SPL-1 Project Report, [2022]

Graph Algorithm Implementation & Visualization

SE 305 : Software Project Lab - 1 (SPL-1)

Submitted by

Ibne Bin Rafid

BSSE Roll : 1330

BSSE Session : 2020-21

Supervised by

Kishan Kumar Ganguly

Assistant Professor

Institute of Information Technology (IIT)

Institute of Information Technology (IIT)

University of Dhaka

[24-05-2023]

Project Name : Graph Algorithm Implementation & Visualization

Prepared by :

Ibne Bin Rafid
BSSE 13th batch
Roll : 1330

Supervised by :

Kishan Kumar Ganguly
Assistant Professor
Institute of Information Technology (IIT),
University of Dhaka.

Date of Submission : 24 May, 2023

Supervisor's Approval :

_____

(Signature of Kishan Kumar Ganguly)

Table of Contents

1. Introduction

The purpose of this project was to implement and visualize various graph algorithms. Graph algorithms are fundamental in computer science and have applications in various domains such as network analysis, social networks, route planning, and more. This report provides an overview of the project, including the background study, challenges faced, project overview, user manual, conclusion, and appendix.

1.1.Background Study

To successfully implement the project on Graph Algorithms Implementation and Visualization, a comprehensive background study was conducted on relevant graph algorithms. The following algorithms were studied in detail:

Breadth-First Search (BFS): BFS is another graph traversal algorithm that explores or searches through the nodes of a graph, but in a breadth-first manner. It starts at a specific node and visits all its neighboring nodes before moving to the next level of neighbors. BFS is commonly used to find the shortest path in an unweighted graph and to determine the connectivity of a graph.

Depth-First Search (DFS): DFS is a fundamental graph traversal algorithm used to explore or search through the nodes of a graph in a systematic manner. It is often used as a building block for other graph algorithms and has applications in cycle detection, topological sorting, and connected components.

Dijkstra's Algorithm: Dijkstra's algorithm is a popular shortest path algorithm used to find the shortest path between two nodes in a weighted graph. It operates by iteratively selecting the node with the smallest distance from a source node and updating the distances of its neighboring nodes. Dijkstra's algorithm guarantees finding the shortest path if all edge weights are non-negative.

Bellman-Ford Algorithm: The Bellman-Ford algorithm is another shortest path algorithm that handles graphs with negative edge weights. It iterates over all edges and relaxes them repeatedly until the optimal distances are obtained. Bellman-Ford can detect negative cycles and is often used in scenarios where negative edge weights are present.

Johnson's Algorithm: Johnson's algorithm is a combination of Dijkstra's algorithm and Bellman-Ford algorithm used to find the shortest paths between all pairs of nodes in a graph, even in the presence of negative edge weights. It involves adding a virtual source node and modifying the

edge weights using a potential function. Johnson's algorithm is advantageous when dealing with graphs with both positive and negative edge weights.

Shortest Path A* Algorithm: The A* algorithm is a popular pathfinding algorithm used to find the shortest path between two nodes in a weighted graph. It combines the advantages of Dijkstra's algorithm and heuristics to guide the search towards the goal node efficiently. A* uses an evaluation function that estimates the cost of reaching the goal through a specific node, allowing it to prioritize nodes that are likely to lead to the shortest path. The algorithm guarantees optimal paths if the heuristic function satisfies certain conditions.

Minimum Spanning Tree (MST) - Prim's Algorithm: Prim's algorithm is a greedy algorithm used to find the minimum spanning tree in a connected, undirected graph. It starts with an arbitrary node and iteratively adds the minimum-weight edge that connects a visited node to an unvisited node. Prim's algorithm ensures that the resulting tree spans all the nodes with the minimum total weight.

Minimum Spanning Tree (MST) - Kruskal's Algorithm: Kruskal's algorithm is another greedy algorithm used to find the minimum spanning tree in a connected, undirected graph. It initially treats each node as a separate component and iteratively selects the minimum-weight edges, combining components until all nodes are part of a single tree. Kruskal's algorithm guarantees the creation of a minimum spanning tree and can handle disconnected graphs.

Strongly Connected Components (SCC) - Tarjan's Algorithm: Tarjan's algorithm is a graph traversal algorithm used to find strongly connected components (SCCs) in a directed graph. A strongly connected component is a subgraph where there is a directed path between any two nodes in the component.

The algorithm is based on depth-first search (DFS) and uses a stack to keep track of visited nodes. It assigns a unique discovery time and low-link value to each node during the DFS traversal. The discovery time represents the order in which nodes are visited, while the low-link value indicates the earliest node reachable from the current node. Tarjan's algorithm assigns a unique component identifier to each strongly connected component, allowing for efficient identification and analysis of interconnected regions within the graph.

Graph Isomorphism - VF2 Algorithm: VF2 algorithm is a graph matching algorithm used to determine whether two given graphs are isomorphic, i.e., if they have the same structure. The

algorithm takes two input graphs and tries to find a mapping between their nodes that preserves their adjacency relationships.

VF2 stands for "viable and feasible" and is based on the concept of backtracking search. The algorithm uses a state space search technique, exploring the possible mappings between the node of the two graphs. VF2 algorithm is efficient in handling large graphs and has applications in graph database matching and pattern recognition.

The study involved understanding the core concepts, working principles, time and space complexities, and applications of these algorithms. It also included exploring various data structures such as adjacency lists, adjacency matrices, priority queues, and stack/queue data structures to efficiently represent and manipulate graphs.

1.2. Challenges

The implementation of graph algorithms presented several challenges, including:

- **Developing an understandable visualization system to enhance user understanding:**
  It was most challenging part for me in the whole project to take it in a good shape using good visualization system. Connecting different graph algorithms in graphical window with different advantageous buttons like clicking in a button it will show the step by step visualization, clicking in a button it will take you in the previous window etc. took a lot time to understand and implement.
- **Understanding the intricacies of each algorithm and their underlying concepts:**
  It was quite challenging to understand and remember different graph algorithms altogether, as we know different graph algorithms have different grasping concepts. Furthermore, each graph algorithm has its own set of requirements and constraints, making it challenging to grasp and remember all of them simultaneously. Understanding the intricacies of different graph algorithms, such as breadth-first search (BFS), depth-first search (DFS), Dijkstra's algorithm, Bellman-Ford algorithm, Prim's algorithm, Kruskal's algorithm, and many more, required dedicated effort and practice.
- **Handling different graphs and try to optimizing their performance:**
  Graphs vary in size and connectivity, which poses me to face challenge in handling different graph types efficiently. Adaptation of the data structures and algorithms based on the characteristics of different graphs were difficult.

- **Improving the correctness of the algorithms:**

  Graph algorithms are prone to errors, such as infinite loops, incorrect results, or crashes. Rigorous testing and debugging were essential for me to ensure the correctness of the implemented algorithms. Including edge cases, were helpful to identify and fix bugs in the codes. As there are algorithms like VF2, A*, Tarjans, Johnsons, Fleurys etc. which were tough for me to implement these algorithms in a improved way and then connect with the graphical window.

2. Project Overview

The project on Graph Algorithms Implementation and Visualization aims to develop a comprehensive software solution that enables users to explore and understand various graph algorithms. The project focuses on implementing and visualizing several fundamental graph algorithms, including Depth-First Search (DFS), Breadth-First Search (BFS), Dijkstra's algorithm, Johnson's algorithm, A* algorithm, Prim's algorithm, Kruskal's algorithm, Tarjan's algorithm, and VF2 algorithm.

- The project provides a good interface (Fig-1) that allows users to interact with the implemented algorithms and visualize the results in an intuitive and informative manner. The application supports both directed and undirected graphs, with the flexibility to handle graphs of varying sizes and complexities. But it gives better sense of visualization for small size graphs.

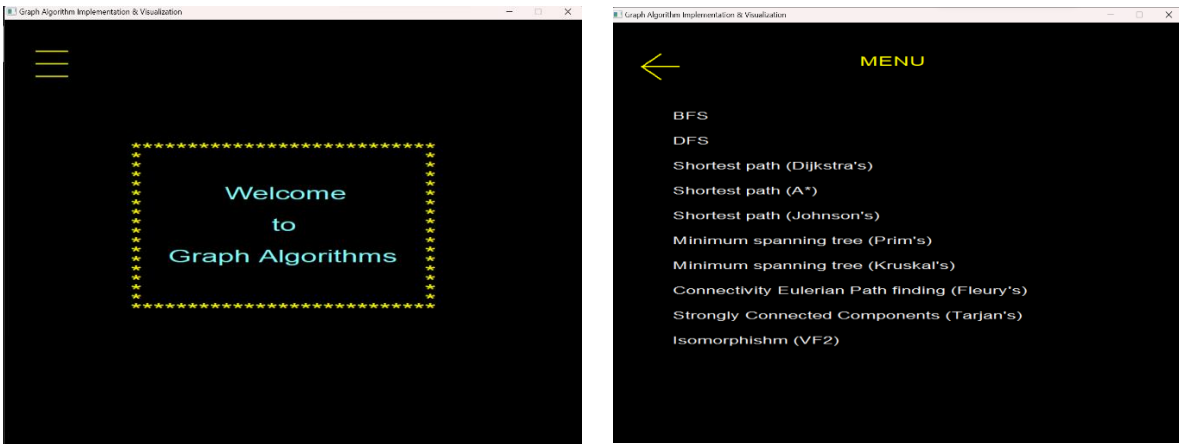

Fig-1

- Users can apply DFS and BFS for graph traversal and can understand how the traversal of these algorithms work. Dijkstra's algorithm and Bellman-Ford algorithm can be utilized to find the shortest paths in weighted graphs, with Dijkstra's algorithm suitable for non-negative

weights and Bellman-Ford algorithm capable of handling graphs with negative edge weights. While Johnson's algorithm handles graphs with both positive and negative edge weights.

- In addition to pathfinding algorithms, the project implements minimum spanning tree algorithms, including Prim's algorithm and Kruskal's algorithm, which can find the minimum cost tree that spans all the nodes in a graph.
- The application also incorporates Tarjan's algorithm for identifying strongly connected components within directed graphs. Moreover, the VF2 algorithm is implemented for graph isomorphism, enabling users to determine whether two graphs have the same structure.
- The project places emphasis on providing a seamless user experience by offering an interactive visualization component. Users can input their own graphs or generate random graphs or use fixed graphs (Fig-2) to apply the implemented algorithms. The visualization component presents the graph and algorithm execution step-by-step, allowing users to observe the algorithm's progress and understand its behavior. Visual cues, such as highlighting visited nodes, shortest paths, or spanning trees, aid in comprehending the algorithm's output and analysis.

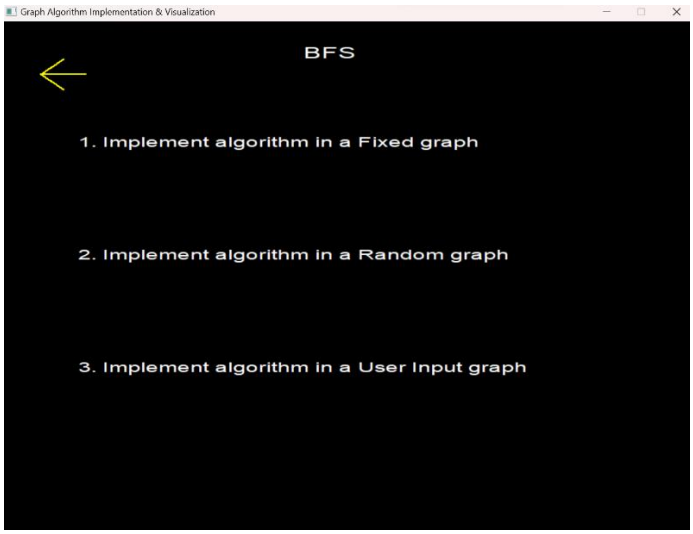

Fig-2

- The visualization interface provides buttons like menu button, previous window button etc. (Fig-3) It helps the graphical window to be more user friendly.

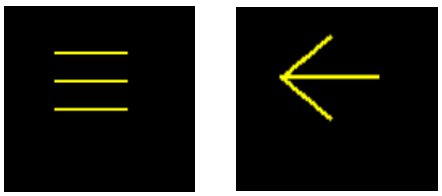

Fig-3

- By implementing and visualizing these fundamental graph algorithms, the project equips users with a valuable tool for learning and exploring graph theory concepts, algorithmic problem-solving techniques, and graph-related applications. It aims to foster a deeper understanding of graph algorithms and their practical implications while providing an engaging and interactive learning experience.
- With its good visualizing interface, well algorithm implementations, and informative visualization capabilities, the project on Graph Algorithms Implementation and Visualization offers a valuable resource for students or anyone interested in exploring the basic of graphs and their algorithms.

3. User Manual

The user manual provides detailed instructions on installing and using the graph algorithm visualization tool. It includes the following parts:

3.1 Necessary Tools

- ➢ Install an Integrated Development Environment (IDE) such as Code::Blocks or Visual Studio Code (VS Code) on your machine.
- ➢ Install a compatible C++ compiler, such as GCC or MinGW, to compile the source code.
- ➢ Install graphics.h library, for the visualization component of the tool.

3.2 Usage Procedure

- ✓ Launch the installed IDE and open the project's source code files. (Fig-4)

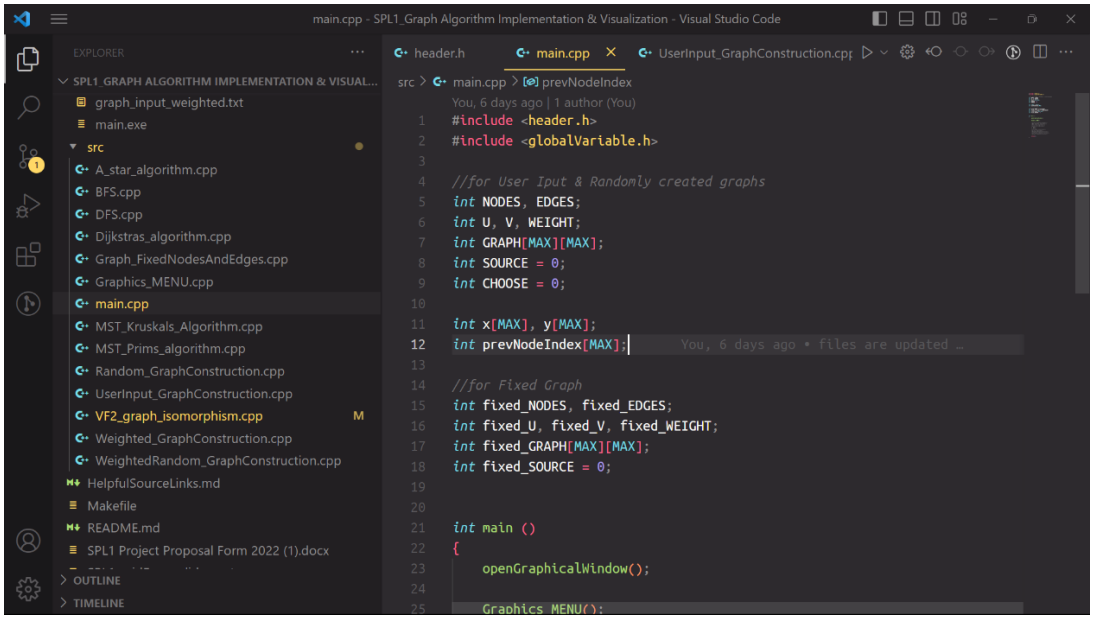

Fig-4

✓ Compile the source code to generate the executable file (main.exe) for the graph algorithm visualization tool. For VS code press Ctrl, Shift, B together. (Fig-5)

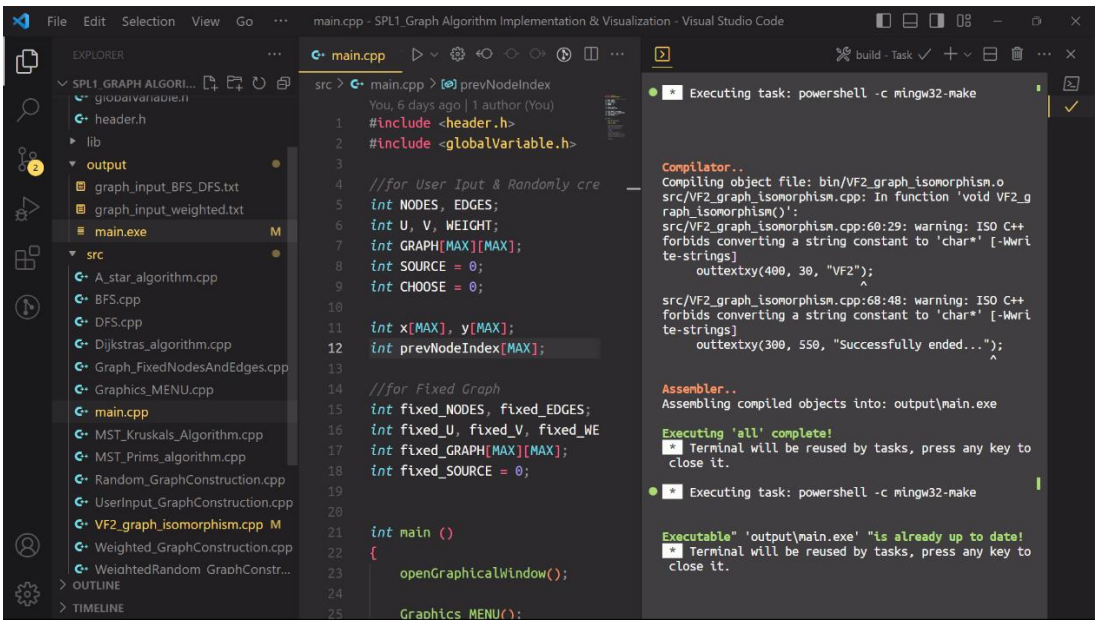

Fig-5

✓ Open a new terminal and run from terminal command. Make sure while you are running, you are in the current folder of the executable file. In this case, it is output folder. (Fig-6)

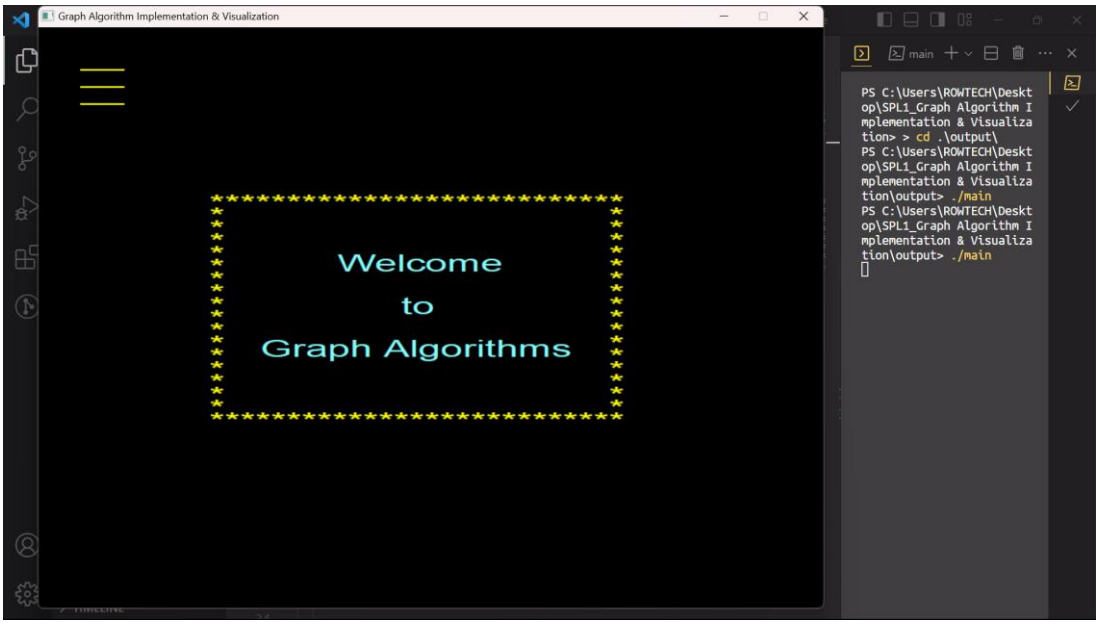

Fig-6

✓ Now, just go through different options and visualize any graph algorithms as you wish. For example:

1. Clicking to the menu button in the left corner of the graphical window it will take to the menu with various graph algorithm names. (Fig-7)

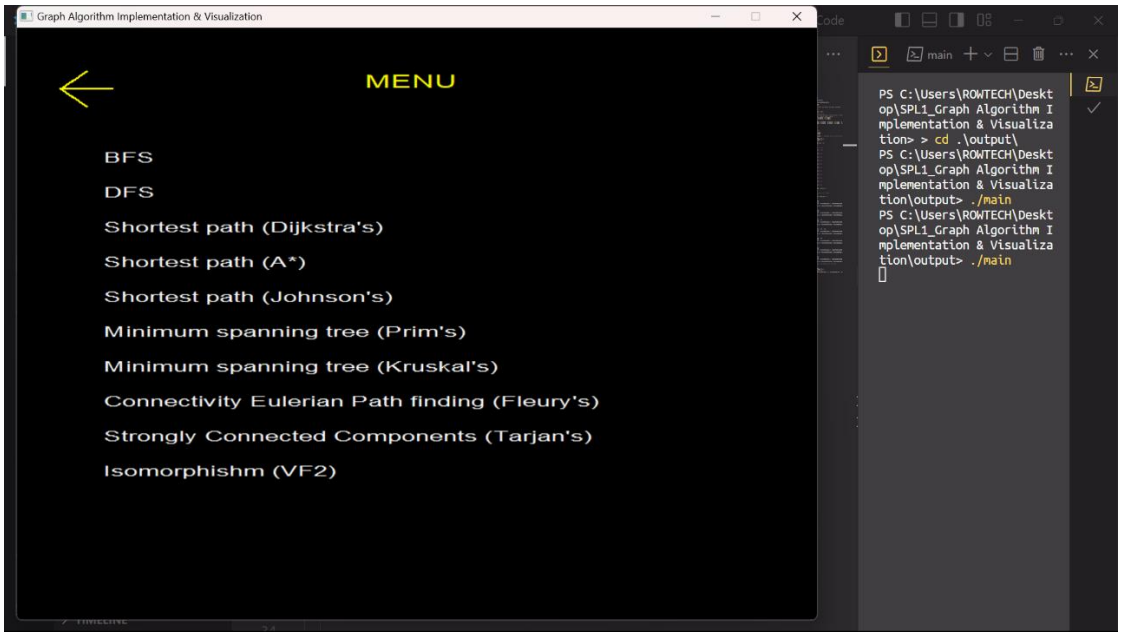

Fig-7

2. Click any of the graph algorithm names to proceed to the next step. (Fig-8)

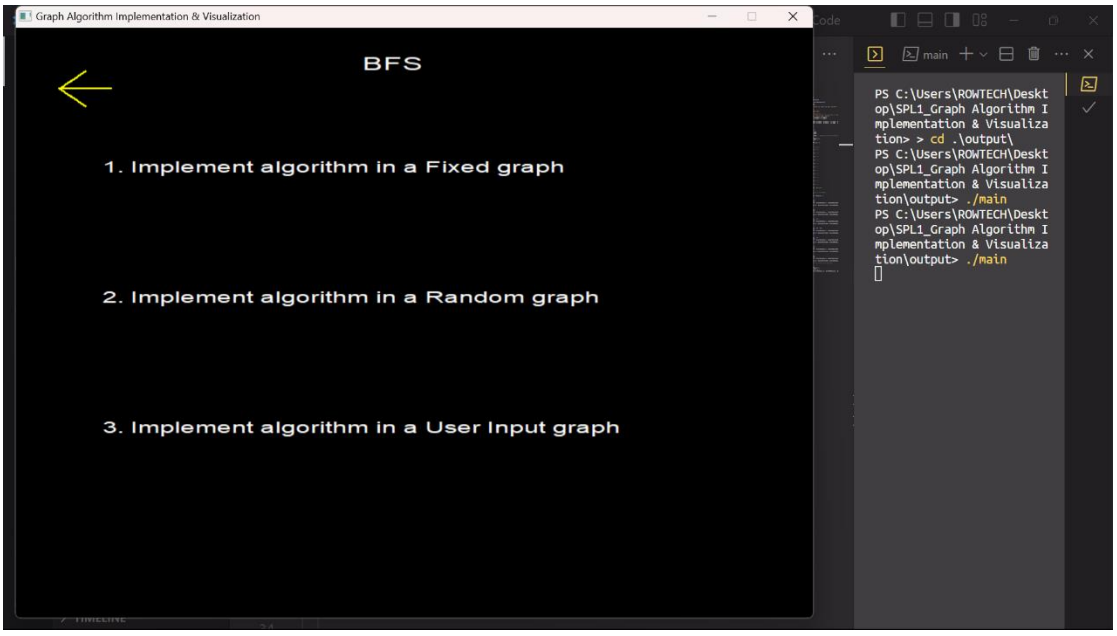

Fig-8

3. Click how you want to create a graph and want to run the algorithm you selected. I have chosen no. 1 here for example. Then, a graph created automatically. (Fig-9)

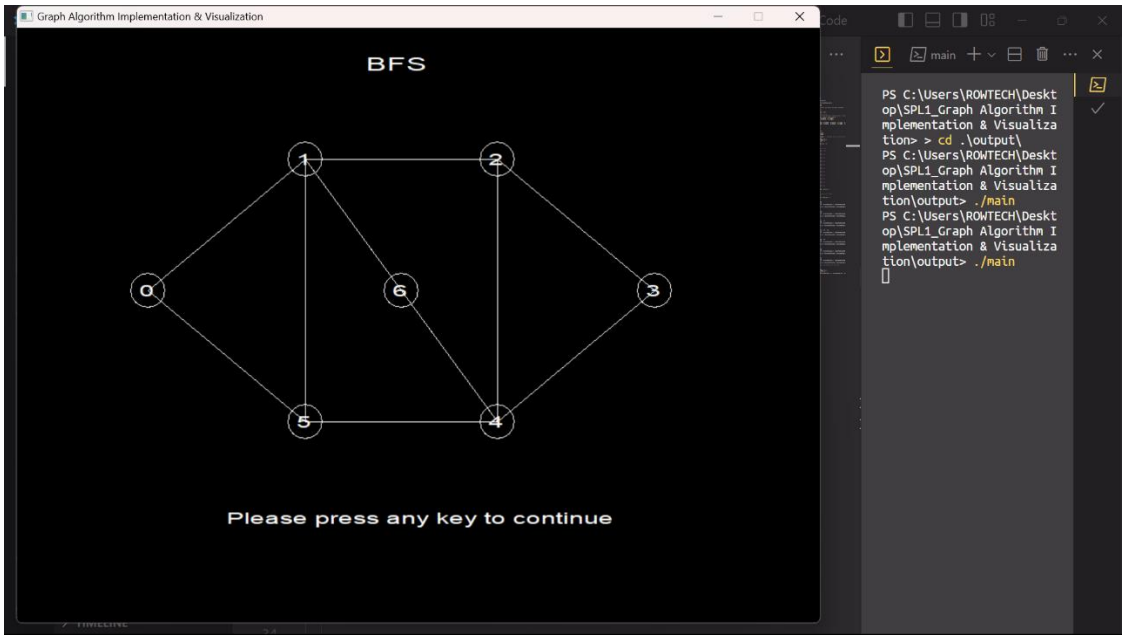

Fig-9

4. It says to enter any key from the key board (Fig-9). When you hit a key, it starts the visualization process. (Fig-10)

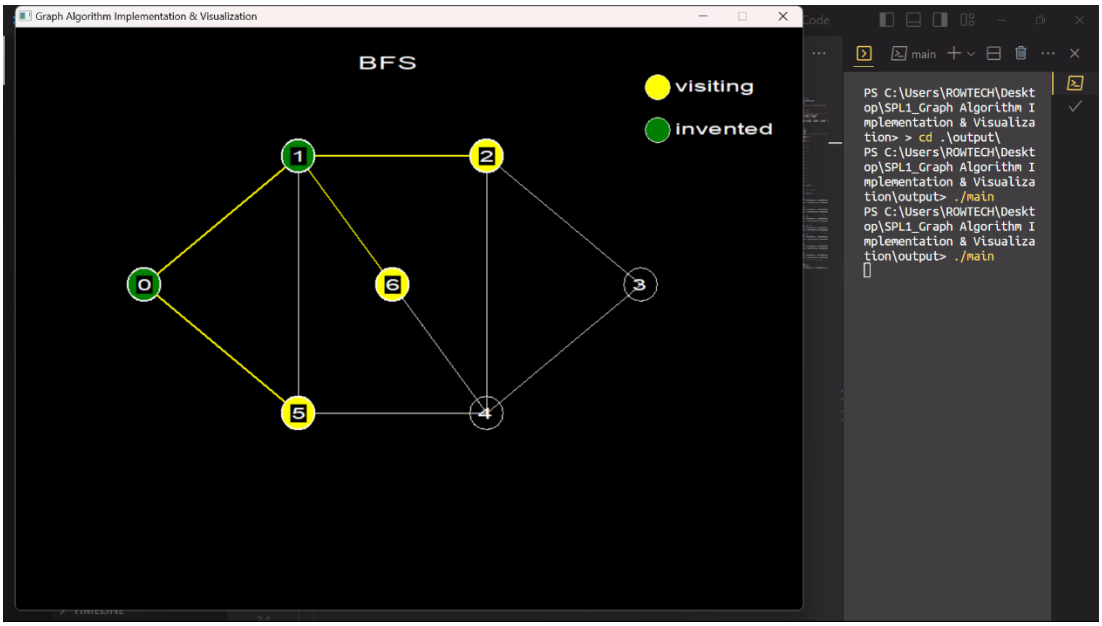

Fig-10

5. It will automatically stop when the visualization process ends. (Fig-11)

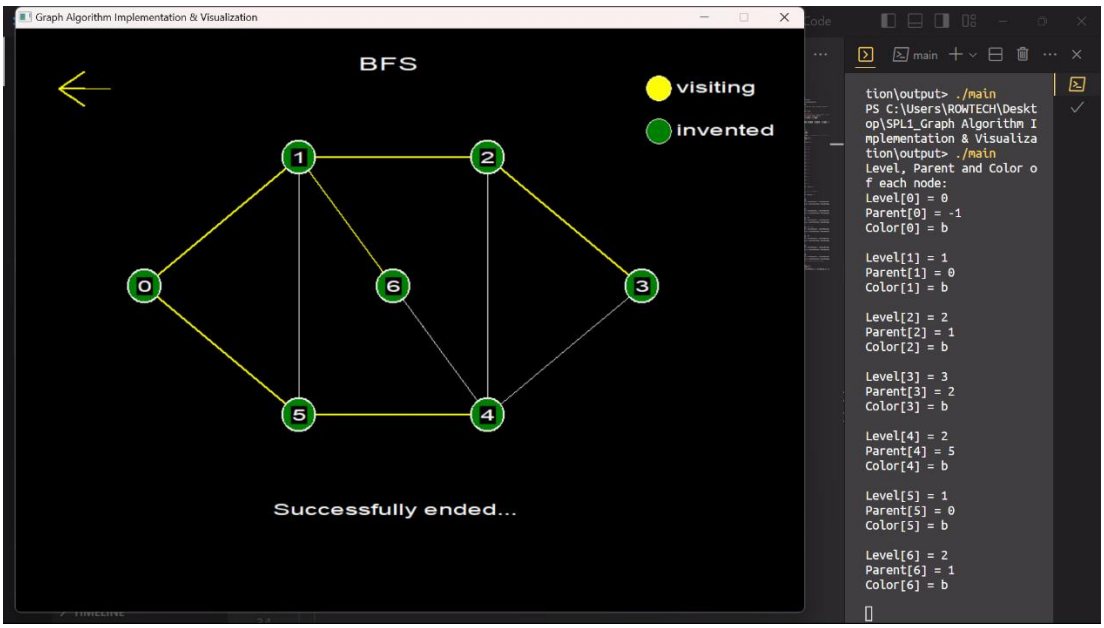

Fig-11

By following the user manual, users will be able to successfully install the required software components, navigate the tool's interface, and effectively utilize the visualization capabilities to explore graph algorithms and their output. The provided screenshots and sample input/output scenarios will assist users in gaining a clear understanding of the tool's functionality and the insights it can offer for graph-related problems.

4. Conclusion

In conclusion, this project successfully implemented and visualized various graph algorithms. Through the project, a deeper understanding of graph theory and algorithmic concepts was gained. The visualization component enhanced the learning experience by providing a graphical representation of the algorithms' execution. The project also addressed the challenges faced during implementation and achieved the objectives set at the beginning.

References

[1] https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/ , Geeks for geeks, 21-05-23

[2] https://www.researchgate.net/publication/2790612_Graph_Algorithms_Iteration_Data_Structures_-_The_Structure_of_Graph_Algorithms_and_a_Corresponding_Style_of_Programming , researchgate.net, 21-05-23

[3] https://www.codewithharry.com/videos/data-structures-and-algorithms-in-hindi-83/ , CodeWithHarry, 21-05-23

[4] https://www.gatevidyalay.com/graph-isomorphism/, gatevidyalay, 21-05-23

[5] https://www.techcrashcourse.com/2015/08/c-program-draw-bar-graph-using-graphics.html?m=1 , teacherscourse, 21-05-23