CSE 421 Section 8

P, NP, Reductions

Administrivia

Announcements & Reminders

- HW 6
  - If you think something was graded incorrectly, submit a regrade request!
- HW 7
  - Was due yesterday, Wednesday 11/30
- HW 8
  - It’s the last homework, woohoo!
  - Due Wednesday 12/7
- Final Exam
  - Scheduled for **Monday 12/12 @ 2:30-4:20 in our normal room, KNE 220**

Some Definitions & A Strategy

First, some Definitions:

- **Problem**: a set of inputs and the correct outputs
- **Instance**: a single input to a problem
- **Decision Problem**: a problem where the output is “yes” or “no”
- **Reduction**: $A \leq B$
  - Informally: **A reduces to B** means **“we can solve A using a library for B”**
  - Formally: A reduces to B in **polynomial time** if there is an algorithm to solve problem A, which, if given access to a library function for solving problem B, calls the library at most polynomially-many times and takes at most polynomial-time otherwise excluding the calls to the library.

P, NP, and P vs. NP

- **P (“polynomial”)**: The set of all decision problems for which there exists an algorithm that runs in time $\mathcal{O}(n^k)$ for some constant k, where n is the size of the input
- **NP (“nondeterministic polynomial”)**: The set of all decision problems such that for every YES-instance (of size n), there is a certificate (of size $\mathcal{O}(n^k)$) for that instance which can be verified in polynomial time
- **P vs. NP**: Are P and NP the same complexity class?
  - That is, can every problem that can be **verified** in polynomial time also be **solved** in polynomial time?

NP-hard, NP-complete

- **NP-hard**: The problem B is NP-hard if for all problems A in NP, A reduces to B
- **NP-complete**: The problem B is NP-complete if B is in NP and B is NP-hard
- Why do we care about NP-hard and NP-complete?
  - Let B be an NP-hard problem.
  - Suppose you found a polynomial time algorithm for B; you now have for free a polynomial time algorithm for every problem in NP, so $P = NP$.
  - On the other hand, if any problem in NP is not in P (any doesn't have a polynomial time algorithm), then no NP-complete problem is in P

Goal of NP-Completeness Reductions

How do you remember which direction? The core idea of an NP-completeness reduction is a proof by contradiction:

Suppose, for the sake of contradiction, there were a polynomial time algorithm for B. But then if there were I could use that to design a polynomial time algorithm for problem A.

But we really, really, really don't think there's a polynomial time algorithm for problem A. So we should really, really, really think there isn't one for B either!

Key Idea: Reduce FROM the known hard problem TO the new problem.

Strategy for Reductions

1. Read and Understand the Problem
2. Design the Reduction
3. Write the Proof
    - Prove Run-Time
    - Prove correctness; requires TWO implications:
        - If the correct answer is YES, then our algorithm says YES
        - If our algorithm says YES, then the correct answer is YES

1. A Fun Reduction

Problem 1 – A Fun Reduction

Define `5-SAT` as the following problem:

Input: An expression in CNF form, where every term has exactly 5 literals.
Output: `true` if there is a variable setting which makes the whole expression true, `false` otherwise.

Prove that `5-SAT` is NP-complete.

Problem 1.1 – Read and Understand the Problem

First understand 5−SAT:

- What is the input type?
- What is the output type?
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"
- What is the output type for your reduction?

Problem 1.1 – Read and Understand the Problem

First understand 5−SAT:

- What is the input type? an expression in CNF form where each clause has 5 literals
- What is the output type?
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"

- What is the output type for your reduction?

Problem 1.1 – Read and Understand the Problem

First understand 5−SAT:

- What is the input type? an expression in CNF form where each clause has 5 literals
- What is the output type? true or false
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"

- What is the output type for your reduction?

Problem 1.1 – Read and Understand the Problem

First understand 5−SAT:

- What is the input type? an expression in CNF form where each clause has 5 literals
- What is the output type? true or false
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

CNF form is AND of ORs like $(z_a \vee z_d \vee z_f \vee z_h \vee z_i) \wedge \ldots \wedge (z_c \vee z_i \vee z_j \vee z_m \vee z_p)$
literals z_i are boolean variables or the negation of boolean variables x_i or $\neg x_i$

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"

- What is the output type for your reduction?

Problem 1.1 – Read and Understand the Problem

First understand 5−SAT:

- What is the input type? an expression in CNF form where each clause has 5 literals
- What is the output type? true or false
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

CNF form is AND of ORs like $(z_a \vee z_d \vee z_f \vee z_h \vee z_i) \wedge \ldots \wedge (z_c \vee z_i \vee z_j \vee z_m \vee z_p)$
literals z_i are boolean variables or the negation of boolean variables x_i or $\neg x_i$

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"

We need to reduce an NP-complete problem to 5-SAT in polynomial time. Assume we have an algorithm for 5-SAT. We want to solve 3-SAT. In other words, we want to show that 3-SAT ≤ 5-SAT.

- What is the output type for your reduction?

Problem 1.1 – Read and Understand the Problem

First understand 5−SAT:

- What is the input type? an expression in CNF form where each clause has 5 literals
- What is the output type? true or false
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

CNF form is AND of ORs like $(z_a \vee z_d \vee z_f \vee z_h \vee z_i) \wedge \ldots \wedge (z_c \vee z_i \vee z_j \vee z_m \vee z_p)$
literals z_i are boolean variables or the negation of boolean variables x_i or $\neg x_i$

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"

We need to reduce an NP-complete problem to 5-SAT in polynomial time. Assume we have an algorithm for 5-SAT. We want to solve 3-SAT. In other words, we want to show that 3-SAT ≤ 5-SAT.

- What is the output type for your reduction?

a Boolean which is the answer to the 3-SAT (which we get by calling 5-SAT like a library function)

Problem 1.2 – Design the Reduction

Now write a reduction. Remember a reduction is an algorithm! It often helps to think about the “certificates” (the thing that makes it a YES instance) and transform from one type of certificate to the other.

Problem 1.2 – Design the Reduction

Now write a reduction. Remember a reduction is an algorithm! It often helps to think about the "certificates" (the thing that makes it a YES instance) and transform from one type of certificate to the other.

Let $x_1, \ldots x_n$ be the variables in the 3-SAT instance and $C_1, C_2, \ldots, C_m$ be the clauses.

Create two dummy variables d_1, d_2. For each clause C_i, create four clauses:
$C_i \vee d_1 \vee d_2$
$C_i \vee \neg d_1 \vee d_2$
$C_i \vee d_1 \vee \neg d_2$
$C_i \vee \neg\ d_1 \vee \neg d_2$
Our 5-SAT instance is: $x_1, \ldots x_n, d_1, d_2$
The $4m$ clauses described above.

Problem 1.3 – Write the Proof

a) To be NP-Complete, 5-SAT needs to be in NP. Argue that it is (this argument is usually only 2-3 sentences).

b) Show your reduction is correct. Remember you need to prove two implications and that the running time is polynomial.

Problem 1.3 – Write the Proof

a) To be NP-Complete, you 5-SAT needs to be in NP. Argue that it is (this argument is usually only 2-3 sentences).

Problem 1.3 – Write the Proof

a) To be NP-Complete, you 5-SAT needs to be in NP. Argue that it is (this argument is usually only 2-3 sentences).

A verifier would take in the settings of the variables to true and false. Given a setting, a verifier would check that each clause (i.e., each constraint) is satisfied. This will take time linear in the length of the constraints, so it is polynomial time.

Problem 1.3 – Write the Proof

b) Show your reduction is correct. Running Time:

Problem 1.3 – Write the Proof

b) Show your reduction is correct. Running Time:

Running Time: Our algorithm makes 4 copies of every clause and adds a constant length set of literals to each clause, so the running time to create the instance is polynomial (and we call the library only once, which is also at most polynomial).

Problem 1.3 – Write the Proof

b) Show your reduction is correct. Suppose correct answer is YES and our reduction returns YES:

Problem 1.3 – Write the Proof

b) Show your reduction is correct. Suppose correct answer is YES and our reduction returns YES:

Let φ_3 be our 3-SAT instance and φ_5 be our 5-SAT instance.

Suppose φ_3 is satisfiable, we show that our reduction returns true. Since φ_3 is satisfiable, there is a setting of the variables which causes φ_3 to be true. Take that setting, and set d_1, d_2 arbitrarily. Every clause of φ_5 is a clause of φ_3 with extra literals ORed on, so since each clause of φ_3 is true, each clause of φ_5 is as well, and this is a satisfying assignment.

Problem 1.3 – Write the Proof

b) Show your reduction is correct. Suppose our reduction returns YES and correct answer is YES:

Problem 1.3 – Write the Proof

b) Show your reduction is correct. Suppose our reduction returns YES and correct answer is YES:

Conversely, suppose that our reduction returns true, and therefore φ_5 was satisfiable. Consider a satisfying assignment for φ_5. We claim that (ignoring d_1, d_2) the same assignment satisfies φ_3. Consider an arbitrary clause C_i of φ_3. In φ_5 there were four clauses built from C_i (each ORed with all combinations of literals of d_1, d_2. One of the created clauses in φ_5 had both inserted literals involving d_1, d_2 being false (since we included all possible combinations). Since φ_5 was satisfied, this clause evaluated to true, which means that C_i evaluated to true. Since C_i was arbitrary, we have that every clause is true, and therefore a satisfying assignment for φ_3, as required.

2. A Tricky Reduction

Problem 2 – A Fun Reduction

Define IND−SET as follows:
Input: An undirected graph G and a positive integer k
Output: true if there is an independent set in G of size k (or more), false otherwise.

And 3−SAT as in class
Input: expression in CNF form, where every term has exactly 3 literals.
Output: true if there is a variable setting which makes the whole expression true, false otherwise.

Prove that IND−SET is NP-complete using 3−SAT.

Problem 2.1 – Read and Understand the Problem

First understand `IND-SET`:

- What is the input type?
- What is the output type?
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"
- What is the output type for your reduction?

Work through these questions with the people around you, and then we'll go over them together!

Problem 1.1 – Read and Understand the Problem

First understand `IND-SET`:

- What is the input type? a graph
- What is the output type?
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"

- What is the output type for your reduction?

Problem 2.1 – Read and Understand the Problem

First understand `IND-SET`:

- What is the input type? a graph
- What is the output type? true or false
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"

- What is the output type for your reduction?

Problem 2.1 – Read and Understand the Problem

First understand `IND-SET`:

- What is the input type? a graph
- What is the output type? true or false
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

An independent set is a set of vertices so that G has no edges between any pair in the set

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"

- What is the output type for your reduction?

Problem 2.1 – Read and Understand the Problem

First understand `IND-SET`:

- What is the input type? a graph
- What is the output type? true or false
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

  An independent set is a set of vertices so that G has no edges between any pair in the set

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"

  We want to show that 3-SAT ≤ IND-SET. Assume we have an algorithm for IND-SET. We're trying to solve 3-SAT
- What is the output type for your reduction?

Problem 2.1 – Read and Understand the Problem

First understand `IND-SET`:

- What is the input type? a graph
- What is the output type? true or false
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

An independent set is a set of vertices so that G has no edges between any pair in the set

Then think about the reduction:

- Which problem are use solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is "going the right direction"

We want to show that 3-SAT ≤ IND-SET. Assume we have an algorithm for IND-SET. We're trying to solve 3-SAT

- What is the output type for your reduction?

a Boolean which is the answer to the 3-SAT (which we get by calling IND-SET like a library function)

Problem 2.2 – Design the Reduction

Now write a reduction. Remember a reduction is an algorithm! It often helps to think about the “certificates” (the thing that makes it a YES instance) and transform from one type of certificate to the other.

Work through this problem with the people around you, and then we’ll go over it together!

Problem 2.2 – Design the Reduction

Now write a reduction. Remember a reduction is an algorithm! It often helps to think about the "certificates" (the thing that makes it a YES instance) and transform from one type of certificate to the other.

The key idea is to think about solving 3-SAT as picking a literal from each clause and finding a truth assignment to make all of them true (where none of the literals we pick are in conflict, like x_i and $\neg\, x_i$). We want the independent set that IND-SET finds to correspond to those literals that make each clause true.

Let $x_1, \ldots x_n$ be the variables in the 3-SAT instance and $C_1, C_2, \ldots, C_m$ be the clauses.

Let G be the graph we construct to input into IND-SET. Add one vertex to G for each literal in a clause. For each clause, connect the 3 literals to form a triangle. IND-SET will pick at most one vertex from each clause, which will be set to true. To ensure that no conflicting literals are picked, connect all pairs of vertices that correspond to complementary literals. Let k be the number of clauses m.

Our 5-SAT instance is: The graph G and number k described above.

Problem 2.3 – Write the Proof

a) To be NP-Complete, IND-SET needs to be in NP. Argue that it is (this argument is usually only 2-3 sentences).

b) Show your reduction is correct. Remember you need to prove two implications and that the running time is polynomial.

Work through this problem with the people around you, and then we'll go over it together!

Problem 2.3 – Write the Proof

a) To be NP-Complete, IND-SET needs to be in NP. Argue that it is (this argument is usually only 2-3 sentences).

Problem 2.3 – Write the Proof

a) To be NP-Complete, IND-SET needs to be in NP. Argue that it is (this argument is usually only 2-3 sentences).

A verifier would take in the subset of k vertices that are independent. Given this set of vertices, a verifier would check that these vertices are actually independent (i.e. they are not adjacent). This will take time $\mathcal{O}(E + V)$, so it is polynomial time.

Problem 2.3 – Write the Proof

b) Show your reduction is correct. Running Time:

Problem 2.3 – Write the Proof

b) Show your reduction is correct. Running Time:

Running Time: Our algorithm adds every clause to the graph, adds edges for the 3 literals in each clause, and adds edges connecting complementary literals, so the running time to create the instance is polynomial (and we call the library only once, which is also at most polynomial).

Problem 2.3 – Write the Proof

b) Show your reduction is correct. Suppose correct answer is YES and our reduction returns YES:

Problem 2.3 – Write the Proof

b) Show your reduction is correct. Suppose correct answer is YES and our reduction returns YES:

Let φ_3 be our 3-SAT instance and G be our IND-SET instance.

Suppose φ_3 is satisfiable, we show that our reduction returns true. Since φ_3 is satisfiable, there is a setting of the variables which causes φ_3 to be true. Take that setting, it gives us that at least one literal in every constraint is true, and no conflicting literals are both true. To get the assignment that satisfies G, for each constraint, choose one of the true literals, and add the corresponding vertex to the independent set. This gives an independent set of the required size.

Problem 2.3 – Write the Proof

b) Show your reduction is correct. Suppose our reduction returns YES and correct answer is YES:

Problem 2.3 – Write the Proof

b) Show your reduction is correct. Suppose our reduction returns YES and correct answer is YES:

Conversely, suppose that our reduction returns true, and therefore φ_3 was satisfiable. Suppose there is an independent set of at least the size of the number of constraints. Because of the "in-constraint" edges in G, an independent set has at most one per constraint. Thus every constraint has exactly one vertex in the independent set. The variables of the φ_3 match to the chosen literals. No set literals could be conflicting because of the edges in G connecting corresponding literals, and we satisfy every constraint because we chose a good setting of one piece with the independent set. Therefore we have a satisfying assignment of φ_3!

That's All, Folks!

Thanks for coming to section this week!
Any questions?