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.

1 A Reduction Warm-up

In the Undirected Rudrata path problem (aka the Hamiltonian Path Problem), we are given a graph G with undirected edges as input and want to determine if there exists a path in G that uses every vertex exactly once.

In the Longest Path in a DAG, we are given a DAG, and a variable k as input and want to determine if there exists a path in the DAG that is of length k or more.

Is the following reduction correct? If so, provide a concise proof. If not, justify your answer and provide a counter example.

Undirected Rudrata Path can be reduced to Longest Path in a DAG. Given the undirected graph G, we will use DFS to find a traversal of G and assign directions to all the edges in G based on this traversal. In other words, the edges will point in the same direction they were traversed and back edges will be omitted, giving us a DAG. If the longest path in this DAG has $|V| - 1$ edges then there must be a Rudrata path in G since any simple path with $|V| - 1$ edges must visit every vertex, so if this is true, we can say there exists a Rudrata path in the original graph. Since running DFS takes polynomial time ($O(|V| + |E|)$), this reduction is valid.

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 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.)

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 problem in which a potential solution can be verified in polynomial time.
- P: a problem which can be solved in polynomial time.
- NP-Complete: a 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 prove that a problem **is in NP,** you must show there exists a polynomial verifier for it.

To prove that a problem **is NP hard,** you can reduce an NP-Complete problem to your problem.

4 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 $A \in \mathsf{NP}$, there exists a polynomial-time reduction from A to B.

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

(c) 2 SAT is NP-complete.

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

(d) Minimum Spanning Tree is in NP.

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

5 Runtime of NP

True or False (with brief justification): Suppose we can show for some fixed k, an NP-complete problem P has a time $O(n^k)$ algorithm. Then every problem in NP has a $O(n^k)$ time algorithm.