Section 9: P, NP, and Reductions

1. SATisfy This

Determine whether each instance of 3SAT is satisfiable. If it is, list a satisfying variable assignment.

(a) $(\neg a \vee \neg b \vee c) \wedge (a \vee c \vee \neg d) \wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee b \vee c) \wedge (\neg b \vee c \vee \neg d)$

(b) $(\neg a \vee b \vee d) \wedge (\neg b \vee c \vee d) \wedge (a \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg d) \wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee c \vee \neg d) \wedge (a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$

(c) $(a \vee \neg c \vee d) \wedge (\neg a \vee b \vee c) \wedge (b \vee \neg c \vee \neg d) \wedge (a \vee \neg c \vee d) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee b \vee \neg d) \wedge (\neg a \vee c \vee d) \wedge (b \vee \neg c \vee d) \wedge (a \vee c \vee \neg d)$

(d) $(\neg a \vee \neg b \vee c) \wedge (a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee b \vee c) \wedge (a \vee \neg b \vee c)$

2. A Fun Reduction

Define 5SAT 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.

And 3SAT 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 5SAT is NP-complete using 3SAT.

2.1. Read and Understand the Problem

Read the problem and answer these quick-check-questions.

Make sure you understand 5SAT.

- What is the input type?
- What is the output type?
- Are any words in the problem technical terms? Do you know them all?

You're going to design a reduction – what will that reduction look like?

- Which problem are you 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?

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.

2.3. Write The Proof

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

  (b) How does a verifier check it efficiently?
- Show your reduction is correct. Remember you need to prove two implications **and** that the running time is polynomial.
  (a) Running time:

  (b) Correctness:

3. A Reduction between different kinds of problems

Define integer linear programming (ILP) as follows:
Input: An integer matrix A and integer vector b
Output: `true` if there is an integer vector x such that $Ax \leq b$, `false` otherwise.

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

We already know from class that 3SAT $\leq_P$ ILP by a long series of reductions. Prove this directly using a single reduction.

3.1. Read and Understand the Problem

Read the problem and answer these quick-check-questions.

Make sure you understand ILP and 3SAT.

- What is the input type?
- What is the output type?
- Are any words in the problem technical terms? Do you know them all?

You're going to design a reduction – what will that reduction look like?

- Which problem are you 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?

3.2. 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.

3.3. Write The Proof

(a) To be NP-Complete, ILP 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.

4. Reduce to decision

NP is a set of decision (yes/no) problems, but in practice we're often interested in optimization problems (instead of "is there a vertex cover of size k?" we usually want to "find the smallest vertex cover"). **Usually**, this isn't a problem, though; we'll see an example in this problem.

Let $\mathsf{VC_D}$ be the problem: Given a graph G and an integer k, return `true` if and only if G has a vertex cover of size k. Let $\mathsf{VC_O}$ be the problem: Given a graph G, return a list containing the vertices in a minimum size vertex cover.

(a) Show that $\mathsf{VC_D} \leq_P \mathsf{VC_O}$ (this is the easy direction).

(b) We'll now start working on the other reduction. Imagine someone came to you and said "See this vertex u, I promise it is in a minimum vertex cover." Use this promise to solve $\mathsf{VC_O}$ on a graph of size $n-1$ instead of n.

(c) Now imagine the same person said "See this vertex v, I promise it is **not** in any minimum vertex cover." Use this promise to solve $\mathsf{VC_O}$ on a graph of size at most $n-1$ instead of n.

(d) Use the ideas from the last two parts to show $\mathsf{VC_O} \leq_P \mathsf{VC_D}$.

5. Another Reduction

Consider an undirected graph G, where each vertex has a non-negative integer number of pebbles. A single *pebbling move* consists of removing two pebbles from a vertex and adding one pebble to an adjacent vertex, where we can choose which adjacent vertex. A pebbling move can only be done on a vertex that already has at least two pebbles, and it will always decrease the total number of pebbles in the graph by exactly one. Our goal is to remove as many pebbles as we can. Observe that at best, we'll have at least one pebble remaining in the graph.

Define the PEBBLE problem as the following problem:

`Input`: An undirected graph and the number of pebbles at each vertex `Output`: `true` if there is a sequence of pebbling moves that leaves exactly one pebble in the graph, `false` otherwise.

Define the Hamiltonian Path Problem as the following problem:

Input: An undirected graph.

Output: `true` if there exists a path in the graph visiting every vertex exactly once, `false` otherwise.

Given that the Hamiltonian Path Problem is NP-complete, show that PEBBLE is as well. You may assume that the total number of pebbles in a graph is polynomial in terms of the size of the graph.

Hint: A single pebbling move can be represented as an ordered pair of vertices (u, v) where we take two pebbles from u and place one pebble in its neighbor v. A sequence of pebbling moves can be represented by a sequence of these pairs. Is there any way we can order these pairs nicely?

5.1. Read and Understand the Problem

Read the problem and answer these quick-check-questions.

Make sure you understand PEBBLE and the Hamiltonian Path Problem.

- What is the input type?
- What is the output type?
- Are any words in the problem technical terms? Do you know them all?

You're going to design a reduction – what will that reduction look like?

- 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?

5.2. 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.

5.3. Write The Proof

(a) to be NP-Complete, PEBBLE 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.

6. Vertex Cover and Independent Set

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 at least k, `false` otherwise.

Define VER-COVER as follows:
Input: An undirected graph G and a positive integer k
Output: `true` if there is an vertex cover in G of size at most k, `false` otherwise.

Prove that VER-COVER is NP-complete using IND-SET.

6.1. Read and Understand the Problem

Read the problem and answer these quick-check-questions.

Make sure you understand IND-SET and VER-COVER.

- What is the input type?
- What is the output type?
- Are any words in the problem technical terms? Do you know them all?

You're going to design a reduction – what will that reduction look like?

- Which problem are you 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?

6.2. 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.

6.3. Write The Proof

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

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