NP-Completeness

Chapter 34

Mei-Chen Yeh

5/30 (二)
START

Introduction

- Polynomial-time algorithm:
  - Input size *n*
  - The worst-case running time is O(n^k)

$f(n) \setminus n$	10	10^2	10^3
$log_2 n$	3.3	6.6	10
n	10	10^2	10^3
$nlog_2 n$	0.33×10^2	0.7×10^3	10^4
n^2	10^2	10^4	10^6
2^n	1024	1.3×10^{30}	$> 10^{100}$
$n!$	3^6	$> 10^{100}$	$> 10^{100}$

Polynomial Time

Exponential Time

Computational Tractability

- When is an algorithm an efficient solution to a problem?
  - When its running time is polynomial in the size of the input.
- A problem is *computationally tractable* if it has a polynomial-time algorithm.

Problem classification

- Polynomial time
  - Shortest path
  - Matching
  - Minimum cut
  - …
- Probably not
  - Longest path
  - Maximum cut
  - Vertex cover
  - …

Problem classification

- ***Tractable*** problem
  - Can be solved by polynomial-time algorithms
- Intractable problems?
- Non-solvable problems?

The halting problem

- Given ***a program*** and ***an input***, determine whether the program will eventually halt (finish, stop) when running with that input.
  - Examples:

while True: continue	not halt
print "Hello World!"	halt very soon

halting problem 無解

Proven in 1936 that a general algorithm to solve the halting problem for *all* possible program-input pairs ***cannot exist***.

Solving Diophantine equations

- Given a polynomial equation, is there a solution in integers?
  有整數解回答是，沒有整數解回答不是
  - Example: $4xy^2 + 2xy^2z^3 - 11x^3y^2z^2 = -1164$

Proven in 1971 that **no** such a decision procedure can exist.

無解

這個章節的討論會限縮在decision problem

Decision problem: Y/N

- **The subset sum problem**
  - Given a set of integers, is there a non-empty subset whose sum is exactly zero?
  - Input: {−7, −3, −2, 8, 5}
  - “yes”
  - { −3, −2, 5}

Decision problem: Y/N

- **The partition problem**
  - Decide whether a given set of integers can be partitioned into two disjoint sets that have the same sum
  - Input: {13, 2, 17, 20, 8}
  - “yes”
  - {13, 17} and {2, 20, 8}

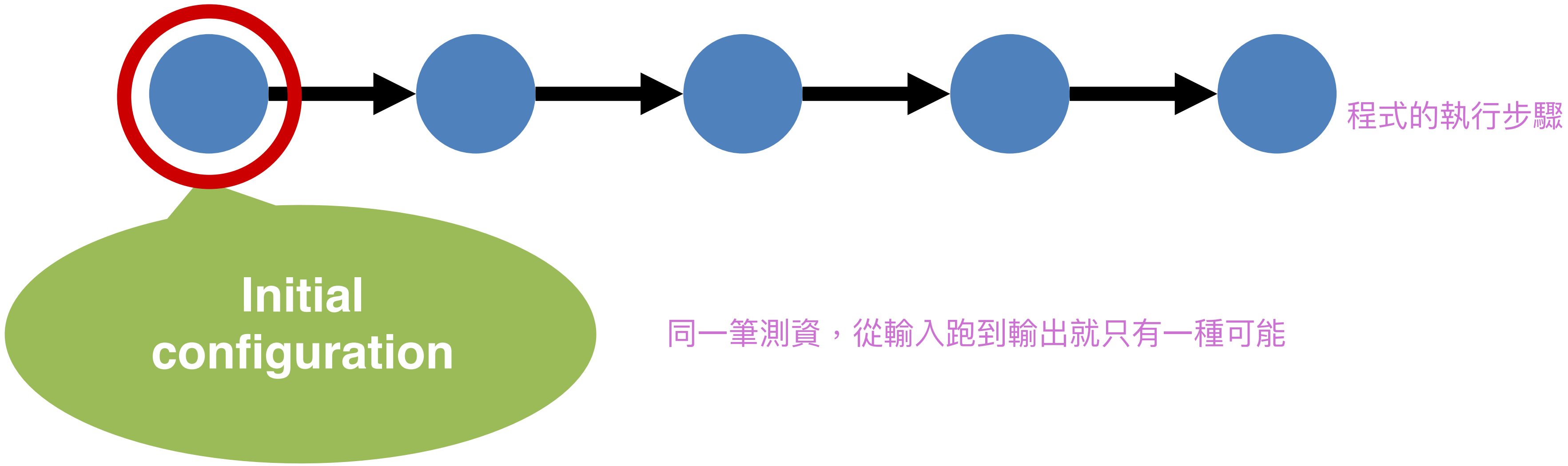
Problem classes: P vs. NP

根據問題的困難度分類，分成四類

- P 有polynomial time的演算法的問題，叫做P問題
  - Problems that can be solved in **p**olynomial time.
  - Examples: sorting, finding shortest paths, ….
- NP
  - Problems that can be solved in **n**on-deterministic **p**olynomial time.

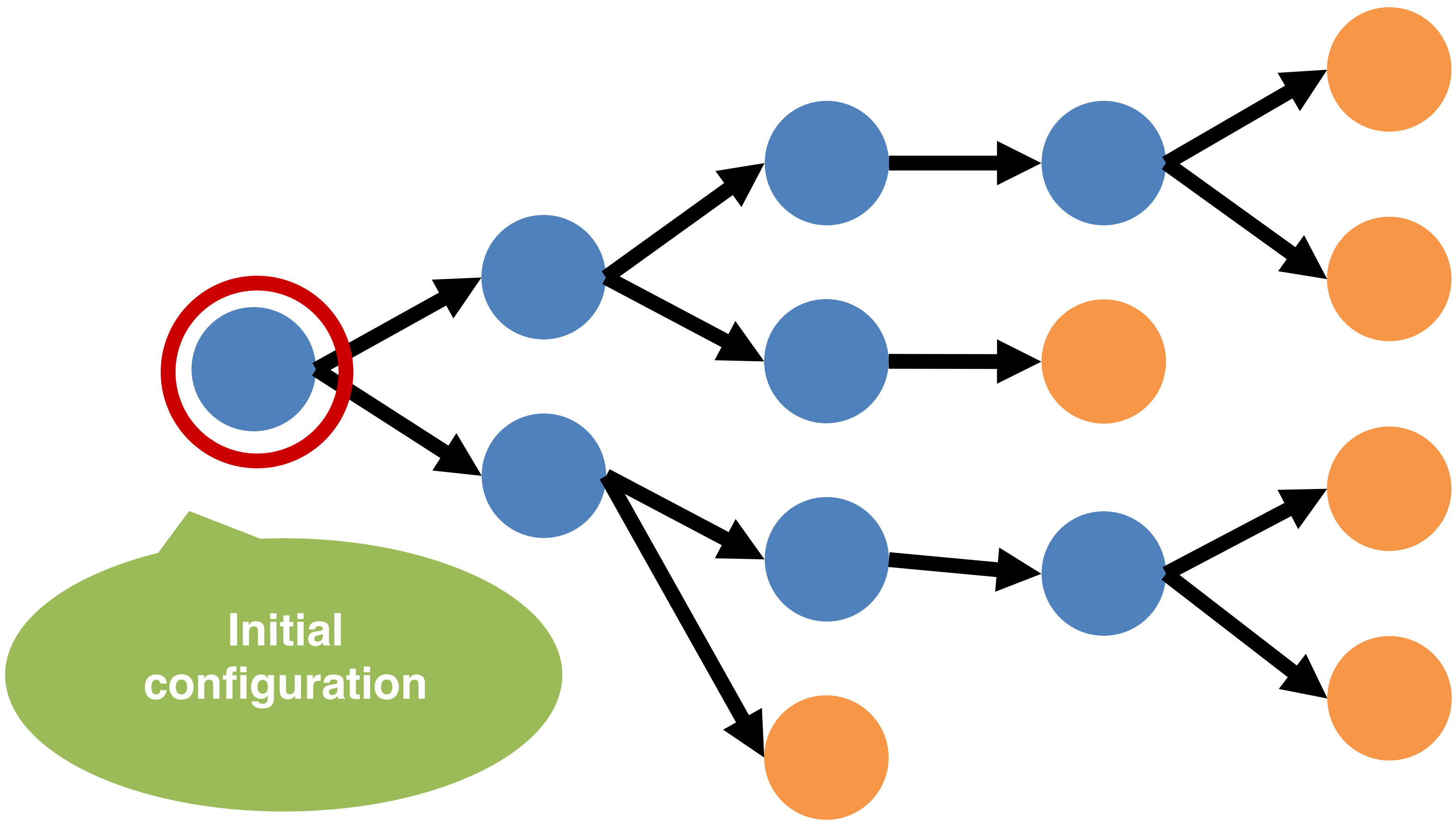
NP問題是指，可以被解掉，但時間會是 non–deterministic polynomial time 的問題

Deterministic algorithm



Permit at most one next move at any step in a computation.

Non-deterministic algorithm



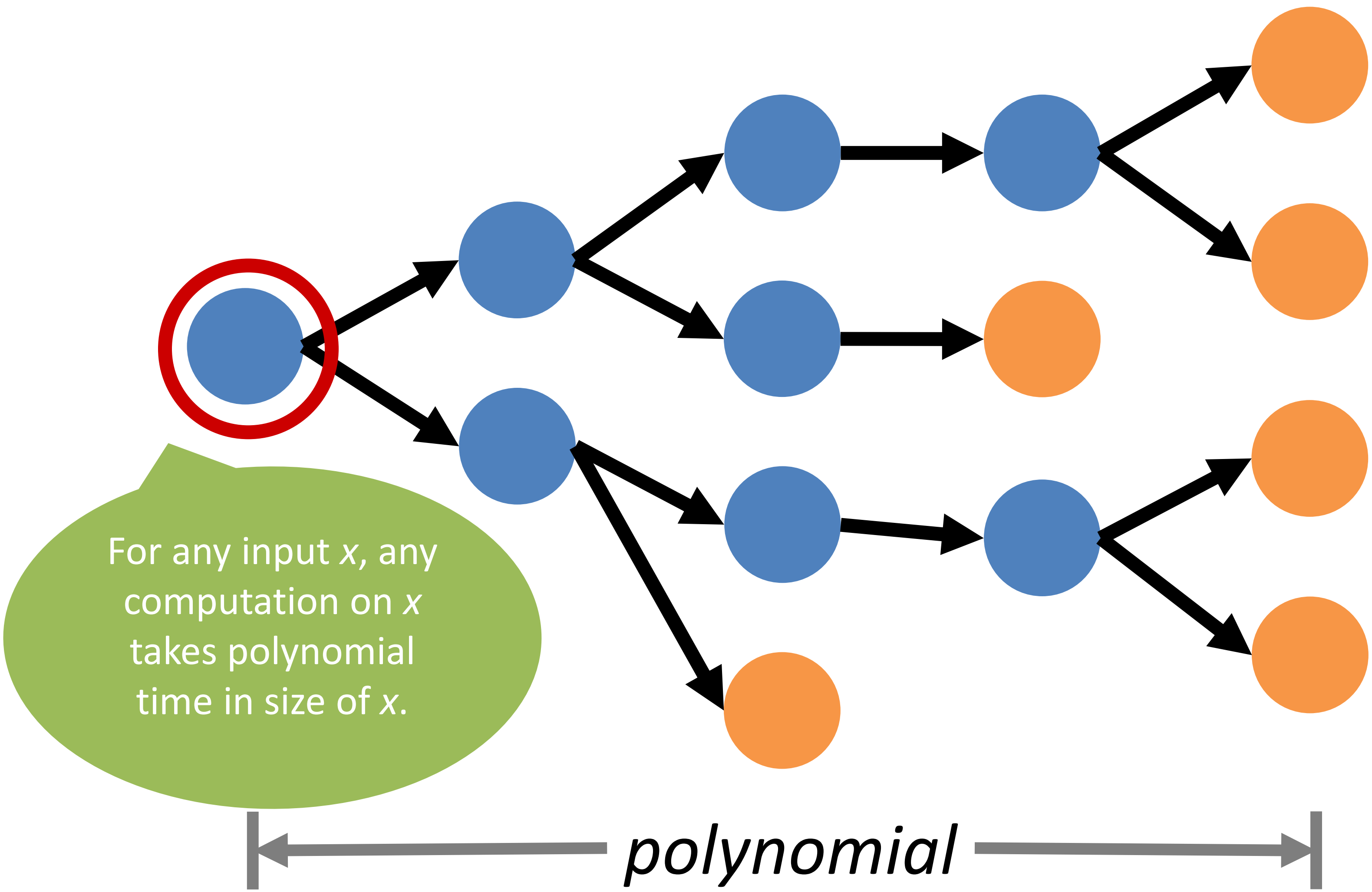
Permit more than one choices of next move at some step in a computation.

Non-deterministic search

```
int j = Choice(1, n);
    if A[j] == x
        return j
    else
        return 0
```

演算法裡面如果有隨機（random）的
成分，就算是一種non–deterministic

Non-deterministic Polynomial time



可以保證輸入到輸出的時間在polynomial time內

P and NP

- P
  - Problems that can be ***solved*** in polynomial time
- NP
  - Problems that can be ***verified*** in polynomial time

可以快速驗證答案的正確性

Finding vs. Checking

- Is it easy to check if two given disjoint sets have the same sum?
- Is it easy to check if a given subset of integers whose sum is exactly zero?
- We draw a contrast between finding a solution and checking a solution (in polynomial time).

P 問題是finding solution很快，N P 問題是checking solution很快

P vs. NP

BCT ✓

Q.

P 問題與 N P 問題之間的關係？

$$P \subseteq NP$$

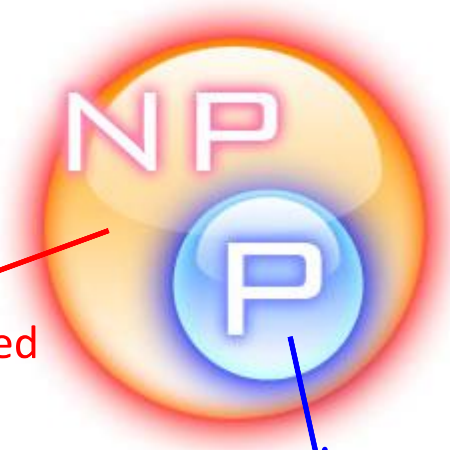
P 問題是 N P 問題的一種
(找答案都很快了檢查答案當然也快)

$$P \stackrel{?}{=} NP$$

但還沒有人可以證明 P 問題就是 N P 問題

if 'yes'-answers to a 'yes'-or-'no'-question can be verified "quickly" can the answers themselves also be computed "quickly"?

P = NP?



solutions can
quickly be checked

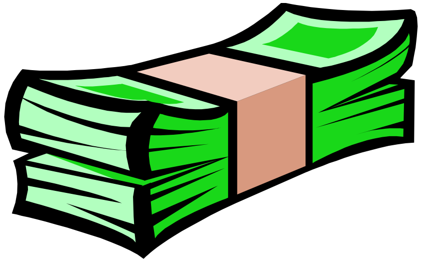
problems can quickly
be solved

- P
  - Can be solved in polynomial time
- NP
  - Can be verified in polynomial time
  - Can also be solved in polynomial time?
  - **No body knows.** So far, no one can prove or disprove it.

One of the Millennium Prize Problems!

since 2000

- US $1,000,000 per problem
  - Birch and Swinnerton-Dyer Conjecture
  - Hodge Conjecture
  - Navier-Stokes Equations
  - **P versus NP**
  - Poincaré Conjecture
  - Riemann Hypothesis
  - Yang-Mills Theory



solved

by Grigori Perelman in 2003,
reviewed in 2006,
awarded on March 18, 2010

But he *declined* the money and the award.
He either declined the Fields Medal award in 2006.

http://www.claymath.org/millennium/

P, NP, NP-hard, NP-complete

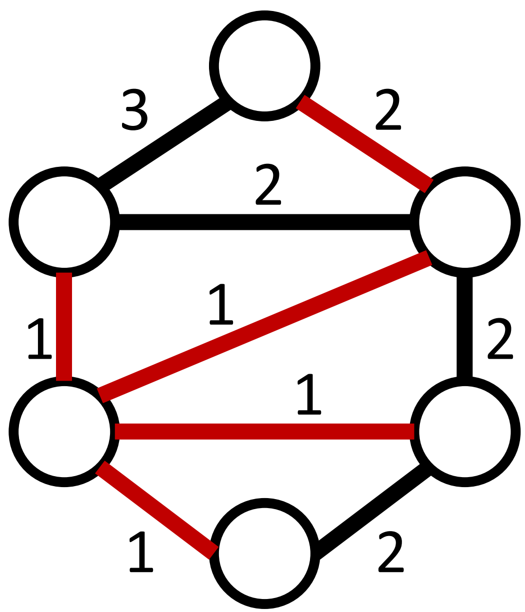
- An optimization problem always has a decision problem corresponding to it.

轉換問題的能力

Example

- The MST problem
  - Input: A connected, undirected, ***weighted*** graph G
  - Output: An acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight is ***minimized***.

最小生成樹：付出最少的成本，
把所有的點連起來。



$w(T) = 6$

→ decision problem

- The MST problem
  - Given a graph G and a constant c
  - if $w(T) < c$, return “yes”
  - otherwise, return “no”

所有的最佳化問題給他一個常數，如果找得到比這個常數小的解，就回答是

If the decision version of the optimization problem is difficult, the optimization version must be difficult.

D2是比較難的問題，所以可以解D2的演算法，也可以解D1

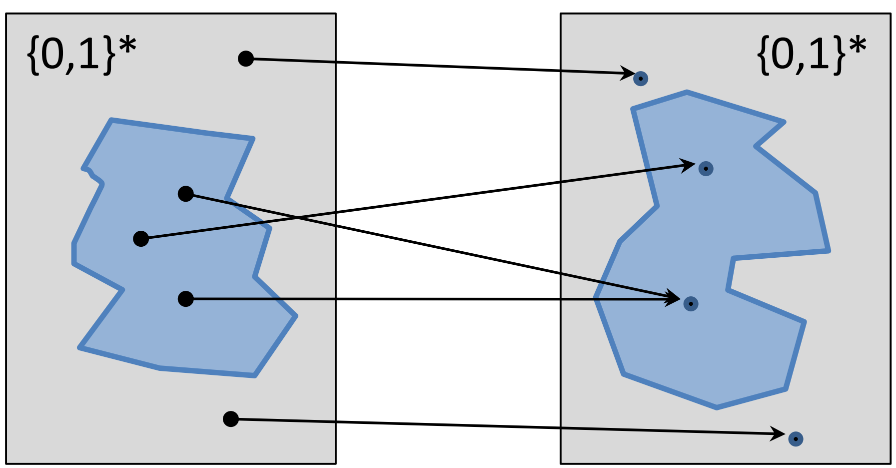
Reducibility $D_1 \leq_P D_2$

有一個問題D1可以轉換成D2，並且保證轉換的過程可以在polynomial time以內做完

p代表的是polynomial time

- A problem D_1 is ***polynomial-time reducible*** to a problem D_2 if there is a function t that transforms instances of D_1 to instances of D_2 such that
  - t maps all yes instances of D_1 to yes instances of D_2 and all no instances of D_1 to no instances of D_2 答案要一致
  - t is computed by a polynomial-time algorithm

轉換過程要在polynomial time以內



$$D_1 \leq_P D_2 \qquad \longrightarrow_P$$

- Problem D_1 can be reduced (in polynomial time) to problem D_2.
- Problem D_2 can be reduced (in polynomial time) from problem D_1.
- If problem D_2 can be solved in polynomial time algorithm, then problem D_1 can be solved in polynomial time.

Example 1

- D_1
  - Does a linear equation $bx + a = 0$ have an integer solution?

測資給a,b 解x

- D_2
  - Does a quadratic equation $cx^2 + bx + a = 0$ have an integer solution?

把D1的a,b測資帶進來，把c設成0就好，這個問題就可以轉換

- $D_1 \leq_P D_2$?

Example 1

不急著攻克新問題，先想想能不能轉換成舊問題，
就可以拿舊問題的演算法來解

- D_1
  - Does a linear equation $bx + a = 0$ have an integer solution?
- D_2
  - Does a quadratic equation $cx^2 + bx + a = 0$ have an integer solution?

D_1 is **no harder to solve** than D_2

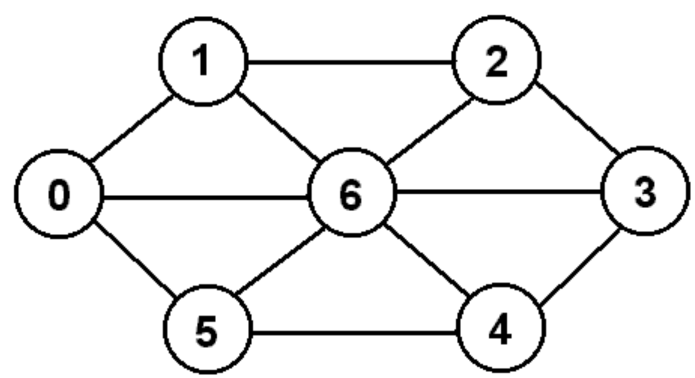
If x^* is a solution to D_2, it must be a solution to D_1.

If D_2 has a polynomial-time algorithm,
so does D_1.

Example 2: D_1

- Hamiltonian path problem
- Input
  - a graph G and two nodes x and y
- Output
  - whether or not G admits a path from x to y passing each node of G exactly once.

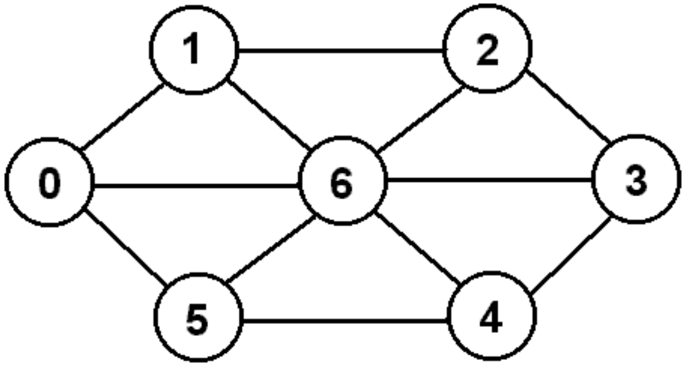
跳過



Example 2: D_2

- Longest path problem
- Input
  - a graph G and two nodes x and y.
- Output
  - a longest simple path in G from x to y.

跳過



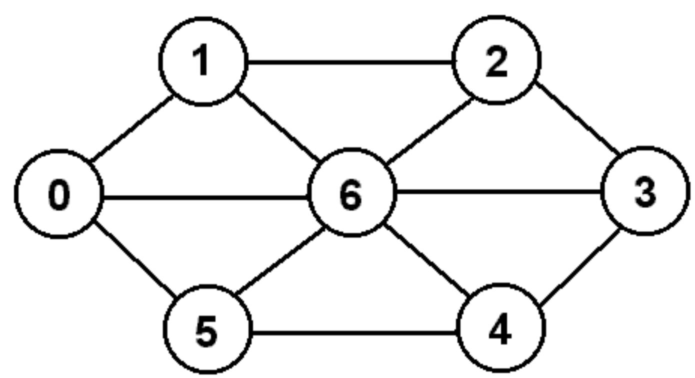
Hamiltonian path $\leq_P$ Longest path

跳過

- Let (G, x, y) be an input for the Hamiltonian Path problem.
- Suppose that the Longest-Path Problem has a polynomial-time algorithm A.
- We run $A(G, x, y)$. If the output path passing all nodes of G exactly once, we answer *yes* for the Hamiltonian Path problem; answer *no* otherwise.
- The Hamiltonian Path problem also has a polynomial-time algorithm!

Example 2 (decision version): D_1

- Hamiltonian path problem
- Input
  - a graph G and two nodes x and y
- Output
  - whether or not G admits a path from x to y passing each node of G exactly once.

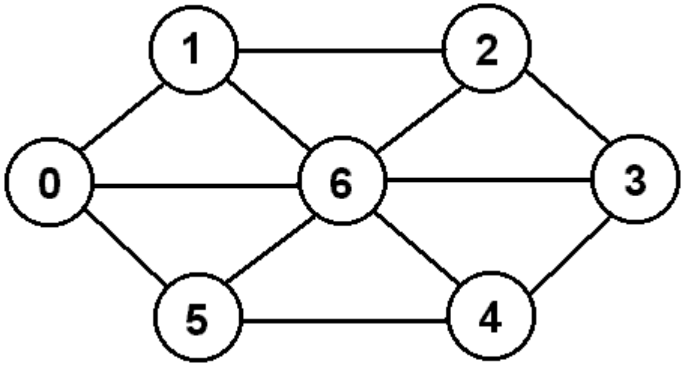


Example 2 (decision version): D_2

- Longest path problem
- Input
  - a graph G and two nodes x and y.
  - a constant c

D2的input 比D1多了一個c

- Output
  - whether or not there is a simple path in G from x to y whose length exceeds c.



試圖用longest path的演算法去解決hamiltonian path的問題

Hamiltonian path $\leq_P$ Longest path

D1可以轉換成D2

圖，起點，終點

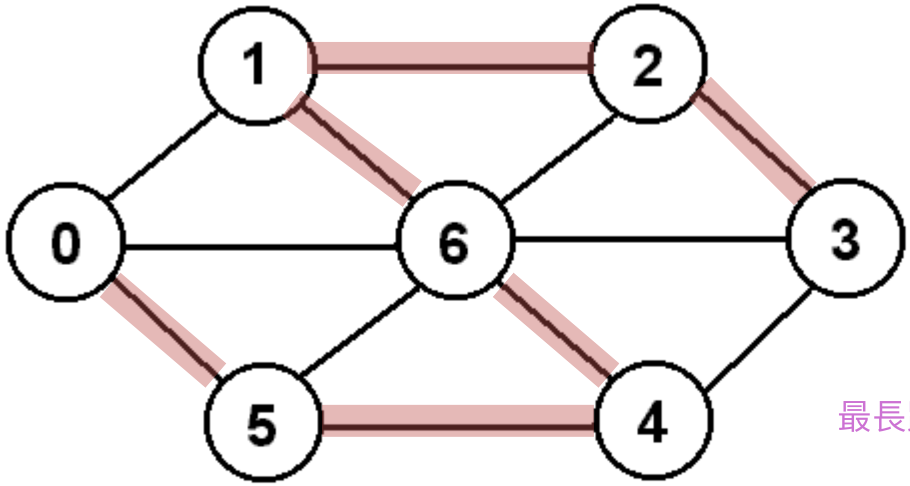
- Let (G, x, y) be an input for the Hamiltonian Path problem.
- Suppose that the Longest-Path Problem has a polynomial-time algorithm A. 假設D2用 A 演算法，已經有解了

差一個常數值c，填入節點數–1

- We run $A(G, x, y, ?)$. If it returns *yes*, we answer *yes* for the Hamiltonian Path problem; answer *no* otherwise.

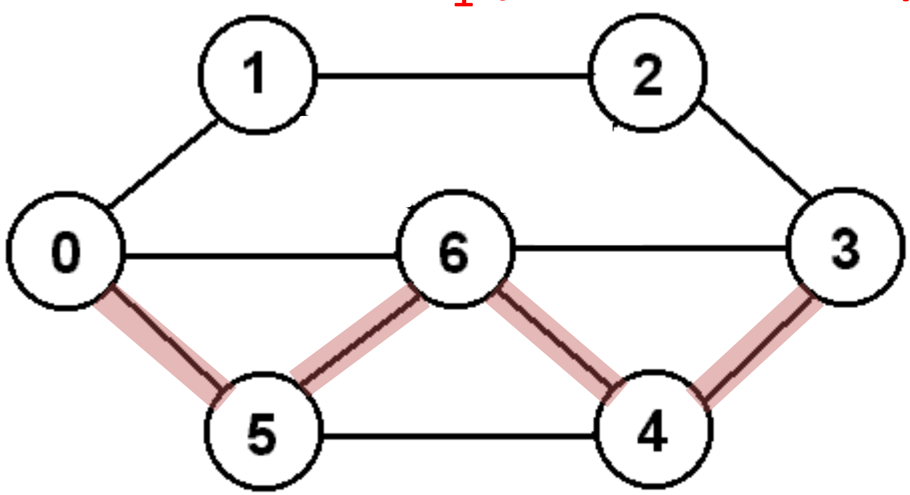
設這個c的目的是，要確保D2的演算法可以拿來解D1

- The Hamiltonian Path problem also has a polynomial-time algorithm!



最長路徑，而且長度大於等於c

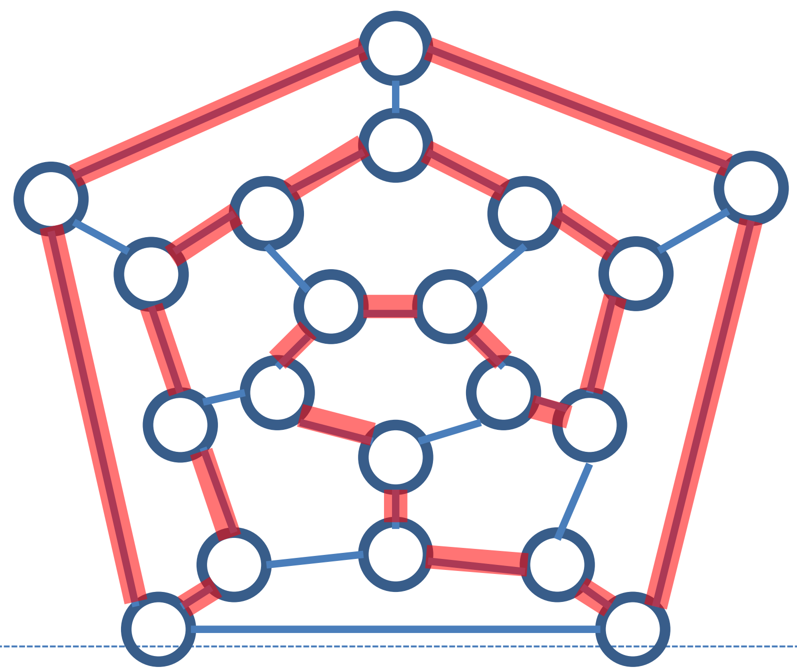
If D_2 (longest path) has a polynomial-time algorithm, so does D_1 (Hamiltonian path).



5/30 (二)
End

Example 3

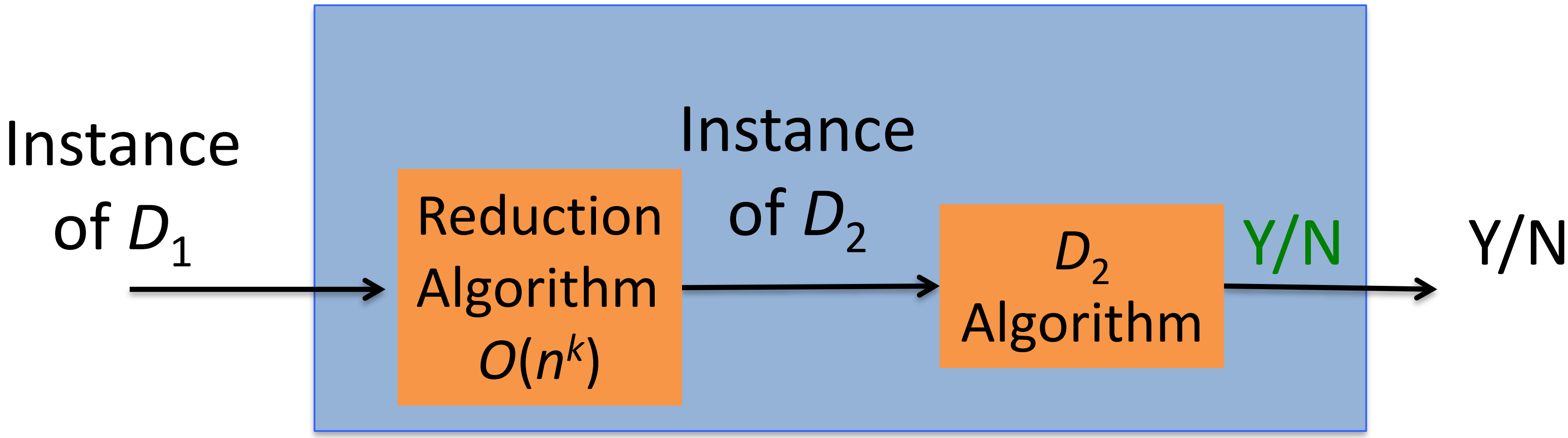
- Hamiltonian-cycle problem
  - Input: a graph $G = (V, E)$
  - Output: whether or not G has a hamiltonian-cycle (Y/N)



a simple cycle that contains each vertex in V

6/1 ([illegible])
START

$$D_1 \leq_P D_2$$



If D_2 has a polynomial-time algorithm, so does D_1.

把D1轉換成D2的過程。
又因為D2的問題難度較高，所以如果D2有解的話，這個演算法也可以拿來解D1。

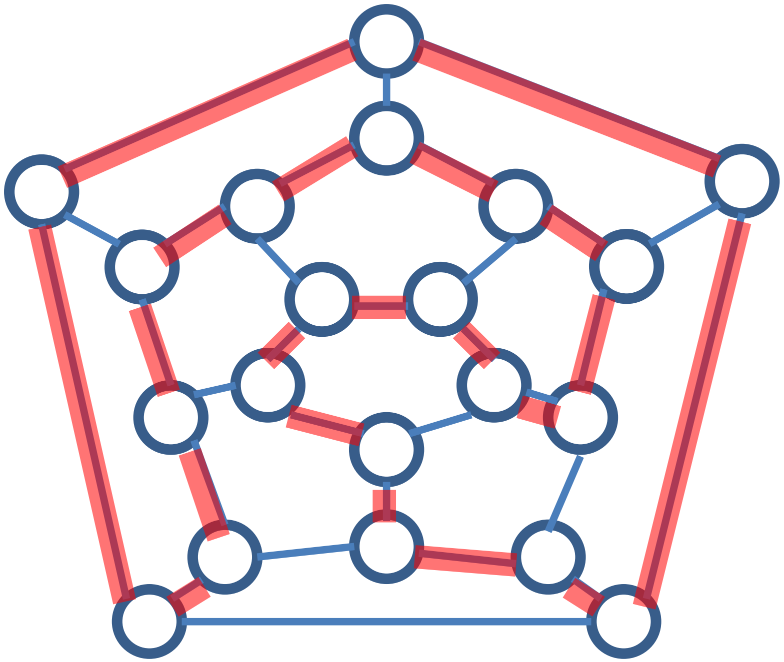
這個問題的邊上是有權重的

- The traveling-salesman problem (TSP)
  - A salesman must visit *n* cities.
  - Find a tour that visits each city exactly once and finishes at the city he starts from.
  - The cost (total number of edges on the tour) is minimum.

input有一個k，希望路徑總長可以小於等於k。
可以就回傳yes不行就no

  - Input: *G, c, k*
  - Output: whether or not there is a tour with cost at most *k* (Y/N)

HAM-CYCLE $\leq_P$ TSP



G

G, c, k

HAM-CYCLE $\leq_P$ TSP

這個轉換不一定是唯一的

這個k要設多少？
才能讓D2的解法
在D1也能用

- Let $G = (V, E)$ be an instance of HAM-CYCLE.
- Construct a complete graph $G' = (V, E')$
- Define the cost function c by

$$c(i, j) = \begin{cases} 0 & \text{if } (i, j) \in E, \\ 1 & \text{if } (i, j) \notin E. \end{cases}$$

polynomial time

輸入圖裡面原本就權重，就
設成1，沒有就設成0

- The instance of TSP is then $G', c, 0$.

G		$G', c, 0$
Yes	$\leftrightarrow$	Yes
No	$\leftrightarrow$	No

如果回答是yes，代表原圖中有
hamiltonian cycle

- If the TSP problem has a polynomial-time algorithm, so does the Hamiltonian-cycle problem!

如果D2（tsp）有解的話，那D1（hamiltonian cycle）就一定有解

Transitivity of Reductions

- If $D_1 \leq_P D_2$ and $D_2 \leq_P D_3$, then $D_1 \leq_P D_3$.

也可以再轉換成D3

P, NP, NP-hard, NP-complete

NP-hard

- A problem is NP-hard if it is ***at least as hard as*** all the problems in NP.

NP hard一定比NP問題難

NP-hard

沒有保證說可以馬上檢查答案的正確性

- A problem D is NP-hard if
  - every problem in NP is polynomial-time reducible to D.

$$p \leq_P D, \text{ for all } p \in \mathrm{NP}$$

- If D can be solved in polynomial time, then all the problem in NP can be solved in polynomial time.

『所有』的NP問題都可以轉換成NP hard問題

NP-complete

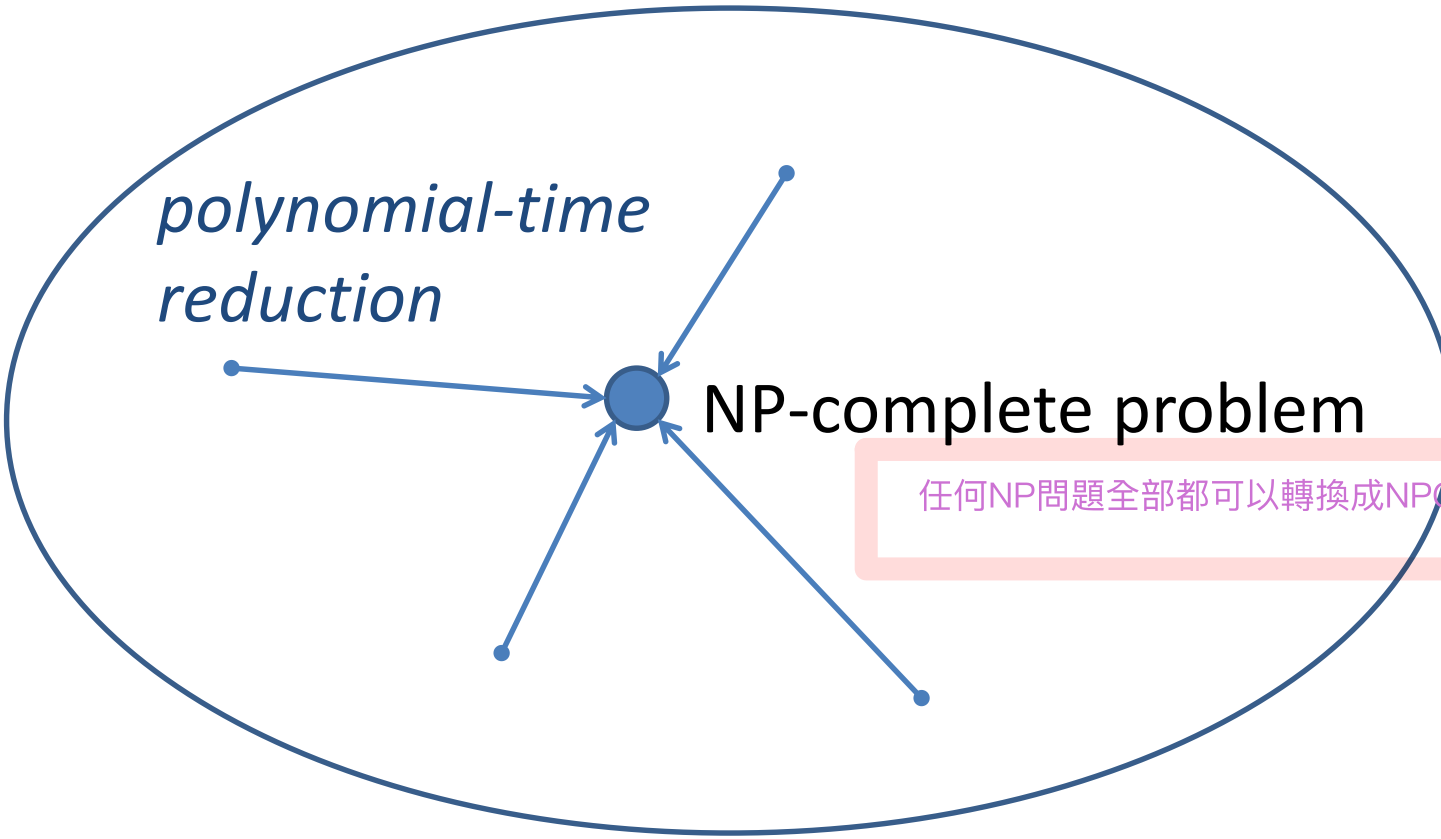
- A problem *D* is NP-complete if
  - every problem in NP is polynomial-time reducible to *D*, and
  - *D* belongs to NP.

『所有』的NP問題都可以轉換成某一個NP complete問題，所以只要某一個NPC有解，全部的NP都有解了

- If ***any*** NP-complete problem can be solved in polynomial time, then ***every*** problem in NP has a polynomial-time solution.
- NP-complete problems are the “hardest” problems in NP.

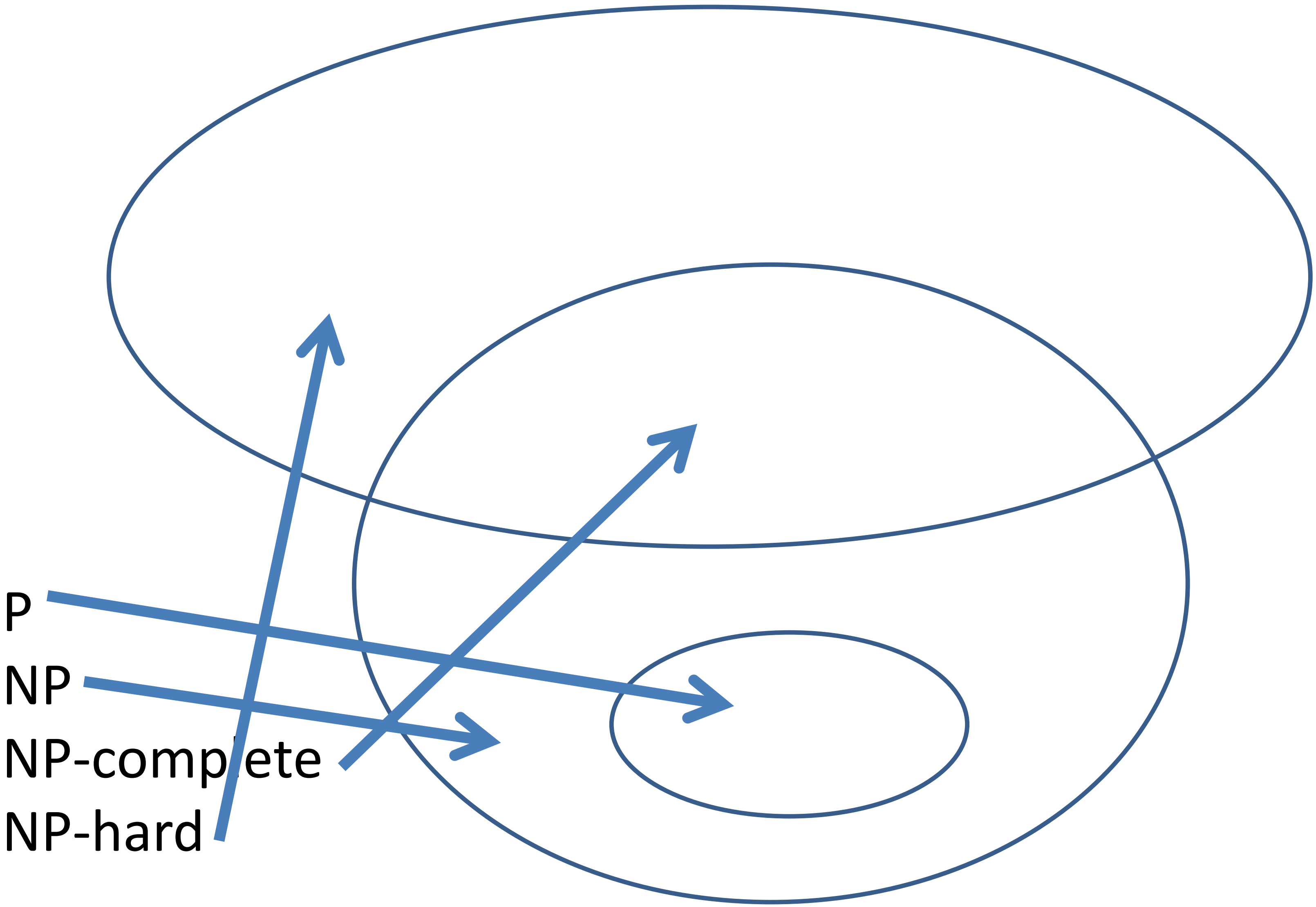
NPC是NP問題裡最難的

NP problems



任何NP問題全部都可以轉換成NPC問題

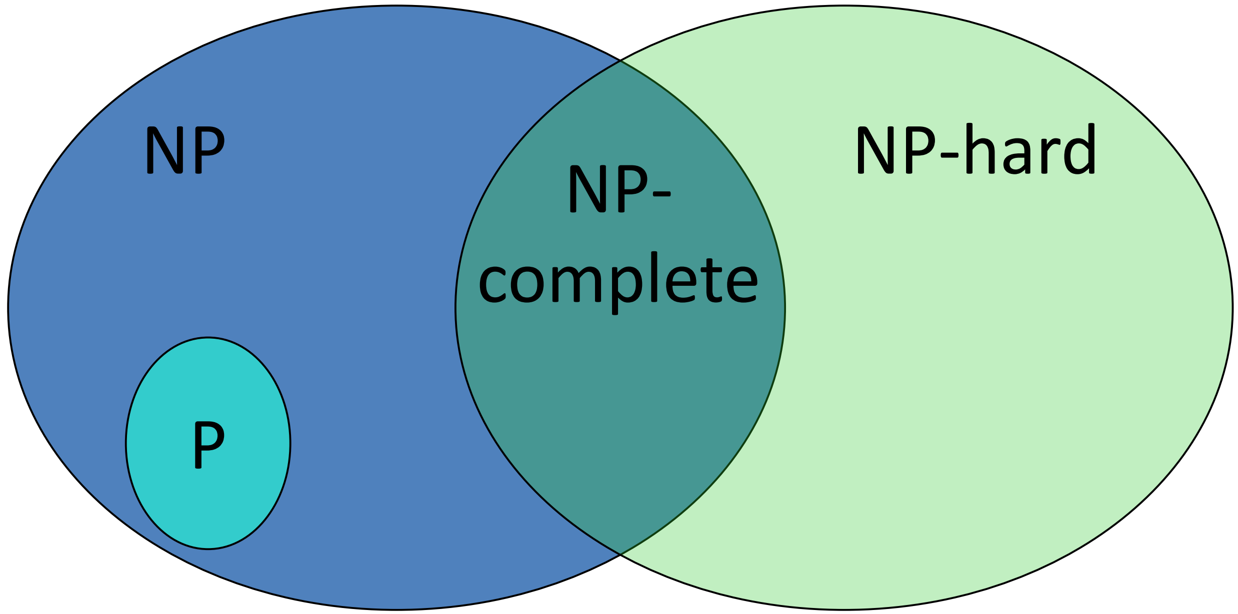
How most theoretical computer scientists view the relationships among P, NP, NP-complete, and NP-hard.



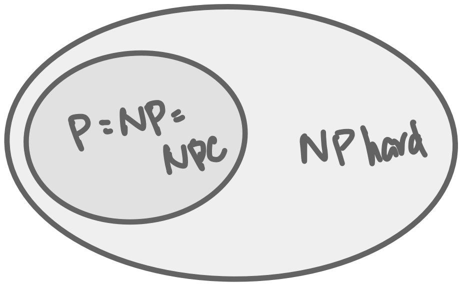
Supposing P ≠ NP

如果P跟NP是不同集合的話，關係圖如下：

npc一定是np–hard



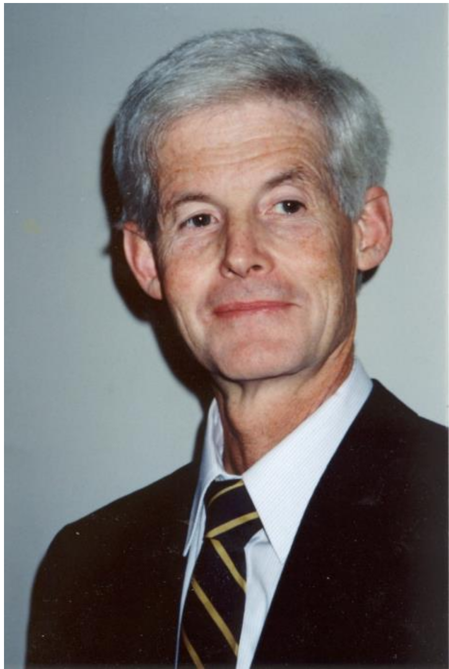
Supposing P = NP?



So, any NP-complete problems?

The first known NP-complete problem

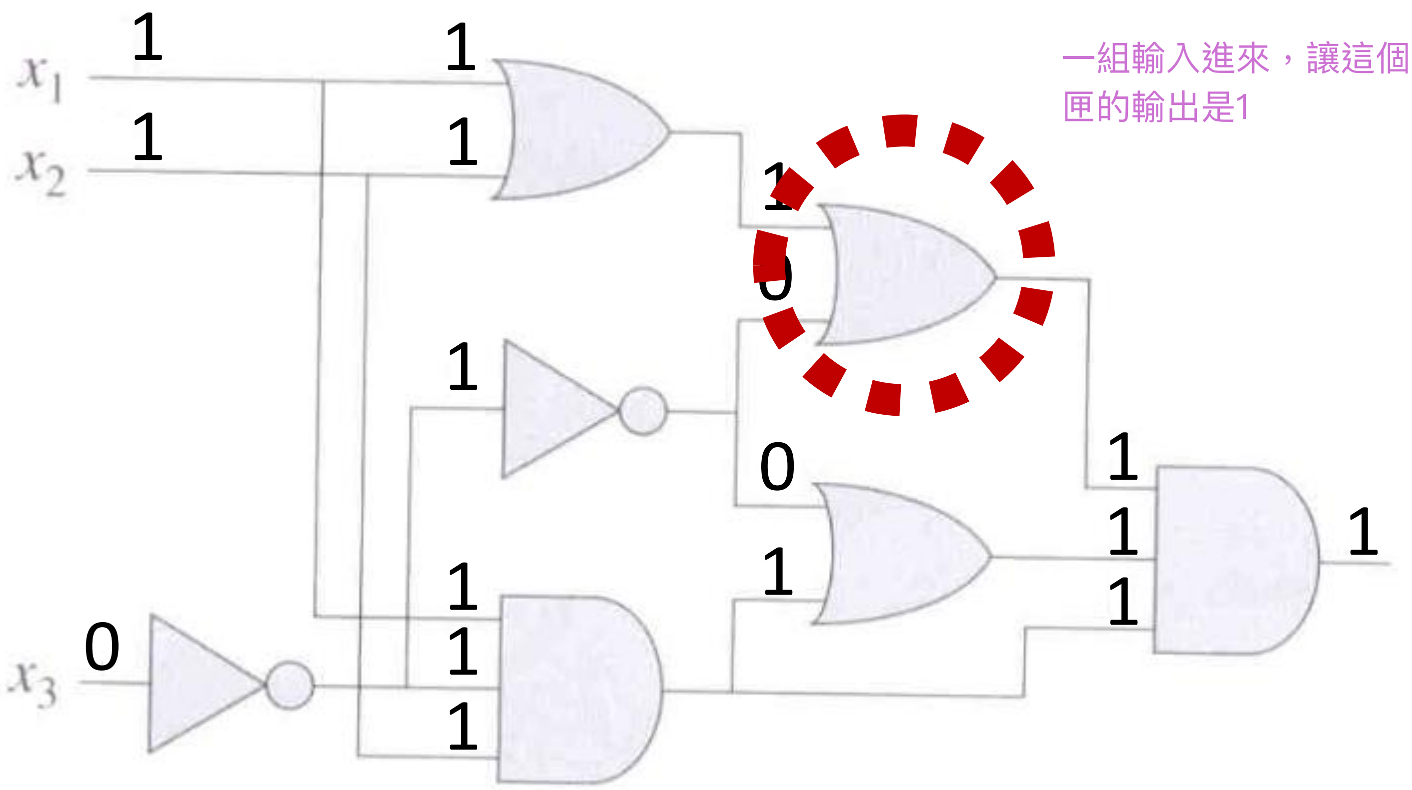
- The SAT (Satisfiability) problem SAT問題是世界上第一個NPC問題
- Shown by Stephen A. Cook in 1971
  - Born December 14, 1939
  - Currently a professor at University of Toronto
  - Turing Award, 1982



SAT Problem

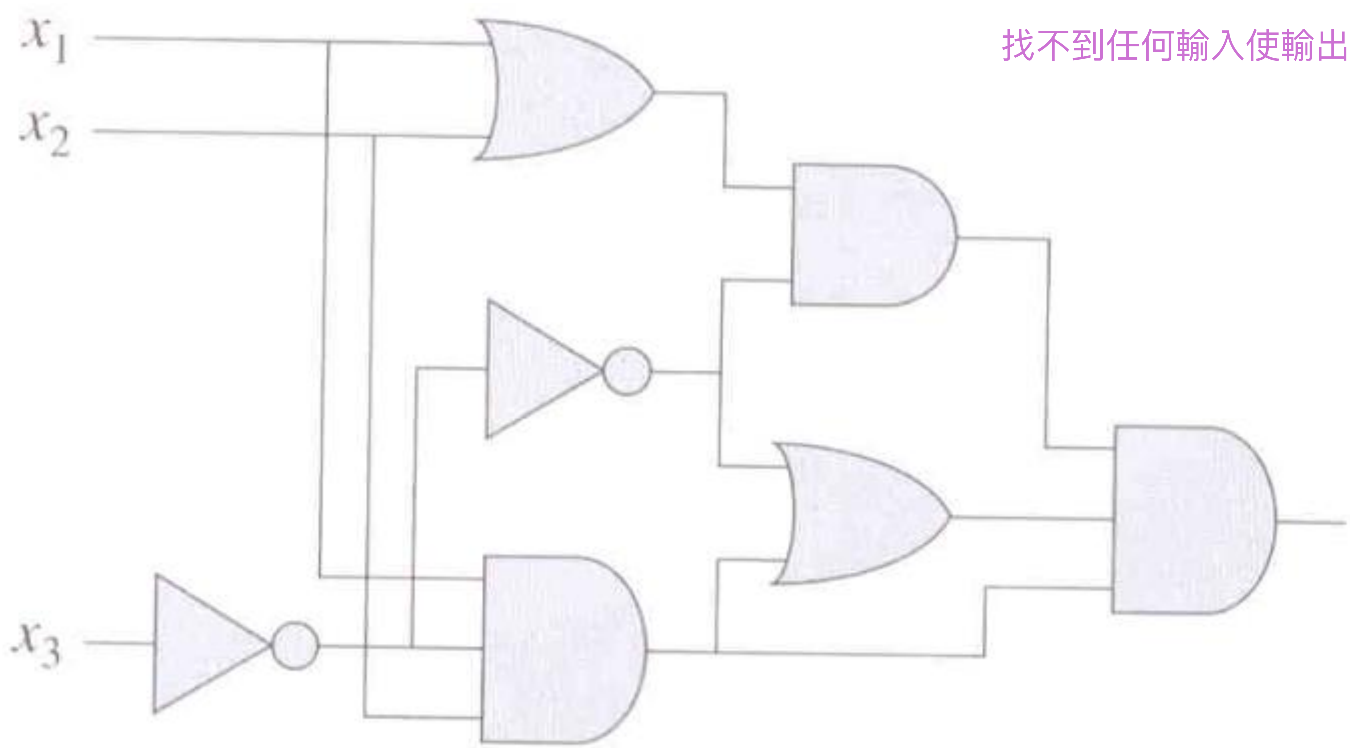
- Input
  - A boolean formula with variables
- Output
  - Yes/No: whether there is a truth assignment for the variables that satisfies the input boolean formula

Satisfiable? ***Yes!***



一組輸入進來，讓這個邏輯
匣的輸出是1

Satisfiable? *No!*



找不到任何輸入使輸出是1，就回答NO

Cook's contribution

- The SAT problem is clearly in NP.
- Cook proved that
  - If SAT can be solved in polynomial time, then **every** problem in NP is solvable in polynomial time.

所有的NP問題都是SAT問題

The Complexity of Theorem Proving Procedures, *ACM SIGACT Symposium on the Theory of Computing*, 1971.

SAT: a key to P versus NP

- If one proves that SAT can be solved by a polynomial-time algorithm, then NP = P.
- If somebody proves that SAT cannot be solved by any polynomial-time algorithm, then NP ≠ P.

Previous Final Exam

- Given
  - Y is a P problem
  - Z is a NP-complete problem
- Now, we have a new problem X, which is know to be NP.
  1. Show $Y \leq_P X$
  2. Show $X \leq_P Y$
  3. Show $Z \leq_P X$
  4. Show $X \leq_P Z$ ✓

4. 只能證明說他可以變成NPC問題，但不代表他本身是NPC問題

Q1: Q: 已知X是NP問題，想證明它是P問題，要做哪一個？ Ans: 2

To show that problem X is *computational tractable*

Q2: Q: 已知X是NP問題，想證明它是NPC問題，要做哪一個？ Ans: 3

To show that problem X is *NP-complete*

Proving problems NP-complete

- Claim: If Y is NP-complete and X is NP such that $Y \leq_P X$, then X is NP-complete.
- Given a new problem X, a general strategy for proving it NP-complete is
  1. Prove that X is NP.
  2. Select a problem Y known to be NP-complete.
  3. Prove that $Y \leq_P X$.

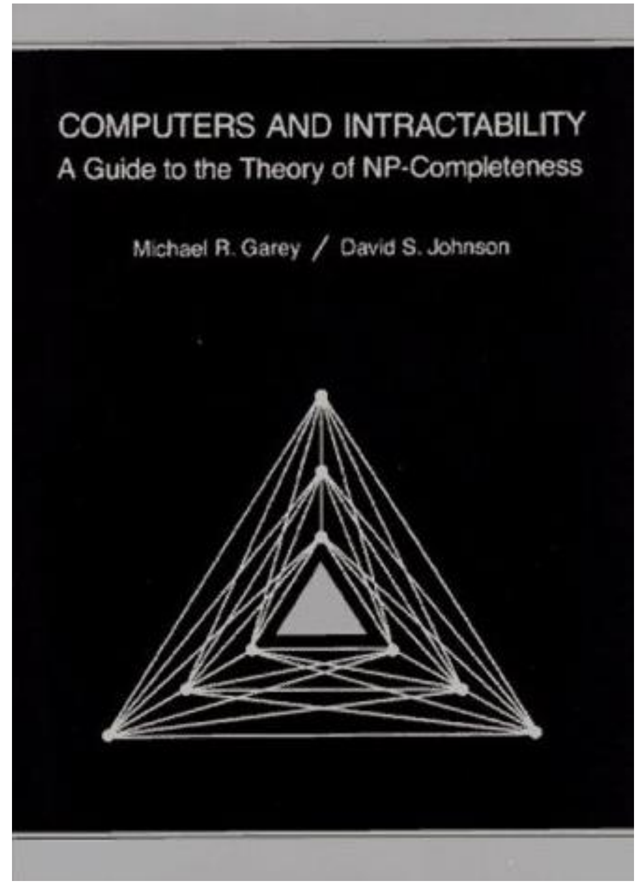
Other NP-complete problems?

- The clique problem
- The vertex-cover problem
- The Hamiltonian-cycle problem
- The traveling-salesman problem
- The longest path problem
- The partition problem
- The subset sum problem

- A compendium of NP optimization problems
  - http://www.nada.kth.se/~viggo/problemlist/compendium.html

Garey and Johnson

- A huge collection of NP-complete problems



Extended reading

- R. M. Karp, "Combinatorics, Complexity, and Randomness," Communications of the ACM, pp. 98–109, Vol. 29, No. 2, February 1986.

Closing remarks

Techniques

- Brute force / exhaustive search
- Divide and conquer
- Dynamic programming
- Greedy
- Iterative improvement
- Problem reduction

Fundamentals

- Complexity analysis
- NP-completeness
- Graph

Closing remarks

- Thank you all for your attention and participation to the class!
- Please be prepared for the final exam. I hope I will ***not*** see you in this class next year.