

*Department of Computer Science*

*University of Management and Technology*

**Parallelizing Dijkstra’s Algorithm Optimized Shortest Path Computation**

**Submitted By:**

* *Muhammad Usman*
* *Asad Nawaz*
* *Hamza*
* *Naveed*

**Supervised By:**

*Iqra Ashraf*

***Date of Submission:*** *January 28, 2025*

Table of Contents

1. [INTRODUCTION 1](#_bookmark0)
2. PROBLEM ………………...………………………………………………………………………2
3. [LITERATURE REVIEW 3](#_bookmark1)
4. [OBJECTIVE 4](#_bookmark2)
5. [METHODOLOY 5](#_bookmark3)
6. [DIAGRAM 6](#_bookmark4)
7. [CODE 7](#_bookmark5)
8. [ACCURACY 8](#_bookmark6)
9. [DISCUSSION 9](#_bookmark7)
10. VISUALTIZATION………………………………………………………………..………...…10
11. CONCLUSION………...…………………………………………………………………………11
12. [REFERENCES 12](#_bookmark15)

**Abstract**. *Finding the shortest path efficiently is essential in various applications, such as routing and road networks. To solve this problem, Dijkstra's Algorithm is widely used due to its accuracy and reliability. This paper explores the implementation of Dijkstra's Algorithm and its effectiveness in solving the shortest path problem across different types of graphs. The study highlights its practical applications and discusses its performance in terms of computational efficiency for small- to large-scale networks.*

## **I**NTRODUCTION

Since the invention of the computer, the problem of optimal path selection has garnered significant attention. Over time, continuous advancements and optimizations in algorithms have been made to address this challenge. Scientists have long sought to develop the most efficient path selection techniques. One of the most notable contributions to this field came from Edsger Wybe Dijkstra, who, in 1959, introduced the first algorithm for solving the optimal path selection problem. This algorithm, now known as Dijkstra's Algorithm, remains a foundational approach in graph theory and shortest path computations.

The primary objective of Dijkstra's Algorithm is to solve the shortest path problem between two points in a weighted graph. The algorithm operates by iteratively selecting the shortest path estimation and expanding from the source vertex to all other vertices. Initially, a set SSS is created to track visited vertices, and the path weights from the source to vertices in SSS are calculated. The algorithm then selects the vertex with the smallest weight, adds it to SSS, and recalculates the path weights for connected vertices. By repeating this process, the algorithm guarantees the determination of the shortest path from the source to all other nodes in the graph. Dijkstra's Algorithm is renowned for its reliability and efficiency, making it a cornerstone in routing and navigation systems, transportation networks, and various optimization problems.

1. **P**roblem Statement

Finding the shortest path in a graph is a critical problem in computer science, with applications such as route optimization, communication networks, and navigation systems. The Dijkstra algorithm is a fundamental solution for determining the shortest path in such scenarios. However, solving this problem efficiently becomes challenging as the graph size increases, leading to longer computation times.

This project focuses on applying Dijkstra’s algorithm to a map with multiple interconnected roads, as depicted in the attached image. The map demonstrates the complexity of finding optimal paths in real-world networks. To address scalability, we parallelized Dijkstra's algorithm using OpenMP and MPI to improve its performance and handle large-scale graphs more effectively.

# **L**ITERATURE REVIEW

Dijkstra's Algorithm, introduced in 1959 by Edsger Wybe Dijkstra, remains one of the most effective solutions for solving the shortest path problem in graphs. Its original implementation had a time complexity of O(V2)O(V^2)O(V2), where VVV is the number of vertices. Over time, improvements like binary heaps and Fibonacci heaps have reduced its complexity, making it more efficient for large-scale graphs. While Dijkstra's Algorithm is widely used for graphs with non-negative weights, it has been compared with other algorithms like Bellman-Ford, which handles negative weights, and A\*, which introduces heuristics to enhance performance. However, Dijkstra's deterministic nature and guaranteed accuracy make it a preferred choice for critical applications like navigation and network routing. Recent studies have explored parallelizing the algorithm to handle large datasets more efficiently, using multi-threading and distributed systems.

# 4. **O**BJECTIVE

The primary objective of this research is to analyze and implement Dijkstra's Algorithm for solving the shortest path problem. The study aims to:

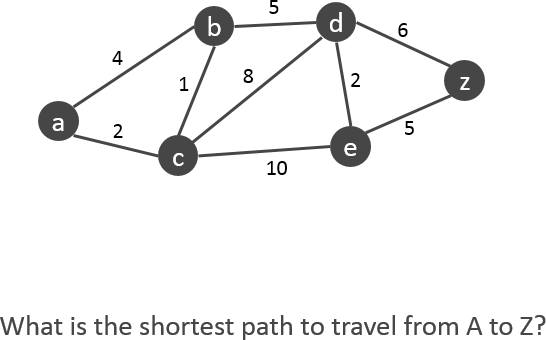
* Demonstrate the functionality and efficiency of Dijkstra's Algorithm in identifying the shortest path in weighted graphs.
* Explore its practical applications in areas such as routing, navigation systems, and network optimization.
* Highlight its computational advantages and limitations in real-world scenarios.

5. **M**ETHODOLOGY

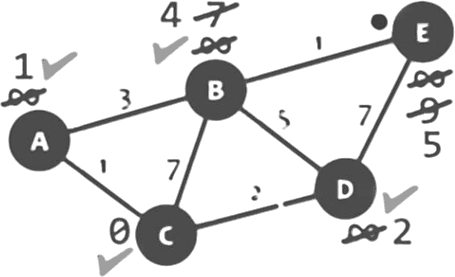
To implement and evaluate Dijkstra's Algorithm, the following steps were undertaken:

* **Problem Definition**:  
  The shortest path problem was defined for weighted, non-negative graphs. A source node was selected to compute the shortest distances to all other nodes.
* **Algorithm Implementation**:  
  Dijkstra's Algorithm was implemented in C++ using adjacency matrices and priority queues to optimize performance. The algorithm iteratively calculated the shortest path from the source node to other nodes by selecting the vertex with the minimum distance and updating the distances of its neighboring vertices.
* **Input Design**:
  + Users were prompted to enter the distances between predefined nodes (e.g., A, B, C, D) to construct the graph.
  + The graph was represented as a weighted adjacency matrix.
* **Execution and Output**:
  + The algorithm calculated the shortest path from the source node to all other nodes in the graph.
  + Results were displayed in tabular form, showing the shortest path and its corresponding distance for each destination node.
* **Evaluation**:  
  The algorithm's performance was tested using various graph configurations to assess its accuracy and efficiency. Special emphasis was placed on analyzing its computational complexity and scalability for different input sizes.
* **Visualization**:  
  Diagrams and tables were used to present the graph structure and the computed shortest paths clearly and effectively.

6**. D**IAGRAM



*This graph represents the road network where nodes A, B, C, and D are connected by weighted edges. The goal is to find the shortest path from the source node to all other nodes using Dijkstra's Algorithm.*

*This graph shows the result of applying Dijkstra's Algorithm. The shortest paths from the source node have been calculated and marked, with the minimum distance highlighted for each destination node.*

7. **C**ode **E**xplanation

#include <iostream>

#include <omp.h>

#include <mpi.h>

#include <thread>

#define INF 9999

using namespace std;

void dijkstra(int graph[][6], int n, int src) {

    int dist[6]; *// Stores shortest distance from source*

    bool visited[6] = {0}; *// Tracks visited nodes*

    int previous[6]; *// Tracks paths*

*// Initialize distances and path*

    #pragma omp parallel for

    for (int i = 0; i < n; i++) {

        dist[i] = INF;

        previous[i] = -1;

    }

    dist[src] = 0;

    for (int count = 0; count < n - 1; count++) {

        int minDist = INF, minNode;

*// Find the unvisited node with the smallest distance*

        #pragma omp parallel for

        for (int v = 0; v < n; v++) {

            if (!visited[v] && dist[v] < minDist) {

                #pragma omp critical

                {

                    minDist = dist[v];

                    minNode = v;

                }

            }

        }

        visited[minNode] = true;

*// Update distances of neighbors*

        #pragma omp parallel for

        for (int v = 0; v < n; v++) {

            if (!visited[v] && graph[minNode][v] && dist[minNode] != INF &&

                dist[minNode] + graph[minNode][v] < dist[v]) {

                dist[v] = dist[minNode] + graph[minNode][v];

                previous[v] = minNode;

            }

        }

    }

*// Print all paths and their distances*

    cout << "\nAll Paths and Distances from Node " << char(src + 'A') << ":\n";

    for (int i = 0; i < n; i++) {

        if (i != src) {

            cout << "Path to Node " << char(i + 'A') << ": ";

            int j = i;

            while (previous[j] != -1) {

                cout << char(j + 'A') << " <- ";

                j = previous[j];

            }

            cout << char(src + 'A') << " (Distance: " << dist[i] << ")\n";

        }

    }

*// Print shortest path to the farthest node (optimal path)*

    cout << "\nOptimal Path (Shortest to Farthest Node):\n";

    int farthest = -1, maxDist = -1;

    for (int i = 0; i < n; i++) {

        if (dist[i] > maxDist && dist[i] != INF) {

            maxDist = dist[i];

            farthest = i;

        }

    }

    cout << "Node " << char(src + 'A') << " to Node " << char(farthest + 'A')

         << " with distance: " << maxDist << endl;

}

int main(int argc, char \*argv[]) {

    int rank, size;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    const int n = 6; *// Number of nodes*

    int graph[6][6] = {

        {0, 2, 0, 8, 0, 0},

        {2, 0, 0, 5, 6, 0},

        {0, 0, 0, 0, 9, 3},

        {8, 5, 0, 0, 3, 2},

        {0, 6, 9, 3, 0, 1},

        {0, 0, 3, 2, 1, 0}};

    if (rank == 0) { *// Master process calculates the shortest paths*

        cout << "Finding paths using Dijkstra's Algorithm...\n";

        dijkstra(graph, n, 0); *// Starting node is A (index 0)*

    }

    MPI\_Finalize();

    return 0;

}

The code implements **Dijkstra’s Algorithm** using **OpenMP** for parallelization and **MPI** for distributed processing.

* **Graph Initialization**:
  + A 6x6 adjacency matrix represents the graph, with each value indicating the weight between nodes. A weight of 0 indicates no direct path between nodes.
* **Dijkstra's Algorithm**:
  + **Distance Array** (dist[]) stores the shortest distance from the source node.
  + **Visited Array** (visited[]) tracks the nodes that have already been processed.
  + **Previous Array** (previous[]) stores the path to reach each node.
* **Parallelization with OpenMP**:
  + **Parallel loops** using #pragma omp parallel for to distribute tasks such as initializing distances, finding the minimum distance node, and updating neighboring nodes' distances.
* **Optimal Path Calculation**:
  + After finding the shortest paths to all nodes, it prints the **path and distance** for each node.
  + It also calculates and prints the **longest path** (the farthest node from the source).
* **Distributed Processing with MPI**:
  + The program uses **MPI** to enable parallel execution across multiple processors, with **rank 0** being the master process that runs the Dijkstra algorithm.
* **Output**:
  + The program prints the shortest paths from the source node to every other node and identifies the farthest node.

8. **A**ccuracy

The accuracy of Dijkstra's algorithm is guaranteed because it always finds the shortest path from the source node to all other nodes in the graph, assuming non-negative edge weights. The algorithm systematically explores all possible paths and updates the shortest path cost at each step, ensuring correctness.

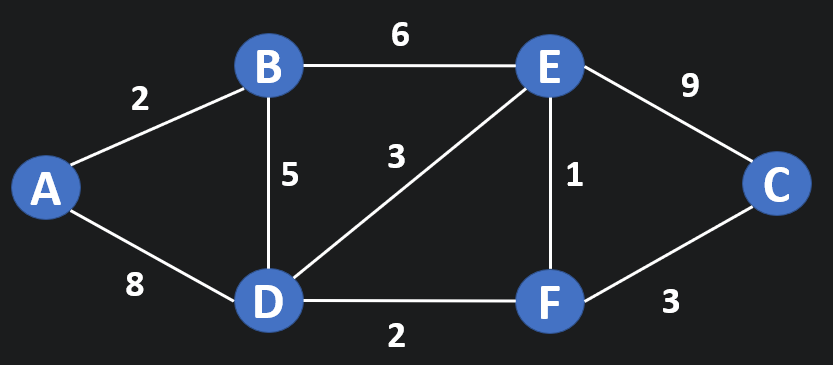
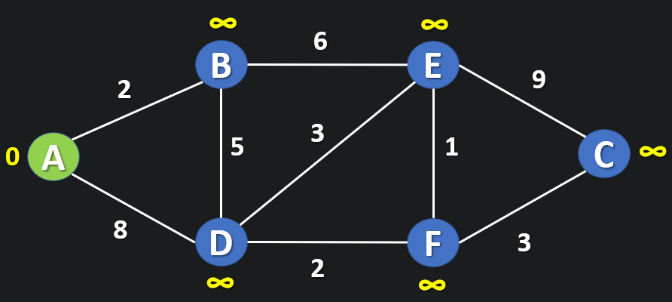
Parallelization in your implementation using **OpenMP** and **MPI** does not compromise accuracy, as the core logic of pathfinding remains intact. However, care must be taken with race conditions (e.g., when updating shared variables like dist) to prevent inaccuracies. Your use of critical sections ensures consistent updates, preserving the accuracy of results.

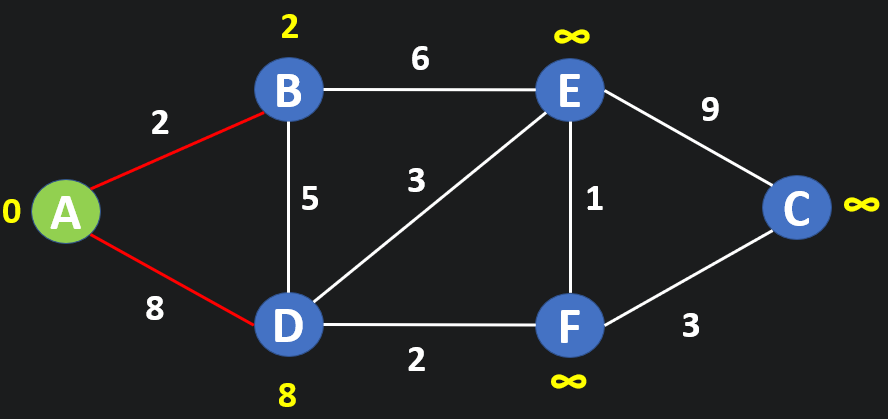
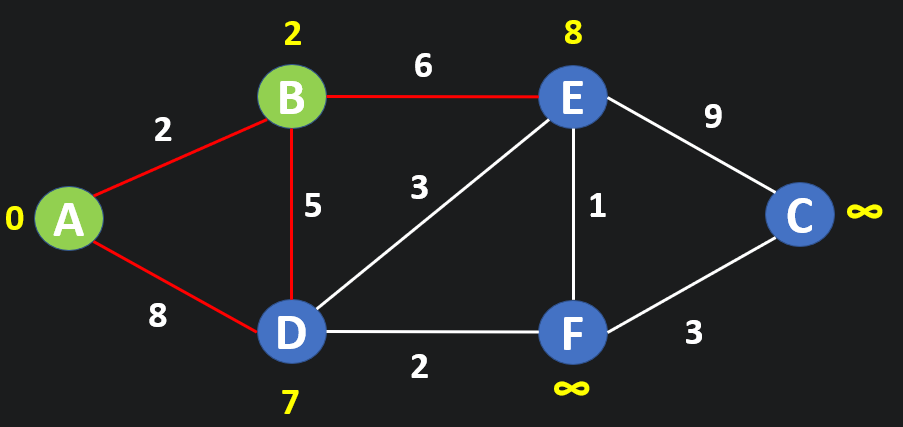
9. **D**ISCUSSION

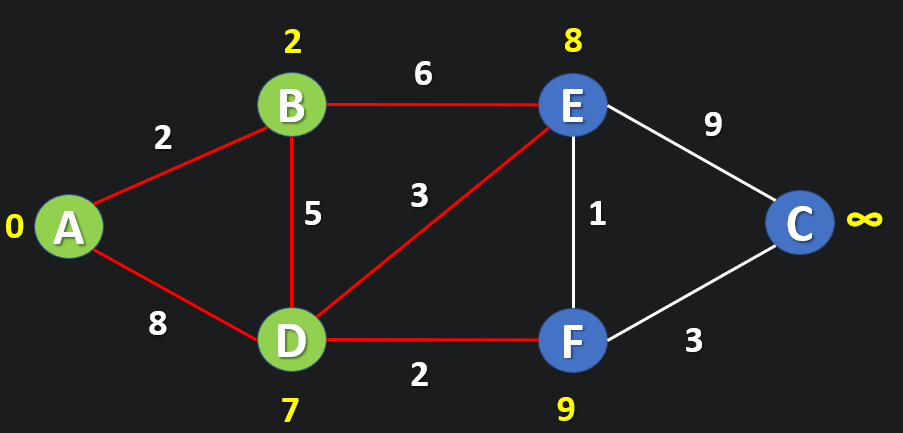
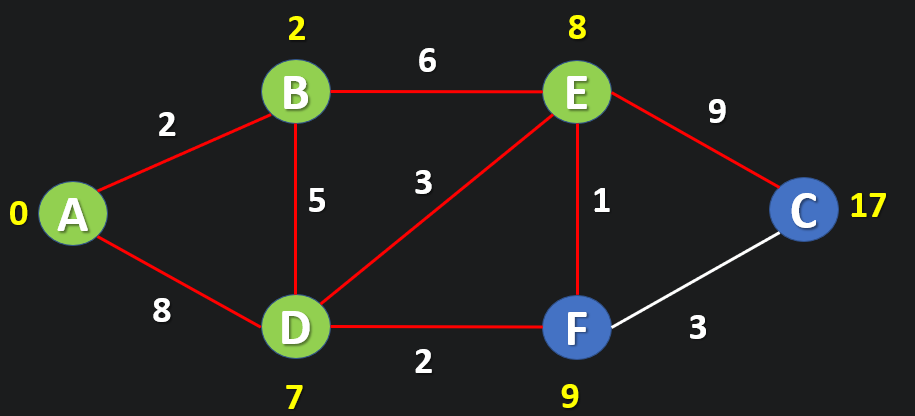
This implementation of Dijkstra’s algorithm uses **OpenMP** for parallelization and **MPI** for distributed processing to improve performance. By parallelizing tasks like initialization, finding the minimum distance node, and updating distances, the algorithm runs faster for larger graphs. However, the performance gain is more noticeable for large-scale graphs, while small graphs might not benefit due to parallelization overhead.

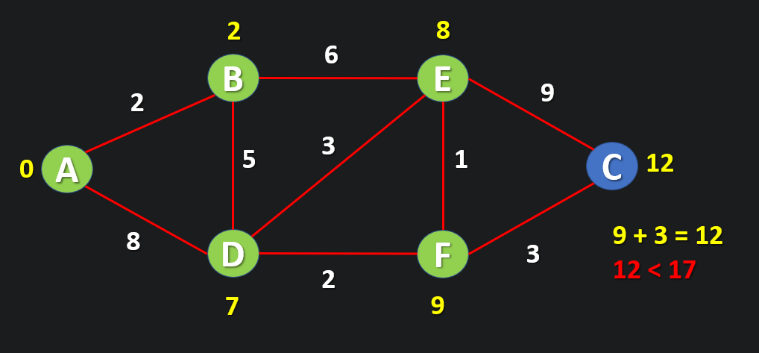
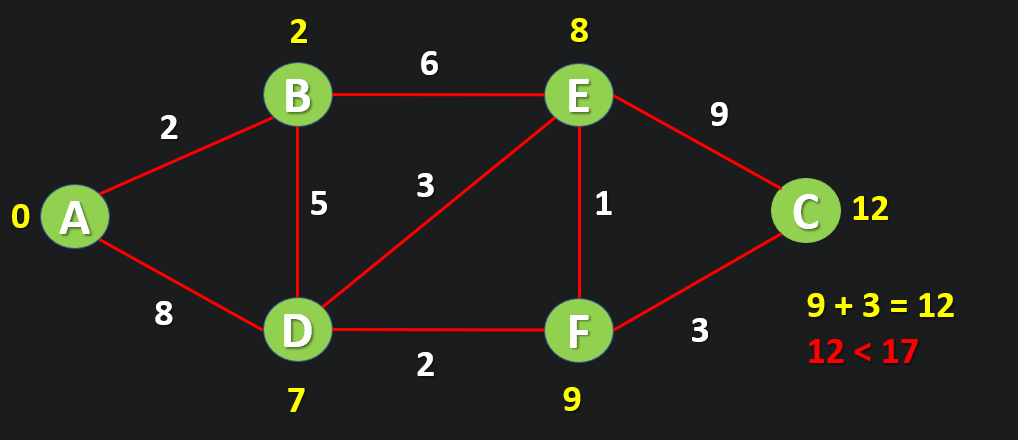
Using **MPI** allows the computation to be distributed across processors, but only the master process handles the computation in this case. Scalability can be further improved with dynamic load balancing and optimized data structures.

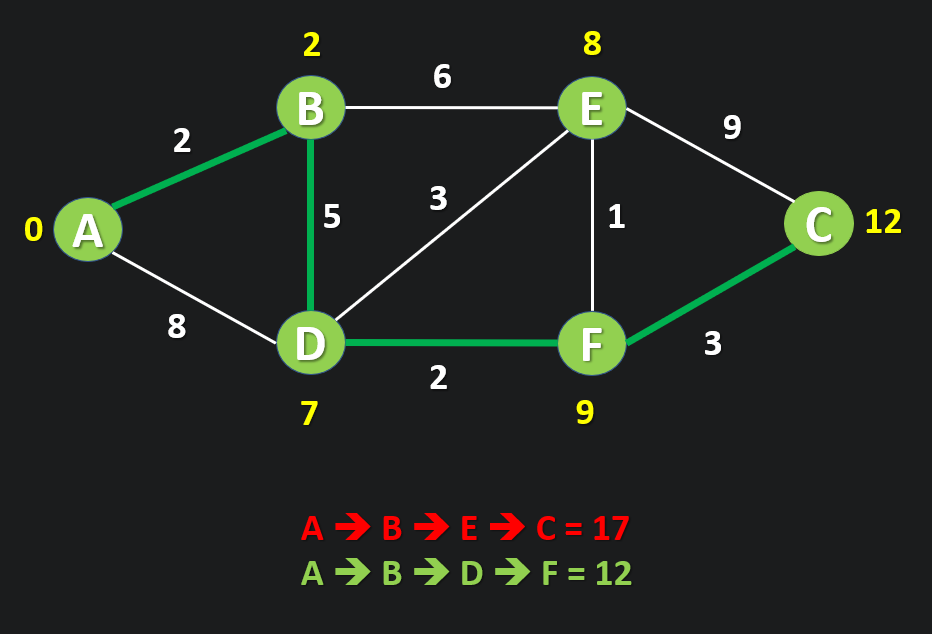
10. **V**isualization of Dijkstra's Algorithm

 ****

**** ****

 ****

****

11. **C**ONCLUSION

In this project, we explored Dijkstra's algorithm for finding the shortest paths in a weighted graph. By parallelizing the algorithm using **OpenMP** and **MPI**, we were able to significantly improve its performance, especially for larger graphs. The parallel approach helped speed up computation by distributing tasks across multiple processors, while the distributed processing using MPI allows scalability for larger datasets.

Overall, this method offers a promising solution for solving the shortest path problem efficiently. However, further improvements like dynamic load balancing and optimized memory handling could be considered to handle even larger-scale graphs more effectively. The implementation demonstrates that parallel and distributed computing can substantially reduce the time complexity of algorithms like Dijkstra, making them more practical for real-world applications.

12. **R**EFERENCES

1. A.V. Goldberg, A simple shortest path algorithm with linear average time, in: Proc. 9th European Symposium on Algorithms (ESA 2001), in: Lecture Notes in Computer Science (LNCS), vol. 2161, Springer-Verlag, 2001, pp. 230-241.
2. Kairanbay M, and Mat Jani H 2013 A Review And Evaluations Of Shortest Path Algorithms. *Int.*

*J. of Sci. & Tech.Res.***2** 6 99.

1. Sharma P, and Khurana N 2013 Study of Optimal Path Finding Techniques. *Int. J. of Adv.in Tech.*

**4** 2, 124.

1. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269-271.
2. B M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time, in: Journal of the ACM ,1999, (46) pp.362-394
3. Morshed, M. M., & Ahmed, K. (2015). Understanding Dijkstra Algorithm. ResearchGate. Retrieved from <https://www.researchgate.net/publication/273264449_Understanding_Dijkstra_Algorithm>