## K. K. Wagh Institute of Engineering Education and Research, Nashik. Department of Computer Engineering

**Academic Year 2022-23**

|  |  |  |
| --- | --- | --- |
| **Course Name:** Laboratory Practice-III |  | **Course Code:** 410246 |
| **Class:** BE |  | **Div:** B |
| **Name of Students:** Pranita Sonawane | (61) |  |
| Anagha Kulkarni | (62) |  |
| Gauri Pawar | (63) |  |
| Nandini Marke | (64) |  |
| **Name of Faculty:** Prof. K.P.Birla |  |  |

# Mini-Project Report

**Title of Mini-Project:** - Implement Dijkstras Algorithm to find out the shortest path distance between any two vertices of a graph.

**Problem Statement:** Implement Dijkstras Algorithm to find out the shortest path distance between any two vertices of a graph.

## Objective:

1. Learn how to implement algorithms that follow greedy approach to find out the shortest path distance.

## Introduction of Mini-Project:

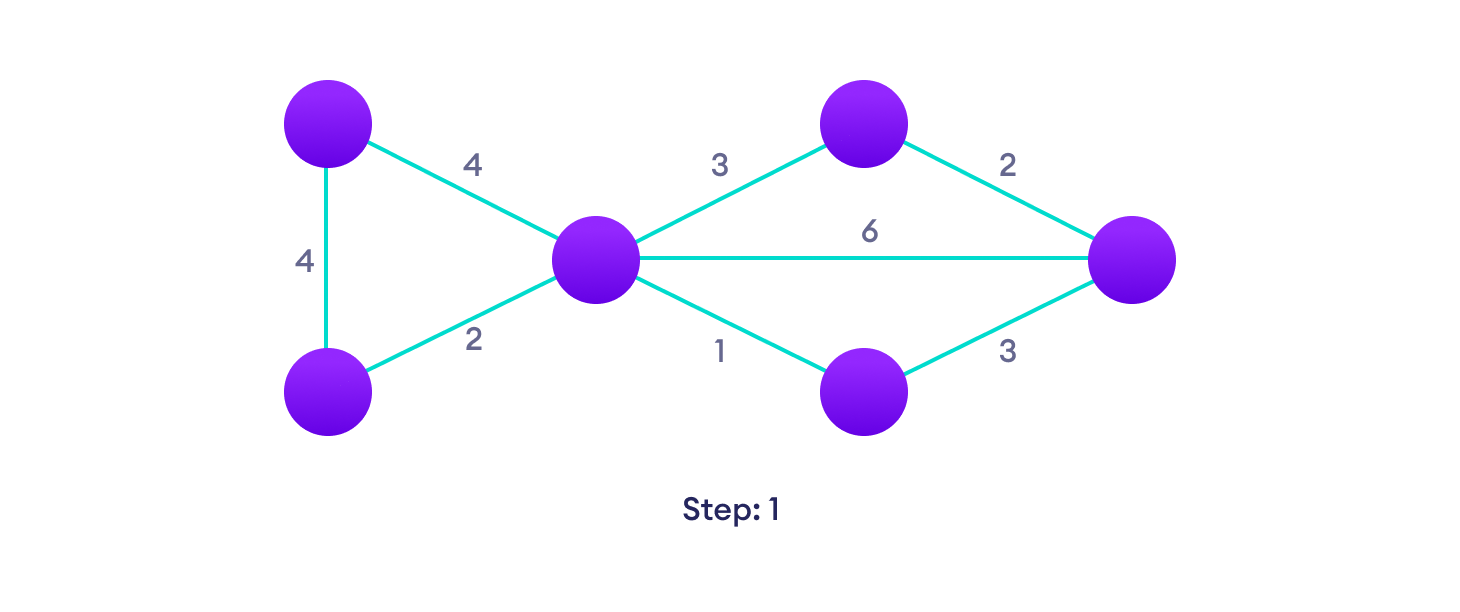
Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph. It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

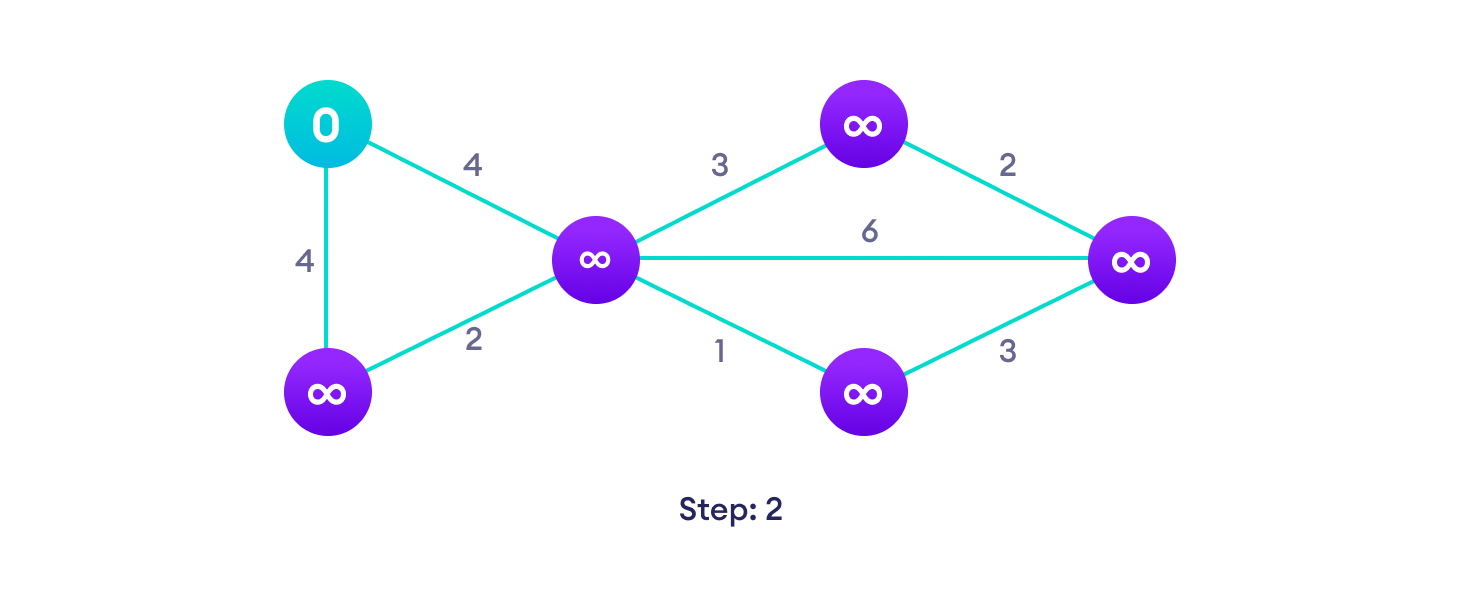
## Djikstras Algorithm Working Process:

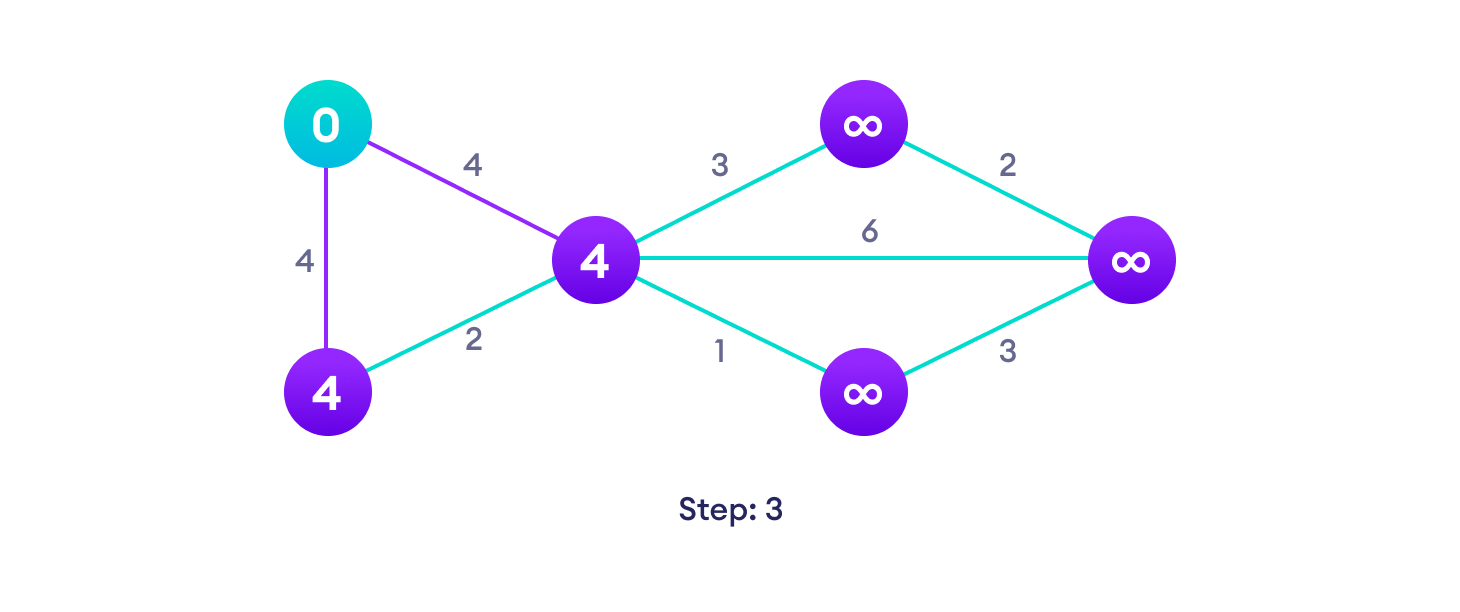
Djikstra's Algorithm works on the basis that any sub path B -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D.

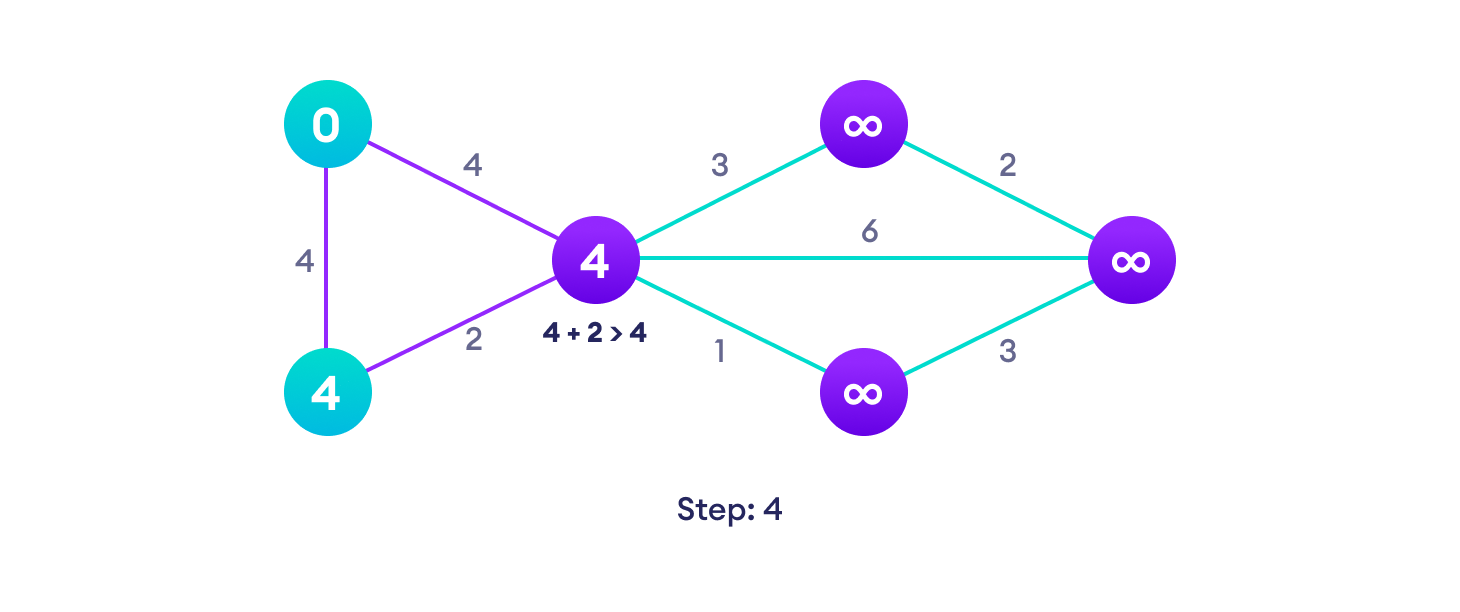
Djikstra used this property in the opposite direction i.e we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbors to find the shortest subpath to those neighbours algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

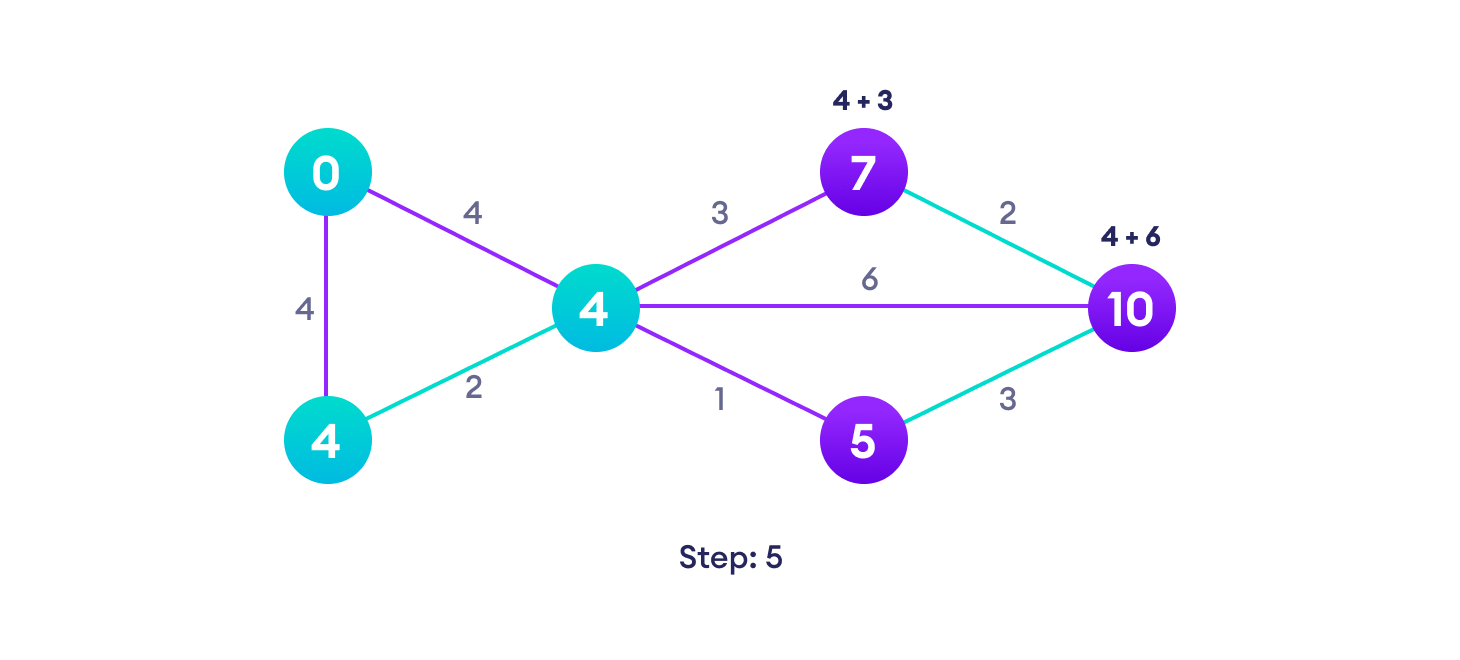
## Illustration:

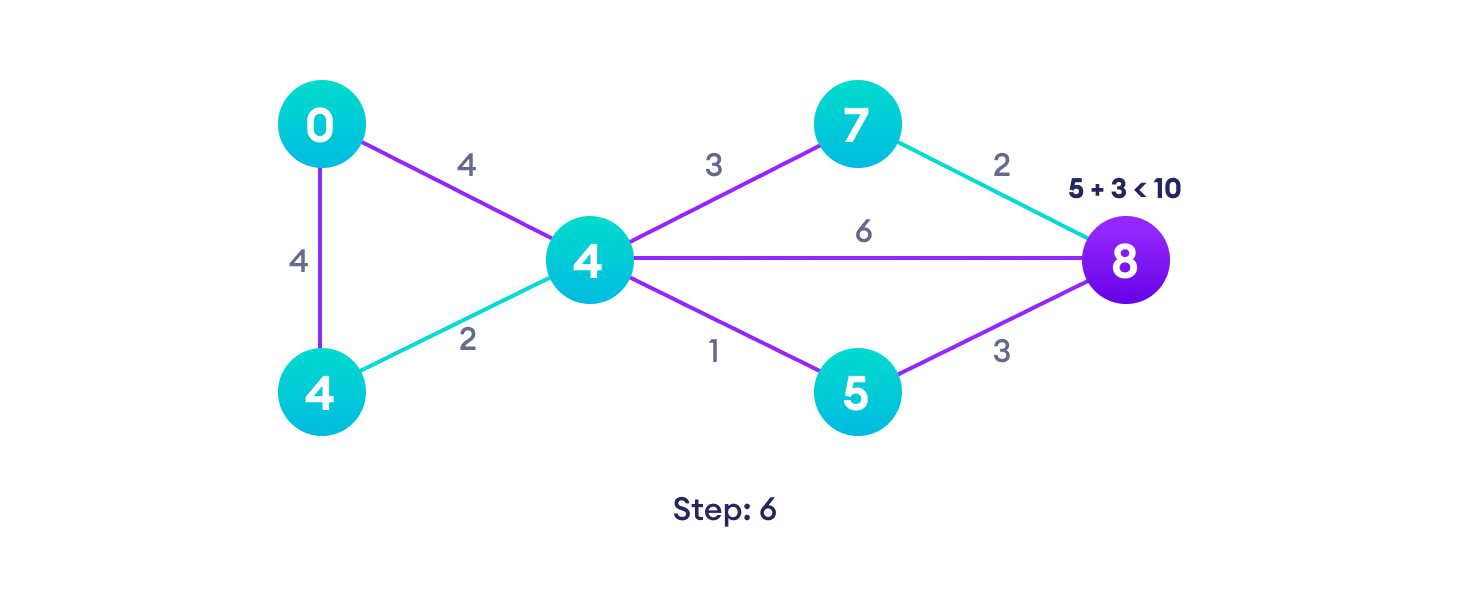


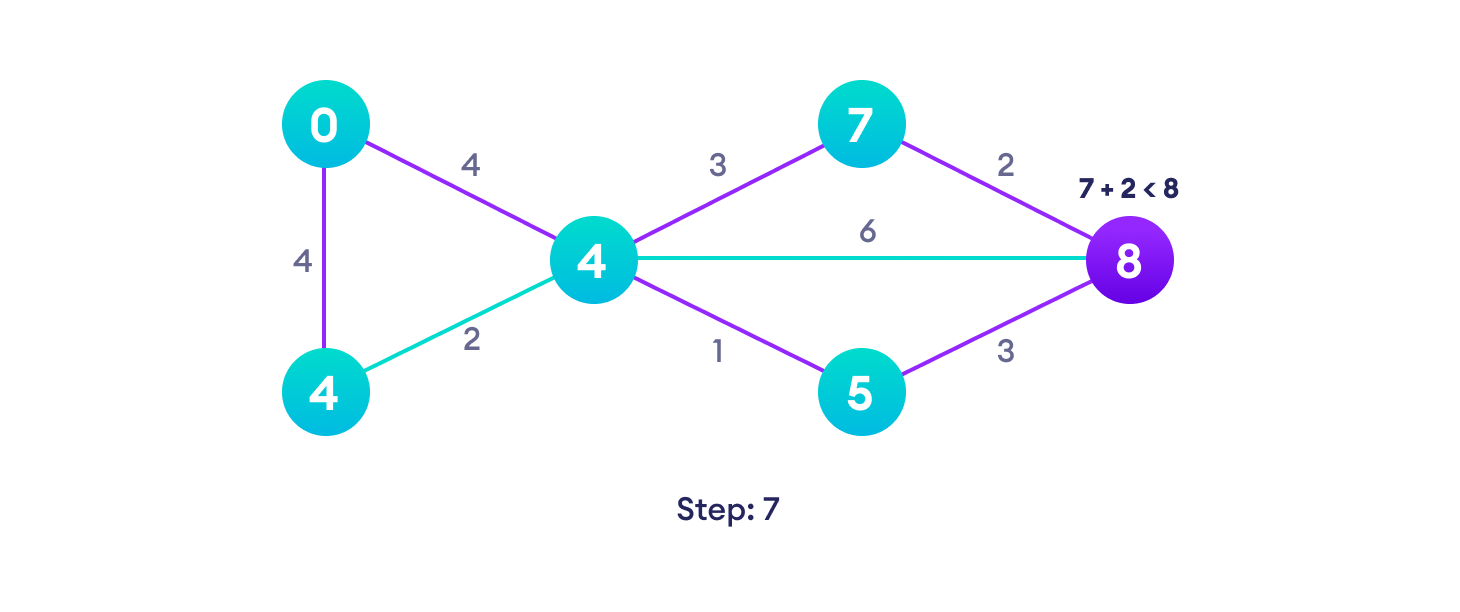


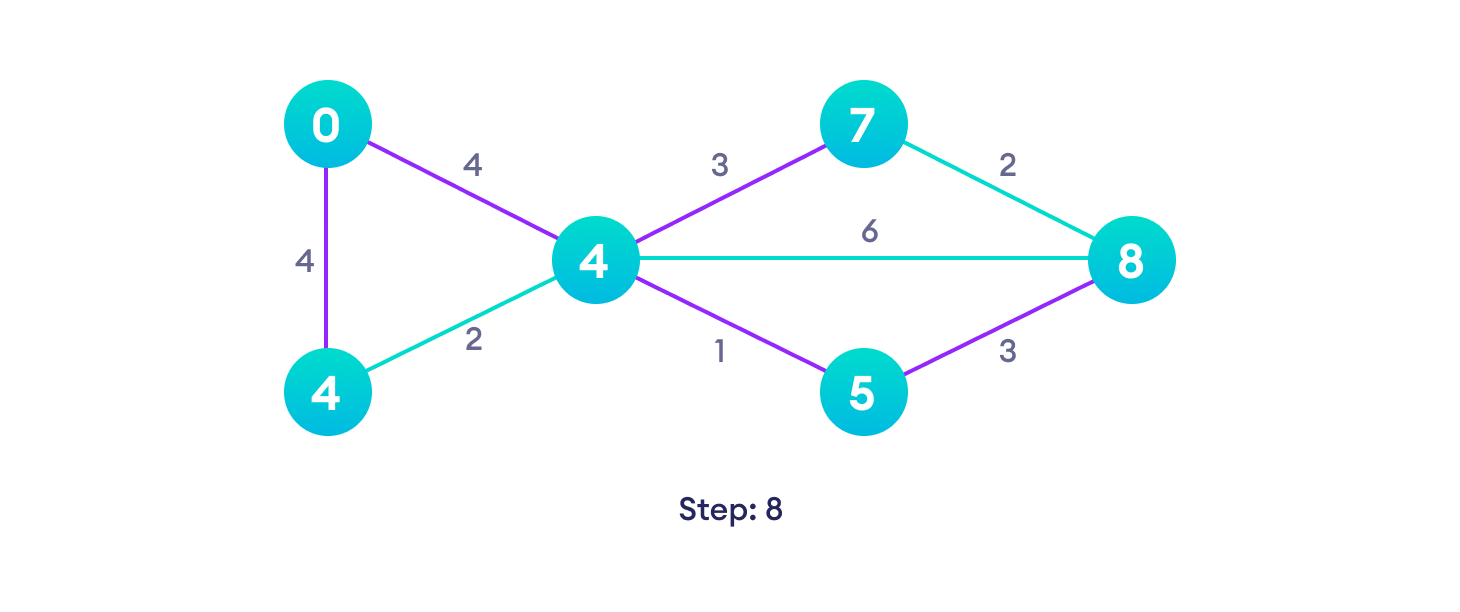












## Algorithm:

Step 1: Start at the end location. This will be the current node. Mark this point as zero using brackets.

Step 2: Add all the weighted information on the nodes that connect to this point. Mark each node with an assigned number (the sum of the prior node connecting to it and the weighted value on the segment). To mark, use brackets and the sum inside the frame.

Step 3: Put an X on the prior node. After adding its value to the newer nodes, there is no more use for it, and the node can be marked X and not be compared to other nodes in the future steps.

Step 4: Check all the nodes connected to the current node and find the smallest value. The smallest value node becomes the new current node. Go back to step 2 again.

Step 5: Once the "start" point has been reached and marked, trace the path that ended in this node, and this is the shortest path.

**Program for Djikstra Algorithm**

using namespace std;

#include <limits.h>

#define#include <iostream>

int minDistance(int dist[], bool sptSet[])

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v]<=min)

min = dist[v], min\_index =v;

return min\_index;

}

void printSolution(int dist[])

{

cout << "Vertex \t Distance from Source" << endl;

for (int i = 0; i < V; i++)

cout << i << " \t\t\t\t" << dist[i] << endl;

}

void dijkstra(int graph[V][V], int src)

{

int dist[V];

bool sptSet[V];

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = false;

dist[src] = 0;

for (int count = 0; count < V - 1; count++) {

int u = minDistance(dist,sptSet);

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v]

&& dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist);

}

int main()

{

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

dijkstra(graph, 0);

return 0;

}

## Output:

Vvertex Distance from Source Distance from Source

1. 0
2. 4
3. 12
4. 19
5. 21
6. 11
7. 9
8. 8
9. 14 0

1

**Time Complexity** : O(E log V)

# Conclusion

Hence, Here we Implement. Dijkstras Algorithm to find out the shortest path distance between any two vertices of a graph.