# PROJECT AND TEAM INFORMATION

## Project Title

|  |
| --- |
| *Graph Traversal Algorithms Visualizer*  *(BFS, DFS, Topological sort, Kruskal and Prim Algorithm, Dijkstra Algorithm, Bellman-Ford Algorithm, Floyd Warshall Algorithm, Disjoint Set Union, Kosaraju’s and Tarjan’s Algorithm)* |

## Student / Team Information

|  |  |
| --- | --- |
| Team Name: | ***AlgoVisionaries*** |
| **Team member 1 (Team Lead)**  Sharma Aayushi  23121926  [sharmaaayushi953@gmail.com](mailto:sharmaaayushi953@gmail.com) |  |
| **Team member 2**  Tanuja  230124746  [tanujachuphal7@gmai.com](mailto:tanujachuphal7@gmai.com) |  |

|  |  |
| --- | --- |
| **Team member 3**  Saniya  230122509  [s02aniyakhatri@gmail.com](mailto:s02aniyakhatri@gmail.com) |  |
| **Team member 4**  Petwal Jatin  230111171  [jatinpetwal41@gmail.com](mailto:jatinpetwal41@gmail.com) |  |

# PROPOSAL DESCRIPTION

## Motivation

|  |
| --- |
| *The problem I want to solve is the lack of accessible, interactive tools for visualizing and understanding fundamental graph traversal and algorithmic concepts. Graph theory is a crucial area of study in computer science, with applications in areas such as networking, artificial intelligence, resource optimization, and data analysis. Algorithms like BFS, DFS, Dijkstra, Kruskal, and others are essential for solving real-world problems like pathfinding, network flow, clustering, and graph partitioning.*  ***Importance :***   * An interactive visualizer that allows users to input graphs, choose algorithms, and see step-by-step executions would make these concepts far more intuitive. By visualizing the traversal and transformation of graphs, users can directly observe how different algorithms handle nodes and edges, how they detect cycles, or how they find shortest paths. This immediate feedback loop is crucial for reinforcing learning and improving algorithmic problem-solving skills. * This tool would be especially beneficial for students, researchers, and professionals who are looking to solidify their understanding of graph algorithms and their real-world applications. Furthermore, it would foster an engaging, hands-on learning environment where users can experiment with different graph structures and algorithms to gain deeper insights. By bridging the gap between theory and practice, this tool would enhance both academic learning and practical problem-solving capabilities. |

## State of the Art / Current solution

|  |
| --- |
| Currently, the problem of visualizing and understanding graph algorithms is addressed through a mix of static resources, textbooks, and software tools, but there are limitations in terms of interactivity and engagement.   1. **Textbooks and Online Tutorials**: These resources provide theoretical explanations of graph algorithms, often with static diagrams and step-by-step descriptions of how the algorithms work. However, they fail to provide a dynamic, visual understanding of how the algorithms operate on actual graphs. 2. **Graphing Software**: Tools like **Gephi** and **Graphviz** can visualize graphs and their structures, but they typically do not focus on algorithmic visualization. They allow users to draw and explore graphs but lack step-by-step algorithmic executions. 3. **Online Visualizers**: Websites like **VisuAlgo** and **Algorithm Visualizer** provide interactive visualizations of various algorithms, including graph algorithms like BFS, DFS, and Dijkstra. These tools allow users to see how the algorithms work step-by-step on a graph, providing a more interactive experience. However, many of these visualizers are limited in the scope of algorithms they cover or may not support advanced algorithms like Kruskal, Prim, or graph-related algorithms |

## Project Goals and Milestones

|  |
| --- |
| *The goal of this project is to create an interactive* ***Graph Traversal Algorithms Visualizer*** *that provides users with an intuitive, hands-on experience of understanding various graph algorithms.*  **General Goals:**   1. **Interactive Graph Visualization**: Enable users to draw custom graphs by adding/removing nodes and edges, or use randomly generated graphs. 2. **Algorithm Execution**: Allow users to select from various algorithms and visualize the execution in real-time, showing how each algorithm processes the graph. 3. **Educational Tool**: Provide a learning resource for students and developers to understand and compare different graph algorithms. 4. **Customizable Features**: Implement features to adjust graph complexity, algorithm speed, and visual feedback.   **Initial Milestones:**   1. **Graph Creation and Basic Visualization**: Implement a basic interface that allows users to draw graphs interactively. Include basic visualization (nodes and edges). 2. **BFS and DFS Visualizations**: Implement BFS and DFS algorithms and display their execution on the graph. 3. **Dijkstra’s Algorithm**: Add Dijkstra's algorithm and visualize the shortest path calculation. 4. **Prim’s and Kruskal’s Algorithms**: Implement the Minimum Spanning Tree algorithms and show the selection of edges in MST.   **Subsequent Milestones:**   1. **Advanced Algorithms**: Implement Bellman-Ford, Floyd-Warshall, Topological Sort, and Disjoint Set Union (DSU). 2. **SCC Algorithms**: Integrate Kosaraju's and Tarjan’s algorithms for detecting strongly connected components. 3. **Optimization & UI Enhancements**: Improve the user interface, provide algorithm speed controls, and enable resetting graphs and re-running algorithms. 4. **Deployment and User Testing**: Finalize the tool and test with real users to refine the tool’s usability and educational value. |

## Project Approach

|  |
| --- |
| *To articulate and design the solution for the* ***Graph Traversal Algorithms Visualizer****, I will follow an iterative approach that combines a user-friendly interface with efficient algorithm implementations.*  **1. Research and Planning:**   * **Define Scope**: Identify the specific graph algorithms to include in the initial release (BFS, DFS, Dijkstra, etc.), ensuring they address common graph problems. * **User Research**: Understand the target audience (students, developers, educators) and gather insights on the most useful features, such as customizable graphs, algorithm step tracking, and visual feedback.   **2. Core Development:**   * **Graph Representation**: Design the data structures needed to represent the graph (adjacency list, matrix, etc.) and allow dynamic graph creation. Use **React.js** to manage the app’s state and render the graph interface. * **Algorithm Implementation**: Implement each algorithm on the backend:   + **Basic Algorithms** (BFS, DFS): Implement graph traversal algorithms that process nodes and edges.   + **Advanced Algorithms** (Dijkstra, Bellman-Ford, etc.): Focus on pathfinding and optimization algorithms, ensuring correct step-by-step execution.   + **MST Algorithms** (Prim, Kruskal): Implement Minimum Spanning Tree algorithms and visualize the MST construction process.   + **SCC Algorithms** (Kosaraju, Tarjan): Add components for detecting Strongly Connected Components.   **3. Frontend Visualization:**   * **Graph Rendering**: Use ***NetworkX*** or **Matplotlib** for graph visualization. These libraries will allow dynamic, animated graph rendering and real-time feedback on algorithm execution. * **Algorithm Visualization**: Show active nodes, edges, and algorithm progress through color changes, animations, or highlighting. Ensure clarity in representing which parts of the graph are being processed at each step. |

## 

## System Architecture (High Level Diagram)

|  |
| --- |
| +----------------------------+  | User Interface |  | (Frontend - React.js) |  | |  | - Graph Input |  | - Algorithm Selection |  | - Visualization Display |  | - Control Panel |  +------------+---------------+  |  | WebSocket / REST API  v  +----------------------------+  | Backend (Node.js/Python)|  | - Algorithm Engine |  | - Graph Data Storage |  +------------+---------------+  |  v  +----------------------------+  | Visualization Library |  | (*Matplotlib* / *NetworkX*) |  | - Real-time Graph Update |  +------------+---------------+ |

## Project Outcome / Deliverables

|  |
| --- |
| The outcomes and deliverables of the **Graph Traversal Algorithms Visualizer** project are:   1. **Interactive Visualization Tool**: A fully functional web-based tool where users can input graphs, choose algorithms (like BFS, DFS, Dijkstra, Kruskal, etc.), and visualize the algorithm execution in real time. The tool will support dynamic graph creation and allow users to see algorithm steps through visual animations. 2. **Algorithm Implementations**: A comprehensive set of graph algorithms implemented and integrated into the system, including:    * Traversal algorithms (BFS, DFS)    * Pathfinding algorithms (Dijkstra, Bellman-Ford)    * Minimum Spanning Tree (Kruskal, Prim)    * Advanced algorithms (Floyd-Warshall, Kosaraju’s, Tarjan’s) 3. **User Interface**: A clean, intuitive user interface built with **React.js**, enabling users to interact with graphs, control algorithm execution, and adjust settings like speed and complexity. 4. **Deployment**: A publicly accessible web app, hosted on **Netlify** or **Vercel** for frontend and **Heroku** or **AWS** for backend, ensuring easy access for students, developers, and educators. 5. **Documentation and Tutorials**: A user manual or guide, explaining how to use the tool, the functionality of each algorithm, and providing educational resources about graph theory and the algorithms implemented. |

# Assumptions

|  |
| --- |
| The assumptions made for this project include:   1. **User Knowledge**: The tool assumes users have basic understanding of graph theory and algorithms. 2. **Graph Size**: The system is designed to handle graphs with a moderate number of nodes and edges. Extremely large graphs may impact performance. 3. **Algorithm Complexity**: We assume that algorithms are executed in real-time for educational purposes, so they may not be optimized for production-level efficiency on very large datasets. 4. **Internet Connectivity**: The tool relies on internet access for cloud hosting and real-time communication between frontend and backend. |

## 

References

|  |
| --- |
| Here is a list of resources and references utilized for the development of the **Graph Traversal Algorithms Visualizer** project:   1. **NetworkX Documentation**:    * NetworkX Documentation – For graph creation, manipulation, and basic algorithm implementations like BFS, DFS, Dijkstra, etc. 2. **Plotly**:    * Plotly Python Documentation – For interactive graph visualization, especially in 2D and 3D. 3. **Flask** (if using Python for backend):    * Flask Documentation – For setting up the backend server (optional if Python is used). 4. **WebSocket API**:    * [MDN WebSocket Documentation](https://developer.mozilla.org/en-US/docs/Web/API/WebSocket) – For real-time communication between frontend and backend. 5. **Educational Resources**:  * VisuAlgo – A reference for visualizing algorithms. * GeeksforGeeks Graph Algorithms – For understanding graph algorithms in-depth. |