Approach – <br/>

1. To solve this team assignment problem, we try to assign each member one by one based on the least conflicting cost that would be generated by placing a person in a specific team. Since this approach is sensitive to the arrangement of the survey we receive, we run this ‘form\_teams’ functions for squared times the length of the survey with random arrangements of the provided survey. <br/>
2. For a survey list of n rows of data, there are n! possibilities of arrangement. Hence, checking all the possible permutations is not possible in polynomial time (even generating n! permutations would take infinite time) for a large value of n, hence we came up with the idea of checking a randomly generated arrangement of the surveys and try to assign teams. <br/>
3. Initial state – Set of all survey result collected.

Goal state – Team assignment with minimum cost

Successor state – Placing a new team member in a team with least cost

Edge weight – Total conflict cost of placing a new member into a team.

The search algorithm does check the cost of assigning a new member to an existing team and forming a new team, but the cost function is not admissible as the team formation if sensitive to the arrangement of data in the surveys, it is not able to find the best team configuration with least cost for a given survey list.

Code Explanation – <br/>

1. In the main function, we read the input values of k, m, n, and the electronic survey file. We initialize a ‘previous\_cost’ variable with a high value to remember the least previous cost that was generated after team assignment. If a team with a lower cost is found, we print it during further execution of the program. A ‘checked\_arrangements’ list is used to track the previously computed states and we do not have to check them again. <br/>
2. For a randomly generate arrangement of the surveys [Line: 119], we call ‘form\_teams’ function. Iterating over each person, if there are no teams available, we create a new team and add the person in it, along with its cost of incorrect team size and missing preferred people. [Line: 68-70] The ‘calculation\_new\_cost’ function is used to calculate the cost of creating a new empty team (if new\_member = None) or adding a team member to an existing team. The ‘total\_cost’ function is used to calculate the cost of forming a new team for a new member [if new\_team = True). <br/>
3. Then, for each next member, we find out the most preferable team to place the new team member in or create a new team for them, based on the overall cost [Line: 71 – 97]. Based on the ‘minimum\_cost\_team’ and ‘minimum\_cost’, we either insert the new member to an existing team or create a new team for them and call ‘add\_new\_member’ function to add a new member to an existing team. <br/>
4. Once we are done with placing each member to a team, we call the ‘total\_cost’ function (with new\_team = False) to calculate the overall cost of all the formed teams. We continue this process for n^2 times. Since the states are generated randomly, it does not guarantee us to find an optimal solution in optimal time. <br/>