LINK STATE ROUTING

Link state routing is the second family of routing protocols. While distance-vector routers use a distributed algorithm to compute their routing tables, link-state routing uses link-state routers to exchange messages that allow each router to learn the entire network topology. Based on this learned topology, each router is then able to compute its routing table by using the shortest path computation.

Features of Link State Routing Protocols

- **Link State Packet:** A small packet that contains routing information.
- **Link-State Database:** A collection of information gathered from the link-state packet.
- **Shortest Path First Algorithm (Dijkstra algorithm):** A calculation performed on the database results in the shortest path
- **Routing Table:** A list of known paths and interfaces.

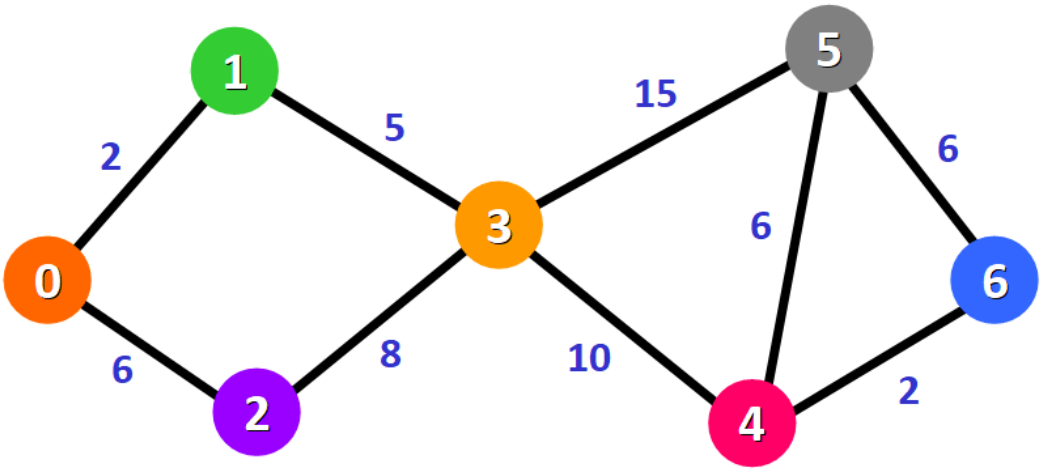
Dijkstra's shortest path algorithm

This algorithm is used to calculate and find the shortest path between nodes using the weights given in a graph. (In a network, the weights are given by link-state packets and contain information such as the health of the routers, traffic costs, etc.).

A node is then marked as **visited** and added to the path if the distance between it and the source node is the shortest. This continues until all the nodes have been added to the path, and finally, we get the shortest path from the source node to all other nodes, which packets in a network can follow to their destination.

An example illustrating the working of the algorithm

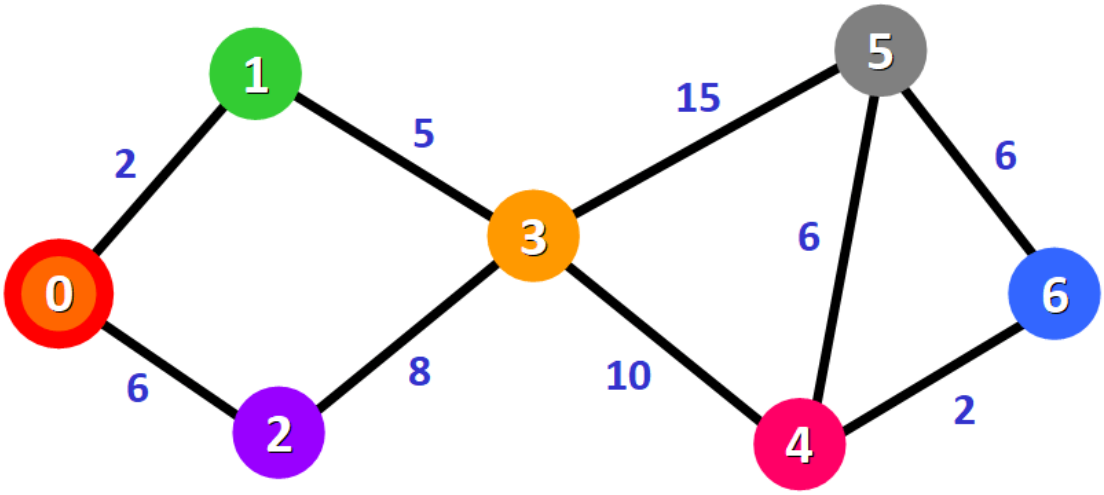
The source node here is node **0**. We assume the weights show the distances.



Initially, we have this list of distances. We mark the initial distances as INF (infinity) because we have not yet determined the actual distance except for node 0. After all, the distance from the node 0 to itself is 0.

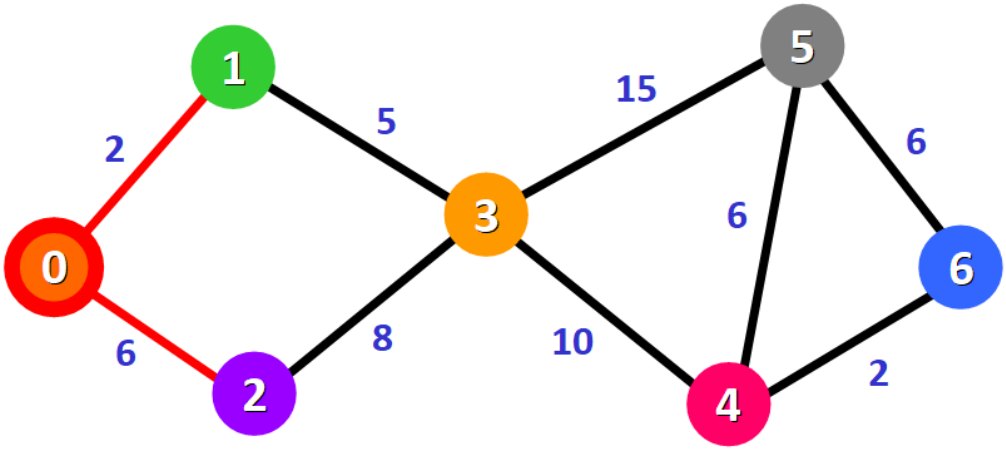
NODE	DISTANCE
0	0
1	INF
2	INF
3	INF
4	INF
5	INF
6	INF

We also have a list to keep track of only the visited nodes, and since we have started with node 0, we add it to the list (we denote a visited node by adding an asterisk beside it in the table and a red border around it on the graph).



{0}

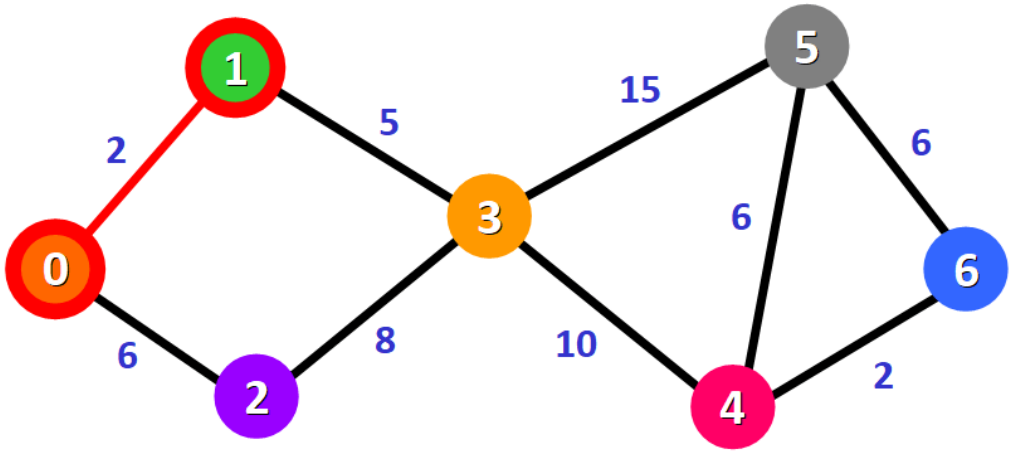
We check the distances 0 -> 1 and 0 -> 2, which are 2 and 6, respectively. We first update the distances from nodes 1 and 2 in the table.



NODE	DISTANCE
0	0
1	2
2	6
3	INF
4	INF

NODE	DISTANCE
5	INF
6	INF

We then choose the shortest one, which is 0 -> 1 and mark node 1 as visited and add it to the visited path list.

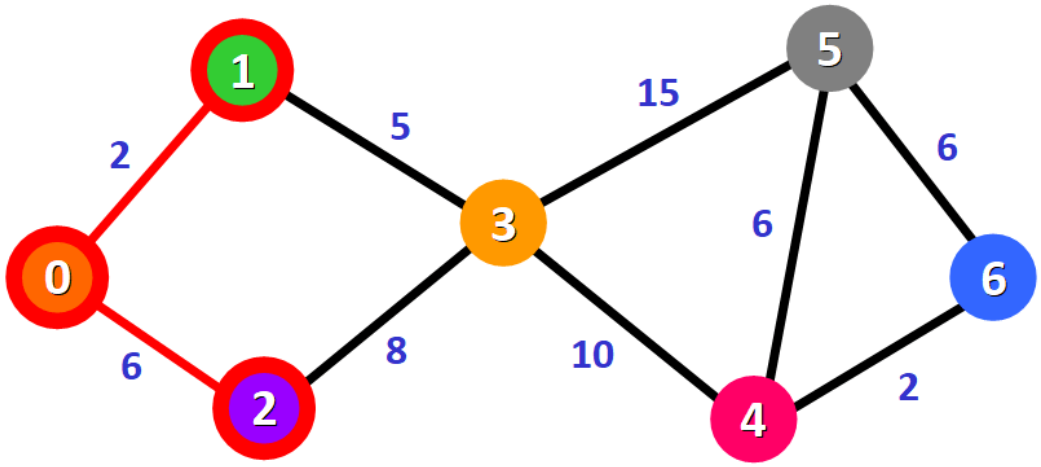


NODE	DISTANCE
0	0
1	2*
2	6
3	INF
4	INF
5	INF
6	INF

{0,1}

Next, we check the nodes adjacent to the nodes added to the path(Nodes 2 and 3). We then update our distance table with the distance from the source node to the new adjacent node, node 3 (2 + 5 = 7).

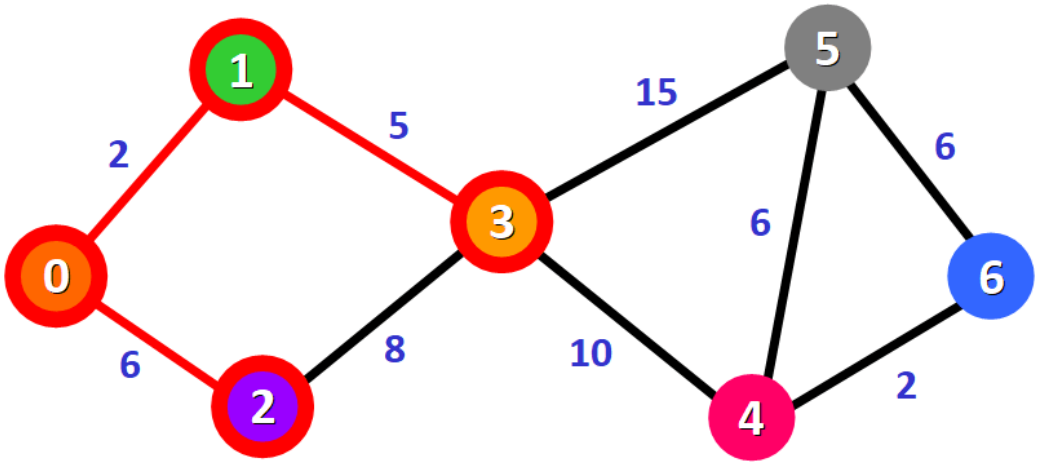
To choose what to add to the path, we select the node with the shortest currently known distance to the source node, which is 0 -> 2 with distance 6.



NODE	DISTANCE
0	0
1	2*
2	6*
3	7
4	INF
5	INF
6	INF

{0,1,2}

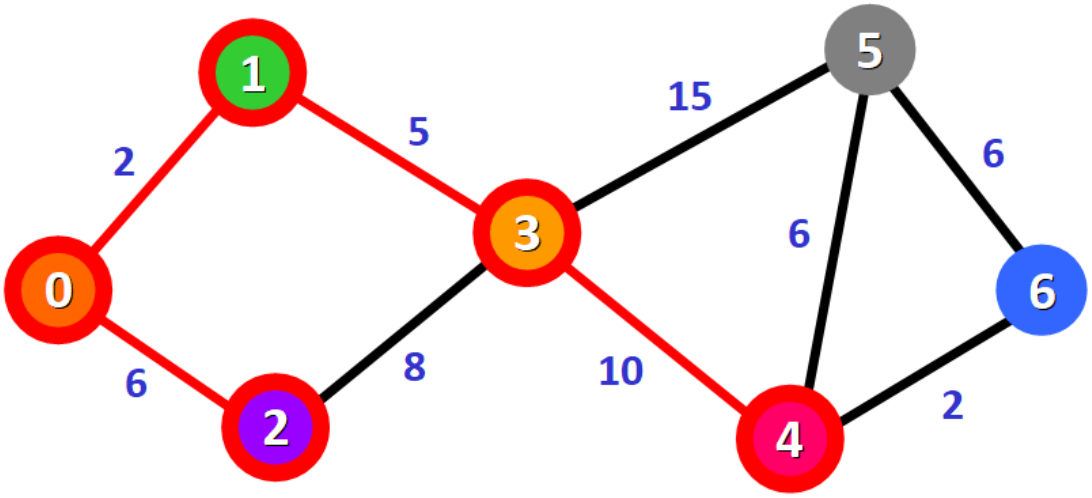
Next we have the distances 0 -> 1 -> 3(2 + 5 = 7) and 0 -> 2 -> 3(6 + 8 = 14) in which 7 is clearly the shorter distance, so we add node 3 to the path and mark it as visited.



NODE	DISTANCE
0	0
1	2*
2	6*
3	7*
4	INF
5	INF
6	INF

{0,1,2,3}

We then check the next adjacent nodes (node 4 and 5) in which we have 0 -> 1 -> 3 -> 4 (7 + 10 = 17) for node 4 and 0 -> 1 -> 3 -> 5 (7 + 15 = 22) for node 5. We add node 4.



NODE	DISTANCE
0	0
1	2*
2	6*
3	7*
4	17*
5	22
6	INF

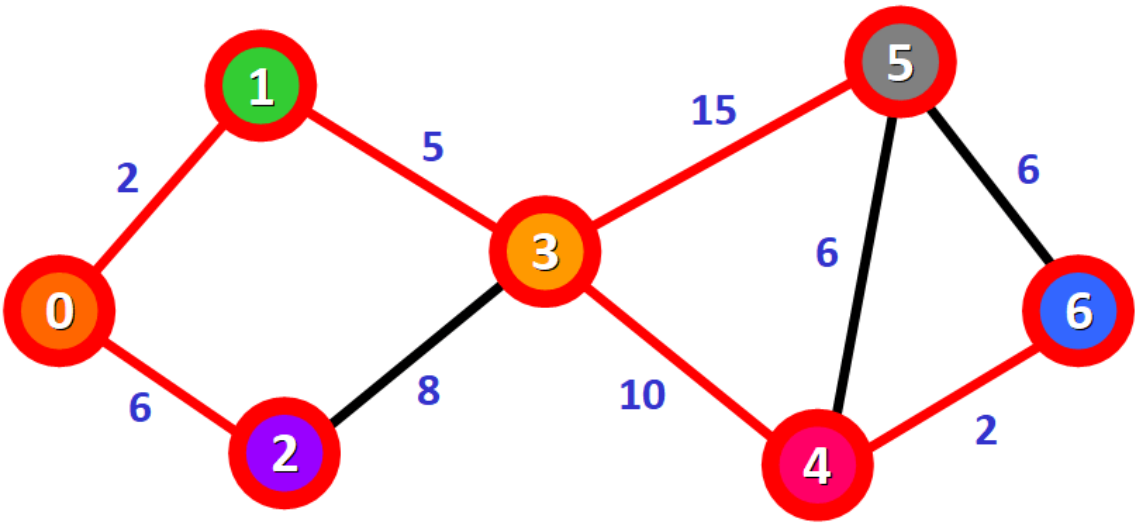
{0,1,2,3,4}

In the same way, we check the adjacent nodes(nodes 5 and 6).

Node 5:

- Option 1: 0 -> 1 -> 3 -> 5(7 + 15 = 22)
- Option 2: 0 -> 1 -> 3 -> 4 -> 5(17 + 6 = 23)
- Option 3: 0 -> 1 -> 3 -> 4 -> 6 -> 5(17 + 2 + 6 = 25) We choose 22.

Node 6 0 -> 1 -> 3 -> 4 -> 6(17 + 2 = 19)



NODE	DISTANCE
0	0
1	2*
2	6*
3	7*
4	17*
5	22*
6	19*

{0,1,2,3,4,5,6}