- 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 *classifying problems*.

In the lecture we introduced two of the most important complexity classes:

- ***P***: the class of (decision) problems that can be *solved* (by some TM) in polynomial time
- ***NP***: the class of (decision) problems that can be *verified* (by some TM) in polynomial time

The general goal of complexity theory is to study the power of various algorithmic paradigms (that is, Turing machines with certain behaviors), their relations between each other, in hope to understand how to classify problems into various difficulty levels, so that we know what tools to look for that would give us a better chance to solve a new problem we encountered.

In this exercise we just want to classify the problems into three buckets: those that are efficiently solvable (thus in P), those that are efficiently verifiable (thus in NP), and those that are merely computable (but maybe outside NP). Of course every problem in P is also in NP, and every problem in NP is computable (do you see why?), so try to put the problem in the smallest complexity class possible. (The whole purpose of CS31 is to give you tools to pull more problems into P. Tomorrow I'll going to show you how to demonstrate some problems are likely *not* in P.)

A small technicality: from this point on we will take liberty and define a problem either formally as a language, or simply as a collection of input-output pairs. In other words, the language

$$\textsc{Connected} := \{\langle G\rangle : G \text{ is a connected undirected graph}\}$$

is intuitively the same as the decision problem:

> CONNECTED
> - ***Input:*** *An undirected graph G*
> - ***Output:*** *Is graph G connected?*

Notice that the encoding $\langle G\rangle$ is doing all the heavy-lifting; but we will make the assumption that the encoding is *reasonable*: that is, one can transfer from one natural encoding to another using a Turing machine in polynomial time. Do you see how robustness of the definition of P helps?

Example. Put the following problem into the smallest complexity classes possible: P, NP, or merely computable?

> SHORTESTPATH
> - ***Input:*** *An undirected graph G, a parameter k, two vertices s and t*
> - ***Output:*** *Is there a (simple) path between s and t with length at most k?*

Solution: SHORTESTPATH is in P, because we can perform a breadth-first search from s and decide the distance between s and t in linear time, and compare the distance with k before returning an answer.

Example. Put the following problem into the smallest complexity classes possible: P, NP, or merely computable?

> LONGESTPATH
> - ***Input:*** *An undirected graph G, a parameter k, two vertices s and t*
> - ***Output:*** *Is there a (simple) path between s and t with length at least k?*

Solution: LONGESTPATH is in NP, because if someone else provided a sequence of vertices and edges as proof, we can verify that (1) it is indeed a simple path between s and t, (2) the length is at least k, both in linear time.

Put the following problems into the smallest complexity classes possible: P, NP, or merely computable?

1. TRIANGLE
   - ***Input:*** *An undirected graph G*
   - ***Output:*** *Does graph G contain a triangle?*

2. MAXCLIQUE
   - ***Input:*** *An undirected graph G, a parameter k*
   - ***Output:*** *Does graph G contain a clique of size at least k?*

3. PRIMEPOWER
   - ***Input:*** *An integer n, and a prime p*
   - ***Output:*** *Is $n = p^k$ for some positive integer k?*

4. COMPOSITES
   - ***Input:*** *An integer n*
   - ***Output:*** *Is $n = pq$ for some positive integers $p, q > 1$?*

To think about later: (No submissions needed)

5. ALLDFA
   - ***Input:*** *A DFA M*
   - ***Output:*** *Does M accept every string in Σ^*?*

⋆6. ALLNFA
   - ***Input:*** *A NFA N*
   - ***Output:*** *Does N accept every string in Σ^*?*

Conceptual question: The class P is closed under complement exactly as in the case of regular languages: we can just flip the accepting and rejecting states. Is the class NP closed under complement?