# EXPERIMENT NO. 06

Roll No.:24141001
Batch: I1

**Title:** Shortest paths to other Vertices using Dijkstra's algorithm from a given vertex in a weighted connected graph.

**Objective:**

- Understand the concept of single-source shortest path in weighted graphs.
- Study and implement Dijkstra's Algorithm using C.
- Learn how greedy techniques help in optimizing path-finding problems.
- Compute the shortest distance from a given source vertex to all other vertices in a weighted connected graph.
- Analyze the algorithm's time complexity and understand its practical applications in real-world systems.

**Theory:**

In a weighted graph, the shortest path problem aims to find the path between two vertices such that the sum of edge weights is minimized. For graphs with non-negative weights, the most commonly used algorithm is Dijkstra's Algorithm.

It follows a greedy strategy, selecting the vertex with the smallest known distance at every step and updating the distances of its adjacent vertices. The algorithm guarantees the shortest path for all vertices from the given source.

**Algorithm:**

Dijkstra(Graph, source):

1. For each vertex v:

    dist[v] = ∞

    visited[v] = false

2. dist[source] = 0

3. Repeat (V - 1) times:

    Select the unvisited vertex u with minimum dist[u]

    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)

4. Print dist[] values

**Applications:**

Applications of Dijkstra's Algorithm

1. GPS & Navigation Systems

Finding the shortest route in maps.

2. Internet Routing (OSPF)

Routers compute least-cost communication paths.

3. Robotics & AI

Used in path planning for movement.

4. Transportation & Logistics

Optimizing delivery routes and travel cost.

5. Social Networks

Finding the shortest link path between users.

6. Gaming

NPC pathfinding and map navigation.

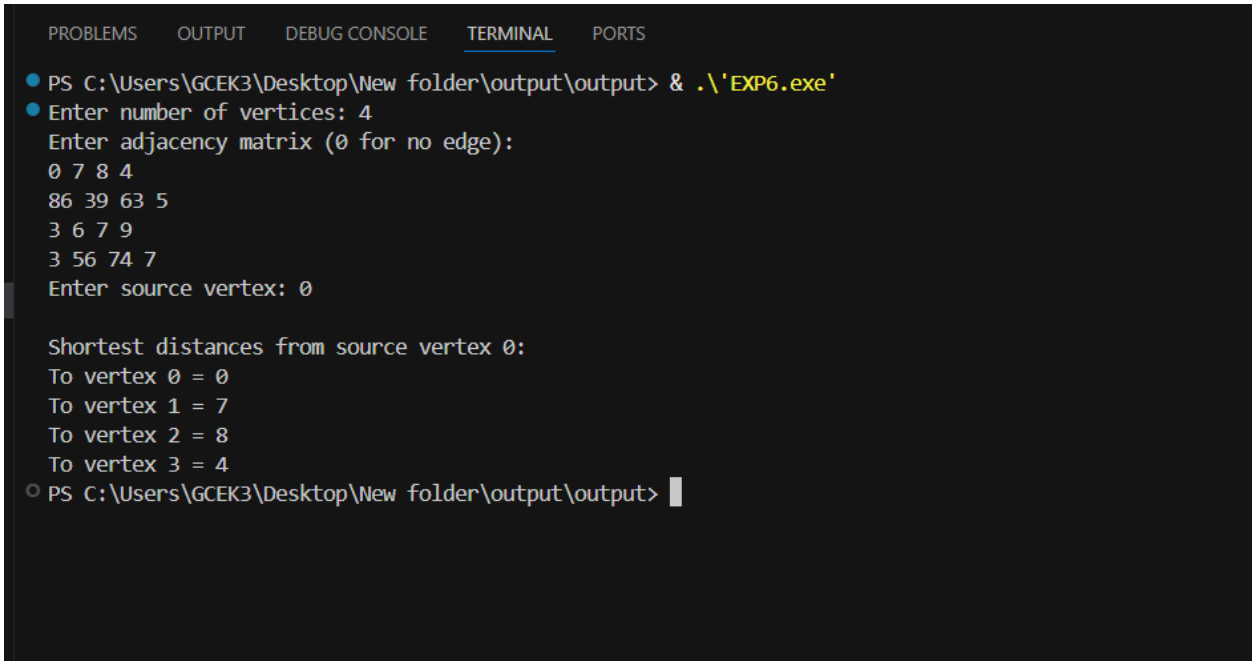
**Program:**

```
#include <stdio.h>
#include <stdbool.h>
#define INF 99999
#define MAX 50
int findMinDist(int dist[], bool visited[], int n) {
    int min = INF, minIndex;
    for (int i = 0; i < n; i++) {
        if (!visited[i] && dist[i] <= min) {
            min = dist[i];
            minIndex = i;
        }
    }
    return minIndex;
}
void dijkstra(int graph[MAX][MAX], int n, int src) {
    int dist[MAX];
    bool visited[MAX];
    for (int i = 0; i < n; i++) {
        dist[i] = INF;
        visited[i] = false;
    }
    dist[src] = 0;
    for (int count = 0; count < n - 1; count++) {
        int u = findMinDist(dist, visited, n);
        visited[u] = true;
```

```
        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];
            }
        }
    }
    printf("\nShortest distances from source vertex %d:\n", src);
    for (int i = 0; i < n; i++) {
        printf("To vertex %d = %d\n", i, dist[i]);
    }
}
int main() {
    int n;
    int graph[MAX][MAX];
    int src;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix (0 for no edge):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
```

```
    printf("Enter source vertex: ");
    scanf("%d", &src);
    dijkstra(graph, n, src);
    return 0;
}
```

**OUTPUT:**




**Time and space complexity:**

Time Complexity: $O(V^2)$

Space Complexity: $O(V^2)$

**Conclusion:**

Dijkstra's Algorithm is a powerful and efficient method for computing shortest paths in weighted graphs with non-negative weights. Its greedy approach ensures optimal performance and makes it highly applicable in routing, navigation, robotics, and network analysis.