EXPERIMENT 9

OBJECTIVE:

Implement Dijkstra's Algorithm:

Dijkstra's Algorithm. Dijkstra's Algorithm is a Graph algorithm that finds the shortest path from a source vertex to all other vertices in the Graph (single source shortest path). It is a type of Greedy Algorithm that only works on Weighted Graphs having positive weights.

The following are the basic concepts of Dijkstra's Algorithm:

1. Dijkstra's Algorithm begins at the node we select (the source node), and it examines the graph to find the shortest path between that node and all the other nodes in the graph.
2. The Algorithm keeps records of the presently acknowledged shortest distance from each node to the source node, and it updates these values if it finds any shorter path.
3. Once the Algorithm has retrieved the shortest path between the source and another node, that node is marked as 'visited' and included in the path.
4. The procedure continues until all the nodes in the graph have been included in the path. In this manner, we have a path connecting the source node to all other nodes, following the shortest possible path to reach each node.

Pseudocode for Dijkstra's Algorithm.

1. We have to maintain a record of the path distance of every node. Therefore, we can store the path distance of each node in an array of size n, where n is the total number of nodes.
2. Moreover, we want to retrieve the shortest path along with the length of that path. To overcome this problem, we will map each node to the node that last updated its path length.
3. Once the algorithm is complete, we can backtrack the destination node to the source node to retrieve the path.
4. We can use a minimum Priority Queue to retrieve the node with the least path distance in an efficient way.

```
#include <stdio.h>
#include <limits.h>

#define V 5  // Number of vertices in the graph

// Function to find the vertex with the minimum distance value
intminDistance(intdist[], intsptSet[]) {
int min = INT_MAX, min_index;

for (int v = 0; v < V; v++)
if (sptSet[v] == 0 &&dist[v] <= min)
min = dist[v], min_index = v;

returnmin_index;
}

// Function to print the shortest path from the source to all vertices
voidprintSolution(intdist[]) {
printf("Vertex \t Distance from Source\n");
for (int i = 0; i < V; i++)
printf("%d \t\t %d\n", i, dist[i]);
}

// Function implementing Dijkstra's Algorithm
voiddijkstra(int graph[V][V], intsrc) {
intdist[V];    // Array to hold the shortest distance from the source
intsptSet[V];  // Array to track vertices included in shortest path tree
```

```
    // Initialize all distances as infinite and sptSet[] as false
for (int i = 0; i < V; i++)
dist[i] = INT_MAX, sptSet[i] = 0;

    // Distance from the source to itself is always 0
dist[src] = 0;

    // Find the shortest path for all vertices
for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not yet processed
int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
sptSet[u] = 1;

        // Update dist[] for adjacent vertices of the picked vertex
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];
    }

    // Print the constructed distance array
printSolution(dist);
}

int main() {
    // Example graph represented as an adjacency matrix
int graph[V][V] = { {0, 10, 0, 0, 5},
                        {0, 0, 1, 0, 2},
                        {0, 0, 0, 4, 0},
                        {7, 0, 6, 0, 0},
                        {0, 3, 9, 2, 0} };

int source = 0;  // Starting node
dijkstra(graph, source);

return 0;
}
```

OUTPUT

```
Vertex   Distance from Source
0        0
1        8
2        9
3        7
4        5
```