1 Problem 8.5

Perfect Completeness

Notice that we can alter the Set Lower Bound Protocol by making the prover send the hash function $h$ to the verifier initially. This clearly doesn’t hurt completeness, and it doesn’t hurt soundness because the verifier still picks a $y$ at random after the prover picks an $h$, so the prover effectively hasn’t gained any additional knowledge compared to the original protocol. We’ll also alter the protocol to distinguish between the cases $|S| \geq K$ and $|S| \leq K/(2k)$. Now suppose that the prover chooses multiple hash functions $h_1, \ldots, h_l$ and sends them all to the verifier. Then the verifier picks a random $y$ and asks the prover for an $x$ and $i$ such that $h_i(x) = y$ and $x \in S$. To get perfect completeness, we need only show that there’s always a set of hash functions that can be chosen such that $\bigcup_i h_i(S) = \{0,1\}^k$, since that implies that the prover will always be able to find a desired $x$. We now consider the probability: $p = Pr[\bigcup_i h_i(S) = \{0,1\}^k] = 1 - Pr[\exists z : z \notin \bigcup_i h_i(S)] \geq 1 - \Sigma_i Pr[z \notin \bigcup_i h_i(S)]$. The latter equation follows from union bound. Now, since the hash functions are chosen uniformly at random from a pairwise-independent family, they must distribute $S$ uniformly over $\{0,1\}^k$. In other words, for all $z$ and $i$, $Pr[z \notin h_i(S)] = 1 - |S|/2^k$. Thus for all $z$, $Pr[z \notin \bigcup_i h_i(S)] = Pr[\bigcap_i z \notin h_i(S)] = (1 - |S|/2^k)^i$, and so $p \geq 1 - 2^k(1 - |S|/2^k)^i$. If $|S| \geq K$, then $|S| > 2^{k-2}$, and so $|S|/2^k > 1/4$. Thus $p \geq 1 - 2^k(3/4)^i$. Picking $l = 3k$, we get $p \geq 1 - 2^k(27/64)^k = 1 - (54/64)^k > 0$. So we see that, if we choose $3k$ hash functions uniformly at random, the probability of covering $\{0,1\}^k$ is non-zero, which means that there must be some particular set of $3k$ hash functions that covers it. Hence there exists a prover which picks that particular set, making the verifier correctly accept with probability 1.

For soundness, we need only show that there is no set of $3k$ hash functions that cover $\{0,1\}^k$ if $|S| \leq K/(2k)$, since this means no prover can guarantee the verifier will always accept. This creates a gap between the completeness probability of 1 and the soundness probability of less than 1, which can be sufficiently expanded using Chernoff bounds. Given that $|S| \leq K/(2k)$, the maximum number of elements in the range that can be mapped to is
\(|S| = 3k|S|\) (since each \(h_i(S)\) can map to at most \(|S|\) elements). Now
\(3k|S| \leq 3k(\frac{K}{2k}) \leq 3(2^{k-2}) < 4(2^{k-2}) = 2^k\). Since we can only map to
fewer than \(2^k\) elements, it’s impossible to cover \(\{0, 1\}^k\), so we’re done.

2 Problem 8.9

Downward-Self-Reducibility

Suppose \(L\) is downward-self-reducible, and let \(M\) be a poly-time TM
deciding \(L\) that has access to an oracle for all smaller input instances. Let
\(M'\) be a TM that simulates \(M\), and whenever \(M\) makes an oracle query, \(M'\) saves
the entire configuration of \(M\) and recurses on the input to the oracle query.
Since all oracle queries can only be for smaller instances, \(M'\) will always
terminate, and the maximum depth of recursion is polynomial. Furthermore,
the total size of a configuration of \(M\) can only be polynomial since \(M\) is
poly-time, so assuming we reuse the space that stores \(M'\)’s configuration at a
particular recursion depth, \(M'\) decides \(L\) using only polynomial space. Hence
\(L \in \text{PSPACE}\).

3 Problem 8.10

Graph Isomorphism Checker

Let \(A(G, i)\) be a function that takes a graph \(G\), adds two new vertices \(v_b\) and \(v_s\) to \(G\), and adds edges \((v_b, v)\) for each \(v \in V(G)\), edge \((v_b, v_b)\), and
edge \((v_s, v_i)\). For the remainder of this solution, assume \(u's\) refer to vertices
in \(G_1\), and \(v's\) refer to those in \(G_2\).

\textbf{Claim 1:} \(A(G_1, i) \cong A(G_2, j) \iff \exists \pi. \pi \) is an isomorphism from \(G_1\) to \(G_2 \land \pi(u_i) = v_j\)

(\(\Rightarrow\)) Suppose \(A(G_1, i) \cong A(G_2, j)\), and let \(\pi\) be an isomorphism from \(A(G_1, i)\)
to \(A(G_2, j)\). Let \(u_b, v_b, u_s, v_s\) be the vertices added by \(A\) in the corresponding
graphs. Clearly, \(|V(G_1)| = |V(G_2)| = n\) for some \(n\). Notice that the
maximum out-degree of any vertex in either \(G_1\) or \(G_2\) is \(n\), since the vertex
can have at most one edge from it to any other vertex (including itself).
Furthermore, \(A(G, i)\) doesn’t add to the out-degree of any vertex in \(G\), and
\(u_b\) and \(v_b\) have out-degree \(n + 1\). Thus \(\pi(u_b) = v_b\) since those are the only
vertices with out-degree \(n + 1\). Also, \(u_s\) and \(v_s\) are the only vertices that don’t
have an incoming edge from \(u_b\) or \(v_b\), respectively, so \(\pi(u_s) = v_s\). Finally,
\(u_i\) and \(v_j\) are the only vertices that have an incoming edge from \(u_s\) and \(v_s\),

\(\Rightarrow\) Suppose \(A(G_1, i) \cong A(G_2, j)\), and let \(\pi\) be an isomorphism from \(A(G_1, i)\)
to \(A(G_2, j)\). Let \(u_b, v_b, u_s, v_s\) be the vertices added by \(A\) in the corresponding
graphs. Clearly, \(|V(G_1)| = |V(G_2)| = n\) for some \(n\). Notice that the
maximum out-degree of any vertex in either \(G_1\) or \(G_2\) is \(n\), since the vertex
can have at most one edge from it to any other vertex (including itself).
Furthermore, \(A(G, i)\) doesn’t add to the out-degree of any vertex in \(G\), and
\(u_b\) and \(v_b\) have out-degree \(n + 1\). Thus \(\pi(u_b) = v_b\) since those are the only
vertices with out-degree \(n + 1\). Also, \(u_s\) and \(v_s\) are the only vertices that don’t
have an incoming edge from \(u_b\) or \(v_b\), respectively, so \(\pi(u_s) = v_s\). Finally,
\(u_i\) and \(v_j\) are the only vertices that have an incoming edge from \(u_s\) and \(v_s\),
respectively, so \( \pi(u_i) = v_j \). Now we can build an isomorphism \( \pi' \) from \( G_1 \) to \( G_2 \) where \( \pi'(u_a) = \pi(u_a) \) for any \( u_a \in V(G_1) \) (so \( \pi' \) is just \( \pi \) with \( u_i \) and \( u_j \) removed from its domain). \( \pi' \) is obviously a valid isomorphism because \( A(G,i) \) doesn’t add or remove any edges between two vertices of \( G \). Thus we have an isomorphism \( \pi' \) from \( G_1 \) to \( G_2 \), and \( \pi'(u_i) = v_j \).

\( \Leftarrow \) Suppose we have an isomorphism \( \pi \) as above. Since \( A(G,i) \) doesn’t add or remove any edges between vertices of \( G \), we can clearly extend \( \pi \) to an isomorphism \( \pi' \) from \( A(G_1,i) \) to \( A(G_2,j) \) by setting \( \pi'(u_a) = \pi(u_a) \) for any \( u_a \in V(G_1) \), \( \pi'(u_s) = v_h \), and \( \pi'(u_s) = v_k \). Since \( \pi'(u_i) = v_j \), we can use the same reasoning as in the previous part to show that \( \pi' \) is indeed a valid isomorphism.

**Claim 2:** \( \exists \pi. \pi \) is an isomorphism from \( G_1 \) to \( G_2 \) \( \land \pi(u_i) = v_j \Rightarrow G_1 \sim G_2 - \{v_j\} \)

**Proof:** Suppose we have such an isomorphism \( \pi \). Define \( \pi' \) to be exactly \( \pi \) except with \( u_i \) removed from the domain. Since \( \pi(u_i) = v_j \), the domain of \( \pi' \) is \( V(G_1 - \{u_i\}) \) and the range is \( V(G_2 - \{v_j\}) \). Furthermore, \( u_i \) was removed from the neighbor set of a vertex \( u \in V(G_1 - \{u_i\}) \) if and only if \( v_j \) was removed from the neighbor set of \( \pi'(u) \). Thus \( \pi' \) is a valid isomorphism from \( G_1 - \{u_i\} \) to \( G_2 - \{v_j\} \), and so the two graphs are isomorphic.

Now we can describe a deterministic checker to check an oracle claiming \( G_1 \sim G_2 \). We will put together an isomorphism \( \pi \) between the two graphs as follows: at each step, iterate through all pairs of vertices \( (u_i,v_j) \in V(G_1) \times V(G_2) \). For each pair, ask our oracle whether \( A(G_1,i) \sim A(G_2,j) \). If the oracle says yes, then, assuming it’s telling the truth, we know by Claim 1 above that there must be some isomorphism from \( G_1 \) to \( G_2 \) that maps \( u_i \) to \( v_j \), so set \( \pi(u_i) = v_j \). Now, assuming our oracle isn’t lying, Claim 2 above tells us that \( G_1 - \{u_i\} \sim G_2 - \{v_j\} \), so we simply recurse on these graphs. Any isomorphism we come up with mapping \( G_1 - \{u_i\} \) to \( G_2 - \{v_j\} \) can be extended to an isomorphism from \( G_1 \) to \( G_2 \) by just mapping \( u_i \) to \( v_j \). If our oracle is correct, then we will find a valid isomorphism from \( G_1 \) to \( G_2 \) in polynomial time, so we double-check that the mapping we came up with is a correct isomorphism, and then accept. If our oracle ever lies, then one of two things can happen: if it says that \( A(G_1,i) \) and \( A(G_2,j) \) are isomorphic when they actually aren’t, then by the backward direction of Claim 1 there can be no isomorphism from \( G_1 \) to \( G_2 \) mapping \( u_i \) to \( v_j \), and so no mapping we end up with at the end of our process will be a valid isomorphism, so we will correctly reject the oracle. The other possibility is that at some step the oracle says that \( A(G_1,i) \) and \( A(G_2,j) \) are not isomorphic for any pair \( (i,j) \). But then the forward direction of Claim 1 above tells us that either
the oracle has lied at some point during this step, or it lied in the previous step – either way, we correctly reject it. Hence we have a valid deterministic checker for graph isomorphism.

4 Problem 8.11

\( MIP \subseteq NEXP \)

We will use the oracle formulation of \( MIP \) mentioned in class. Pick any \( L \in MIP \). Then there is a probabilistic poly-time TM \( M \) such that for any input \( x \):

\[
\begin{align*}
    x \in L & \implies \exists O : M^O(x) = 1 \text{ with prob 1} \\
    x \notin L & \implies \forall O : M^O(x) = 1 \text{ with prob at most } 1/4
\end{align*}
\]

The following nondeterministic algorithm will recognize \( L \): on input \( x \), nondeterministically choose an oracle \( O \). Then simulate \( M^O(x, r) \) on all possible poly-length random bit strings \( r \). If \( M^O(x, r) = 1 \) for all \( r \), then accept \( x \). Otherwise, reject \( x \). Clearly, the deterministic part of this algorithm can be done in singly exponential time. The only thing we need to show, then, is that the nondeterministically-chosen oracle \( O \) can be represented in at most a singly exponential number of bits (so that the certificate for this algorithm isn’t too long). Indeed, to represent an oracle, we simply need to record a single answer for every possible question it could receive. Since \( M \) runs in polynomial time, all oracle questions and answers must be polynomial length. Hence we can record an answer for all possible questions using singly-exponential space.

5 Problem 8.12

\( MIP_2 = MIP_{poly} \)

For a full proof of this, see Section 5 of the original paper by Ben-Or, Goldwasser, Kilian and Wigderson: “Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions” (1988). ([http://dl.acm.org/citation.cfm?doid=62212.62223](http://dl.acm.org/citation.cfm?doid=62212.62223))