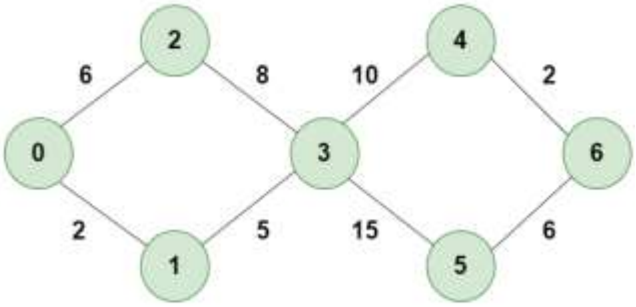
Dijkstra's Algorithm

The algorithm maintains a set of visited vertices and a set of unvisited vertices. It starts at the source vertex and iteratively selects the unvisited vertex with the smallest tentative distance from the source. It then visits the neighbors of this vertex and updates their tentative distances if a shorter path is found. This process continues until the destination vertex is reached, or all reachable vertices have been visited.

- **In a directed graph,** each edge has a direction, indicating the direction of travel between the vertices connected by the edge. In this case, the algorithm follows the direction of the edges when searching for the shortest path.
- **In an undirected graph,** the edges have no direction, and the algorithm can traverse both forward and backward along the edges when searching for the shortest path.

Algorithm for Dijkstra's Algorithm:

1. Mark the source node with a current distance of 0 and the rest with infinity.
2. Set the non-visited node with the smallest current distance as the current node.
3. For each neighbor, N of the current node adds the current distance of the adjacent node with the weight of the edge connecting 0->1. If it is smaller than the current distance of Node, set it as the new current distance of N.
4. Mark the current node 1 as visited.
5. Go to step 2 if there are any nodes are unvisited.



For this graph, we will assume that the weight of the edges represents the distance between two nodes.

As, we can see we have the shortest path from,
Node 0 to Node 1, from
Node 0 to Node 2, from
Node 0 to Node 3, from
Node 0 to Node 4, from
Node 0 to Node 6.

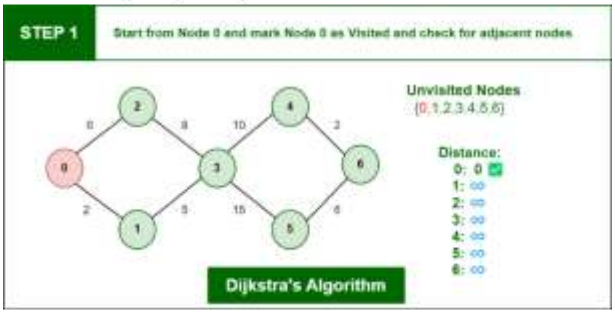
Initially we have a set of resources given below :

- *The Distance from the source node to itself is 0. In this example the source node is 0.*
- *The distance from the source node to all other node is unknown so we mark all of them as infinity.*

Example: 0 -> 0, 1-> ∞,2-> ∞,3-> ∞,4-> ∞,5-> ∞,6-> ∞.

- *we'll also have an array of unvisited elements that will keep track of unvisited or unmarked Nodes.*
- *Algorithm will complete when all the nodes marked as visited and the distance between them added to the path.* ***Unvisited Nodes:- 0 1 2 3 4 5 6.***

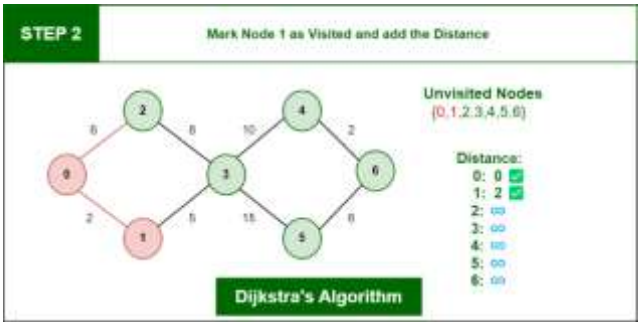
Step 1: *Start from Node 0 and mark Node as visited as you can check in below image visited Node is marked red.*



Dijkstra's Algorithm

Step 2: *Check for adjacent Nodes, Now we have to choices (Either choose Node1 with distance 2 or either choose Node 2 with distance 6) and choose Node with minimum distance. In this step* ***Node 1*** *is Minimum distance adjacent Node, so marked it as visited and add up the distance.*

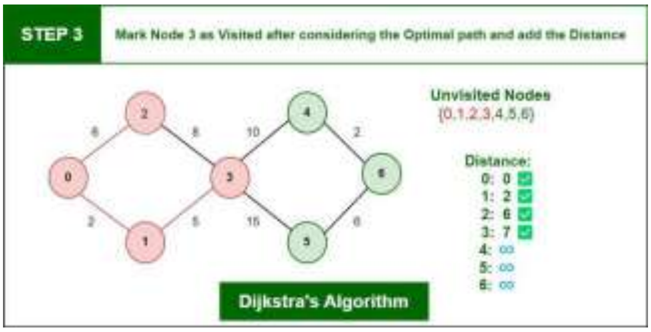
Distance: Node 0 -> Node 1 = 2



Dijkstra's Algorithm

Step 3: *Then Move Forward and check for adjacent Node which is Node 3, so marked it as visited and add up the distance, Now the distance will be:*

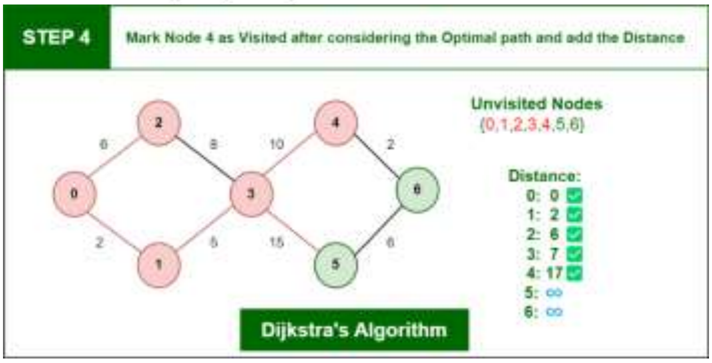
Distance: Node 0 -> Node 1 -> Node 3 = 2 + 5 = 7



Dijkstra's Algorithm

Step 4: *Again we have two choices for adjacent Nodes (Either we can choose Node 4 with distance 10 or either we can choose Node 5 with distance 15) so choose Node with minimum distance. In this step* ***Node 4*** *is Minimum distance adjacent Node, so marked it as visited and add up the distance.*

Distance: Node 0 -> Node 1 -> Node 3 -> Node 4 = 2 + 5 + 10 = 17


STEP 4

Mark Node 4 as Visited after considering the Optimal path and add the Distance

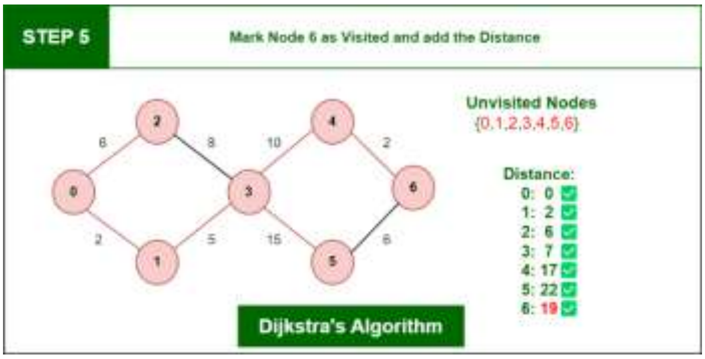
Unvisited Nodes
{0,1,2,3,4,5,6}

Distance:
0: 0 ✓
1: 2 ✓
2: 6 ✓
3: 7 ✓
4: 17 ✓
5: ∞
6: ∞

Dijkstra's Algorithm

Dijkstra's Algorithm

Step 5: *Again, Move Forward and check for adjacent Node which is* ***Node 6****, so marked it as visited and add up the distance, Now the distance will be:*
Distance: Node 0 -> Node 1 -> Node 3 -> Node 4 -> Node 6 = 2 + 5 + 10 + 2 = 19


STEP 5

Mark Node 6 as Visited and add the Distance

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

Distance:
0: 0 ✓
1: 2 ✓
2: 6 ✓
3: 7 ✓
4: 17 ✓
5: 22 ✓
6: 19 ✓

Dijkstra's Algorithm

Dijkstra's Algorithm

So, the Shortest Distance from the Source Vertex is 19 which is optimal one