31 (100 PTS.) My friend, parting time is pending.

The following question is long, but not very hard, and is intended to make sure you understand the following problems, and the basic concepts needed for proving NP-Completeness.

All graphs in the following have n vertices and m edges.

For each of the following problems, you are given an instance of the problem of size n. Imagine that the answer to this given instance is "yes", and that you need to convince somebody that indeed the answer to the given instance is `yes`. To this end, describe:

(I) An algorithm for solving the given instance (not necessarily efficient). What is the running time of your algorithm?
(II) The format of the proof that the instance is correct.
(III) A bound on the length of the proof (its have to be of polynomial length in the input size).
(IV) An efficient algorithm (as fast as possible [it has to be polynomial time]) for verifying, given the instance and the proof, that indeed the given instance is indeed `yes`. What is the running time of your algorithm?

(EXAMPLE)

Shortest Path

Instance: A weighted undirected graph G, vertices s and t and a threshold w.
Question: Is there a path between s and t in G of length at most w?

Solution:

(I) **Algorithm**: We seen in class the Dijkstra algorithm for solving the shortest path problem in $O(n \log n + m) = O(n^2)$ time. Given the shortest path, we can just compare its price to w, and return yes/no accordingly.
(II) **Certificate**: A "proof" in this case would be a path π in G (i.e., a sequence of at most n vertices) connecting s to t, such that its total weight is at most w.
(III) **Certificate length**: The proof here is a list of $O(n)$ vertices, and can be encoded as a list of $O(n)$ integers. As such, its length is $O(n)$.
(IV) **Verification algorithm**: The verification algorithm for the given solution/proof, would verify that all the edges in the path are indeed in the graph, the path starts at s and ends at t, and that the total weight of the edges of the path is at most w. The proof has length $O(n)$ in this case, and the verification algorithm runs in $O(n^2)$ time, if we assume the graph is given to us using adjacency lists representation.

31.A. (20 PTS.)

Socially Distanced Set

Instance: A graph G, integer k
Question: Is there a distanced set in G of size k? A set $X \subseteq \mathsf{V}(\mathsf{G})$ is a distanced set if no two vertices of X are connected by an edge, or a path of length at most 4.

Solution:

(I) **Solver**: Compute for every pair of vertices in the graph if they are in distance $1, 2, 3$ or r (in edge distance). This can be done by doing **BFS** for each vertex, thus taking $O(nm)$ time. Build a graph $\mathsf{H} = (\mathsf{V}(\mathsf{G}), \mathsf{E}')$ where a pair of vertices is connected $\iff$ they are in distance ≤ 4 from each other in G. Clearly, the task in hand is to find an independent set in H of size k.
So, enumerate all subsets of vertices of $\mathsf{V} = \mathsf{V}(\mathsf{G})$ of size k. This takes $O\left(\binom{n}{k}\right)$ time with careful implementation (verify you know how to do this!). Now, for each such set generated, check if it is independent in G – that takes $O(k^2)$ time (after $O(n^2)$ preprocessing to compute the adjacency graph). Let k' be the largest independent set in H found by this process. If $k' \geq k$ return YES, otherwise return NO. The running time of the algorithm is $O\left(\binom{n}{k}k^2 + nm\right) = O(n^k + n^3)$.

(II) **Proof format:** A list of k integer numbers between 1 and n (i.e., assume $\mathsf{V} = \{1, \ldots, n\}$).

(III) **Proof length:** $O(k) = O(n)$ words. In bits, the length of the encoding is $O(k \log n)$.

(IV) **Certifier:** Given a list L of k vertices, do **BFS** from each vertex in L, and check that the distance of any pair of vertices in L is indeed > 4. This can be done in $O(nm + k^2) = O(nm)$ time.

31.B. (20 PTS.)

Edge Independent

Instance: A graph G, a set a parameter k.
Question: Is there a subset of k edges in the graph, such that no pair of edges is adjacent?

Solution:

(I) **Solver**: This problem is known as maximum matching and can be solved efficiently, but the algorithm is quite complicated. For our purposes here, we can just enumerate all subsets of k edges. This takes $O(\binom{m}{k})$ time. For each such subset check if it is independent, but checking if they share an endpoint. This can be done in $O(k)$ time. As such, if there is an independent set of edges of size k, it can be computed in $O(m^k k)$ time.

(II) **Proof format:** A list of k edges, each edge is a pair of two vertices. So if the vertices are represented as numbers $1, \ldots, n$, then this is just a list of $2k$ numbers.

(III) **Proof length:** $O(k)$.

(IV) **Certifier:** Just check that every edge specified is indeed an edge in the graph, and that the edges do not share endpoints. With careful implementation this can

be done in $O(m)$ time (assuming $m \geq n$). With less careful implementation the running time is $O(mk)$.

31.C. (20 PTS.)

Sum to target

Instance: S: Set of positive integers. t: An integer number (target).
Question: Is there a subset $X \subseteq S$ such that $\sum_{x \in X} x - \sum_{y \in S \setminus X} y = t$?

Solution:

Assume $S = \{s_1, \ldots, s_n\}$.

(I) **Solver**: Enumerate all possible subsets of S, and check for each such subset if it sum minus the sum of all the numbers in the complement sum up to t. This takes $O(2^n n)$ time.

(II) **Proof format:** The certificate is a binary string $b_1, \ldots, b_n$, where b_i is one if and only if the ith number is supposed to be in the solution subset X.

(III) **Proof length:** So the certificate length is $O(n)$.

(IV) **Certifier:** The certifier adds up the numbers with $b_i = 1$, and subtracts all the numbers with $b_i = 0$. Then it check if the resulting quantity is t. This takes $O(n)$ time.

31.D. (20 PTS.)

4DM

Instance: X, Y, Z, W sets of n elements, and T a set of quadruples, such that $T \subseteq X \times Y \times Z \times W$.
Question: Is there a subset $S \subseteq T$ of n disjoint quadruples, such that every element of $X \cup Y \cup Z \cup W$ is covered exactly once by one of the quadruples of S?

Solution:

Assume $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$, $Z = \{z_1, \ldots, z_n\}$, and $W = \{w_1, \ldots, w_n\}$. Furthermore, the quadruples are $T = \{t_1, \ldots, t_n\}$, where t_i is of the form $(i_1, i_2, i_3, i_4) \in \{1, \ldots, n\}^4$.

(I) **Solver**: Let $m = |T|$. Enumerate all possible subsets of T of size n, there are $\binom{m}{n} = O(m^n)$ such subsets. For each subset verify that all the quadruples in it are disjoint, and if so they form a valid solution. If so, return this as a valid solution. Otherwise, the algorithm returns false.
the running time of the algorithm is $O(m^n n)$ time.

(II) **Proof format:** The certificate is as such a list of n numbers $i_1, \ldots i_n$ in the range $\{1, \ldots, m\}$.

(III) **Certificate length:** The length of certificate is $O(n)$.

(IV) **Certifier:** The certifier checks that all the indices are distinct (by sorting the numbers in $O(n)$ time [using radix sort]). Next, it checks that the first coordinate of all the quadruples $t_{i_1}, \ldots, t_{i_n}$ are distinct (by again, say, using radix sort). It repeat this for the other three coordinates. If everything goes through then this is a valid solution.

As such, the verification takes $O(n)$ time.

31.E. (20 PTS.)

SET DISJOINT COVER

Instance: $(\mathsf{U}, \mathcal{F}, k)$:
U: A set of n elements
$\mathcal{F}$: A family of m subsets of U, s.t. $\bigcup_{X \in \mathcal{F}} X = \mathsf{U}$.
k: A positive integer.
Question: Are there k pairwise-disjoint sets $S_1, \ldots, S_k \in \mathcal{F}$ that cover U?

Formally, the sets $S_1, \ldots, S_k$ ***cover*** U if $\bigcup_i S_i = \mathsf{U}$. They are ***pairwise-disjoint*** if for any $i \neq j$, we have that $S_i \cap S_j = \emptyset$.

Solution:

Assume $\mathsf{U} = \{1, \ldots, n\}$, and $\mathcal{F} = \{F_1, \ldots, F_m\}$, where $F_i \subseteq \mathsf{U}$ is defined by a binary vector of length n, where the jth bit decides if $j \in F_i$.

(I) **Solver**: Enumerate all possible subsets of $\mathcal{F}$ of size k. There are $O(m^k)$ such sets. Now, for each such set of k sets of $\mathcal{F}$, verify that they cover disjointly the ground set. This takes $O(kn)$ time, as described below. As such, overall, the running time is $O(m^k kn)$.

(II) **Proof format:** The certificate is a list of k numbers $n_1, \ldots, n_k \in \{1, \ldots, m\}$.

(III) **Proof length:** $O(k)$.

(IV) **Certifier:** The certifier computes the bitwise-or of the vectors $F_{n_1}, \ldots, F_{n_k}$ and verifies that the resulting vector has 1 in all coordinates. It also verifies that every bit is set exactly once. This takes $O(kn)$ time.

32 (100 PTS.) Clique of SATs

Given an undirected graph $G = (V, E)$, a partition of V into $V_1, V_2, \ldots, V_k$ is a ***clique cover*** of size k if each V_i is a clique in G. CLIQUE-COVER is the following decision problem: given G and integer k, does G have a clique cover of size at most k?

32.A. (80 PTS.) Describe a polynomial-time reduction from CLIQUE-COVER to SAT.

Solution:

The reduction. The input to the CLIQUE-COVER problem is the graph $G(V, E)$ and integer k, while that to the SAT problem is a boolean formula ϕ. We reduce CLIQUE-COVER to SAT as follows. For every vertex u in V, we define k variables $x(u, i)$, $\forall i \in [\![k]\!] = \{1, \ldots, k\}$. A variable $x(u, i) = 1$ in a satisfying assignment of SAT will imply that the corresponding node is in the ith clique in the CLIQUE-COVER instance.

A solution to the CLIQUE-COVER problem must satisfy some additional constraints. To have the corresponding SAT solution reflect these criteria, we define different types of clauses, one for each criterion, for the SAT instance as follows.

(A) **A vertex belong to a single clique.** Non-adjacent vertices cannot belong in the same clique. As such, for every non-edge $uv \notin E$, define k clauses

$$\left(\overline{x(v, i)} \vee \overline{x(u, i)}\right), \ \forall i \in [\![k]\!] .$$

That is, variables corresponding to two non-adjacent vertices and the same partition cannot both be 1.

Overall, there are at most $k(\binom{n}{2} - m) = O(kn^2)$ such clauses, and they can be computed in this time.

(B) **Every vertex belongs to a clique.**

Every vertex should belong to at least one clique. For every vertex $v \in V$, we define the clause

$$\vee_{i \in [k]} x(u, i).$$

Note that, if all variables $x(u, 1), \ldots, x(u, k)$ are all 0, then the clause (OR of all variables) will be false.

There n such clauses overall, and they can be computed in $O(kn)$ time.

(C) **Every vertex belong to at most one clique.** Every vertex should belong to at most one clique. For a vertex $u \in V$, we define the clauses:

$$\bigwedge_{i,j \in [\![k]\!], i \neq j} \left(\overline{x(u, i)} \vee \overline{x(u, j)}\right).$$

That is, exactly one of the $x(u, i)$s can be 1 in every satisfying assignment. while if two or more of these variables are 1, the clause corresponding to the OR of the complement of the pair will be false. As a satisfying assignment must satisfy every clause, every solution will enforce exactly one variable (among these k variables) is 1.

The formula for a node is made out of k^2 clauses, so overall this would result in $O(nk^2)$ clauses. They can be computed in $O(nk^2)$ time.

The resulting boolean formula ϕ is an conjunction of all the above clauses.

If the graph has n vertices, ϕ is a formula of exactly nk variables. Also ϕ has at most kn^2 clauses, of size 2 each, of the first type, with k clauses for every non-edge, and kn clauses, of size k each, of the second type. Thus, the total number of clauses is at most $O(kn^2)$, which is polynomial in the size of the CLIQUE-COVER instance. Clearly, this formula can be computed in $O(kn^2)$ time.

Correctness of reduction. It is easy to verify that given an assignment to ϕ that satisfies it then the original graph can be covered by k cliques. As for the other direction, given a clique cover of G of size k, it is straightforward to turn it into a satisfying assignment for ϕ. We conclude that ϕ can be satisfied $\iff$ G has a clique cover of size k.

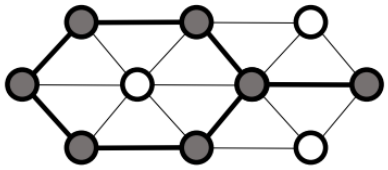
32.B. (20 PTS.) Does this prove that CLIQUE-COVER is NP-Complete? You just need to provide a yes/no answer with clear/concise explanation.

Solution:

By reducing to SAT, a known NP-Complete problem, we have proved CLIQUE-COVER is in NP. However, this reduction does not prove CLIQUE-COVER is NP-Complete, as this requires proving NP-Hardness too. For doing the same, a reduction from a known NP-Complete problem to CLIQUE-COVER must be shown.

33 (100 PTS.) No Triangles club.

A subset S of vertices in an undirected graph G is ***triangle-free*** if, for every triple of vertices $u, v, w \in S$, at least one of the three edges uv, uw, vw is *absent* from G. Prove that finding the size of the largest triangle-free subset of vertices in a given undirected graph is NP-hard.



Above is a triangle-free subset of 7 vertices.
This is ***not*** the largest triangle-free subset in this graph.

Solution:

We show a reduction from Independent Set to this problem.

Suppose we want to find the size of the largest independent set in a given undirected graph $G = (V, E)$ with n vertices. We construct a graph $G' = (V', E')$ by adding $2n$ new vertices to G. Each new vertex is connected to every vertex in the original graph G, but none of the new vertices are connected. Concretely, we have

$$V' = V \cup W \quad \text{and} \quad E' = E \cup \{vw \mid v \in V \text{ and } w \in W\},$$

where W is the set of $2n$ new vertices.

Clearly, this reduction runs in polynomial time. We now show that the largest subset of vertices in G' corresponds (after adjustment) to the largest independent set in the original graph.

Lemma 11.1. *Let S be the largest independent set in G, and let S' be the largest triangle free set in G'. Then, we have that $|S| = |S'| - 2n$.*

Proof: $\Longrightarrow$ Let $T = S \cup W$. Any triple of vertices in T must have either two vertices in S or two vertices in W, and both S and W are independent sets in G'. Thus, T is a triangle-free set in G', and $|T| = |S| + 2n \leq |S'|$.

$\Longleftarrow$ Conversely, suppose S' is the largest triangle-free subset in G'. If S' contained two vertices that are neighbors in G, it cannot contain any vertex in W, so $\left|S'\right| \leq n+1$; but this is impossible, because W is a triangle-free set of size $2n$, and thus S' is not maximal triangle free set. Thus, $W \subseteq S'$, and $X = S' \cap V$ is an independent set in G, of size $\left|S'\right| - 2n \leq |S|$.

We conclude that $|S| = |S'| - 2n$. ■

Thus, solving the problem of largest triangle free graph in G' is polynomially equivalent to solving independent set in G. Thus, if we can solve the triangle free problem in polynomial time, then we could solve Independent Set in polynomial time, which in turn would imply that NP is contained in P. Namely, triangle free is NP-HARD.