Lecture #21
NP Complexity (1)

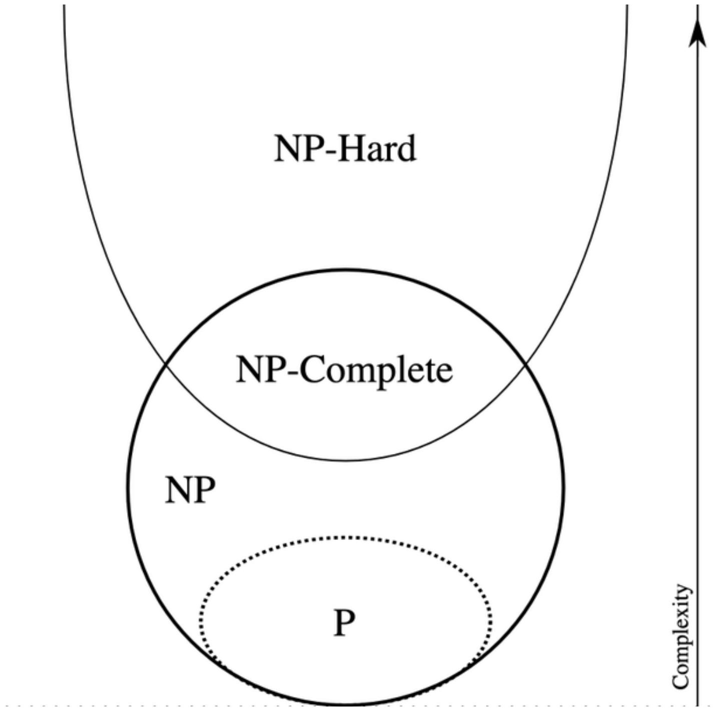
Algorithm

JBNU

Jinhong Jung

In This Lecture

- ❑ Concept of NP
- ❑ P=NP v.s. P≠NP
- ❑ Polynomial-time reduction
- ❑ NP-Hard and NP-Complete



Outline

- ❑ Motivation
- ❑ Types of Problems
- ❑ Concept of NP
- ❑ Polynomial-time Reduction
- ❑ NP-Complete

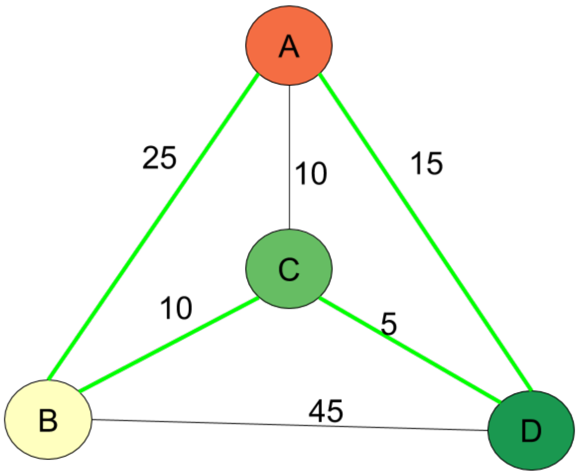
Motivation with TSP

❑ Travelling salesman problem [TSP]

- **Input**: weighted, undirected, and complete graph
- **Output:** shortest distance visiting all nodes and going back to the starting node (e.g., $\mathrm{A} \to \mathrm{B} \to \mathrm{C} \to \mathrm{D} \to \mathrm{A}$)

❑ Algorithms for TSP

- Brute-force: $O(n!)$ time
  - Take all permutation of n nodes
  - Check if each permutation forms a cycle
  - Pick a cycle having the minimum cost



❑ Can we solve TSP quickly in polynomial time?

- Theoretically, no! Why?

Outline

- ❑ Motivation
- ❑ Types of Problems
- ❑ Concept of NP
- ❑ Polynomial-time Reduction
- ❑ NP-Complete

Types of Problems

❑ Unsolvable v.s. solvable problems

Unsolvable problems

- Halting problem
- Hilbert's tenth problem
- …

Solvable problems

Our interest

Cannot be solved within practical time

- Presburger arithmetic
- …

Can be solved within practical time

- Sorting problem
- Minimum spanning tree
- Single source shortest path
- …

Definition of Practical Time

❑ Practical time means a polynomial time (poly-time)

- If a problem of size n takes $f(n)$ time, and $f(n)$ is a polynomial, then the time is called **polynomial time**

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0 \in \Theta(n^k)$$

- A problem is solved within practical time if it is solved within poly-time
  - Note $n^k \log n \leq n^{k+1}$; thus, poly-logarithmic time is also practical
  - Too large degree k leads to impractical time, but in general $k \leq 6$

❑ Exponential or factorial time is impractical time

- e.g., $\Theta(2^n)$ or $\Theta(n!)$ is impractical

Optimization v.s. Decision

❑ Solvable problems are classified as follows:

- **Optimization problem**: What is its best solution?
- **Decision problem**: Is this problem solvable? ⇒ **Yes** or **No**
- Example of TSP
  - **O**: What is the shortest distance visiting all n nodes and going back to the starting node?
  - **D**: Is there a path of length at most K visiting all n nodes and going back to the starting node?

❑ NP complexity theory focuses on decision problems

- Intuitively, answering a decision problem is simpler than answering its optimization problem
  - i.e., if a decision problem is hard, its optimization problem is also hard

Outline

- ❑ Motivation
- ❑ Types of Problems
- ❑ Concept of NP
- ❑ Polynomial-time Reduction
- ❑ NP-Complete

What is NP?

❑ Two complexity classes: P and NP

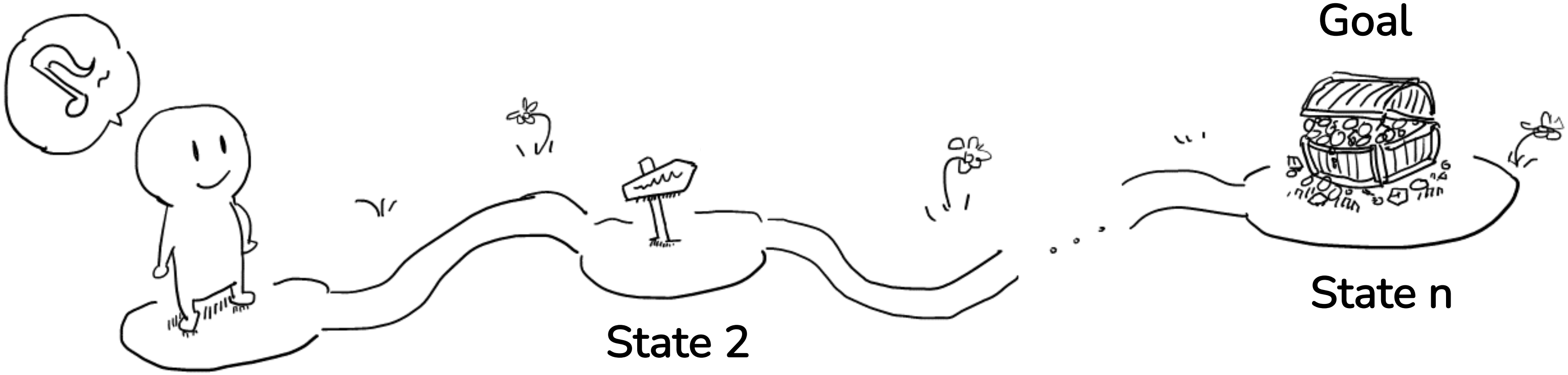
- P is the set of decision problems that can be solved in poly-time (i.e., able to answer Yes or No in poly-time)
  - All problems what we've learnt are in **P** (Polynomial);
    we've focused on developing efficient algorithms under poly-time

- Then, does Non-Polynomial stand for NP? **Never!!!**
  - NP is the abbreviation of **Non-deterministic Polynomial**
  - To understand NP precisely, we first need to know the concept of "**non-deterministic**"

Deterministic Computer (1)

❑ "Deterministic" means we can determine a state of a computer for solving a problem

- The computer can move only one step at a time
- This computer is called "deterministic computer"

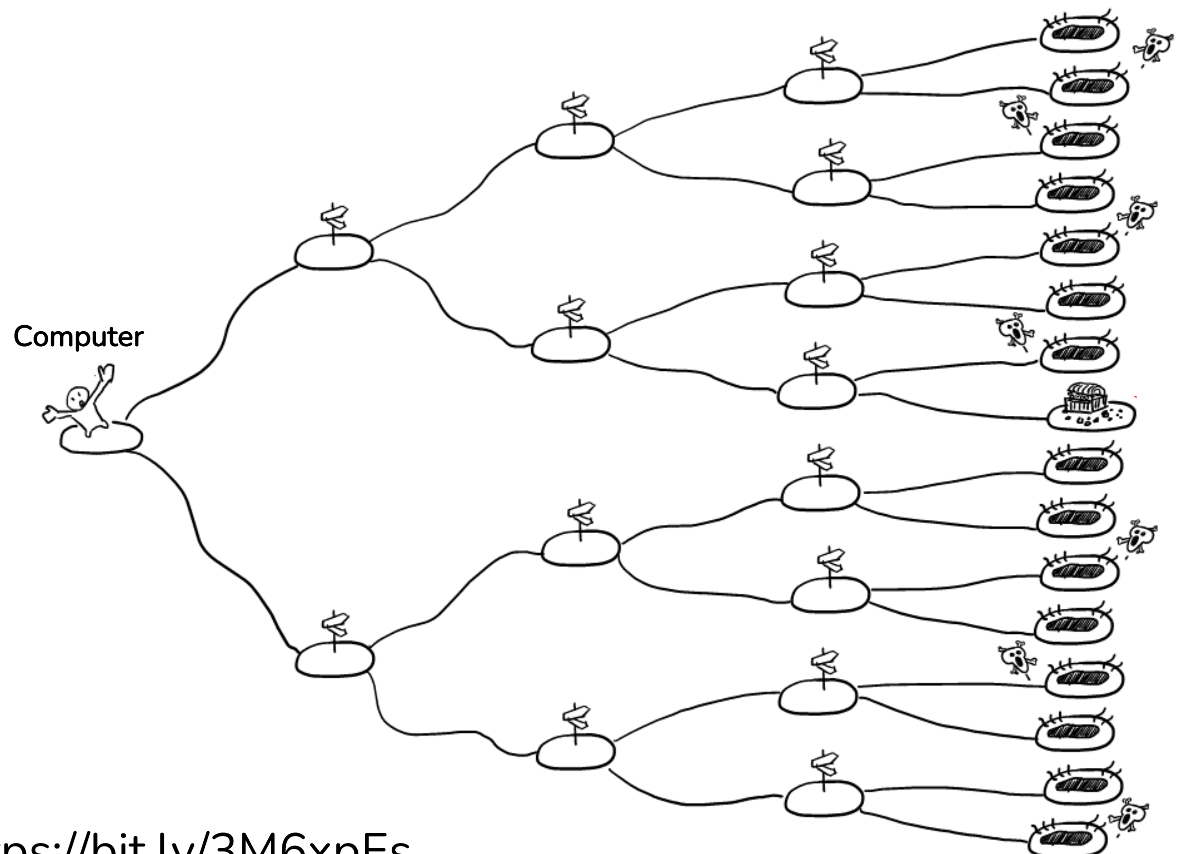


This procedure takes $O(n)$ steps to the final state

Reference: https://bit.ly/3M6xpEs

Deterministic Computer (2)

- **If a problem has the following recursion tree, the computer should check all of states step by step**
  - If the tree's depth is n, it takes $O(2^n)$ steps (exponential)

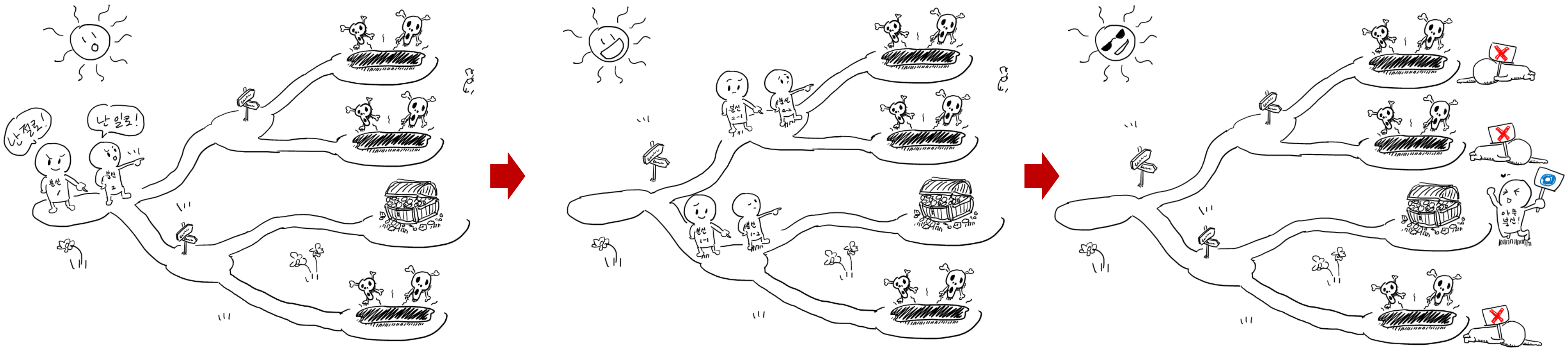


Reference: https://bit.ly/3M6xpEs

Non-deterministic Computer (1)

❑ Consider a computer can make its mirror images (분신)

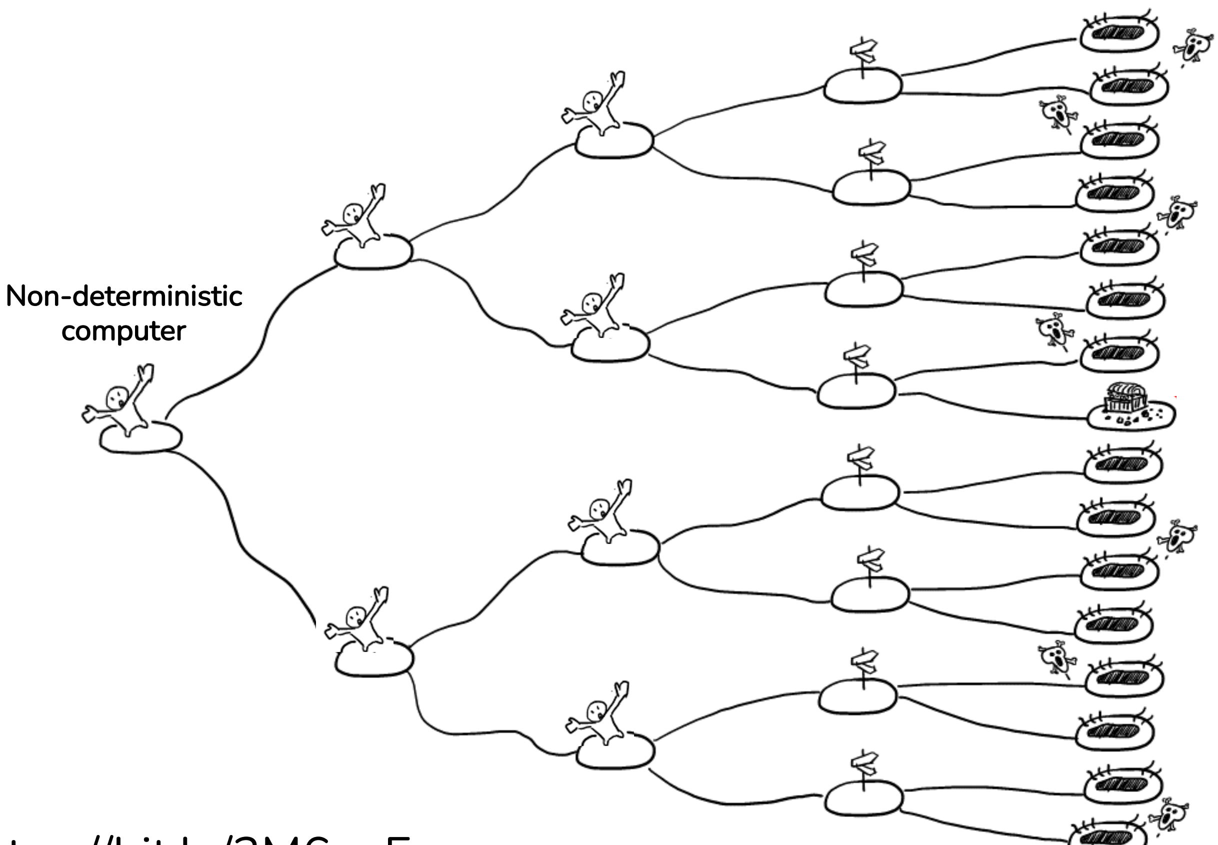
- If it encounters a branch, it makes new images
  - All images share their emotions and thoughts
- If one image gets a goal, the problem is solved
- Such a computer is called "non-deterministic computer"
  - Because it's hard to determine a state of the computer in this case



Reference: https://bit.ly/3M6xpEs

Non-deterministic Computer (2)

- **Non-deterministic computer can solve the problem having the following tree of n depth in $\mathbf{O}(n)$ steps**
  - Meaning non-deterministic computer solves it in poly-time



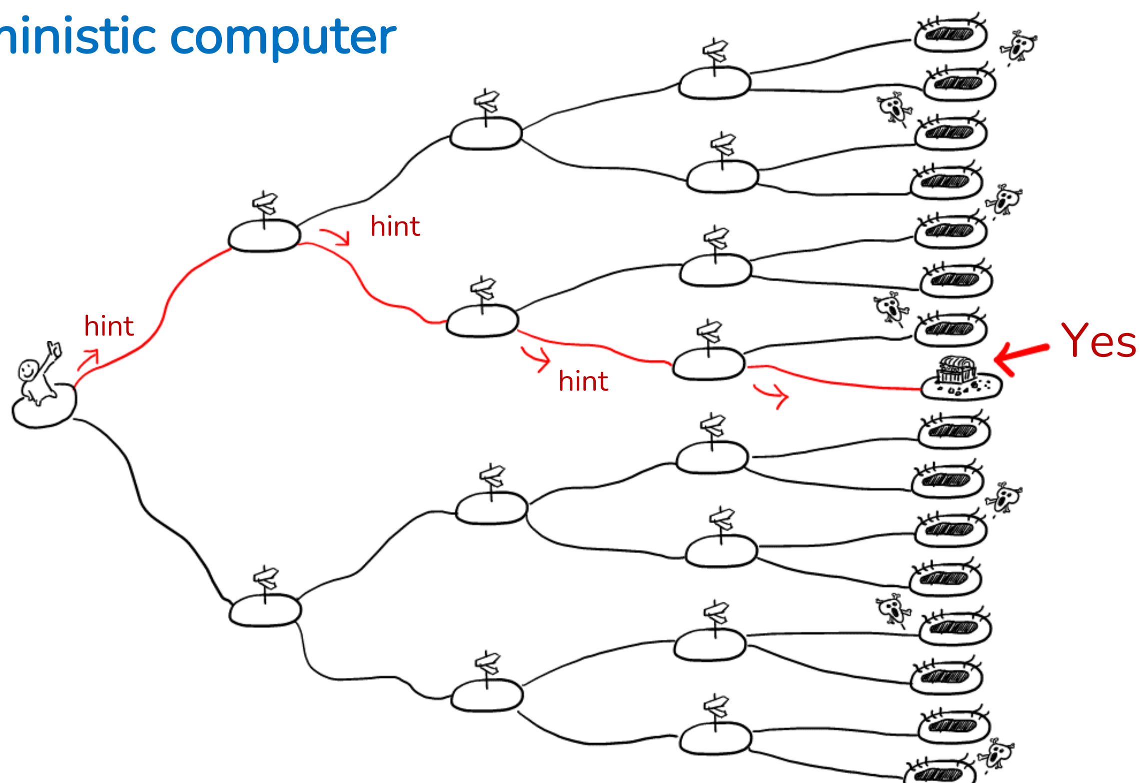
Reference: https://bit.ly/3M6xpEs

What is NP?

- Two complexity classes: P and NP
  - P is the set of decision problems that can be solved in poly-time by a deterministic computer
    - It's called (deterministic) Polynomial time
  - NP is the set of decision problems that can be solved in poly-time by a non-deterministic computer
    - It's called Non-deterministic Polynomial time
- Remark
  - Modern computer is deterministic
  - Non-deterministic computer does not yet exist
    - Cannot make infinite images and communicate them without any cost

Other Definition of NP

- Can be defined in terms of "verify" instead of "solve"
  - NP is the set of decision problems such that given hints leading to the answer "**Yes**", the statement <u>**"the answer for the hints is Yes"**</u> can be verified in **poly-time**
    - By a deterministic computer



Example of NP Problem

❑ Subset sum problem [SUBSET-SUM]

- Given a set S of n positive integers and positive integer t, is there a sub-set whose sum is t?
- This problem is NP. Why?
  - 1) This problem is a decision problem
  - 2) Input and its hint for the answer "**Yes**" is given
    - Input: $S = \{1, 2, 3, 4, 5\}$ and $t = 12$
    - Hint: $A = \{3, 4, 5\}$
  - 3) "The answer for the hint is **Yes**" can be verified by performing the summation on all entries in $A = \{3, 4, 5\}$, which takes $O(n)$ time at most

Summary of Definition of NP

❑ Two complexity classes: P and NP

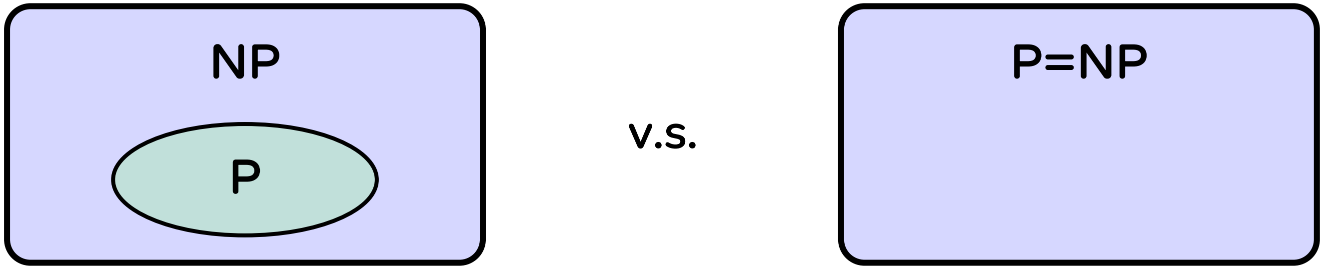
- P is the set of decision problems that
  - Can be solved in poly-time by a deterministic computer
    - Intuitively, problems in P are quickly solvable

- NP is the set of decision problems that
  - 1) Can be solved in poly-time by a non-deterministic computer
  - 2) Can be verified if "**the hint for Yes**" is correct or not in poly-time by a deterministic computer
    - Intuitively, problems in NP are quickly verified

Big Question: P = NP?

❑ Major unsolved question in computer science!

- Every problem in **P** is in **NP**, i.e., **P** $\subseteq$ **NP**
  - Because verifying a problem is easier than solving the problem
- However, we don't know if **P** $\supseteq$ **NP** (nobody answers this)
  - Don't know if there is an **NP** problem can be solvable in poly-time by a deterministic computer
- Two possible situations: **P** $\neq$ **NP** or **P** $=$ **NP**?
  - Surprisingly, the claim has not yet proven!



Many believe this like
the sun rises in the east

But, what if this is true?

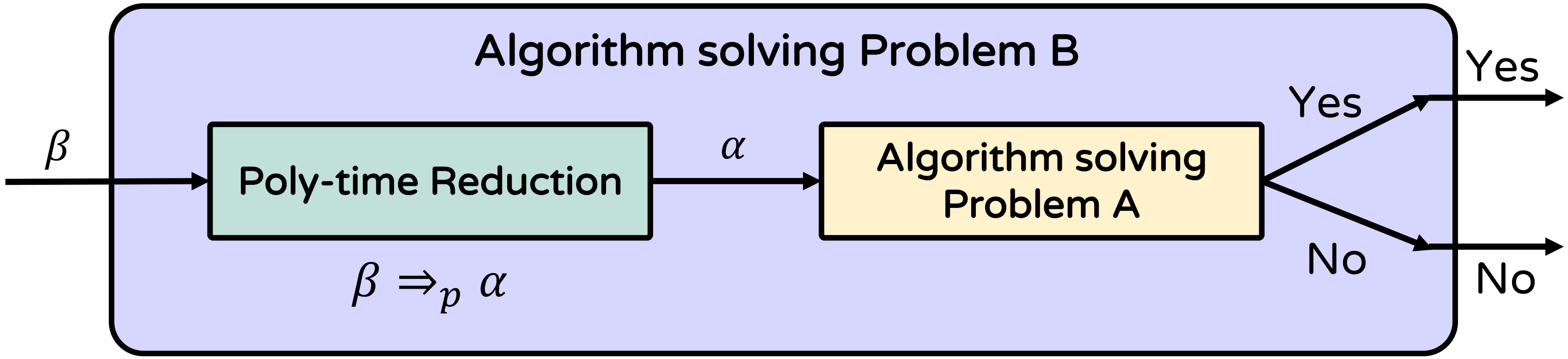
Outline

- ❑ Motivation
- ❑ Types of Problems
- ❑ Concept of NP
- ❑ Polynomial-time Reduction
- ❑ NP-Complete

Polynomial-time Reduction

❑ Definition of polynomial-time reduction

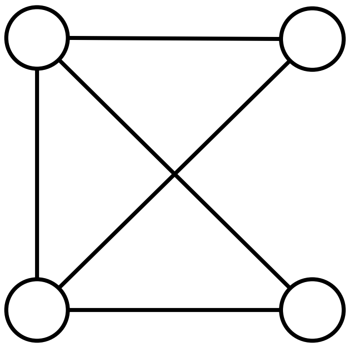
- Given problems A and B, let α and β denote their inputs.
- If β is reduced into α in poly-time and their answers are the same, it's called polynomial time reduction, denoted by $\beta \Rightarrow_p \alpha$



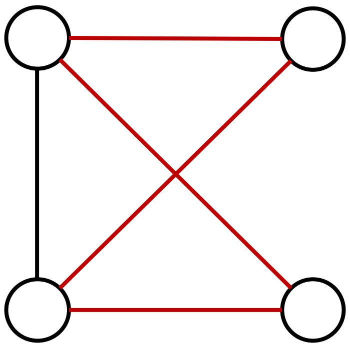
Example of Reduction (1)

❑ Hamiltonian cycle problem [HAM-CYCLE]

- Is there a Hamiltonian cycle in an undirected graph G?
  - Hamiltonian cycle is a cycle where every node is visited exactly once



Undirected graph G



Hamiltonian cycle

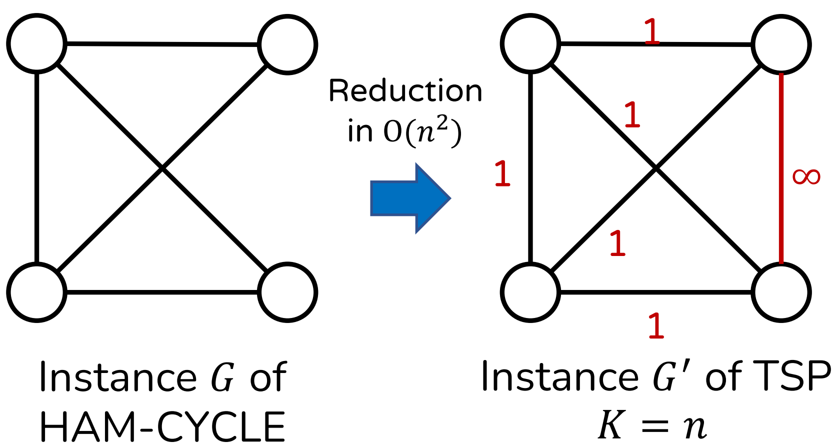
❑ Travelling Salesman Problem [TSP]

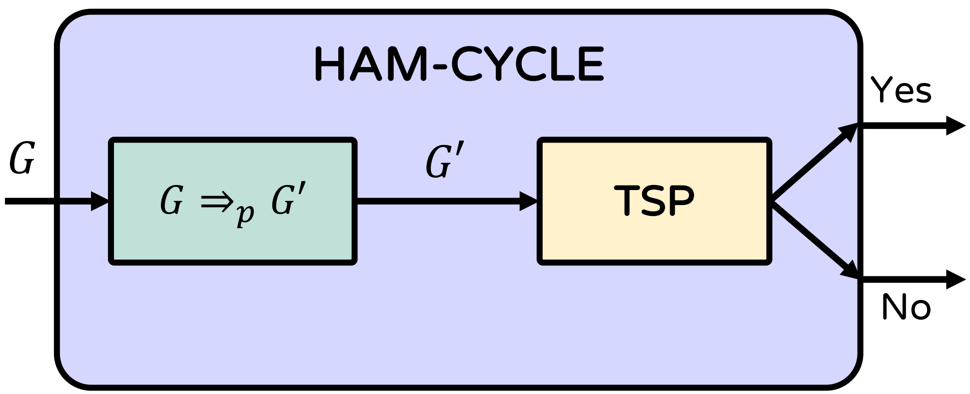
- Given a weighted, undirected, & complete graph G,
  is there a Hamiltonian cycle such that its cost $\leq K$ in G?
  - Note that there is always such a cycle in a complete graph

Example of Reduction (2)

❑ Poly-time reduction from HAM-CYCLE to TSP

- G is the graph of HAM-CYCLE, & G is reduced to G' of TSP
  - For existing edges, put weight 1 to each edge
  - For non-existing edges, fill each of them with weight ∞
  - This takes $O(n^2)$ time because there are $O(n^2)$ edges
- By $K = n$, the answer of TSP is that of HAM-CYCLE

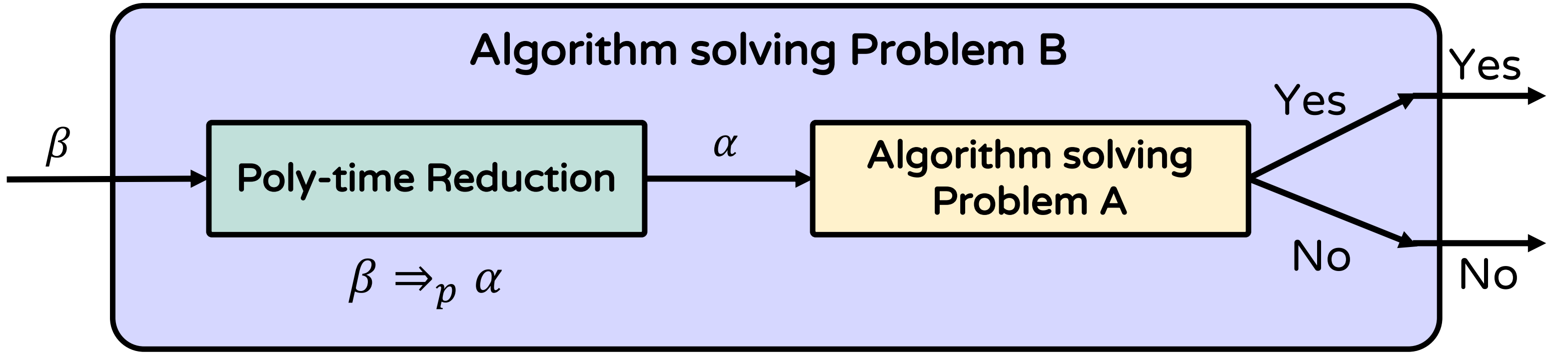




Polynomial-time Reduction

❑ Implication of poly-time reduction

- Let's assume Problem A is solved in (im-)practical time
- Suppose the input of Problem B is reduced in that of Problem A in practical time
  - i.e., quickly translate B's input to A's input & their answers are equal
- Meaning that Problem B is solved in (im-)practical time!
  - i.e., **Problem B is as difficult or easy as Problem A.**



Outline

- ❑ Motivation to NP-Complete
- ❑ Background
- ❑ Concept of NP
- ❑ Polynomial-time reduction
- ❑ **NP-Complete**

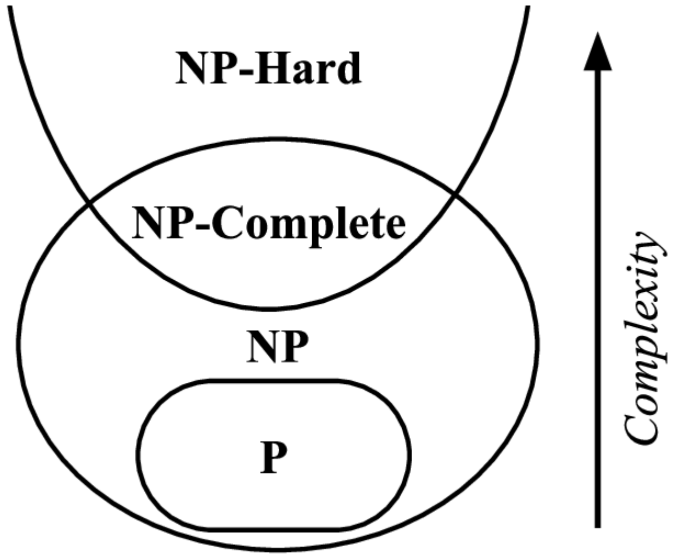
NP-Hard and NP-Complete

❑ Definition of NP-Hard

- Problem A is NP-Hard if every NP problem is reduced to Problem A in poly-time
  - If $\mathrm{L} \Rightarrow_p \mathrm{A}\ \forall\ \mathrm{L} \in \mathrm{NP}$, then A is in **NP-Hard**
  - Both decision & optimization problems are included in **NP-Hard**

❑ Definition of NP-Complete

- Problem A is **NP-Complete** if
  - Problem A is NP-Hard, and
  - Problem A is NP as well
    - Decision problem verifiable in poly-time
    - Don't know if it's solvable in poly-time



Under the assumption P ≠ NP

Note that NP ≠ NP-C ∪ P

What You Need To Know

❑ Concepts of NP

- P = set of decision problems s.t. can be solved in **poly-time**
- NP = set of decision problems s.t.
  - 1) Solved in poly-time by a **non-deterministic** computer
  - 2) Verified if "**the hint for Yes**" is correct or not in poly-time
- NP-Hard = set of problems s.t. each is reduced from every NP problem in poly-time
- NP-Complete = set of decision problems s.t. each is in NP-Hard as well as NP

Thank You