CS3230 Tutorial 12

NP-Completeness (The Last Tutorial)

(AY 25/26 Semester 2)

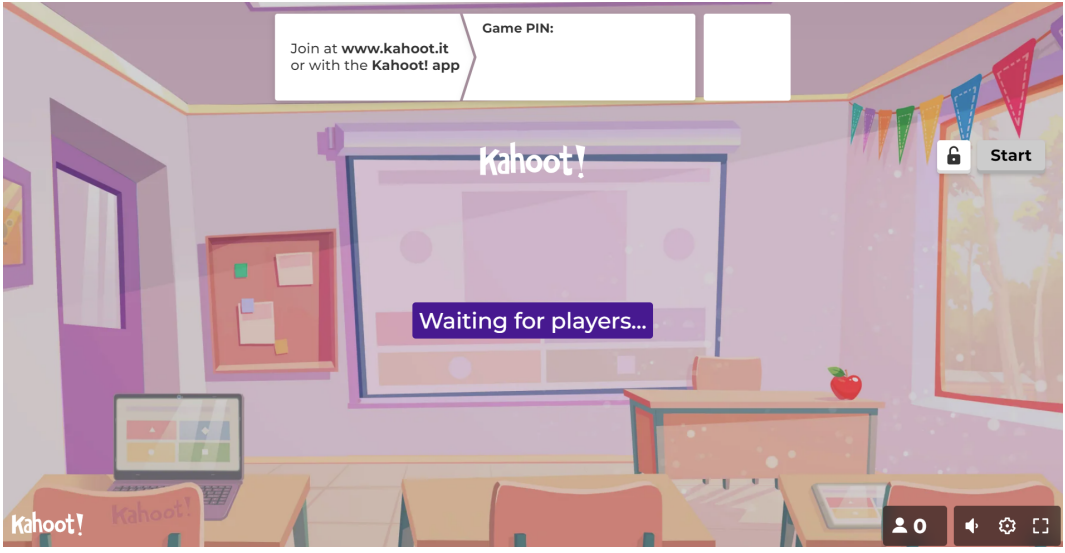
April 13, 2026

(Prepared by Benson)

Reference:
Tutorial Slides by Arpan and Edbert (AY 22/23 Sem 2)

Contents

Quick Quiz on Reductions



Quick Quiz on Reductions

1. Which of the following best describes the input size of the knapsack problem? N is the number of items and W is the target weight.

▲ $\Theta(N + W)$

♦ $\Theta(N \log \max\{V_i, W_i\} + \log W)$ 

● $\Theta(N \log \max\{V_i, W_i\} + W)$

Explanation: W occupies $\log W$ bits. The N weight-value pairs occupies $N \times \log \max\{V_i, W_i\}$ bits.

Quick Quiz on Reductions

2. We can use a blackbox of the KNAPSACK problem to solve the 3-SAT problem in polynomial time. This implies...

▲ KNAPSACK $\leq_p$ 3-SAT

♦ 3-SAT $\leq_p$ KNAPSACK 

● There is nothing we can infer.

Explanation: We are reducing 3-SAT to KNAPSACK.

Quick Quiz on Reductions

3. It is known that $B \leq_p A$, and we can solve B in polynomial time. This implies...

- ▲ A can be solved in polynomial time.
- ♦ A cannot be solved in polynomial time.
- ● There is nothing we can infer. 

Explanation: It's possible that we solved B in polynomial time using other methods, without involving A.

Quick Quiz on Reductions

4. It is known that $B \leq_p A$, and we can solve A in $\Theta(n)$ time, where n is the size of the input. This implies...

- ▲ We can solve B in $\Theta(n)$ time.
- ♦ We can solve B in polynomial time. ✓
- ● There is nothing we can infer.

Explanation: We can apply the reduction to solve B in polynomial time. However, the reduction may take more than $\Theta(n)$ time, as long as it is still polynomial in n.

Quick Quiz on Reductions

5. Consider the problems Is-Even and Is-Power-Of-2 with an input integer n. Which of the following is correct?

- ▲ Only Is-Even $\leq_p$ Is-Power-Of-2
- ♦ Is-Even $\leq_p$ Is-Power-Of-2 and Is-Power-Of-2 $\leq_p$ Is-Even 
- ● None of the other choices.

Explanation: Both Is-Even and Is-Power-Of-2 can be solved in $O(\log n)$ time. Therefore, we can construct a reduction by ignoring the blackbox and solving the problem directly first, then feed either 2 or 3 to the blackbox (Yes/No).

Quick Quiz on Reductions

6. If KNAPSACK cannot be solved in polynomial time, which of the following implies 3-SAT cannot be solved in polynomial time?

- ▲ 3-SAT $\leq_p$ KNAPSACK
- ♦ KNAPSACK $\leq_p$ 3-SAT ✓
- ● None of the other choices.

Explanation: Proof by contradiction. If 3-SAT can be solved in polynomial time, then by KNAPSACK $\leq_p$ 3-SAT, KNAPSACK can be solved in polynomial time, a contradiction.

Recap: The Complexity Classes

P: Set of decision problems that can be **solved in polynomial time**.

- Polynomial in the size of input.

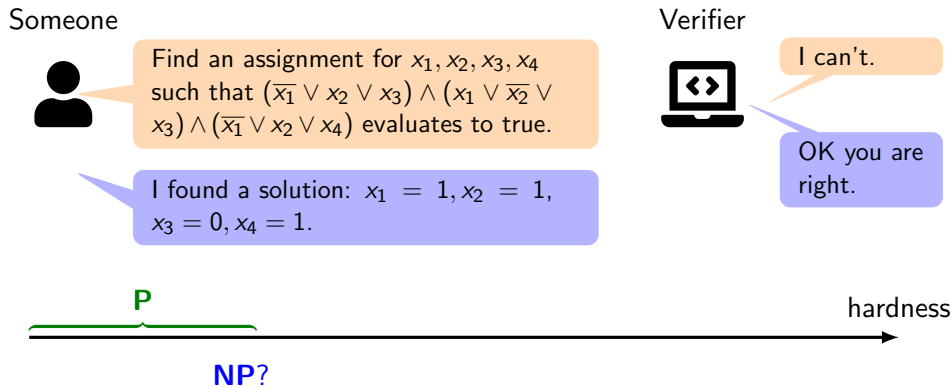
How to show that a problem is in P?

Come up with an algorithm and show that it runs in polynomial time.

Recap: The Complexity Classes

NP: Set of decision problems whose **YES-instance can be verified in polynomial time** given a **certificate**.

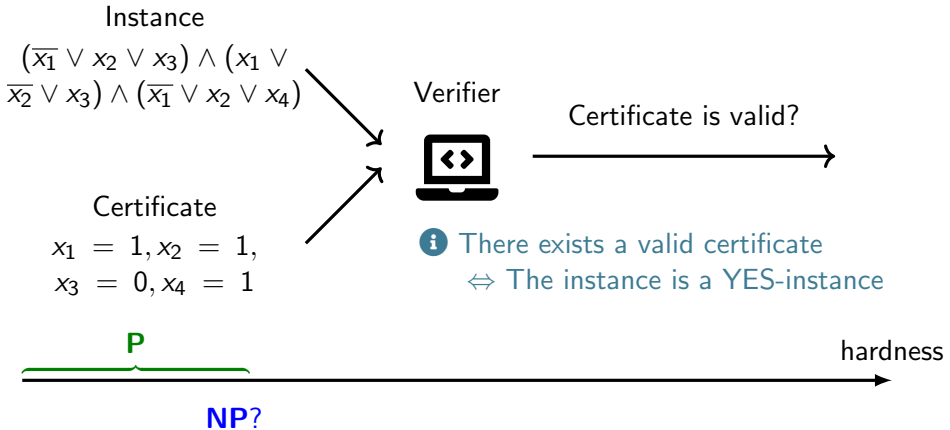
NP: "Non-deterministic Polynomial"



Recap: The Complexity Classes

NP: Set of decision problems whose **YES-instance can be verified in polynomial time** given a **certificate**.

- NP: "Non-deterministic Polynomial"

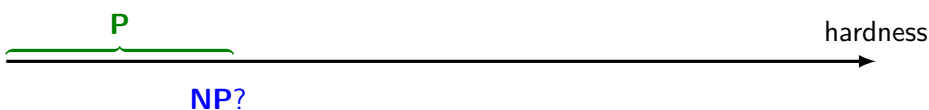


Recap: The Complexity Classes

NP: Set of decision problems whose **YES-instance can be verified in polynomial time** given a **certificate**.

How to show that a problem is in NP?

1. Come up with a **polynomially-sized certificate** format.
   - Usually, it is the proposed solution itself.
2. Design a **polynomial-time algorithm that verifies the certificate**.
   - You have to confirm that a given certificate is not bluffing!



Recap: The Complexity Classes

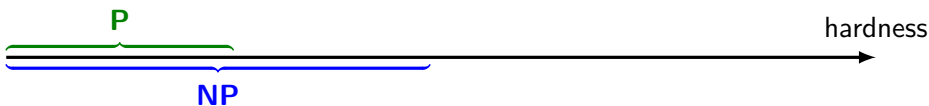
CS1231/S Refresher: To prove "$A \subseteq B$", show that $x \in A \Rightarrow x \in B$.

What is the relation between **P** and **NP**? Answer: **P** $\subseteq$ **NP**.

Proof. Consider any problem in P.

1. Come up with a **polynomially-sized certificate** format.
   Certificate: "Trust me bro"
2. Design a **polynomial-time algorithm that verifies the certificate**.
   The verifier can solve the problem on its own and check whether the answer matches. The verifier runs in polynomial time since the problem is in P.

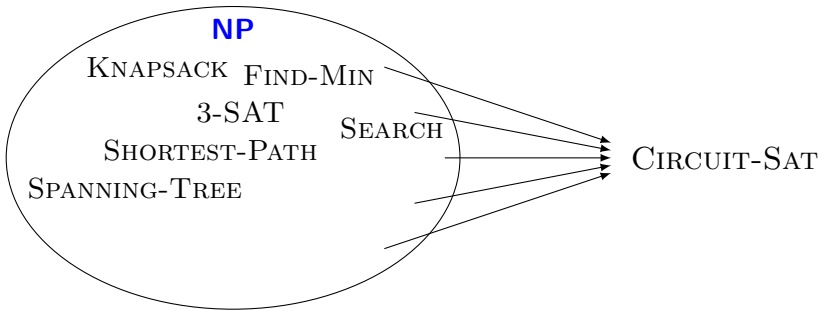
Thus the problem is in NP.

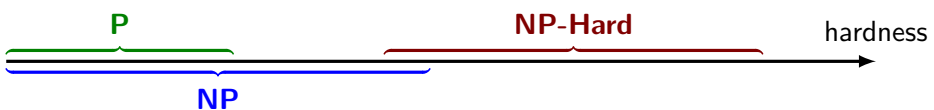


Recap: The Complexity Classes

NP-Hard: Set of decision problems that are "at least as hard" as all problems in **NP**.

X is NP-hard $\Leftrightarrow$ For all problems $Y \in$ NP, we have $Y \leq_p X$.





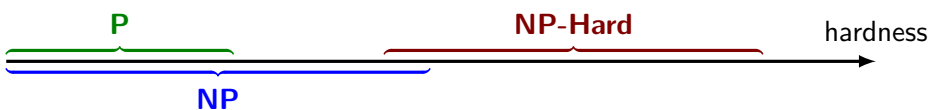
Recap: The Complexity Classes

NP-Hard: Set of decision problems that are "at least as hard" as all problems in **NP**.

ℹ X is NP-hard $\Leftrightarrow$ For all problems $Y \in$ NP, we have $Y \leq_p X$.

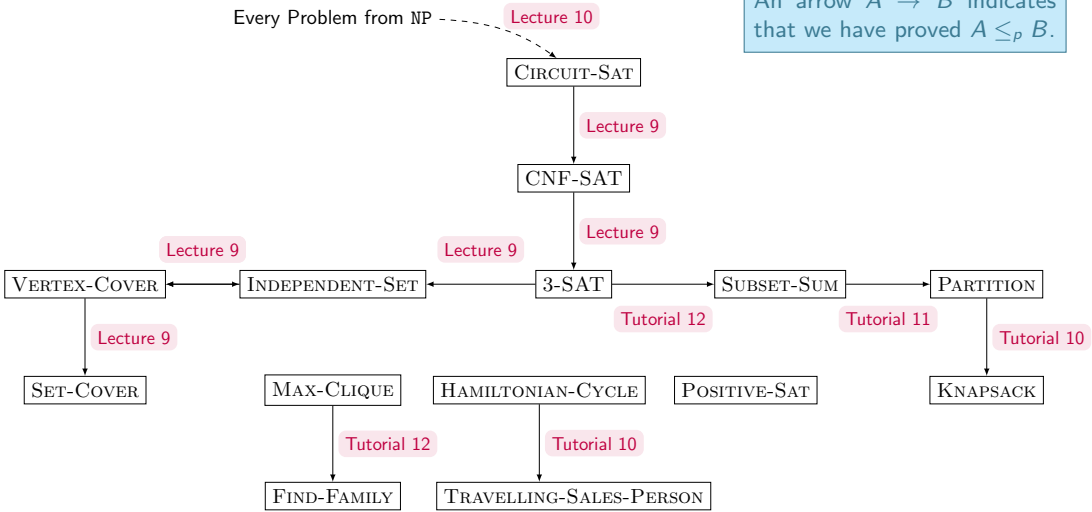
How to show that a problem is in NP-Hard? Hard to show by definition ☹

1. Find another NP-Hard problem Z.
2. Show that X is "not easier" than Z, i.e. $Z \leq_p X$.
   ℹ For all $Y \in$ NP, we already have $Y \leq_p Z$. Thus $Y \leq_p X$ by transitivity.



Recap: The Complexity Classes

An arrow $A \to B$ indicates that we have proved $A \leq_p B$.



Recap: The Complexity Classes

NP-Complete: Set of decision problems that are both in **NP-hard** and **NP**.

Intuition:

- All problems in **NP-Complete** are reducible to and from each other.
- Hardest problems in the **NP** class.

How to show that a problem is in NP-Complete?

1. First, show that the problem is in NP.
2. Next, show that the problem is in NP-Hard.

Recap: The Complexity Classes

Is P = NP? We don't know ☹

- If P ≠ NP:

- If P = NP:

Q1. P = NP?

Which of the following imply P = NP?

A. There is a problem in P that is also in NP-Complete.

B. There is a problem in P that is also in NP. Any problem in P.

C. There is a problem in NP that is also in NP-Hard. Any problem in NP-Complete.

Q1. P = NP?

There is a problem in P that also in NP-Complete. $\Rightarrow$ P = NP.

Intuition:

Proof:

- Let X be the problem that is both in P and NP-Complete.
- X is NP-Hard. (since $X \in$ NP-Complete)
- For all problems $Y \in$ NP, $Y \leq_p X$. (by definition of NP-Hard)
- We can solve X in polynomial time. (since $X \in$ P)
  $\Rightarrow$ We can solve Y in polynomial time.

Q2. SUBSET-SUM is NP-Complete

Show that SUBSET-SUM is NP-Complete.

- **SUBSET-SUM**: Given a set of n non-negative integers $S = \{w_1, \ldots, w_n\}$ and a target W, decide whether there exists a subset $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} w_i = W$. (In other words, decide whether there exists a subset of S with sum W.)

YES-instance	NO-instance
$W = 0$ 	$W = 4$ 

Q2. SUBSET-SUM is NP-Complete

SUBSET-SUM is in **NP**.

1. Come up with a **polynomially-sized certificate** format.
   Certificate: The set $I \subseteq \{1, 2, \ldots, n\}$
   Size of Certificate: $O(n \log n)$, Size of input: $n \log S_{\max} + \log W$
   Certificate is polynomially sized since $O(n \log n) = O(poly(n)) = O(poly(|I|))$.
2. Design a **polynomial-time algorithm that verifies the certificate**.
   $A((S; W); I)$ verifies whether $\sum_{i \in I} w_i = W$.
   Runtime of verifier algorithm: $O(n \log W) = O(poly(|I|))$

Q2. SUBSET-SUM is NP-Complete

SUBSET-SUM is in **NP-Hard**.

We can reduce 3-SAT to SUBSET-SUM.

- **3-SAT**: Given a CNF formula Φ where each clause has exactly 3 literals corresponding to different variables, does it have a satisfying truth assignment?

YES-instance	NO-instance
$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$ Solution: $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3)$ $\wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$ $\wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

Q2. SUBSET-SUM is NP-Complete

SUBSET-SUM is in **NP-Hard**. (Goal: Prove that 3-SAT $\leq_p$ SUBSET-SUM.)

Intuition: We need to somehow transform the

- 3-SAT constraints (a variable and its negation cannot be both true, each clause must be true)
- to SUBSET-SUM constraints (numbers in chosen subset must add to the target).

We can use the sum of one digit to represent a 3-SAT constraint!

	1	0	0	0	1	1
+	0	1	1	1	0	0
	1	1	1	1	1	1

sum down vertically ⇒ one constraint

Q2. SUBSET-SUM is NP-Complete

SUBSET-SUM is in **NP-Hard**. (Goal: Prove that 3-SAT $\leq_p$ SUBSET-SUM.)

Example: $(\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3)$.

Exactly one of x_i or $\overline{x_i}$ — All clauses should be satisfied

Element	x_1	x_2	x_3	C_1	C_2	C_3	C_4
$x_1 = 1$	1	0	0	0	0	0	1
$x_1 = 0$	1	0	0	1	1	1	0
$x_2 = 1$	0	1	0	1	0	1	0
$x_2 = 0$	0	1	0	0	1	0	1
$x_3 = 1$	0	0	1	1	1	0	1
$x_3 = 0$	0	0	1	0	0	1	0
Sum	1	1	1	2 !?	3 !?	1 !?	2 !?

SUBSET-SUM requires a fixed target W

The sum must be 1
$\Rightarrow$ We did not choose both.

A clause is satisfied if the sum is 1/2/3.
$\Rightarrow$ At least one term is fulfilled.

Q2. SUBSET-SUM is NP-Complete

SUBSET-SUM is in **NP-Hard**. (Goal: Prove that 3-SAT $\leq_p$ SUBSET-SUM.)

Example: $(\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3)$.

Exactly one of x_i or $\overline{x_i}$ — All clauses should be satisfied

Element	x_1	x_2	x_3	C_1	C_2	C_3	C_4
Dummy for C_1	0	0	0	1	0	0	0
Dummy for C_1	0	0	0	1	0	0	0
Dummy for C_2	0	0	0	0	1	0	0
Dummy for C_2	0	0	0	0	1	0	0
Dummy for C_3	0	0	0	0	0	1	0
Dummy for C_3	0	0	0	0	0	1	0
Dummy for C_4	0	0	0	0	0	0	1
Dummy for C_4	0	0	0	0	0	0	1

Q2. SUBSET-SUM is NP-Complete

SUBSET-SUM is in **NP-Hard**. (Goal: Prove that 3-SAT $\leq_p$ SUBSET-SUM.)

Example: $(\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3)$.

Element	x_1	x_2	x_3	C_1	C_2	C_3	C_4
	Exactly one of x_i or $\overline{x_i}$			All clauses should be satisfied			
$x_1 = 0$	1	0	0	1	1	1	0
$x_2 = 0$	0	1	0	0	1	0	1
$x_3 = 1$	0	0	1	1	1	0	1
Dummy for C_1	0	0	0	1	0	0	0
Dummy for C_3	0	0	0	0	0	1	0
Dummy for C_3	0	0	0	0	0	1	0
Dummy for C_4	0	0	0	0	0	0	1
Sum	1	1	1	3	3	3	3

Q2. SUBSET-SUM is NP-Complete

SUBSET-SUM is in **NP-Hard**. (Goal: Prove that 3-SAT $\leq_p$ SUBSET-SUM.)

Reduction:

- Construct $2n + 2m$ numbers for a given 3-SAT instance, each with $n + m$ digits (from the previous slides).
  - First $2n$ numbers: "$x_i = 0$" and "$x_i = 1$" – the i-th digit, as well as the clauses with the term x_i or $\overline{x_i}$, are set to 1.
  - Next $2m$ numbers: "dummy for C_i" – only the $(i + n)$-th digit is set to 1.
- We also construct a target W (of $n + m$ digits) by setting
  - The first n digits to 1.
  - The next m digits to 3.
- Why does that take polynomial time to the size of input?
  It is $O((n + m)^2) = O(poly(|I|))$!

Q2. SUBSET-SUM is NP-Complete

SUBSET-SUM is in **NP-Hard**. (Goal: Prove that 3-SAT $\leq_p$ SUBSET-SUM.)

Theorem. ϕ is a YES-instance of 3-SAT if and only if (S, W) is a YES-instance of SUBSET-SUM.

Proof:

- ($\Rightarrow$) Given that ϕ is a YES-instance of 3-SAT.
  - Choose the item x_i if x_i is true, choose the item $\overline{x_i}$ otherwise.
  - The last m digits of the sum will either be 1/2/3 since the assignment satisfies the 3-SAT instance. Take the suitable number of dummy values to make it 3.
  - Since there is no possibility of a carry-over when adding, the sum must be equal to W.

Q2. SUBSET-SUM is NP-Complete

SUBSET-SUM is in **NP-Hard**. (Goal: Prove that 3-SAT $\leq_p$ SUBSET-SUM.)

Theorem. ϕ is a YES-instance of 3-SAT if and only if (S, W) is a YES-instance of SUBSET-SUM.

Proof:

- ($\Leftarrow$) Given that (S, W) is a YES-instance of SUBSET-SUM.
  - The first n digits are 1 $\Rightarrow$ Exactly one of x_i and $\overline{x_i}$ are taken. Assign true to x_i if x_i is taken and assign false otherwise.
  - The next m digits are 3 (there are at most 2 from "dummy values") $\Rightarrow$ For each digit, there is at least 1 contributed from the x_i or $\overline{x_i}$ terms. $\Rightarrow$ Each clause is satisfied.

Q2. SUBSET-SUM is NP-Complete

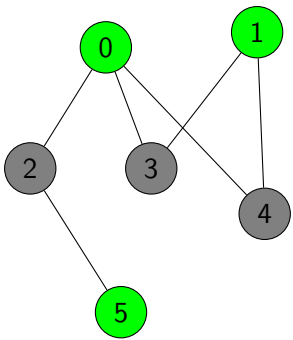
Since SUBSET-SUM is both in **NP** and **NP-Hard**, SUBSET-SUM is in **NP-Complete**.

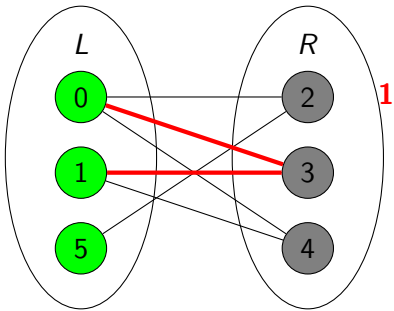
Q3. Find Family

An undirected bipartite graph $G = (L \cup R, E)$ has two disjoint vertex sets L and R where each edge has one endpoint in L and another in R.

We call a pair $u, v \in L$ **siblings** if there exists a vertex in $r \in R$ such that both the edges (u, r) and (r, v) are present.

A subset $F \subseteq L$ is said to be a **family** if for all distinct $u, v \in F$, u and v are siblings.





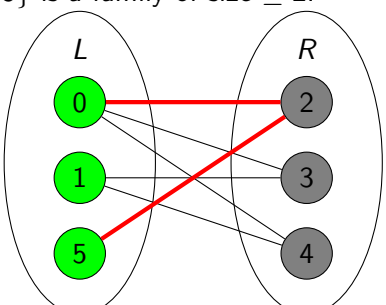
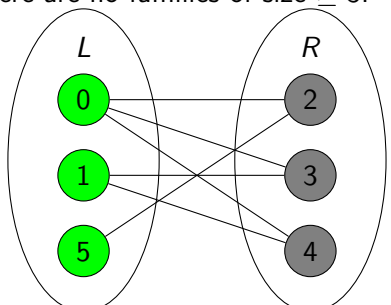
0 and 1 are siblings

1 and 5 are not siblings

Q3. Find Family

Show that FIND-FAMILY is in NP-Complete.

- **FIND-FAMILY**: Does there exist a family of size $\geq k$?

YES-instance	NO-instance
$\{0, 5\}$ is a family of size ≥ 2.	There are no families of size ≥ 3.
	

Q3. Find Family

FIND-FAMILY is in **NP**.

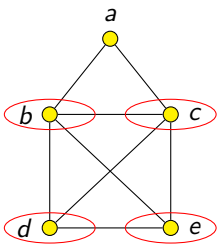
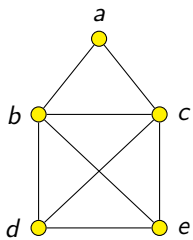
1. Come up with a **polynomially-sized certificate** format.
   Certificate: The family itself.
   Size of Certificate: $O(n \log n)$, Size of input: $\geq n \log n$
2. Design a **polynomial-time algorithm that verifies the certificate**.
   For each pair in the family, check if they have a common neighbour.
   Runtime of verifier algorithm: $O(|L|^2 \cdot |R|) = O(n^3)$

Q3. Find Family

FIND-FAMILY is in **NP-Hard**.

We can reduce MAX-CLIQUE to FIND-FAMILY.

- **MAX-CLIQUE**: Given an undirected graph $G = (V, E)$ and an integer k, a **clique** is a subset of the vertex set $C \subseteq V$, such that for every two vertices $u, v \in C$, $(u, v) \in E$ (in other words, the subgraph induced by C is complete). Does there exist a clique of size $\geq k$?

YES-instance	NO-instance
Here shows a clique of size ≥ 4. 	There are no cliques of size ≥ 5. 

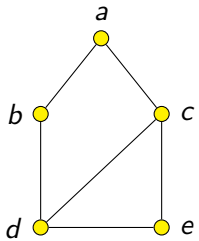
Q3. Find Family

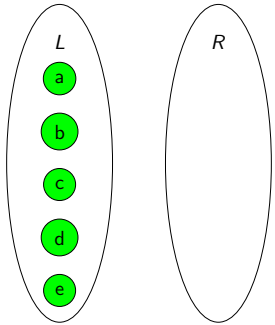
FIND-FAMILY is in **NP-Hard**.

Given an instance of MAX-CLIQUE, how will you reduce it to a FIND-FAMILY instance?

Intuition:

1. FIND-FAMILY finds the largest family in L.
   MAX-CLIQUE finds the largest clique in V. $\Rightarrow$ We should set $L = V$.





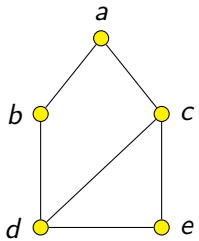
Q3. Find Family

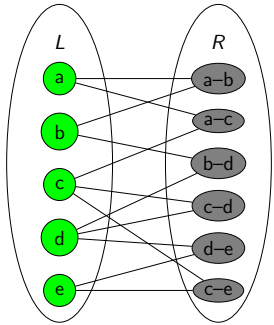
FIND-FAMILY is in **NP-Hard**.

Given an instance of MAX-CLIQUE, how will you reduce it to a FIND-FAMILY instance?

Intuition:

2. Family: Every two vertices in a family must be connected a common $r \in R$.
   Clique: Every two vertices must be connected by an edge.



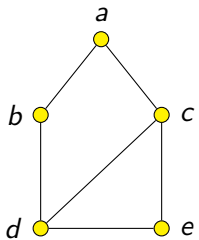


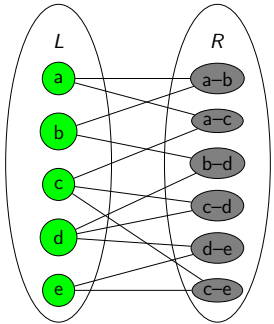
Q3. Find Family

FIND-FAMILY is in **NP-Hard**.

Reduction:

- Given an instance $G = (V, E)$ for MAX-CLIQUE.
- Construct an instance of FIND-FAMILY by letting $L = V$, $R = E$ and for each edge $(u, v) \in E$, build an edge from $u \in L$ to $(u, v) \in R$ and another edge from $v \in L$ to $(u, v) \in R$.





Q3. Find Family

FIND-FAMILY is in **NP-Hard**.

Theorem. $G = (V, E)$ is a YES-instance of MAX-CLIQUE if and only if $G' = (L \cup R, E')$ is a YES-instance of FIND-FAMILY.

Proof:

- ($\Rightarrow$) $G = (V, E)$ is a YES-instance of MAX-CLIQUE.
  - There exists a clique $C \subseteq V$ of size $\geq k$.
  - Then for every two vertices $u, v \in C$, $(u, v) \in E$.
    $\Rightarrow$ They are siblings in G' since $(u, (u, v)) \in E'$ and $(v, (u, v)) \in E'$.
  - Therefore, $C \subseteq L$ is a family of size $\geq k$.
- ($\Leftarrow$) $G' = (L \cup R, E')$ is a YES-instance of FIND-FAMILY.
  - There exists a family $F \subseteq L$ of size $\geq k$.
  - Then for every two vertices $u, v \in F$, $(u, x) \in E'$ and $(v, x) \in E'$ for some x. Since $u \neq v$, the only possibility is $x = (u, v)$. $\Rightarrow (u, v) \in E$.
  - Therefore, $F \subseteq L$ is a clique of size $\geq k$.

Q3. Find Family

Since FIND-FAMILY is both in **NP** and **NP-Hard**, FIND-FAMILY is in **NP-Complete**.

A Final Review

Design and **Analysis** of Algorithms

Design

- Divide and Conquer Week 3
- Dynamic Programming Week 6

- Greedy Algorithms Week 7

(Locally optimal choice)

Analysis

Correctness

- Proof of Correctness Week 3
- Probabilistic Analysis Week 5

Efficiency

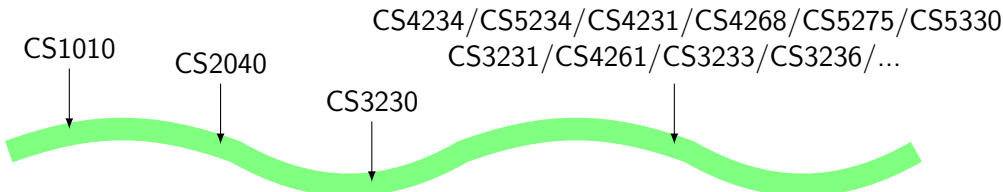
- Asymptotic Analysis Week 1
- Recurrences and Master Theorem Week 2
- Probabilistic Analysis Week 5
- Amortized Analysis Week 9

Hardness

- Decision Trees and Lower Bounds Week 4
- Reductions and NP-Completeness Week 12

A Final Review

Hope CS3230 ~~wraps up~~ sets the stage for your algorithm journey in NUS nicely!



Student Feedback Exercise:

Your Voice Matters!

Provide your feedback now >>

https://blue.nus.edu.sg/blue/

Bonus. Settling Debts

Extra Slide

This question follows from the bonus question in tutorial 11. In particular, you may assume Subset-Sum $\leq_p$ Zero-Subset-Sum in your proof.

Consider the following problem Settling-Debts:

- **Settling-Debts**: You and your friends have just returned from a beautiful vacation in the mountains of the Netherlands. You kept all the receipts from the vacation, and wrote down who paid how much for who.
  Assuming everyone has enough spare cash to transfer an arbitrary amount of money to another person, you are tasked to figure out the least number of transactions needed to settle the debts (i.e. everyone's balance is 0).

(a) Suppose there are N people in the group. Prove that all debts can always be settled within $N - 1$ transactions.

(b) Prove that the decision version of Settling-Debts is NP-Complete.

(Solutions are not provided but you're welcomed to check your ideas with me.)

Bonus. Settling Debts

Extra Slide

Example. Assume there are 4 people in the group and we have the following 3 receipts:

1. Evan paid \$1 for Cherie.
2. Cherie paid \$2 for Sam.
3. Evan paid \$1 for Sean.

Then the debts can be settled with the following 2 transactions:

- Sam pays \$2 to Evan.
- Sean pays \$1 to Cherie.