CS311: Homework #8

Due on December 13, 2013 at 4:30pm

Professor Lathrop Section 3

Josh Davis

Note:

I've already taken **ComS 331**, so that explains the more formal treatment of these proofs. I just don't want the formalness to be mistaken as cheating. =]

Cheers!
Josh

Problem 1

Prove or disprove: P $\subseteq$ NP

Solution

Proof. To prove that P $\subseteq$ NP, lets first look at what it means to be in P.

A given problem A is in P if it can be solved/decided in polynomial time. A given problem is in NP if it can be verified by a verifier in polynomial time.

Let A be a problem in P. This means that A can be solved by an algorithm in polynomial time.

Thus given a certificate c that is a solution to A, it easily follows that c can be verified to be a solution to A because c was originally found in polynomial time because A is in P.

Thus since A can be verified in polynomial time by such a verifier, it proves that A is also in NP. This shows that $P \subseteq NP$ and thus concludes the proof. $\square$

Problem 2

Consider the Hitting Set problem which is defined as:

Hitting Set

Input: A collection C of subsets of a set S, a positive integer k.
Output: Does S contain a subset S' such that $|S'| \leq k$ and each subset in C contains one element from S'?

Part A
Prove that Hitting Set *in* NP.

Proof. To prove that Hitting set in NP, we must prove that there exists a verifier such that given a valid certificate c, it can be verified in polynomial time.

Let V be such a verifier for the Hitting Set:

$V =$“ On input $\langle\langle S, C, k\rangle, c\rangle$ where S is the set and C is the collection of subsets and k is a positve integer:

1. Decode the certifcate into a new set T.
2. Check that T is a valid subset of S.
3. Check that $|T| \leq k$.
4. Check that for every subset in C, there is at least one item in C that is in T.
5. If all the checks pass, *accept*, else *reject*.

□

Given the input, all calculates operate polynomially on the length of the input, thus the Hitting Set problem is in NP.

Problem 2 continued on next page. . .

Part B

Prove that Hitting Set is hard for NP.

Proof. To prove that the Hitting Set is NP-hard, we must show that a problem already in NP-complete is polynomially reducible to the Hitting Set.

Let us consider the $VETEX - COVER$ problem, which tells us whether or not there is a k-node vertex cover in the graph. $VERTEX - COVER$ is known to be NP-complete thus we can create a polynomial reduction from it to the Hitting Set problem.

Let M be a Turing machine that solves the Hitting Set problem. We can then reduce $VETEX - COVER$ to the Hitting Set, $VETEX - COVER \leq_p$ Hitting Set, problem by constructing a new Turing machine called N that solves $VERTEX - COVER$ and that is constructed as follows:

$N =$" On input $\langle G, k\rangle$ where G is a graph and k is a positve integer:

1. Let $S = G.V$, the vertices in the graph G.
2. Let $C = G.E$ where each edge, e, is a subset consisting of just $\{u, v\}$ where $(u, v) = e$.
3. Run M on the input $\langle S, C, k\rangle$.
4. If M accepts, then *accept*, else *reject*."

This runs in polynomial time because it doesn't exceed polynomial time in the amount of work done outside of M.

Since the reduciton is polynomial, the Hitting Set problem is NP-hard. □

Reference

1. Vertex-Cover details from *Introduction to the Theory of Computation*, *2nd* edition, by Michael Sipser, pg. 288

Part C

Prove that Hitting Set is complete for NP.

Proof. The qualifications for being NP-complete are the following:

1. The problem is in NP.
2. The problem is in NP-hard.

The first part was solved in **Part A**, and the second part was solved in **Part B**, by definition this means that the Hitting Set problem is in NP-complete and we conclude the proof. □

Problem 3

Show that the following problem is NP-complete.

Dense Subgraph

Input: A graph G and integers k and y.
Output: Does G contain a subgraph with exactly k vertices and at least y edges?

Solution

Proof. To prove that the Dense Subgraph problem is in NP-complete, we must first show the following:

1. The problem is in NP.
2. The problem is in NP-hard.

Part One
First let's show that Dense Subgraph is in NP. To prove that Dense Subgraph is in NP, we must prove that there exists a verifier such that given a valid certificate c, it can be verified in polynomial time.

Let V be such a verifier for Dense Subgraph where c is a valid certificate thus a subgraph:

$V =$" On input $\langle\langle G, k, y\rangle, c\rangle$:

1. Decode the certificate c into a new graph, G'.
2. Test that all nodes and edges in G' are in G.
3. Count up the number of vertices in G' and check that it equals k.
4. Count up the number of edges in G' and check that it equals y.
5. If everything checks out, *accept*, else *reject*."

The verifier V thus runs in time less than $O(V + E)$ where $V = |G.vertices|$ and $E = |G.edges|$ because G' is a subgraph of G. Since the verifier runs in polynomial time, Dense Subgraph is in NP.

Problem 3 continued on next page. . .

Part Two
Second we must show that a problem already in NP-complete is polynomially reducible to Dense Subgraph.

Let us consider the $VETEX - COVER$ problem, which tells us whether or not there is a k-node vertex cover in the graph. $VERTEX - COVER$ is known to be NP-complete thus we can create a polynomial reduction from it to the Dense Subgraph problem.

Let M be a Turing machine that solves the Dense Subgraph problem. We can then reduce $VETEX - COVER$ to the Dense Subgraph, $VERTEX - COVER \leq_p$ Dense Subgraph, by constructing a new Turing machine called N that solves the $VERTEX - COVER$ problem as follows:

$N =$" On input string $\langle G, k \rangle$ where G is a graph and k is an integer:

1. Let y be a value related to k that would make it dense (there are a few different ways to define what makes a graph dense, all can be ran in polynomial time, thus irrelevant).
2. Run M on the input $\langle G, k, y \rangle$.
3. If M accepts, then *accept*, else *reject*."

Clearly this is polynomial because the additional steps can be performed in polynomial time outside of calling M.

Conclusion
Since we have proven both conditions for NP-complete, the problem of Dense Subgraph is thus NP-complete and this concludes the proof. □

Reference

1. Vertex-Cover details from *Introduction to the Theory of Computation*, *2nd* edition, by Michael Sipser, pg. 288

Problem 4

Show that the following problem is NP-complete:

Longest Path

Input: A graph G and positive integer k.
Output: Does G contain a path that visits at least k different vertices without visiting any vertex more than once?

Solution

Prove that the Longest Path problem is NP-complete.

Proof. To prove that the Longest Path problem is in NP-complete, we must first show the following:

1. The problem is in NP.
2. The problem is in NP-hard.

Part One
First let's show that Longest Path is in NP. To prove that Longest Path is in NP, we must prove that there exists a verifier such that given a valid certificate c, it can be verified in polynomial time.

Let V be such a verifier for Longest Path where c is a valid certificate thus a list of nodes along the path:

$V =$" On input $\langle\langle G, k\rangle, c\rangle$:

1. Iterate over every node in the certficate c and check that each node is in G.
2. Count up the number of nodes in c and check that it equals k.
3. If everything checks out, *accept*, else *reject*."

Clearly this runs in polynomial time because it only iterates over the list of nodes in the path of size k twice.

Problem 4 continued on next page. . .

Part Two
Second we must show that a problem already in NP-complete is polynomially reducible to Longest Path.

Let us consider the $HAMPATH$ problem, which tells us whether or not there is a path through the graph that visists each vertex only once. $HAMPATH$ is known to be NP-complete thus we can create a polynomial reduction from it to the Longest Path problem.

Let M be a Turing machine that solves the Longest Path problem. We can then reduce $HAMPATH$ to the Longest Path, $HAMPATH \leq_p$ Longest Path by construct a new TM N that is created as follows:

$N =$" On input string $\langle G \rangle$ where G is a graph:

1. Let $k = |G.vertices| - 1$ because the longest path in a $HAMPATH$ problem *must* be the Hamiltonian Path as well.
2. Run M on the input $\langle G, k \rangle$.
3. If M accepts, then *accept*, else $reject$."

Clearly this is polynomial because besides running M, it only takes constant time.

Conclusion
Since we have proven both conditions for NP-complete, the problem of Longest Path is thus NP-complete and this concludes the proof. □

Reference

1. $HAMPATH$ details from *Introduction to the Theory of Computation*, 2*nd* edition, by Michael Sipser, pg. 260

Problem 5

Why doesn't the following algorithm suffice to prove it is in P, since it runs in $O(n)$ time?

```
function PrimalityTesting(n)
    composite ← false

    for all i ∈ 2 ... n − 1 do
        if n mod i = 0 then
            composite ← true
        end if
    end for

    return composite
end function
```

Algorithm 1: Simple primality test

Solution

This is a common mistake when first learning about the complexity classes. The reason is that **PrimalityTesting(n)**, actually *doesn't* run in polynomial time, or $O(n)$.

The reason for this is because in order for it to be in P, it must run in polynomial time with respect to the *length* of n, not the value.

Thus the actual length of n, let this be n_0, is actually $n_0 = 1 + \log_{10} n$ for any positive integer n. Thus in order for **PrimalityTesting(n)** to run in polynomial time, it must run in $O(n_0) = O(\log n)$ time if n is the number to check.