My Data Set: https://snap.stanford.edu/data/twitch_gamers.html
My data set is a social network of Twitch users which was collected from the public API in Spring 2018. Nodes are Twitch users and edges are mutual follower relationships between them. This data zip file came with an edge list csv showing who follows who which is what I used for my code. This is an undirected graph situation! I used this because I thought it would be interesting to see the connection between twitch users and because the data set I originally wanted to use was not recommended by Professor K.

What my code does:
Basically, I originally just wanted to use BFS to find the shortest path from random pairs of nodes in my edge list but I wasn't meeting the minimum line requirement which is why I added depth first search to see how many connected components there are and I also calculated the average distance/degree between nodes. Using BFS to find the distance between pairs was the project idea given to me by Professor K in office hours so it was not an original idea. But DFS and average degree/distance was added by me just to add lines of code. I decided to find connected nodes/components using DFS by googling "what can I do with dfs" and that was the first thing that came up on google. What each line is for and what sources I used are in my rust code but here is generally what I did:

1.) First, I coded a function to read my csv and take out the edges and nodes and add them to an empty vector.
2.) Then, I coded a function to pair up the nodes randomly and put pairs into a new vector.
3.) After that, I built my adjacency list using the nodes and edges I would get when reading through the csv (my first function).
4.) After that, I used BFS to find the shortest path between pairs of nodes by keeping track of nodes visited in a hashset and making sure I'm going through all the nodes in the queue.
5.) After, I had tests to see if my adjacency list, pairing method, and bfs distances even worked.
6.) After, I find the average degree of nodes with a simple function
7.) Then, I use DFS to find all the connected nodes/components by putting all nodes into a stack and visiting each node to see if it had any neighbors
8.) Lastly, I ran my main function to get my output

Sample of my output for cargo test and cargo run:

Distance between 69018 and 161167: 3
Distance between 123360 and 114363: 3
Distance between 162045 and 58427: 3
Distance between 159256 and 162761: 3

My connected component is just a really large matrix of all the nodes that are connected. A lot of my node distances were about 2 to 4 degrees away and some were 5 but if two nodes had no connection to each other at all then my output would be 18446744073709551615 for the distances which is the max value for a usize variable. If there were disconnected nodes that had that default output then that may explain why my average degree for nodes was 80.87. The tests showed me that my distances, pairing, and adjacency list were all correct. I will say that before I chose to only take a sample of 1000 pairs of nodes, my code took way too long to run and I realized that my data set was too large. Now, even though I chose 1000 pairs, it still takes a really long time for the code to run every single pair (hours). I am not sure how to fix this while still managing the 1000 node minimum so I messed around with the number of pairs generator under my main function to see my complete output including the average degree/distance and connected nodes by limiting it to 20 pairs and it worked. It works for 1000 pairs but it just takes too long.

Sources:

- Used chat GPT for errors. Here are the ways I used it:
    - I needed to add pub before my fn test_run_tests function because my function wasn't working in main because my function was private
    - Stack was overfilled (something like this?) and my code crashed
    - Needed to add dependencies
    - Need to add "adjacency_list" and "pairs" after I wrote each function because that returns it back to me
    - Told me to add this before my mod test function: super::*;
    - My breadth first search code wasn't working sp chat gpt told me to add the lines:
        - distances.insert(neighbor, distance + 1)
        - queue.push_back(neighbor)
        - Helped me fix and understand the difference between my run_tests1 and run_tests2 because my tests were not working
        - Added ok(()) to the end of my main function but I'm not really sure why. I think I need it to show that I am supposed to get results and

that the main function worked. It wasn't working if I didn't have the ok.

- Got the "let line" use from https://stackoverflow.com/questions/30186037/how-can-i-read-a-single-line-from-stdin-in-rust
- Used this source to figure out how to pair nodes together: https://www.reddit.com/r/rust/comments/r4ovyl/how_to_choose_a_random_string_or_integer_from_a/
- Used this to help with breadth first search code: https://gist.github.com/vTurbine/16fbb99225ad4c0ac80b24855dd61a7c
- Used for depth first search: https://codereview.stackexchange.com/questions/184046/dfs-implementation-in-rust
- Used for depth first search: https://docs.rs/petgraph/latest/petgraph/visit/struct.Dfs.html
- Used for depth first search: https://www.programiz.com/dsa/graph-dfs
- Decided to use DFS to find connected components because this source told me that that's what people use it for: https://en.wikipedia.org/wiki/Depth-first_search