**Artificial Intelligence**

**Path Finding Algorithms Visualizer**

**Group Name:** AAks

**Group Members:**

Khizer Farooq 211400037

Muhammad Aqib 211400056

Muhammad Awais 211400075

Samiullah 211400108

**Implement BFS, DFS, UCS with Frontend**

**Purpose**

The purpose of this assignment is to develop a **Graph Pathfinding Visualizer** that demonstrates the functionality of three popular pathfinding algorithms:

1. **Breadth-First Search (BFS)**
2. **Depth-First Search (DFS)**
3. **Uniform Cost Search (UCS)**

The implementation allows users to:

* Build a graph dynamically by adding nodes and weighted edges.
* Visualize the graph using an interactive graphical interface.
* Execute the algorithms to find and display paths between a start and goal node.

The project also helps users understand how these algorithms traverse graphs and prioritize nodes.

**Scope**

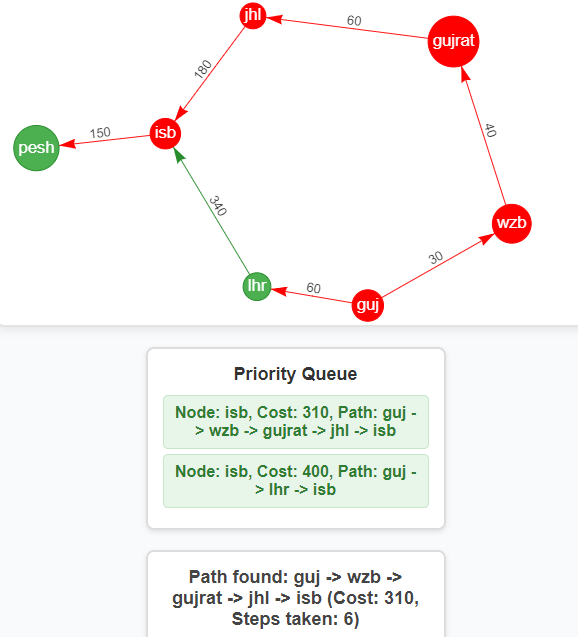
The scope of the project includes:

* **Graph Input:** Allowing users to input nodes, edges, and costs dynamically through an interactive interface.
* **Graph Visualization:** Displaying the graph with nodes and edges using the **Vis.js library** to improve user experience.
* **Pathfinding Algorithms:**
  + **BFS:** Exploring the graph level by level for unweighted or shortest paths.
  + **DFS:** Exploring nodes depth-first, useful for a deeper search of solutions.
  + **UCS:** Finding the least-cost path based on edge weights.
* **Result Display:** Showing the result (path and cost) interactively.
* **User Interface:** A user-friendly design to add edges, run algorithms, and view results dynamically.

**Key Findings**

1. **Interactive Visualization Enhances Understanding**  
   The use of Vis.js provides an intuitive and clear visualization of nodes, edges, and the graph structure, which aids in understanding graph traversal. We also tried other libraries for graph but this was simple and easy to use. Also we weren’t able to change color of each node visited like the sample you provided. It would be convenient if you had provided as the source code.
2. **Algorithm Efficiency and Behavior**
   * **BFS** performs well for unweighted graphs and explores neighbors level by level.
   * **DFS** can quickly go deep but may not always find the optimal path.
   * **UCS** is efficient for weighted graphs, ensuring the least-cost path is found.
3. **Dynamic User Input**  
   Users can dynamically add nodes and edges, demonstrating how real-world graphs are constructed and updated.
4. **Code Reusability and Simplicity**  
   We use javascript, html and css the code maintains modular functions, which improve reusability and readability. Algorithms are implemented in a straightforward manner using standard concepts of queues, stacks, and priority queues. Also we made three different files for easy understanding of the code.

**UCS:**

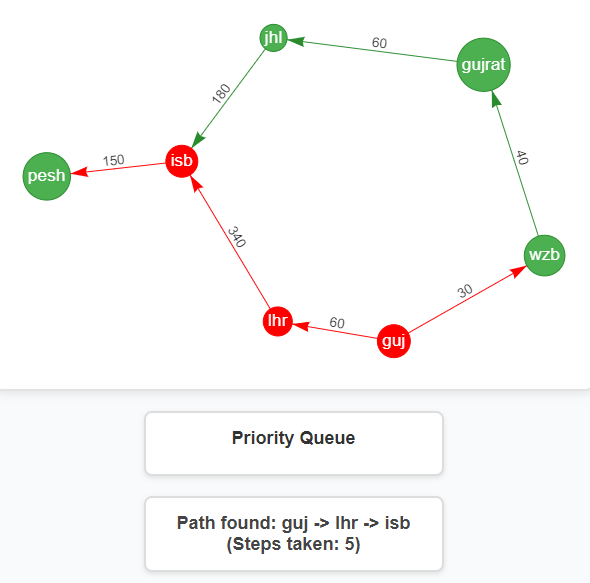


* **Path**: guj -> wzb -> gujrat -> jhl -> isb
* **Cost**: 310
* **Steps Taken**: 6

**Time Complexity:** O(b1+⌊C∗/ϵ⌋)

C\*= 310, ϵ = 30, let b=2 O(211)

BFS:

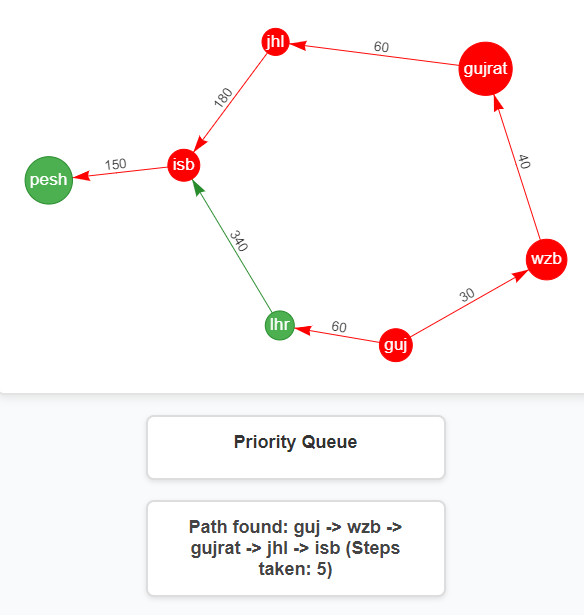


* **Path:** guj -> lhr -> isb
* **Cost:** 400
* **Steps** Taken: 5

Time Complexity:

O(bd) O(2^6)

DFS:

****

**Path :** guj -> wzb -> gujrat -> jhl -> isb

**Steps taken: 5**

**Cost:** 310

**Time Complexity:** O(b^m) O(2^6)