Text

Description automatically generated with medium confidence

**Assignment no 1**

**Submitted to:**

Sir Zuhaib Hussain Butt

**Submitted by:**

Fizza Rehman (Group Leader) …….. 211400089

Hira Imtiaz ……...….......................... 211400060

Sahoora Jamshed …………………... 211400084

Haroon Azam Ali …………………... 211400110

**Course Name:**

Artificial Intelligence

**Course Code:**

CS-415

**Section:**

“C”

**Submission date:**

December 22,2024.

**Topic Name:**

“Report on graph traversal using BFS, DFS, UCS”

**“Report on graph traversal using BFS, DFS, UCS”**

**Introduction**

In Computer Science, graph traversal describes the process of exploring the nodes (vertices) as well as the edges which connect them. This work is focused on developing a web-based, interactive interface capable of visualizing and providing solutions to pathfinding through using of three popular algorithms: **BFS, DFS, and UCS.**

It would, therefore enhance user understanding in graph traversal through visualization in real-time, animation and interaction. It is further enabled for users to interact by modifying the graph, hence add routes and assess various strategies for traversal impacts pathfinding. This report thus outlines the purpose of this assignment, the approach applied to solve the problem at hand, the primary functions developed and the significance of this project overall.

**Purpose**

This assignment aims to:

* Demonstrate graph traversal algorithms.
* Allow users to visualize the traversal of graphs step-by-step.
* Investigate the performance and behavior differences between BFS, DFS, and UCS.
* Facilitate user interaction through the ability to specify start and end nodes, reset the graph, and dynamically add new routes.

The assignment focuses not only on algorithmic implementation but also on user interaction and educational value. It provides an interactive interface that makes complex graph traversal concepts easier to understand for users, making it an effective learning tool.

**Graph Representation**

Our web application demonstrates graph traversal using BFS, DFS, and UCS algorithms. All algorithms visualize traversal on the same graph but with different behaviours:

**Uniform Cost Search (UCS):** It shows the cumulative cost along edges. On selection of UCS, the graph updates to show the costs on edges dynamically during traversal. It guarantees the least-cost path in weighted graphs.

**Breadth-First Search (BFS):** It explores the graph level by level but does not consider edge costs. No costs are displayed during traversal, since BFS prefers the number of edges over their weights.

**BFS Time Complexity**: **O(V + E)**

* **V** = Number of vertices (nodes) in the graph.
* **E** = Number of edges in the graph.

**Depth-First Search (DFS):** Explore deeply along paths before backtracking regardless of costs. Like BFS, no costs are shown during traversal.

**DFS Time Complexity**: **O(V + E)**

* **V** = Number of vertices (nodes) in the graph.
* **E** = Number of edges in the graph.

**Visualization**

In our code we use the **Vis.js library** to display the graph interactively. The nodes correspond to cities, and edges to routes, optionally annotated with costs. In our website we use three colors in graph that perform different activities like:

* **Blue Color:** Nodes in default positions.
* **Red Color:** Visited nodes in traversal.
* **Green Color:** Final destination.

**Dynamic Features**

In the website there is a button with name “Add Route” that allows the graph structure to be dynamically changed by adding new connections, so users can experiment with custom graphs and see how the algorithms adapt to these changes.

**Languages and Tools Used**

1. **HTML (HyperText Markup Language):**

Defines the structure of the application, including input fields, buttons, and containers for the graph.

**Example:** Provides the basic layout for graph visualization using three different options of algorithm like BFS, DFS, UCS and user interaction.

1. **CSS (Cascading Style Sheets):**

Styles the application for better visualization and user experience.

**Example:** Ensures the layout is clean, buttons are clearly labeled, and graphs are displayed attractively.

1. **JavaScript:**

In script.js file we write the logic for graph traversal, visualization, and interactivity.

**Example:** Deals with algorithms, user input and dynamic updates of graph

* 1. **Vis.js Library:**

In our code we use the JavaScript library whose main purpose is to streamlines rendering nodes and edges, supports customization, animation.

**Example:** Displays the graph features: dynamic coloring, highlighting the path.

**Functions used in code along their working**

**1. renderGraph()**

Displays the graph on the screen and updates it dynamically.

**How It Works:**

Iterates through the adjacency list to create nodes and edges. Colors nodes based on their roles default, visited, or destination). Uses Vis.js to render the graph in the container.

**2. bfs(graph, start, end)**

Implements Breadth-First Search to find the shortest path by edge count**.**

**How It Works:**

Uses a queue to explore nodes level by level. Tracks visited nodes by keeping a set to avoid revisiting**.** Returns when the destination node is reached, returning path and steps.

**3. dfs(graph, start, end)**

Implement Depth-First Search to extend paths as far as possible.

**How It Works:**

Explore one path as far as possible before backtracking with a stack. Use a set for visited nodes. Return a path and steps when target found.

**4. ucs(graph, start, end)**

Implement Uniform Cost Search where the goal is to use the least-cost path.

**How It Works:**

Explores paths via priority queue in order increasing cost. Ensuring that the lowest-cost path is explored first. Guarantees an optimal solution in weighted graphs.

**5. highlightPath(path)**

Animate the traversal by progressively highlighting nodes

**How It Works:**

Iterate through the path, update the graph visualization at every step. Introduce a delay for smooth animation.

**6. addRoute()**

Add a new route between two cities dynamically.

**How It Works:**

Ask for start and end cities, and cost of the route. Update the adjacency list, re render the graph.

**7. findPath()**

It takes in user input, runs the chosen algorithm, and then displays results.

**How it Works:**

Reads start and end cities from input fields. Calls the chosen algorithm (BFS, DFS, or UCS). Marks the path and displays results like path, cost, and steps.

**Conclusion**

This assignment manages to integrate theoretical graph concepts with practical implementation, which forms an interesting and educative tool. With real-time visualization, algorithm execution, and user interaction, the tool gives insight into the behaviors of BFS, DFS, and UCS. It identifies the strengths and weaknesses of the algorithms while providing features to customize such as adding routes. Such interactive websites allow the user to tinker with graph traversal techniques along with observing how different algorithms solve pathfinding problems.

**Github Link**  
<https://github.com/Fizza-Rehman424/Graph-Traversal-using-BFS-DFS-UCS-Code/tree/main>