Practice Assignment 3

Philip Warton

December 3, 2020

Problem 1

(a) F
(b) U
(c) U
(d) U
(e) T
(f) T
(g) U
(h) U

Problem 2

(a)

Proof. We assume without proof that the 3-SAT is NP-hard. Then we show that any 3-SAT problem can be solved by some algorithm for 4-SAT problems. It follows that if a reduction from 3-SAT to 4-SAT exists, then 4-SAT must also be NP-hard. Then, we must show that 4-SAT is NP, so that being both NP-hard and NP, it is NP complete.

We write a 3-SAT problem as a 3-CNF problem consisting of clauses of three literals $(a \vee b \vee c)$. These clauses are appended by '$\wedge$' symbols. Taking any clause, we can map $(a \vee b \vee c) \rightarrow (a \vee a \vee b \vee c)$, and the two will be logically equivalent. Then, assuming an algorithm for the 4-SAT problem exists, we can solve this 4-CNF string using it. Thus a 4-SAT algorithm can determine if a string logically equivalent to the 3-SAT CNF string is satisfiable, thus solving the 3-SAT problem. Therefore 4-SAT must be NP-hard, since an algoirthm for 4-SAT solves an NP-hard problem.

To show that 4-SAT is an NP problem, we must show that a correct solution can be verified in polynomial time. Run the literal of true and false variables, and the solution can be verified in constant time. □

(b)

For 1-SAT write the CNF formula out with logical literals. Then for each logical variable, check if its negation can be found. If it can at any point, return false. Otherwise return true. This takes two for loops in $O(n^2)$ running time.

Problem 3

If G is not 3-colorable, we trivially return that no 3-coloring exists. If G is 3-colorable, then we must find some proper coloring. Choose some arbitrary starting vertex. Then we can color each of its neighbors with some coloring. If this coloring is invalid, choose another until we find some valid coloring. Then if we end up with a vertex that we cannot choose any valid color for, we go back to the next step and try each other possible combination. This algorithm is guaranteed to provide some valid 3-coloring of a 3-colorable graph eventually. The running time of this backtracking algorithm, will not be polynomial time, but it will guarantee a correct solution. Upon first glance, I would guess that this will be some kind of $O(n!)$ running time, since we can count possible incorrect permutations of the graph we may arrive at using combinatorics. However to provide some algorithm that will produce a valid 3-coloring in polynomial time I do not know. From what I have seen, neither greedy or divide and conquer approaches seem to guarantee a proper solution so this is the best that I am able to come up with at the time of writing.