NP-Completeness：數學家眼中的 CS

數學家的特性：

1. 很聰明但太多話說不清楚
   - 太聰明：腦袋會轉彎，簡單事想的太複雜，也說的太複雜
   - 數學家：其實是聽的人程度太差
2. 目標遠大，心中想的是全世界
   - 喜歡一次解決一大堆（甚至全世界）問題，而不是一個問題

想像自己是數學家才有辦法理解！

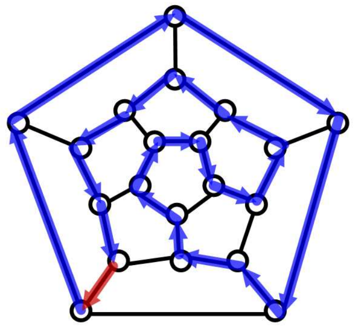
數學家的目標：

寫一個程式，一次解決全世界的所有問題

（目標很好，但心很壞，想讓 CS 的都失業）

The Hamiltonian cycle problem

Input: G = (V, E)



a Hamiltonian cycle H

A nondeterministic algorithm

Step 1: Guess a cycle H

Step 2: Verify whether H is a Hamiltonian cycle

(i) all edges exist?
(ii) visit each vertex exactly once?

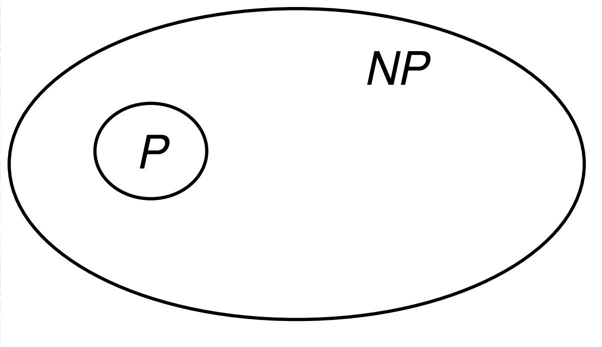
(return 1 if H is correct;
otherwise return 0.)

Non-deterministic sort (A)

```
Step 1:  Guess B as the answer
Step 2:  /* verification
          for i = 1 to n-1 /* sorted ???
               if not (B[i] <= B[i+ 1])  else return 0;

          for i = 1 to n  /* each A[i] is in B ???
              if B[i] is in A, remove it from A
              else return 0;

          return 1;
```



會計算就一定會驗算

Example: $(x^4-6x+8)^{1/2} - 3x^2+6 = 0$

Which of the following is easier?

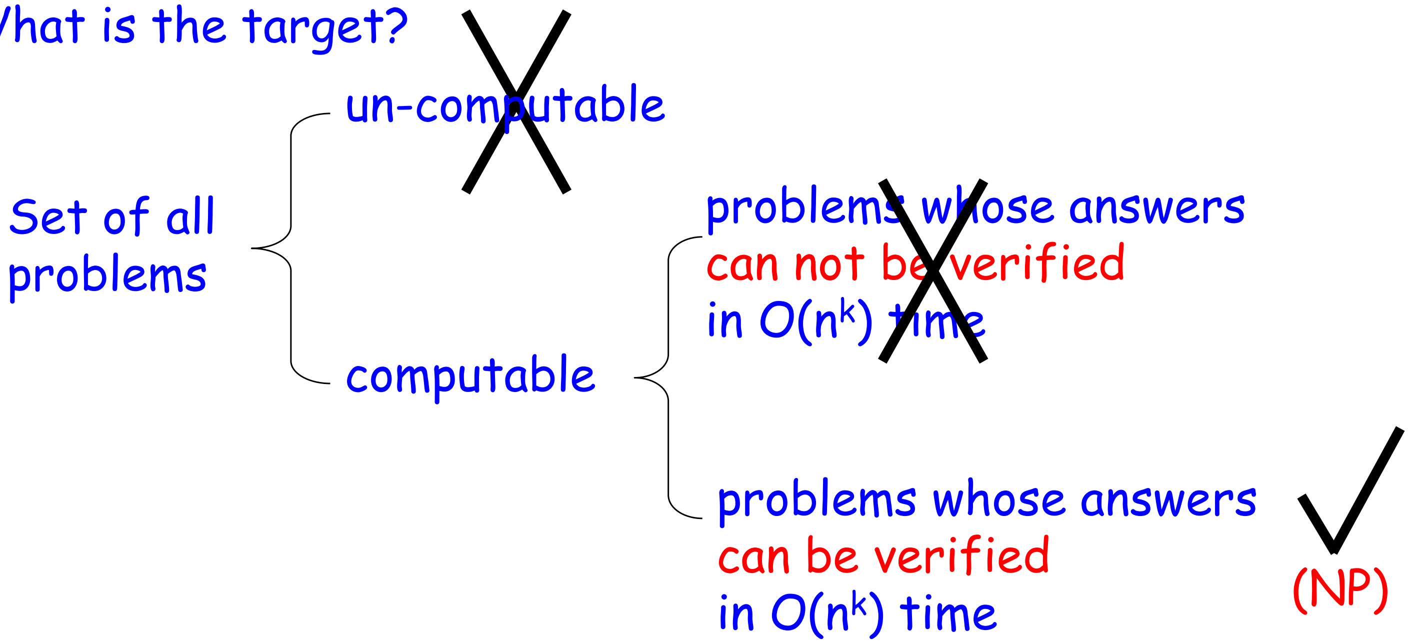
P: $x = ???$ (計 算 in polynomial time)

NP: $x = 3.412$??? (驗 算 in polynomial time)

What dose "solve" mean?

⇨ an algorithm that runs in polynomial time (P)

What is the target?



Set of all problems

- ~~un-computable~~
- computable
  - ~~problems whose answers can not be verified in $O(n^k)$ time~~
  - problems whose answers can be verified in $O(n^k)$ time ✓ (NP)

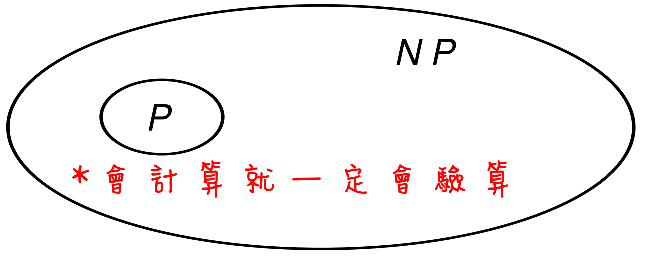
NP-Completeness (數學家眼中的 CS) - Review

目標： 寫一個程式，一次解決全世界的所有問題，為 CS 的人帶來光明（解救所有 programmers）

何謂解決（定義標準）： polynomial (easy , can be solved)

- P (problems "can be solved")

何謂全世界（決定對手）： NP (problems (answers) "can" be verified)



*會計算就一定會驗算

- $A \in NP$ 表示 A 是一個對手
- $A \in P$ 表示 A 被解決了
- 何謂解決全世界： prove NP = P

CS: I gave an $O(n^4)$ algo for A
=> Math: I proved $A \in P$

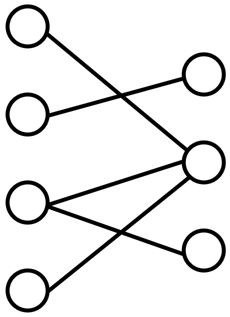
Reduction:

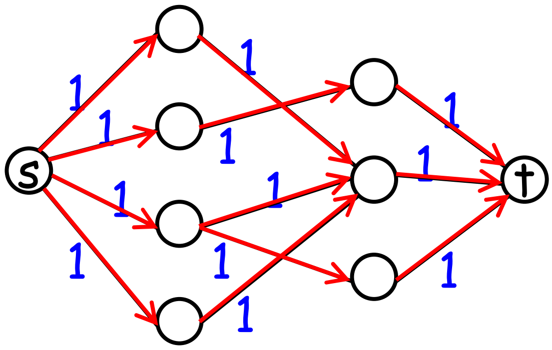
Problem A $\xrightarrow[\text{reduction}]{O(T)}$ Problem B

Example:

max bipartite matching $\xrightarrow[\text{reduction}]{O(V + E)}$ maximum flow

$G = (V, E)$





Partition Problem:

$S = (a_1, a_2, \ldots, a_n)$ $\Rightarrow$ S_1 and S_2 such that $\text{Sum}(S_1) = \text{Sum}(S_2)$

3-Partition Problem:

$S = (a_1, a_2, \ldots, a_n)$ $\Rightarrow$ S_1, S_2, S_3 such that $\text{Sum}(S_1) = \text{Sum}(S_2) = \text{Sum}(S_3)$

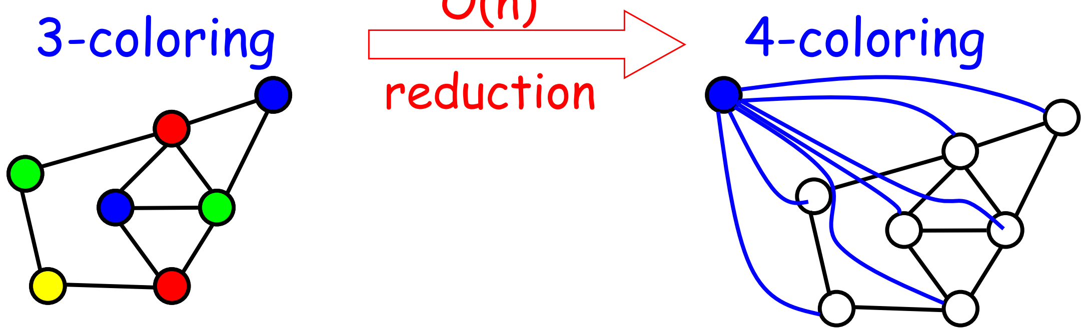
Partition problem $\xrightarrow[\text{reduction}]{O(n)}$ 3-partition problem

$S = (a_1, a_2, \ldots, a_n)$ $S' = (a_1, a_2, \ldots, a_n, \text{Sum}(S)/2)$

$S = (3, 5, 7, 9)$ $S' = (3, 5, 7, 9, 24/2)$

$= (3, 5, 7, 9, 12)$

Coloring: Given G, assign color to each node such that adjacent nodes have different colors.



3-coloring $\xrightarrow[\text{reduction}]{O(n)}$ 4-coloring

3-coloring: Determine whether a given G can be colored by using {0, 1, 2}.

4-coloring: Determine whether a given G can be colored by using {0, 1, 2, 3}.

Reduction:

Problem A $\xrightarrow[\text{reduction}]{O(T)}$ Problem B

* usually: easy ⇒ hard (or, as hard as)

A can be solved by solving B

B is more difficult

* may: hard ⇒ easy

Example:

embed the difficulty

Find(A, x) $\xrightarrow[\text{sort } A]{O(n \lg n)}$ Binary Search(S, x)

O(n) O(lg n)

(i) If $A \xrightarrow{\quad} B$ then A: $O(n^a + n^b)$

? $O(n^a)$ $O(n^b)$

imply a solution of A

(ii) If $A \xrightarrow{\quad} B$ then B: ?

$O(n^a)$ $O(n^b)$?

imply nothing for B

(iii) If $A \xrightarrow{\quad} B$ then

$\Omega(n^a)$ $O(n^b)$?

may imply difficulty of B

case 1. $\Omega(n^{10})$ $O(n^5)$

case 2. $\Omega(n^{10})$ $O(n^{11})$

case 1: b < a ⇒ B: $\Omega(n^a)$

/* e.g. B: $O(n^9)$ then A: $O(n^5) + O(n^9) = \cancel{O}(n^9)$

case 2: b ≥ a ⇒ B: ?

/* may: hard ⇒ easy, or easy ⇒ hard

Polynomial Reduction is transitive

$$A \xrightarrow{T_1}_{p} B$$

$$B \xrightarrow{T_2}_{p} C$$

➡ $A \xrightarrow{T_1 + T_2}_{p} C$

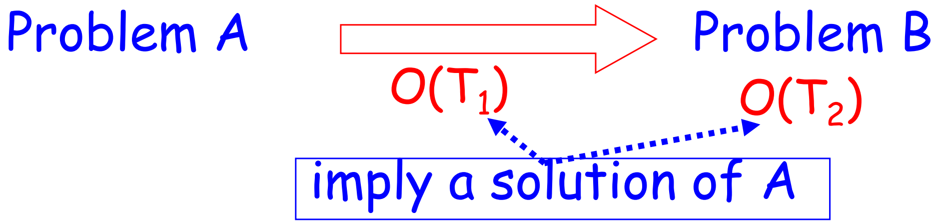
in sense of hardness

$B \geq_{\text{hard}} A$

$C \geq_{\text{hard}} B$

➡ $C \geq_{\text{hard}} A$

(解 掉 C 就 解 掉 A 和 B)



$$\text{Problem } A \xrightarrow{\qquad} \text{Problem } B$$

$O(T_1)$ $O(T_2)$

imply a solution of A

* A: $O(T_1 + T_2)$

* Assume that T_1 is polynomial

數學家心中的 reduction 都是 polynomial

⇨ $A \in P$ if T_2 is polynomial

⇨ $A \in P$ if $B \in P$

解掉 B 就解掉 A

* If $A \xrightarrow{O(n^k)} B$, then $A \in P$ if $B \in P$
(i.e., A can be solved by solving B.)

解掉 B 就解掉全世界 (NP)

* If all $NP \xrightarrow{O(n^k)} B$, then all $NP \in P$ if $B \in P$
(i.e., all NP can be solved by only solving B.)
(i.e., $NP = P$ if $B \in P$.)

假設真的存在

Example: 說服 全班 去聯誼
(解決) (全世界)

method 1. 一個一個的說服 (by CS people)

method 2. 透過 reduction 減少對手

小明： 小王去我就去

$\text{小明} \Longrightarrow \text{小王}$ (解決小王就解決小明)
(對付小王不必管小明)

全班： 康樂去我就去

$\text{所有人} \Longrightarrow \text{康樂}$ (對付康樂不必管全世界)
(擒賊先擒王)

What is the goal?

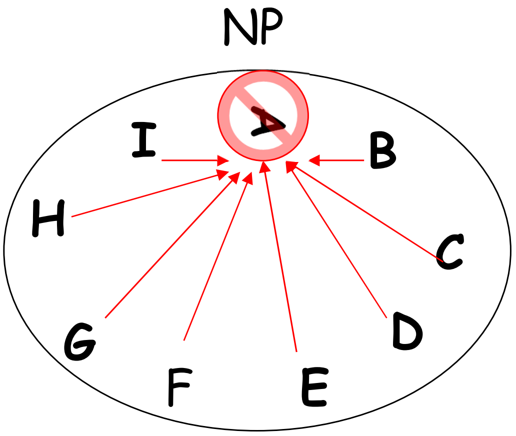
⇨ solve all problems in NP (in polynomial time) at a time
(i.e., prove NP = P)

$A \in NP$ 表示 A 是一個對手
$A \in P$ 表示 A 被解決了

How can "all problems" be solved at a time?

⇨ Idea: reduction

(1) find a problem A in NP such that all problems in NP can be reduced to A in polynomial time
(2) solve A in polynomial time

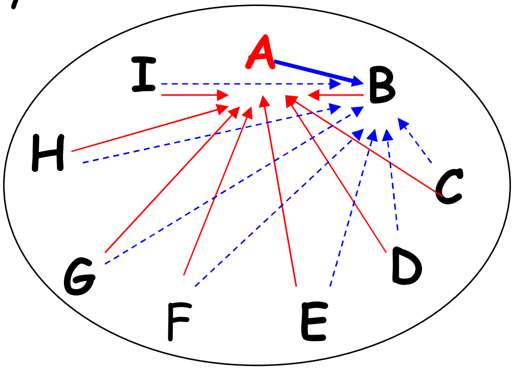


such a problem A is NP-C
(if exists)

A is NP-C:
(1) A is in NP
(2) all NP problems can be reduced to A in polynomial time

Assume that there exists an NP-C A.

If A can be reduced to B in polynomial time
(1) all NP problems can also be reduced to B
(2) B is NP-C



all NP $\Rightarrow^{p}$ B $\cong$ an NP-C $\Rightarrow^{p}$ B

Example: 說服 全班 去聯誼

(解決) (全世界)

method 1. 一個一個的說服 (by CS people)

method 2. 透過 reduction 減少對手

小明：小王去我就去

小明 ⇨ 小王 (解決小王就解決小明)

(對付小王不必管小明)

全班：康樂去我就去

所有人 ⇨ 康樂 (康樂 is NP-C)

康樂：班代去我就去

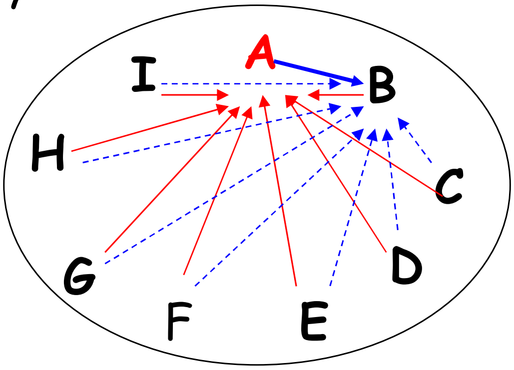
康樂 ⇨ 班代

所有人 ⇨ 班代 (班代 is also NP-C)

A is NP-C:

(1) A is in NP

(2) all NP problems can be reduced to A in polynomial time



Assume that there exists an NP-C A.

If A can be reduced to B in polynomial time

(1) all NP problems can also be reduced to B

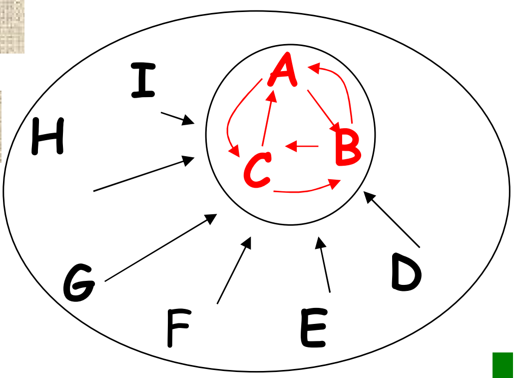
(2) B is NP-C

(3) B is as hard as A

All NP-C problems are of the same difficulty.

(They can be reduced to each other.)

all NP $\Rightarrow^{p}$ B ≅ an NP-C $\Rightarrow^{p}$ B ≅ all NP-C $\Rightarrow^{p}$ B

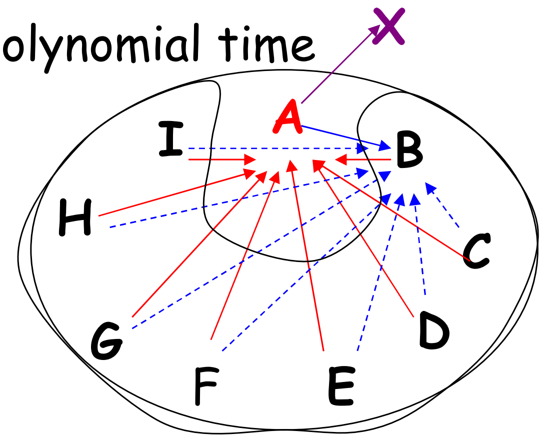


If an NP-C is solved

⇨ all NP (including all NP-C) are solved,

A is NP-C:
(1) A is in NP
(2) all NP problems can be reduced to A in polynomial time

A is NP-H: only (2)



Assume that there exists an NP-C A.

If A can be reduced to B in polynomial time
(1) all NP problems can also be reduced to B
(2) B is NP-C
(3) B is as hard as A

If A can be reduced to an X $\notin$ NP in polynomial time
(1) all NP problems can be reduced to X
(2) X is NP-H, but not NP-C
(3) X is harder than A

Example: 說服 全班 去聯誼
(解決)(全世界)

method 1. 一個一個的說服 (by CS people)

method 2. 透過 reduction 減少對手

全班：康樂去我就去

所有人 ⇒ 康樂 (康樂 is NP-C) (& NP-H)

康樂：班代去我就去

康樂 ⇒ 班代

所有人 ⇒ 班代 (班代 is NP-C) (& NP-H)

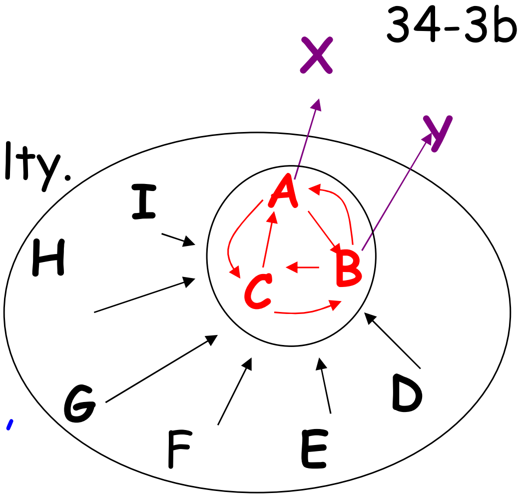
班代：我最尊敬女朋友小美 (外系), 言聽計從

班代 ⇒ 小美

所有人 ⇒ 小美 (小美 is NP-H, not NP-C)

NP-H: A, B, C, X, Y
NP-C: A, B, C

All NP-C problems are of the same difficulty.
But, all NP-H problems are not.



If an NP-H or NP-C is solved

⇨ all NP (including all NP-C) are solved, but not all NP-H

Goal: Solve all problems in NP at a time.

How: Solve a problem in NP-C?

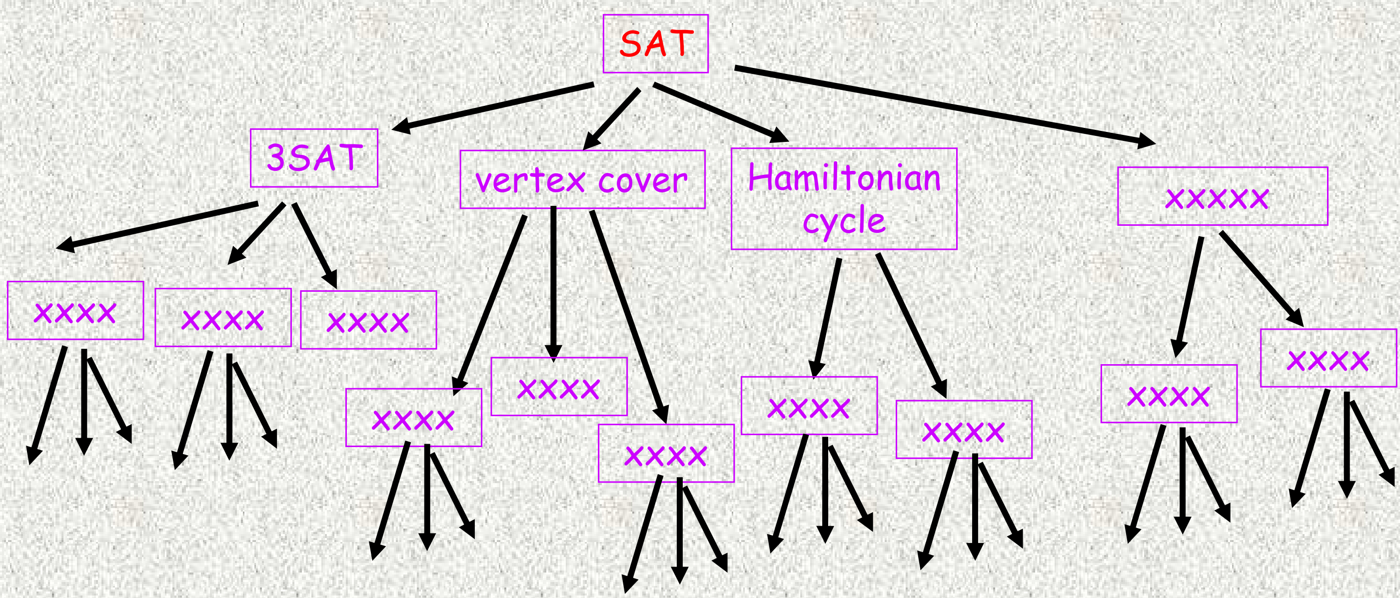
QUESTION: Dose NP-C exist?

Note that it is impossible to solve a problem in NP-H, but not in NP-C. Why?

Two Efforts

(1) Find problems in NP-C ✓

- 1st: SAT /* all NP-C ⇨^{p} SAT (turing award)


SAT
3SAT
vertex cover
Hamiltonian cycle
xxxxx
xxxx
xxxx
xxxx
xxxx
xxxx
xxxx
xxxx
xxxx
xxxx
xxxx

(2) Solve an NP-C ✗

- ????!!!!

How to prove a problem A is in NP-C (NP-H)?

① Show that A is in NP.
give an $O(n^k)$-time nondeterministic algo for A

② Show that all in NP $\xrightarrow{O(n^k)}$ A.

(a) Find a problem Y ∈ NP-C

(b) Show Y $\xrightarrow{O(n^k)}$ A

$$\begin{tikzcd} \text{all NP} \arrow[r, "O(n^k)"] \arrow[rr, bend right, "O(n^k)"'] & Y \arrow[r, "O(n^k)"] & A \end{tikzcd}$$

Example:

Prove 3-partition ∈ NP-C.

① Show that A is in NP.

(i) guess S_1, S_2, S_3

(ii) check $S_1 \cup S_2 \cup S_3 = S$ and
$\text{Sum}(S_1) = \text{Sum}(S_2) = \text{Sum}(S_3)$
$O(n \lg n)$ time

② Show that all in NP $\xrightarrow{O(n^k)}$ A.

(a) It is known 2-partition ∈ NP-C

(b) 2-partition $\xrightarrow{O(n)}$ 3-partition

(omit ① for NP-H)

NP-Completeness (數學家眼中的 CS) - Review

目標： 寫一個程式，一次解決全世界的所有問題，
為 CS 的人帶來光明 (解救所有 programmers)

何謂解決 (定義標準)： polynomial (easy , can be solved)
- P (problems "can be solved")

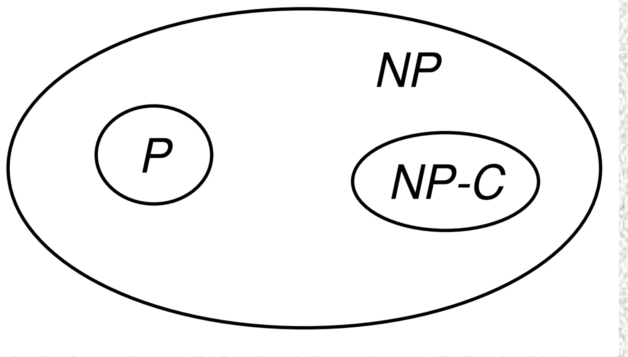
何謂全世界 (決定對手)： NP (problems "can" be verified)
- 解決全世界： prove NP = P

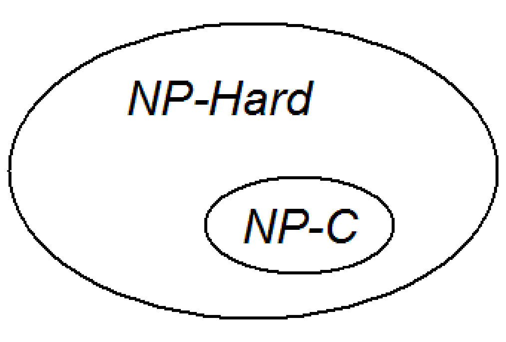
如何下手：擒賊先擒王 (直接解決敵軍中的大將軍)
- reduction: 找出對手中的大魔王 NP-C
  - 解決大魔王

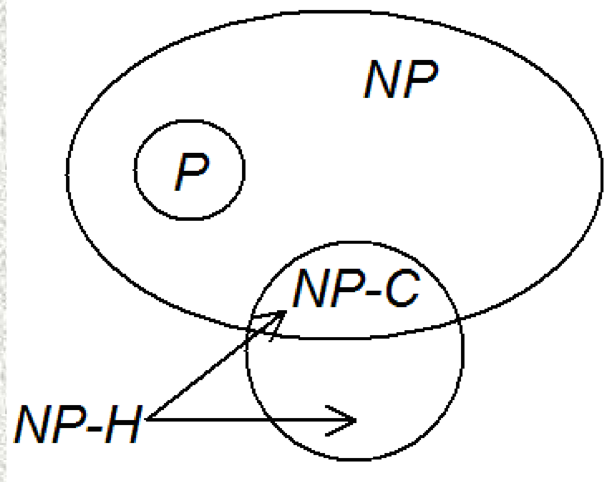
問題： 大魔王真的存在嗎？ (yes, 1st - SAT, Turing award)

剩下的問題：大魔王可以被打敗嗎 (can an N-PC be solved) ?
- NP = P or NP ≠ P? (unknown, but believe NP ≠ P)

結果：只帶來悲慘的消息 (沒帶來光明，反而帶來黑暗)
- 這世界到處都是不知如何對付的大魔王







(a) NP-C: No one knows how to solve these problems. (Y/N)
(No algorithms exist for these problems.)

(b) If we consider NP as an army, then
NP-C: ? NP-H but not NP-C: ?

If an NP-C surrenders, then
all NP too? all NP-C too? all NP-H too?

If an NP-H surrenders, then
all NP too? all NP-C too? all NP-H too?

(1) How to prove a problem is P?

(2) How to prove a problem is NP?

(3) How to prove a problem is NP-C?

(4) How to prove a problem is NP-H?

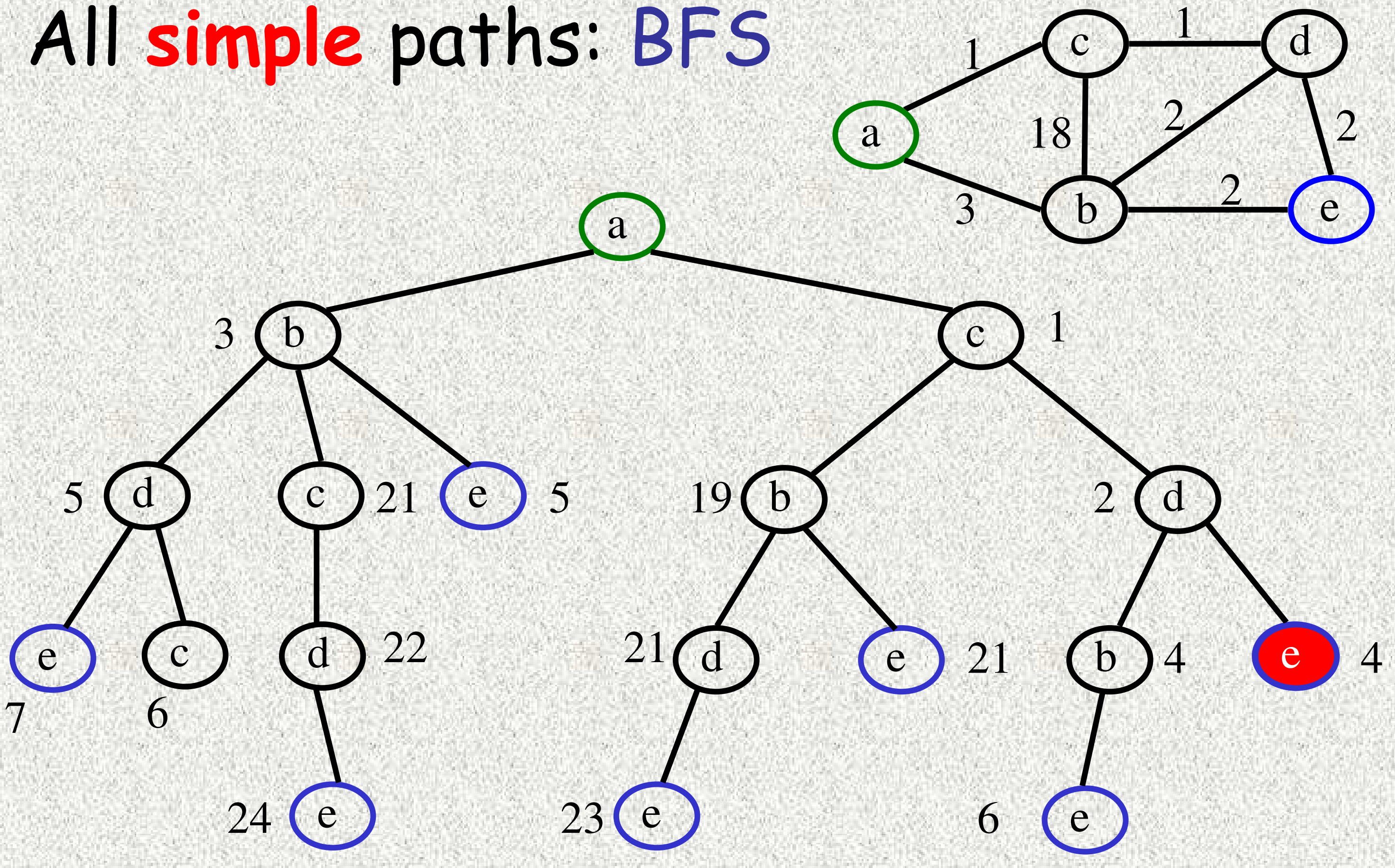
(5) How to prove NP = P?

(6) How to prove NP $\neq$ P?

(7) NP = P or NP $\neq$ P ?

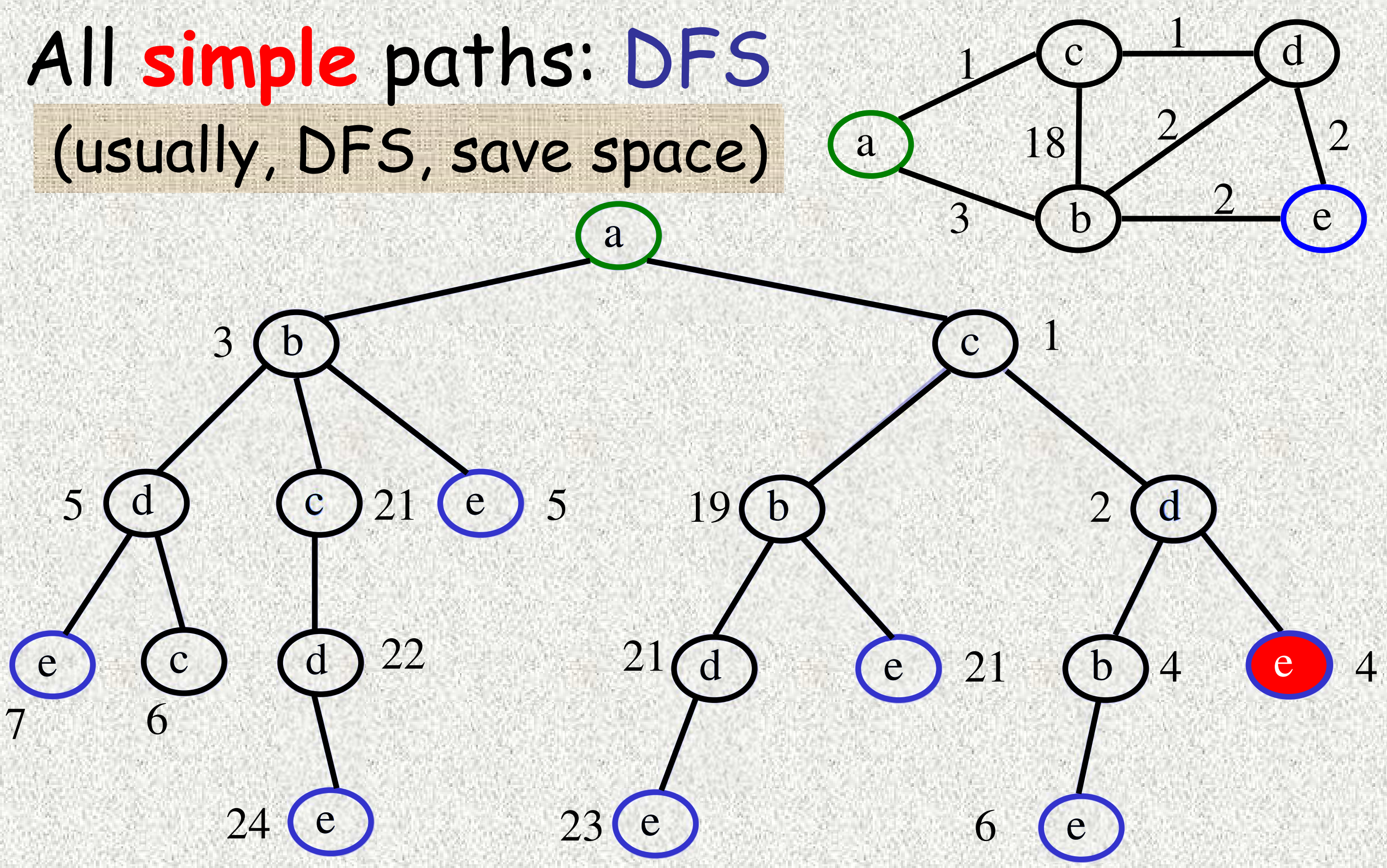
(8) What do we learn ?

All simple paths: BFS



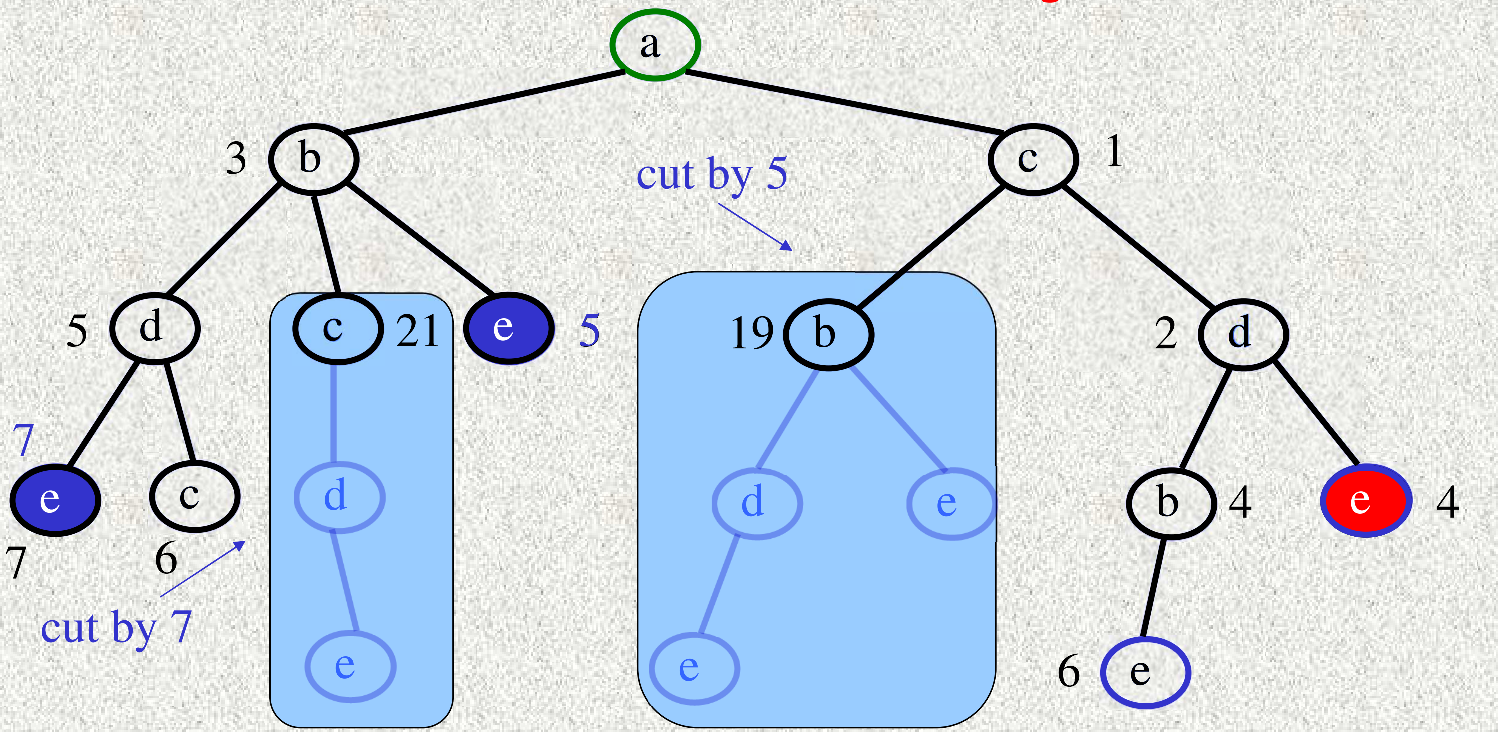
All simple paths: DFS

(usually, DFS, save space)



Branch-and-bound

- Branch-and-bound: Brute-force + intelligent cuts



Optimization Problems

P
- * trivial
- * greedy
- * DP

NP-hard
- * brute-force (n ≤ 20)
- * B & B (n ≤ 30~100)
- * n > 100 ???

⇨ exact solution

- * approximation
- * heuristic

⇨ near-optimal solution

Remark: Pseudo-polynomial algorithms (DP, usually) are possible for NP-H problems (when inputs are small integers)