CS570 - Spring 2016 - HW11 - Solution

1. State True/False. If $A \leq_p B$ and $B \in NP$, then $A \in NP$. True. The important observation is that you can construct a certifier for A by composing the polynomial time reduction map and the certifier for B. A proof sketch for Karp reductions follows if you are interested.

   Proof : $B \in$ **NP** implies there exists a polynomial time certifier C_B. That is, there exists a constant k such that, if x is a "no" instance for B then $\forall c, C_B(x, c) = 0$ and if $x \in B$ then there exists a c_x such that $||c_x|| \leq ||x||^k$ and $C_B(x, c_x) = 1$. Here, $||y||$ denotes the number of bits used to write y.

   Let f be a polynomial time reduction from A to B with running time bounded by a polynomial of degree d. That is, f is a polynomial time algorithm that maps instances of A to instances of B, such that x is an "yes" instance of A if and only if $f(x)$ is a "yes" instance of B. Compose the algorithms C_B and f to obtain the polynomial time algorithm C_A, that is

   $$\forall x, \forall c, C_A(x, c) := C_B(f(x), c)$$

   We claim that C_A is a polynomial time certifier for A with certificate size bounded by kd bits.

2. State True/False. If $A \leq_p B$ and $A \in$ *NP-complete*, then $B \in$ *NP-complete*.

   False. If $A \leq_p B$ and $A \in$ *NP-complete*, then B is not necessarily in $NP-$complete (since B need not be in NP).

3. State True/False. Assume you have a polynomial time algorithm that given a 3-SAT instance, decides in polynomial time if it has a satisfying assignment. Then you can build a polynomial time algorithm that finds a satisfying assignment(if it exists) to a given 3-SAT instance.

   True. Let A be a polynomial time algorithm that decides 3-SAT.

   Let $\phi(x_1, x_2, \ldots, x_n) = c_1 \wedge c_2 \wedge \ldots \wedge c_m$ be the boolean formula corresponding to a 3-SAT instance where $c_1, c_2, \ldots, c_m$ are the clauses and $x_1, x_2, \ldots, x_n$ are the variables.

If $A(\phi((x_1, x_2, \ldots, x_n))) = 0$, then return that the instance is non satisfiable.

If $A(\phi((x_1, x_2, \ldots, x_n))) = 1$, then we can find an assignment for x_1 as follows. If $A(\phi((1, x_2, \ldots, x_n))) = 1$, then we can set $x_1 = 1$ and be guaranteed that $\phi_1 := \phi(1, x_2, \ldots, x_n)$ is satisfiable. Else, we can set $x_1 = 0$ and be guaranteed that $\phi_1 := \phi(0, x_2, \ldots, x_n)$ is satisfiable.

In either case, we know ϕ_1 has a satisfying assignment. That is, there exists a satisfying assignment for ϕ that assigns to x_1 the same assignment that our algorithm made.

In the second iteration, we find an assignment for x_2 given that we have already made a choice for x_1. If $A(\phi_1(1, x_3, \ldots, x_n)) = 1$, we can set $x_2 = 1$ and $\phi_2(x_3, \ldots, x_n) = \phi_1(1, x_3, \ldots, x_n)$. Else we can set $x_2 = 0$ and $\phi_2(x_3, \ldots, x_n) = \phi_1(0, x_3, \ldots, x_n)$.

By iterating, we find an assignment for the variables such that ϕ_n (which is nothing but the evaluation of ϕ at this assignment) is 1.

4. State True/False. If someone proves $P = NP$, then it would imply that every decision problem can be solved in polynomial time.

   False. If $P = NP$, then we can conclude that every problem in NP can be solved in polynomial time. However, there are decision problems (that are not in NP) that are known to not have polynomial time algorithms. One such example is the Halting problem which cannot be solved even if there were no restriction on the resources (like time or space).

5. State True/False. Assume $P \neq NP$. Let A and B be decision problems. If $A \in NP\text{-}Complete$ and $A \leq_P B$, then $B \notin P$.

   True. If B were in **P**, then $A \leq_p B$ would imply $A \in \mathbf{P}$. Since $A \in$ **NP-Complete**, $\forall D \in \mathbf{NP}$, $D \leq_p A$. Since $A \leq_p B$, this implies $\forall D \in$**NP**, $D \in \mathbf{P}$ which contradicts $\mathbf{P} \neq \mathbf{NP}$.

6. State True/False. Assume $P \neq NP$. Let A and B be decision problems. If $A \in P$ and $B \in NP\text{-}Complete$, then $A \leq_P B$.

   True. $B \in$ **NP-Complete** implies $\forall D \in \mathbf{NP}$, $D \leq_P B$. Since $\mathbf{P} \subseteq \mathbf{NP}$, $A \leq_P B$. (Note that we dont use the assumption $\mathbf{P} \neq \mathbf{NP}$)

7. Given an n bit positive integer, the problem is to decide if it is composite. Here the problem size is n. Is this decision problem in **NP** ?

   Yes. For every "yes" instance (the number is composite), a factor of the

number is a certificate. Certification proceeds by dividing the number by the factor and making sure that the reminder is zero and also making sure that the certificate is neither 1 nor the input number itself. The factor is at most n bits and verification can be done in time polynomial in n. Thus deciding if a number is composite is in **NP**.

8. Show that vertex cover remains *NP-Complete* even if the instances are restricted to graphs with only even degree vertices.

   Let $< G; K >$ be an input instance of VERTEX-COVER, where $G = (V; E)$ is the input graph.

   Because each edge in E contributes a count of 1 to the degree of each of the vertices with which it is incident, the sum of the degrees of the vertices is exactly $2|E|$, an even number. Hence, there is an even number of vertices in G that have odd degrees.

   Let U be the subset of vertices with odd degrees in G.

   Construct a new instance $< \bar{G}; k + 2 >$of VERTEX-COVER, where $\bar{G} = (V_0; E_0)$ with $V_0 = V \cup \{x, y, z\}$ and $E_0 = E \cup \{(x, y), (y, z), (z, x)\} \cup \{(x, v)|v \in U\}$. In words, we make a triangle with the three new vertices, and then connect one of them (say x) to all the vertices in U.

   The degree of every vertex in V_0 is even. Since a vertex cover for a triangle is of (minimum) size 2, it is clear that $\bar{G}$ has a vertex cover of size $k + 2$ if and only if G has a vertex cover of size k.

9. Given an undirected graph $G = (V, E)$, the Half-Clique problem is to decide if there is a subset $A \subseteq V$ of vertices satisfying the following two conditions:

   (i) $|A| \geq \frac{|V|}{2}$
   (ii) For every pair of vertices $u, v \in A$, if $u \neq v$, then $(u, v) \in E$.

   Show that Half-Clique is in *NP-Complete*. You are allowed to use the fact that Independent-Set is in *NP-Complete*.

   Given a set of vertices A as the certificate, it is easy to verify that the two conditions listed in the question are satisfied. Thus Half-Clique is in NP.

   We will reduce Independent set to Half-Clique in two steps. The intermediate step concerns the Clique problem.

   We begin by defining a few terms that we require. A subset of vertices is called as a clique if and only if every distinct pair of vertices in the subset is connected by an edge. Given a graph and a number m, the clique

problem is to decide if the graph has a clique of size m. For a graph G_1, its complement (denoted by $\bar{G_1}$) is defined as the graph that has the same vertex set as G_1, but with the edge incidence inverted. That is, an edge e is in G_1 if and only if e in not in $\bar{G_1}$.

Observe that a set of vertices B is an independent set in G_1 if and only if it is a clique for its complement $\bar{G_1}$. Thus an Independent Set instance (G_1, k) can be reduced to the Clique problem by mapping it to the Clique instance $(\bar{G_1}, m = k)$. Thus

$$Independent - Set \leq_P Clique.$$

Given a clique instance $(G_2 = (V_2, E_2), m)$ we next reduce it to a Half-Clique instance. If $m = |V_2|/2$, then we already have a Half-Clique instance.

If $m < |V_2|/2$, then add $|V_2| - 2m$ new vertices to G_2, between every distinct pair of new vertices add an edge and between every new vertex and every existing vertex add an edge. The new graph has $2(|V_2| - m)$ vertices. The new graph has a clique of size at least $|V_2| - m$ if and only if G_2 had a clique of size m (prove this !).

If $m > |V_2|/2$, then add $2m - |V_2|$ new vertices to G_2 and do not introduce any new edges. The new graph has $2m$ vertices. The new graph has a clique of size at least m if and only if G_2 had a clique of size m (prove this !).

Thus we can conclude that

$$Clique \leq_P Half - Clique.$$

From the transitivity of polynomial time reductions, it follows that

$$Independent - Set \leq_P Half - Clique.$$

10. Given an undirected graph G and a positive integer k, consider the decision problem which asks if a simple path (no repeating vertices) of length at least k exists.

    Is this decision problem in NP ? Assuming P $\neq$ NP, is it in P ?

    Yes, the decision problem is in **NP**. A simple path of length k is a certificate and can be verified by traversing the path (making sure all the edges in the path are indeed in G, the length is indeed k and that there are no repeated vertices).

    We claim that the problem is **NP** complete (which assuming P $\neq$ NP

would imply that it is not in P).

Call the decision problem in question K-PATH.

Consider the Hamiltonian Path problem (HAM-PATH) where given a graph with n vertices, we have to decide if it contains a simple path that visits all nodes. Clearly HAM-PATH $\leq_p$ K-PATH, since HAM-PATH is a special case of K-PATH (with $k = n - 1$).

It turns out that HAM-CYCLE $\leq_p$ HAM-PATH, where HAM-CYCLE is the Hamiltonian cycle problem. We show this by the following reduction. Let $\bar{G}$ be the graph input to HAM-CYCLE. Choose one vertex u in $\bar{G}$ and duplicate it, i.e. add another vertex u' and for each edge (u, v) add the edge (u', v). Also add two more vertices t and t' and the edges (t, u) and (t', u'). It is fairly easy to see that the new graph has a Hamiltonian Path if and only if G has a Hamiltonian Cycle.

By transitivity of polynomial time reductions, it follows that HAM-CYCLE $\leq_p$ K-PATH. Thus K-PATH is **NP**-complete which assuming **P**≠**NP** implies that K-PATH is not in **P**.

11. Given an undirected graph with positive edge weights, the BIG-HAM-CYCLE problem is to decide if it contains a Hamiltonian cycle C such that the sum of weights of edges in C is at least half of the total sum of weights of edges in the graph. Show that BIG-HAM-CYCLE is NP-Complete. You are allowed to use the fact that deciding if an undirected graph has a Hamiltonian cycle is NP-complete.

    The certifier takes as input an undirected graph (the BHC instance) and a sequence of edges (certificate). It verifies that the sequence of edges form a Hamiltonian cycle and that the total weight of the cycle is at least half the total weight of the edges in the graph. Thus BIG-HAM-CYCLE is in NP. We claim that Hamiltonian Cycle is polynomial time reducible to BIG-HAM-CYCLE. To see this, given an undirected graph $G = (V, E)$ (instance of HC), pick an edge e and set its weight to $|E|$ and assign the rest of the edges a weight of 1. When this weighted graph is fed into the BIG-HAM-CYCLE decider blackbox, it returns "yes" if and only if G has a Hamiltonian cycle containing the edge e. By repeating the above once for every edge e in the graph G, we can decide if the graph has a Hamiltonian cycle.

12. You are given an undirected graph $G = (V, E)$ and for each vertex $v \in V$, you are given a number $p(v)$ (which is either 0 or 1 or 2) that denotes the number of pebbles (stones) placed on v. We will now play a game where the following move is the only move allowed. You can pick a vertex u that contains at least two pebbles, and remove two pebbles from u and add one pebble to one (your choice) of the neighboring vertices of u. The objective

of the game is to perform a sequence of moves such that we are left with exactly one pebble in the whole graph. Show that the problem of deciding if we can reach the objective is NP-complete. Call our decision problem Pebble. For an instance, given a sequence of moves as a certificate, we can verify efficiently if each move is valid and that we are left with a single pebble in the graph. Thus Pebble is in NP.

We claim that HAM-PATH $\leq_P$ Pebble. Unlike in most other problems that we have encountered, we will use a black box that solves Pebble more than once.

Let $G = (V, E)$ be an instance of HAM-PATH. Pick a vertex $s \in V$ as the starting vertex, place 2 pebbles at s and at every other vertex, place 1 pebble. We claim that this instance of the Pebble problem is an "yes" instance is and only if there is a Hamiltonian path starting from s. To see this observe that the only allowed move in the first step is to go from s to a neighbouring vertex (say u). After this move, there are no pebbles left at s and 2 pebbles at u. By induction, we see that for every vertex that we leave, there are no pebbles left and at every new vertex that we arrive at there are 2 pebbles. Thus we can never revisit a vertex (otherwise, we get stuck since that vertex would only have a single pebble after arrival). To be left with only a single pebble in the graph, prior to the last move, the whole graph should have had exactly 2 pebbles, both of which on a single vertex. This can happen and can only happen if there is a Hamiltonian path starting from s.

Thus by calling the blackbox once for every starting vertex s, we can decide is G has a Hamiltonian path.

13. Assume that you are given a polynomial time algorithm that decides if a directed graph contains a Hamiltonian cycle. Describe a polynomial time algorithm that given a directed graph that contains a Hamiltonian cycle, lists a sequence of vertices (in order) that form a Hamiltonian cycle.

    Let $G = (V, E)$ be the input graph. Let A be an algorithm that decides if a given directed graph has a Hamiltonian cycle. Hence $A(G) = 1$.

    Pick an edge $e \in E$ and remove it from G to get a new graph $\bar{G}$.
    If $A(\bar{G}) = 1$, then there exists a Hamiltonian cycle in $\bar{G}$ which is a subgraph of G, set $G = \bar{G}$.
    If $A(\bar{G}) = 0$, then every Hamiltonian cycle in G contains e. Put e back into G.

    Iterate the above three lines until we are left with exactly $|V|$ edges. Since after each step we are left with a subgraph that contains a Hamiltonian cycle, at termination we are left with the set of edges that forms a Hamil-

tonian cycle. Starting from an edge, do a BFS to enumerate the edges of the Hamiltonian cycle in order.