Algorithms

NP Completeness Continued

Homework 5

- Extension: due midnight Monday 22 April

Review: Tractibility

- Some problems are *undecidable*: no computer can solve them
  - E.g., Turing's "Halting Problem"
  - We don't care about such problems here; take a theory class
- Other problems are decidable, but *intractable*: as they grow large, we are unable to solve them in reasonable time
  - *What constitutes "reasonable time"?*

Review: **P**

- Some problems are provably decidable in polynomial time on an ordinary computer
  - We say such problems belong to the set **P**
  - Technically, a computer with unlimited memory
  - *How do we typically prove a problem $\in$ **P**?*

Review: **NP**

- Some problems are provably decidable in polynomial time on a nondeterministic computer
  - We say such problems belong to the set **NP**
  - Can think of a nondeterministic computer as a parallel machine that can freely spawn an infinite number of processes
  - *How do we typically prove a problem $\in$ **NP**?*
- *Is **P** $\subseteq$ **NP**? Why or why not?*

Review: **P** And **NP** Summary

- **P** = set of problems that can be solved in polynomial time
- **NP** = set of problems for which a solution can be verified in polynomial time
- **P** $\subseteq$ **NP**
- The big question: Does **P = NP**?

Review: NP-Complete Problems

- The *NP-Complete* problems are an interesting class of problems whose status is unknown
  - No polynomial-time algorithm has been discovered for an NP-Complete problem
  - No suprapolynomial lower bound has been proved for any NP-Complete problem, either
- *Intuitively and informally, what does it mean for a problem to be NP-Complete?*

Review: Reduction

- A problem P can be *reduced* to another problem Q if any instance of P can be rephrased to an instance of Q, the solution to which provides a solution to the instance of P
  - This rephrasing is called a *transformation*
- Intuitively: If P reduces in polynomial time to Q, P is "no harder to solve" than Q

An Aside: Terminology

- *What is the difference between a problem and an instance of that problem?*
- To formalize things, we will express instances of problems as strings
  - *How can we express a instance of the hamiltonian cycle problem as a string?*
- To simplify things, we will worry only about *decision problems* with a yes/no answer
  - Many problems are *optimization problems*, but we can often re-cast those as decision problems

NP-Hard and NP-Complete

- If P is *polynomial-time reducible* to Q, we denote this P $\leq_p$ Q
- Definition of NP-Hard and NP-Complete:
  - If all problems R $\in$ **NP** are reducible to P, then P is *NP-Hard*
  - We say P is *NP-Complete* if P is NP-Hard and P $\in$ **NP**
  - **Note: I got this slightly wrong Friday**
- If P $\leq_p$ Q and P is NP-Complete, Q is also NP- Complete

Why Prove NP-Completeness?

- Though nobody has proven that **P != NP**, if you prove a problem NP-Complete, most people accept that it is probably intractable
- Therefore it can be important to prove that a problem is NP-Complete
  - Don’t need to come up with an efficient algorithm
  - Can instead work on *approximation algorithms*

Proving NP-Completeness

- *What steps do we have to take to prove a problem P is NP-Complete?*
  - Pick a known NP-Complete problem Q
  - Reduce Q to P
    - Describe a transformation that maps instances of Q to instances of P, s.t. "yes" for P = "yes" for Q
    - Prove the transformation works
    - Prove it runs in polynomial time
  - Oh yeah, prove P $\in$ **NP** (*What if you can't?*)