Kelsey Helms
CS 325
11/16/16
Homework 4

1. **Let X and Y be two decision problems. Suppose we know that X reduces to Y in polynomial time. Which of the following can we infer? Explain.**
    a. **If Y is NP-complete then so is X.**
    b. **If X is NP-complete then so is Y.**
    c. **If Y is NP-complete and X is in NP then X is NP-complete.**
    d. **If X is NP-complete and Y is in NP then Y is NP-complete.**
    e. **X and Y can't both be NP-complete.**
    f. **If X is in P, then Y is in P.**
    g. **If Y is in P, then X is in P.**

    (d) and (g) are the only statements we can infer. This is because "X reduces to Y" means that if we had a way to solve Y efficiently, without previous knowledge of what exactly the method is, we can use it to solve X efficiently. X is no harder than Y.

2. **Consider the problem COMPOSITE: given an integer y, does y have any factors other than one and itself? For this exercise, you may assume that COMPOSITE is in NP, and you will be comparing it to the well-known NP-complete problem SUBSET-SUM: given a set S of n integers and an integer target t, is there a subset of S whose sum is exactly t? Clearly explain whether or not each of the following statements follows from that fact that COMPOSITE is in NP and SUBSET-SUM is NP-complete:**
    a. **SUBSET-SUM ≤p COMPOSITE.**
    This doesn't follow from what is given. NP-complete doesn't always reduce to a problem in NP.

    b. **If there is an O(n3) algorithm for SUBSET-SUM, then there is a polynomial time algorithm for COMPOSITE.**
    This is true because, since SUBSET-SUM is NP-complete, all problems in NP reduce to it. The existence of a polynomial time algorithm for SUBSET-SUM implies that there is a polynomial time algorithm for all problems in NP. Since we know that COMPOSITE is in NP, then COMPOSITE has a polynomial time algorithm.

    c. **If there is a polynomial algorithm for COMPOSITE, then P = NP.**
    P=NP can't be concluded. COMPOSITE has a polynomial time algorithm. However, we only know that COMPOSITE is in NP, but not that it is NP-complete, so we can't conclude that P=NP.

    d. **If P ≠ NP, then no problem in NP can be solved in polynomial time.**

This doesn't follow from what is given. P ≠ NP means that P is a subset of NP and there is a polynomial time algorithm for each problem in P. The only conclusion that can be drawn if P ≠ NP is that no NP-complete problem can be solved in polynomial time.

3. **Two well-known NP-complete problems are 3-SAT and TSP, the traveling salesman problem. The 2-SAT problem is a SAT variant in which each clause contains at most two literals. 2-SAT is known to have a polynomial-time algorithm. Is each of the following statements true or false? Justify your answer.**
    a. **3-SAT ≤p TSP.**
    True. 3-SAT is NP-complete, so it is in NP. Since TSP is NP-complete, it follows that all problems in NP reduce to it.

    b. **If P ≠ NP, then 3-SAT ≤p 2-SAT.**
    False. It is given that there is a polynomial time algorithm for 2-SAT. If there was a reduction from 3-SAT to 2-SAT then it would follow that P=NP. Because P ≠ NP, the supposition is false.

    c. **If P ≠ NP, then no NP-complete problem can be solved in polynomial time.**
    True. Since all problems in NP can be reduced to every NP-complete problem, if any NP-complete problems can be solved in polynomial time then it would imply P=NP. The contrapositive says that if P ≠ NP then no NP-complete problem can be solved in polynomial time.

4. **LONG-PATH is the problem of, given (G, u, v, k) where G is a graph, u and v vertices and k an integer, determining if there is a simple path in G from u to v of length at least k. Show that LONG-PATH is NP-complete.**

    Long Path is in NP since the path is the certificate (we can easily check in polynomial time that it is a path, and that its length is k or more), and NP-complete since Hamiltonian Path (the variant where we specify a start and end node) is a special case of Long Path, namely where k equals the number of vertices of G minus 1.

5. **Graph-Coloring. Mapmakers try to use as few colors as possible when coloring countries on a map, as long as no two countries that share a border have the same color. We can model this problem with an undirected graph G = (V,E) in which each vertex represents a country and vertices whose respective countries share a border are adjacent. A k-coloring is a function c: V -> {1, 2, … , k} such that c(u) ≠ c(v)for every edge (u, v) ∈ E. In other words the number 1, 2, .., k represent the k colors and adjacent vertices must have different colors. The graph-coloring problem is to determine the minimum number of colors needed to color a given graph.**

a. **State the graph-coloring problem as a decision problem K-COLOR. Show that your decision problem is solvable in polynomial time if and only of the graph-coloring problem is solvable in polynomial time.**

   The decision problem is: Given an undirected graph G and an integer k, can we color G with k colors such that adjacent vertices have different colors?

   i. If we have solution to this problem in polynomial time, we can assign k values from |V| down to 2 and check if it is colorable, and stop when k reaches some value that it is not colorable. Then we choose the minimum number from all those numbers make the graph colorable, that is the number we want in Graph-coloring problem. It is easy to know the time is still polynomial.
   ii. If Graph-coloring problem is solvable in polynomial time, then we know the minimum number of colors needed, say l, then we can just compare the given number k in decision problem with l, if $k < l$, then we can answer NO, in decision problem; if $k \geq l$, then we can answer YES in decision problem. It must be polynomial time solvable.

b. **It has been proven that 3-COLOR is NP-complete by using a reduction from SAT. Use the fact that 3-COLOR is NP-complete to prove that 4-COLOR is NP-complete.**

   i. 4-COLOR is in NP: The coloring is the certificate. The following is a verifier V for 4-COLOR. V= On input (G, c)
      1. Check that c includes ≤ 4 colors.
      2. Color each node of G as specified by c.
      3. For each node, check that it has a unique color from each of its neighbors.
      4. If all checks pass, accept; otherwise, reject.
   ii. 4-COLOR is NP-hard: We give a polynomial-time reduction from 3-COLOR to 4-COLOR. The reduction maps a graph G into a new graph G' such that G $\in$ 3-COLOR if and only if G' $\in$ 4-COLOR. We do so by setting G' to G, and then adding a new node y and connecting y to each node in G'. If G is 3-colorable, then G' can be 4-colored exactly as G with y being the only node colored with the additional color. Similarly, if G' is 4-colorable, then we know that node y must be the only node of its color – this is because it is connected to every other node in G'. Thus, we know that G must be 3-colorable.

   Since 4-COLOR is in NP and NP-hard, we know it is NP-complete.