Experiment – 8

Student Name: Himanshu Gupta **UID:** 23BCS10889
Branch: BE-CSE **Section/Group:** KRG-2B
Semester: 5th **Date of Performance:** 16/10/25
Subject Name: DAA **Subject Code:** 23CSH-301

1. **Aim:** Develop a program and analyze complexity to find shortest paths in a graph with positive edge weights using Dijkstra's algorithm.

2. **Objective:** Code and analyze to find shortest paths in a graph with positive edge weights using Dijkstra's

3. **Input/Apparatus Used:** Graph (G =(V,E)) is taken as input for this problem.

4. **Procedure:**
Follow the steps below to solve the problem:

- Create a set sptSet (shortest path tree set) that keeps track of vertices included in the shortest- path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
- Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign the distance value as 0 for the source vertex so that it is picked first.
- While sptSet doesn‟t include all vertices
- Pick a vertex u which is not there in sptSet and has a minimum distance value.
- Include u to sptSet.
- Then update distance value of all adjacent vertices of u.
- To update the distance values, iterate through all adjacent vertices.
- For every adjacent vertex v, if the sum of the distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

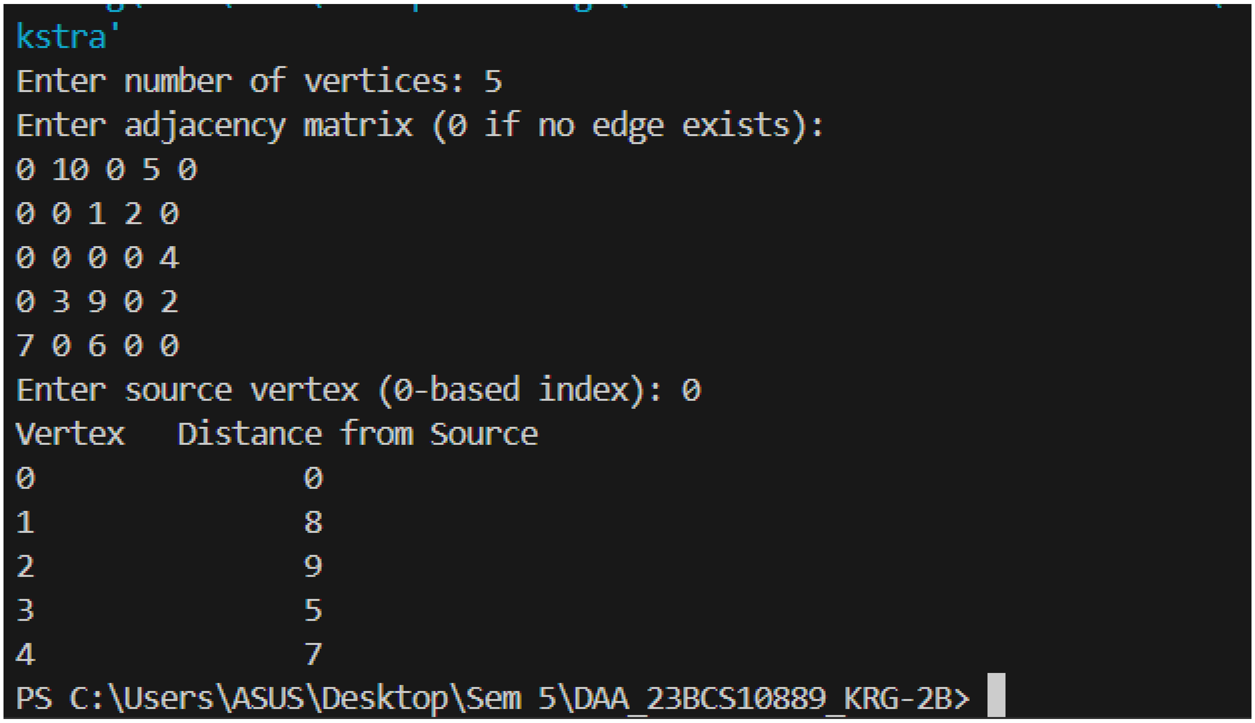
5. Algorithm

- **Step 1:** SET STATUS = 1 (ready state) for each node in G
- **Step 2:** Push the starting node A on the stack and set its STATUS = 2 (waiting state)
- **Step 3:** Repeat Steps 4 and 5 until STACK is empty

- **Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)
- **Step 5:** Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their
    - STATUS = 2 (waiting state) [END OF LOOP]
- **Step 6:** EXIT

6. Code and Output:

```
import java.util.Scanner;

public class Dijkstra {
    private static final int INF = 1000000;

    private static int minDistance(int[] dist, boolean[] sptSet, int V) {
        int min = INF;
        int min_index = -1;

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

        return min_index;
    }

    private static void printSolution(int[] dist, int V) {
        System.out.println("Vertex \t Distance from Source");
        for (int i = 0; i < V; i++) {
            System.out.println(i + "\t\t" + dist[i]);
        }
    }

    private static void dijkstra(int[][] graph, int src, int V) {
        int[] dist = new int[V];
        boolean[] sptSet = new boolean[V];

        for (int i = 0; i < V; i++) {
            dist[i] = INF;
            sptSet[i] = false;
        }

        dist[src] = 0;
        for (int count = 0; count < V - 1; count++) {
            int u = minDistance(dist, sptSet, V);
            if (u == -1) break; // no reachable vertex left
            sptSet[u] = true;

            for (int v = 0; v < V; v++) {
                if (!sptSet[v] && graph[u][v] != 0 && dist[u] + graph[u][v] < dist[v]) {
                    dist[v] = dist[u] + graph[u][v];
                }
            }
        }

        printSolution(dist, V);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of vertices: ");
        int V = sc.nextInt();

        int[][] graph = new int[V][V];
        System.out.println("Enter adjacency matrix (0 if no edge exists):");
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                graph[i][j] = sc.nextInt();
            }
        }

        System.out.print("Enter source vertex (0-based index): ");
        int src = sc.nextInt();

        if (src < 0 || src >= V) {
            System.out.println("Invalid source vertex.");
            sc.close();
            return;
        }

        dijkstra(graph, src, V);
        sc.close();
    }
}
```



```
kstra'
Enter number of vertices: 5
Enter adjacency matrix (0 if no edge exists):
0 10 0 5 0
0 0 1 2 0
0 0 0 0 4
0 3 9 0 2
7 0 6 0 0
Enter source vertex (0-based index): 0
Vertex   Distance from Source
0               0
1               8
2               9
3               5
4               7
PS C:\Users\ASUS\Desktop\Sem 5\DAA_23BCS10889_KRG-2B>
```