Experiment no.06

Name: Shivani Kolekar

Roll no. : 24141031

Batch no.: I2

Aim: To implement Dijkstra's algorithm to find the shortest path from a source vertex to all other vertices in a graph.

Theory:

What is Dijkstra's Algorithm?

Dijkstra's Algorithm is a greedy algorithm used to find the shortest path from a source node to all other nodes in a weighted graph with non-negative edge weights.

It was proposed by Edsger W. Dijkstra in 1956.

Applications:

- **GPS Navigation** systems
- **Network routing protocols** (e.g., OSPF)
- **Game AI** for pathfinding
- **Shortest route in maps**

Terminologies

- **Vertex (Node):** A point in the graph.
- **Edge:** A connection between two vertices.
- **Weight:** The cost of moving from one vertex to another.
- **Source Vertex:** The starting point.
- **Distance Array:** Stores the shortest distance from the source to every vertex.
- **Visited Set:** Keeps track of vertices whose shortest distance from the source is already known.

Basic Idea

1. Start with the **source vertex**, and set the distance to 0.
2. Set distances to all other vertices as **infinity (∞)**.
3. At each step, choose the **unvisited vertex** with the **smallest known distance**.

4. Update the distances of its **neighbors**.
5. Repeat until all vertices have been visited.

Pseudocode

```
function Dijkstra(Graph, source):
  dist[] = array of distances from source to all vertices, initialized to ∞
  dist[source] = 0
  visited[] = array to mark visited vertices, initialized to false
  for i from 1 to number of vertices:
    u = vertex with minimum dist[u] not visited
    visited[u] = true
    for each neighbor v of u:
      if not visited[v] and dist[u] + weight(u, v) < dist[v]:
        dist[v] = dist[u] + weight(u, v)
  return dist[]
```

Dry Run Example

Graph: Vertices: A, B, C, D, E

Edges: A-B: 4, A-C: 1

C-B: 2, B-E: 4

C-D: 4, D-E: 4

Final distances from A:

- A: 0
- B: 3
- C: 1
- D: 5
- E: 7

Adjacency Matrix

	A	**B**	**C**	**D**	**E**
A	0	4	1	∞	∞
B	4	0	2	∞	4
C	1	2	0	4	∞
D	∞	∞	4	0	4
E	∞	4	∞	4	0

Time Complexity

Let:

- **V** = number of vertices
- **E** = number of edges

Using an adjacency matrix:

- Time = $\mathbf{O(V^2)}$

Using a min-heap (priority queue) with an adjacency list:

- Time = **O((V + E) log V)** — more efficient for sparse graphs.

Code:

```
#include <stdio.h>
#include <limits.h>
#define V 10  // Maximum number of vertices (adjust as needed)
#define INF 99999
// Function to find the vertex with the minimum distance value
int minDistance(int dist[], int visited[], int n) {
   int min = INF, min_index = -1;
   for (int v = 0; v < n; v++) {
```

```
      if (!visited[v] && dist[v] <= min) {
         min = dist[v];
         min_index = v;
      }   }
   return min_index;
}
// Dijkstra's algorithm
void dijkstra(int graph[V][V], int n, int src) {
   int dist[V];      // Output array. dist[i] holds the shortest distance from src to i
   int visited[V];    // visited[i] is true if vertex i is included in the shortest path tre
   // Initialize all distances as INFINITE and visited[] as false
   for (int i = 0; i < n; i++) {
      dist[i] = INF;
      visited[i] = 0;
   }
   dist[src] = 0;  // Distance to source is 0
   // Find shortest path for all vertices
   for (int count = 0; count < n - 1; count++) {
      int u = minDistance(dist, visited, n);
      visited[u] = 1;
      // Update dist[v] if there is a shorter path through u
      for (int v = 0; v < n; v++) {
         if (!visited[v] && graph[u][v] && dist[u] != INF
            && dist[u] + graph[u][v] < dist[v]) {
            dist[v] = dist[u] + graph[u][v];
         }    }  }
   // Print the result
   printf("Vertex \t Distance from Source %d\n", src);
```

```
    for (int i = 0; i < n; i++)
        printf("%d \t\t %d\n", i, dist[i]);}
int main() {
    int n, src;
    int graph[V][V];
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix (use 0 if no edge):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
            if (graph[i][j] == 0 && i != j)
                graph[i][j] = INF;  // No direct edge
        }  }
    printf("Enter the source vertex (0 to %d): ", n - 1);
    scanf("%d", &src);
    dijkstra(graph, n, src);
    return 0;}
```

OUTPUT:-