Problem Statement:

There are N no. of students who has given their preferences with whom they want to work or with whom they don't want to work. For professor, it is tedious task to fulfill everyone's wish because there may be many conflicting requirements. So, in such conflicting cases they will contact professor which will consume professor's valuable time. After all assignment, each group will submit a set of homework which have to be evaluated by professor. We have to write a code for group assignment which will reduce overall time consumption of professor.

State space:

Sample state for this problem will be the set of all possible combination of students of group size 1, 2 and 3.

Goal state:

There are many states for this problem. All states in which no student is remaining without assigned to any group will the goal state. But, we have to find the goal state whose cost will be the comparatively minimum.

Cost function:

There are 3 types of cost associated with every student i.e., team size mismatch cost, absence of wanted student cost and presence of don't want student cost. In addition to that every group has an assignment evaluation cost. Hence, there is a cost function to calculate a combined cost of a group which works as follows:

- +1 for group size mismatch for all members of group
- +n for every absence of wanted student for every member of the group
- +m for every presence of don't wanted student for every member of the group
- +k for cost of evaluation

Solution description:

To solve this problem, I have come with an approach which uses local search algorithm combined with DFS and conditional pruning.

Some top-level steps of the solution are as follows:

- Process the input
- Generate groups with full consent. Put them in separate pool1. Remove these students from pool
- Generate state space (all possible combinations of team size 1, 2 and 3) of remaining students
- Sort state space according to cost values of each team
- Generate 3 separate sorted lists for team size 1, team size 2 and team size 3.
- Generated groups
- Recollect all groups
- Calculate overall cost
- Display result

Detailed working of program is as follows:

- **User class:**
  To create user defined data type for holding details of each user like
  - username,
  - expected team size,
  - list of students with whom he would like to work and
  - list of students with whom he does not want to work

- **Node class:**
  To create user defined data type for each state to hold details like
  - current team,
  - cost of all finalized teams including current team,
  - already selected teams and
  - list of students who has not been allocated to any group till now

- **Reading inputs:**
  While program is executed from the command line, 4 different inputs are provided. i.e. file-name, value of k, m and n. Program read these 4 values and stores them in variables for further use.

- **Process input:**
  Program reads the input file provided from command line as input. Creates a separate object of User class for every line from the file and store it in a pool.

  For every line, it tokenizes it by space separation. 1st token will be username, 2nd token will be expected team size, 3rd token will be list of wanted students, 4th token will be list of unwanted students.

  If value of 3rd and 4th token is _, then it means no preference, so program stores empty list for such cases for that instance variable.

  This will be repeated for all lines in file and generates a pool of all students.

- **Extract full consent groups:**

  Iterate through pool of students, for each student create a temp list of User.username + User.want. If this list of all wanted students matches that means this is full consent group. Club them together as a tuple. Store them into a new pool1 and remove these students from main pool. This is how we will get all full consent groups separated in new pool1 and main pool will hold all remaining students.

- **Generation of sample space:**

  A sample state if generated for all possible combinations of group size 1, 2 and 3 for all remaining students.

  This is block of code which will iterate over the pool of all remaining students. For each student it will create a create a group of single students. Then it will create a group of size 2 with all remaining students. Then will create a group of 3 with all remaining students. A list of all these possible combinations are stored in a dictionary having key as group and value as cost of the group.

- Sort the sample space as the per the cost values of groups.

  Create 3 different lists for storing groups of size 1, 2 and 3 by iterating over sorted sample space. Now we already have sorted sample space, so by iterating over it and assigning elements to corresponding lists will always create sorted lists.

- **Generation of Groups:**

  Now here is the main work is happening. The main approach is:

  Create tree by considering each node in sorted sample space. Iterate over each tree by using DFS technique. When number of remaining students becomes zero, it stops going into depth. This is my branch cut-off condition. Whenever it reaches to this branch cut-off condition, we will get a possible answer. It may or may not be optimal answer. The cost very first found answer is considered as minimum. Node of the tree is expanded only if cost of the node is less than or equal to minimum cost. Otherwise leave this branch, no need to expand more. Because it is confirming that the solution we will get by expanding these nodes will not be the optimum solution than what we already have. If we reach to the branch cut off condition and still the cost is less than minimum cost we have, then it means we have got an optimum solution than what we already have. In this case, value of minimum will be updated with new minimum cost we found.

  This need to be done for every element is sorted sample list, because selection of first element can affect the cost. But, we are doing pruning, so it will reduce the time to iterating over the trees.

  By using this technique, it is confirmed that we will get most optimum solution, but this will take time. Hence, there is tradeoff of optimum solution and cost. As of now, I am opting time over optimum solution. The same thing will execute for (total number of students)/3 seconds. And returns a most optimum solution found in this time, which is not as much far away from most optimum solution.

Assumptions:

- The basic assumption is always 4 arguments will be provided from command line while running the program, input file path, value of m, n and k.
- Input file has the data given in the exact same format mentioned in question.
- Value of k is comparatively higher than m and n.
- Running time has given more preference than the optimum solution.