Computational Complexity Theory

This document contains summary of some of the items discussed as part of the COMS 3110 Lectures. This document **does not replace the lecture materials**. This document may contain some topics that are not covered as part of the lecture; you will not be tested on those parts, they are made available to you for gaining further knowledge on topics/concepts that are related to class-lecture.

A **decision problem** is a computational problem that can be posed as a yes-or-no question about a given input. Decision problems are typically formalized using the concept of languages. A **language** is defined as a set of strings over a finite alphabet Σ. A decision problem X is represented by a language $L \subseteq \Sigma^*$, where each string in L encodes an instance of X for which the correct answer is YES. In this framework, solving the decision problem means deciding, for any input string $s \in \Sigma^*$, whether $s \in L$.

Let $L \subseteq \Sigma^*$ be the language representing a decision problem X. We say that algorithm A **solves** X if for every string $s \in \Sigma^*$:

$$A(s) = \begin{cases} \text{YES} & \text{if } s \in L \\ \text{NO} & \text{if } s \notin L \end{cases}$$

A **verifier** is an algorithm $V(s,t)$ that takes as input a string $s \in \Sigma^*$ representing an instance of X and a string $t \in \Sigma^*$ called a "certificate" or "withness". We say that V **verifies** L if for every string $s \in \Sigma^*$:

$$s \in L \iff \exists t \in \Sigma^* \text{ such that } V(s,t) = \text{YES}.$$

In this context, the string t serves as a certificate or proof that s is a YES instance of X.

Complexity theory is concerned with classifying computational problems according to their inherent difficulty and the resources required to solve them. Key classes and concepts in computational complexity include **P**, **NP**, **NP-Hard**, **NP-complete**, **EXP**, **R**.

- **Class P** contains problems that can be solved in polynomial time, e.g., $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^3)$, etc. Problems in this class are considered **tractable**, meaning that they can be solved efficiently even for large inputs.
  Example: Sorting an array, finding the shortest path in a graph, finding cycles in a graph, MST, 2-coloring (using BFS), 2-SAT, Euler Cycle.

- **Class NP** contains *decision problems* for which a solution, once provided, can be *verified* in polynomial time, i.e., $V(s,t)$ is a polynomial-time algorithm and t is of polynomial size. These problems may or may not be solvable in polynomial time. But even if finding the solution may be difficult, once a solution is provided, we can efficiently check whether it is correct.
  Example: The Traveling Salesman Problem (TSP), Boolean Satisfiability (SAT).

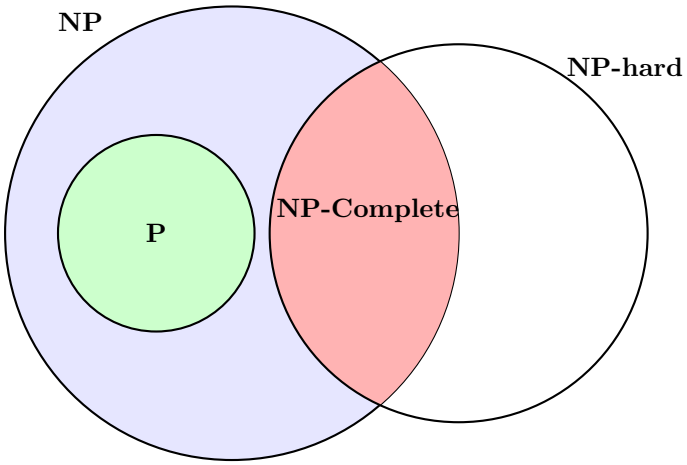
- **Class NP-Hard** contains problems that are at least as hard as every problem in **NP**. We have no polynomial algorithms for these problems, but we have no proof that such algorithms do not exist either. In other words, if we could solve any **NP-hard** problem in polynomial time, we could solve all **NP** problems in polynomial time. Note that **NP-hard** problems do not have to be in **NP**. They may not even be decision problems.
  Example: The optimization version of the Traveling Salesman Problem (TSP).

- **Class NP-Complete** contains problems that are both in **NP** and **NP-hard**. In other words, **NP-complete** problems are the "hardest" problems in **NP**. The importance of **NP-complete** problems lies in their connection to the famous **P vs NP** question. If any **NP-complete** problem can be solved in polynomial time, then every problem in **NP** can and we can conclude that **P = NP**.
  Example: SAT, 3-SAT, CDP, 3-coloring, Hamiltonian Cycle.

- **Class EXP** contains problems that can be solved in exponential time, e.g., $O(2^n)$, $O(n!)$, or $O(2^{p(n)})$, where $p(n)$ is a polynomial function of n. **EXP** is a larger class than **NP**, i.e., **NP** $\subseteq$ **EXP**. In other words, any problem in **NP** can be solved in exponential time by exhaustively checking all possible solutions because the number of possible solutions to an **NP** problem is bounded by the polynomial-time verification process.
  Example: Solving certain types of games (like chess) exhaustively.

- **Class R** (also known as the class of **recursive** problems) contains problems that can be decided in finite time, meaning that an algorithm exists that always terminates with a solution. Problems in this class are consider **decidable**.
  Example: The Halting Problem is undecidable and not in class **R**.

Assuming P $\neq$ NP, then we have the following relationship for decision problems (i.e., NP-complete does not overlap with P).



Polynomial-Time Reductions: A polynomial-time reduction is a technique used to show that one problem is at least as hard as another. This is typically done by transforming an instance of one problem into an instance of another problem in polynomial time. If a solution to the second problem can be found, it can be used to solve the first problem.
Example: Reducing the 3-SAT problem to the Clique Decision Problem (CDP).

P vs NP Problem

P vs NP, namely whether P=NP, is one of the most fundamental and long-standing open problems in theoretical computer science and mathematics. It was first mentioned in a 1956 letter from Kurt Gödel to John von Neumann, two of the greatest mathematical minds of the twentieth century. Informally, it asks whether every problem whose solution can be *verified* efficiently (i.e., in polynomial time) can also be *solved* efficiently. The precise statement of this problem was introduced by Stephen Cook (and independently by Leonid Levin, then in Russia) in his 1971 paper, *The Complexity of Theorem Proving Procedures*. The problem became a phenomenon when Richard Karp demonstrated in his 1972 paper that many well-known combinatorial problems are NP-complete.

This question is listed on the *Millennium Prize Problems* by the Clay Mathematics Institute, with a $1 million prize for a correct solution. Despite decades of research, the problem remains unsolved. Most experts (99% according to 2019 poll) believe that P $\neq$ NP but there is no proof.