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.

Reduction: Suppose we have an algorithm to solve problem A, how to use it to solve problem B?

This has been and will continue to be a recurring theme of the class. Examples so far include

- Use SCC to solve 2SAT.
- Use LP to solve max flow.
- Use max flow to solve mincut.
- Use max flow to solve maximum bipartite matching.

In each case, we would transform the instance of problem B we want to solve into an instance of problem A that we can solve. Importantly, the transformation is efficient, say, in polynomial time.

Conceptually, a efficient reduction means that problem B is no harder than A. On the other hand, if we somehow know that B cannot be solved efficiently, we cannot hope that A can be solved efficiently.

To show that the reduction works, you need to prove (1) if there is a solution for an instance of problem A, there must be a solution to the transformed instance of problem B and (2) if there is a solution to the transformed instance of B, there must be a solution in the corresponding instance of problem A.

1 Vertex Cover to Set Cover

In the minimum vertex cover problem, we are given an undirected unweighted graph $G = (V, E)$ and asked to find the smallest set $U \subseteq V$ that "covers" the set of edges E. In other words, we want to find the smallest set U such that for each $(u, v) \in E$, either u or v is in U.

Now recall the definition of the minimum set cover problem: Given a set U of elements and a collection $S_1, \ldots, S_m$ of subsets of U, the problem asks for the smallest collection of these sets whose union equals U.

Give an efficient reduction from the minimum vertex cover problem to the minimum set cover problem.

2 Maximum Spanning Tree

In this class, we have been talking about minimum spanning tree. What about maximum spanning tree? Can you use the minimum spanning tree algorithms we learned, Prim's and Kruskal's, as blackbox to find maximum spanning tree? Assume the graph is undirected and with positive edge weights.

3 SAT and Integer Programming

Consider the 3SAT problem, where the input is a set of clauses and each one is a OR of 3 literals. For example, $(x_1 \vee \overline{x}_4 \vee \overline{x_7})$ is a clause which evaluated to true iff one of the literals is true. We say that the input is satifiable if there is an assignment to the variables such that all clauses evaluate to true. We want to decide whether the input is satifiable.

On the other hand, consider the 0-1 linear programming problem. The setup is exactly the same as LP, except that the optimization problem is allowed to have 0-1 constraints such as $x_i \in \{0, 1\}$.

Show how to use 0-1 linear programming to solve 3SAT.

4 Decision vs. Search vs. Optimization

The following are three formulations of the VERTEX COVER problem:

- As a *decision problem*: Given a graph G, return TRUE if it has a vertex cover of size at most b, and FALSE otherwise.
- As a *search problem*: Given a graph G, find a vertex cover of size at most b (that is, return the actual vertices), or report that none exists.
- As an *optimization problem*: Given a graph G, find a minimum vertex cover.

At first glance, it may seem that search should be harder than decision, and that optimization should be even harder. We will show that if any one can be solved in polynomial time, so can the others.

(a) Suppose you are handed a black box that solves VERTEX COVER (DECISION) in polynomial time. Give an algorithm that solves VERTEX COVER (SEARCH) in polynomial time.

(b) Similarly, suppose we know how to solve VERTEX COVER (SEARCH) in polynomial time. Give an algorithm that solves VERTEX COVER (OPTIMIZATION) in polynomial time.