Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as 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 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

(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

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

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

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

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

2 California Cycle

Prove that the following problem is NP-hard

Input: A directed graph $G = (V, E)$ with each vertex colored blue or gold, i.e., $V = V_{\text{blue}} \cup V_{\text{gold}}$

Goal: Find a *Californian cycle* which is a directed cycle through all vertices in G that alternates between blue and gold vertices (Hint : Directed Rudrata Cycle)

3 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? Each part should be considered independent; i.e., you should not use the fact given in part (a) as part of your analysis of part (b).

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

4 Local Search for Max Cut

Sometimes, local search algorithms can give good approximations to NP-hard problems. In the Max-Cut problem, we have an unweighted graph $G(V, E)$ and we want to find a cut (S, T) with as many edges "crossing" the cut (i.e. with one endpoint in each of S, T) as possible. One local search algorithm is as follows: Start with any cut, and while there is some vertex $v \in S$ such that more edges cross $(S - v, T + v)$ than (S, T) (or some $v \in T$ such that more edges cross $(S + v, T - v)$ than (S, T)), move v to the other side of the cut. Note that when we move v from S to T, v must have more neighbors in S than T.

(a) Give an upper bound on the number of iterations this algorithm can run for (i.e. the total number of times we move a vertex).

(b) Show that when this algorithm terminates, it finds a cut where at least half the edges in the graph cross the cut.

5 Cycle Cover

In the cycle cover problem, we have a directed graph G, and our goal is to find a set of directed cycles $C_1, C_2, \ldots C_k$ in G such that every vertex appears in exactly one cycle (a cycle cannot revisit vertices, e.g. $a \to b \to a \to c \to a$ is not a valid cycle, but $a \to b \to c \to a$ is), or declare none exists.

In the bipartite perfect matching problem, we have a undirected bipartite graph (a graph where the vertices can be split into L, R, and there are no edges between two vertices in L or two vertices in R), and our goal is to find a set of edges in this graph such that every vertex is adjacent to exactly one edge in the set, or declare none exists.

Give a reduction from cycle cover to bipartite perfect matching. (Hint: In a cycle cover, every vertex has one incoming and one outgoing edge.)