1 CS302 Lecture notes – NP Completeness

- James S. Plank

- December 1, 2009.
- Latest revision: Dec 2 01:05:24 EST 2015

This is not a complete treatment of NP-Completeness. Like the Halting Problem lecture notes, they introduce you to a concept that you will see later in your CS careers and will provide you with fodder for endless conversations around the family dinner table.

As always, you can spend quite a bit of time reading Wikipedia on the subject. Their page is in This is not required reading, but (as of 2015) is a nice treatment of the topic.

P, NP, NP-Complete and NP-Hard are sets of problems, defined as follows:

- P: problems whose solution is polynomial time in the size of their inputs.
- NP: problems whose solutions can be verified in polynomial time. (NP stands for *non-deterministic polynomial time*).
- NP-Complete: A collection of problems in NP whose solutions may or may not polynomial time. We don't know. However, if we can prove that one of them may be solved in polynomial time, then all of them can.
- NP-Hard: A collection of problems that are not in NP, whose solutions are at least as hard as the NP-Complete problems.

In this lecture, we are going to see what it takes to prove that problems belong to these sets. Suppose you have a problem to solve, and you want to know its complexity class. This takes two steps:

1. *Prove that it is in NP.* Typically the problem is couched as a yes or no problem involving a data structure, such as "does there exist a simple cycle through a given directed graph that visits all the nodes?" To prove it is in NP, you need to show that a yes solution can be checked in polynomial time. In the above example, you can check to see if a given path through the graph is indeed a simple cycle in linear time. Therefore, the problem is in NP. You don't have to prove anything about the no solutions, and you don't have to prove anything about how you'd calculate a solution. ¡p¿ ¡/p¿

2. *Transform a known NP-Complete problem to this one in polynomial time.* Suppose the problem in question is Q, and that L is a well-known NP-Complete problem like the *3-satisfiability* problem. You need to show that if you have any instance of problem L, you can transform it into an instance of problem Q in polynomial time. Thus, if you could solve problem Q in polynomial time, you could solve problem L in polynomial time.

If you can do both of these things, then you have proved that a problem is NP-Complete. If you can prove that either of these things cannot be done, then you have proved that a problem is not NP-Complete. Sometimes you can't do come up with good proofs, and you just don't know.

The complexity classes P and NP-Hard may be put in terms of the above:

- **P**: If we can prove that the solution to a problem may be calculated in polynomial time, then the problem is in P. All of the algorithms that we have studied in this class, with the exception of enumeration, are in P. ¡p¿ ¡/p¿
- NP-Hard: These are problems that are not in NP; however, we can perform the transformation in step 2 of a known NP-Complete problem to these problems. Thus, they are *at least as hard* as the NP-Complete problems.