# Dijkstra's Algorithm

an algorithm that is used for finding the shortest distance, or path, from starting node to target node in a weighted graph is known as Dijkstra’s Algorithm.

The algorithm uses a greedy approach (so it won’t work correctly with negative weights) to obtain the best solution.

This algorithm makes a tree of the shortest path from the starting node, the source, to all other nodes (points) in the graph.

Dijkstra's algorithm makes use of weights of the edges for finding the path that minimizes the total distance (weight) among the source node and all other nodes. This algorithm is also known as the single-source shortest path algorithm.

# Algorithm

1.create a function with name Dijkstra(int arr[N][N],int src)

2.create a bool array called visited and initialize to false

3.create a integer array called dij and initialize all elements as INT_MAX

4.declare visited[src] 1 and dij[src]=0

5.run a loop upto length -1

6.find the minimum index by min function with dij and visited

7.make visited[minindex]=true

8.run second forloop from 0 to n

9.if(visted[i]==false and graph[minindex][i]==0 and dij[i]!=INT_MAX and dij[minindex]+graph[minindex][i]<dij[i])

10.intialize dij[i]=graph[minindex][i]+dij[minindex]

# Dijkstra's Algorithm Applications

- To find the shortest path
- In social networking applications
- In a telephone network
- To find the locations in the map
- In telecommunications to determine transmission rate.
- In robotic design to determine shortest path for automated robots.

# DIJKSTRA CODE

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.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\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t\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;
}
```

- Time Complexity: $O(V^2)$
  Auxiliary Space: $O(V)$

- Although the complexity can be improved depending on the implementation, seeing the pseudocode that I provided above, the time complexity of the algorithm is $O(n^2)$ being n the number of vertexes. And the space complexity O(n).