Experiment No: 9

Student Name: Arpit Anand
UID: 23BCS12710
Branch: BE-CSE
Section/Group: KGR-3(A)
Semester: 5th
Subject Name: DAA

Subject Code: 23CSH-301
Date of Performance:27th Oct, 2025

Aim: Develop a program and analyze complexity to find shortest paths in a graph with positive edgeweights using Dijkstra‟s algorithm.

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

Procedure/Algorithm/Pseudocode:

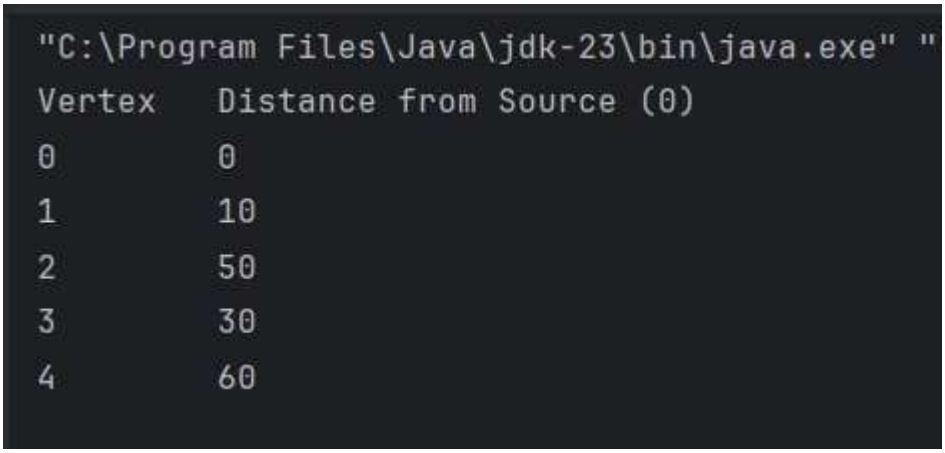
1. Algorithm: Dijkstra's Algorithm
2. Input: A weighted graph G(V, E) with non-negative edge weights and source vertex s.
   Initialize:
3. Set distance dist[v] = ∞ for all vertices v, except dist[s] = 0.
4. Mark all vertices as unvisited.
5. Repeat until all vertices are processed:
6. Select the unvisited vertex u with the smallest distance value.
7. Mark u as visited.
   For each neighbor v of u, update:
8. dist[v] = min(dist[v], dist[u] + weight(u, v)).
   Output:
9. The array dist[] now contains the shortest distances from the source to all vertices..

Code:

```
package DP;

import java.util.*;

public class DijkstraAlgorithm {

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

      Arrays.fill(dist, Integer.MAX_VALUE);
      dist[src] = 0;

      PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
      pq.offer(new int[]{src, 0});

      while (!pq.isEmpty()) {
         int[] node = pq.poll();
         int u = node[0];

         if (visited[u]) continue;
         visited[u] = true;

         for (int v = 0; v < V; v++) {
            if (graph[u][v] != 0 && !visited[v]) {
               int newDist = dist[u] + graph[u][v];
               if (newDist < dist[v]) {
                  dist[v] = newDist;
                  pq.offer(new int[]{v, dist[v]});
               }
            }
         }
      }

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

   public static void main(String[] args) {
      int V = 5;
      int[][] graph = {
            {0, 10, 0, 30, 100},
            {10, 0, 50, 0, 0},
            {0, 50, 0, 20, 10},
            {30, 0, 20, 0, 60},
```

```
            {100, 0, 10, 60, 0}
        };

        dijkstra(V, graph, 0);
    }
}
```

Output :



Time Complexity :

Time Complexity

O((V + E) log V)

Using PriorityQueue (Min-Heap)

Learning Outcomes:

1. Understood the concept of shortest path in weighted graphs.
2. Learned how Dijkstra's algorithm efficiently finds minimal paths.
3. Implemented the algorithm using Priority Queue (Min-Heap) in Java.
4. Analyzed time and space complexity for both matrix and adjacency list representations.
5. Gained hands-on experience in graph algorithms using Dynamic Data Structures.