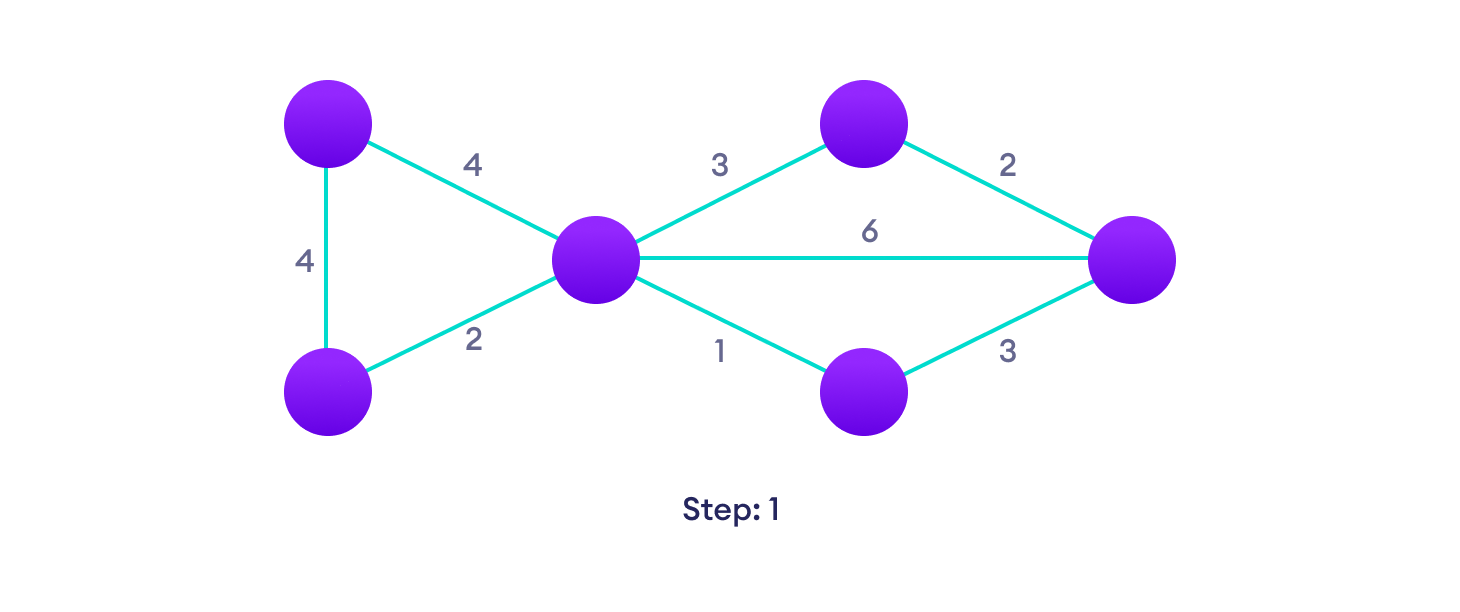
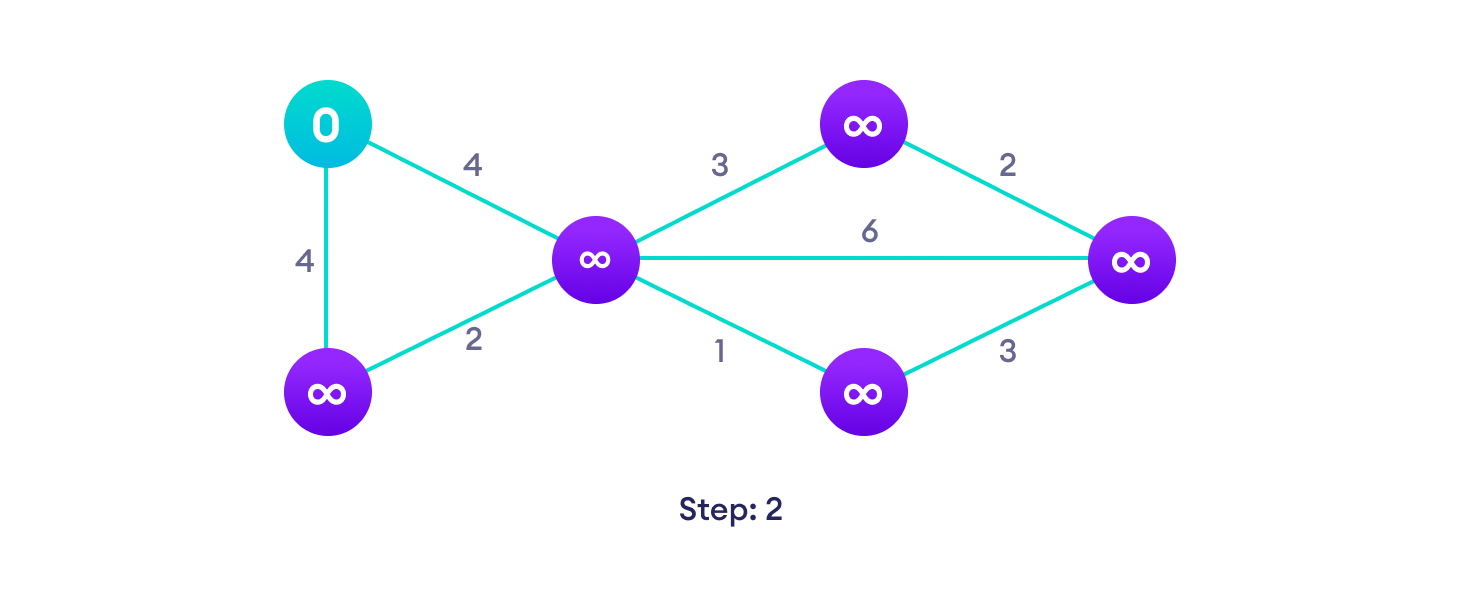
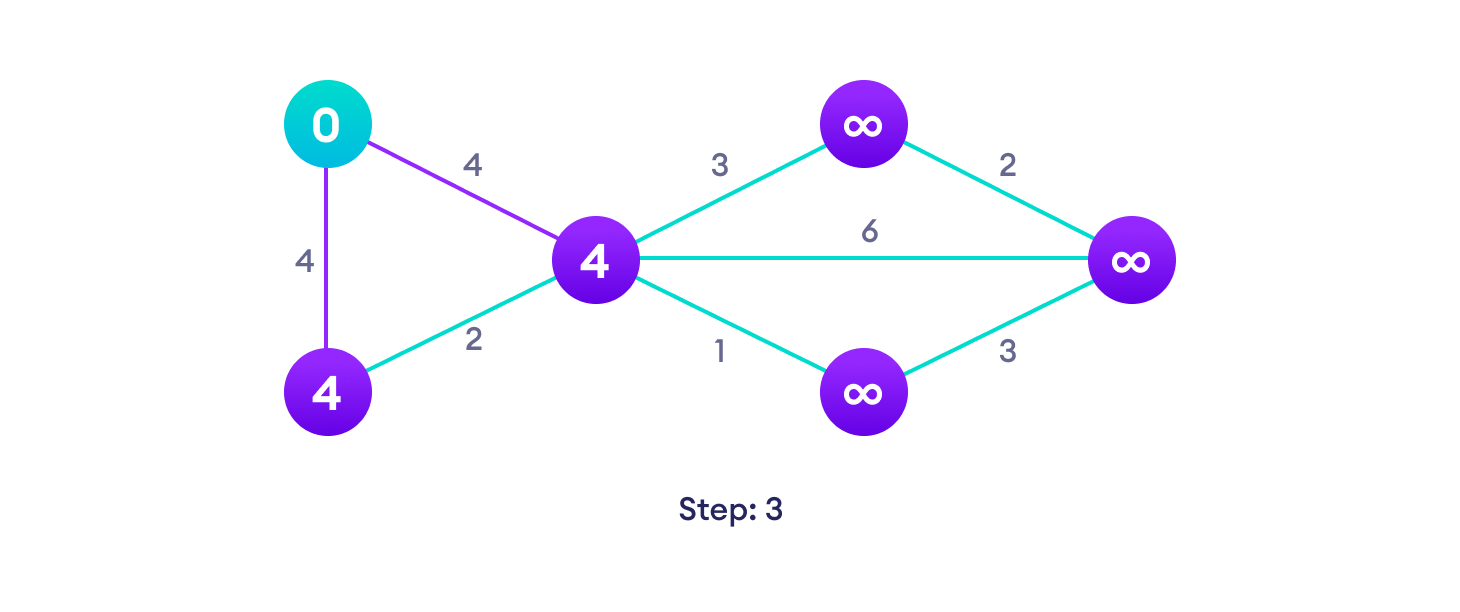
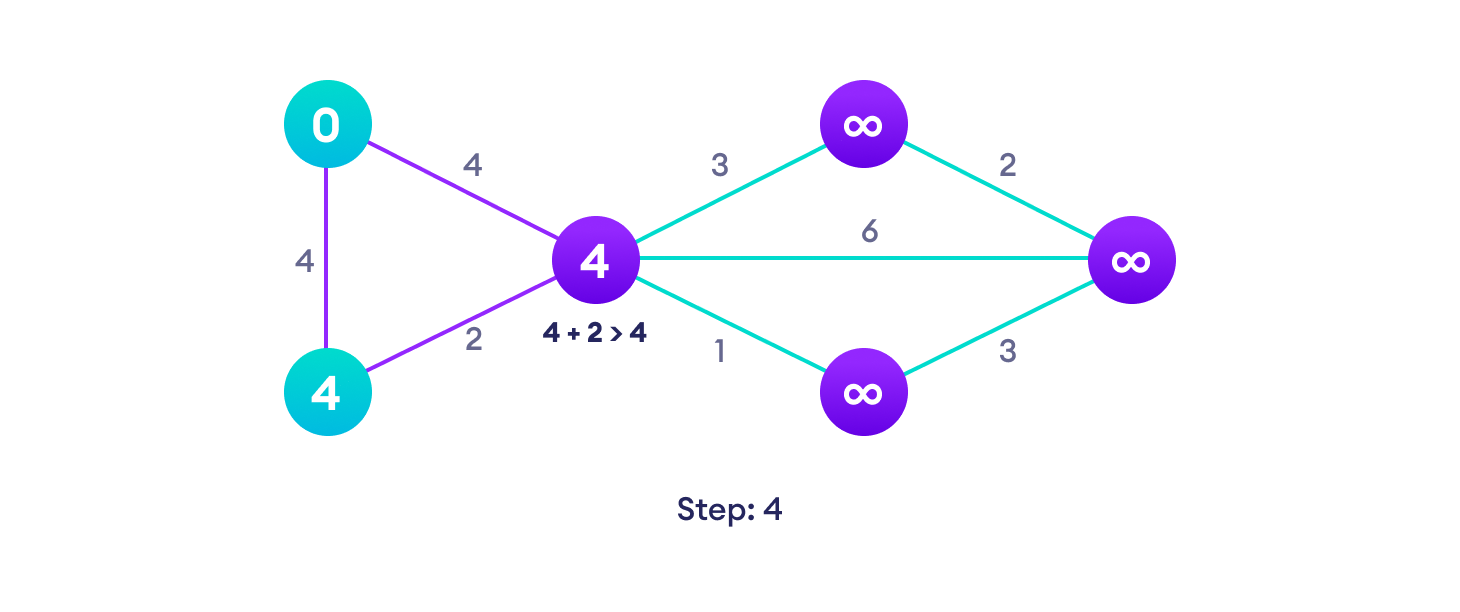
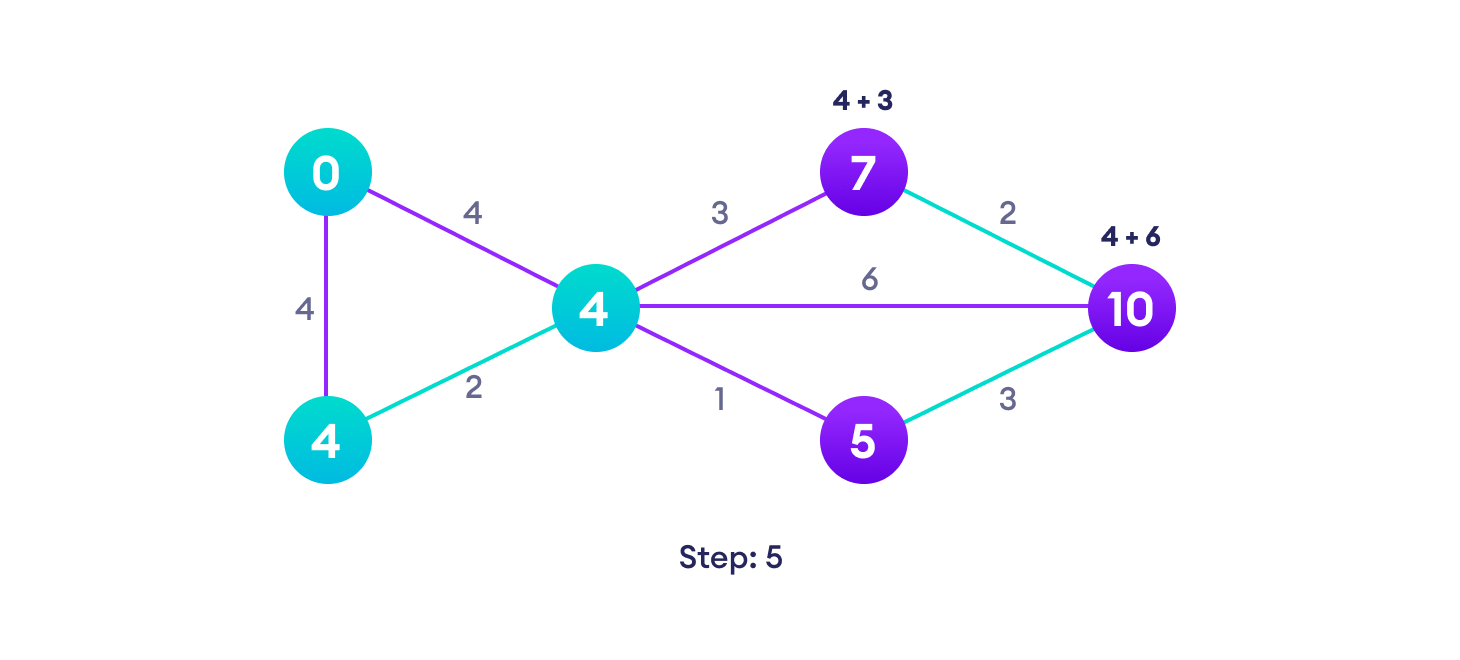
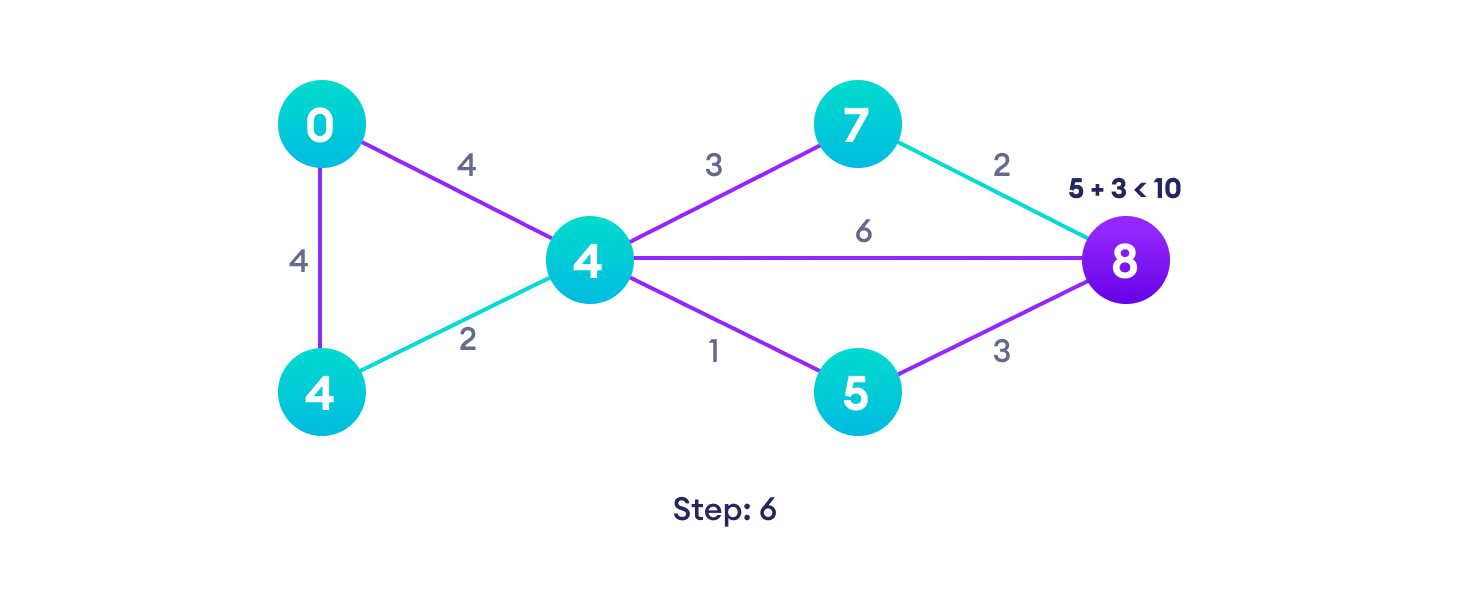
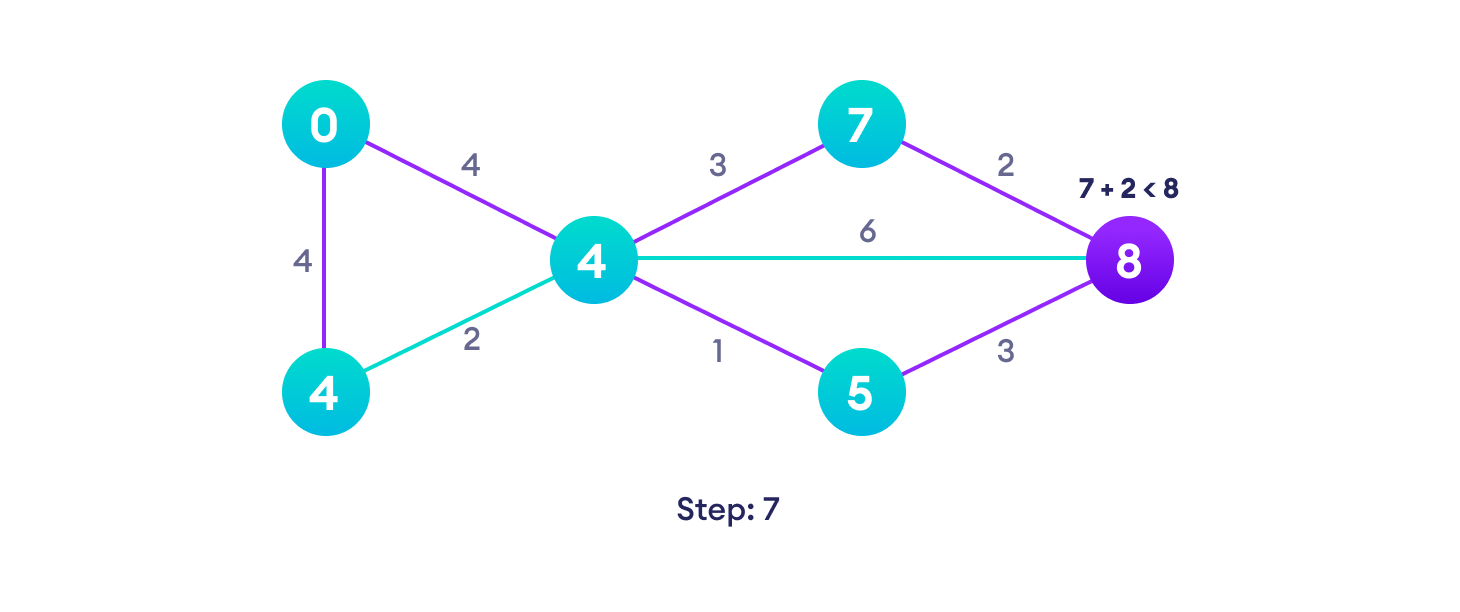
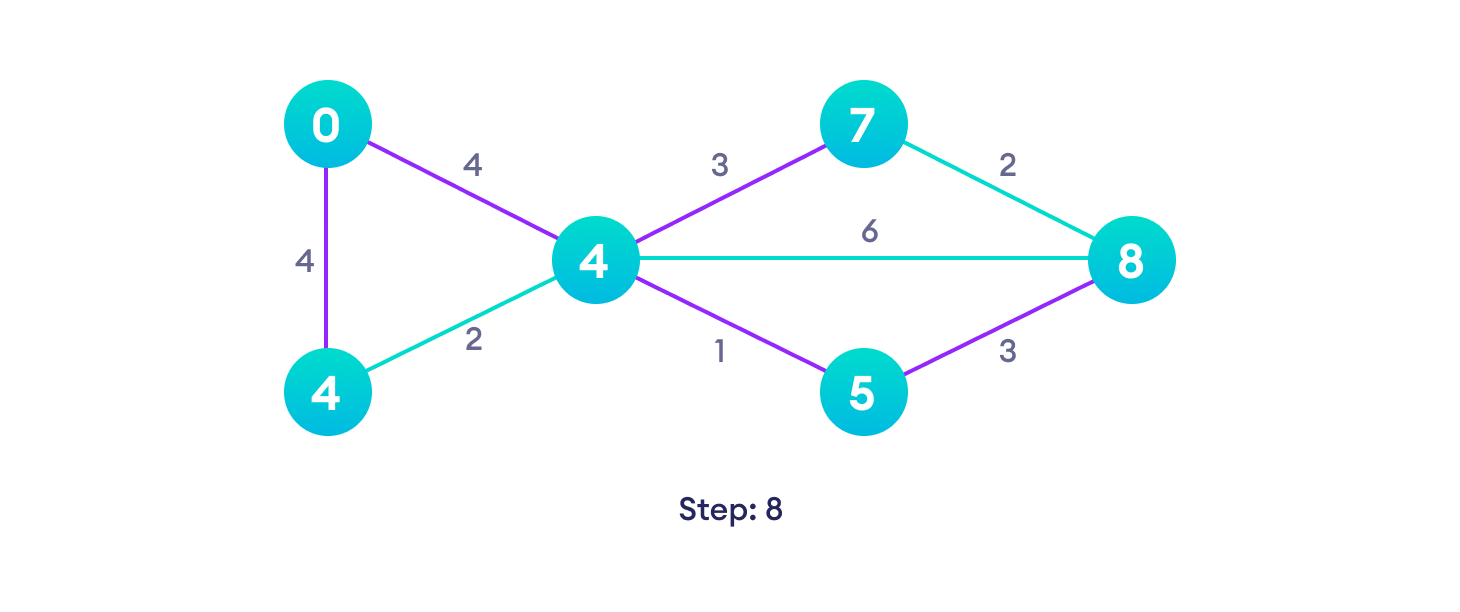
# **Dijkstra's Algorithm**

Dijkstra's algorithm aims to find the shortest path in a directed or undirected graph with non-negative edge weights.

It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

## **Example of Dijkstra's algorithm**

It is easier to start with an example and then think about the algorithm.

Start with a weighted graphChoose a starting vertex and assign infinity path values to all other devicesGo to each vertex and update its path lengthIf the path length of the adjacent vertex is lesser than new path length, don't update itAvoid updating path lengths of already visited verticesAfter each iteration, we pick the unvisited vertex with the least path length. So we choose 5 before 7Notice how the rightmost vertex has its path length updated twiceRepeat until all the vertices have been visited

## **Djikstra's algorithm pseudocode**

We need to maintain the path distance of every vertex. We can store that in an array of size v, where v is the number of vertices.

We also want to be able to get the shortest path, not only know the length of the shortest path. For this, we map each vertex to the vertex that last updated its path length.

Once the algorithm is over, we can backtrack from the destination vertex to the source vertex to find the path.

A minimum priority queue can be used to efficiently receive the vertex with the least path distance.

function dijkstra(G, S)

for each vertex V in G

distance[V] <- infinite

previous[V] <- NULL

If V != S, add V to Priority Queue Q

distance[S] <- 0

while Q IS NOT EMPTY

U <- Extract MIN from Q

for each unvisited neighbour V of U

tempDistance <- distance[U] + edge\_weight(U, V)

if tempDistance < distance[V]

distance[V] <- tempDistance

previous[V] <- U

return distance[], previous[]

## **Dijkstra's Algorithm Complexity**

Time Complexity: O(E log V)

where E is the number of edges and V is the number of vertices.

Space Complexity: O(V)

## **Dijkstra's Algorithm Applications**

* To find the shortest path
* In social networking applications
* In a telephone network
* To find the locations on the map
* Traffic information systems use Dijkstra’s Algorithm for tracking destinations from a given source location
* Open Source Path First (OSPF), an Internet-based routing protocol, uses Dijkstra’s Algorithm for finding best route from source router to other routers in the network
* It is used by Telephone and Cellular networks for routing management
* It is also used by Geographic Information System (GIS), such as Google Maps, for finding shortest path from point A to point B