Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

If there exists a polynomial reduction from problem A to problem B, problem B is at least as hard as problem A. From this, we can define complexity class which sort of gauge 'hardness'.
Complexity Definitions

- NP: a decision problem in which a potential solution can be verified in polynomial time.
- P: a decision problem which can be solved in polynomial time.
- NP-Complete: a decision problem in NP which all problems in NP can reduce to.
- NP-Hard: any problem which is at least as hard as an NP-Complete problem.

Prove a problem is NP-Complete
To prove a problem is NP-Complete, you must prove the problem is in NP and it is in NP-Hard. To do this, you must show there exists a polynomial verifier, and reduce an NP-Complete problem to the problem.

1 NP Basics

Assume A reduces to B in polynomial time. In each part you will be given a fact about one of the problems. What information can you derive of the other problem given each fact?

1. A is in **P**.
2. B is in **P**.
3. A is **NP**-hard.
4. B is **NP**-hard.

Solution: If A reduces to B, we know B can be used to solve A, which means B is at least as hard as A. As a result, if B is in **P**, we can say that A is in **P**, and if A is **NP**-hard, we can say that B is **NP**-hard. If A is in **P**, or if B is **NP**-hard, we cannot say anything about the complexity of B or A respectively.

2 NP or not NP, that is the question

For the following questions, circle the (unique) condition that would make the statement true.

(a) If B is **NP**-complete, then for any problem $A \in \mathbf{NP}$, there exists a polynomial-time reduction from A to B.

Always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ Always False

Solution: Always True: this is the definition of **NP**-hard, and all **NP**-complete problems are **NP**-hard

(b) If B is in **NP**, then for any problem $A \in \mathbf{P}$, there exists a polynomial-time reduction from A to B.

Always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ Always False

Solution: Always true: since we have polynomial time for our reduction, we have enough time to simply solve any instance of A during the reduction.
Note that in this class, we ignore decision problems which always returns YES, and the decision problems which always returns NO.

(c) 2 SAT is **NP**-complete.

Always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ Always False

Solution: True iff $\mathbf{P} = \mathbf{NP}$:
By definition, in order to be NP-Complete a problem must be in NP, and there must exist a polynomial reduction from every problem in NP.
If $\mathbf{P} \neq \mathbf{NP}$, then there does not exist a polynomial time reduction from NP-Complete problems like 3-SAT to 2-SAT.
If $\mathbf{P} = \mathbf{NP}$, then a polynomial reduction is as follows:
since $\mathbf{P} = \mathbf{NP}$ there must exist a polynomial times algorithm to solve $3-SAT$. Thus, when we are preprocessing 3-SAT we can solve for whether there exists a solution in the instance or not. If the instance has a solution, then we will map it to an instance of $2-SAT$ that has a solution, and if it doesn't have a solution, we will map it to an instance that doesn't have a solution. Thus all problems in NP will have a polynomial time reduction to 2-SAT as all problems in NP are reducible to 3-SAT.

(d) Minimum Spanning Tree is in **NP**.

Always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ Always False

Solution: Always True. MST is solvable in polynomial time, which means it is verifiable in polynomial time.
Note that explicitly, the decision problem would be "does there exist a spanning tree whose cost is less than a budget b?".

3 Graph Coloring Problem

In the k-coloring problem, we are given an undirected graph $G = (V, E)$ and are asked to assign every vertex a color from the set $1, \cdots, k$, such that no two adjacent vertices have the same color.
As you will prove in the homework 3-coloring is NP-Complete.

Prove that 10-coloring is also NP-Complete.

Solution:

Proof that 10-coloring is in NP

We can verify an assignment of colors to vertices satisfy the constraints of 10-coloring, by making sure every vertex is assigned a color in the set $1, \cdots, k$, and that for every edge the two vertices are different colors. This takes O(V + E), and so 10-coloring is verifiable in polynomial time and is in NP.

Proof that 10-coloring is in NP-Hard

To do this we need to reduce an NP-Complete problem, 3-coloring to 10-coloring. The transformation is introduce 7 vertices that are completely connected to each other and add an edge from each of these 7 vertices to every vertex in V to produce a new graph G'. This new graph is what we will run 10-coloring on.

To prove this transformation is valid, we need to prove both directions which is shown as follows.

1. *If 3-coloring has a solution, then our transformation of the problem to 10-coloring has a solution.*
In G' you assign the colors of the vertices that come from G to be that of the colors of the solution of the 3-coloring. Then, you assign every one of the 7 new vertices you introduced to a different color, and you get a 10-coloring solution.

2. *If 10-coloring that is in the format of our transformation has a solution, then the 3-coloring that corresponds to the 10-coloring has a solution.*
The 10-coloring solution of G' must assign every one of the 7 vertices a different color because there is an edge between every one of the 7 vertices. Then, the vertices of the original problem must be constrained to only use 3 colors as there is an edge from every one of the 7 vertices (with different colors) to the original vertices.

Because 10-coloring is in NP and it is NP-hard, 10-coloring is NP-complete.

4 2-SAT and Variants

Max-2-SAT is defined as follows. Let $C_1, \ldots, C_m$ be a collection of 2-clauses and b a non-negative integer. We want to determine if there is some assignment which satisfies at least b clauses.

Max-Cut is defined as follows. Let G be an undirected unweighted graph, and k a non-negative integer. We want to determine if there is some cut with at least k edges crossing it. Max-Cut is known to be NP-complete.

Show that Max-2-SAT is NP-complete by reducing from Max-Cut. Prove the correctness of your reduction.

Solution:

Proof that Max-2-SAT is in NP.

Given an assignment of variables, we can iterate through the clauses and count how many are satisfied in polynomial time, and check that this number is less than or equal to k.

Proof that Max-2-SAT is in NP-Hard.

As suggested, we reduce unweighted `Max-Cut` to `Max-2-SAT`, where the graph $G = (V, E)$ and an integer k are part of the instance of Max-Cut we are trying to reduce. When approaching this reduction, we want to determine a way to represent all the restrictions of Max-Cut in a manner that Max-2-SAT will understand, which means figuring out how vertices and edges become variables or clauses in our Max-2-SAT instance.

We will represent every vertex as a variable which is true when it is on one side of the cut, and false when it on the other side. Now, we need to figure out how to represent the edges. We want to be able to distinguish between edges crossing the cut vs edges not crossing the cut. If we draw a truth table we will notice that for an edge (u, v) we want a pairing (T, F) and (F, T) to be able to be distinguished from (T, T) and (F, F). This is the boolean function XOR, which can be represented as 2 OR clauses: $(x_u \vee x_v)$ and $(\overline{x_u} \vee \overline{x_v})$ where x_u represents the vertex u and x_v represents the vertex v.

Notice how if both x_u and x_v are the same, then only one clause is satisfied, but if there are different both clauses are satisfied. So if z edges cross the cut, then there will be $2z$ clauses satisfied from the crossing, and $|E| - z$ clauses satisfied from edges not crossing, giving us a total of $|E| + z$ satisfying clauses.
Since we want there to be at least k edges crossing the cut, we will look for at least $|E| + k$ satisfying clauses.

To summarize our Max-2-SAT instance from Max-Cut is b = $|E| + k$, every vertex v is a variable x_v, and every edge (u, v) is translated into 2 clauses $(x_u \vee x_v)$ and $(\overline{x_u} \vee \overline{x_v})$.

Now we proceed to our proof of correctness.

1. *If an instance of Max-Cut has a solution, then the corresponding instance of Max-2-SAT has a solution.*
If Max-Cut has a solution, then there must be at least k edges crossing the cut. This means that the corresponding instance of Max-2-SAT has $2k$ clauses that will evaluate to true that were made from these edges. All edges that don't cross the cut contribute one clause that evaluates to true, meaning that there are $|E| - k + 2k = |E| + k$ clauses that are true which is equal to b thus proving that the corresponding instance of Max-2-SAT has a solution.

2. *If an instance of Max-2-SAT in the transformed format has a solution, then the corresponding*

instance of Max-Cut has a solution.
If an instance of Max-2-SAT in the transformed format has a solution, then there must be at least $|E| + k$ clauses that are satisfied. This corresponds to k edges of the instance of Max-Cut being satisfied, from the logic in paragraph 3 of the "Proof that Max-2-SAT is in NP-Hard" section, as in order for $|E| + k$ clauses to be satisfied, $|E| - k$ clauses must correspond to edges not crossing the cut, and $2 * k$ clauses must correspond to edges crossing the cut meaning k edges cross the cut.

Since we proved Max-2-SAT is in NP and is NP-Hard, it must be NP-Complete.