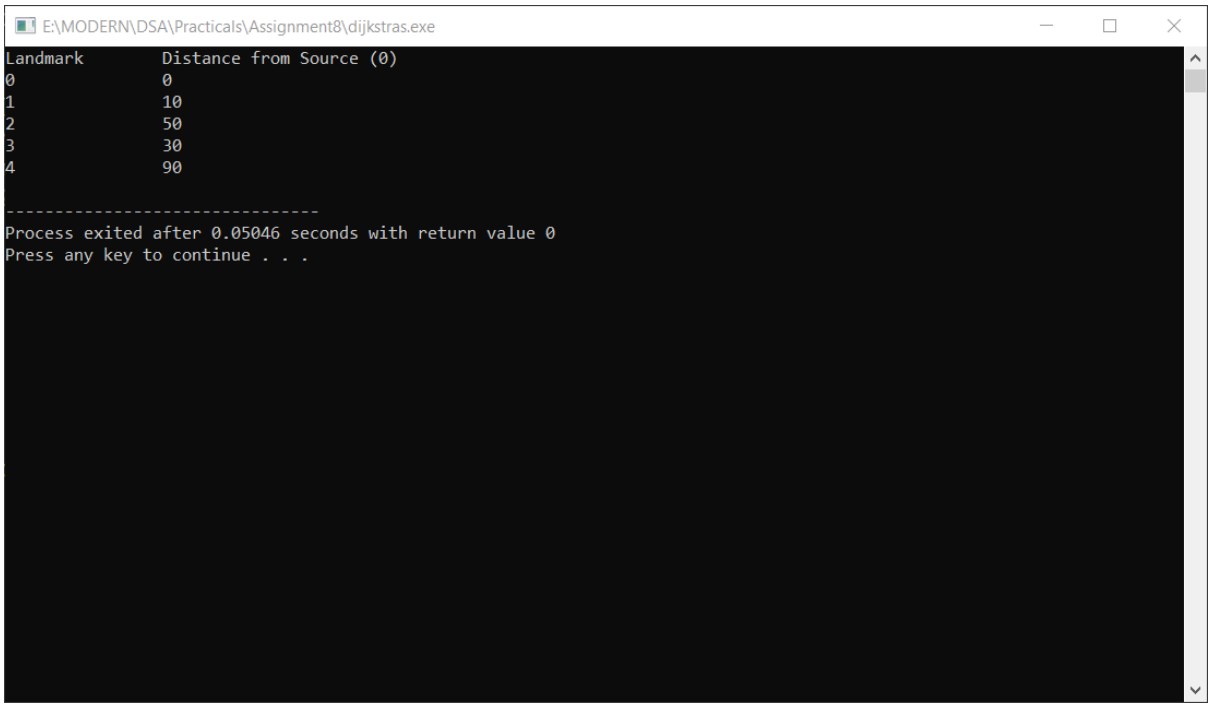
```
 /*
                            ASSIGNMENT NO.8
NAME- ABRAR SHAIKH                                  ROLL NO. - 23570
                       TOPIC- Dijkstra's Algorithm
*/


#include <iostream>
#include <limits.h> // For INT_MAX
#define INF 9999 // Define a large number to represent infinity
#define MAX 5   // Maximum number of landmarks

using namespace std;

class Graph {
    private:
        int adjMatrix[MAX][MAX]; // Adjacency matrix to store
distances
        bool visited[MAX];       // Track visited nodes
        int distance[MAX];       // Store the shortest distance from
source

    public:
        Graph() {
            for (int i = 0; i < MAX; i++) {
                for (int j = 0; j < MAX; j++) {
                    adjMatrix[i][j] = (i == j) ? 0 : INF; // 0 for
same node, INF for no connection
                }
                visited[i] = false;
                distance[i] = INF;
            }
        }
```

```
        // Function to add an edge between two landmarks with
distance
        void addEdge(int u, int v, int dist) {
            adjMatrix[u][v] = dist;
            adjMatrix[v][u] = dist; // For an undirected graph
        }

        // Dijkstra's Algorithm to find shortest path from source
        void dijkstra(int source) {
            distance[source] = 0;

            for (int count = 0; count < MAX - 1; count++) {
                int u = minDistance(); // Get the unvisited node
with the smallest distance

                visited[u] = true; // Mark the node as visited

                for (int v = 0; v < MAX; v++) {
                    // If v is unvisited and there's a direct edge
from u to v
                    if (!visited[v] && adjMatrix[u][v] != INF &&
distance[u] != INF
                        && distance[u] + adjMatrix[u][v] <
distance[v]) {
                        distance[v] = distance[u] + adjMatrix[u][v];
                    }
                }
            }

            // Print the shortest path from source to all nodes
            printSolution(source);
        }
```

```
        // Helper function to find the node with the minimum
distance that hasn't been visited
        int minDistance() {
            int min = INF, min_index;

            for (int v = 0; v < MAX; v++) {
                if (!visited[v] && distance[v] <= min) {
                    min = distance[v];
                    min_index = v;
                }
            }
            return min_index;
        }

        // Print the result of Dijkstra's Algorithm
        void printSolution(int source) {
            cout << "Landmark\tDistance from Source (" << source <<
")\n";
            for (int i = 0; i < MAX; i++) {
                cout << i << "\t\t" << distance[i] << endl;
            }
        }
};

int main() {
    Graph g;

    // Adding edges with distances between landmarks
    g.addEdge(0, 1, 10);
    g.addEdge(0, 3, 30);
```

```
    g.addEdge(0, 4, 100);
    g.addEdge(1, 2, 50);
    g.addEdge(2, 3, 20);
    g.addEdge(3, 4, 60);


    int source = 0;
    g.dijkstra(source); // Find the shortest path from landmark 0 to
all other landmarks


    return 0;
}
```



GitHub Repository- https://github.com/abssha/DSA.git