Fall 2021 - Analysis and Design of Algorithms

Lecture 10: NP-Completeness 1

Ahmed Kosba

Department of Computer and Systems Engineering
Faculty of Engineering
Alexandria University

Review - Polynomial-time Algorithms

- **Most** of the problems we studied in the course can be solved using **polynomial-time** algorithms.

  What is a polynomial-time algorithm?

  For an input size *n*, the worst-case running time is $O(n^k)$ for some constant *k*.

- Exercises
  - Is Merge sort a polynomial-time algorithm?
  - Is Dijkstra's algorithm a polynomial-time algorithm?
  - Is the dynamic programming algorithm for solving the 0-1 knapsack problem a polynomial-time algorithm?

Exercise – Naïve Primality Testing

The following is one possible way to inspect whether an integer *n* is a prime number.

```
isPrime(n){
        if n <= 1
                return false
        for i = 2 to sqrt(n)
                if n mod i == 0
                        return false
        return true
}
```

Is this a polynomial-time algorithm?

No. Although the above runs in O(sqrt(*n*)), *n* is not the input size.

Note that the input size here is lg *n*.

Primality Testing

- The previous pseudocode for deciding whether a number is prime or not is not a polynomial-time algorithm.

- This does not mean that the problem is not solvable in polynomial-time.
- There are more efficient polynomial-time algorithms to check whether a number is prime or not, but beyond our scope here.

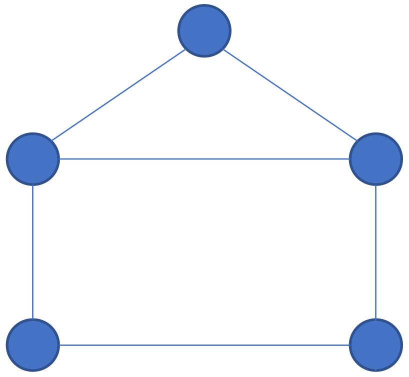
- The next question is:
    - Do all problems have polynomial-time solutions?

Classification of Problems

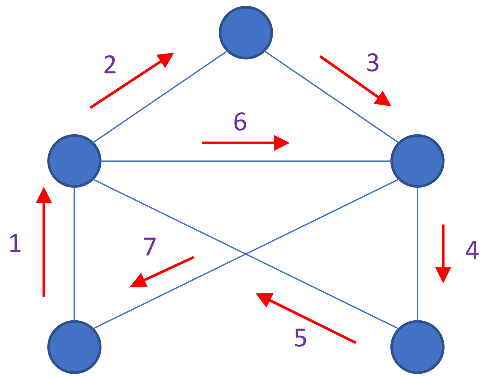
- Undecidable problems
  - Problems that cannot be solved by a computer, regardless of how much time/resources are allowed.
  - Example: Turing's Halting Problem
  - This will be out of the scope of this lecture.
- **Intractable or hard problems**
  - Problems that do not have efficient (polynomial-time) solutions.
- **Tractable or easy problems**
  - Problems that have polynomial-time solutions.

Tractable Problem Example

- **Determine whether a graph has an Euler tour:**
  An Euler tour is a **cycle** that traverses **each edge** of the graph **exactly once**.
  Note: it is allowed to visit each vertex more than once.



This graph cannot have an Euler tour, why?

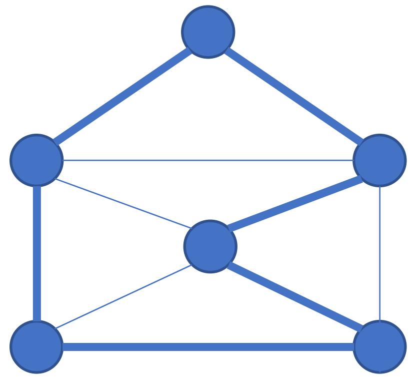


This graph has an Euler tour.

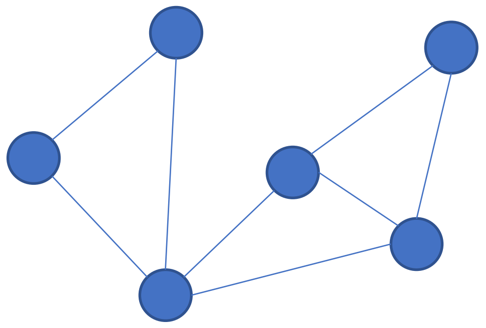
- Determining whether a graph has an Euler tour is easy. This can be solved in O(|E|) time. Finding the tour can also be done in O(|E|).
- A known result: If each vertex has even degree, then an Euler tour exists.

Intractable Problem Example

- **Determine whether a graph has a Hamiltonian cycle:** A Hamiltonian cycle is a **simple cycle** that visits **each vertex** of the graph **exactly once**.



This graph has a Hamiltonian cycle



This graph does not have a Hamiltonian cycle.

This problem is intractable. Also, it's an NP-complete problem. (We will see what this means in this lecture)

Overview of Complexity Classes

Throughout this lecture, we will refer to three complexity classes:

- **P:** The class of decision problems that can be solved in polynomial time.
- **NP:** The class of decision problems that are **verifiable** in polynomial time.
  - Given a certificate of a "yes" answer, the correctness can be verified in polynomial time w.r.t. the size of the input.
  - Note: NP stands for non-deterministic polynomial.
- **NPC (NP-complete):** A problem is NP-complete if it is in NP, and as hard as any problem in NP.

Note: We will discuss the details during the lecture.

Overview of Complexity Classes

Goal: Learn about the classification of problems and the implications.

Examples:

- The Hamiltonian cycle problem is not in P.
  - However, it is in NP. Why?

    A certificate of a "yes" answer, e.g., the cycle itself, can be verified in polynomial time.
  - It is also an NP-complete problem.

- The Euler tour problem is in P and is also in NP.
  - In fact, any problem in P is also in NP.
  - The Euler tour problem is **not** an NP-complete problem.

We will discuss each of the above statements in more depth.

Motivation

- From a practical perspective, why is it important to learn about complexity classes and hard problems?
  - If a problem is proved to be an NP-complete problem, then this means that the problem is intractable (hard).
  - It will be very unlikely to find a polynomial-time algorithm for solving it.
  - Instead of spending effort to find an efficient algorithm, it would be better to develop an approximation algorithm, or to focus on easier cases of the problem.

Lecture Outline

- Basic definitions
    - Definition of a problem
    - Decision vs. optimization problems
    - Complexity classes
- Relation between complexity classes
- How to prove a problem to be NP-complete
- Famous NP-complete problems (Next lecture)

Most of the definitions and examples in the next slides are based on the CLRS textbook.

Abstract Problem Definition

- Two terms appear when discussing problems:
  Problem Instance – Problem Solution
- Definition: An abstract problem is defined as a binary relation on a set of problem instances and a set of problem solutions.
- Example: For the shortest path problem
  - An instance is a graph and two vertices (source and destination).
  - A solution is a sequence of vertices that constitutes a shortest path.
- A problem can have many (infinite) possible instances.
- An instance can have more than one valid solution.

Decision vs. Optimization Problems

- **Optimization problem**
  - The goal is to find a feasible solution with the best value.
  - Example: Find the shortest path between the vertices u and v in an unweighted graph G.
- **Decision problem**
  - A problem that can be answered with either "yes" or "no".
  - Example: does the shortest path between two vertices u, v in an unweighted graph G contain at most k edges? (We will call this problem PATH)
- The complexity classes that we consider are defined with respect to decision problems.
- Focusing on decision problems makes formalization easier.

Decision vs. Optimization Problems

- Any optimization problem can be cast as a decision problem. Example:
  - Find the shortest path between the vertices u and v in an unweighted graph G. -> Optimization problem
  - Does the shortest path between two vertices u, v in an unweighted graph G contain at most k edges? -> Decision problem

- The decision problem is not harder than the related optimization problem.
  - If an optimization problem is easy, then the related decision problem is definitely easy. Why?
    - Just use the solution of the optimization problem to solve the decision problem.
  - If a decision problem is shown to be hard, then the corresponding optimization problem is also hard.

Encodings

- To enable a computer to solve a problem, the instances need to be represented in a way that a computer program understands, i.e., binary strings.
- When the problem instances are represented using binary strings, the problem is described as ***concrete***.
- An encoding maps an abstract problem to a concrete problem.
- The running time of an algorithm T(*n*) is measured with respect to *n*, where *n* is the length of the binary string representing the instance, e.g., *n* = |*i*|, where *i* is the binary string representing the instance.

The **P** Class – Definition 1

The set of concrete decision problems that are solvable in polynomial time.

- A problem is solvable in polynomial time if there exists an algorithm that solves **all** problem instances of length n in time $O(n^k)$ where k is a constant.

Another formal definition can be obtained through the use of the formal language framework.

Formal Language Framework

- Notation and Definitions
  - Alphabet Σ: A finite set of symbols
  - A language L over Σ is any set of strings made of symbols of Σ.
  - The empty string is denoted by ε
  - Example: The language of strings that have the same number of zeros and ones.
    $L = \{\varepsilon, 01, 10, 0011, 1100, 0110, 1001, ..\}$, where $\Sigma = \{0, 1\}$
  - The empty language is denoted by ϕ
  - Σ^* is the language of all strings over Σ.
    - If $\Sigma = \{0, 1\}$, then $\Sigma^* = \{$ε, 0, 1, 00, 01, 10, 11, 000, … $\}$
  - For more definitions, check the CLRS textbook.

Decision Problems, Revisited

- For any concrete decision problem, the set of instances of any decision problem is the set Σ^*, where $\Sigma = \{0, 1\}$.

- Each decision problem Q can be viewed as a language L, such that $L = \{x \in \Sigma^* : Q(x) = 1\}$.
  - The language *L* is the set of instances where the output decision is "yes".

Examples

- The language of the Hamiltonian cycle decision problem is:
  $\{\langle G \rangle$: G = (V, E) is a graph with a Hamiltonian cycle}

- The language of the PATH problem
  $\{\langle G, u, v, k \rangle$: G = (V, E) is an undirected unweighted graph,
  u, v are both in V,
  k >= 0 and there exists a path from u to v with at most k edges }

Algorithms and Languages

- An algorithm A accepts a string $x \in \{0,1\}^*$ if A(x) outputs 1.
- An algorithm A rejects a string $x \in \{0,1\}^*$ if A(x) outputs 0.
- The language accepted by an algorithm A is the set of strings that the algorithm accepts

$$L = \{x \in \{0,1\}^* : A(x) = 1\}.$$

- Even if a language L is accepted by an algorithm A, the algorithm A might not reject a string $x \notin L$.
    - The algorithm might get into an infinite loop when $x \notin L$.

- We need an additional notion: Decidability.
    - A language L is **decided** by an algorithm A, if A accepts every binary string in L, and rejects every binary string that is not in L.

Definition of a Complexity Class

- Previously, we defined complexity classes as a set of decision problems.
- Definition using the formal language framework:
  - A complexity class is a set of languages (that correspond to decision problems).
  - For a language L to belong to this set, the membership is determined by a complexity measure of an algorithm that determines whether an input string x belongs to the language L or not.
    - The complexity measure can be the running time or the space used, etc.

The **P** Class – Definition 2

- The set of languages that can be decided in polynomial time.

$$P = \left\{ \begin{matrix} L \subseteq \{0,1\}^*: \text{there exists an algorithm } A \text{ that} \\ \text{decides } L \text{ in polynomial time} \end{matrix} \right\}$$

- It can be shown that P is also the set of languages that can be accepted in polynomial time (The theorem is in the textbook).

The **NP** Class – Informal Def.

The definition of the class NP can appear in different forms:

- The set of decision problems, where the "yes" answers can be **verified** by polynomial-time algorithms given a certificate.
    - This is sometimes expressed as "set of problems verifiable in polynomial time", but it should be noted that we mean verifying the "yes" answers.
- The set of problems that can be solved by a **non-deterministic polynomial-time** algorithm.
    - A non-deterministic algorithm is an algorithm that magically guesses the solution (if it exists) and verifies it.

The first definition is more important for our discussions in this course.

The **NP** Class

A more formal definition:

A language L belongs to the NP class if and only if there exist a two-input polynomial-time algorithm A and a constant c such that

$$L = \left\{ \begin{array}{r} x \in \{0,1\}^* : \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \\ \text{such that } A(x,y) = 1 \end{array} \right\}$$

A is a poly-time verification algorithm that receives both
- The input x (the problem instance)
- The certificate y, which shows that x is in L.

For each x in L, there must exist a valid certificate y.

Ensures that the length of the certificate is polynomial w.r.t. the original input size.

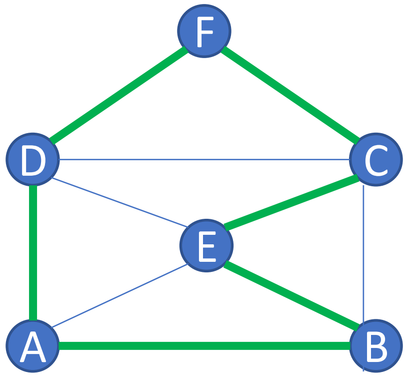
The **NP** Class - Example

Consider the Hamiltonian cycle decision problem:

Is there a Hamiltonian cycle in an input graph G = (V, E)?

- We don't know a polynomial-time solution to this problem.
- However, the "yes" solution to any instance of the above problem can be verified in polynomial time.
- The certificate in this case would be the vertices in the order they appear in the Hamiltonian cycle.
- What would the verification algorithm here do?

The **NP** Class - Example



A possible certificate to verify that the graph has a Hamiltonian cycle is: A -> B -> E -> C -> F -> D -> A

The verification algorithm can simply check that

- The given vertices are a permutation of the input vertices.
- There is an edge between every two consecutive vertices in the certificate
- No vertex was visited twice.

All of this must be done in polynomial time, which is the case.

P vs. NP

- P is the class of decision problems that can be solved in polynomial time.
- NP is the class of decision problems whose "yes" solutions can be verified in polynomial time.

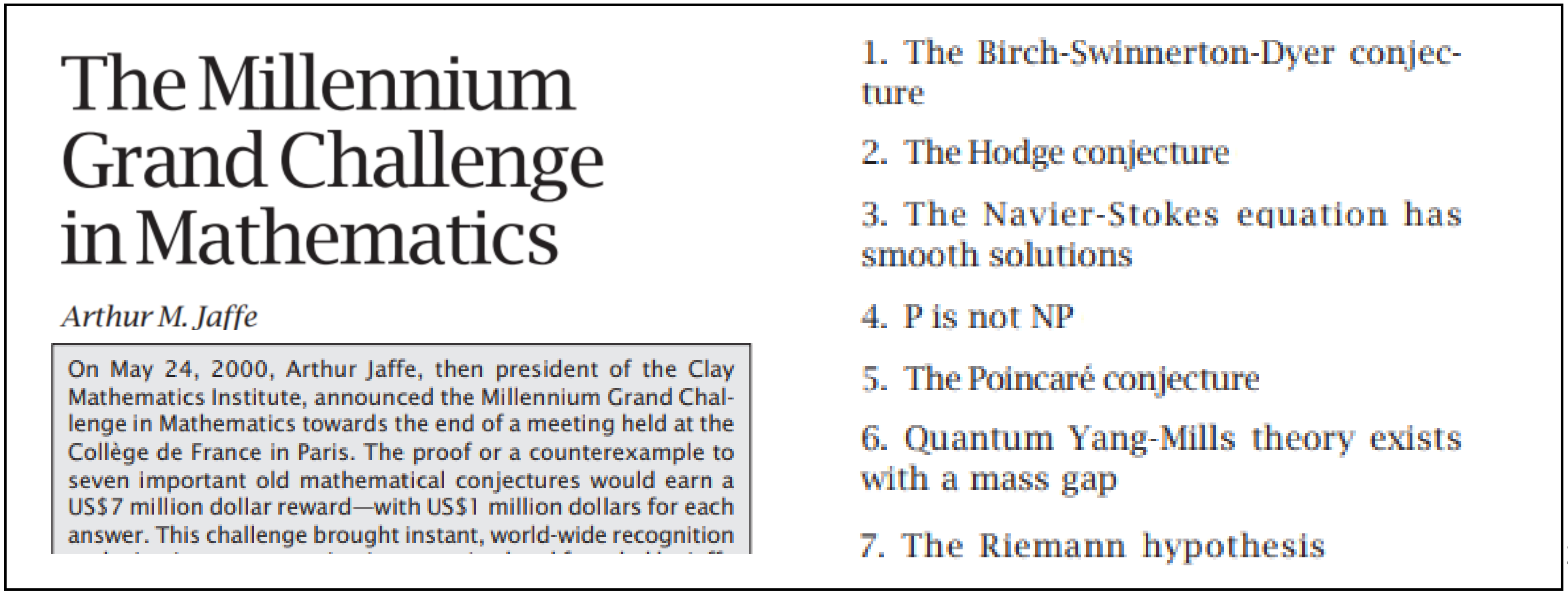
- What is the relation between P and NP?

$$P \subseteq NP$$

The proof is in the textbook.

Open Problem: $P = NP$?

- We don't know whether P = NP or not.
- Most scientists think this is not the case, but there is no proof, yet.
- This is one of seven million-dollar problems.



The Millennium Grand Challenge in Mathematics

Arthur M. Jaffe

On May 24, 2000, Arthur Jaffe, then president of the Clay Mathematics Institute, announced the Millennium Grand Challenge in Mathematics towards the end of a meeting held at the Collège de France in Paris. The proof or a counterexample to seven important old mathematical conjectures would earn a US$7 million dollar reward—with US$1 million dollars for each answer. This challenge brought instant, world-wide recognition

1. The Birch-Swinnerton-Dyer conjecture
2. The Hodge conjecture
3. The Navier-Stokes equation has smooth solutions
4. P is not NP
5. The Poincaré conjecture
6. Quantum Yang-Mills theory exists with a mass gap
7. The Riemann hypothesis

The class co-NP

- If $L \in NP$, does this mean that $\overline{L} \in NP$?

Example:

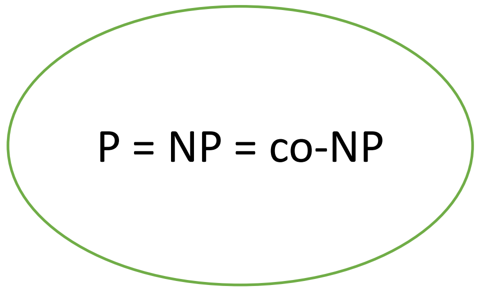
- Let L denotes the language of graphs that have Hamiltonian cycles.
- Then $\overline{L}$ is the language of graphs that do not have Hamiltonian cycles.
- Is $\overline{L}$ in NP?
  - Note: it's not easy to find a certificate of polynomial size for a graph in $\overline{L}$.
  - In general, it's unknown whether $\overline{L}$ is in NP.
  - Instead, we introduce another complexity class co-NP.

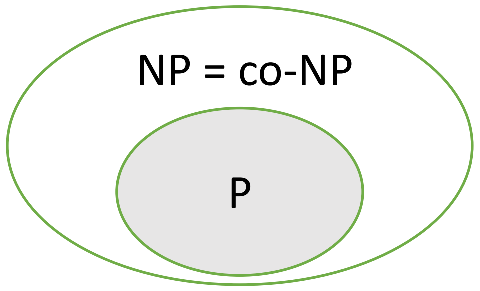
- co$-NP$ is the class of languages L, s.t. $\overline{L} \in NP$.
  - In other words, a language is in co-NP, if its complement is in NP.

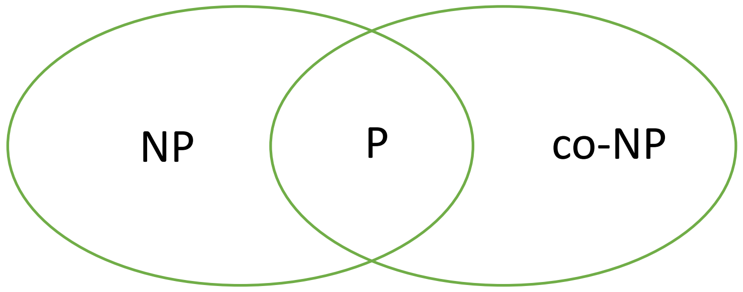
- Open problem 2: Is NP closed under complement? i.e., Is NP = co$-NP$?

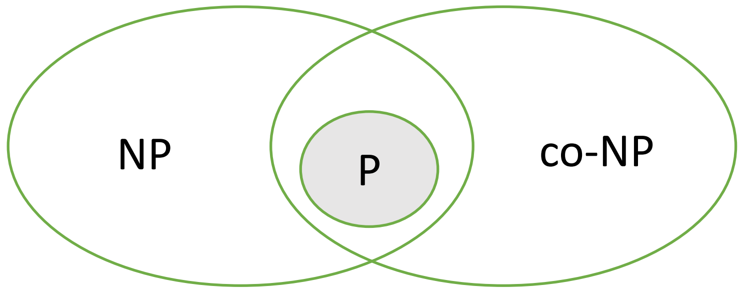
Relation between P, NP, co-NP

Four Possibilities – Based on CLRS:









Believed to be the most likely.

NP-Complete Class

NP-complete Problems

- Properties (informal):
  - Hardest problems in NP.
  - Furthermore, if any NP-complete problem is solved in polynomial-time, then P = NP, i.e., all problems in NP can be solved in polynomial time.
    - **Important: How does solving one problem have this huge implication?**
- Any problem in NP can be **reduced** to any NP-complete problems.
  - This is part of the definition of NP-complete problems, as we will see shortly.
- Before showing the formal definition, we will discuss a key concept: Reducibility.

Reducibility

- A problem Q can be reduced to another problem Q', if we can *easily rephrase* any instance of Q as an instance of Q'.

- We are interested in the cases when this *rephrasing* can be done efficiently, e.g., in polynomial time.

Reducibility Example 1

Q: Problem of solving linear equations

Q': Problem of solving quadratic equations

Any instance of Q can be expressed as:

Find x, such that $ax + b = 0$

This can be rephrased as an instance of Q'

Find x, such that $0*x^2 + ax + b = 0$

Note: The above problems are not decision problems, but the concept will apply normally.

Exercise: Can you find the corresponding decision problems of the above?

Reducibility Example 2

Q: Given a set of n Boolean variables, is all of them True?

Q': Given a set of n integers, is their sum greater than or equal to n?

Any instance of Q can be expressed as:

- Given Boolean variables (x_1, x_2, …, x_n) is all of them True?

This can be rephrased as an instance of Q'

- Given the integer variables (y_1, y_2, …, y_n), where
  $y_i = 1$ if x_i is True, 0 otherwise
  Is their sum greater than or equal to n?

More illustration for example 2

- Reducing Q to Q' means that we can solve Q if we have an algorithm that solves Q'.
- Let's illustrate this using a code example.

Suppose that Q' can be solved by an algorithm A'.

```
A' (int[] y){
        int sum = 0;
        int n = y.length;
        for(int i = 0; i < n; i++){
                sum+=y[i];
        }
        return sum >= n;
}
```

How can we use A' to solve an instance of Q?

More illustration for example 2

- Let's write an algorithm A for solving Q using A'.

```
A (boolean[] x){
        int n = x.length;
        int[] y = new int[n];
        for(int i = 0; i < n; i++){
                y[i] = x[i]? 1: 0;
        }
        return A'(y);
}
```

```
A' (int[] y){
        int sum = 0;
        int n = y.length;
        for(int i = 0; i < n; i++){
                sum += y[i];
        }
        return sum >= n;
}
```

We can use A' to solve any instance of A, but we have to do a reduction first (the blue code).

Additional Examples

- Activity selection can be reduced to finding maximum independent set in a graph.
- Sorting can be reduced to the convex hull problem.

We will see more examples in the next lecture, but in the context of NP-complete problems.

Polynomial-time Reducibility

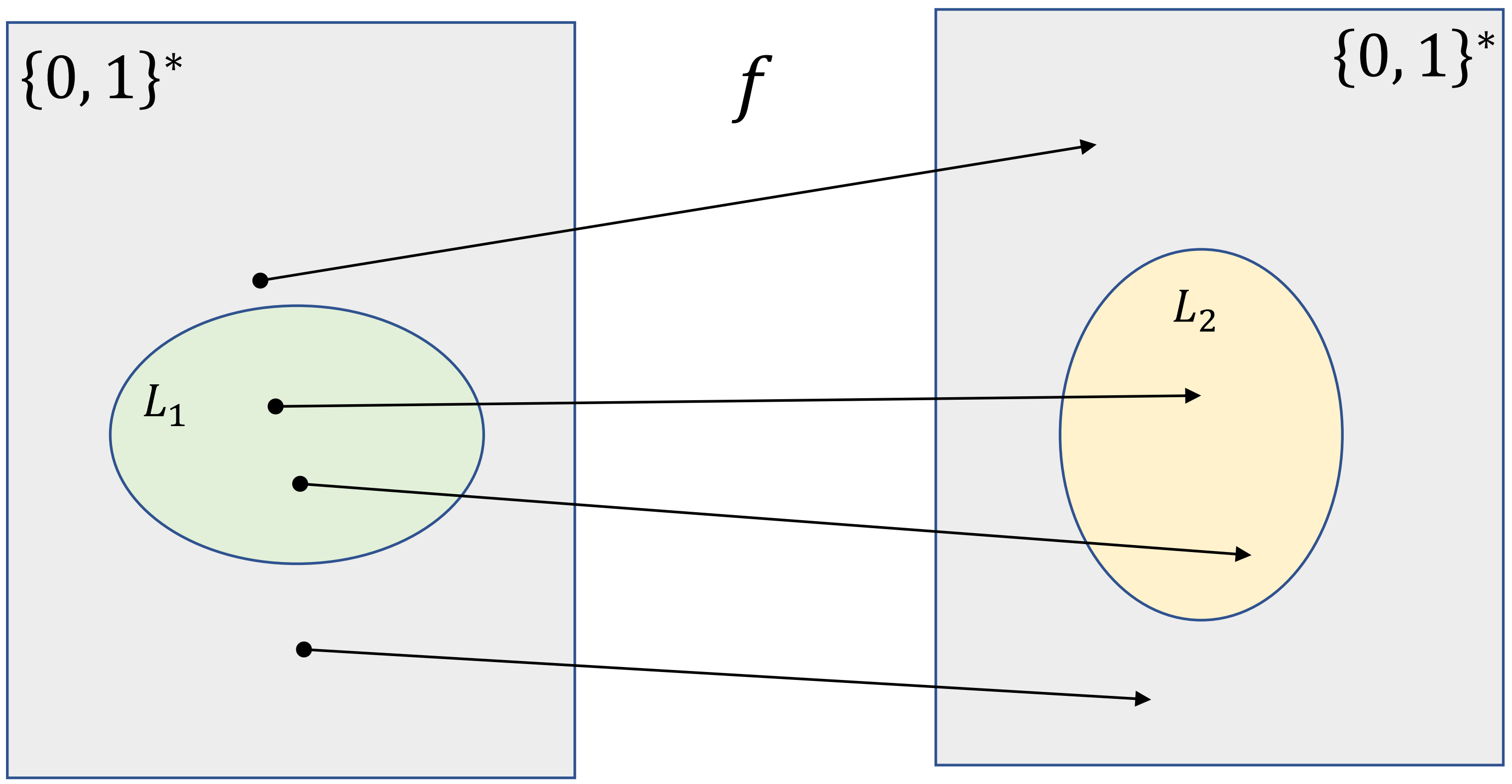
A language L_1 is polynomial-time reducible to a language L_2 if there exists a polynomial-time computable function $f: \{0,1\}^* \rightarrow \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$$x \in L_1 \text{ if and only if } f(x) \in L_2$$

Notation:

$$L_1 \leq_p L_2$$

L_1 is polynomial-time reducible to L_2



Typical decision problem:
Given x, is $x \in L_1$?

Another way to express this question is to check if $f(x) \in L_2$

NP-Complete Class

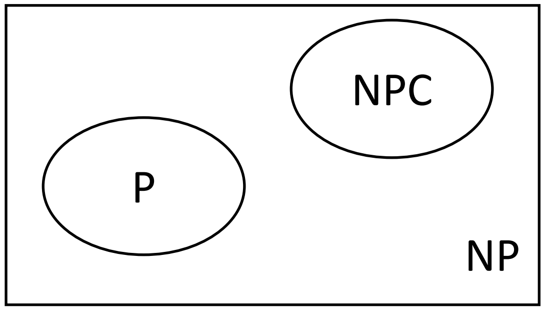
A language $L \subseteq \{0,1\}^*$ is NP-complete if the following conditions hold

1- $L \in NP$

2- $L' \leq_p L$ for every $L' \in NP$

How to show that a problem is NP-complete?

- Proving condition 1 can be similar to what we did in the Hamiltonian cycle problem.
- However, how to prove condition 2?
  The number of problems in NP is infinite!



Assuming P is not equal to NP

NP-Complete Class

- One way to prove that a problem Q is NP-complete:
  - Step 1:
    - Prove that the problem is in NP.
  - Step 2:
    - Start from a known NP-complete problem Q'.
    - Show that Q' is reducible in polynomial-time to Q.

- Next lecture:
  - We will discuss a first NP-complete problem.
  - Then show multiple known NP-complete problems.