EXPERIMENT NO. 09

Roll No.: 24141001
Batch: I1

Title: Single Source Shortest Path Problem

Objectives:

Understand the concept of shortest path from one source vertex to all other vertices in a graph.

Learn how to solve the Single Source Shortest Path (SSSP) problem using Dijkstra's algorithm.

Implement Dijkstra's algorithm in C.

Analyze the time and space complexity.

Theory:

The Single Source Shortest Path (SSSP) problem finds the minimum distance from a selected start vertex (source) to every other vertex in a weighted graph.

Dijkstra's Algorithm

Dijkstra's algorithm is a greedy algorithm used to compute shortest paths from a source node in a graph with non-negative edge weights.

Works For:

Graphs with positive weights
Directed or undirected graphs

Does NOT Work For:

Graphs with negative edge weights

Algorithm:

1. Input the graph and source vertex.

2. Initialize:

    dist[i] = infinity for all i

    dist[source] = 0

visited[i] = false

3. Repeat for all vertices:

a. Select the unvisited vertex u with minimum dist[u]

b. Mark u as visited

c. For every neighbor v of u:

If dist[v] > dist[u] + weight(u, v)

dist[v] = dist[u] + weight(u, v)

4. Print the distance array.

Program:

```
#include <stdio.h>


#define INF 9999

#define MAX 20


int main() {

    int graph[MAX][MAX], dist[MAX], visited[MAX];

    int n, i, j, src, min, u;


    printf("Enter number of vertices: ");

    scanf("%d", &n);


    printf("Enter adjacency matrix (9999 for no edge):\n");

    for (i = 0; i < n; i++)

        for (j = 0; j < n; j++)
```

```
        scanf("%d", &graph[i][j]);

    printf("Enter the source vertex (0 to %d): ", n - 1);
    scanf("%d", &src);

    // Initialize
    for (i = 0; i < n; i++) {
        dist[i] = graph[src][i];
        visited[i] = 0;
    }
    dist[src] = 0;
    visited[src] = 1;

    // Dijkstra’s algorithm
    for (i = 1; i < n; i++) {
        min = INF;

        for (j = 0; j < n; j++)
            if (!visited[j] && dist[j] < min) {
                min = dist[j];
                u = j;
            }

        visited[u] = 1;
```

```
        for (j = 0; j < n; j++)
            if (!visited[j] && dist[j] > dist[u] + graph[u][j])
                dist[j] = dist[u] + graph[u][j];
    }

    // Output
    printf("\nShortest distances from source %d:\n", src);
    for (i = 0; i < n; i++)
        printf("To %d → %d\n", i, dist[i]);

    return 0;
}
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\GCEK3\Desktop\New folder\output> cd 'c:\Users\GCEK3\Desktop\New folder\output\output'
PS C:\Users\GCEK3\Desktop\New folder\output\output> & .\'EXP9.exe'
Enter number of vertices: 5
Enter adjacency matrix (9999 for no edge):
0 10 9999 30 100
9999 0 50 9999 9999
9999 9999 0 9999 10
9999 9999 20 0 60
9999 9999 9999 9999 0
Enter the source vertex (0 to 4): 0

Shortest distances from source 0:
To 0 ΓåÆ 0
To 1 ΓåÆ 10
To 2 ΓåÆ 50
To 3 ΓåÆ 30
To 4 ΓåÆ 60
PS C:\Users\GCEK3\Desktop\New folder\output\output>
```

Applications of SSSP (Dijkstra's Algorithm):

- GPS navigation systems
- Network routing protocols
- Finding minimum travel time
- Transport and logistics optimization
- Game development (AI pathfinding)

Time and space complexity:

Time Complexity: $O(n^2)$

Space Complexity: $O(n^2)$

Conclusion:

Dijkstra's algorithm efficiently solves the Single Source Shortest Path problem by finding the minimum distance from one source vertex to all other vertices in a graph with positive weights. It is easy to implement, fast, and widely used in real-world applications like routing and navigation.