Bansilal Ramnath Agarwal Charitable Trust's
VISHWAKARMA INSTITUTE OF TECHNOLOGY – PUNE

COMPUTER NETWORKS

Branch	CSE - AIML
Division	A
Batch	2
GR-no	12311305
Roll no	04
Name	Adwyte Karandikar

Experiment No. 2:

TITLE: Routing in the Internet

PROBLEM STATEMENT:

Write a program to find the shortest path using Dijkstra Equation for Link State Routing Protocol which is used by Open Shortest Path First Protocol (OSPF) in the Internet for the network flow provided by instructor.

THEORY:

The Link State Routing Protocol is a type of routing protocol used in computer networks to determine the shortest path from one node to another. It is based on the concept of each router or node in the network maintaining a database of its neighbors and the cost or distance to reach them. Dijkstra's algorithm is commonly employed to compute the shortest path in Link State Routing

Protocol. Below is a write-up explaining how Dijkstra's algorithm is used to find the shortest path in a network.

Overview of Dijkstra’s Algorithm:

Dijkstra's algorithm is a graph search algorithm that efficiently finds the shortest path from a source node to all other nodes in a weighted graph. It starts by initializing distances to all nodes as infinity, except for the source node, which is set to zero. Then, it iteratively selects the node with the smallest distance from the source and updates the distances to its neighboring nodes. The process continues until all nodes have been visited.

Steps to Find the Shortest Path:

- **Initialization:** Initialize a distance vector for each node, setting the distance to the source node as zero and all other distances as infinity. Also, maintain a set of unvisited nodes.

- **Selecting the Nearest Node:** At each iteration, select the node with the smallest distance from the source node among the unvisited nodes.

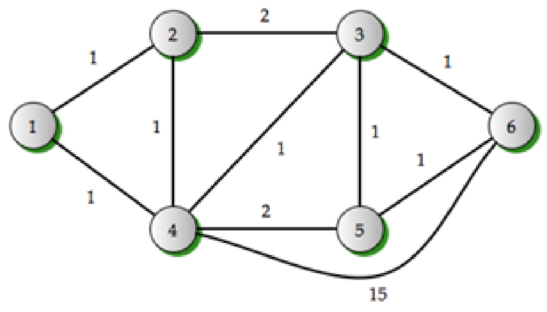
- **Updating Distances:** For the selected node, update the distances to its neighboring nodes if the distance through the selected node is shorter than the current distance. This involves comparing the sum of the distance to the selected node and the distance from the selected node to its neighbor with the current distance to the neighbor.

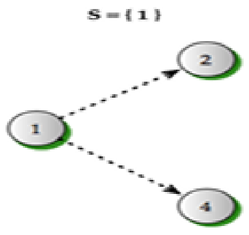
- **Marking Visited Nodes:** Mark the selected node as visited and remove it from the set of unvisited nodes.

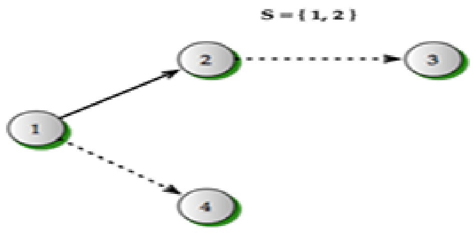
- **Repeat:** Repeat steps 2 to 4 until all nodes have been visited.

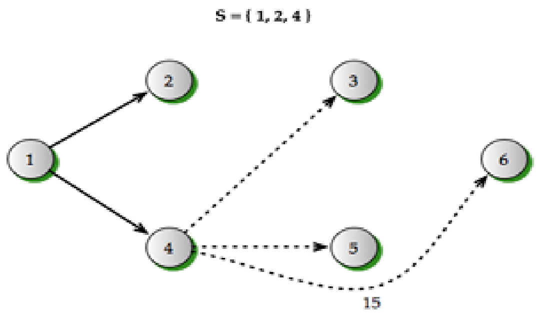
- **Shortest Path Tree:** Once the algorithm is complete, a shortest path tree is constructed, with the source node as the root and edges representing the shortest paths to each node.

Example Walkthrough:









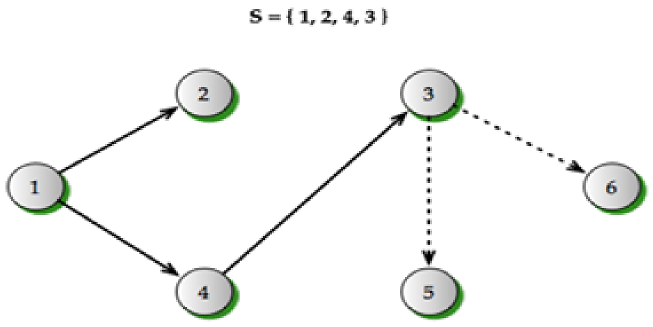
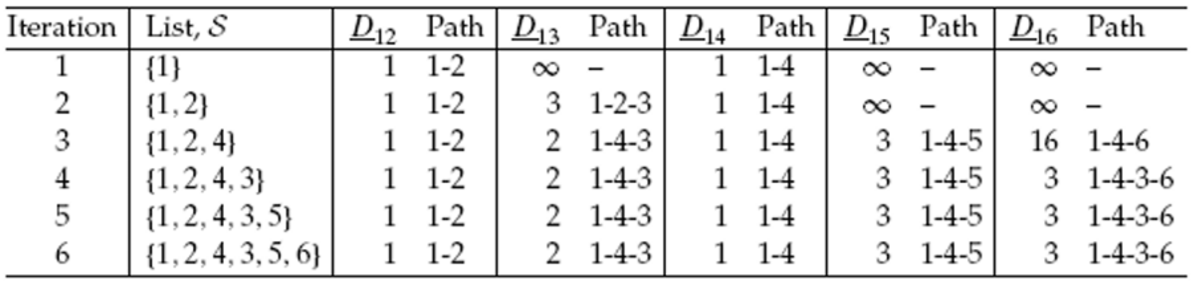


TABLE: Iterative steps in Dijkstra's Algorithm.



Iteration	List, $\mathcal{S}$	$\underline{D}_{12}$	Path	$\underline{D}_{13}$	Path	$\underline{D}_{14}$	Path	$\underline{D}_{15}$	Path	$\underline{D}_{16}$	Path
1	{1}	1	1-2	∞	–	1	1-4	∞	–	∞	–
2	{1, 2}	1	1-2	3	1-2-3	1	1-4	∞	–	∞	–
3	{1, 2, 4}	1	1-2	2	1-4-3	1	1-4	3	1-4-5	16	1-4-6
4	{1, 2, 4, 3}	1	1-2	2	1-4-3	1	1-4	3	1-4-5	3	1-4-3-6
5	{1, 2, 4, 3, 5}	1	1-2	2	1-4-3	1	1-4	3	1-4-5	3	1-4-3-6
6	{1, 2, 4, 3, 5, 6}	1	1-2	2	1-4-3	1	1-4	3	1-4-5	3	1-4-3-6

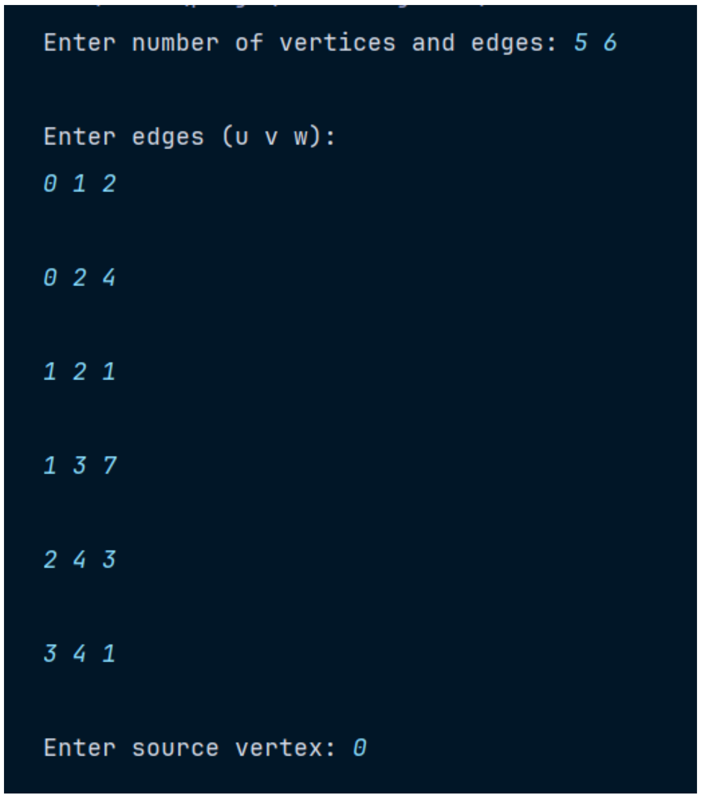
Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;

// Function to implement Dijkstra's Algorithm
void dijkstra(int V, vector<vector<pair<int, int>>> &adj, int src) {
    // Distance vector to store shortest distance from src to each vertex
    vector<int> dist(V, INT_MAX);

    // Min-heap priority queue: {distance, vertex}
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
int>>> pq;

    // Distance to the source is 0
    dist[src] = 0;
    pq.push({0, src});

    while (!pq.empty()) {
        int u = pq.top().second; // Current vertex
        int d = pq.top().first;  // Current distance
        pq.pop();

        // If current distance is greater than already found shortest, skip
        if (d > dist[u]) continue;

        // Traverse all neighbors of u
        for (auto &edge : adj[u]) {
            int v = edge.first;   // Neighbor vertex
```

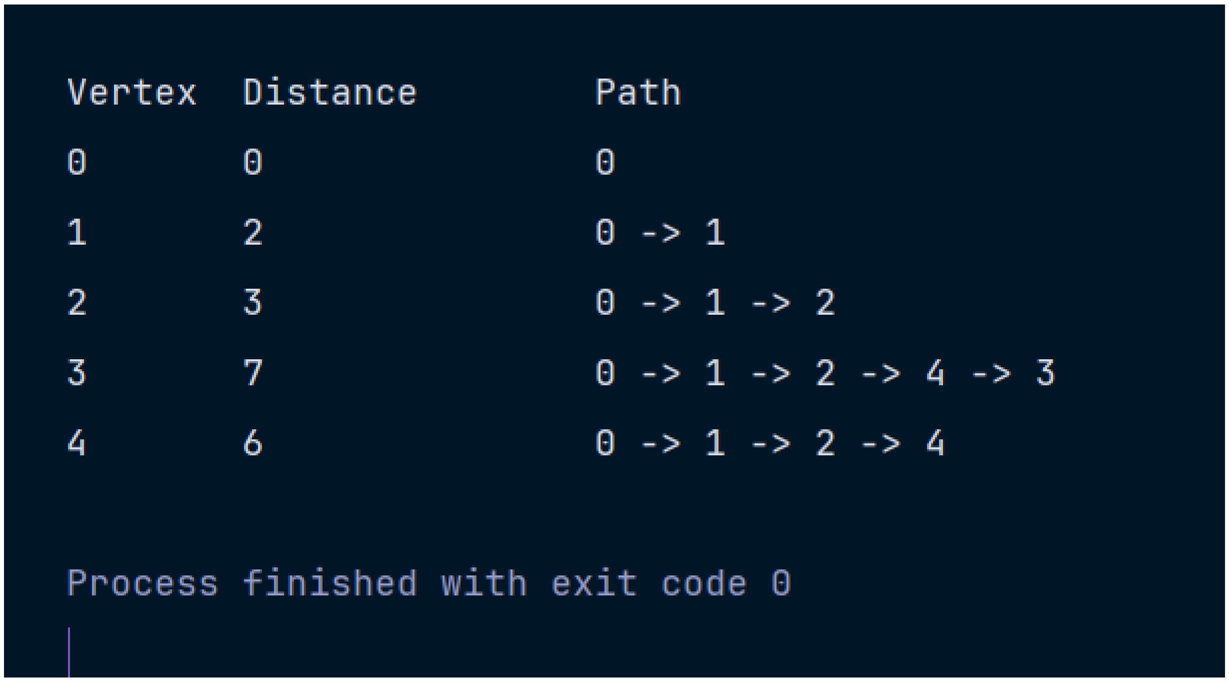
```cpp
            int w = edge.second;  // Weight of edge u -> v

            // Relaxation step
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }

    // Print shortest distances
    cout << "Vertex\tDistance from Source " << src << endl;
    for (int i = 0; i < V; i++) {
        cout << i << "\t" << dist[i] << endl;
    }
}

int main() {
    int V, E;
    cout << "Enter number of vertices and edges: ";
    cin >> V >> E;

    vector<vector<pair<int, int>>> adj(V); // adjacency list

    cout << "\nEnter edges (u v w):" << endl;
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w}); // For undirected graph
    }

    int src;
    cout << "\nEnter source vertex: ";
    cin >> src;
    cout << "\n";


    dijkstra(V, adj, src);

    return 0;
}
```

Screenshots/Output:

Ex. 1)

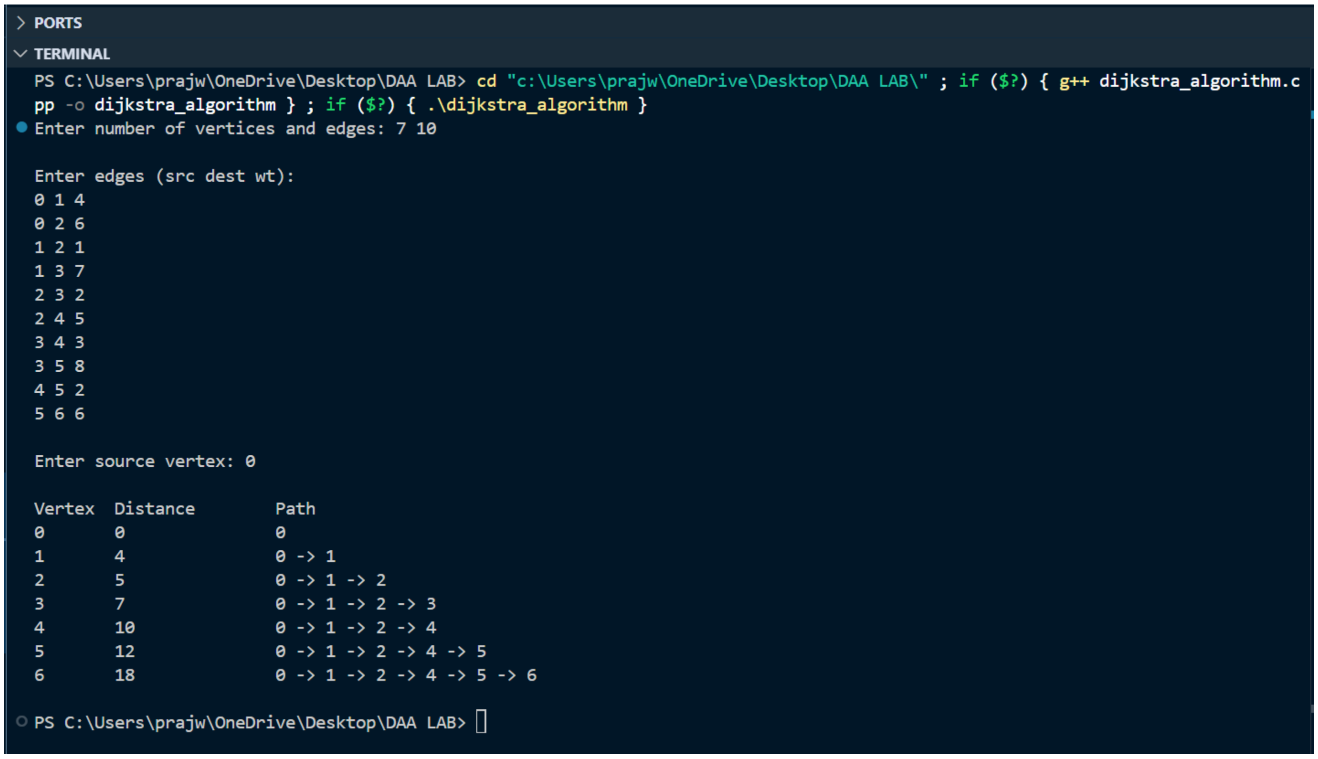


```
Enter number of vertices and edges: 5 6

Enter edges (u v w):
0 1 2

0 2 4

1 2 1

1 3 7

2 4 3

3 4 1

Enter source vertex: 0
```



```
Vertex  Distance        Path
0       0               0
1       2               0 -> 1
2       3               0 -> 1 -> 2
3       7               0 -> 1 -> 2 -> 4 -> 3
4       6               0 -> 1 -> 2 -> 4

Process finished with exit code 0
```

Ex. 2)



```
PS C:\Users\prajw\OneDrive\Desktop\DAA LAB> cd "c:\Users\prajw\OneDrive\Desktop\DAA LAB\" ; if ($?) { g++ dijkstra_algorithm.c
pp -o dijkstra_algorithm } ; if ($?) { .\dijkstra_algorithm }
Enter number of vertices and edges: 7 10

Enter edges (src dest wt):
0 1 4
0 2 6
1 2 1
1 3 7
2 3 2
2 4 5
3 4 3
3 5 8
4 5 2
5 6 6

Enter source vertex: 0

Vertex  Distance        Path
0       0               0
1       4               0 -> 1
2       5               0 -> 1 -> 2
3       7               0 -> 1 -> 2 -> 3
4       10              0 -> 1 -> 2 -> 4
5       12              0 -> 1 -> 2 -> 4 -> 5
6       18              0 -> 1 -> 2 -> 4 -> 5 -> 6

PS C:\Users\prajw\OneDrive\Desktop\DAA LAB>
```

Application in Link State Routing Protocol:
In Link State Routing Protocol, each router exchanges information about its directly connected links with other routers in the network. This information includes the link cost or distance. Using this information, Dijkstra's algorithm is applied at each router to compute the shortest path to all other routers in the network.

Conclusion:
Dijkstra's algorithm plays a crucial role in the Link State Routing Protocol by enabling routers to compute the shortest paths efficiently. By maintaining a database of network topology information and applying Dijkstra's algorithm, routers can make informed routing decisions, leading to efficient and reliable communication within the network.