- You know the drill now: Find students around you to form a ***small group***; use ***all resources*** to help to solve the problems; ***discuss*** your idea with other group member and ***write down*** your own solutions; raise your hand and pull the ***course staffs*** to help; ***submit*** your writeup through Gradescope in *24 hours*.

Our topic for this working session is on *self-reducibility*.

Let's recall the fundamental concept of polynomial-time reducibility. There are two variants:

- A problem L is ***oracle reducible*** to a problem R when we can solve any instance of L in polynomial time by querying an oracle of R that correctly solves any instance of R.
- A problem L is ***mapping reducible*** to a problem R if there is a polynomial-time algorithm f such that given any input $w \in \Sigma^*$, we can produce an output $f(w)$, such that $w \in L$ if and only if $f(w) \in R$.

Mapping reduction is stricter than oracle reduction because we can query the oracle *only once*, and have to make the same decision as the oracle. Put it differently, oracle reduction allows us to query the oracle *adaptively*—we get to decide what question to ask next depending on the previous responses. In this worksheet we will focus on the notion of oracle reduction. We leave the mapping reduction to Wednesday, when we try to prove various problems being NP-hard.

Let's say we can solve the decision version of a problem in polynomial time. Can we solve the corrsponding search version efficiently as well? This is the concept of *self-reducibility*: many practical problems (in fact, all NP-complete problems) have this property, which is the main reason we mostly work with decision problems instead of the search problems.[1]

Example. First write down the decision version of the following problem, then describe a polynomial-time self-reduction assuming an oracle to the decision problem.

> FINDSAT
> - ***Input:*** *A circuit C*
> - ***Output:*** *A satisfying assignment x for C (that is, $C(x) = \text{TRUE}$).*

Solution: We will describe an oracle reduction to the following decision problem.

> CIRCUITSAT
> - ***Input:*** *A circuit C*
> - ***Output:*** *Is there a satisfying assignment x for C?*

To obtain a satisfying assignment $x = (x_1, \ldots, x_n)$: For each iteration i, assume that $x_1, \ldots, x_{i-1}$ has been hardwired, we query the oracle for CIRCUITSAT by hardwiring x_i to be 1.

- If the oracle replies YES, then we proceed to the next iteration (without changing the hardwired $x_i = 1$).
- If the oracle replies NO, then we proceed to the next iteration by hardwiring $x_i = 0$.

We made n oracle queries and the reduction can be done in polynomial time. By the oracle promise the resulting hardwired assignment must be satisfying.

[1] There are, however, problems where the decision version is trivially true but the corresponding search version is difficulty. Computing prime factors of an integer, or Nash equilibrium of a two-player game are two examples. Look up the complexity classes PPAD and PLS, and some problems within.

Write down the decision version of each of the following problems, then describe a polynomial-time self-reduction assuming an oracle to the decision problem.

1. FINDMAXCLIQUE
   - ***Input:*** *An undirected graph G*
   - ***Output:*** *Compute a maximum-size clique in G.*

2. FIND3COLORING
   - ***Input:*** *An undirected graph G*
   - ***Output:*** *Compute a proper vertex 3-coloring of G (if exists).*

To think about later: (No submissions needed)

3. FINDLONGESTPATH
   - ***Input:*** *An undirected graph G, a parameter k, two vertices s and t*
   - ***Output:*** *A path between s and t with length at least k (if exists).*

Conceptual question: Is it possible to construct an oracle reduction from the CIRCUITUNSAT problem—given a circuit C, decide if there are *no* satisfying assignment for C—to CIRCUITSAT? Since CIRCUITSAT is NP-hard, does this prove that CIRCUITUNSAT is also NP-hard? But then knowing that CIRCUITSAT is in NP, we just showed that CIRCUITUNSAT is in co-NP, and thus establishing NP = coNP?