CSE 531 MIDTERM EXAM

November 9, 2000

Solutions by Evan Schrier

1. **Answer the following true or false (no justification needed):**

   (a) **If $A \leq_m^P B$ and B is not in P, then A is not in P.** False.

   (b) **$A \leq_T^P \overline{A}$ for all languages A.** True.

   (c) **$A \leq_m^P \overline{A}$ for all languages A.** False.

   (d) **If a problem is NP-complete then it is known *not* to be solvable in polynomial time.** False.

   (e) **The complement of some Turing-recognizable language is not Turing-recognizable.** True.

   (f) **If $A \leq_m^P B$ and B is in NP, then A is in NP.** True.

   (g) **If a problem is in P then there are known practical algorithms to solve the problem.** False.

   (h) **If a problem is in NP then any deterministic algorithm to solve the problem runs poorly on every input.** False.

   (i) **If A is undecidable then $A \leq_T A_{TM}$.** True.

   (j) **The set of all primes written in binary is in NP.** True.

2. **Consider the problem 4-COLOR $= \{\langle G \rangle : G$ is 4-colorable$\}$. An undirected graph is 4-colorable if the vertices can be colored in 4 colors such that no two adjacent vertices in the graph have the same color. Use the fact that 3-COLOR is NP-complete to show that 4-COLOR is also NP-complete.**

   To prove that 4-COLOR in NP-Complete we must show that it is a member of NP and that some other NP-Complete problem, in this case 3-COLOR, is polynomial-time reducible to it.

   First, we can show that it is a member of NP by providing a polynomial time verifier for it. The verifier is just a solution, which is a list of colorings for each node in the graph. The verifier algorithm will compare the color of each node in the graph to the color assigned to each connected node and will reject the solution if the colors of two adjacent nodes match. In the worst case, each node will be compared to every other node in the graph for at most n^2 steps.

   Second, we show a reduction from 3-Color which is given as NP-Complete in the problem. Given a 3-color problem for a graph G with n nodes, we can map it to a 4-color problem by constructing a new graph G'. The graph G' is identical to G but has one additional node that is connected to every other node in the graph. This graph can be created with $n+1$ steps so the mapping is polynomial time. Since the new node is connected to every other node in the graph, it cannot share a color with any other node. So if G was three-colorable G' can be four-colored by assigning the original nodes the same colors as in the three-color solution of G and assigning the new node the fourth color. For any G' that is four-colorable the new node must have a color of its own since it is connected to every other node in the graph. Therefore G can be three colored by using the same color assignments for all the remaining nodes. Therefore, G' is four-colorable if and only if G is three-colorable.

3. **Consider the problem HALT-FINITE** $= \{\langle M \rangle : M$ **halts on finitely many inputs**$\}$. **Show that HALT-FINITE is not Turing-recognizable by showing that** $\overline{A_{TM}} \leq_m$ **HALT-FINITE. Recall that** $A_{TM} = \{\langle M, w \rangle : M$ **accepts** $w\}$.

   We show that A_{TM} is many-one reducible to HALT-INFINITE which is equivalent to showing that $\overline{A_{TM}}$ is many-one reducible to HALT-FINITE. We construct M' from $\langle M, w \rangle$ with the property that M accepts w if and only if M' halts on infinitely many inputs.

   M' is defined as follows:

   M' on input x:

   1 Run M on w.

   2 If M accepts w then halt

   3 If M rejects w then loop forever

   First, if M accepts w then M' halts on all inputs. Hence, $\langle M, w \rangle$ in A_{TM} implies $\langle M' \rangle$ halts on infinitely many inputs.

   Second, if M does not accept w then either M halts and rejects w or M does not halt. In the former case M' loops forever in step 3 and in the latter case M' never leaves step 1. In either case, M' halts on no inputs. Hence, $\langle M' \rangle$ not in A_{TM} implies $\langle M' \rangle$ halts on finitely many inputs.

4. **Assume that** G **has a clique of size** k**. Show how to find a clique of size** k **in** G **in polynomial time using the oracle CLIQUE. In other words, show that if CLIQUE can be solved in polynomial time then so can the problem of finding a clique of a given size.**

   We take a graph G with vertices v_1, v_2, ... v_n. We maintain a working set which starts with all vertices in G and we iteratively modify the set until there is only a k-vertex clique remaining in the set.

   The algorithm begins by adding all nodes to the working set and then testing the set with the oracle for a k-clique. If the oracle returns **false** then there is no solution and the algorithm reject. Otherwise the algorithm tries to remove nodes from the working set one at a time, and tests the resulting set with k-CLIQUE again. If k-CLIQUE returns **false** then the last node removed is added back to the set. Otherwise, it loops and removes another node. This continues until only k nodes remain in the working set and CLIQUE returns **true**. At this point the current working set is a solution.

Add all nodes to working set **WS**
Test **WS** on k-CLIQUE Oracle, reject if oracle returns **FALSE**
LOOP $i = 1$ to $n - k$
    IF $|\mathbf{WS}| = k$, terminate loop, current **WS** is solution
    Remove v_i from working set
    Test **WS** for k-CLIQUE
    IF k-CLIQUE returns **false**, return v_i to working set
REPEAT LOOP

The algorithm works because at the beginning of each pass through the loop there is a k-clique in the working set. This invariant is verified at the beginning and maintained through each pass. Consider a step of the algorithm where a vertex is removed from the working set. If the remaining graph has a k-clique then the vertex is permanently removed and the working set still has a k-clique. If the remaining graph does not have a k-clique then the vertex is returned to the working set, which by the invariant at the end of the previous step, has a k-clique. Since the working set will always contain a k-clique, it follows that when the working set is reduced to only k vertices at the end of any pass through the loop, the k vertices in the set must be a k-clique.