

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
#define V 9;
```

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

```
{
```

```
    int min = 9999, min_index;
```

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

```
    {  
        if (sptSet[i] == false &&  
            dist[i] < min)
```

```
        min = dist[i], min_index = i;
```

```
    }  
    return min_index;
```

```
}
```

```
void printPath (int parent[], int j)
```

```
{
```

```
    if (parent[j] == -1)
```

```
        return;  
    printPath (parent, parent[j]);
```

```
    cout << j << " ";
```

```
}
```

```

void printsolution (int dist[], int n, int parent)
{
    int src = 0;
    cout << "Vertex | Distance | Path" << endl;
    for (int i = 1; i < n; i++)
    {
        cout << "\n" << src << " -> " << i << " | "
        << dist[i] << " | " << src << " ";
        printpath (parent, i);
    }
}

```

```

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

```

```

{
    int dist[V];
    bool bool optset[V];

```

```

    int parent[V];

```

```

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

```

```

        dist[i] = 0;

```

```

        parent[i] = -1;

```

```

        dist[i] = 9999;

```

```

        optset[i] = false;
    }
}

```

```
dist[src] = 0;
```

```
for (int count = 0; count < V; count++)
```

```
{ int u = minDistance(dist, sptset);
```

```
sptset[u] = true;
```

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

```
{ if (!sptset[v] && graph[u][v]  
&& dist[u] + graph[u][v]  
    < dist[v])
```

```
{
```

```
    parent[v] = u;
```

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

```
}
```

```
}  
printSolution(dist, V, parent);
```

```
}
```

```
int main()
```

```
{
```

```
    int graph[V][V];
```

```
    cout << "Enter the graph: " << endl;
```

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

```
    { for (int j = 0; j < V; j++)
```

```
        cin >> graph[i][j];
```

```
}
```

```

cout << "Enter the source : ";
int src;
cin >> src;
dijkstra (graph, src);
cout << endl;
return 0;
}

```

```

// Dijkstra's Algorithm
// Graph is represented as an adjacency list
// Source is given as input
// Distance array is used to store the shortest distance from source to all vertices
// Priority queue is used to select the vertex with minimum distance
// Time complexity: O(V^2)
// Space complexity: O(V)

```

```

// Function to calculate the shortest distance from source to all vertices
// Parameters: graph (adjacency list), src (source vertex)
// Returns: distance array

```

```

// Dijkstra's Algorithm
// Graph is represented as an adjacency list
// Source is given as input
// Distance array is used to store the shortest distance from source to all vertices
// Priority queue is used to select the vertex with minimum distance
// Time complexity: O(V^2)
// Space complexity: O(V)

```

```

// Dijkstra's Algorithm
// Graph is represented as an adjacency list
// Source is given as input
// Distance array is used to store the shortest distance from source to all vertices
// Priority queue is used to select the vertex with minimum distance
// Time complexity: O(V^2)
// Space complexity: O(V)

```

```

// Dijkstra's Algorithm
// Graph is represented as an adjacency list
// Source is given as input
// Distance array is used to store the shortest distance from source to all vertices
// Priority queue is used to select the vertex with minimum distance
// Time complexity: O(V^2)
// Space complexity: O(V)

```

```

// Dijkstra's Algorithm
// Graph is represented as an adjacency list
// Source is given as input
// Distance array is used to store the shortest distance from source to all vertices
// Priority queue is used to select the vertex with minimum distance
// Time complexity: O(V^2)
// Space complexity: O(V)

```