**23. Implementation of Shortest Path Algorithms using Dijkstra’s Algorithm.**

**PROGRAM:**

#include <stdio.h>

#include <limits.h>

// Number of vertices in the graph

#define V 9

// Function to find the vertex with the minimum distance value, from the set of vertices not yet included in shortest path tree

int minDistance(int dist[], int sptSet[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == 0 && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

// Function to print the constructed distance array

void printSolution(int dist[], int n) {

printf("Vertex \t Distance from Source\n");

for (int i = 0; i < V; i++)

printf("%d \t %d\n", i, dist[i]);

}

// Function that implements Dijkstra's single source shortest path algorithm for a graph represented using adjacency matrix representation

void dijkstra(int graph[V][V], int src) {

int dist[V]; // The output array. dist[i] will hold the shortest distance from src to i

int sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest path tree or shortest distance from src to i is finalized

// Initialize all distances as INFINITE and stpSet[] as false

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = 0;

// Distance of source vertex from itself is always 0

dist[src] = 0;

// Find shortest path for all vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum distance vertex from the set of vertices not yet processed. u is always equal to src in first iteration.

int u = minDistance(dist, sptSet);

// Mark the picked vertex as processed

sptSet[u] = 1;

// Update dist value of the adjacent vertices of the picked vertex

for (int v = 0; v < V; v++)

// Update dist[v] only if it is not in sptSet, there is an edge from u to v, and total weight of path from src to v through u is smaller than current value of dist[v]

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

// Print the constructed distance array

printSolution(dist, V);

}

int main() {

// Sample graph represented using adjacency matrix

int graph[V][V] = {

{0, 4, 0, 0, 0, 0, 0, 8, 0},

{4, 0, 8, 0, 0, 0, 0, 11, 0},

{0, 8, 0, 7, 0, 4, 0, 0, 2},

{0, 0, 7, 0, 9, 14, 0, 0, 0},

{0, 0, 0, 9, 0, 10, 0, 0, 0},

{0, 0, 4, 14, 10, 0, 2, 0, 0},

{0, 0, 0, 0, 0, 2, 0, 1, 6},

{8, 11, 0, 0, 0, 0, 1, 0, 7},

{0, 0, 2, 0, 0, 0, 6, 7, 0}

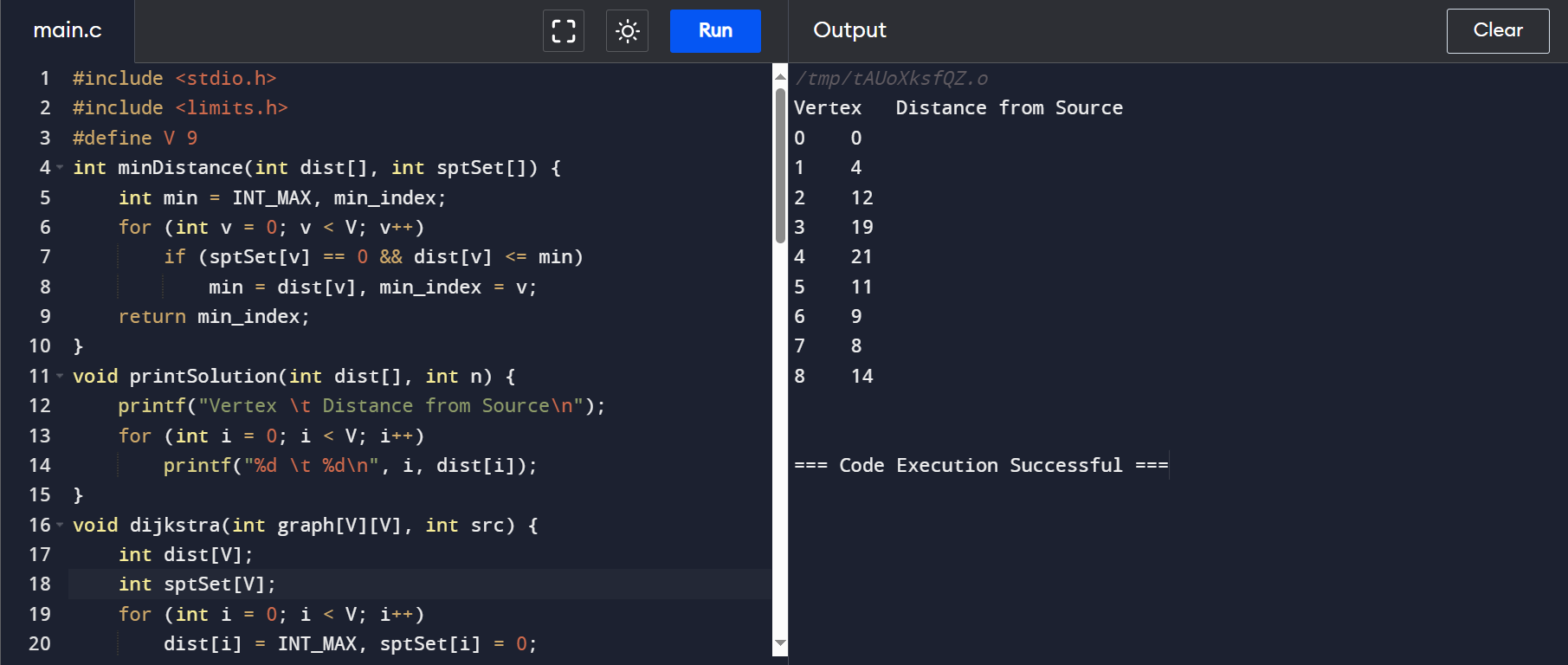
};

dijkstra(graph, 0); // Call Dijkstra's algorithm with source vertex 0

return 0;

}

**INPUT:**



**OUTPUT:**

