1 Problem Statement

You are a tired, overworked teacher who has spent the last week organizing a field trip for your entire middle school. The night before the trip, you realize you forgot to plan the most important part – transportation! Fortunately, your school has access to a large fleet of buses. Being the caring teacher you are, you'd like to ensure that students can still end up on the same bus as their friends. After some investigative work on social media, you've managed to figure out exactly who is friends with who at your school and begin to assign students to buses with the intent of breaking up as few friendships as possible. You've only just begun when you receive a frantic email from one of the chaperones for the trip. The kids this year are particularly rowdy, and the chaperones have given you a list of groups of students who get too rowdy when they are all together. If any of these groups are seated assigned to the same bus, they will all have to be removed from the bus and sent home. Can you plan transportation while keeping both the students and the chaperones happy?

Formally, you're given an undirected graph $G = (V, E)$, an integer k, and an integer s, where each node in the graph denotes a student, and each edge (v_1, v_2) denotes that students v_1 and v_2 are friends. The integer k denotes the number of buses available and the integer s denotes the number of students that can fit on a single bus. Furthermore, you're given a list L, where each element L_j is some subset of V which corresponds to a group that should be kept apart.

You must return a partition of G – a set of sets of vertices V_i such that $V_1 \cup V_2 \cup V_3, \cup ... \cup V_k = V$ and $\forall i \neq j$, $V_i \cap V_j = \emptyset$. Additionally, $\forall i$, $0 < |V_i| \leq s$. In other words, every bus must be non-empty, and must not have more students on it than its capacity allows.

Consider a vertex v to be *valid* if there is no i and j such that $v \in L_j$ and $L_j \subset V_i$. In other words, a vertex is valid if it is not in a rowdy group whose members all end up on the same bus. For example, if one of the rowdy groups was 'Alice', 'Bob', and 'Carol', then putting 'Alice', 'Bob', 'Carol', and 'Dan' on the same bus would lead to 'Alice', 'Bob', and 'Carol' being considered invalid vertices. However, a bus with just 'Alice', 'Bob', and 'Dan' would have no invalid vertices.

We'd like you to produce a partition that maximizes the percent of edges that occur between valid vertices in the same partition in the graph. The score for your partition is the percentage of edges (u, v) where u and v are both valid, and $u, v \in V_i$ for some i. You'd like to produce a valid partition with as high a score as possible.

2 Input Format

Each input consists of two files inside of a folder, **graph.gml** and **parameters.txt**. The first file will contain your graph G in the GML file format. Note that the labels of the vertices in your graph **MUST** be unique alphanumeric strings (this also means no whitespace). The details of the GML file format can be found here. We highly recommend students do not attempt to write their own methods for converting their graphs to .gml files. We instead encourage you to use existing libraries to do this – notably, NetworkX. For more information on NetworkX, check out the Piazza thread on NetworkX resources.

The second file is a text file and will start with two integers each on their own line; the first being k, the number of buses and the second being s, the number of students that fit on each bus. Every subsequent line will contain a non-empty list of vertices which compose a single "rowdy group" that should not all belong to the same bus.

For a single input, the name of the folder that contains the above two files denotes the identifier of that input.

Sample input parameters (Second File):

```
3
5
['Harry', 'Hermione', 'Ron']
['Ron', 'Fred', 'George']
['Malfoy', 'Crabbe', 'Goyle']
```

3 Output Format

For each input folder, you will generate a single output file with the name <**input-identifer**>**.out** (eg. if the folder was named "easy-input" then the output should be named "easy-input.out"). On each line of this file there will be a non empty list containing one of the subsets in your partition of the graph. Every node in the graph should appear exactly once in this file. The number of lines in this file should be exactly equal to the number of buses specified in the input file and no line should have more elements than the bus capacity specified in the input file.

Sample output:

```
['Harry', 'Hermione', 'George']
['Ron', 'Malfoy', 'Fred']
['Crabbe', 'Goyle']
```

Timeline

Deliverable 1 (Due November 11th, 11:59 pm)

Overview

You are allowed to form groups of up to 3 people for the project. This is a hard limit. You must stick with the same group throughout the entire project. While you are allowed to do the project alone or with 2 people, we highly encourage you to find a group of 3. Beyond making the project more manageable, working collaboratively is a skill in and of itself, and working alone only deprives you of an opportunity to develop this skill.

Each group must submit three input files, along with a valid solution to each of those instances. You will also submit a "design doc" detailing a few potential approaches you plan on taking.

You must submit exactly 1 input and a corresponding valid output file for each of the following size categories:

- Small: 25 - 50 students and at most 100 rowdy groups
- Medium: 250 - 500 students and at most 1000 rowdy groups
- Large: 500 - 1000 students and at most 2000 rowdy groups

There are no restrictions on the number of buses or the capacity of each bus for any of the above categories. You must, however, be able to produce a feasible solution.

Your design doc should be **at least 200 words**, and **no more than 500 words long**. It should be a high-level description of algorithms/approaches that you have tried or plan to try. Additionally, you must include your reasoning for what led you to these approaches, and why you think they will work. **This design doc will be graded on a binary basis** – you will initially receive either a score of 0 or 1. If you receive a 0, you must attend a mandatory meeting with a few TAs in which we will discuss project approaches and strategies. After attending this meeting you will receive your point back. The purpose of this system is not to punish students, but to make sure everyone has a good starting off point for working on the second deliverable. Your group will only receive a 0 initially if your initial report indicates that you may not be able to come up with a better-than-naive solution for the problem statement.

We will collect all of the input files and release them to you all after the first deliverable is due. **The difficulty of your input files will determine part of your grade for this deliverable** (details of this under the Grading section).

Some starting points for solver ideas include:

- Developing heuristics for some greedy algorithms. How could you solve this problem by choosing greedily?

- Figuring out what problems this problem is similar to and reducing this problem to those problem. We recommend doing some Googling in addition to looking back at recent discussion and homework questions.
- Reading about approximation algorithms for those other problems
- Finding solvers to those other problems (you have access to all freely available libraries!)

Feel free to use some of these ideas in your preliminary report, but make sure you actually explain concretely what you intend to do. For instance, if you intend to use some greedy heuristics, please specify those heuristics and state why you believe them to be good.

Submission Details

You will submit two folders for the first deliverable of the project, "inputs" and "outputs". The inputs folder will contain 3 subfolders which must be named "large", "medium", and "small". Each subfolder corresponds to one of the input size categories and in each of these subfolders, you will place two files "graph.gml" and "parameters.txt" which comprise the input data. Within the outputs folder, you will place 3 files, "large.out", "medium.out" and "small.out". These are the corresponding valid outputs you generate for each input. One member of the group should zip up these two folders, submit it to **Project Deliverable 1: Inputs**, and add their teammates. Do NOT put the two folders into another folder and zip that super-folder when submitting to Gradescope or the autograder will complain. You should be selecting both folders and zipping them together.

Only one member of your team needs to submit the design doc and inputs, and that member must add the other members on Gradescope.

In order to get out a complete list of inputs to students in a timely fashion, there will be **NO** late submissions for this part of the project. We believe two weeks to be more than enough time to come up with 3 inputs for the project, and would like to give the other students in the class as much time as possible with the final list of inputs for working on their solvers. As with the homeworks, **our official policy is that if you can submit your inputs to Gradescope, then your submission is considered to be submitted on time. We make no guarantees about the availability of the Gradescope submission after the stated deadline.**

Deliverable 2 (Due December 2nd, 11:59 pm)

Overview

You will be provided the pool of inputs generated by the class. These inputs will be divided into folders based on their size category. Design an algorithm and run it on the entire pool of input files. You will **submit your output for**

every input provided. Your grade for this deliverable will be determined in part by how your solver performs compared to other students. In addition to your outputs, you will **write a final report** on the approaches you took and how they performed. You will also **give a brief presentation** to a TA going over the same ideas as in your report.

We have released starter code on Piazza to iterate over a folder and parse inputs (you do not have to use the starter code). You may use any programming language you wish. However, we must be able to replicate your results by running your code. Furthermore, we cannot guarantee that staff will be able to help you with usage of languages and libraries outside of Python and NetworkX.

You may only use free services (academic licenses included). This includes things like free AWS credits for students. If you use AWS or any other cloud computing service, you must cite exactly how many hours you used it for in your project report. We should be able to replicate the results that you cite. If you use any non-standard libraries, you need to explicitly cite which you used, and explain why you chose to use those libraries. If you choose to use the instructional machines, **you may only use one at a time PER TEAM**. We will be strict about enforcing this, and will be sending out a Google form for students to self-report people who they see using multiple instructional machines. **Anybody caught using multiple instructional machines will receive a zero for this part of the project**.

The reason we have to enforce this rule is because CS 170 students always end up overloading the instructional machines in the week before the project is due, and this makes them unavailable to other students. We want to make sure that you are not inconveniencing other students with your project work.

The top 5% of teams in this part will receive a modest amount of extra credit.

Submission

You will make three folders named "small", "medium", and "large". In each folder, you should have a .out file for each input provided in the corresponding size category. For example, if the "small" folder in the pool of inputs has individual inputs named "a", "b", and "c", your output submission should have a "small" folder with files "a.out", "b.out", and "c.out". This will make more sense once you receive the pool of inputs, as the organization of these outputs parallels the organization of the inputs. One member of the group should zip up these three folders, submit it to **Project Deliverable 2: Outputs**, and add their teammates. Do NOT put the three folders into another folder and zip that super-folder when submitting to Gradescope or the autograder will complain. You should be selecting all three folders and zipping them together.

The auto-grader will score your output files immediately (although it may take a while to run). Gradescope will maintain a leaderboard of how well your solutions perform compared to the rest of the class. Your rank will be determined by the average of your individual output scores. When submitting to

Gradescope, you will provide a team name to display your team's results on the leaderboard. Of course it goes without saying, but please keep team names civil and PG13.

You will also submit your code and a **project report** for the second deliverable. Along with your code, you should submit a README containing precise instructions on how to run your code. If we cannot parse your instructions then you run the risk of receiving a penalty to your overall grade.

The report should be a brief summary of how your algorithm works, what other methods you tried along the way, what computing resources you used (e.g. AWS, instructional machines, etc.). Your report should be at least 200 words, and no more than 500 words long. We should be able to replicate all your results using the code you submit as well as your project report. You will submit this report via Gradescope.

In addition to this report, you will sign up for a presentation slot where you will be asked to be give a brief **5 minute presentation** to a TA. In this presentation, your entire group will explain a few of the approaches you took in attempting to solve the project problem. More information on this will be released closer to the final deadline.

We strongly suggest you start working on this deliverable early. In fact we recommend that students to begin working on working on their solvers *before* the deadline for the first deliverable.

Grading

Overall, this project is worth 5% of your final grade. You will earn these percentage points as follows:

- 1% will come from your initial report.
- 1% will come from your final report and project presentation.
- 1% will come from the quality of the inputs you provided us.
- 2% will come from the the quality of your outputs.

Your inputs will be given a score based on how well you scored on your inputs compared to other teams. We consider this a good metric of quality as we want inputs which have good solutions which are not easy to find. The score per input will range from 1 to 5 based on the following:

- 5 points (full credit): at least 80% of student submissions score less than your solution.
- 4 points: 60-80% of submissions score less than your solution.
- 3 points: 40-60% of submissions score less than your solution.
- 2 points: 20-40% of submissions score less than your solution.

- 1 points: 0-20% of submissions score less than your solution.

Given the above grading scheme, we believe that any team that puts in a reasonable amount of effort into this project will receive at least **3% out of the overall 5%**. Of course, we encourage students to create the best solver they possibly can, and as staff we love to see a healthy competitive spirit in project groups. However, we'd like to emphasize that **the point of this project is not to be purely an evaluation of your abilities against those of your peers**. We really want to encourage students to view this as an opportunity to apply what they've learned to an open-ended project in an exploratory way. Do your best, try approaches that sound interesting to you, and have fun!

Late submissions to the second deliverable will receive a 20% penalty on the score for their outputs for each day it is late. The penalty is measured as a percentage of the total score. Lateness is measured by Gradescope submission, and immediately, by the second. For instance, a submission on 12:01:00AM is considered a day late. This late penalty may not be reflected in your Gradescope submission initially and your position on the leaderboard will not be accurate if you submit late.

Academic Honesty

Here are some rules and guidelines to keep in mind while doing the project:

1. No sharing of any files (input files or output files), in any form.
2. No sharing of code, in any form.
3. Informing others about available libraries is encouraged in the spirit of the project. You are encouraged to do this openly, on Piazza.
4. Informing others about available research papers that are potentially relevant is encouraged in the spirit of the project. You are encouraged to do this openly, on Piazza.
5. Informing others about possible reductions is fine, but treat it like a homework question – you are not to give any substantial guidance on how to actually think about formulating the reduction to anyone not in your team.
6. As a general rule of thumb: don't discuss things in such detail that after the discussion, the other teams write code that produces effectively the same outputs as you. This should be a general guideline about how deep your discussions should be.
7. If in doubt, ask us. Ignorance is not an excuse for over-collaborating.

If you observe a team cheating, or have any reason to suspect that someone is not playing by the rules, please report it here.

As a final note, as staff, we generally trust that students will make the right decisions when it comes to academic honesty, and can distinguish collaboration from cheating. However, we'd like to point out that what has made this project so interesting in the past is the wide diversity of student approaches we observe to a single problem. We could have elected to give you yet another problem set, but we believe that the open ended nature of this project is a valuable experience for students to have. Do not deprive yourselves or us of this by over-collaborating or simply taking the same approaches you find your peers using.