MAIN.RS

- **Purpose**: The entry point for the program that reads graph data, processes connected components, visualizes data, and outputs results.
- **Steps**:
    - Loads the graph from CSV files (`edges.csv` and `nodes.csv`).
    - Calculates connected components for the graph.
    - Counts component sizes and calculates component scale.
    - Prints component sizes and scale.
    - Creates an aggregated visualization of the component scale.
    - Visualizes the connectivity of the entire graph.
    - Calculates and processes subgraphs for each subject.
    - For each subgraph, calculates connected components, visualizes connectivity, and prints information on component distribution.

Module: graph/mod.rs

- **NodeData struct**
    - **Purpose**: Represents node data with attributes such as label, subject, and features.
    - **Steps**:
        - Reads CSV line and converts it into NodeData.
        - Parses label, subject, and feature values from the line.
        - Used for partial serialization from CSV data
- **Graph struct**
    - **Purpose**: Represents a graph with nodes, edges, and associated data.
    - **Steps**:
        - Stores adjacency list, node data, and reverse mapping of nodes.
        - Provides methods for graph-related computations and visualizations.
    - **create_directed**
        - **Purpose**: Creates a directed graph from node data and edges.
        - **Steps**:
            - Initializes adjacency list for the graph.
            - Populates adjacency list with edges from input.
    - **from_csvs**
        - **Purpose**: Reads graph data from CSV files for edges and node data.
        - **Steps**:
            - Parses nodes and edges from respective CSV files.
            - Creates a directed graph using parsed data.
    - **calc_num_edges**
        - **Purpose**: Calculates the total number of edges in the graph.

        - **Steps**:
            - Iterates over the adjacency list to sum the edge counts.
    - **calculate_subgraphs**
        - **Purpose**: Divides the graph into subgraphs based on node subjects.
        - **Steps**:
            - Groups nodes by their subject.
            - Creates new subgraphs with the grouped nodes and their edges.
                - This subgraph creation filters and remaps the adjacency list to ensure that each node in the graph is correctly mapped. This is the main use of the Node_data stuct
    - **connected_components**
        - **Purpose**: Finds and returns connected components in the graph.
        - **Steps**:
            - Performs BFS to find all connected components.
    - **visualize_connectivity function**
        - **Purpose**: Visualizes the connected components of the graph. *This is my personal favorite part of the project, really shows (in subgraphs) that certain fields are WAY more self-referential than other fields*
        - **Steps**:
            - Uses Plotters to draw a graph with nodes and edges.
            - Assigns positions to nodes and colors based on components.
            - Draws edges and nodes on the graph.
            - Images stored in plots\subgraphs

Module: graph/component_functions/mod.rs

- **mark_component_bfs**
    - **Function Purpose**: Identifies connected components using BFS.
    - **Function Steps**:
        - Start from a vertex, assign it a component ID.
        - Use BFS to visit all connected vertices.
        - Mark all reachable vertices with the same component ID.
- **count_components**
    - **Function Purpose**: Counts the number of nodes in each component.
    - **Function Steps**:
        - Iterate through the component vector.
        - Count nodes for each component ID.
        - Return a vector with the node count for each component.
- **get_component_scale**

    - **Function Purpose**: Calculates the percentage distribution of component sizes.
    - **Function Steps**:
        - Get component sizes using count_components.
        - Optionally, sort components by size in descending order.
        - Calculate cumulative percentage of total nodes for each component.

Module: graph/visualization_support/mod.rs

- **show_aggregation**
    - **Function Purpose**: Creates an elbow shaped line chart to visualize aggregation progress with a given number of components.
    - **Function Steps**:
        - Initialize a drawing area with a file name and size.
        - Set chart title and configure axes.
        - Plot the provided points as a line on the chart.
        - Save the chart to the specified file.
- **get_graph_dimensions**
    - **Function Purpose**: Calculates positions and sizes for graph components in a 2D space. Used as a helper function for graph::visualize_components()
    - **Function Steps**:
        - Calculate the total number of nodes and sort components by size.
        - Calculate the radius and position for each component.
        - Ensure components do not overlap using a grid-based spatial check.
        - Store positions and sizes of components, adjusting placement as needed.
        - Create node generation ranges based on the portion of components, but also based on a largest circle size from the user
            - This is because we don't want 99% of the graphing area to be used for one component
- **interpolate_color**
    - **Function Purpose**: Interpolates between two RGB colors based on a fraction.
    - **Function Steps**:
        - Calculate the red, green, and blue components of the color at the given fraction t.
        - Return the interpolated color as a tuple of RGB values.
- **get_color_from_gradient**
    - **Function Purpose**: Returns a color from a gradient based on an index.
    - **Function Steps**:
        - Normalize the index to a value between 0 and 1.
        - Interpolate between dark blue and teal colors based on the normalized value.
        - Return the resulting color as an RGBA value.

Test Case Descriptions

1. **test_connected_components_single_component**
   - **Description**: This test ensures that the `connected_components` function correctly identifies a graph with a single connected component. It loads a graph from CSV files that represent a single component and checks that the number of components detected is 1.
   - **Purpose**: Validates that the component detection logic works when there is only one component.
2. **test_connected_components_multiple_components**
   - **Description**: This test checks the `connected_components` function with a graph that contains multiple disconnected components. It loads the graph from CSV files and verifies that the function detects exactly 3 components in the graph.
   - **Purpose**: Ensures that the component detection logic works correctly for graphs with multiple disconnected components.
3. **test_count_components**
   - **Description**: This test checks the `count_components` function to ensure that it correctly counts the number of nodes in each component of a graph. It loads a graph with a single component and checks that the function returns the correct component size (5 nodes in this case).
   - **Purpose**: Verifies that the component size counting function returns the expected result for a graph with a single component.
4. **test_show_aggregation**
   - **Description**: This test evaluates the `show_aggregation` function, which generates an aggregated visual representation of a graph. It loads a multi-component graph and checks if the function successfully creates a visualization file (PNG image) showing the aggregation of components.
   - **Purpose**: Ensures that the graph aggregation functionality generates a valid output visualization.
5. **test_visualize_connectivity**
   - **Description**: This test verifies the `visualize_connectivity` function by generating a visualization of the connectivity between components. It loads a multi-component graph and checks that the function correctly creates a connectivity visualization (PNG image) at the specified resolution.
   - **Purpose**: Ensures that the graph's connectivity is accurately visualized and the function produces the expected output.
6. **test_subgraphs**
   - **Description**: This test checks the `calculate_subgraphs` function to ensure it correctly detects subgraphs within a multi-component graph. The test ensures

that the graph is divided into 2 subgraphs and checks if the number of subgraphs is correctly identified.

- **Purpose**: Validates that the subgraph calculation function can correctly identify and return subgraphs in a graph with multiple components.