<Bellman-Ford Algorithm Visualizer>

A

DAA Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

By

<SHIVA PALLAVI R><1602-22-737-165>

<SHARATH CHANDRA B><1602-22-737-175>

Department of Information Technology

Vasavi College of Engineering(Autonomous)

ACCREDITEDBY NAAC WITH 'A++'GRADE

(Affiliated to Osmania University and Approved by

AICTE)Ibrahimbagh,Hyderabad-31

2024

Vasavi College of Engineering(Autonomous)

ACCREDITEDBYNAACWITH'A++'GRADE

(Affiliated to Osmania University and Approved by

AICTE)Hyderabad-500031

Department of Information Technology

DECLARATION BY THE CANDIDATE

We,**<SHIVA PALLAVI R>,<SHARATH CHANDRA B>,** bearing hall ticket numbers**,<1602-22-737-165>,<1602-22-737-175>,**hereby declare that the project report entitled**<"Bellman-Ford Algorithm Visualizer">**is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering in Information Technology.**

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

<SHIVA PALLAVI R>

<1602-22-737-165>

<SHARATH CHANDRA B>

<1602-22-737-175>

(FacultyIn-Charge)

(Head,DeptofIT)

ACKNOWLEDGMENT

We extend our sincere thanks to Dr. S. V. Ramana, Principal, Vasavi College of Engineering for his encouragement.

We express our sincere gratitude to Dr. K. Ram Mohan Rao, Professor & Head, Department of Information Technology, Vasavi College of Engineering, for introducing the Mini-Project module in our curriculum, and also for his suggestions, motivation, and co-operation for the successful completion of our Mini Project.

We also want to thank and convey our gratitude towards our DAA Faculty, Haseeba Yaseen for guiding us in understanding the process of project development & giving us timely suggestions at every phase.

ABSTRACT:

The Bellman-Ford algorithm stands as a fundamental method for finding the shortest paths in both positive and negative weighted graph, widely applied in various fields including network routing and distributed systems. This abstract presents an innovative Bellman-Ford Algorithm Visualizer, designed to facilitate a deeper comprehension of this algorithm through interactive visualization. The web-based tool enables users to input graph parameters such as source node, number of vertices, and edges, dynamically generating a visual representation of the graph. Leveraging D3.js, the visualization displays the progression of the Bellman-Ford algorithm, showcasing updates to node distances and the shortest paths. Additionally, arrow markers with labeled costs illustrate the connections between vertices, aiding in understanding the algorithm's inner workings. Through this visualization, users can intuitively explore and grasp the Bellman-Ford algorithm, enhancing their understanding of its concepts and applications.

Introduction:

Project Domain Overview:

Data visualization plays a pivotal role in understanding complex datasets, providing insights, and facilitating decision-making across various domains. In the realm of algorithms and graph theory, visualizations are indispensable for comprehending intricate structures and processes. The Bellman-Ford Algorithm Visualizer serves as a dynamic tool to elucidate the workings of the Bellman-Ford algorithm, a fundamental method for finding the shortest paths in weighted graphs.

Graph theory finds applications in diverse fields such as network routing, transportation systems, social networks, and computer graphics. Understanding algorithms like Bellman-Ford is essential for optimizing network paths, detecting negative cycles, and ensuring efficient resource allocation in such systems.

Features of the Bellman-Ford Algorithm Visualizer:

1. Interactive Graph Input:Users can input graph parameters such as the source node, number of vertices, and edges dynamically through an intuitive interface. This feature enables users to tailor the visualization to their specific graph scenarios.

2. Real-Time Visualization:The visualizer generates an interactive representation of the graph, showcasing the progression of the Bellman-Ford algorithm in real-time. Users can observe the algorithm's iterations and the evolution of shortest path computations as they occur.

3. Dynamic Edge Addition: Users can add edges to the graph dynamically, allowing for the exploration of different graph structures and scenarios. This feature enhances the versatility of the visualizer and enables users to experiment with various graph configurations.

4. Node Distance Updates: The visualizer displays updates to node distances during each iteration of the Bellman-Ford algorithm. Users can observe how the algorithm calculates and refines the shortest paths from the source node to all other nodes in the graph.

5. Arrow Marks with Cost Labels: Each edge in the graph is visualized with an arrow mark indicating the direction of the connection, along with a labeled cost representing the weight of the edge. This visual representation helps users understand the connections between vertices and the associated costs.

6. Step-by-Step Explanation: The visualizer provides a step-by-step explanation of the algorithm's execution, highlighting key insights and updates at each iteration. This feature aids users in understanding the underlying principles of the Bellman-Ford algorithm and its application in solving shortest path problems.

Overall, the Bellman-Ford Algorithm Visualizer offers an interactive and informative platform for learning and exploring the Bellman-Ford algorithm, fostering a deeper understanding of graph theory concepts and their practical implications.

Technology:

Software Requirements:

1. Web Browser: The Bellman-Ford Algorithm Visualizer is a web-based application and therefore requires a modern web browser to run. Recommended browsers include Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.

2. HTML, CSS, and JavaScript: The front-end of the visualizer is built using HTML for structure, CSS for styling, and JavaScript for interactivity. These are standard web development technologies that are supported by all modern web browsers.

3. D3.js Library: The visualization component of the visualizer relies on D3.js, a JavaScript library for creating dynamic and interactive data visualizations in web browsers. D3.js facilitates the manipulation of HTML, SVG, and CSS based on data, enabling the creation of custom visualizations.

4. Bootstrap Framework: The visualizer utilizes the Bootstrap framework for responsive design and styling. Bootstrap provides a collection of CSS and JavaScript components that streamline the development of responsive, mobile-first web interfaces.

.

Proposed work:

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Bellman-Ford Algorithm Visualizer</title>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
<script src="https://d3js.org/d3.v7.min.js"></script>
<style>
 body {
  font-family: Arial, sans-serif;
  background-color: #f8f9fa;
 }
 .container {
  margin-top: 50px;
 }
 #visualization {
  margin-top: 20px;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0px 0px 10px 0px rgba(0,0,0,0.1);
  background-color: #ffffff;
 }
 .step {
  margin-bottom: 10px;
  padding: 10px;
  border-radius: 5px;
  box-shadow: 0px 0px 5px 0px rgba(0,0,0,0.1);
  background-color: #f0f0f0;
 }
 .link {
  fill: none;
  stroke: #666;
  stroke-width: 1.5px;
 }
 .arrow {
  fill: #666;
 }
 .cost-label {
  font-size: 10px;
  text-anchor: middle;
  fill: black;
 }
 .highlighted {
  fill: orange;
  stroke: #ff7f0e;
 }
</style>
</head>
<body>
<div class="container">
 <h2 class="mb-4">Bellman-Ford Algorithm Visualizer</h2>
 <div class="form-group">
  <label for="source">Source Node:</label>
  <input type="text" id="source" class="form-control" placeholder="Enter source vertex">
 </div>
 <div class="form-group">
  <label for="vertices">Number of Vertices:</label>
  <input type="number" id="vertices" class="form-control" min="2" value="5">
 </div>
 <div class="form-group">
  <label for="edges">Number of Edges:</label>
  <input type="number" id="edges" class="form-control" min="1" value="5">
 </div>
 <div id="edge-input">
  <h4>Edge Input:</h4>
  <div id="edge-fields">
   <!-- Edge fields will be added dynamically here -->
  </div>
  <button id="add-edge" class="btn btn-primary">Add Edge</button>
 </div>
 <button id="visualize" class="btn btn-success mt-3">Visualize</button>
 <div id="visualization"></div>
</div>


<script>
 document.getElementById('add-edge').addEventListener('click', function() {
  const edgeFields = document.getElementById('edge-fields');
  const newField = document.createElement('div');
  newField.classList.add('form-group');
  newField.innerHTML = `
   <div class="row">
    <div class="col-sm">
     <label for="vertex-from">From:</label>
     <input type="text" class="form-control vertex-from" placeholder="Enter source vertex">
    </div>
    <div class="col-sm">
     <label for="vertex-to">To:</label>
     <input type="text" class="form-control vertex-to" placeholder="Enter destination vertex">
    </div>
    <div class="col-sm">
     <label for="cost">Cost:</label>
     <input type="number" class="form-control cost" min="-999" max="999" value="0">
    </div>
   </div>
  `;
  edgeFields.appendChild(newField);
```

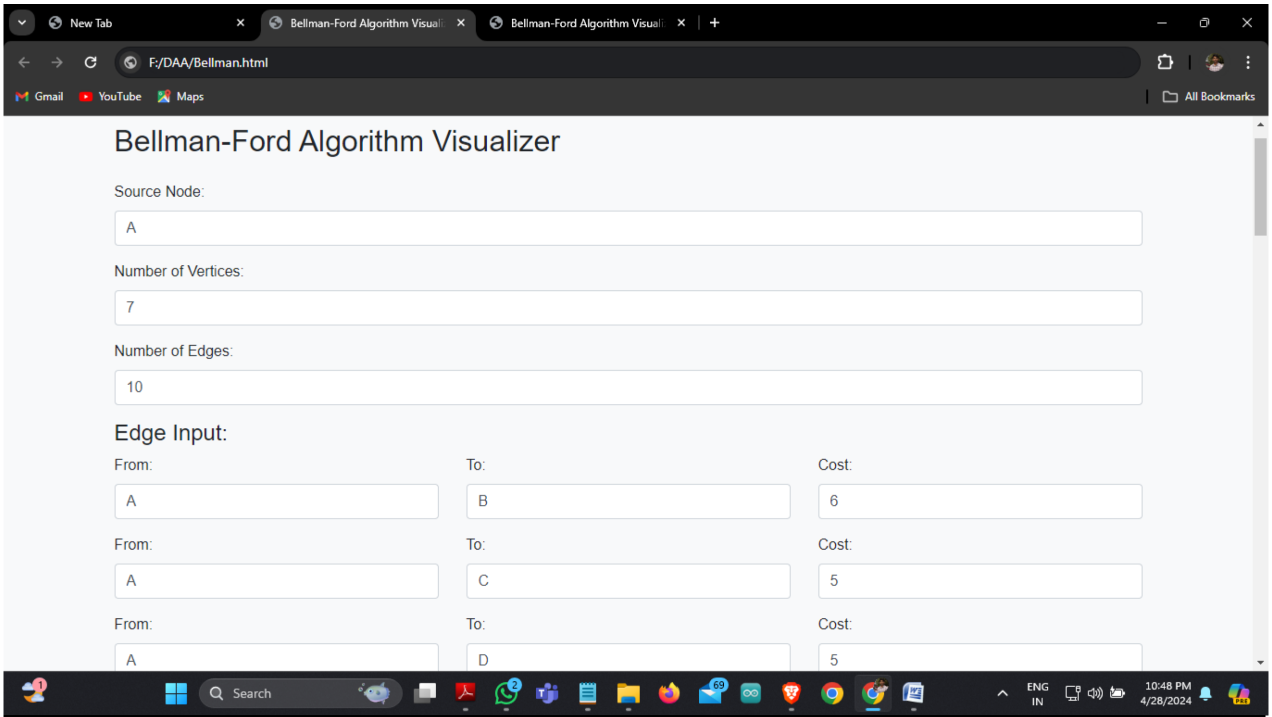
```
});

document.getElementById('visualize').addEventListener('click', function() {
  const visualization = document.getElementById('visualization');
  visualization.innerHTML = '';

  const source = document.getElementById('source').value;
  const vertices = parseInt(document.getElementById('vertices').value);
  const edges = [];
  document.querySelectorAll('#edge-fields > .form-group').forEach(function(edgeField) {
    const from = edgeField.querySelector('.vertex-from').value;
    const to = edgeField.querySelector('.vertex-to').value;
    const cost = parseInt(edgeField.querySelector('.cost').value);
    edges.push({ from, to, cost });
  });

  // Bellman-Ford Algorithm
  const distances = {};
  const previous = {};
  for (let i = 0; i < vertices; i++) {
    const vertex = String.fromCharCode(65 + i); // Convert index to letter (A, B, C, ...)
    distances[vertex] = vertex === source ? 0 : Infinity;
    previous[vertex] = null;
  }

  const steps = [];
  steps.push(`Initial distances: ${JSON.stringify(distances)}`);

  for (let i = 0; i < vertices - 1; i++) {
    for (const edge of edges) {
      const { from, to, cost } = edge;
      if (distances[from] + cost < distances[to]) {
        distances[to] = distances[from] + cost;
        previous[to] = from;
```

```
          steps.push(`Updated distance to node ${to} to ${distances[to]}`);
        }
      }
    }

    // Display final results
    for (const node in distances) {
      const distance = distances[node] === Infinity ? '∞' : distances[node];
      visualization.innerHTML += `<div class="alert alert-info mt-2">Shortest distance from source to node ${node}: ${distance}</div>`;
    }

    // Visualize graph
    const svg = d3.select("#visualization").append("svg")
      .attr("width", 600)
      .attr("height", 400);

    const simulation = d3.forceSimulation()
      .force("link", d3.forceLink().id(d => d.id))
      .force("charge", d3.forceManyBody())
      .force("center", d3.forceCenter(300, 200));

    const graph = {nodes: [], links: []};

    for (const edge of edges) {
      graph.nodes.push({id: edge.from});
      graph.nodes.push({id: edge.to});
      graph.links.push({source: edge.from, target: edge.to, cost: edge.cost});
    }

    const link = svg.selectAll(".link")
      .data(graph.links)
      .enter().append("line")
      .attr("class", "link");
```

```
const arrow = svg.append("svg:defs").selectAll("marker")
  .data(graph.links)
  .enter().append("svg:marker")
  .attr("id", "arrow")
  .attr("viewBox", "0 -5 10 10")
  .attr("refX", 15)
  .attr("refY", 0)
  .attr("markerWidth", 8)
  .attr("markerHeight", 8)
  .attr("orient", "auto")
  .append("svg:path")
  .attr("d", "M0,-5L10,0L0,5");

const linkPath = svg.selectAll(".link")
  .data(graph.links)
  .enter().append("line")
  .attr("class", "link")
  .attr("marker-end", "url(#arrow)");

const node = svg.selectAll(".node")
  .data(graph.nodes)
  .enter().append("g")
  .attr("class", "node")
  .call(d3.drag()
    .on("start", dragstarted)
    .on("drag", dragged)
    .on("end", dragended));

node.append("circle")
  .attr("r", 5);

node.append("text")
  .attr("x", 12)
  .attr("dy", ".35em")
  .text(d => d.id);

const costLabel = svg.selectAll(".cost-label")
  .data(graph.links)
  .enter().append("text")
  .attr("class", "cost-label")
  .text(d => d.cost)
  .attr("x", d => (d.source.x + d.target.x) / 2)
  .attr("y", d => (d.source.y + d.target.y) / 2);

simulation.nodes(graph.nodes).on("tick", ticked);
simulation.force("link").links(graph.links);

function ticked() {
  link
    .attr("x1", d => d.source.x)
    .attr("y1", d => d.source.y)
    .attr("x2", d => d.target.x)
    .attr("y2", d => d.target.y);

  node
    .attr("transform", d => "translate(" + d.x + "," + d.y + ")");

  linkPath
    .attr("x1", d => d.source.x)
    .attr("y1", d => d.source.y)
    .attr("x2", d => d.target.x)
    .attr("y2", d => d.target.y);

  costLabel
    .attr("x", d => (d.source.x + d.target.x) / 2)
    .attr("y", d => (d.source.y + d.target.y) / 2);
}
```

```
    function dragstarted(event, d) {
      if (!event.active) simulation.alphaTarget(0.3).restart();
      d.fx = d.x;
      d.fy = d.y;
    }

    function dragged(event, d) {
      d.fx = event.x;
      d.fy = event.y;
    }

    function dragended(event, d) {
      if (!event.active) simulation.alphaTarget(0);
      d.fx = null;
      d.fy = null;
    }

    // Display steps
    const stepsContainer = document.createElement('div');
    stepsContainer.id = 'steps-container';
    visualization.appendChild(stepsContainer);
    steps.forEach(function(step, index) {
      setTimeout(function() {
        stepsContainer.innerHTML += `<div class="step alert alert-secondary mt-3"><strong>Step ${index + 1}:</strong> ${step}</div>`;
      }, (index + 1) * 1000);
    });
  });
</script>

</body>
</html>
```

Result:



Bellman-Ford Algorithm Visualizer

Source Node: A

Number of Vertices: 7

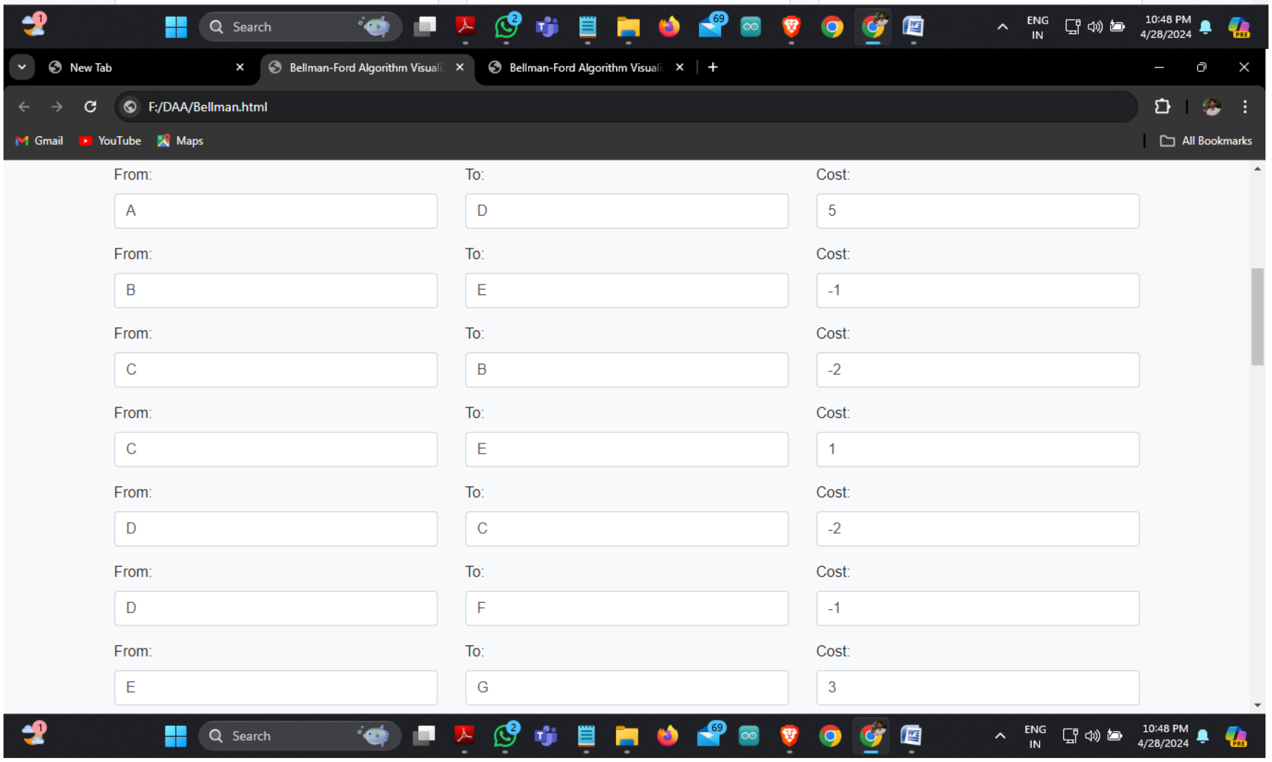
Number of Edges: 10

Edge Input:

From: A To: B Cost: 6

From: A To: C Cost: 5

From: A To: D Cost: 5



From: A To: D Cost: 5

From: B To: E Cost: -1

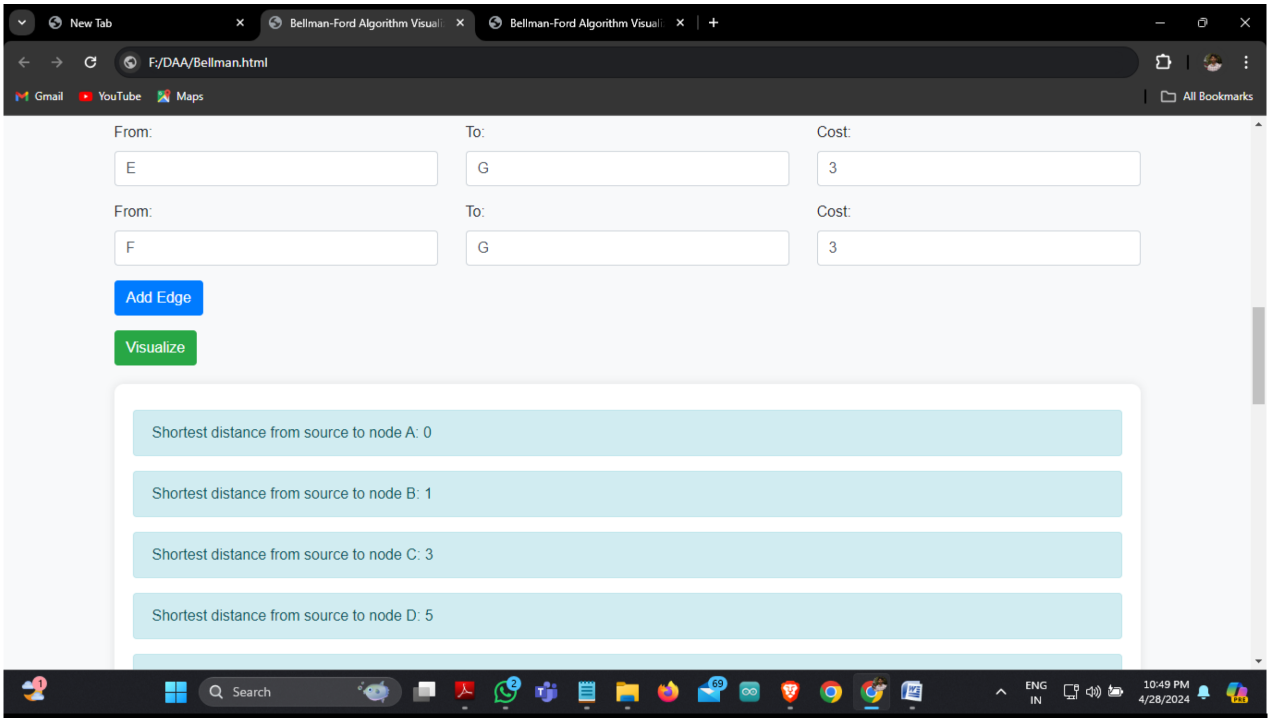
From: C To: B Cost: -2

From: C To: E Cost: 1

From: D To: C Cost: -2

From: D To: F Cost: -1

From: E To: G Cost: 3



From:	To:	Cost:
E	G	3
From:	To:	Cost:
F	G	3

Add Edge

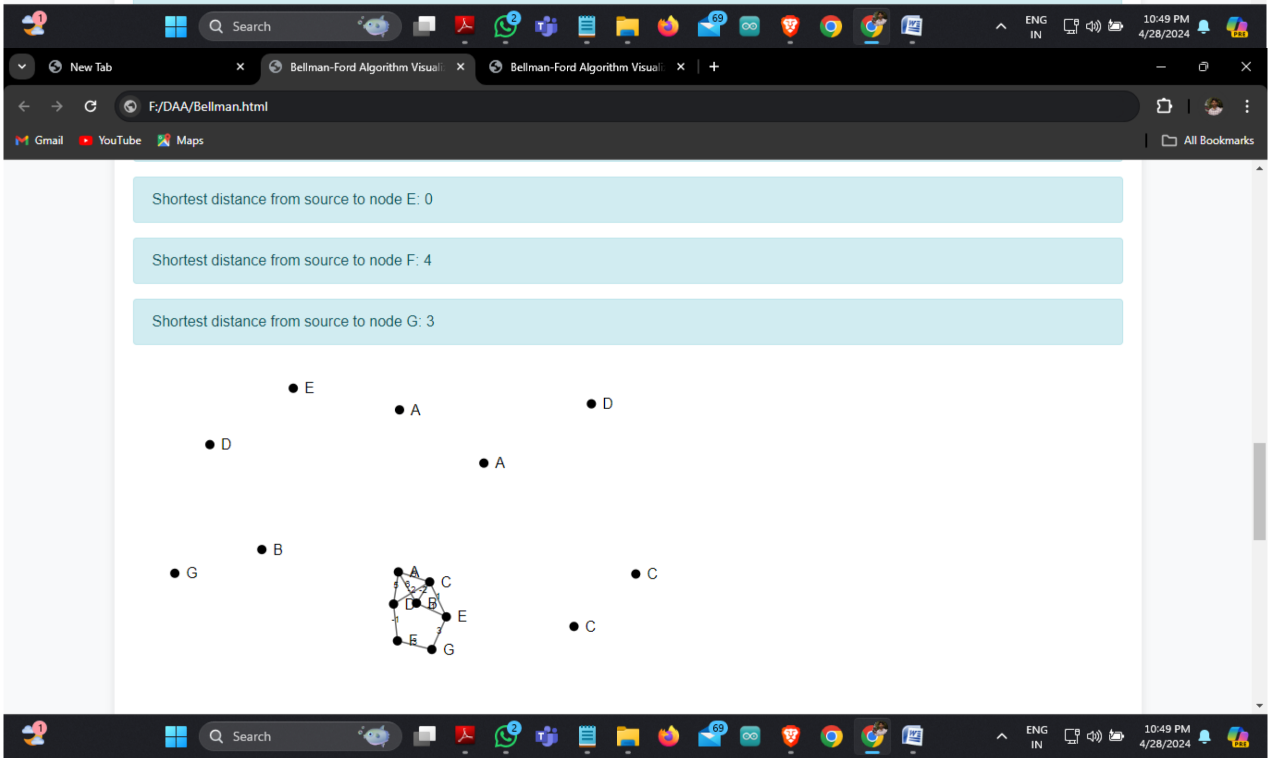
Visualize

Shortest distance from source to node A: 0

Shortest distance from source to node B: 1

Shortest distance from source to node C: 3

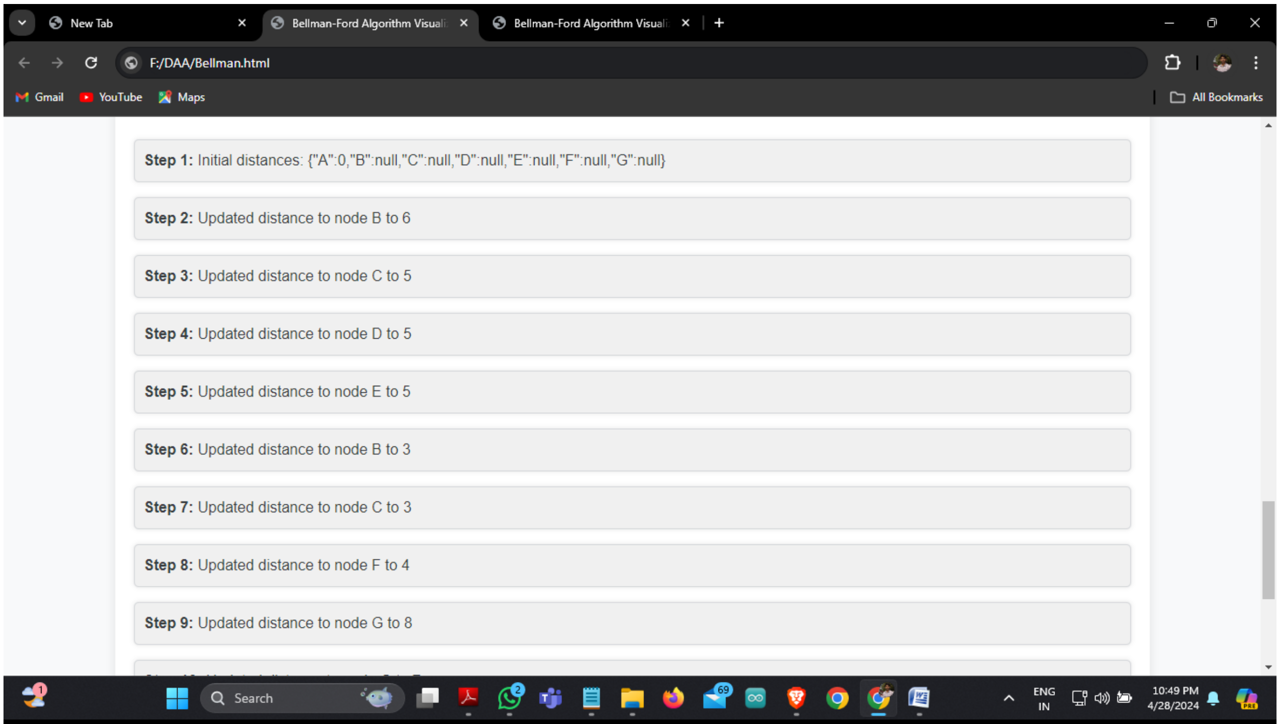
Shortest distance from source to node D: 5



Shortest distance from source to node E: 0

Shortest distance from source to node F: 4

Shortest distance from source to node G: 3



Step 1: Initial distances: {"A":0,"B":null,"C":null,"D":null,"E":null,"F":null,"G":null}

Step 2: Updated distance to node B to 6

Step 3: Updated distance to node C to 5

Step 4: Updated distance to node D to 5

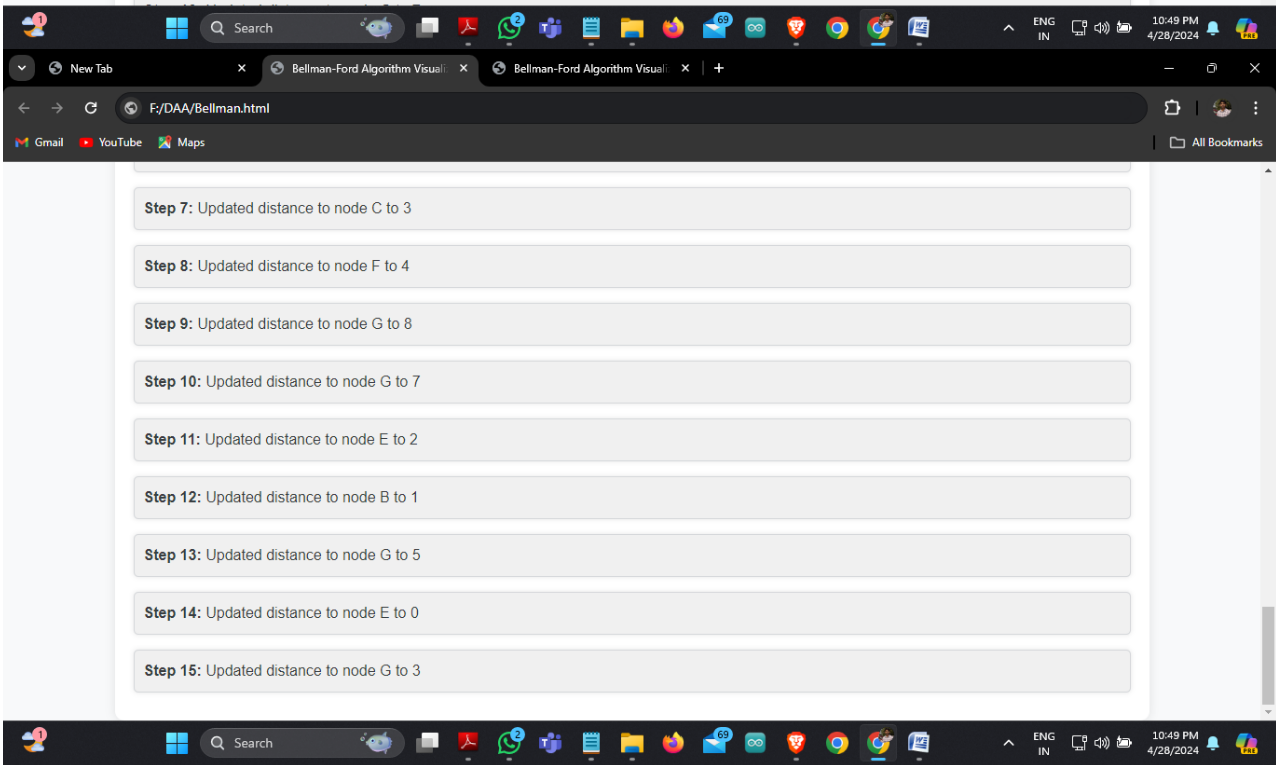
Step 5: Updated distance to node E to 5

Step 6: Updated distance to node B to 3

Step 7: Updated distance to node C to 3

Step 8: Updated distance to node F to 4

Step 9: Updated distance to node G to 8



Step 7: Updated distance to node C to 3

Step 8: Updated distance to node F to 4

Step 9: Updated distance to node G to 8

Step 10: Updated distance to node G to 7

Step 11: Updated distance to node E to 2

Step 12: Updated distance to node B to 1

Step 13: Updated distance to node G to 5

Step 14: Updated distance to node E to 0

Step 15: Updated distance to node G to 3

Conclusion:

In conclusion, the Bellman-Ford Algorithm Visualizer serves as a valuable resource for understanding and exploring the Bellman-Ford algorithm and its applications in graph theory. By providing an interactive platform for visualizing the algorithm's execution and observing its impact on graph structures, the visualizer enhances learning and comprehension for students, researchers, and enthusiasts alike.

Through its intuitive interface and dynamic features, the visualizer enables users to experiment with different graph configurations, observe real-time updates to node distances, and gain insights into the shortest path computations. The inclusion of arrow marks with labeled costs enhances the visualization by illustrating the connections between vertices and the associated edge weights.

The step-by-step explanation provided by the visualizer offers valuable insights into the inner workings of the Bellman-Ford algorithm, fostering a deeper understanding of graph theory concepts and their practical implications. Users can explore various scenarios, analyze algorithmic behavior, and gain proficiency in solving shortest path problems in weighted graphs.

Overall, the Bellman-Ford Algorithm Visualizer represents a significant contribution to the field of algorithm visualization, empowering users to learn, experiment, and innovate in the domain of graph theory and network optimization. As technology continues to evolve, interactive tools like the visualizer play a crucial role in democratizing access to complex algorithms and fostering a community of learners and practitioners dedicated to advancing knowledge and solving real-world challenges.

Reference:

1. Theory and practice from Design and Analysis of Algorithms (DAA) class work.

2. Internet sources related to graph theory, algorithm visualization, and the Bellman-Ford algorithm.

3. Sahni, Sartaj. Data Structures, Algorithms, and Applications in C++. Silicon Press, 2001. (Textbook reference for theoretical concepts related to algorithms and data structures, including the Bellman-Ford algorithm).