CS41 Lab 9:
polynomial-time verifiers and polynomial-time reductions

In typical labs this semester, you'll be working on a number of problems in groups of 3-4 students. You will not be handing in solutions; the primary purpose of these labs is to have a low-pressure space to discuss algorithm design. However, it will be common to have some overlap between lab exercises and homework sets.

This week, we've started to understand what makes some problems seemingly hard to compute. In this lab, we'll consider an easier problem of *verifying* that an algorithm's answer is correct. Recall that a *decision problem* is a problem that requires a YES or NO answer. Alternatively, we can describe decision problem as a set $L \subseteq \{0,1\}^*$; think of L as the set of all YES inputs i.e., the set of inputs x such that one should output YES on input x. Let $|x|$ denote the length of x, in bits.

Consider this lab a **success** if you complete problems 1-2 and make progress on problem 3. Do not feel the need to formally write up solutions.

1. Show that 3-SAT $\in$ NP-HARD, by reducing from SAT.

   $$\text{SAT} \leqslant_P \text{3-SAT}$$

   Given an instance X of SAT (i.e., a list of n variables and m clauses), you should create an instance Y of 3-SAT (i.e., a list of n' variables and m' clauses, each clause having three literals) such that $Y \in$ 3-SAT iff $X \in$ SAT.

2. **Verifier Debugging.** Recall the definition of a polynomial-time verifier:

   Polynomial-time Verifiers. Call V an efficient *verifier* for a decision problem L if

   (a) V is a polynomial-time algorithm that takes two inputs: x, and w.

   (b) There is a polynomial function p such that for all strings x, $x \in L$ if and only if there exists w such that $|w| \leqslant p(|x|)$ and $V(x, w) =$ YES.

   w is usually called the *witness* or *certificate*. Think of w as some *proof* that $x \in L$.

   For V to be a polynomial-time verifier, w must have size some polynomial of the input x. For example, if x represents a graph with n vertices and m edges, the length of w could be n^2 or m^3 or $(n+m)^{100}$ but not 2^n.

   Consider the THREE-COLORING problem: Given $G = (V, E)$ return YES iff the vertices in G can be colored using at most three colors such that each edge $(u, v) \in E$ is *bichromatic*.

   Consider the following verifier for THREE-COLORING. The witness we request is a valid three coloring of the undirected graph $G = (V, E)$, which is specified as a list of two-digit binary strings $w = w_1 w_2 \ldots w_k$ where we interpret

   $$w_i = \begin{cases} 00, & \text{vertex } i \text{ is colored BLUE} \\ 01, & \text{vertex } i \text{ is colored GREEN} \\ 10, & \text{vertex } i \text{ is colored RED} \end{cases}$$

threeColoringVerifier$(G = (V, E), w)$

```
1  for each w_i in w
2      if w_i = 11
3          return NO
4      for j from i + 1 to len(w)
5          if w_i = w_j and (i, j) ∈ E
6              return NO
7  return YES
```

This verifier is not quite right.

(a) Give an example witness w and graph G which is *not* three-colorable, such that

$$\text{threeColoringVerifier}(G, w) = \text{YES}$$

(b) Rewrite threeColoringVerifier so that it is a valid verifier for Three-Coloring. Make sure you have convinced yourself that the entire definition of a polytime verifier has been met!

3. You will show that Three-Coloring is NP-Complete. Before getting there, it will be helpful to create some interesting three-colorable graphs. In all of the following exercises, you are to create a three-colorable graph (say the colors are red, blue, green) with certain special properties. The graphs you create should include three vertices marked a, b, c but can (and often will) include other vertices. Except for the properties specified, these vertices should be *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. (You may fix colors for other vertices, just not a, b, c, and not in a way that constrains the colors of a, b, c.)

- Create a graph such that a, b, c all have different colors.
- Create a graph such that a, b, c all have the same color.
- Create a graph such that a, b, c do *NOT* all have the same color.
- Create a graph such that none of a, b, c can be green.
- Create a graph such that none of a, b, c are green, and they cannot *all* be blue.

4. Show that Three-Coloring $\in$ NP-complete. Hints: reduce from 3-Sat. Associate the color red with True and the color blue with False.

In this problem, you will prove that Three-Coloring is NP-Complete.

(a) Consider the following verifier for Three-Coloring. The witness we request is a valid three coloring of the undirected graph $G = (V, E)$, which is specified as a list of two-digit binary strings $w = w_1 w_2 \dots w_k$ where we interpret

$$w_i = \begin{cases} 00, & \text{vertex } i \text{ is colored BLUE} \\ 01, & \text{vertex } i \text{ is colored GREEN} \\ 10, & \text{vertex } i \text{ is colored RED} \end{cases}$$

```
threeColoringVerifier(G = (V, E), w)
1  for each w_i in w
2      if w_i = 11
3          return NO
4      for j from i + 1 to |w|
5          if w_i = w_j and (i, j) ∈ E
6              return NO
7  return YES
```

This verifier is not quite right.

Give an example witness w and graph G which is *not* three-colorable, such that

$$\text{threeColoringVerifier}(G, w) = \text{YES}$$

(b) Rewrite threeColoringVerifier so that it is a valid verifier for Three-Coloring.

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

(d) 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.

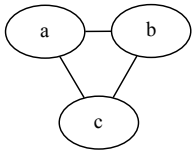


Figure 2: A graph such that a, b, c all have the same color.

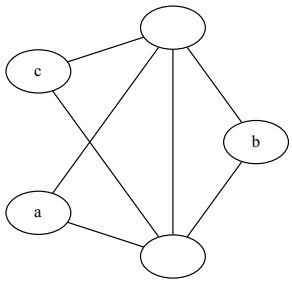


Figure 3: A a graph such that a, b, c do NOT all have the same color.

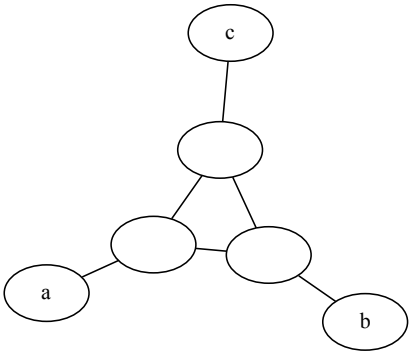


Figure 4: A graph such that none of a, b, c can be green.

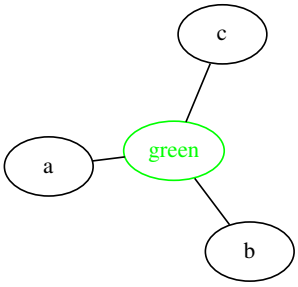
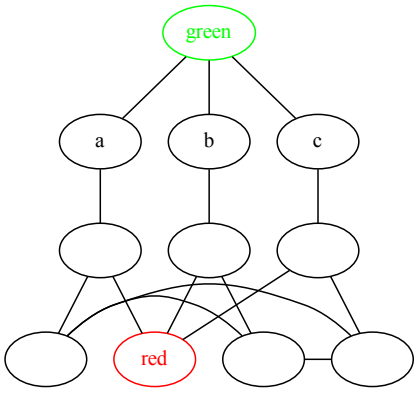


Figure 5: A graph such that none of a, b, c are green, and they cannot *all* be blue.



(e) 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.)