Graph Layout Task — Problem Analysis, Approach, and Visualization

1. Problem Analysis

Visualizing graphs in a clear and readable way is challenging, especially when the graph has many nodes and edges. The main difficulties include:

- Avoiding node overlap: Nodes must not be placed too close to each other to remain distinguishable.
- Minimizing edge crossings: Excessive intersecting edges reduce readability.
- Balanced node distribution: Nodes should be spaced evenly, reflecting their connectivity, while fitting within a limited area.

This task requires arranging Malawi's district nodes and their connections inside a normalized 1x1 unit square, ensuring the layout is visually clear and well-structured.

2. Algorithm Identification

A tiered radial layout algorithm was selected to optimize the graph visualization. The algorithm's key ideas are:

- Identify the hub node — the node with the highest number of connections.
- Place the hub node at the centre of the layout.
- Distribute the first-tier neighbours (directly connected to the hub) evenly around it in a circular ring.
- Arrange second-tier neighbours (connected to first-tier nodes) in smaller arcs around their parents.
- Position leaf nodes (connected to only one other node) close to their parent nodes to avoid isolated placements.

This approach balances clarity and preserves the graph's inherent structure, minimizing overlap and crossing.

3. Approach Explanation

The solution process includes:

- Data loading: Reading nodes and edges from a JSON file.

- Adjacency map construction: For efficient neighbour lookups.
- Hub detection: Finding the most connected node.
- Node placement:
    - ✓ Centering the hub node at coordinates (0.5, 0.5).
    - ✓ Placing first-tier neighbours evenly on a circle around the hub.
    - ✓ Arranging second-tier neighbours on smaller arcs around their parents.
    - ✓ Positioning leaf nodes near their parents with small random offsets to
        - prevent overlap.
- Handling unassigned nodes: Placing any leftover nodes in an outer ring around the layout.
- Coordinate constraints: Ensuring all positions stay within the 1x1 unit square.

4. Code Implementation

The implementation is done in JavaScript (Node.js), which:

- Reads the graph from data.json.
- Builds adjacency lists.
- Applies the tiered radial layout algorithm to compute new node positions.
- Writes the resulting node positions and edges to layout.json.
- Logs the updated node positions to the console for inspection.

The code is well-commented, modular, and avoids hardcoded values by dynamically adjusting to the graph structure.

5. Visualization

To visualize the computed layout, an index.html file is included which:

- Loads the layout.json output.
- Renders nodes and edges graphically using an HTML5 canvas.
- Displays the nodes as points placed according to their optimized (x, y) coordinates.
- Draws edges as lines connecting nodes, reflecting actual graph connections.
- Provides an interactive and clear visual confirmation of the layout quality.

This visualization provides a graphical rendering of the graph layout, allowing for easy verification of node distribution, minimal overlap, and edge clarity.

6. Submission Contents

The GitHub repository includes:

- layout.js: The JavaScript code implementing the layout algorithm.
- data.json: The input graph data.
- layout.json: The computed node positions.
- index.html: The visualization page rendering the graph.
- README.md: Instructions on running the code and viewing the graph.
- Screenshots of console output and the rendered graph visualization.