Proving that a problem X is NP-hard requires several steps:

- Choose a problem Y that you already know is NP-hard (because we told you so in class).
- Describe an algorithm to solve Y, using an algorithm for X as a subroutine. Typically this algorithm has the following form: Given an instance of Y, transform it into an instance of X, and then call the magic black-box algorithm for X.
- ***Prove*** that your algorithm is correct. This always requires two separate steps, which are usually of the following form:
  - ***Prove*** that your algorithm transforms "good" instances of Y into "good" instances of X.
  - ***Prove*** that your algorithm transforms "bad" instances of Y into "bad" instances of X. Equivalently: Prove that if your transformation produces a "good" instance of X, then it was given a "good" instance of Y.
- Argue that your algorithm for Y runs in polynomial time. In particular, it suffices to prove that your reduction runs in polynomial time if you use the simple form of reductions which will suffice for all the problems we will ask you.

---

1. This is to help you recall Boolean formulae. A Boolean function f over r variables $a_1, a_2, \ldots, a_r$ is a function $f : \{0,1\}^r \to \{0,1\}$ which assigns 0 or 1 to each possible assignment of values to the variables. One can specify a Boolean function in several ways including a truth table. Here is a truth table for a function on 3 variables a_1, a_2, a_3.

a_1	a_2	a_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

   Suppose we are given a Boolean function on r variables $a_1, a_2, \ldots, a_r$ via a truth table. We wish to express f as a CNF formula using variables a_1, a_2, a_3

   It may be easier to first think about expressing using a DNF formula (a disjunction of one more conjunctions of a set of literals). For instance the function above can be expressed as

$$(\bar{a}_1 \wedge \bar{a}_2 \wedge a_3) \vee (\bar{a}_1 \wedge a_2 \wedge \bar{a}_3) \vee (a_1 \wedge \bar{a}_2 \wedge a_3) \vee (a_1 \wedge a_2 \wedge \bar{a}_3) \vee (a_1 \wedge a_2 \wedge a_3).$$

   - What is a CNF formula for the function? *Hint:* Think of the complement function and complement the DNF formula.
   - Describe how one can express an arbitrary Boolean function f over r variables as a CNF formula over the variables using at most 2^r clauses.

2. A *Hamiltonian cycle* in a graph G is a cycle that goes through every vertex of G exactly once. Deciding whether an arbitrary graph contains a Hamiltonian cycle is NP-hard.

   A ***tonian cycle*** in a graph G is a cycle that goes through at least *half* of the vertices of G. Prove that deciding whether a graph contains a tonian cycle is NP-hard.

3. *Big Clique* is the following decision problem: given a graph $G = (V, E)$, does G have a clique of size at least $n/2$ where $n = |V|$ is the number of nodes? Prove that *Big Clique* is NP-hard.

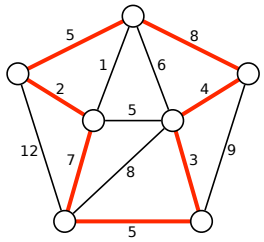
4. Recall the following kCOLOR problem: Given an undirected graph G, can its vertices be colored with k colors, so that every edge touches vertices with two different colors?

   (a) Describe a direct polynomial-time reduction from 3COLOR to 4COLOR.

   (b) Prove that kCOLOR problem is NP-hard for any $k \geq 3$.

To think about later:

5. Let G be an undirected graph with weighted edges. A Hamiltonian cycle in G is ***heavy*** if the total weight of edges in the cycle is at least half of the total weight of all edges in G. Prove that deciding whether a graph contains a heavy Hamiltonian cycle is NP-hard.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

6. **This is an advanced abstract problem for those interested.** An instance of *SAT* is a Boolean formula in CNF form. One may wonder about other kinds of formulas. One can reduce a large class of Boolean constraint satisfaction problems (referred to as CSPs) to *SAT*. CSPs are extensively investigated. For a fixed r there are at most 2^{2^r} Boolean functions on r variables (why?). Let us call this set of function $\mathcal{G}_r$. For each $\mathcal{F} \subseteq \mathcal{G}_r$ we obtain a constraint satisfaction problem defined by $\mathcal{F}$ that we call $\mathcal{F}$-SAT. An instance of $\mathcal{F}$-SAT consists of a set of n Boolean variables $x_1, x_2, \ldots, x_n$ and a set of m constraints (clauses). Each constraint is of the form $f_j(x_{j_1}, x_{j_2}, \ldots, x_{j_r})$ where $f_j \in \mathcal{F}$ and $j_1, j_2, \ldots, j_r$ are distinct indices in $\{1, 2, \ldots, n\}$. Note that no negated literals are technically allowed but one can capture negated literals via the functions in $\mathcal{F}$.

   We given an example. Let $r = 3$ and $\mathcal{F} = \{f_1, f_2\}$ where $f_1(a_1, a_2, a_3)$ is the function $a_1 \vee (\bar{a}_2 \wedge a_3)$ and $f_2(a_1, a_2, a_3)$ is the function $a_1 = (a_2 \wedge a_3)$. Consider an instance of $\mathcal{F}$-SAT on 4 variables x_1, x_2, x_3, x_4 with the following 5 constraints:

   $$f_1(x_1, x_2, x_3), f_1(x_2, x_3, x_4), f_2(x_2, x_1, x_4), f_2(x_1, x_2, x_3), f_1(x_1, x_3, x_4)$$

   An instance of $\mathcal{F}$-SAT is satisfiable if there is an assignment to $x_1, x_2, \ldots, x_n$ such that every constraint is satisfied (evaluates to 1).

- Show that 3-SAT a special case of $\mathscr{F}$-SAT for $r = 3$ where $\mathscr{F}$ is a set of 8 functions corresponding to *OR* over 3 variables and their negations.
- Show that any $\mathscr{F}$-SAT problem for fixed r can be reduced to r-SAT. *Hint:* Use the first problem.