**DIJKSTRA’S ALGORITHM**

**AIM:**

To create A graph and find the shortest path using dijkstra’s algorithum

**ALGORITHM:**

 **Initialize the Graph**:

* Define the number of vertices V.
* Initialize the adjacency matrix graph which represents the graph.

 **Initialize Distance and Set Arrays**:

* Create an array dist to store the shortest distance from the source to each vertex, initialized to INT\_MAX.
* Create a boolean array sptSet to keep track of vertices included in the shortest path tree (SPT), initialized to false.

 **Set Distance of Source Vertex**:

* Set the distance of the source vertex to itself as 0 (dist[src] = 0).

 **Find the Minimum Distance Vertex**:

* For V-1 times:
  + Find the vertex u that has the minimum distance value from the set of vertices not yet included in the SPT using the minDistance function.
  + Mark u as processed (sptSet[u] = true).

 **Update Distance Values**:

* For each vertex v adjacent to u:
  + If v is not in the SPT, there is an edge from u to v, the total weight of the path from the source to v through u is smaller than the current value of dist[v], then update dist[v].

 **Print the Result**:

* Use the printSolution function to print the shortest distances from the source to all other vertices.

**PROGRAM**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 9

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

int min = INT\_MAX, min\_index;

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

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

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

return min\_index;

}

void printSolution(int dist[]) {

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

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

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

}

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

int dist[V];

bool sptSet[V];

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

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

dist[src] = 0;

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

int u = minDistance(dist, sptSet);

sptSet[u] = true;

for (int v = 0; v < V; 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];

}

printSolution(dist);

}

int main() {

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);

return 0;

}

**OUTPUT:**

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 } };

Vertex Distance from Source

0 0

1 4

2 12

3 19

4 21

5 11

6 9

7 8

8 14

**RESULT:** Thus the shortest path between the given source vertex and other vertices in the graph is found using dijkstra’s algorithm