

To prove a problem X is NP-complete, you need to show that it is both in NP and that it is at least as hard as any other problem in NP. This last step is typically done by showing that $Y \leq_P X$ for some problem Y already known to be NP-Complete. This is described in steps 2 through 5.

Step 1: Show that X is in NP. We want to argue that there is an efficient certifier for X . In other words, there is a certifier such that for any yes instance of X , there exists a certificate that the certifier will accept, and for any no instance of X , there is no certificate that the certifier will accept. The running time of the certifier (and hence the size of the certificate) must be polynomial. Typically, a solution to the given problem is a sufficient certificate. This step is very brief, but necessary.

Step 2: Pick a known NP-complete problem. State what problem Y you are reducing to X . You need to show that $Y \leq_P X$. You may use any problem Y which we have proved in class to be NP-complete, as well as any problem you have proved to be NP-complete on the homework assignments. By the very definition of NP-complete, if X is NP-complete, then you can technically use any other NP-complete problem Y to show this. However, some problems will be far easier to use than others in your proof. It is often useful to spend some time thinking about which of the problems you know to be NP-complete would be most natural to use for a given reduction.

Step 3: Construct an algorithm to solve Y given an algorithm to solve X . You need to show that any instance of Y can be solved using a polynomial number of operations, and a polynomial number of calls to a black box that can solve X . **Note:** It is very easy to get mixed up and instead prove that $X \leq_P Y$. Unfortunately, this is not what you want to show (we already know that Y is NP-complete).

Typically, you will show how to solve Y by constructing a single input to the black box for X , and typically, your algorithm will output the same answer as is given by the black box. However, this is not always the case.

Step 4: Prove the correctness of your algorithm. This requires proving an if and only if statement. You want to show that given a yes instance of Y your algorithm returns “yes”, *and* you want to show that if your algorithm returns “yes,” then the given input is indeed a yes instance of Y . It is always trivial to come up with an algorithm that satisfies just one of these two conditions. We want something that satisfies both.

Step 5: Polytime and wrap-up. Finally, you need to conclude that since your algorithm (typically a construction to be used with the black box for X) runs in polynomial time, $Y \leq_P X$. Since Y is NP-complete, and since we also have shown that X is in NP, X is NP-complete.

An Example: INDEPENDENT SET (IS)

Problem: An instance of independent set is given by $G = (V, E)$ and a natural number k . The goal is to determine whether you can select a subset S of k vertices such that no two vertices of S are adjacent in G .

Theorem: Independent Set is NP-Complete.

Proof. First, we argue that Independent Set is in NP, since given any set S , a verifier can efficiently check that S indeed contains at least k vertices, and that no two vertices of S are adjacent. So for a yes instance, we simply use an independent set of size k . And for a no instance, it is clear that no such set exists. Therefore, Independent Set is in NP.

We will now show that $3SAT \leq_P$ Independent Set. Given an instance of 3SAT, (that is, a formula $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_m$, where C_i is the disjunction of 3 variables, drawn from x_1, x_2, \dots, x_n and their negations, $\overline{x_1}, \overline{x_2}, \dots, \overline{x_n}$), we will construct an instance G of the Independent Set problem.

For each variable in each clause, create a node, which we will label with the name of the variable. Therefore there may be multiple nodes with the label x_i or $\overline{x_i}$, if these variables appear in multiple clauses. For each clause, add an edge between the three nodes corresponding the variables from that clause. We will call these three nodes and three edges a “clause gadget”. Finally, for all i , add an edge between every pair of nodes with one labeled x_i and the other labeled $\overline{x_i}$. We now run our black box for Independent Set on (G, m) and return the same result it gives.

To prove this answer is correct, we simply need to show that the original instance ϕ of 3SAT is a yes instance if and only if the Independent Set instance (G, m) we created is also a yes instance.

Suppose ϕ has a satisfying assignment A . Then at least one variable in each clause is satisfied by A . Define S to be a set of nodes in G defined by selecting one of the satisfied variable nodes in each clause gadget. Since we picked one node for each clause, there are clearly m nodes in S . Furthermore, since we only select a single node for each gadget, independence is not violated within any clause gadget. Finally, independence is not violated between clauses, since the only edges between clauses go between nodes with labels x_i and $\overline{x_i}$, and A can not have satisfied both of these, so we never would have selected both. Therefore S is an independent set of size m , so our algorithm will return yes.

Now suppose that our algorithm returns yes, that is, that there is an independent set S of size m in G . We now need to show that there is a satisfying assignment A of ϕ . Since S is independent, at most one node in each clause gadget must be used by S . But in fact, since there are exactly m clause gadgets, S must contain exactly one node from each clause gadget. Since S is independent, no pair of nodes x_i and $\overline{x_i}$ are ever both selected for S . Consider the following assignment. Set $A(x_i) = T$ if $x_i \in S$ and set $A(x_i) = F$ otherwise. Observe that this is a truth assignment, since all variables are assigned either T or F , but not both. Furthermore, A satisfies ϕ , since by our construction, A satisfies each clause.

We have now shown that our algorithm solves the 3SAT problem using a black box for the Independent Set problem. Since our construction takes polynomial time, and we have shown that Independent Set is in NP, we can conclude that Independent Set is NP-Complete. \square