We would use a graph to keep track of each user and his friends. Then each group would contain an ArrayList of people in it. When the user calls up a group of people, i.e. CS majors, the list of majors is added to a priority queue with the priority being the length of the shortest path to the person from the user in the graph.

We chose to have a graph that stores the users and their connections because that is the best way to show people who are connected to multiple people and when there are groups of friends that aren’t necessarily friends with anyone else. An overall graph is also easier to maintain than separate data structures for each user maintaining friends.

Within each user there will be a field for each type of group i.e. major. These fields will contain an ArrayList of strings of each group the user is a part of. We chose this way to represent the data as it would be easy to search through the non-comparable data. This would be useful if a user is looking for people in two or more specific groups. One of the groups is chosen and the users in that group are examined for membership in the other search criteria group.

In addition to groups each user would store meeting as a priority queue of meeting objects. The meeting objects are given a priority based upon the time with the next meeting is the first listed. We thought a priority queue would be the best because meetings aren’t searched for but rather just displayed based on a priority.

Each kind of group is an ArrayList of group objects. We chose to implement an ArrayList because it is a small set without an easily comparable structure. For example majors is an ArrayList with each major being a group object.

For each group we would need to sort the people in it in order of relevance to the user searching it. To avoid creating a large number of data structures each time a new user is added to a group we have an underlying ArrayList of users in each group that is then added to a priority queue upon lookup by a specific user. The priority is the shortest path to the user from the user looking it up. This would look through the ArrayList and use an A\* algorithm to determine the shortest path to each user on the list. This would give the user the most relevant users in a group at the top of the list.