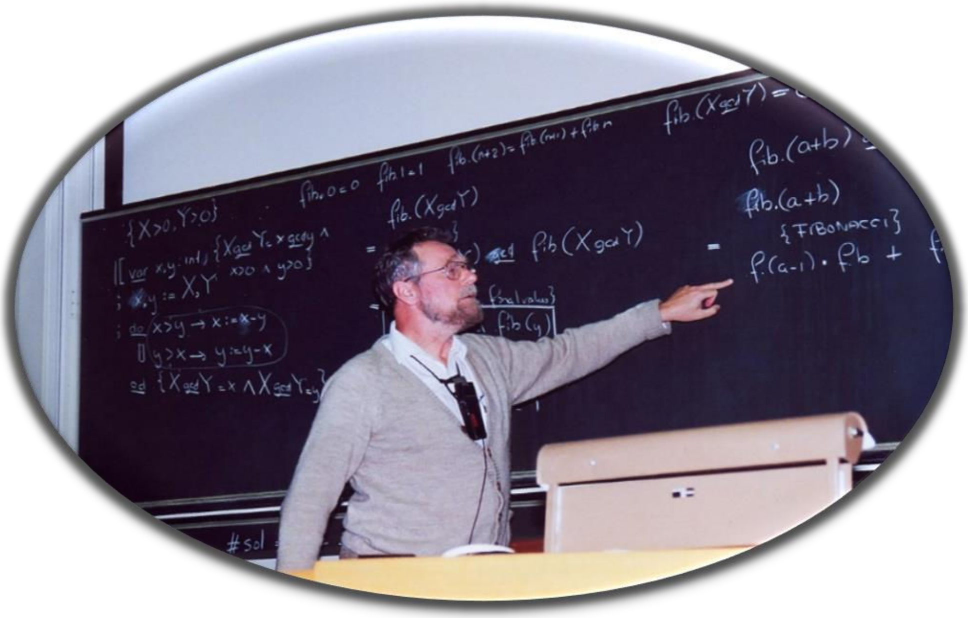
# DIJKSTRA'S ALGORITHM

- **Dijkstra's algorithm** is a popular algorithm in computer science and graph theory, designed to find the **shortest path** between a starting vertex and all other vertices in a weighted graph with non-negative edge weights.
- It was conceived by Dutch computer scientist **Edsger Dijkstra** in **1956** and is commonly used in applications such as routing and network protocols.



## Requirements :

- **Dijkstra's Algorithm** can only work with graphs that have positive weights.
- This is because, during the process, the weights of the edges have to be added to find the shortest path.
- If there is a negative weight in the graph, then the algorithm will not work properly
- Once a node has been marked as "visited", the current path to that node is marked as the shortest path to reach that node.
- And negative weights can alter this if the total weight can be decremented after this step has occurred.

### Dijkstra's Algorithm Complexity

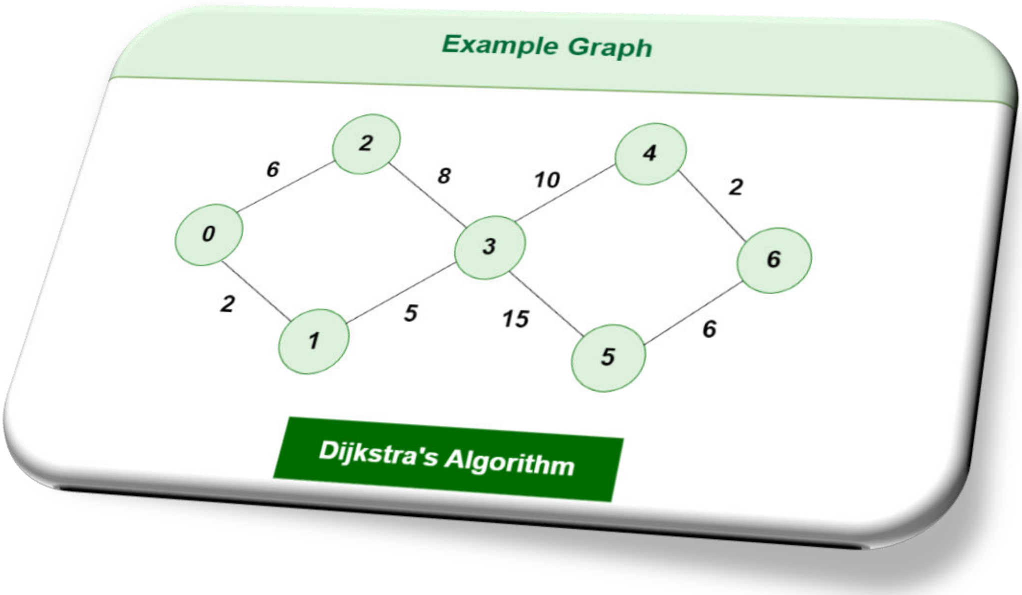
- **Time Complexity:** O(E Log V)

where, E is the number of edges and V is the number of vertices.

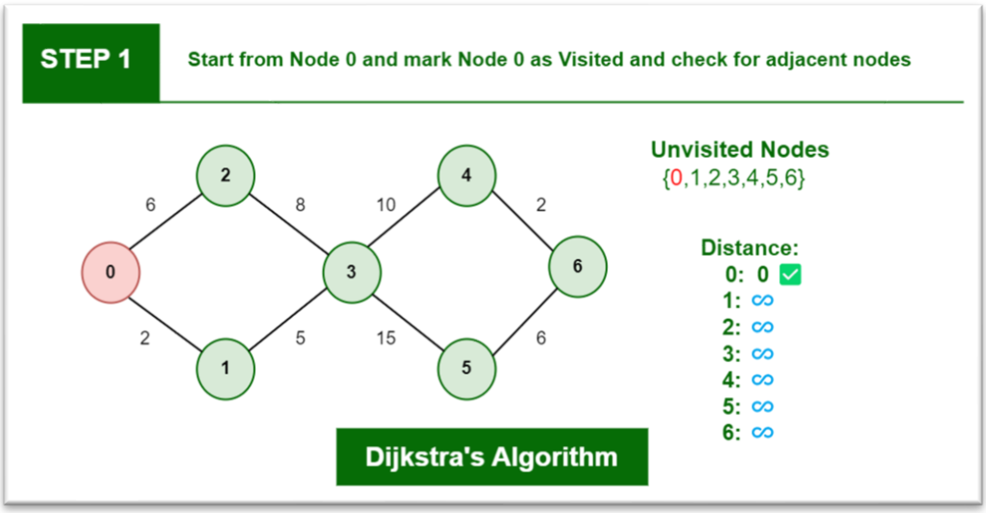
- **Space Complexity:** O(V)



## How does Dijkstra's Algorithm works?

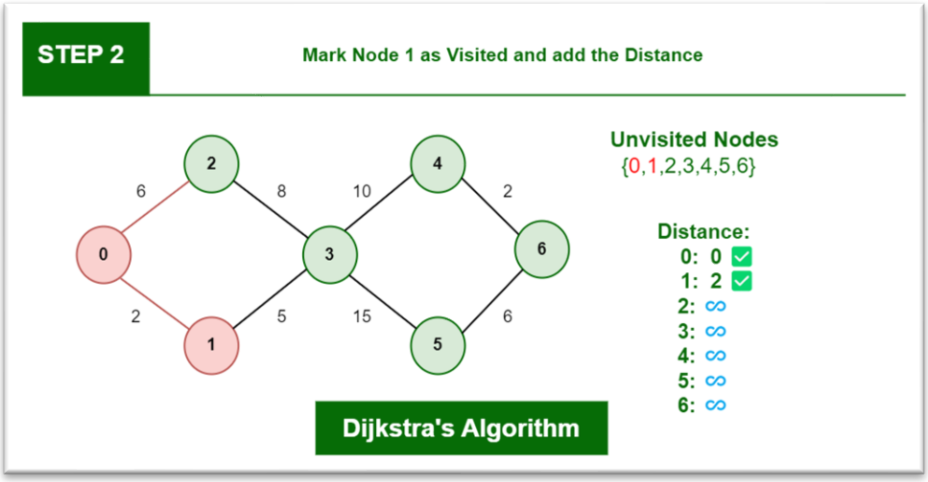



**Step 1:** Start from Node **0** and mark Node as visited as you can check in below image visited Node is marked red.



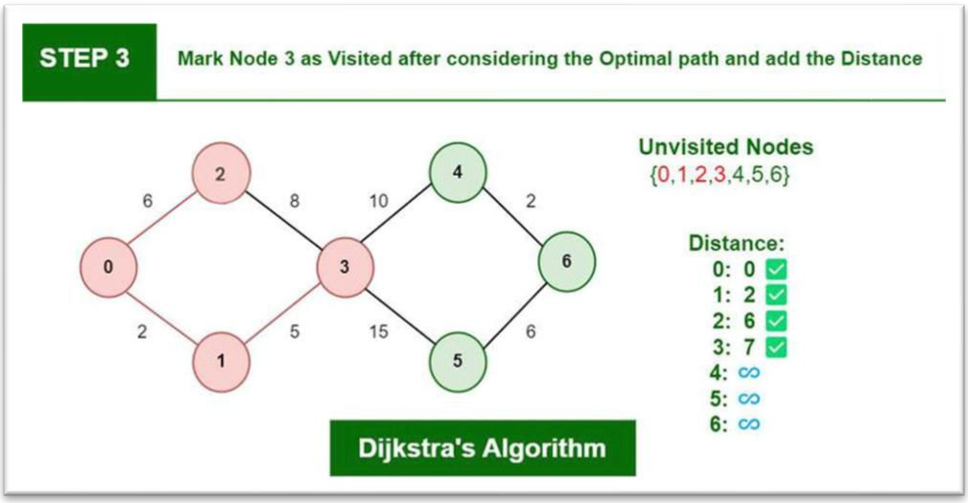
**Step 2:** Check for adjacent Nodes, Now we have to choices (Either choose Node1 with distance **2** or either choose Node **2** with distance **6** ) and choose Node with minimum distance. In this step Node **1** is Minimum distance adjacent Node, so marked it as visited and add up the distance.

**Distance: Node 0 -> Node 1 = 2**




**Step 3:** Then Move Forward and check for adjacent Node which is Node **3**, so marked it as visited and add up the distance, Now the distance will be:

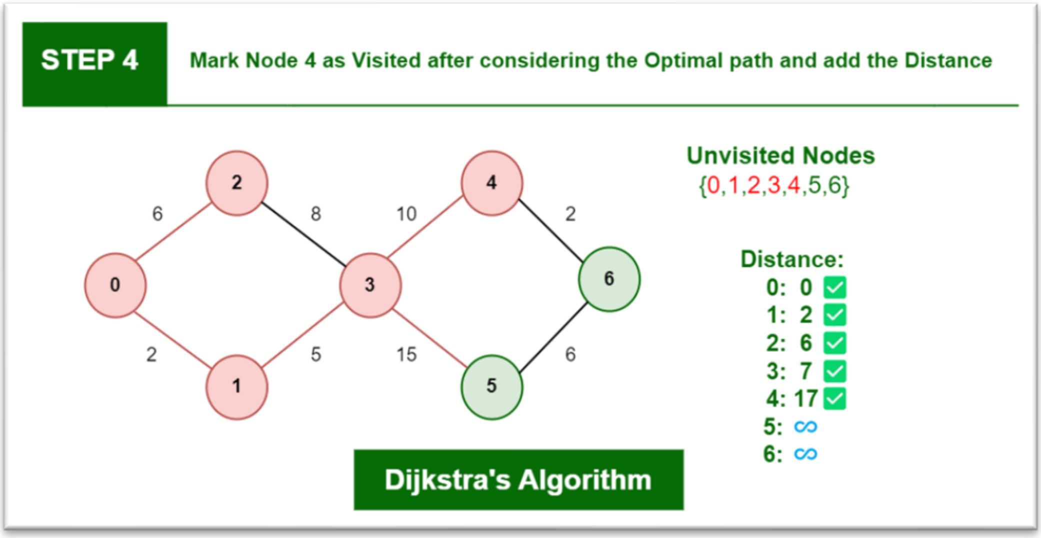
**Distance: Node 0 -> Node 1 -> Node 3 = 2 + 5 = 7**



**Step 4:** Again we have two choices for adjacent Nodes (Either we can choose Node 4 with distance 10 or either we can choose Node 5 with distance 15) so choose Node with minimum distance. In this step Node 4 is Minimum distance adjacent Node, so marked it as visited and add up the distance.

**Distance: Node 0 -> Node 1 -> Node 3 -> Node 4 = 2 + 5 + 10 = 17**

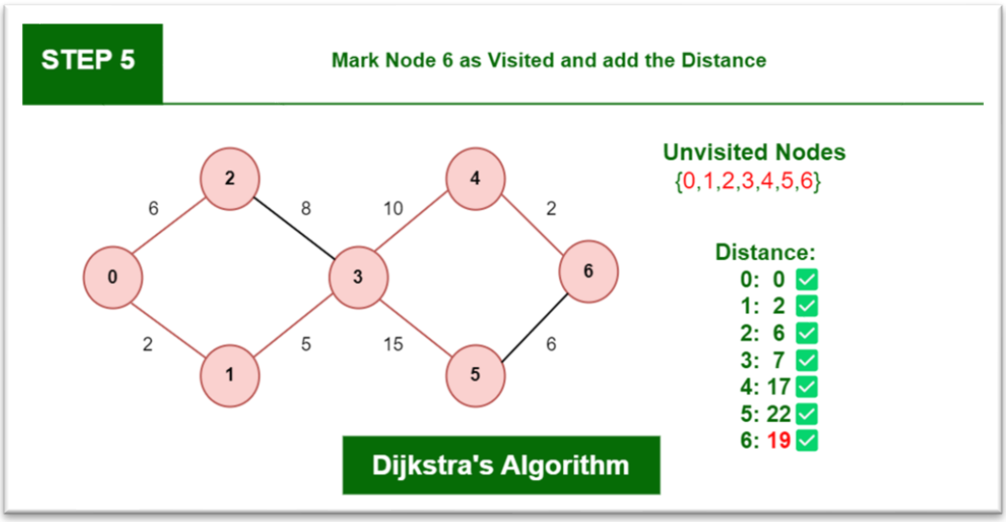





**Step 5:** Again, Move Forward and check for adjacent Node which is **Node 6,** so marked it as visited and add up the distance, Now the distance will be:

**Distance:**

**Node 0 -> Node 1 -> Node 3 -> Node 4 -> Node 6 = 2 + 5 + 10 + 2 = 19**



**So, the Shortest Distance from the Source Vertex is 19 which is optimal one**

## Algorithm for Dijkstra's Algorithm:

1. **Mark the source node with a current distance of 0 and the rest with infinity.**
2. **Set the non-visited node with the smallest current distance as the current node.**
3. **For each neighbor, N of the current node adds the current distance of the adjacent node with the weight of the edge connecting 0->1. If it is smaller than the current distance of Node, set it as the new current distance of N.**
4. **Mark the current node 1 as visited.**
5. **Go to step 2 if there are any nodes are unvisited.**



## Implementation of Dijkstra's Algorithm:

```
#include <bits/stdc++.h>
using namespace std;
#define INF 0x3f3f3f3f
// iPair ==> Integer Pair
typedef pair<int, int> iPair;

class Graph {
        int V; // No. of vertices
        list<pair<int, int> >* adj;
public:
        Graph(int V); // Constructor
```

```
// function to add an edge to graph
        void addEdge(int u, int v, int w);

        // prints shortest path from s
        void shortestPath(int s);
};

// Allocates memory for adjacency list
Graph::Graph(int V)
{
        this->V = V;
        adj = new list<iPair>[V];
}
void Graph::addEdge(int u, int v, int w)
{
        adj[u].push_back(make_pair(v, w));
        adj[v].push_back(make_pair(u, w));
}
// Prints shortest paths from src to all other vertices
void Graph::shortestPath(int src)
{
        priority_queue<iPair, vector<iPair>, greater<iPair> > pq;
        // Create a vector for distances and initialize all
        // distances as infinite (INF)
        vector<int> dist(V, INF);
```

```
// Insert source itself in priority queue and initialize
        // its distance as 0.
        pq.push(make_pair(0, src));
        dist[src] = 0;

        /* Looping till priority queue becomes empty (or all
        distances are not finalized) */
        while (!pq.empty()) {
                int u = pq.top().second;
                pq.pop();
                // 'i' is used to get all adjacent vertices of a
                // vertex
                list<pair<int, int> >::iterator i;
                for (i = adj[u].begin(); i != adj[u].end(); ++i) {
                        // Get vertex label and weight of current
                        // adjacent of u.
                        int v = (*i).first;
                        int weight = (*i).second;
                        // If there is shorted path to v through u.
                        if (dist[v] > dist[u] + weight) {
                                // Updating distance of v
                                dist[v] = dist[u] + weight;
                                pq.push(make_pair(dist[v], v));
                        }
                }
        }
```

```
// Print shortest distances stored in dist[]
        printf("Vertex Distance from Source\n");
        for (int i = 0; i < V; ++i)
                printf("%d \t\t %d\n", i, dist[i]);
}
// Driver program to test methods of graph class
int main()
{
        // create the graph given in above figure
        int V = 7;
        Graph g(V);
        // making above shown graph
        g.addEdge(0, 1, 2);
        g.addEdge(0, 2, 6);
        g.addEdge(1, 3, 5);
        g.addEdge(2, 3, 8);
        g.addEdge(3, 4, 10);
        g.addEdge(3, 5, 15);
        g.addEdge(4, 6, 2);
        g.addEdge(5, 6, 6);
        g.shortestPath(0);

        return 0;
}
```

## Application of Dijkstra's Algorithm:

- It is used in finding Shortest Path.
- It is used in geographical Maps.
- To find locations of Map which refers to vertices of graph.
- Distance between the location refers to edges.
- It is used in IP routing to find Open shortest Path First.
- It is used in the telephone network.