Dijkstra's Algorithm:-

```c
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

// A utility function to find the vertex with minimum distance value,
// from the set of vertices not yet included in the shortest path tree
int minDistance(int dist[], bool visited[], int V) {
    // Initialize min value
    int min = INT_MAX, min_index;

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

    return min_index;
}

// A utility function to print the constructed distance array along with paths
void printSolution(int dist[], int parent[], int V, int src) {
    printf("Vertex \t\t Distance from Source \t Path\n");
    for (int i = 0; i < V; i++) {
        printf("%d \t\t\t %d \t\t\t ", i + 1, dist[i]);

        // Print the path
        int current = i;
        while (current != src) {
            printf("%d <- ", current + 1);
            current = parent[current];
        }
        printf("%d", src + 1);
        printf("\n");
    }
}

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int V, int graph[V][V], int src) {
    int dist[V];    // The output array. dist[i] will hold the shortest
distance from src to i
    int parent[V];  // Array to store the parent of each vertex in the
shortest path
    bool visited[V]; // visited[i] will be true if vertex i is included in the
shortest
```

```
                    // path tree or shortest distance from src to i is
finalized

    // Initialize all distances as INFINITE, visited[] as false, and parent[]
as -1
    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        visited[i] = false;
        parent[i] = -1;
    }

    // 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
        int u = minDistance(dist, visited, V);

        // Mark the picked vertex as processed
        visited[u] = true;

        // 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 visited, there is an edge
from u to v,
            // and the total weight of the path from src to v through u is
smaller than the current value of dist[v]
            if (!visited[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] +
graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
                parent[v] = u;
            }
        }
    }

    // Print the constructed distance array along with paths
    printSolution(dist, parent, V, src);
}

// Driver's code
int main() {
    int V;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &V);

    printf("Enter the values of the adjacency matrix:\n");
```

```
    int graph[V][V];
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            scanf("%d", &graph[i][j]);
        }
    }

    // Function call
    dijkstra(V, graph, 0);

    return 0;
}
```

```
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\AoA\Lab_5>  & 'c:\Users\Mark Lopes\
DebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-vxjstfqd.ens' '--stdout=Micros
'--pid=Microsoft-MIEngine-Pid-kfmlazho.1q0' '--dbgExe=C:\msys64\mingw64\bin\gdb.
Enter the number of vertices in the graph: 6
Enter the values of the adjacency matrix:
0 20 15 0 0 0
2 0 0 0 10 0
0 0 0 4 0 10
0 0 0 0 0 0
0 0 0 15 0 0
0 36 0 4 10 0
Vertex          Distance from Source    Path
1                       0                       1
2                       20                      2 <- 1
3                       15                      3 <- 1
4                       19                      4 <- 3 <- 1
5                       30                      5 <- 2 <- 1
6                       25                      6 <- 3 <- 1
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\AoA\Lab_5>
```

Prims algorithm:-

```c
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define V 6 // Number of vertices in the graph

// A utility function to find the vertex with minimum distance value,
// from the set of vertices not yet included in the MST
int minKey(int key[], bool visited[]) {
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++) {
        if (visited[v] == false && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }

    return min_index;
}

// A utility function to print the constructed MST and distance from the
source
void printMST(int parent[], int graph[V][V], int src) {
    printf("Node \tDistance from Source \tNearest Node\n");
    int cost = 0;
    for (int i = 0; i < V; i++) {
        printf("%d \t\t%d \t\t\t\t", i+1, graph[i][parent[i]]);
        cost += graph[i][parent[i]];

        // Print the path
        int current = i;
        while (current != src) {
            printf("%d <- ", current+1);
            current = parent[current];
        }
        printf("%d", src+1);

        printf("\n");
    }
    printf("The total cost is %d", cost);
}


// Function to construct and print MST for a graph represented using adjacency
matrix representation
```

```
void primMST(int graph[V][V]) {
    int parent[V]; // Array to store constructed MST
    int key[V];    // Key values used to pick minimum weight edge in cut
    bool visited[V]; // To represent set of vertices not yet included in MST

    // Initialize all keys as INFINITE, visited[] as false
    for (int i = 0; i < V; i++) {
        key[i] = INT_MAX;
        visited[i] = false;
    }

    // Always include first vertex in MST
    key[0] = 0;     // Make key 0 so that this vertex is picked as first
vertex
    parent[0] = -1; // First node is always root of MST

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum key vertex from the set of vertices not yet
included in MST
        int u = minKey(key, visited);

        // Add the picked vertex to the MST Set
        visited[u] = true;

        // Update key value and parent index of the adjacent vertices of the
picked vertex.
        // Consider only those vertices which are not yet included in MST
        for (int v = 0; v < V; v++) {
            // graph[u][v] is non-zero only for adjacent vertices of m
            // visited[v] is false for vertices not yet included in MST
            // Update the key only if graph[u][v] is smaller than key[v]
            if (graph[u][v] && visited[v] == false && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    printMST(parent, graph, 0); // Assuming source node is 0
}

// Driver program to test above function
int main() {
    // Example graph representation using adjacency matrix
    int graph[V][V] = {
        {0, 6, 3, 0, 0, 0},
        {6, 0, 2, 5, 0, 0},
```

```
        {3, 2, 0, 3, 4, 0},
        {0, 5, 3, 0, 2, 3},
        {0, 0, 4, 2, 0, 5},
        {0, 0, 0, 3, 2, 0}
    };

    primMST(graph);

    return 0;
}
```

```
 --pid=Microsoft-MIEngine-Pid-xbhravh1.rxx   --dbgExe=C:\msys64\mingw6
Node    Distance from Source    Nearest Node
1               0                               1
2               2                               2 <- 3 <- 1
3               3                               3 <- 1
4               3                               4 <- 3 <- 1
5               2                               5 <- 4 <- 3 <- 1
6               3                               6 <- 4 <- 3 <- 1
The total cost is 13
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\AoA\Lab_5>
```

POSTLAB:-

9913
Mark Lopes

FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

Lab 5: Postlab

1] Dijkstra's algo time complexity

The method 'dijkstra' has a for loop for find shortest path which runs for 'V' times. Inside that loop, is another method 'minDistance' which also has a for loop running for 'V' times
∴ time complexity = V × V = O(V²)

```
dijkstra
{
    for(int i=0; i<V; i++)
    {
        O(V)
    }

for (int count=0; count<V-1 ; count++) → V
    {
        for (int v=0; v<V; v++)  → V
        {
            if (!visited[v] && graph[u][v] && dist[u]
            != INT_MAX && dist[u] + graph[u][v]
            < dist[v])  → log V
            {

            }
        }
    }
}
```

∴ Time complexity = O(V) + O(V² log V) = O(V² log V)

e) Prims algo time complexity.

```
void primMST()
{
    ~~for(int~~
    for loop to initialize keys as INFINITE, visited[] as false   → O(V)

    for(int count=0; count<V-1; count++)
    {
        ~~int u = minKey(key, visited); O(V)~~
        int u = minKey(key, visited) → O(V)

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

∴ Time complexity = $O(V^2)$ //