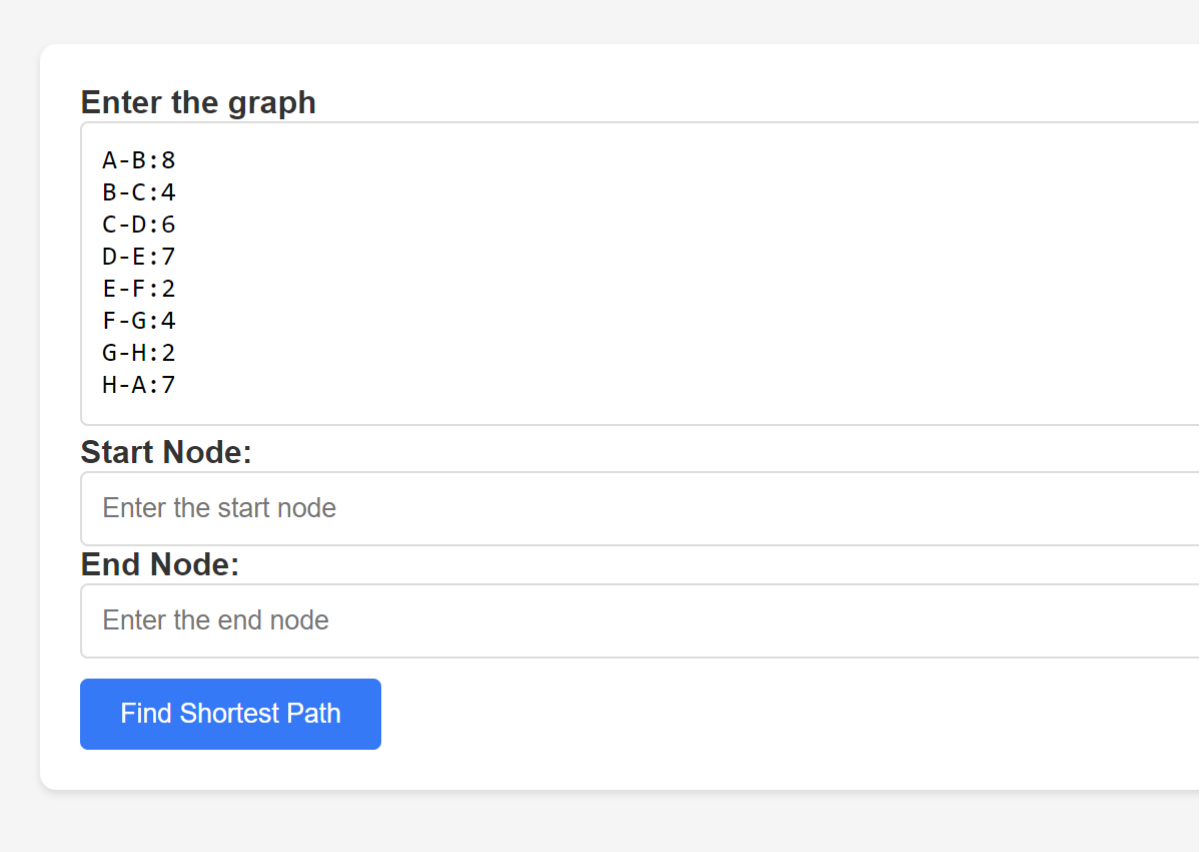
Gaudhiwaa Hendrasto
5025201066
Design & Analysis of Algorithms (G)

Delivery Planner

Demo : https://gaudhiwaa.github.io/deliveryPlanner
Repo : https://github.com/gaudhiwaa/deliveryPlanner



Project Explanation

This project is a delivery planner that helps find the shortest path and total distance between two nodes in a given graph. It uses Dijkstra's algorithm to find the optimal path. The project allows users to input a graph representing nodes and the distances between them. The input format for the graph is "node1-node2:distance". For example, "A-B:8" represents an edge between nodes A and B with a distance of 8. Users can also enter the start and end nodes.

Project Goals

- Provide a user-friendly interface.
- Dijkstra's algorithm.
- Display the results.

By achieving these goals, the delivery planner project helps users plan their deliveries by finding the most efficient route between two locations based on the provided graph.

Tech Stack

- HTML
- CSS
- JavaScript

Gaudhiwaa Hendrasto
5025201066
Design & Analysis of Algorithms (G)
Implementation of Dijkstra's algorithm in JavaScript

```
defaultGraph = "A-B:8\nB-C:4\nC-D:6\nD-E:7\nE-F:2\nF-G:4\nG-H:2\nH-A:7";

document.getElementById("graph-input").defaultValue = defaultGraph;

function findShortestPath() {
  const graphInput = document.getElementById("graph-input").value;
  const startNode = document.getElementById("start-node").value;
  const endNode = document.getElementById("end-node").value;

  if(!graphInput || !startNode || !endNode) {
    alert("Please enter all the input")
    return
  }

  const graph = parseGraph(graphInput);
  const shortestPath = dijkstra(graph, startNode, endNode);

  displayShortestPath(shortestPath.path);
  displayTotalDistance(shortestPath.distance);
}

function parseGraph(graphInput) {
  const graph = {};
  const edges = graphInput.split("\n");

  for (let i = 0; i < edges.length; i++) {
    const edge = edges[i].trim();
    if (edge.length > 0) {
      const [nodes, distance] = edge.split(":");
      const [node1, node2] = nodes.split("-");
      addEdge(graph, node1, node2, parseInt(distance));
    }
  }

  return graph;
}

function addEdge(graph, node1, node2, distance) {
  if (!(node1 in graph)) {
    graph[node1] = {};
  }
  if (!(node2 in graph)) {
    graph[node2] = {};
  }

  graph[node1][node2] = distance;
  graph[node2][node1] = distance;
```

Gaudhiwaa Hendrasto
5025201066
Design & Analysis of Algorithms (G)

```
}

function dijkstra(graph, startNode, endNode) {
  const distances = {};
  const previous = {};
  const unvisited = new Set(Object.keys(graph));

  for (let node in graph) {
    distances[node] = Infinity;
    previous[node] = null;
  }

  distances[startNode] = 0;

  while (unvisited.size > 0) {
    const currentNode = getClosestNode(unvisited, distances);
    unvisited.delete(currentNode);

    for (let neighbor in graph[currentNode]) {
      const distance = distances[currentNode] + graph[currentNode][neighbor];
      if (distance < distances[neighbor]) {
        distances[neighbor] = distance;
        previous[neighbor] = currentNode;
      }
    }
  }

  return {
    path: getPath(previous, endNode),
    distance: distances[endNode]
  };
}

function getClosestNode(unvisited, distances) {
  let minDistance = Infinity;
  let closestNode = null;

  for (let node of unvisited) {
    if (distances[node] < minDistance) {
      minDistance = distances[node];
      closestNode = node;
    }
  }

  return closestNode;
}

function getPath(previous, endNode) {
```

Gaudhiwaa Hendrasto
5025201066
Design & Analysis of Algorithms (G)

```
  const path = [];
  let currentNode = endNode;

  while (currentNode !== null) {
    path.unshift(currentNode);
    currentNode = previous[currentNode];
  }

  return path;
}

function displayShortestPath(path) {
  const shortestPathElement = document.getElementById("shortest-path");
  shortestPathElement.textContent = path.join(" -> ");

  const outputContainer = document.getElementById("output-container");
  outputContainer.style.display = "block";
}

function displayTotalDistance(distance) {
  const totalDistanceElement = document.getElementById("total-distance");
  totalDistanceElement.textContent = distance;

  const outputContainer = document.getElementById("output-container");
  outputContainer.style.display = "block";
}
```

Explanation of Dijkstra's algorithm

- parseGraph(graphInput).

  This function takes the graphInput string, which represents the graph in the format "Node1-Node2:Distance", and parses it into a graph data structure. It creates an empty graph object and iterates over each line of the graphInput. For each line, it trims any leading or trailing spaces, splits the line into the nodes and distance, and adds an edge to the graph using the addEdge() function. Finally, it returns the constructed graph.

- addEdge(graph, node1, node2, distance)

  Adds an edge between node1 and node2 with a given distance to the graph object. It ensures that both node1 and node2 exist as keys in the graph object. It sets the distance between node1 and node2 and also between node2 and node1 to the given distance.

- dijkstra(graph, startNode, endNode)

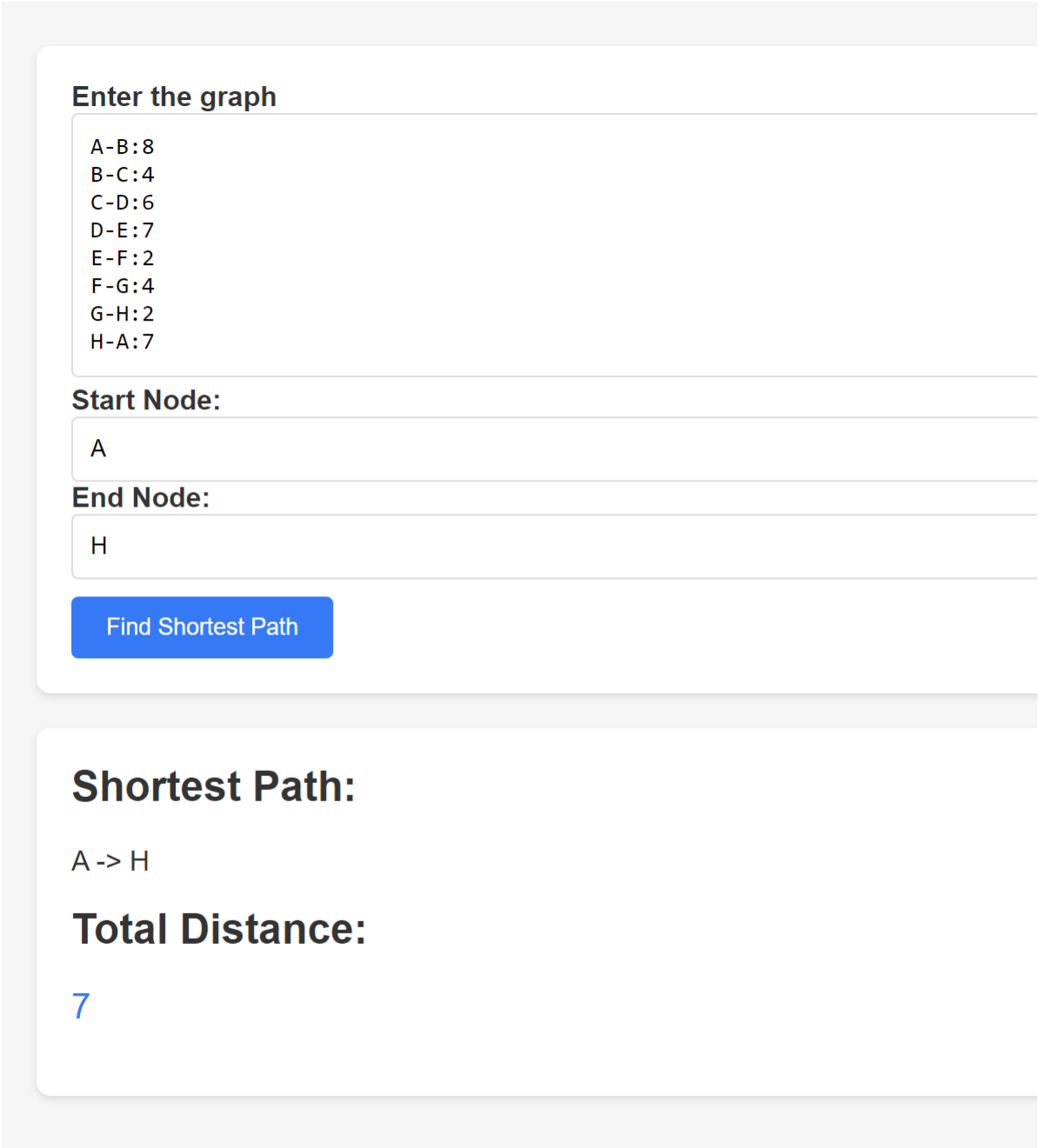
  This function implements Dijkstra's algorithm to find the shortest path between the startNode and endNode in the graph. It initializes distances and previous objects, and a unvisited set to keep track of unvisited nodes. It iteratively selects the node with the smallest distance from the unvisited set using the getClosestNode() function. It updates the distances to neighboring nodes if a shorter

path is found and keeps track of the previous nodes. Finally, it returns an object with the shortest path (path) and its distance (distance).

- getClosestNode(unvisited, distances)

Takes the unvisited set and the distances object and returns the node with the smallest distance. It iterates over the nodes in the unvisited set, comparing their distances from the distances object, and returns the node with the minimum distance.

Result of the program



Gaudhiwaa Hendrasto
5025201066
Design & Analysis of Algorithms (G)

Conclusion

In conclusion, this Delivery Planner project demonstrates the use of graph algorithms to solve a practical problem. By allowing users to input the graph and nodes, it provides a convenient way to plan delivery routes or find the shortest path in various scenarios. The implementation of Dijkstra's algorithm ensures accurate and efficient path finding. Overall, this project offers a helpful tool for route optimization and navigation.