**DS 210 Final Project - Samyuktaa Jayakrishnan**

## Introduction

The Enron scandal was a series of events involving illegal accounting practices that resulted in the bankruptcy of the company. By using accounting tactics to hide their debt and inflate their profits, Enron led their company to bankruptcy. I chose to analyze a dataset of Enron's email communication network that was uncovered by the Federal Energy Regulatory Commission during its investigation of this scandal. There are 36,692 nodes and 18,3831 edges, and if an address *i sent* at least one email to address *j*, the graph contains an undirected edge from *i* to *j*. The main goal of this project is to see if there are any significant connections between nodes that should have been flagged during the investigation, or if there are any specific emails nodes that are highly communicative. I do this by measuring BFS and closeness centrality.

## Code Breakdown

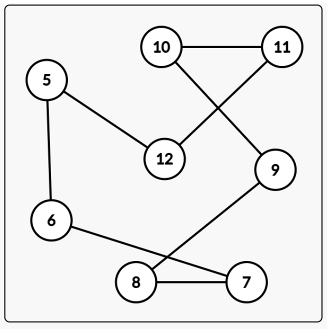
I broke my code down into 4 different modules, main.rs, BFS.rs, centrality.rs and graph.rs.

### main.rs

The main module started off by importing the necessary modules (mod graph, mod BFS, and mod centrality) to define the graph structure, import breadth-first search (BFS) algorithm, and centrality calculations.

Next, I include the function, top_ten_nodes_by_connections, which is designed to find the top 10 nodes in a graph based on the number of connections (or degree) each node has. It initializes node_connections, a vector of tuples and iterates over each node in the graph (from 0 to the length of the adjacency list - graph.adj_list.len()). For each node, it creates a tuple consisting of the node's ID and the number of connections it has, using the map function. The graph.connections method is called to get the number of connections for each node. Lastly, the resulting tuples are collected into the node_connections vector. The next line used sort by to order the vector in descending order based on the number of connections.
Finally, this line transforms the sorted vector into an iterator, takes the first 10 elements and the main function has the mechanisms to print the answer in the format shown in my results section.

Lastly there is a main function that starts by specifying the filename for the dataset and initializing a graph with a specified number of nodes, then reads the dataset file, and adds edges to the graph based on the data. After constructing the graph, it performs the following tasks which are the basis of my analysis.



1. Finds the pair of nodes with the most emails exchanged.
2. Calculates and prints the closeness centrality for a specific node (in this case, node 0).
3. Finds and prints the top 10 nodes by the number of connections.
4. Calculates and prints the top nodes by closeness centrality in the range 5000-5055.

I also included my tests in my main file, creating a sample graph that is shown in the visualization above. I then conducted two tests that evaluate my connections and closeness centrality functions.

**graph.rs**
This module defines a Graph structure, NodeId is defined as an alias for u32, to make sure there is a unique identifier for each node in the graph. Next, AdjacencyList is a vector of hashmaps. Each hash map represents a node, mapping to other nodes with a count of interactions (emails exchanged) as u32.The Graph struct contains a single field adj_list, which is an instance of AdjacencyList.
In the Graph Implementation, the function initializes a Graph with a specified number of nodes. The add_or_update_edge method adds or updates an edge between two nodes, and the connections method retrieves the number of unique connections a node has. It does this by counting the number of entries in the hash map corresponding to the node in the adjacency list. If a node doesn't exist in the graph, it returns 0.

Within the block for Graph, the following methods are defined

1. new(num_nodes: usize)( Creates a new graph with a specified number of nodes, initializing an empty adjacency list.
2. add_or_update_edge(&mut self, src: NodeId, dest: NodeId): Adds or updates an edge between two nodes in the graph by incrementing their connection count in the adjacency list.
3. connections(&self, node: NodeId) -> usize: Returns the number of unique connections (degree) of a node in the graph.

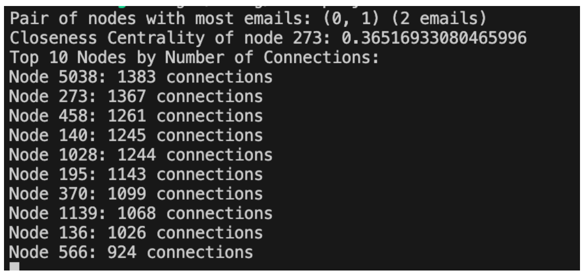
**bfs.rs**
This module implements the Breadth-First Search (BFS) algorithm, using Graph and a starting node ID (NodeId) as inputs. The graph is structured as an adjacency list, where each node (identified by NodeId) maps to a list of its neighbors. The function initializes a vector distances with a length equal to the number of nodes in the graph (num_nodes). Furthermore, a VecDeque queue is used to manage the nodes to be visited, by starting at at the start node, setting its distance to 0 and adding it to the queue. This part was a key component in computing closeness centrality (in centrality.rs).and creating the  graph's structure (graph.rs) is crucial for BFS.

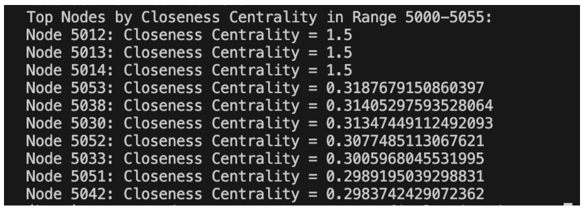
**centrality.rs**
The function closeness_centrality, computes the closeness centrality of a given node in a graph, utilizing the broader functionalities provided by the graph and BFS. It is calculated as the average of the shortest path length from the node to every other node in the network.The function takes a reference to a Graph and a NodeId as inputs. It begins by calling the BFS

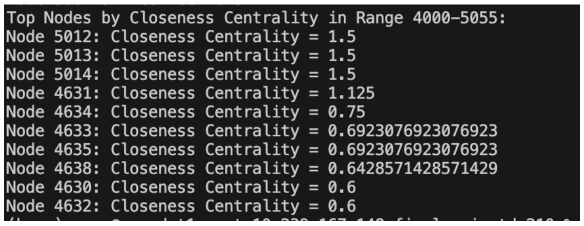
function from the BFS module to compute the shortest path distances from the given node to all other nodes in the graph. The BFS function returns a vector of distances, where the distance to unreachable nodes is represented by u32::MAX. Next, the function calculates sum_distances, the sum of all distances to reachable nodes (ignoring those with a distance of u32::MAX). If sum_distances is greater than zero, the function then counts the number of reachable nodes. The closeness centrality is calculated as the ratio of the number of reachable nodes to the sum_distances. This ratio reflects how centrally located the node is within the network: a higher centrality value indicates a node is closer to all other nodes.

**Outputs and Conclusion**

My project produced 3 notable outputs. First was the pair of nodes with the most amount of emails between, which was Node 0 and Node 1. In regards to the investigation, I would suggest reading the emails of these two employees for information. Furthermore, I found the nodes with the highest number of connections. I assumed that these employees are higher up in the power scale and should be investigated as well. Lastly, I measured the closeness and centrality of employees within a specified range, because my dataset was large. I first measured employees in the range 5000-5055, because I wanted to see the closeness of the centrality of the employee with the most connections (Node 5038).Within this range, this employee was the employee with the 5th highest closeness centrality. I also wanted to see the closeness of the centrality of the employee with the second highest number of connections, which is shown in the 2nd line in the first picture. In my final code, I measured closeness centrality in the range of 4000-5055, and since a higher centrality value indicates a node is closer to all other nodes, these 10 employees are important figures within the company and the scandal.


```
Pair of nodes with most emails: (0, 1) (2 emails)
Closeness Centrality of node 273: 0.36516933080465996
Top 10 Nodes by Number of Connections:
Node 5038: 1383 connections
Node 273: 1367 connections
Node 458: 1261 connections
Node 140: 1245 connections
Node 1028: 1244 connections
Node 195: 1143 connections
Node 370: 1099 connections
Node 1139: 1068 connections
Node 136: 1026 connections
Node 566: 924 connections
```


```
  Top Nodes by Closeness Centrality in Range 5000-5055:
  Node 5012: Closeness Centrality = 1.5
  Node 5013: Closeness Centrality = 1.5
  Node 5014: Closeness Centrality = 1.5
  Node 5053: Closeness Centrality = 0.3187679150860397
  Node 5038: Closeness Centrality = 0.31405297593528064
  Node 5030: Closeness Centrality = 0.31347449112492093
  Node 5052: Closeness Centrality = 0.3077485113067621
  Node 5033: Closeness Centrality = 0.3005968045531995
  Node 5051: Closeness Centrality = 0.2989195039298831
  Node 5042: Closeness Centrality = 0.2983742429072362
```


```
Top Nodes by Closeness Centrality in Range 4000-5055:
Node 5012: Closeness Centrality = 1.5
Node 5013: Closeness Centrality = 1.5
Node 5014: Closeness Centrality = 1.5
Node 4631: Closeness Centrality = 1.125
Node 4634: Closeness Centrality = 0.75
Node 4633: Closeness Centrality = 0.6923076923076923
Node 4635: Closeness Centrality = 0.6923076923076923
Node 4638: Closeness Centrality = 0.6428571428571429
Node 4630: Closeness Centrality = 0.6
Node 4632: Closeness Centrality = 0.6
```

**Sources**

https://lib.rs/crates/graphrs
https://docs.rs/petgraph/latest/petgraph/visit/struct.Bfs.html
https://stackoverflow.com/questions/71439636/create-a-relationship-between-nodes-based-on-existing-relationship-between-the-n