



Lecture 12: Reduction and Computational Intractability

Algorithm

Jeong-Hun Kim

Remind

- ❖ Dynamic programming
 - Recursion
 - Memoization
- ❖ Longest increasing subsequence (LIS)
- ❖ Edit distance

Table of Contents

- ❖ Part 1
 - Reduction
- ❖ Part 2
 - Computational interactability

Part 1

REDUCTION

Reduction

❖ Concept

- Assume that there are two problems X and Y
- There exists an algorithm to solve problem Y
- If it can design an algorithm to solve problem X using the algorithm Y
 - Reduction from problem X to problem Y is possible
 - X is reducible to Y

❖ When should reduction be used?

- Designing an algorithm to solve a specific problem
- Proving the non-existence or extreme difficulty of an algorithm to solve a specific problem

Reduction

❖ Reduction example

- Selection → Sorting
 - Input: an array A of size n, denoted as $A[1, \dots, n]$
 - Selection: find the k-th smallest element in array A
 - Sorting: sort array A in non-decreasing order
- If there is an algorithm to solve the sorting problem, the selection problem can be solved by sorting A and then returning $A[k]$
 - Selection is reducible to Sorting

A	3	7	2	8	6	15	9	10	1	4
---	---	---	---	---	---	----	---	----	---	---

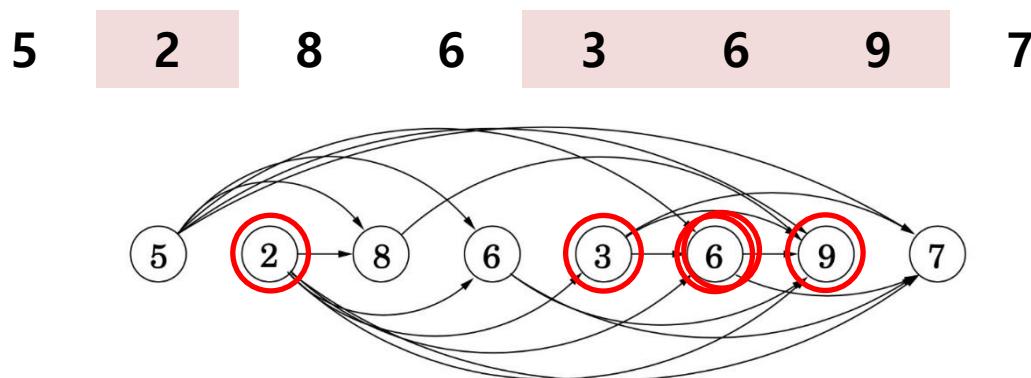


A (sorted)	1	2	3	4	6	7	8	9	10	15
------------	---	---	---	---	---	---	---	---	----	----

Reduction

❖ Reduction example

- Longest increasing subsequence (LIS) → longest path in DAG
- If there exists an algorithm for finding the longest path in a Directed Acyclic Graph (DAG)
 - Converting the sequence into DAG
 - Finding the longest path in DAG
 - Nodes that make up the path correspond to elements of the LIS
 - Therefore, LIS is reducible to DAG



Reduction

- ❖ Type of problems
 - Decision problem
 - E.g., Does there exist an increasing sequence of length 3 or more in A?
 - Search problem
 - E.g., Find all elements greater than or equal to 3 in A
 - Optimization problem
 - E.g., Partition A into the fewest possible substrings such that each substring is an increasing substring
- Because decision problems always have answers in either “YES” or “NO”, it makes it easier to think about reductions between them
- From now on, we will consider only decision problems

Reduction

❖ Problem instance

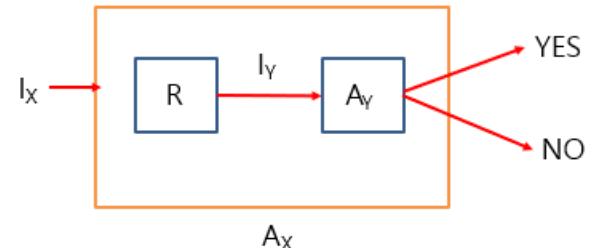
- The input (string) that defines a problem is referred to as a problem instance
- Problem and problem instance are different terms
 - Problem can be defined as a set of problem instances that satisfy specific conditions
- An algorithm that solves a decision problem X takes an instance belonging to X as input and outputs either YES or NO
- E.g., each instance of the decision problem “Does a LIS of length k or more exist in sequence A?” consists of sequence A and the value k

Reduction

❖ Reductions for decision problems

- Assume that there are two decision problems, X and Y
- Reduction R from X to Y
- Algorithm A_Y for solving problem Y
- It is possible to design an algorithm A_X to solve problem X as follows

1. Using R to transform an instance I_X of X into an instance I_Y of Y
2. Determine the answer for instance I_Y using algorithm A_Y
3. The answer obtained in step 2 becomes the answer for I_X



If R and A_Y are polynomial-time algorithms, then A_X also becomes a polynomial-time algorithm

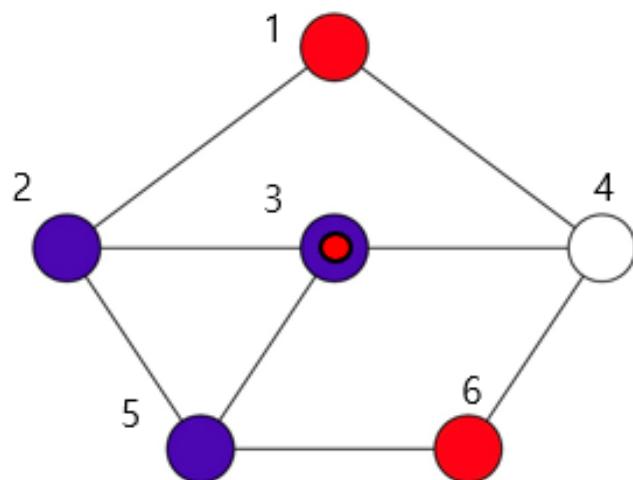
Reduction

- ❖ Reductions for decision problems
 - The possibility of a reduction from X to Y is expressed as $X \leq Y$
 - If $X \leq Y$, solving X can never be more difficult than solving Y
 - Why?
 - In other words, solving Y is at least as difficult as solving X
- Example)
 - Problem X: Determining the existence of a LIS of length k or more
 - Problem Y: Determining whether the length of the longest path in a DAG is at least k
 - $X \leq Y$, so solving Y is at least as difficult as solving X
 - $Y \leq X$?

Reduction

❖ Independent sets and cliques

- For Graph $G = (V, E)$
 - A vertex set V' is an independent set in G if all vertices in V' that belong to V are not adjacent to each other
 - A vertex set V' is a clique in G if all vertices in V' that belong to V are adjacent to each other



Independent Set = {1, 3, 6}
Clique = {2, 3, 5}

Reduction

❖ Independent sets and cliques

- Independent set problem
 - Instance: Graph G , non-negative integer k
 - Goal: determining whether G has an independent set of size k or larger
- Clique problem
 - Instance: Graph G , non-negative integer k
 - Goal: determining whether G has a clique of size k or larger

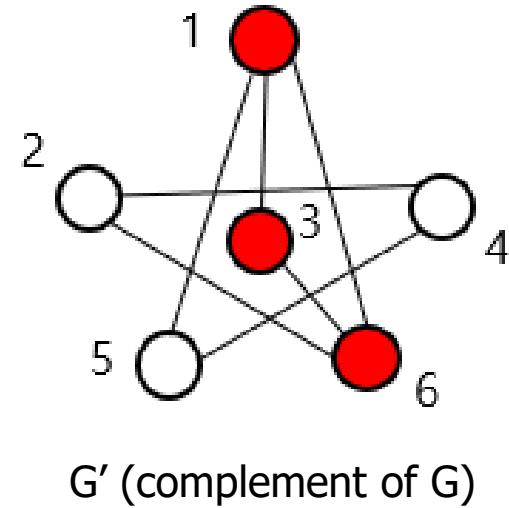
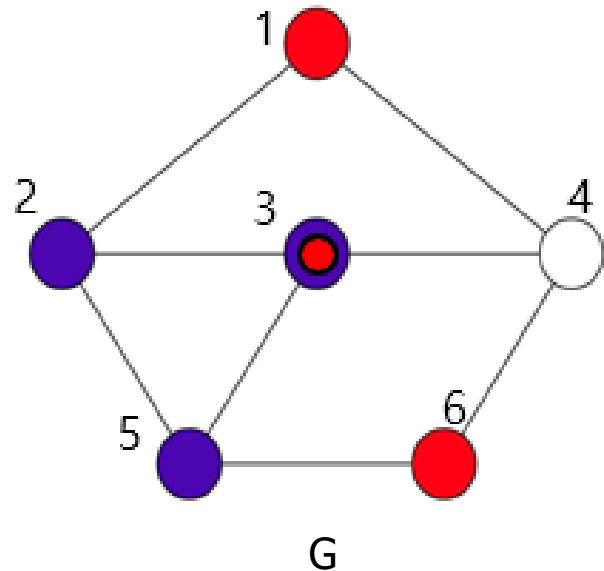
Reduction

❖ Independent sets and cliques

- Theorem: $(\text{independent set}) \leq (\text{clique})$ ($\text{independent set} \rightarrow \text{clique}$)
- Proof:
 - Let's transform an instance of independent set (G, k) into an instance of clique (G', k)
 - G' is the complement of G
 - G' has the same vertex set as G
 - $E' = \forall(u, v) - E \text{ s.t., } u, v \in G$
 - Claim: V is an independent set in $G \leftrightarrow V$ is a clique in G'
 - Since the claim is true, $(\text{independent set}) \leq (\text{clique})$

Reduction

- ❖ Independent sets and cliques



Reduction

❖ Independent sets and cliques

- $(\text{independent set}) \leq (\text{clique})$ means that the independent set problem is **reducible to** the clique problem
- If there is an algorithm to solve the clique problem, then there also exists an algorithm to solve the independent set problem
- The clique problem is **at least as difficult as** the independent set problem

Reduction

❖ Polynomial-time reductions

- When an algorithm operates in **polynomial time** (i.e., $O(\text{poly}(|\text{instance size}|))$), it is typically referred to as an **efficient algorithm**
- To find an efficient algorithm for solving a problem, one typically considers polynomial-time reductions, which operate in polynomial time
- These are referred to as **polynomial-time reductions**
- $X \leq_P Y$

Reduction

❖ Polynomial-time reductions

- If there exists a polynomial-time reduction from decision problem X to Y ($X \leq_P Y$), and there exists an efficient algorithm to solve Y (A_Y operates in polynomial time), then there also exists an efficient algorithm, A_X , to solve X
- If G' can be created in polynomial time, (independent set) \leq_P (clique)
- If there does not exist an efficient algorithm to solve the independent set problem, then there also does not exist an efficient algorithm to solve the clique problem

Reduction

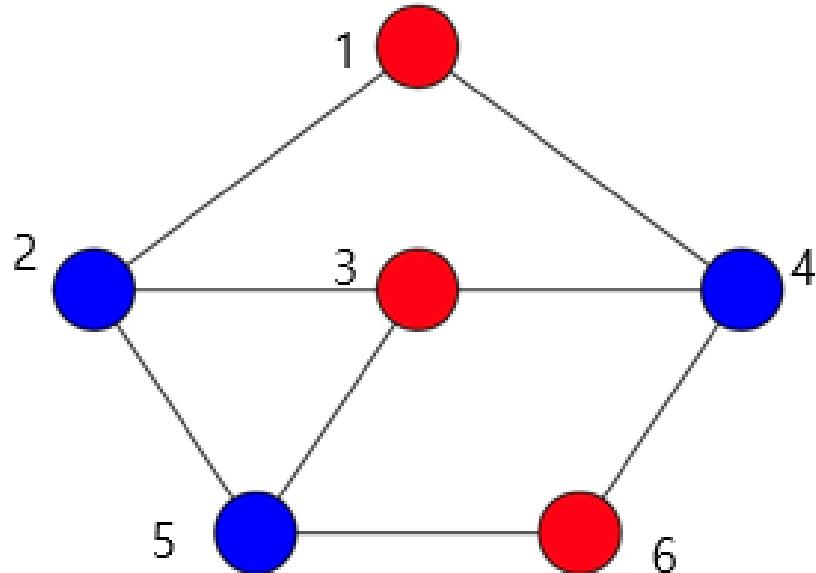
❖ Polynomial-time reductions

- A polynomial-time reduction from decision problem X to Y refers to an algorithm R that satisfies all of the following properties:
 - For any instance I_X of X, R returns an instance I_Y of Y
 - $|I_Y|$ (the size of the instance of Y) = $\text{poly}(|I_X|)$
 - R operates in polynomial time with respect to $|I_X|$
 - I_X is an instance of problem X that yields a YES answer, if and only if I_Y is an instance of problem Y that yields a YES answer
- A polynomial-time reduction defined as above is called a **Karp reduction**
- As a method for defining polynomial-time reductions for **problems other than decision problems**, there is **Cook reduction**

Reduction

❖ Vertex cover problem

- For Graph $G = (V, E)$,
- A vertex set $S \subset V$ is called a vertex cover of V if it includes at least one endpoint of all edges in G



Vertex cover of G : $\{1, 2, 3, 4, 5, 6\}$

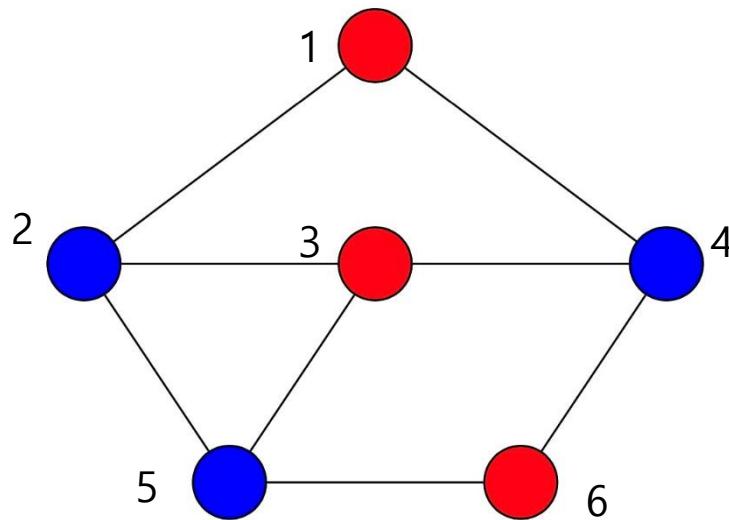
Vertex cover of G : $\{2, 5, 4\}$

$\{1, 3, 6\}$ is not vertex cover in G
→ it does not include the endpoint
of edge(2, 5)

Reduction

❖ Vertex cover problem

- Instance: Graph G, non-negative integer k
- Goal: determining whether G has a vertex cover of size at most k



Reduction

❖ Vertex cover problem

- Claim:
 - For Graph $G = (V, E)$,
 - (i) S is an independent set in $G \leftrightarrow$ (ii) $(V - S)$ is a vertex cover in G
- Proof:
 - ((i) \rightarrow (ii)) Assume that S is an independent set in G . In that case, for any edge(u, v) in G , it must be the case that either $u \notin S$ or $v \notin S$. Therefore, it follows that either $u \in (V-S)$ or $v \in (V-S)$ for any edge(u, v) in G . Hence, $(V-S)$ becomes a vertex cover in G
 - ((i) \leftarrow (ii)) Assume that $(V-S)$ is a vertex cover in G . In that case, for any two vertices u and v in S , (u, v) cannot be an edge. Therefore, S is an independent set in G

Reduction

❖ Vertex cover problem

- $(\text{independent set}) \leq_P (\text{vertex cover})?$
 - The fact that a graph G with n vertices has an independent set of size at least k is equivalent to G having a vertex cover of size at most $n-k$
 - Therefore, the answer for the instance (G, k) of the independent set problem is the same as the answer for the instance $(G, n-k)$ of the vertex cover problem
 - Given instance (G, k) , constructing $(G, n-k)$ can be done in $O(1)$
 - Therefore, we can prove that $(\text{independent set}) \leq_P (\text{vertex cover})$
 - If there does not exist an efficient algorithm to solve the independent set problem, then there also does not exist an efficient algorithm to solve the vertex cover problem

Reduction

❖ Propositional formulas

▪ Definition

- Let's consider a set of Boolean variables $\{x_1, x_2, x_3, \dots\}$
- Literal: Boolean variable x_i or the negation of x_i ($\neg x_i$)
- Clause: the disjunction of literals (e.g., $x_1 \vee x_2 \vee \neg x_3$)
- Conjunctive normal form (CNF) formula: a propositional formula expressed as the conjunction of clauses
(e.g., $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee x_3) \wedge x_5$)
- If each clause of the CNF formula F is composed of exactly k literals, then F is called a k -CNF formula
(e.g., $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_5)$ is 3-CNF formula,
 $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3)$ is not 3-CNF formula)

Reduction

❖ SAT (satisfiability) problem

- The problem of determining whether, given a CNF formula F , it is possible to assign true/false values to each variable in F such that F evaluates to true
- Example)
 - For a given $F = (x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$, if we assign true to x_1, x_2, \dots, x_5 , then the value of F becomes true
 - Therefore, F is satisfiable

Reduction

❖ SAT (satisfiability) problem

- Example)

- $F = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_2)$

x_1	x_2	$x_1 \vee \neg x_2$	$\neg x_1 \vee x_2$	$\neg x_1 \vee \neg x_2$	$x_1 \vee x_2$	T/F
T	T	T	T	F	T	F
T	F	T	F	T	T	F
F	T	F	T	T	T	F
F	F	T	T	T	F	F

→ F is not satisfiable

- The satisfiability problem for a 3-CNF formula F is called 3-SAT

Homework

- ❖ $(k\text{-SAT}) \leq_P (\text{Independent set})$?
 - Instance of $(k\text{-SAT})$ problem
 - $F = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$
 - $k = 3$
 - Task
 - Design a polynomial-time reduction algorithm R
 - $I_{k\text{-SAT}}(F, k) \rightarrow I_{\text{Independent}}(G, k)$
 - Design an efficient algorithm for independent set problem
 - Prove
 - G has an independent set of size $k \leftrightarrow F$ is satisfiable
 - Due date: 20 Dec. 2024 11:59PM

Part 2

COMPUTATIONAL INTERACTABILITY

Computational Interactability

- ❖ Polynomial time (P)
 - It refers to the set of all (decision) problems for which there exists an algorithm with polynomial running time

- ❖ Example of decision problems in P
 - Is there a pair of vertices u and v in a graph with n vertices such that the length of the shortest path between them is at least k ?
 - Is there a graph with n vertices and m edges where there are at least k biconnected components?
 - Is the length of the LIS in a sequence of length n at least k ?

Computational Interactability

- ❖ Polynomial time (P) (cont'd)
 - Problems that are not known to be in P
 - Independent set
 - Clique
 - Vertex cover
 - 3-SAT
 - Etc
 - There exist countless such problems
 - What common characteristics do such problems have?
 - Is there a formal way to define such problems?

Computational Interactability

❖ Polynomial time (P) (cont'd)

- Common characteristics of the aforementioned problems
 - For an instance I_X belonging to problem X, there exists a proof/certificate/solution of length $\text{poly}(|I_X|)$ that can verify whether I_X is actually a YES instance
- Example) 2-SAT
 - $F = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \rightarrow I_X$

x_1	x_2	$x_1 \vee \neg x_2$	$\neg x_1 \vee x_2$	$x_1 \vee x_2$	T/F
T	T	T	T	T	T
T	F	T	F	T	F
F	T	F	T	T	F
F	F	T	T	F	F

Proving that I_X is a YES instance (an instance belonging to X) is done through the proof of I_X

Computational Interactability

❖ Polynomial time (P) (cont'd)

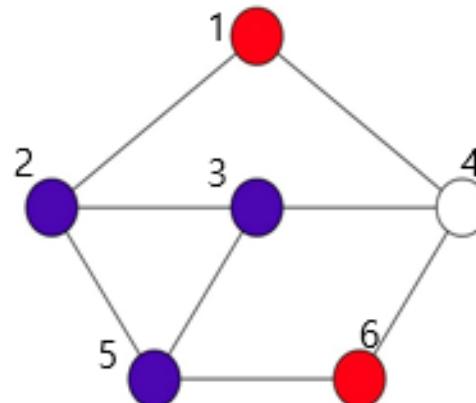
- Certifier
 - Def) Some problem X
 - i) For any YES instance s in X , there exists a t such that $C(s, t) = \text{YES}$
 - ii) Conversely, if $C(s, t) = \text{YES}$ for some s and t , then s is in X
 - An algorithm C that satisfies the above conditions is called the certifier for problem X
 - The t is referred to as the certificate (or proof) for s
- Efficient certifier
 - i) $s \in X \leftrightarrow |t| \leq \text{poly}(|s|)$ and $C(s, t) = \text{YES}$
 - ii) C runs in polynomial time
 - C satisfying both conditions is called an efficient certifier

Computational Interactability

❖ Polynomial time (P) (cont'd)

- Example 1) independent set

- Problem: does the graph $G=(V, E)$ contain an independent set of size at least k ?
- Certificate: some vertex set $S \subset V$
- Certifier
 - It first checks if $|S|$ is at least k and then verifies that any two vertices in S are not adjacent to each other

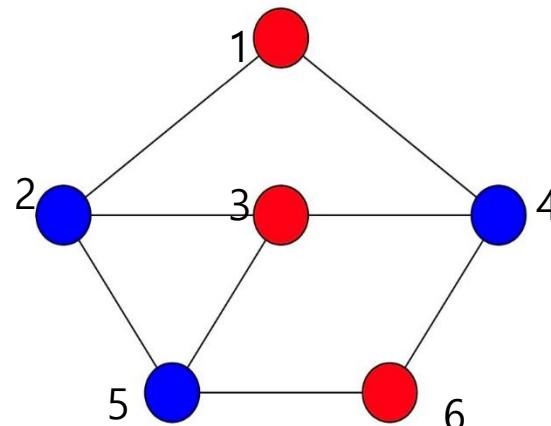


Computational Interactability

❖ Polynomial time (P) (cont'd)

- Example 2) vertex cover

- Problem: does the graph $G=(V, E)$ contain a vertex cover of size at most k ?
- Certificate: some vertex set $S \subset V$
- Certifier
 - It first checks if $|S|$ is at most k and then verifies that all edges in G contain at least one vertex from S



Computational Interactability

❖ Polynomial time (P) (cont'd)

- Example 3) 3-SAT
 - Problem: Is the 3-CNF formula F satisfiable?
 - Certificate: Assigning 0/1 to each variable of F
 - Certifier
 - Determining whether F is true for that assignment
 - If all clauses of F are true, then YES

$$F = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$

Computational Interactability

- ❖ Nondeterministic polynomial time (NP)
 - Set of all problems that have efficient certifiers
 - NP is by no means an abbreviation for NOT P
 - Example)
 - Independent set, vertex cover, and 3-SAT all belong to NP
 - Formal definition
 - If a problem X belongs to NP, then for every instance s, there exist a polynomial p, q, and certifier C
 - If s is a YES instance (i.e., $s \in X$), there exists a certificate t with $|t|=p(|s|)$ such that $C(s, t) = \text{YES}$
 - If s is a NO instance (i.e., $s \notin X$), then it answers NO for all instances t, i.e., $C(s, t) = \text{NO}$
 - $C(s, t)$ operates in $q(|s| + |t|)$ time

Computational Interactability

- ❖ Nondeterministic polynomial time (NP) (cont'd)
 - Example) Proving that independent set is in NP
 - Let's define a certificate t and a certifier $C(s, t)$ for an instance s (G and k) of the independent set problem as described on the previous slide
 - Then, the certificate is smaller than $|s|$, so it can be represented by a polynomial in terms of $|s|$
 - If s is a YES instance, the existence of a certificate t such that $C(s, t)$ holds is trivial by the definition of the problem
 - Since $C(s, t)$ can verify whether each pair (u, v) of vertices in t are adjacent in $O(1)$ time, the total time complexity is $O(|t|^2)$
 - Therefore, $C(s, t)$ is an efficient certifier
 - In other words, independent set belongs to NP

Computational Interactability

- ❖ Nondeterministic polynomial time (NP) (cont'd)
 - The definition of NP solely implies that YES instances have short proofs/certificates
 - It does not apply to NO instances

Computational Interactability

❖ Proposition: $P \subseteq NP$

- Proof
 - Suppose that there exists a problem X in P , and there is a polynomial-time algorithm A that solves X
 - Just need to prove that X has an efficient certifier
 - Assume that the certifier $C(s, t)$ is the answer obtained after executing $A(s)$
 - Case 1. $s \in X$: it outputs YES for all t (therefore, t must exist)
 - Case 2. $s \notin X$: it outputs NO for all t
 - Since C is an efficient certifier, P belongs to NP

Computational Interactability

❖ Exponential time (EXP)

- Set of problems that can be solved for an input (instance) s in $O(2^{\text{poly}(|s|)})$ time or less
- Example)
 - $O(2^n), O(3^n), O(2^{n\log n}), \dots$

Computational Interactability

❖ Proposition: $\text{NP} \subseteq \text{EXP}$

- Proof
 - To solve problem X in NP , design the following exponential algorithm
 - Algorithm
 - Perform $C(s, t)$ for all t such that $t \leq p(|s|)$, and if certifier C returns YES at least once, return YES
 - The above algorithm is a correct algorithm for solving X
 - Given that the runtime of the above algorithm is $O(q(|s| + p(|s|))2^{p(|s|)})$, by the definition of NP , q and p become polynomial
 - Therefore, X belongs to EXP

Computational Interactability

- ❖ $P \subseteq NP \subseteq EXP$
- ❖ Big question: P-NP problem
 - Is there a problem in NP that is not in P?

Computational Interactability

- ❖ Big question: P-NP problem (cont'd)
 - If $P = NP$, what happens?
 - Many difficult problems (especially optimization problems) can be solved in polynomial time
 - Current encryption systems become no longer secure
 - Internet security is also at risk
 - The realm of mathematical theorems that can be proven using computers significantly expands

Computational Interactability

- ❖ Big question: P-NP problem (cont'd)
 - In the event of solving this problem,
 - One million dollars (one of the Millennium problems)
 - Probably Fields medal
 - Probable Turing award
 - Many other honors and distinctions
 - Most people believe that $P \neq NP$
 - But no one knows for sure yet

Computational Interactability

- ❖ Hardest problems
 - How can we define the most difficult problem?
 - The problem must belong to NP
 - It should be at least as hard as any other problem in NP
 - Reduction is a method to compare the difficulty between two problems
 - Let's define the most challenging problem using reduction

Computational Interactability

❖ NP-Complete and NP-Hard problems

- Definition of NP-Complete) Problem X is called NP-Complete if it satisfies the following conditions:
 - $X \in NP$
 - For any $Y \in NP$, $Y \leq_P X$ holds
- Definition of NP-Hard) Problem X is called NP-Hard if it satisfies the following conditions:
 - For any $Y \in NP$, $Y \leq_P X$ holds
- NP-Hard includes NP-Complete
 - E.g., Halting problem is NP-Hard but not NP-Complete

Computational Interactability

- ❖ Proposition: if X is an NP-Complete problem, then the following holds:
 - $P = NP \leftrightarrow X$ can be solved in polynomial time
 - Proof
 - (\leftarrow) Assume that X can be solved in polynomial time. Then, for any $Y \in NP$, $Y \leq_P X$ holds. Therefore, there exists a polynomial-time algorithm to solve Y , and as a result, Y also belongs to P
 - (\rightarrow) If $NP = P$, then X also belongs to P , since X is in NP . In other words, there exists a polynomial-time algorithm to solve X

Computational Interactability

- ❖ If there exists a polynomial-time algorithm to solve the NP-Complete problem X, then it can be concluded that $P = NP$
- ❖ Most people believe that $P \neq NP$, and thus, they think that there is no polynomial-time algorithm to solve problem X
- ❖ How many NP-Complete problems exist?
 - Many problems are indeed NP-Complete

Computational Interactability

- ❖ Typically, steps to prove that a problem X is NP-Complete are as follows:
 - Step 1: Prove that X belongs to NP (very important)
 - Step 2: Prove $Y \leq_P X$ for some NP-Complete problem Y
 - When proving Step 2, for any NP problem Z , if $Z \leq_P Y \leq_P X$ holds, then we can conclude that $Z \leq_P X$ (transitivity)

Computational Interactability

- ❖ $(\text{Independent set}) \leq_P (\text{Vertex cover}) \leq_P (\text{Clique})$
 - Independent set, vertex cover, and clique problems can all be easily shown to belong to NP
 - Thus, those problems are NP-Complete according to the previous proving steps

Computational Interactability

❖ P

- Set of problems that have polynomial-time algorithms

❖ NP

- Set of problems that have efficient certifiers

❖ NP-Complete

- Set of the most difficult problems within NP

❖ Problems within NP-Complete can be solved in polynomial time if $P=NP$, but this has not been proven on either side so far

Summary

❖ Reduction

- Problem, Problem instance
- Decision problem
- Polynomial-time algorithm
- Polynomial-time reduction

❖ Computational intractability

- Polynomial time (P)
 - Certifier
- Nondeterministic polynomial time (NP)
 - Efficient certifier
- NP-Complete
- NP-Hard

Questions?

SEE YOU NEXT TIME!