CS41 Homework 11

This homework is due at 11:59PM on Wednesday, November 24. Note the unusual due date. This is a 14-point homework. Write your solution using LaTeX. Submit this homework using **github** as **.tex** file. This is a **partnered homework**. You should primarily be discussing problems with your homework partner.

It's ok to discuss approaches at a high level with others. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. The only exception to this rule is work you've done with a lab teammate *while in lab*. In this case, note (in your **homework submission poll**) who you've worked with and what parts were solved during lab.

1. In this problem, you will prove that Three-Coloring is NP-Complete. You have already worked on several pieces of this problem in lab, so you should definitely use that work and *not* start from scratch.

   (a) Prove that Three-Coloring $\in$ NP.

   (b) Given an input x for 3-Sat, create an input for Three-Coloring using the gadgets below (Figures 1 through 5). For each clause in x, you should create a piece of the graph G which will be an input for Three-Coloring.

   Describe how to do this, and what the final graph G consists of. How is the satisfiability of the clause related to the colorability of the piece of the graph?

   Recall from lab that our gadgets are three-colorable graphs which include at least three vertices marked a, b, c. Except for the specified property, the remaining vertices are *unconstrained*. For example, unless the problem states that, e.g., a cannot be red, it must be possible to color the graph in such a way that a is red. Colors for other vertices may be fixed, just not a, b, c.

Figure 1: A graph such that a, b, c all have different colors.

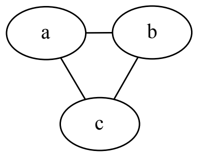


Figure 2: A graph such that a, b, c all have the same color.

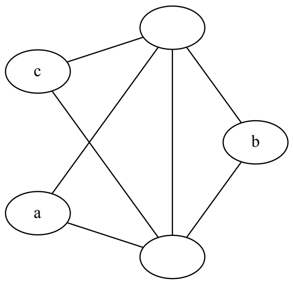


Figure 3: A a graph such that a, b, c do NOT all have the same color.

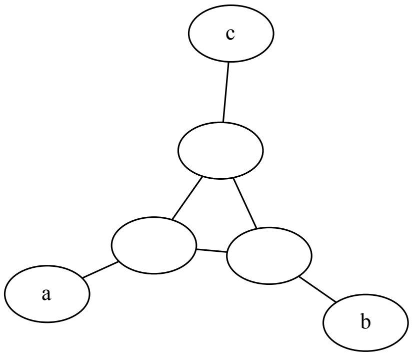


Figure 4: A graph such that none of a, b, c can be green.

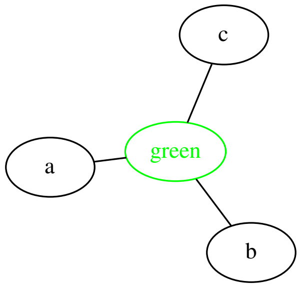
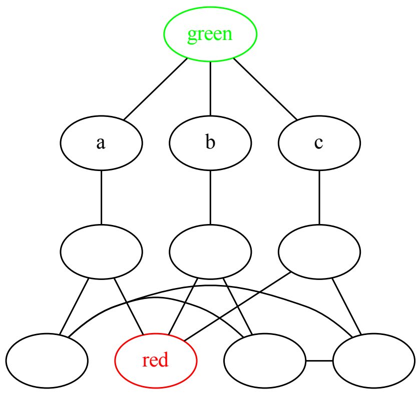


Figure 5: A graph such that none of a, b, c are green, and they cannot *all* be blue.



(c) Run the THREE-COLORING algorithm on the input G you create, and output YES iff the THREE-COLORING algorithm outputs YES. Argue why this procedure gives you a correct answer for 3-SAT. (Hint: Associate the color red with TRUE and the color blue with FALSE.)

2. **Other coloring problems.** It is natural to wonder whether there is something special about the 3 in THREE-COLORING that makes it such a hard problem.

   (a) In the TWO-COLORING problem, the input is a graph $G = (V, E)$, and you should output YES iff the vertices in G can be colored using at most two colors such that each edge $\{u, v\} \in E$ is *bichromatic*. Prove that TWO-COLORING $\in$ P.
   (Hint: look at your notes from earlier in the semester.)

   (b) In the FOUR-COLORING problem, the input is a graph $G = (V, E)$, and you should output YES iff the vertices in G can be colored using at most four colors such that each edge $\{u, v\} \in E$ is *bichromatic*. Prove that FOUR-COLORING $\in$ NP-COMPLETE.
   (Hint: do a reduction from a *very* similar problem.)

3. PATH-SELECTION (K&T 8.9) Consider the following problem. You are managing a communication network, modeled by a directed graph $G = (V, E)$. There are c users who are interested in making use of this network. User i (for each $i = 1, \ldots, c$) issues a *request* to reserve a specific path P_i in G on which to transmit data.

   You are interested in accepting as many of these path requests as possible, subject to the following restriction: if you accept both P_i and P_j, then P_i and P_j cannot share any nodes.

   Thus, the PATH-SELECTION problem asks: Given a directed graph $G = (V, E)$, set of requests $P_1, \ldots, P_c$—each of which is a path in G, and a number k, output YES iff it is possible to select at least k paths so that no two of the selected paths share any nodes.

   Prove that PATH-SELECTION is NP-COMPLETE.

4. Multiple-Independent-Set (K&T 8.14) In this problem, there is a machine that is available to run jobs over some period of time, say 9AM to 5PM.

   People submit jobs to run on the processor; the processor can only work on one job at any simgle point in time. However, in this problem, each job requires a **set of intervals** of time during which it needs to use the machine. Thus, for example, one job could require the processor from 10AM to 11AM and again from 2PM to 3PM. If you accept this job, it ties up your machine during these two hours, but you could still accept jobs that need any other time periods (including the hours from 11AM to 2PM).

   Now, you're given an integer k and a set of n jobs, each specified by a set of time intervals, and you want to answer the following question: is it possible to accept at least k of the jobs so that no two of the accepted jobs have any overlap in time?

   In this problem, you are to show that Multiple-Independent-Set $\in$ NP-complete. To assist you, we've broken down this problem into smaller parts:

   (a) First, show that Multiple-Independent-Set $\in$ NP. (Remember, your verifier must be *proven* to be airtight! Even minor bugs are very bad.)

   (b) In the remaining two parts, you will reduce

   $$\text{Independent-Set} \leqslant_{\mathrm{P}} \text{Multiple-Independent-Set} .$$

   Given input $(G = (V, E), k)$ for Independent-Set, create a valid input for Multiple-Independent-Set. First, divide the processor time window into m distinct and disjoint *intervals* $i_1, \ldots, i_m$. Associate each interval i_j with an edge e_j. Next, create a different job J_v for each vertex $v \in V$. What set of time intervals should you pick for job J_v?

   (c) Finally, run the Multiple-Independent-Set algorithm on the input you create, and output yes iff the Multiple-Independent-Set algorithm outputs yes. Argue that the answer to Multiple-Independent-Set gives you a correct answer to Independent-Set.

5. **(Extra Challenge Problem.)** Does P = NP? Answer yes or no. Justify your response with a formal proof.