two switches, A and B, each connected to a different server, S1 and S2. If a broadcast message is sent from S1, it could potentially circulate endlessly between switches A and B, leading to a broadcast storm.

Applications of minimum spanning tree

The applications of the minimum spanning tree are given as follows -

- Minimum spanning tree can be used to design water-supply networks, telecommunication networks, and electrical grids.
- It can be used to find paths in the map.

Algorithms for Minimum spanning tree

A minimum spanning tree can be found from a weighted graph by using the algorithms given below -

- Prim's Algorithm
- Kruskal's Algorithm

5. What is need of different routing algorithms in Computer network?

Routing algorithm

- In order to transfer the packets from source to the destination, the network layer must determine the best route through which packets can be transmitted.
- Whether the network layer provides datagram service or virtual circuit service, the main job of the network layer is to provide the best route. The routing protocol provides this job.
- The routing protocol is a routing algorithm that provides the best path from the source to the destination. The best path is the path that has the "least-cost path" from source to the destination.
- Routing is the process of forwarding the packets from source to the destination but the best route to send the packets is determined by the routing algorithm.

Adaptive Routing algorithm

- An adaptive routing algorithm is also known as dynamic routing algorithm.
- This algorithm makes the routing decisions based on the topology and network traffic.
- The main parameters related to this algorithm are hop count, distance and estimated transit time.

Non-Adaptive Routing algorithm

- Non Adaptive routing algorithm is also known as a static routing algorithm.
- When booting up the network, the routing information stores to the routers.
- Non Adaptive routing algorithms do not take the routing decision based on the network topology or network traffic.

Conclusion

In the realm of computer networking and graph theory, we've explored various fundamental concepts and algorithms that play a pivotal role in ensuring efficient and reliable data transmission. Routing algorithms, such as Dijkstra's Algorithm and the Spanning Tree Algorithm, are instrumental in determining optimal paths and eliminating network loops. Dijkstra's Algorithm excels at finding the shortest path between nodes, making it vital for network routing and navigation applications. The Spanning Tree Algorithm, including STP and RSTP, prevents network loops, safeguarding network stability. Additionally, the dichotomy between static and dynamic routing showcases the trade-off between manual configuration and automated adaptation in network management. Understanding these principles is crucial for designing, managing, and troubleshooting modern computer networks, ensuring the seamless flow of data in an interconnected world.