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

- Choose a problem Y that you already know is NP-hard.
- 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 almost always requires two separate steps:
  - 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.

Recall that a *Hamiltonian cycle* in a graph G is a cycle that visits every vertex of G exactly once.

1. In class on Thursday, Jeff proved that it is NP-hard to determine whether a given *directed* graph contains a Hamiltonian cycle. Prove that it is NP-hard to determine whether a given *undirected* graph contains a Hamiltonian cycle.

2. A *double Hamiltonian circuit* in a graph G is a closed walk that goes through every vertex in G exactly *twice*. Prove that it is NP-hard to determine whether a given *undirected* graph contains a double Hamiltonian circuit.