Homework 7

1. Consider the following problem: given a set of clauses (each a disjunction of literals) and an integer k, find a satisfying assignment in which *at most* k variables are TRUE, if such an assignment exists. Prove that this problem is NP-complete.

Solution:

Call this problem $\mathsf{k-true-SAT}$.

First to show that this is in NP. Given the solution, a valid assignment of the variables, we can check that at most k of the variables are set to True in polynomial time ($O(n)$), and we can evaluate the formula in polynomial time as well $O(mn)$. Thus $\mathsf{k-true-SAT}$ is in NP.

Next, we'll show that $\mathsf{SAT} \rightarrow \mathsf{k-true-SAT}$. Given an instance of $\mathsf{SAT}$with n variables, we would run our algorithm for $\mathsf{k-true-SAT}$ setting $k = n$. We will use the exact output to $\mathsf{k-true-SAT}$ as the output for SAT.

Since we use the same formula, any solution to $\mathsf{k-true-SAT}$ is also a solution to the original SAT instance. Similarly, since any solution to SAT will always set at most $k = n$ variables to true, any solution to SAT will also give a solution to the corresponding instance of $\mathsf{k-true-SAT}$ with $k = n$.

Note: The reduction and proof of correctness are trivial, because $\mathsf{k-true-SAT}$ is a generalization of SAT. (A problem that includes SAT as a special case). Thus solving $\mathsf{k-true-SAT}$ in polynomial time would immediately imply that we can solve SATin polynomial time.

2. A *tadpole* is a graph on an even number of vertices, say $2n$, in which n of the vertices form a clique and the remaining n vertices are connected in a "tail" that consists of a path joined to one of the vertices of the clique. Given a graph G and integer k, the TadpoleProblem, asks for a subgraph which is a $2k$ sized tadpole (labeled so that we know which part is the clique and which is the tail). Prove that the TadpoleProblem is NP-complete.

Solution: This problem is in NP as given a tadpole in G, (labelled so that we know which part is the clique and which is the tail), it is easy to verify that it is in fact in G and that the clique part is a clique, and the tail part is a path.

We reduce from Clique. Given input G, k to Clique, we do the following: If $k = 1$, we can just return any vertex in G, and if $k = 2$, we can return two vertices on any edge of G. Otherwise, we attach a "tail" of k vertices separately to every vertex in G to create new graph G'. Otherwise, We then run our Tadpole algorithm on this new graph G' with parameter $2k$, and return "YES" iff find a tadpole of size $2k$.

Here we show that the algorithm works.

- Say G has a Clique of size k. Then those vertices are still a clique in G'. At a vertex in this clique, we have added a tail of k vertices in G' that are not a part of this clique. Together, this original clique and one of the added tails forms a tadpole of size $2k$ in G'. Thus if G has a solution to Clique , then G' will return that a solution has been found to Tadpole.

- Say G' has a tadpole of size $2k$. Then the "head" part of this tadpole is a clique of size k. Thus G' has a clique of size k. We need to argue that the vertices of this clique were originally a clique in G. Note that every vertex in the added tails has degree at most 2, and the two neighbors of any vertex in a tail do not have an adjoining edge by construction. Since $k > 2$, this means that no vertex in this clique could be part of the added tails. Thus this clique of size k is contained entirely in the original graph G. Thus any returned solution of Tadpole on G' will correspond to a valid solution of Clique on G.

Since G has a $k-clique$ if and only if G' has a $2k-tadpole$, this algorithm is correct. The work we do in constructing G' is polynomial in the inputs (we add $O(k|V|)$ vertices and edges), thus the input to Tadpole is polynomial in the original input. Thus Clique reduces to Tadpole, and Tadpole is NP- complete.

3. In the Exact4SAT problem, the input is a set of clauses, each of which is a disjunction of exactly 4 literals, and such that each variable occurs at most once in each clause. The goal is to find a satisfying assignment, if one exists. Instead of proving that Exact4SAT is NP-complete you will implement a piece of the proof. Specifically, you will write a method that solves a known NP-complete problem (3SAT) using a solution to Exact4SAT. You have been provided a solution to Exact4SAT in the file `ExactFourSAT.java`. You will write a solution to the 3SAT problem by adding code to the `isSatisfiable` method in the `ThreeSAT.java` file. Your solution should run in *polynomial time* if the Exact4SAT solution ran in polynomial time. Of course, the Exact4SAT solution given does not in fact run in polynomial time.

   Note that you are welcome to add additional private methods or data fields but you may not modify the method signature of the `isSatisfiable` method. You also may not modify the `ExactFourSAT.java` file. Your solution should not print anything.

Solution: This problem is in NP as this problem is a special case of SAT, which is in NP (A polytime verification algorithm for SATwould also work as verification here.)

We reduce from $3-\mathsf{SAT}$. Given an instance of $3-\mathsf{SAT}$ with n variables and m clauses, we create an instance of Exact 4-SAT with an equivalent truth value as follows:

Create three new dummy variables x, y, z that are unused anywhere in the input formula. If a clause has 3 variables, example $(a \vee b \vee c)$, then replace it with the two 4-element clauses $(a \vee b \vee c \vee x) \wedge (a \vee b \vee c \vee \bar{x})$. By the laws of boolean arithmetic, you can check that these two are logically equal no matter what the value of x is. Similarly, you can replace a 2 variable clause $(a \vee b)$ with an equivalent 4 4-variable clauses $(a \vee b \vee x \vee y) \wedge (a \vee b \vee \bar{x} \vee y) \wedge (a \vee b \vee x \vee \bar{y}) \wedge (a \vee b \vee \bar{x} \vee \bar{y})$. Similarly, replace a 1 variable clause with 8 4-variable clauses.

As explained above, our constructed exact-4-SAT forumala is logically equivalent to our input $3-\mathsf{SAT}$ formula, no matter what the assignment of the dummy variables. Thus an assignment of the original variables will satify the constructed formula iff it satisfies the original $3-\mathsf{SAT}$ formula.

Secondly, this construction can be done in polynomial time, because we do at most $32m$ work in creating the clauses, and end with a formula with at most $8m$ clauses, and $3m + n$ variables. Both of these are polynomial, thus the input formula to Exact4SAT will be polynomial in the original input. Because all steps in the reduction take polynomial time, this implies that a polytime algorithm for Exact 4SAT would result in a polytime algorithm for $3 - \mathsf{SAT}$. Thus Exact 4SAT is NP-complete.

4. **(PRACTICE)**Consider the Clique problem restricted to graphs in which every vertex has degree at most 3. Call this problem Clique-3.

   (a) Prove that Clique-3 is in NP.

   Solution: Given a solution to $Clique - 3$, we can check that it is actually a clique in the graph in polynomial time, and that it has the desired number of vertices in polynomial time.

   (b) What is wrong with the following proof of NP-completeness for Clique-3?

   We just showed that Clique-3 is in NP.

   We know that the Clique problem in general graphs is NP-complete, so it is enough to present a reduction from CLIQUE-3 to CLIQUE. In fact, we use the fact that CLIQUE is a generalization of CLIQUE-3. Given an input graph G with vertices of degree ≤ 3 and parameter k, run an algorithm for CLIQUE unchanged on the same graph and same input parameter. If a clique of size $\geq k$ is present, we return it, otherwise return FALSE.

   In CLIQUE-3, we are looking for a clique of size $\geq k$ in G. This is exactly what CLIIQUE will return, if one exists. This proves the correctness of the reduction and, therefore the NP-completeness of Clique-3.

   Solution: The reduction from Clique-3 to Clique is valid, but this doesn't imply that Clique-3 is NP-Hard. You would need to reduce Clique to Clique-3 (the other direction).

   (c) It is true that the Vertex Cover remains NP-complete even when restricted to graphs in which every vertex has degree at most 3. Call this problem VC-3. What is wrong with the following proof of NP-completeness for Clique-3?

   We present a reduction from VC-3 to Clique-3. Clique-3 is in NP as proved in part a.

   Given an input graph $G = (V, E)$ with max-degree 3 to VC-3, and a parameter b, we create an instance of Clique-3 by leaving the graph unchanged and switching the parameter to $k = |V| - b$.

   Now, a subset $C \subseteq V$ is a vertex cover in G if and only if the complementary set $V - C$ is a clique in G. Therefore G has a vertex cover of size $\leq b$ if and only if it has a clique of size $\geq |V| - b$. This proves the correctness of the reduction and, consequently, the NP-completeness of Clique-3.

   Solution: In this proof, the statement: a subset $C \subseteq V$ is a vertex cover in G if and only if the complementary set $V - C$ is a clique in G: is false. The correct statement

from class is that a subset $C \subseteq V$ is a vertex cover in G if and only if the complementary set $V - C$ is *an independent set* in G.

(d) Describe a poly-time algorithm for Clique-3.

Solution: Because the node degrees are at most 3, the largest possible size of a clique is 4. Thus if $k > 4$, we can automatically return $FALSE$.

After this, we can brute force check all $\binom{n}{k}$ k-element subsets of V, and check if each is a clique in polynomial time. There are at most 6 edges to check in each of these checks, and at most $O(n^4)$ subsets to check, thus this is a poly-time algorithm.

5. **(PRACTICE)** Integer Inequality Satisfaction: In this problem, you are given a set of n variables $x_1, x_2, \ldots x_n$, and a set of m inequalities, where the jth inequality looks like $\sum_i a_{i,j} x_i \geq c_j$. The constants $a_{i,j}$ are also integers. Our goal is to find *an integer* set of x_i such that every equation is satisfied. Prove that this problem is NP-complete via a direct reduction from either SATor $3 - \mathsf{SAT}$. If your clause has n variables and m clauses, how many variables and inequalities does your reduction require?

Hint: Add inequalities to force your variables to behave like "True" or "False". What do you need after that?

Solution:

- First we show that this is in NP. Given a valid assignment of all the variables, we can check that every equation is satisfied using $O(n)$ arithmetic operations, for a total runtime of $O(mn)$.
- We reduce from SAT. Given an input clause with variables $x_1, \ldots x_n$, we first make the equations: $x_i \geq 0$ and $-x_1 \geq -1$, so that each $0 \leq x_i \leq 1$. Since our final solution must be an integer, this means each x_i can only be 0 or 1, which will represent false and true.

  Then for each clause of the form $(a \vee b \vee c \ldots)$, we add the equation $a + b + c \ldots \geq 1$. Since each literal is forced to be 0 or 1, then the requirement that the sum of all literals is at least 1 is EXACTLY the requirement that at least one of them is 1 - in otherwords that at least one of the variables in the clause is $TRUE$.

  For negative literals, we use the fact that $\bar{x_i} = (1 - x_i)$, and substitute this into the equations wherever we need to represent a negative literal. We can then subtract all the 1's thus generated from both sides of the equation to put it into the desired form.
- Half of the proof of correctness was explained along with the algorithm above: a valid solution to these equations MUST represent a satisfying assignment. Also, any satisfying assignment correspods to assignments of the variables to 0 if false, 1 if true, that would satisfy the above constructed equations. Thus this problem has a solution on this construction if and only if the original input to SAT had a solution.

  Constructing the integer satisfiability instance takes poly time, and will result in $2n$ equations of constant size, and m equations of $O(n)$ size. Thus the input to this

algorithm will be $O(mn)$, which is polynomial in the input. Thus a poly-time solution for this problem implies a poly-time algorithm for SAT, which means that this problem is NP-complete.

6. **(PRACTICE)** Consider the following *longest* $(s-t)$-path problem. Given a graph G an integer k and two vertices s and t the goal is to find a simple path using k or more edges from s to t, if one exists. Prove that, unlike the problem of finding the shortest path in a graph, the *longest* $(s-t)$-path problem is NP-complete by directly reducing from Hamiltonian (Rudrata) Cycle.

Solution: First we show that this is in NP. Given a path, we can verify that it is a path and visits k distinct vertices in time $O(V)$ by iterating through the edges of the path, and keeping track of which vertices have been visited in an array.

We reduce from Hamiltonian Cycle. There are two distinct ways to approach this problem. I will go over both. The first is easier for many people to see.

Approach 1:

Given an instance G of Hamiltonian Cycle, pick a single vertex v in G. For each neighboring vertex w, we run Longest s − t Path with $s = w$, $t = v$, and $k = |V| - 1$. We return "YES" to Hamiltonian Cycle if any of these runs of Longest s − t Path returns "YES".

- If we find a $w - v$ path of size $|V| - 1$, then that path together with the edge (v, w) is a cycle of size $|V|$. Thus it must visit every vertex. Thus we found a Hamiltonian Cycle in G.
- If G has a Hamiltonian Cycle, then this cycle must visit vertex v. The edge in this cycle leading out from v will be *some* neighbor w of v. The rest of the cycle is a $w - v$ path of size $|V| - 1$. Thus in the algorithm above, we will consider the neighbor w eventually while iterating among the neighbors of v, and find a $w - v$ path of size $|V|$.

The removal of these edges takes constant time, and we run Longest s − t Path at most $|V| - 1$ times. Thus if we have a polynomial time algorithm for longest s-t path, the above algorithm for Hamiltonian Cycle will also be polynomial. Thus Hamiltonian Cycle reduces to Longest s − t Path, so this problem is NP-complete.

Approach 2: Given an input graph G, take a vertex v and split it into v and v'. Make copies of all the edges incident to v $e = (v, w)$ and attach them to v' - $e = (w, v')$. Can you see why G has a hamiltonian cycle iff G' has a path from v to v' of size $|V|$? It will be good practice for you to finish this proof yourself.