My project is based on the Kaggle dataset "Twitter Friends". The dataset contains information about a social network of Twitter users, including their follower and friend lists. The purpose of the project is to analyze the Twitter social network and determine the usual distance between pairs of vertices in the graph. Additionally, the project aims to determine how often friends of your friends are also your friends, as well as identify pairs of vertices who have the most similar or dissimilar sets of connections.

To start the project, I began by downloading the dataset from Kaggle and importing it into Jupyter Notebook. The dataset includes 40,000 unique values and includes information about the followers and friends of a set of Twitter users such as, id, screenName, tags, avatar, followersCount, friendsCount, lang, lastSeen, tweetId, and friends which can be represented as a graph. In the graph, each Twitter user is represented as a vertex, and their followers and friends are represented as edges connecting them to other vertices.

To compute the distance between pairs of vertices in the graph, I used the Breadth-First Search algorithm. This algorithm is a graph traversal technique that starts at the source vertex and explores all the vertices at a distance of one edge, then all the vertices at a distance of two edges, and so on. The shortest path between two vertices can be found using this algorithm by terminating the search as soon as the target vertex is found.

First, I defined some types and data structures. We define a type called Vertex to represent a vertex in the graph, and a type called Component to represent a connected component of the graph. We also define a Graph structure to hold the graph data, which consists of the number of vertices n, a list of edges, and a list of outgoing edges for each vertex.

Next, I defined a method on the Graph structure to create an undirected graph from a list of edges. I did this by iterating through the list of edges and adding each edge to the outedges list for both vertices. We also create a copy of the list of edges and store it in the edges field of the Graph structure.

After that, I defined a function to compute the connected components of the graph using breadth-first search (BFS). We create a vector called component to hold the component numbers for each vertex and initialize it to None. We then iterate through all vertices in the graph, and if a vertex hasn't been assigned to a component yet, we call another function called mark\_component\_bfs to mark all vertices in the component with a unique component number.

The mark\_component\_bfs function does the actual BFS traversal. I start by marking the starting vertex with the given component number and add it to a queue. I then loop through the queue until it's empty, and for each vertex in the queue, I add its neighbors to the queue if they haven't been visited yet and mark them with the same component number.

Next, I defined a function to compute the average distance between pairs of vertices in the graph. I started by creating an empty list to hold the distances, and loop through all pairs of vertices in the graph using nested loops. For each pair of vertices, we call another function called shortest\_path\_length\_bfs to compute the shortest path length between the vertices using BFS. If the function returns a valid distance, I added it to the list of distances.

Then, I defined another function to compute the average distance between vertices in each connected component. I started by creating an empty list to hold the average distances, and for each connected component in the graph, I computed the distance between all pairs of vertices in the component using BFS. I then computed the average distance for the component and add it to the list.

Finally, in the main function, I loaded a graph from a CSV file, compute the connected components, the average distance between pairs of vertices, and the average distance between vertices in each connected component. I then printed out some statistics about the graph, such as the number of vertices, edges, and connected components, as well as the average distances.

Overall, my project involved analyzing the Twitter social network using BFS to find the similarity between sets of connections. By doing so, I was able to determine the usual distance between pairs of vertices in the graph, how often friends of your friends are also your friends and identify pairs of vertices with the most similar or dissimilar sets of connections. Unfortunately, I was unsuccessful in getting my code to run properly but I hope that with this write up it is explained what my goal and purpose was. Below I added relevant sources that helped me complete my project.

Kaggle Dataset: <https://www.kaggle.com/datasets/hwassner/TwitterFriends>.