1. Friendship Network Generation

   First let us set up the basics that there are 1000 students in the school and they are divided to 20 classes based on the proximity. Ie first 50 students in vlass1 and next in class 2 and so on.

   Next let us generate friendships based on random probability. We define the probability differently for same class or not. Students within the same class had a higher chance(random values between 0.7 to 1.0), while those from different classes had lower chances (between 0.0 to 0.3). A threshold of 0.98 determined whether two students became friends.We modeled a school of 1000 students, divided into 20 classes of 50 students each. To create a more realistic network, we applied 1-level triadic closure: if a student A was Friends with both B and C, we slightly increased the probability of friendship between B and C. If this new probability crossed the threshold, they too became friends. This step created Denser clusters and mimicked real-world social groupings.Each friendship was assigned a weight between 1 and 10 (inversely proportional to the Probability) to indicate the strength of the relationship. The resulting graph was stored as a Weighted adjacency list.

2. Friend Group Analysis

We performed connected component analysis using both BFS and DFS algorithms. Each Student was assigned a group number representing their friendship circle. The DFS Approach was used in our final implementation. The analysis revealed the following:

The majority of students were part of a few large interconnected groups, while isolated Students formed their own single-member groups.

3. Shortest Path Algorithms

To explore how students are socially connected, we selected 5 random student pairs and computed the shortest path between them using Dijkstra's algorithm. The path cost was computed as the sum of friendship weights along the path. For one of the pairs, we also implemented the A* algorithm with a simple heuristic: students in the same class had a heuristic cost of 0, and students in different classes had a cost of 5.

In this specific case, A* and Dijkstra gave the same result, but A* was slightly more efficient due to the heuristic narrowing the search space.