**BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY),**

**COLLEGE OF ENGINEERING ,**

**PUNE- 411043**

**Department of “Information Technology”**



**A**

**Project Based Learning**

**On**

**“Shortest path finder Using Dijkstra's algorithm”**

**Submitted By**

**Sarthak Srivastava (37)** - **2314110314**

**Shashank Kumar (50)** - **2314110330**

**Rushikesh Girpunje (56)** - **2314110678**

**Rahul Kumar (66)** – **2414111034**

**Under the guidance of**

**PROF. LAXMI PAWAR**

**Department of “Information Technology”**

**BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)**

**COLLEGE OF ENGINEERING, PUNE- 43**



**CERTIFICATE**

This is to certify that the Project Based Learning report titled ‘Shortest path finder Using Dijkstra's Algorithm’, submitted by RUSHIKESH GIRPUNJE (2314110678), RAHUL KUMAR (2414111034), SARTHAK SRIVASTAVA (23141110314), SHASHANK KUMAR (2314110330), to the Bharati Vidyapeeth (Deemed to be University), College of Engineering, Pune for the award of the degree of BACHELOR OF TECHNOLOGY in Information Technology is a Bonafide record of the PBL work done by them under my supervision**.**

PROF. LAXMI PAWAR

Name of Subject Teacher

Place: Pune

Date: 10-10-2024

# GROUP MEMBERS

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Roll No. | Name | PRN No. |
| 1 | 37 | Sarthak Srivastava | 2314110314 |
| 2 | 50 | Shashank Kumar | 2314110330 |
| 3 | 56 | Rushikesh Girpunje | 2314110678 |
| 4 | 66 | Rahul Kumar | 2414111034 |

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Name of Chapter** | **Page No.** |
| 1 | ACKNOWLEDGEMENT | 5 |
| 2 | INTRODUCTION | 6-7 |
| 3 | BASIC STRUCTURE AND COMPONENTS | 7 |
| 4 | EXPRESSION EVALUATION PROCESS | 8 |
| 5 | IMPORTANCE OF OPERATOR PRECEDENCE and ASSOCIATIVITY | 9-14 |
| 6 | DATA STRUCTURES USED IN EXPRESSION EVALUATION | 15-17 |
| 7 | APPLICATIONS IN REAL WORLD | 18-19 |
| 8 | SNIPS AND OUTPUT CODE | 20-22 |
| 9 | CONCLUSION | 23 |

# ACKNOWLEDGEMENT

We would like to express deepest appreciation towards Dr Vidula Sohoni, Principal, Bharati Vidyapeeth (Deemed to Be University) College of Engineering, Pune, India and Dr.Sandeep Vanjale Head of the Department (IT), who invaluable supported us in completing this project. We are profoundly grateful to Prof. Laxmi Pawar, Project guide for her expert guidance and continuous encouragement throughout to see that this project rights, its target since its commencement to its completion. At last we must express our sincere heartfelt gratitude to all the staff members of IT Department who helped us directly or indirectly during this course of work.

RUSHIKESH GIRPUNJE

RAHUL KUMAR

SARTHAK SRIVASTAVA

SHASHANK KUMAR

IT (Semester – III)

INTRODUCTION

Graph algorithms are essential in solving complex problems across various fields such as computer networks, transportation, and geographical information systems. Among these, **Dijkstra's algorithm** is a widely-used method for finding the shortest path in a weighted graph, making it invaluable for tasks like network routing and optimization.

This project report presents the implementation of a **Shortest Path Finder** using Dijkstra's algorithm in **C++**. The project aims to solve the single-source shortest path problem, allowing users to input nodes and calculate the shortest distance between them in a graph. By representing locations as nodes and paths as weighted edges, the algorithm efficiently computes the shortest path from a starting node to any target node.

The report discusses the theory behind Dijkstra's algorithm, its implementation in C++, and real-world applications. It highlights the importance of graph algorithms in addressing complex optimization tasks, offering insights into the broader field of graph theory. **[1]**



# PROJECT OVERVIEW

The **Shortest Path Finder** project implements a weighted, undirected graph where each edge represents a possible route between two nodes, and the edge weights represent the cost (or distance) of traveling between these nodes. The project is designed to calculate the shortest path between two user-specified nodes using Dijkstra's algorithm, a greedy algorithm that computes the shortest path from a source node to all other nodes in the graph. [2]

This project covers the following aspects:

* **Graph Representation**: Implemented using an adjacency list.
* **Algorithm**: Dijkstra’s algorithm for shortest path calculation.
* **Input Validation**: Ensures valid input from the user.
* **Graph Visualization**: Displays the graph structure to the user.
* **Output**: Presents the shortest path and the total distance.



# Theoretical Background

Graphs are essential mathematical structures used to model connections between objects, consisting of nodes (vertices) and edges (connections). In weighted graphs, edges have weights that represent cost, distance, or time. Such graphs are widely used in fields like transportation, networking, and geographic mapping, where finding the most efficient path is crucial.

This project addresses the shortest path problem using Dijkstra’s algorithm, which works on graphs with non-negative weights. Dijkstra's algorithm efficiently computes the shortest path from a starting node to all other nodes by selecting the nearest unexplored node and updating distance estimates. Its efficiency makes it ideal for applications like route planning and network routing, especially in large, sparse graphs.

The project demonstrates how Dijkstra’s algorithm can be applied to solve practical pathfinding problems, providing optimal solutions for minimizing traversal costs like time or resources. [3]

# Problem Statement

The *Shortest Path Finder* project addresses a key problem in graph theory: finding the shortest path between two specified nodes in a weighted, undirected graph. This has significant applications in fields like transportation systems, computer networking, and geographic information systems (GIS), where determining the most efficient route between two points is crucial for minimizing time, cost, or resources.

# Implementation Details

The project was developed using C++ and the Standard Template Library (STL) for managing data structures such as vectors, unordered maps, and priority queues. Below are the major components of the implementation. [4]

**5.1 Graph Representation**

The graph is represented using an **adjacency list**, which maps each node to a list of its neighbors and the associated edge weights. The Graph class encapsulates the graph structure and provides methods for adding edges, retrieving neighbors, and checking node existence.

**5.2 Dijkstra's Algorithm**

The **Dijkstra** function is responsible for computing the shortest path from the source node to all other nodes. It uses a **min-heap (priority queue)** to ensure that the next node processed is always the one with the smallest tentative distance. [2][5]

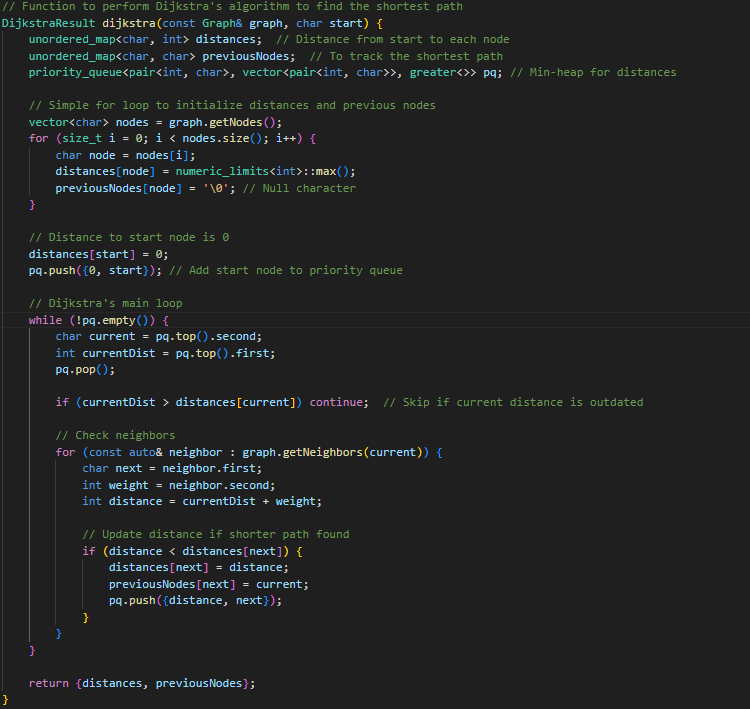
**5.3 Path Reconstruction**

Once the shortest path is calculated, the program reconstructs the path from the destination back to the source using a previousNodes map. The reconstructed path is then reversed and displayed to the user.

**5.4 Input Validation**

To ensure that the user inputs valid nodes, the program checks whether the input is a valid alphabet character (A-F) and if the node exists in the graph. [4]

**5.5 Key Code Snippets**

****

[5]

**Algorithm Analysis**

**6.1 Time Complexity**

The time complexity of Dijkstra's algorithm is **O((V + E) log V)**, where V is the number of vertices and E is the number of edges. This complexity arises from the need to process each vertex and update the distances for each edge in the graph.

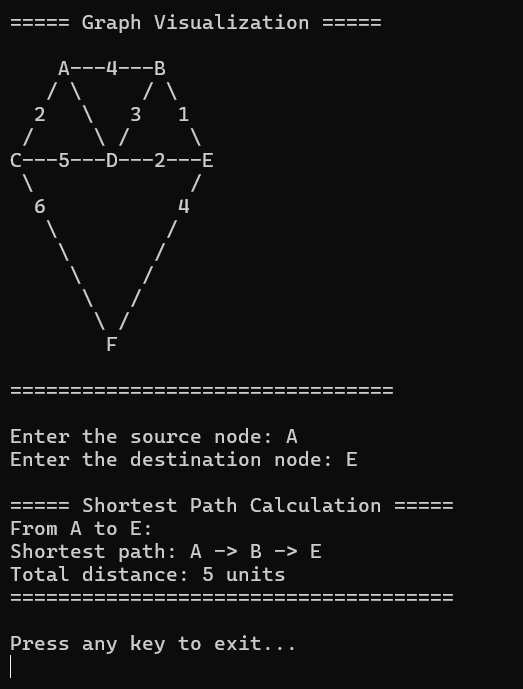
**6.2 Space Complexity**

The space complexity is **O(V + E)** due to the adjacency list representation and the additional storage required for the priority queue and distance arrays. [6]



**User Interface**

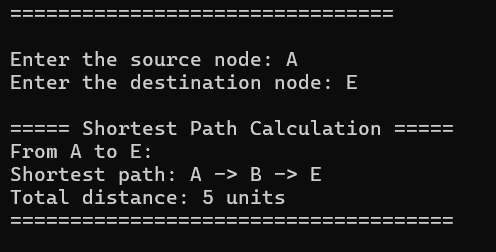
The user interface is simple and text-based. After displaying the graph structure, the program prompts the user for the source and destination nodes. It validates the input to ensure that it is a valid node in the graph. Upon successful input, the program computes and displays the shortest path. [7]



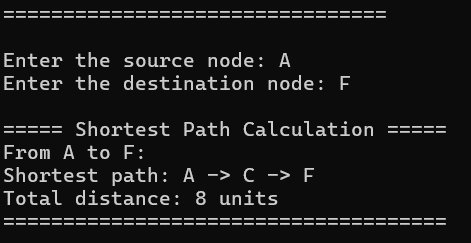
**Testing and Results**

The project was tested with multiple combinations of source and destination nodes to ensure the correctness of the output. Below are some sample test cases:

**Test Case 1:**



**Test Case 2:**



**Conclusion**

The **Shortest Path Finder** project effectively demonstrates the implementation and practical application of **Dijkstra's algorithm** in solving the shortest path problem within a weighted, undirected graph. Several key aspects of the project contribute to its success:

1. **Efficient Pathfinding**: By utilizing Dijkstra’s algorithm, the project accurately computes the shortest path between two nodes, ensuring optimality in the solution for graphs with non-negative weights.
2. **Clear Output**: The program provides a detailed output, displaying not only the shortest path but also the total cost or distance of that path, offering users a comprehensive understanding of the results.
3. **User Interactivity**: The project allows users to input custom start and end nodes, enhancing the flexibility and real-world applicability of the solution in different scenarios, such as route planning or network optimization.
4. **Predefined Graph Structure**: The graph, consisting of six nodes (A, B, C, D, E, and F) and weighted edges, simulates real-world connections, making the problem tangible and relatable to users.
5. **Use of Modern C++**: By employing modern C++ techniques, such as efficient data structures (e.g., priority queues and adjacency lists), the project achieves not only clarity in code but also optimal performance in processing the graph and calculating shortest paths.
6. **Scalability**: Although designed for small to medium-sized graphs, the project’s approach lays a solid foundation for scalability and can be extended to handle larger graphs with more nodes and edges.
7. **Practical Relevance**: The project highlights the importance of graph theory and shortest path algorithms in real-world applications, such as navigation systems, communication networks, and resource optimization. Dijkstra’s algorithm, understanding both its theoretical basis and its practical implementation in C++, bridging the gap between academic knowledge and practical problem-solving.

**Future Enhancements**

Although the current implementation meets its objectives, there are several ways to improve and extend the project:

1. **User-Defined Graphs**: Allowing users to input their own graphs dynamically instead of using a predefined graph.
2. **Directed Graphs**: Extending the program to handle directed graphs with one-way edges.
3. **Graphical User Interface (GUI)**: Providing a more intuitive visual representation of the graph and results using graphical elements.
4. **Other Algorithms**: Incorporating other shortest path algorithms like A\* or Bellman-Ford for comparison.
5. **Performance Optimization**: Optimizing the code to handle larger graphs more efficiently. [8]

**References**

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. <https://mitpress.mit.edu/books/introduction-algorithms-third-edition>
2. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*(1), 269-271. <https://link.springer.com/article/10.1007/BF01386390>
3. Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional. <https://algs4.cs.princeton.edu/home/>
4. Boost C++ Libraries - Graph Library. <https://www.boost.org/doc/libs/1_76_0/libs/graph/doc/index.html>
5. GeeksforGeeks - Dijkstra's Shortest Path Algorithm. <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>
6. Skiena, S. S. (2020). *The Algorithm Design Manual* (3rd ed.). Springer. <https://link.springer.com/book/10.1007/978-3-030-54256-6>
7. C++ Reference - Priority Queue. <https://en.cppreference.com/w/cpp/container/priority_queue>
8. Kleinberg, J., & Tardos, E. (2006). *Algorithm Design*. Pearson Education. <https://www.pearson.com/us/higher-education/program/Kleinberg-Algorithm-Design/PGM319216.html>