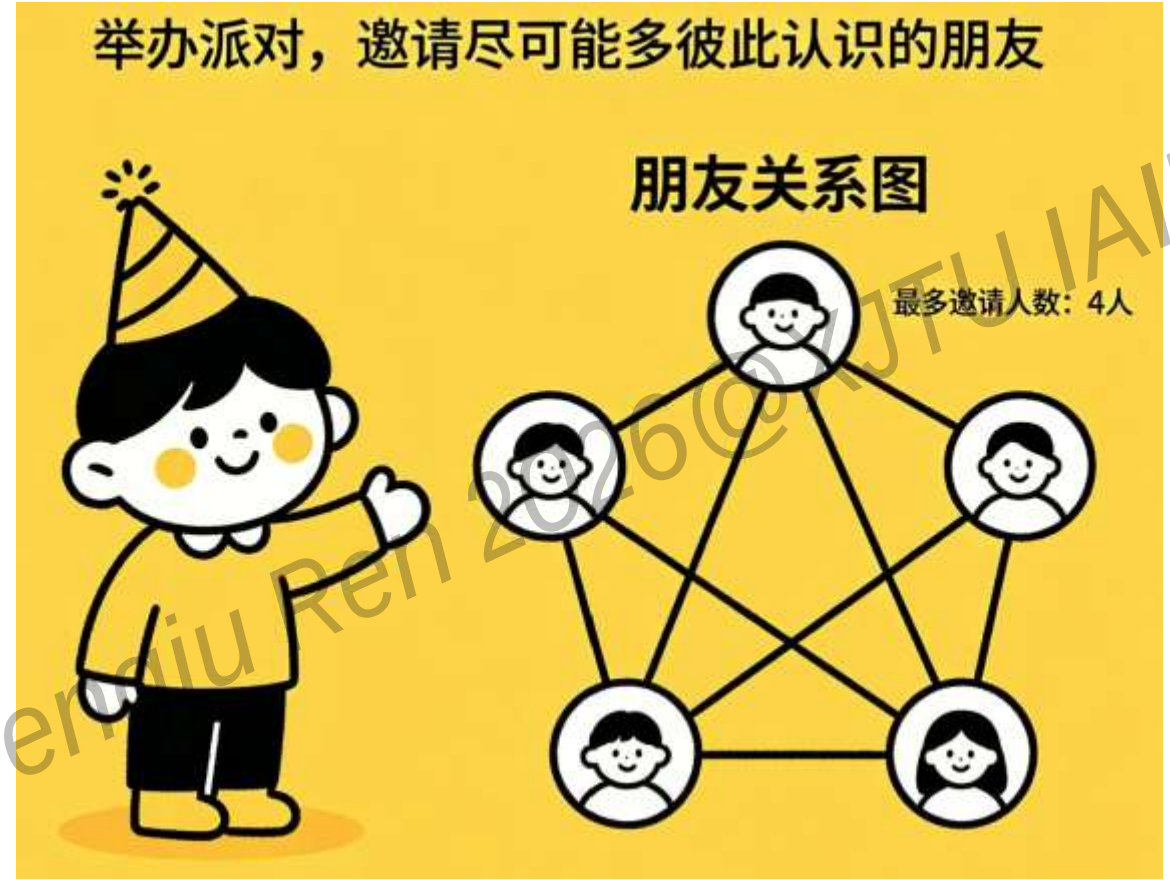
Data Structures and Algorithms

Lecture 11 – Nondeterministic Polynomial(NP)

Pengju Ren
Institute of Artificial Intelligence and Robotics
Xi'an Jiaotong University

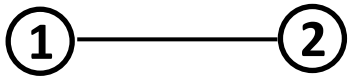
http://gr.xjtu.edu.cn/web/pengjuren

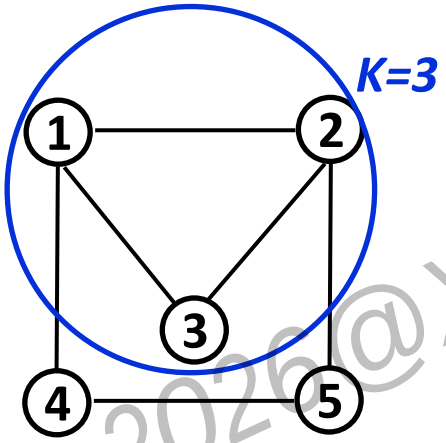
Problem Solving — The party invitation problem

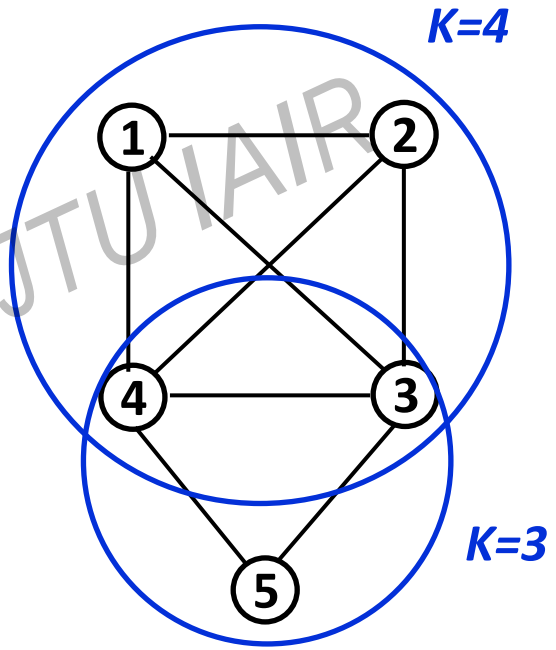


You want to invite *as many friends as possible*, but require that every invited pair *knows each other*. Given a friendship graph, what is the *maximum number of people* you can invite?

Clique Decision Problem(CDP)







Complete Graph: $|V| = n, |E| = \frac{n(n-1)}{2}$

Clique is the Complete subgraph

Does G contain a clique of size k? ***Decision Problem (Yes or No)***

What is the maximum Clique of G? ***Optimization Problem (K)***

Decision and Optimization Problem

Decision problem (Clique): Given an undirected graph $G = (V, E)$ does G contain a clique of size k (i.e., k vertices pairwise adjacent)?

Optimization problem (Maximum Clique): Given G, find the largest k such that a clique exists, and optionally output the clique.

For **discrete**, **bounded** optimization problems, they *can be converted into a finite number of decision problems* (often logarithmic many). This is a common technique in algorithm design and is one of the reasons why the decision version and the optimization version of NP-complete problems are closely related.

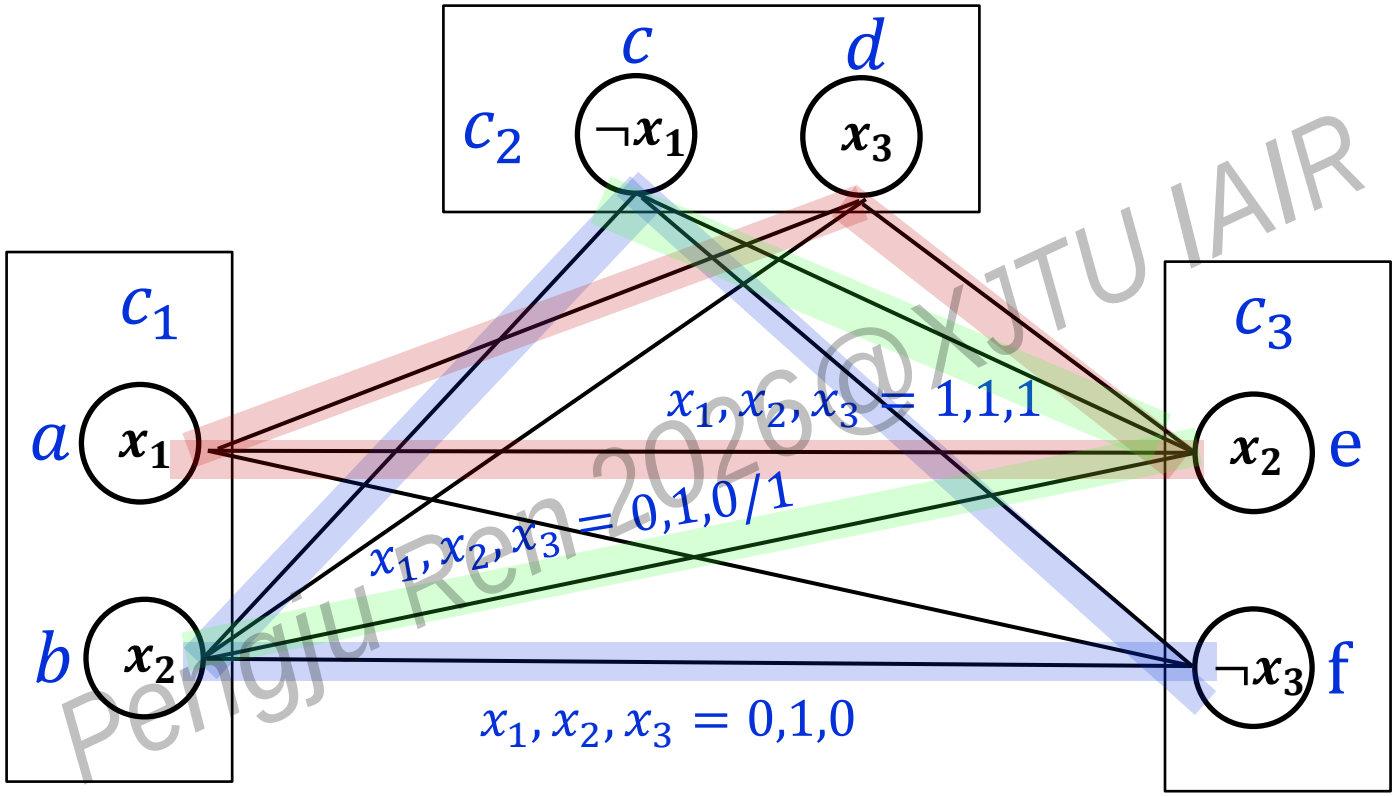
Boolean Satisfiability Problem (SAT)

Given a Boolean formula in *conjunctive normal form (CNF)*, determine whether there exists a truth assignment that makes the formula true.

Let the *CNF* formula be $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$

e.g. $\Phi = \underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(x_2 \vee \neg x_3)}_{C_3}$

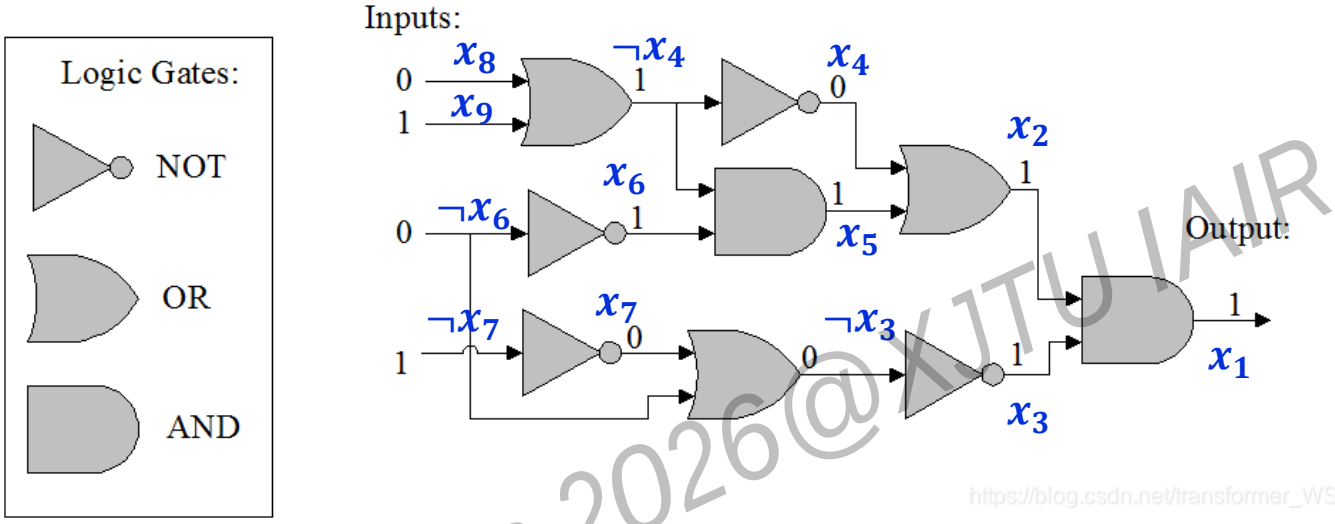
Does G contain a clique of size k(k=3)?



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

$$E\colon (u - v)\ exit\ if\ u \neq \neg v \text{ and } u, v\ in\ diff\ C_i$$

Circuit-SAT



Circuit Satisfiability Problem: given a combinational logic circuit composed of basic gates (AND, OR, NOT), determine whether there exists an assignment of input values that makes the circuit output 1 (high, true).

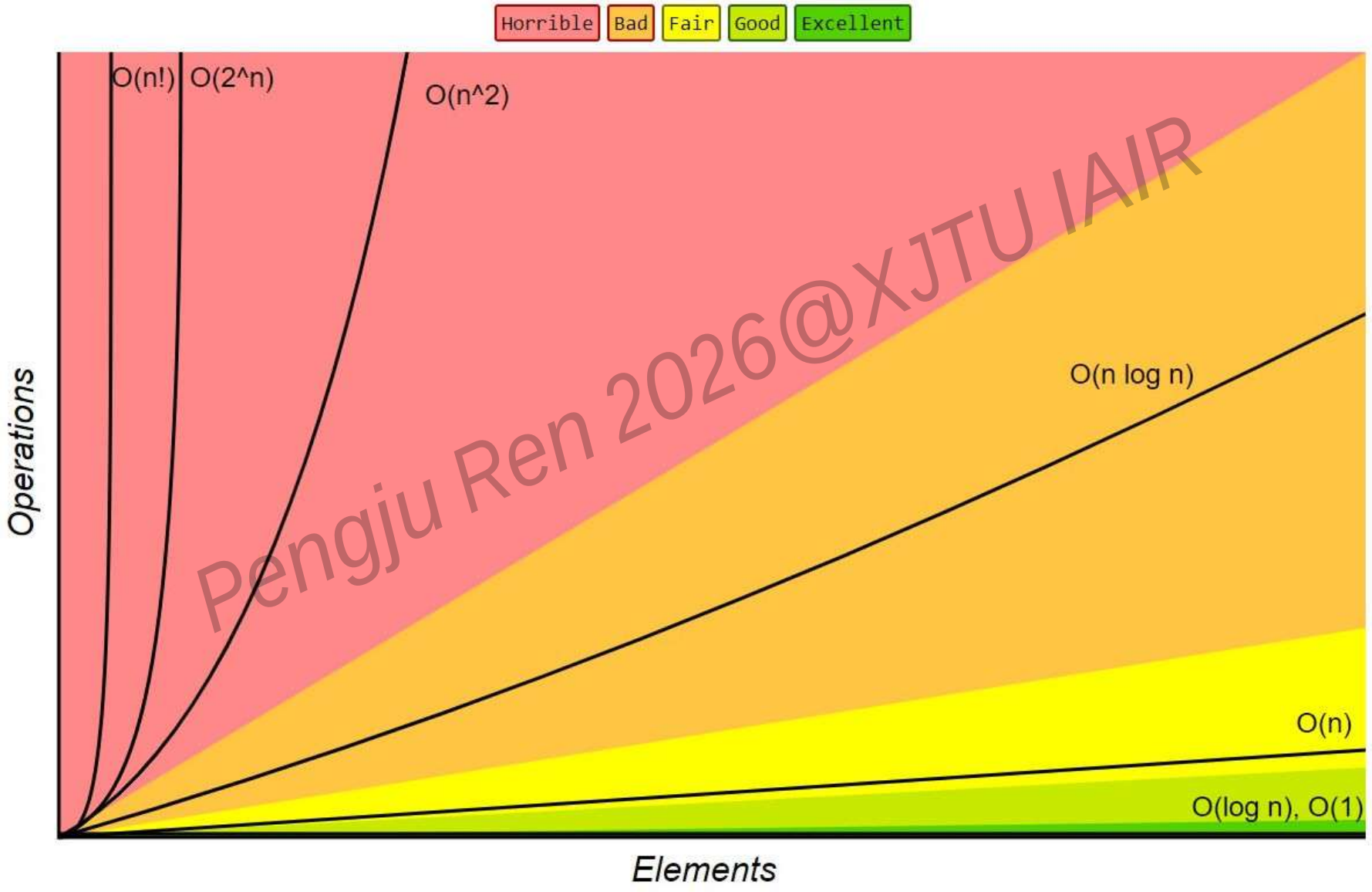
It was so fundamental (*prototypical NP-complete problem*) and was the *first problem proven to be NP-complete* (Cook-Levin theorem, 1971).

Time Complexities of Algo

Polynomial Time		Exponential Time	
Binary Search	$O(\log n)$	SAT	$O(2^n)$
Linear Search	$O(n)$	0/1 Knapsack	$O(2^n)$
Merge Sort	$O(n \log n)$	Traveling SP	$O(2^n)$
Insertion Sort	$O(n^2)$	Sum of Subsets	$O(2^n)$
Matrix Multiplication	$O(n^3)$	Graph Coloring	$O(2^n)$
	$O(n^{10})$	Hamiltonian Cycle	$O(2^n)$
	$O(n^{100})$		

Problems that can be solved in polynomial time

Big-O Complexity Chart



P and NP Problems

Polynomial time Problem: Problems that can be solved in ***polynomial time*** with respect to the input size n.

Non-deterministic Polynomial time problems (**Verifier definition):** NP problems are those for which ***a solution can be verified in polynomial time*** **(**or there exists a **deterministic polynomial-time verification algorithm).**

- $P \subseteq NP$: any problem solvable in deterministic polynomial time is trivially verifiable.
- Whether $P = NP$ is *an open millennium problem*. Most researchers believe $P \neq NP$, i.e., there exist NP problems (like SAT, TSP) that have no deterministic polynomial-time algorithm.

NP-Complete(NPC) and NP-hard Problems

A decision problem Q is **NP-complete** if:

- $Q \in NP$
- Every problem in NP *can be reduced to* Q in polynomial time

A problem H is **NP-hard** if there exists an **NPC** problem (or any problem in **NP**) that can be reduced to H in polynomial time.

Note: H need not be a decision problem, nor does it have to belong to NP (it could be an optimization problem or even harder).

Non-Deterministic Problem

- **Non-deterministic algorithm**: At some step, the computation can face *multiple possible choices* and it can "magically" guess the correct branch.

```
Algorithm: NonDeterministic_HamiltonianCycle(G)
Input: 无向图 G = (V, E), |V| = n
Output: 如果存在哈密顿回路则输出 "Yes", 否则输出 "No"
1.  // 非确定性阶段：猜测一个顶点的排列
2. Let v1, v2, ..., vn be a permutation of V
3. // 验证阶段：检查猜测是否构成合法回路
4.  for i = 1 to n-1 do
5.      if (vi, vi+1) not in E then
6.          reject   // 相邻顶点无边，此路径失败
7.      end if
8.  end for
9. // 检查最后一点与起点是否有边
10. if (vn, v1) not in E then
11.     reject
12. end if
13. accept   // 所有检查通过，存在哈密顿回路
```


If ***at least one path leads to a "yes"*** answer, the non-deterministic algorithm returns ***"yes"***; only when ***all paths return "no"*** does it return ***"no"***.

CNF-Satisfiability

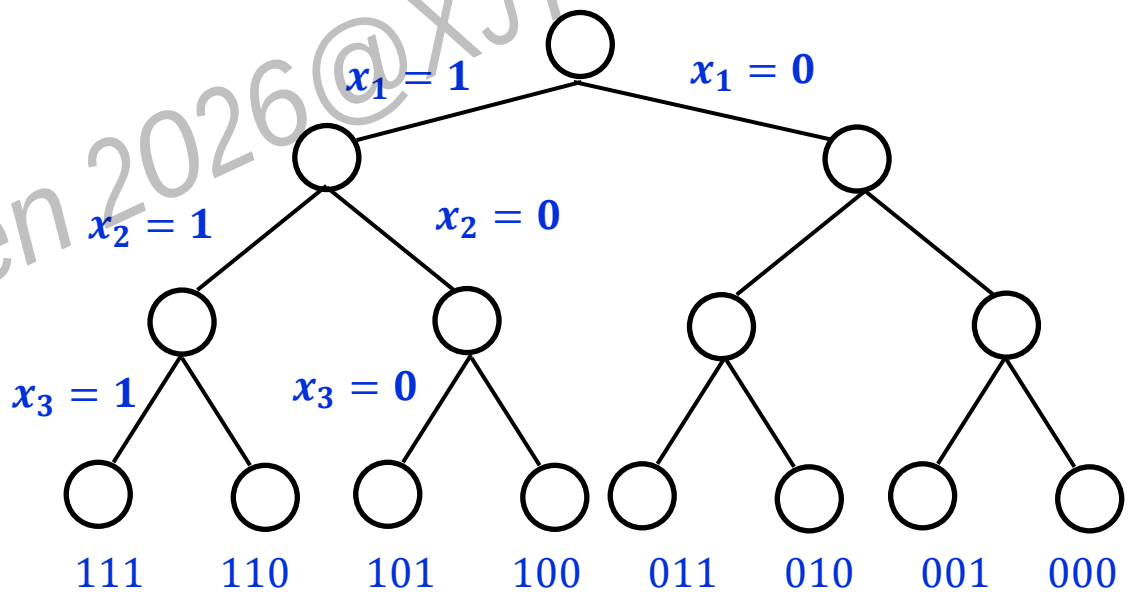
$$\Phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$$

$$x = \{x_1, x_2, x_3\}$$

x_1	x_2	x_3	x_4	x_5
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

2^n

n



Equivalent of Graph Coloring and CNF-Satisfiability

Idea: Given a graph $G = (V, E)$,construct a Boolean formula Φ_G (in CNF) such that Φ_G is satisfiable ***iff*** G is 3-colorable.

- **Each vertex gets at least one color**

$$(x_{v,1} \vee x_{v,2} \vee x_{v,3}) \text{ for every } v$$

- **Each vertex gets at most one color** (mutual exclusion)
  For every v and every pair of distinct colors $c \neq d$:

$$(\neg x_{v,c} \vee \neg x_{v,d})$$

- **Adjacent vertices must have different colors**
  For every edge $(u, v) \in E$ and every color c:

$$(\neg x_{u,c} \vee \neg x_{v,c})$$

The conjunction of all these clauses is Φ_G

Graph Coloring
0/1 Knapsack
Sum of Subsets
Traveling SP
Hamiltonian Cycle

Equivalent of 0/1 Knapsack and CNF-Satisfiability

Object	1	2	3	4
Profits	40	42	25	12
Weight	4	7	5	3
x_i	0/1	0/1	0/1	0/1

$$Total\ Profits = \sum p_i x_i$$

$$Total\ Weights = \sum w_i x_i$$

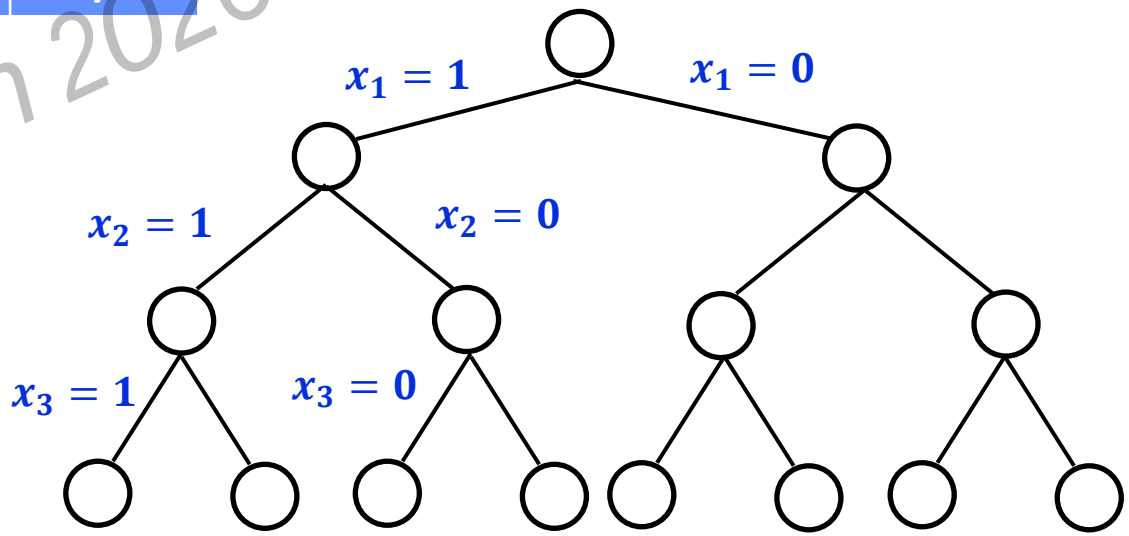
Graph Coloring

0/1 Knapsack

Sum of Subsets

Traveling SP

Hamiltonian Cycle

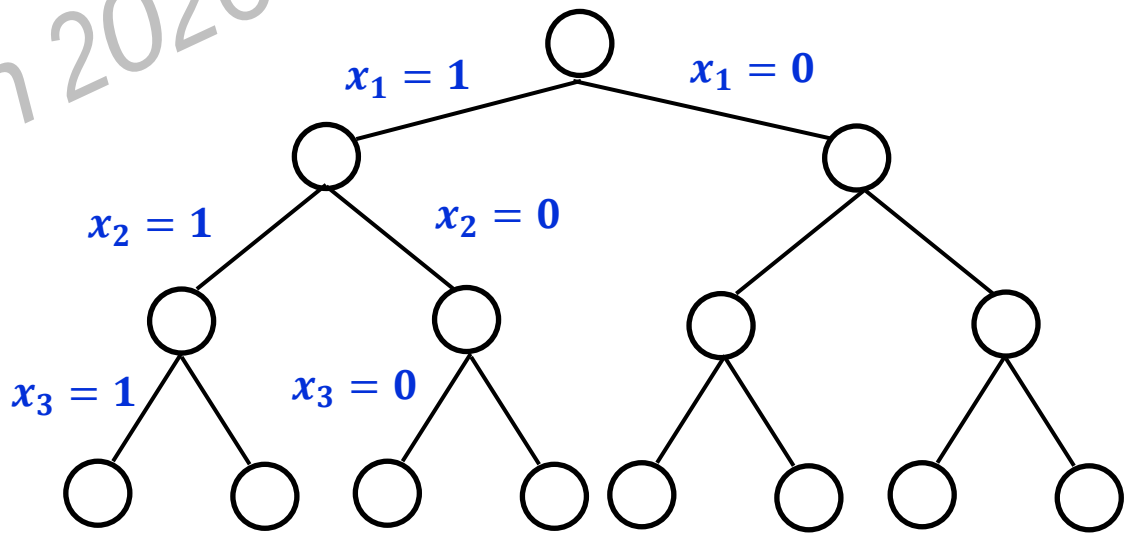


Equivalent of sum-of-subset and CNF-Satisfiability

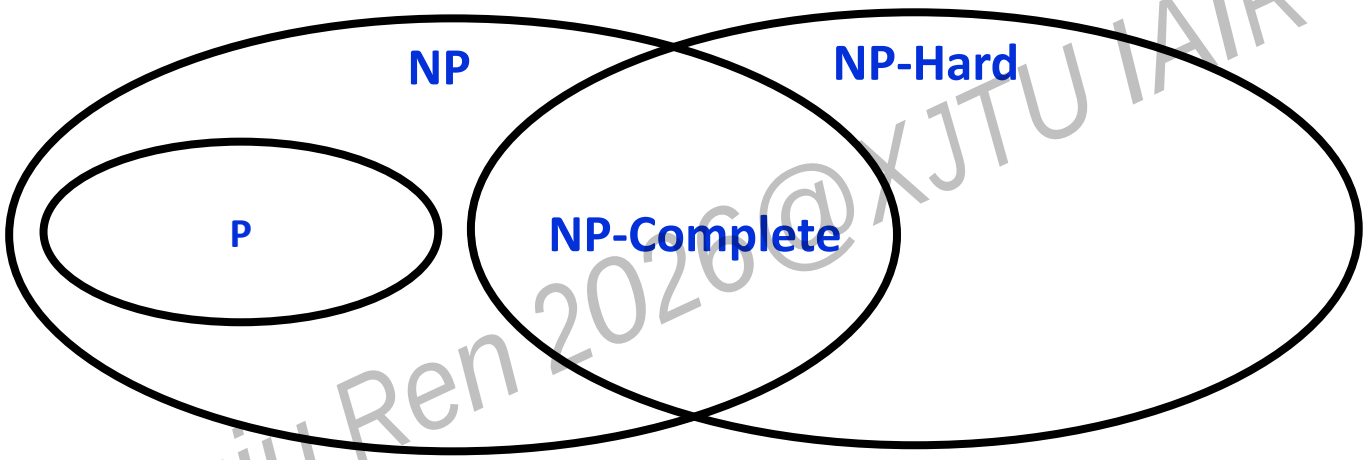
subset	1	2	3	4
N_i	2	3	5	7
x_i	0/1	0/1	0/1	0/1

$$10 = \sum N_i x_i$$

Graph Coloring
0/1 Knapsack
Sum of Subsets
Traveling SP
Hamiltonian Cycle



P、NP、NP-Complete and NP-Hard



$P \subseteq NP \quad NPC = NP \cap NP\text{-}hard$

Summary

Class	Full Name	Key Definition	Intuitive Understanding	Typical Example
P	Polynomial time	Decision problems solvable by a deterministic Turing machine in polynomial time	Can be **solved quickly**	Sorting, shortest path, bipartite matching
NP	Nondeterministic Polynomial time	Decision problems solvable by a nondeterministic Turing machine in polynomial time; equivalently, solutions can be **verified** in polynomial time	Can be **verified quickly**	SAT, Hamiltonian cycle, knapsack
NPC	NP-Complete	① In NP; ② Every problem in NP can be reduced to it in polynomial time	The hardest problems in NP	3-SAT, clique, subset sum (decision)
NP-hard	NP-hard	Every problem in NP can be reduced to it in polynomial time; **need not be in NP** (could be optimization problems, non-decision problems)	**At least** as hard as NP	TSP optimization, Halting problem, Circuit-SAT