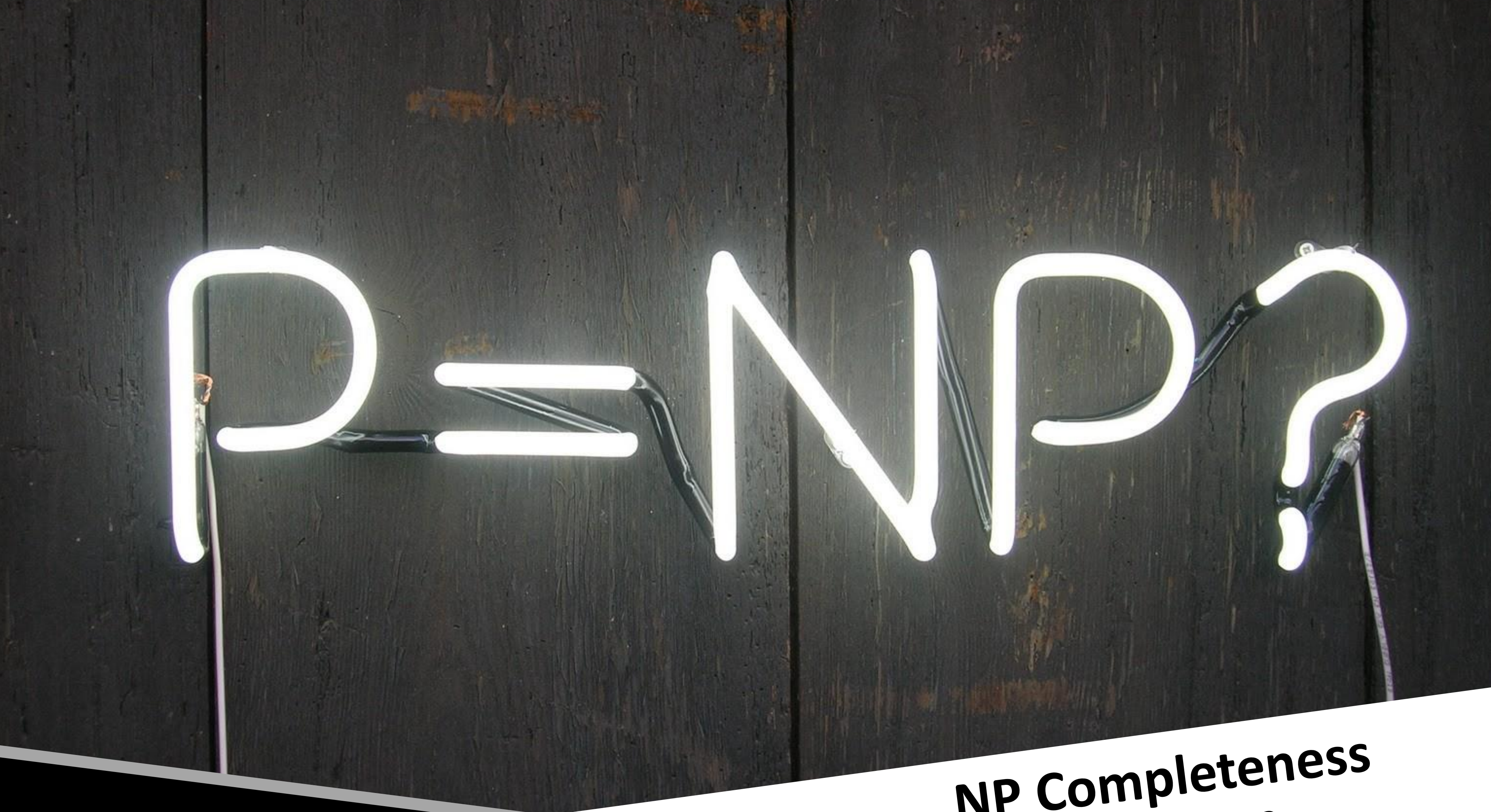


Algorithm Design and Analysis

NP Completeness

Dec 6^{th}, 2018

YUN-NUNG (VIVIAN) CHEN HTTP://ADA.MIULAB.TW

國立臺灣大學
National Taiwan University

Slides credited from Hsueh-I Lu & Hsu-Chun Hsiao

Outline

- Decision Problems v.s. Optimization Problems
- Complexity Classes
  - P v.s. NP
  - NP, NP-Complete, NP-Hard

Algorithm Design & Analysis

- Design Strategy
  - Divide-and-Conquer
  - Dynamic Programming
  - Greedy Algorithms
  - Graph Algorithms
- Analysis
  - Amortized Analysis
  - NP-Completeness

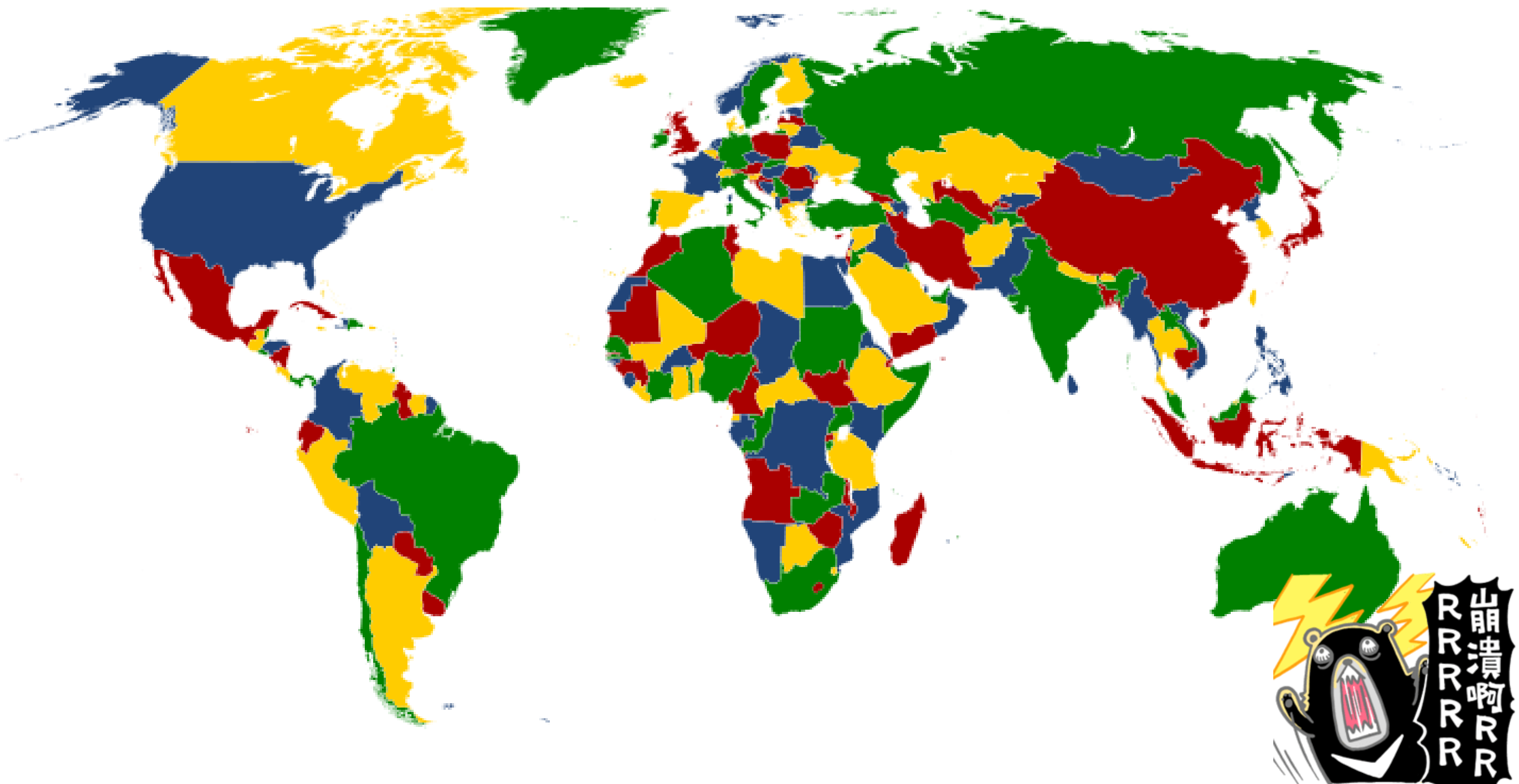
Polynomial Time Algorithms

- For an input with size *n*, the worst-case running time is $O(n^k)$ for some constant *k*

- Problems that are solvable by polynomial-time algorithms as being *tractable*, *easy*, *or efficient*
- Problems that require superpolynomial time as being *intractable*, or *hard*, *or inefficient*

Four Color Problem

- Use total four colors s.t. the neighboring parts have different colors



Four Color Problem (after 100 yrs)

- Finally proven (with the help of computers) by Kenneth Appel and Wolfgang Haken in 1976
  - Their algorithm runs in $O(n^2)$ time
- First major theorem proved by a computer
- Open problems remain...
  - Linear time algorithms to find a solution
  - Concise, human-checkable, mathematical proofs

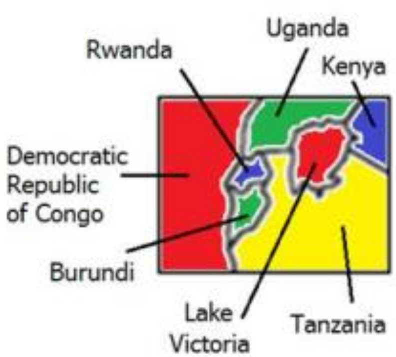
Planar *k*-Colorability

- Given a planar graph *G* (e.g., a map), can we color the vertices with *k* colors such that no adjacent vertices have the same color?
- $k = 1$?
- $k = 2$?
- $k = 3$?
- $k \geq 4$?

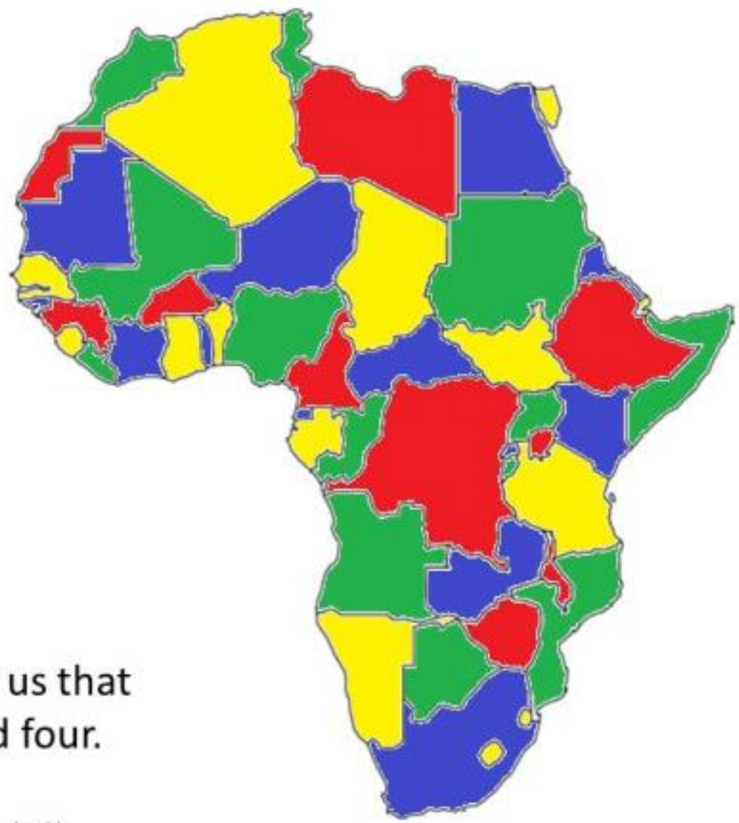
How hard is it when $k = 3$?
Can we know its level of difficulty before solving it?

Planar *k*-Colorability

It turns out that FOUR will do.



This section shows us that we certainly need four.



9

Decision Problems v.s. Optimization Problems

Decision Problems

- Definition: the answer is simply “yes” or “no” (or “1” or “0”)
  - MST: Given a graph $G = (V, E)$ and a bound K, is there a spanning tree with a cost at most K?
  - KNAPSACK: Given a knapsack of capacity C, a set of objects with weights and values, and a target value V, is there a way to fill the knapsack with at least V value?

Optimization Problems

- Definition: each feasible solution has an associated value, and we wish to find a feasible solution with the best value (maximum or minimum)
  - MST-OPT: Given a graph $G = (V, E)$, find the *minimum* spanning tree of G
  - KNAPSACK-OPT: Given a knapsack of capacity C and a set of objects with weights and values, fill the knapsack so as to *maximize* the total value

Which is Easier? Why?

How to convert an optimization problem to a related decision problem?

Imposing a (lower or upper) bound on the value to be optimized

Difficulty Levels

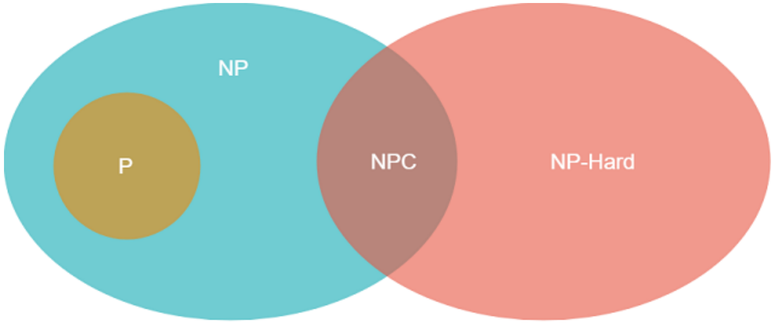
- Every optimization problem has a decision version that is no harder than the optimization problem.

A_{opt}: given a graph, find the length of the shortest path

A_{dec}: given a graph, determine whether there is a path ≤ k

- ➡ Using A_{opt} to solve A_{dec}
  - check if the optimal value ≤ k, constant overhead
- ⬅ Using A_{dec} to solve A_{opt}
  - apply binary search on the value range, logarithmic overhead

14 P v.s. NP



Textbook Chapter 34 – NP-Completeness

Algorithm Design

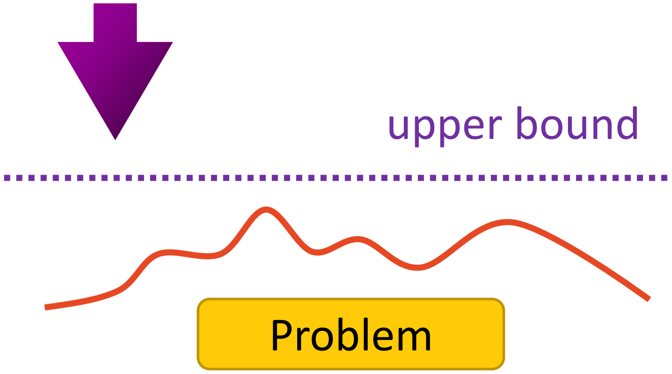
- Algorithmic design methods to solve problems efficiently (polynomial time)
  - Divide and conquer
  - Dynamic programming
  - Greedy
- "Hard" problems without known efficient algorithms
  - Hamilton, knapsack, etc.

Complexity Classes

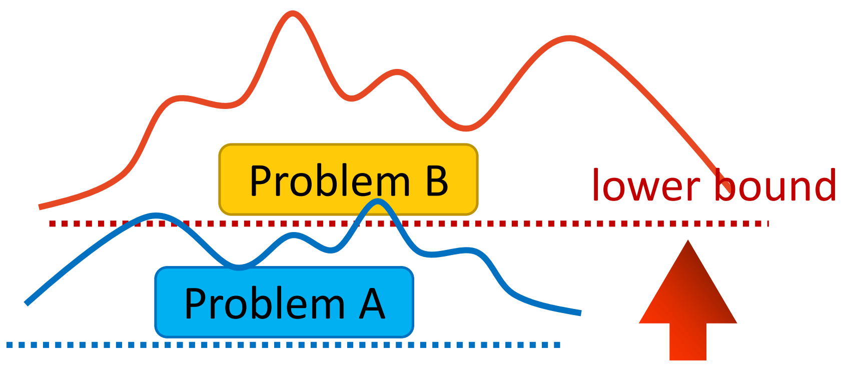
- Can we decide whether a problem is “too hard to solve” before investing our time in solving it?
- Idea: decide which *complexity classes* the problem belongs to via reduction
  - 已知問題A很難。若能證明問題B至少跟A一樣難，那麼問題B也很難。

To Solve v.s. Not to Solve

- Algorithm design
  - Design algorithms to solve computational problems
  - Mostly concerned with *upper bounds* on resources



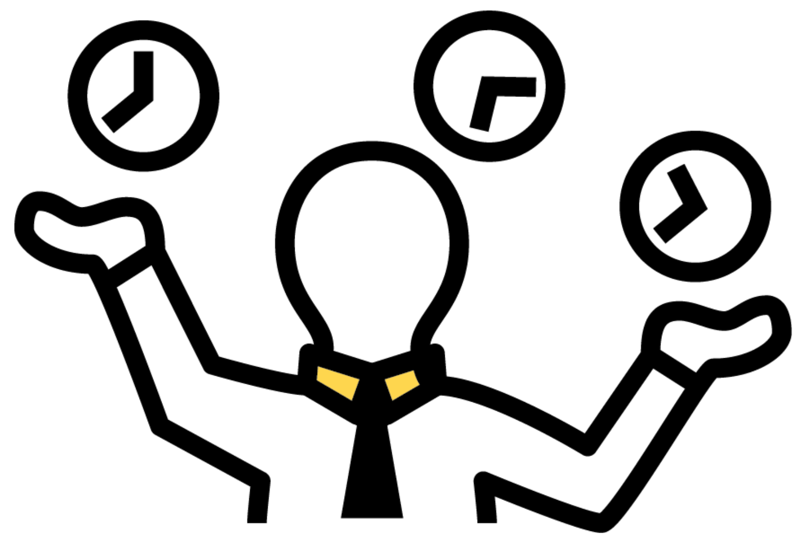
- Complexity theory
  - Classify problems based on their difficulty and identify relationships between classes
  - Mostly concerned with *lower bounds* on resources



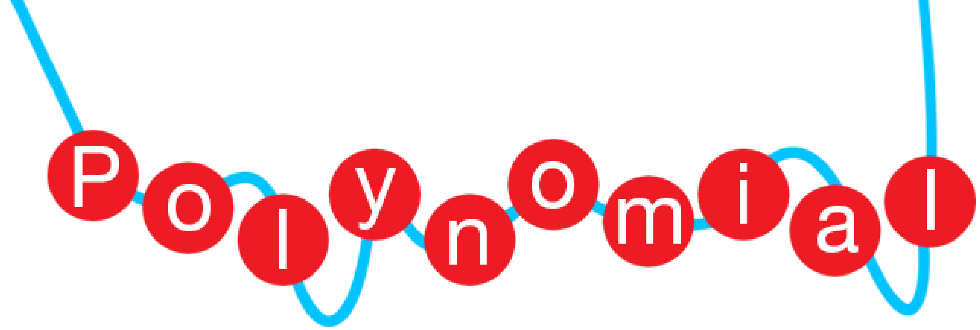
Problem B is no easier than A

Complexity Classes

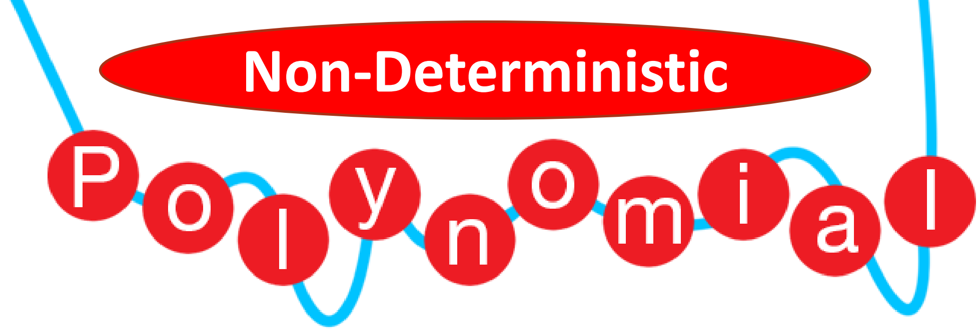
- A complexity class is “a set of problems of related resource-based complexity”
  - Resource = time, memory, communication, ...
- Focus: *decision problems* and the resource of *time*



P



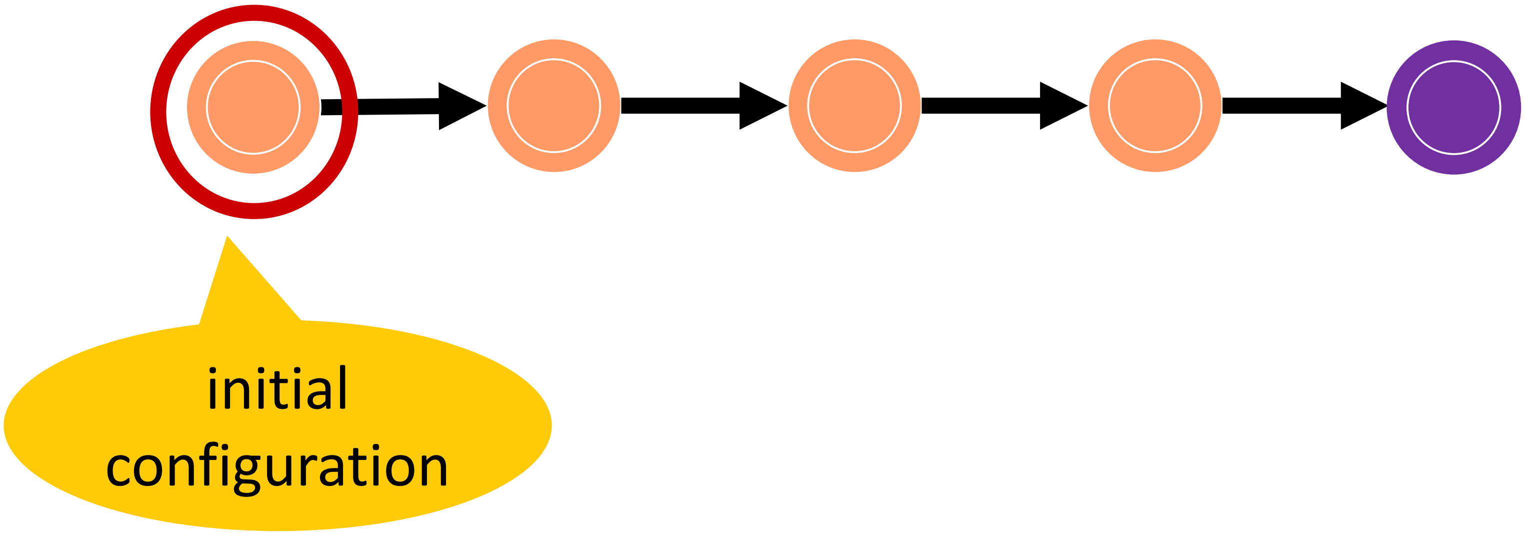
- The class **P** consists of all the problems that can be solved in *polynomial time*.
  - Sorting
  - Exact string matching
  - Primes
  - …
- Polynomial time algorithm
  - For inputs of size *n*, their worst-case running time is $O(n^k)$ for some constant *k*



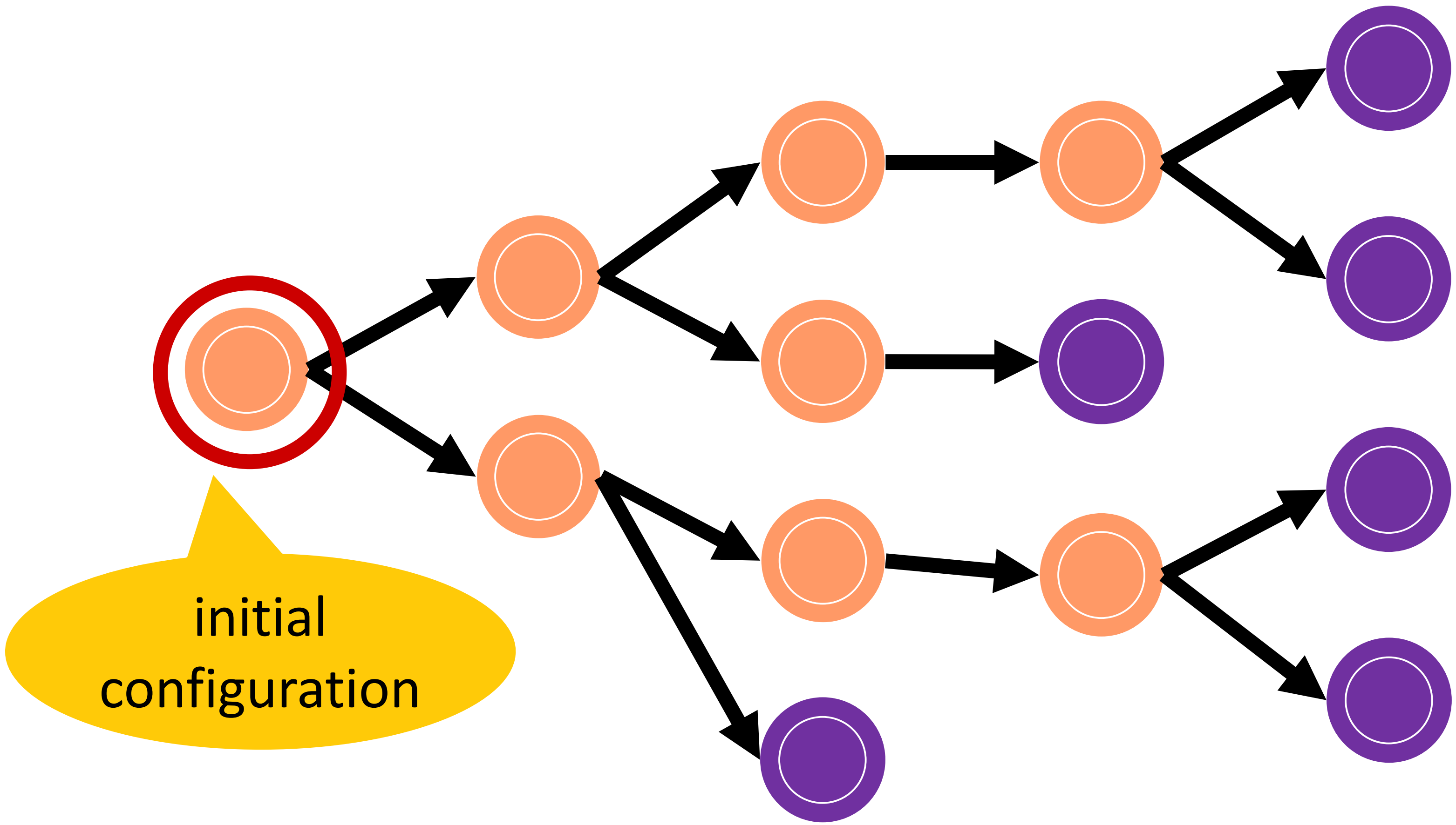
NP

- NP consists of the problems that can be solved in *non-deterministically polynomial time*.
- NP consists of the problems that can be "verified" in polynomial time.
- P consists of the problems that can be solved in (deterministically) polynomial time.

Deterministic Algorithm



Non-Deterministic Algorithm



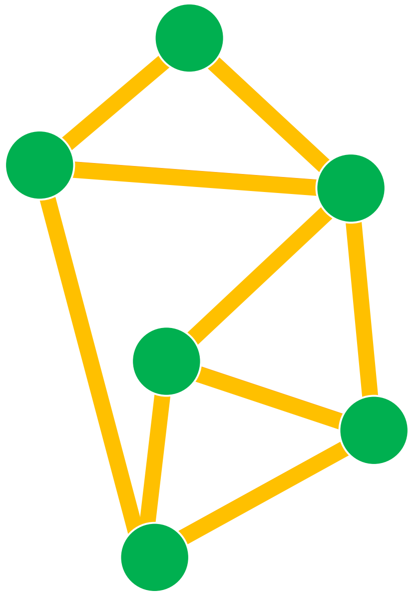
Non-Deterministic Bubble Sort

```
Non-Deterministic-Bubble-Sort(n)
  for i = 1 to n
    for j = 1 to n - 1
      if A[j] < A[i+1] then
        Either exchange A[j] and A[i+1] or do nothing
```

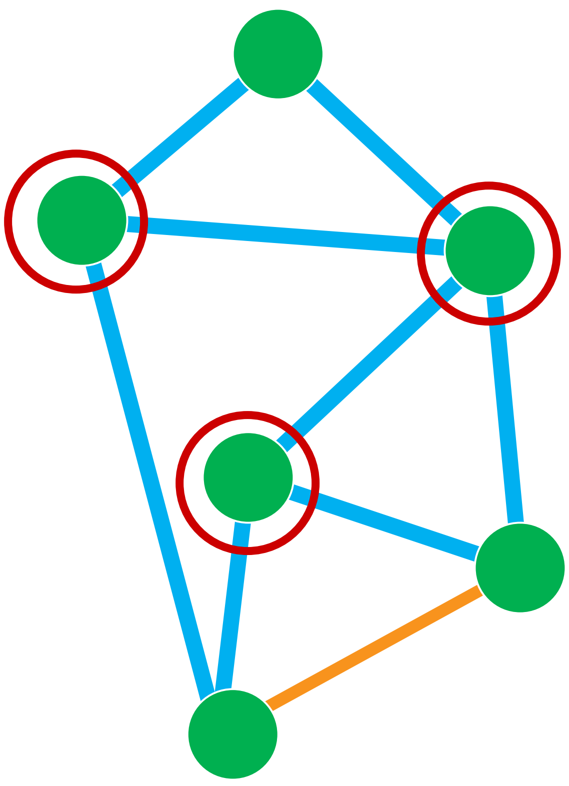
This is not a randomized algorithm.

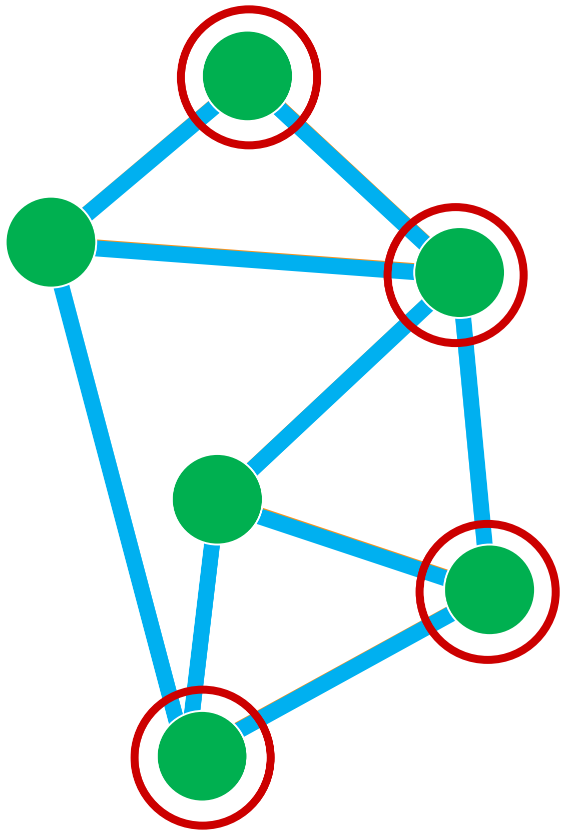
Vertex Cover Problem (路燈問題)

- Input: a graph *G*
- Output: a smallest vertex subset of *G* that covers all edges of *G*.
- Known to be NP-complete



Illustration





Vertex Cover (Decision Version)

- Input: a Graph *G* and an integer *k*.
- Output: Does *G* contain a vertex cover of size no more than *k*?

- Original problem → optimization problem
  - 原先的路燈問題是要算出放路燈的方法
- Yes/No → decision problem
  - 問*k*盞路燈夠不夠照亮整個公園

Non-Deterministic Algorithm

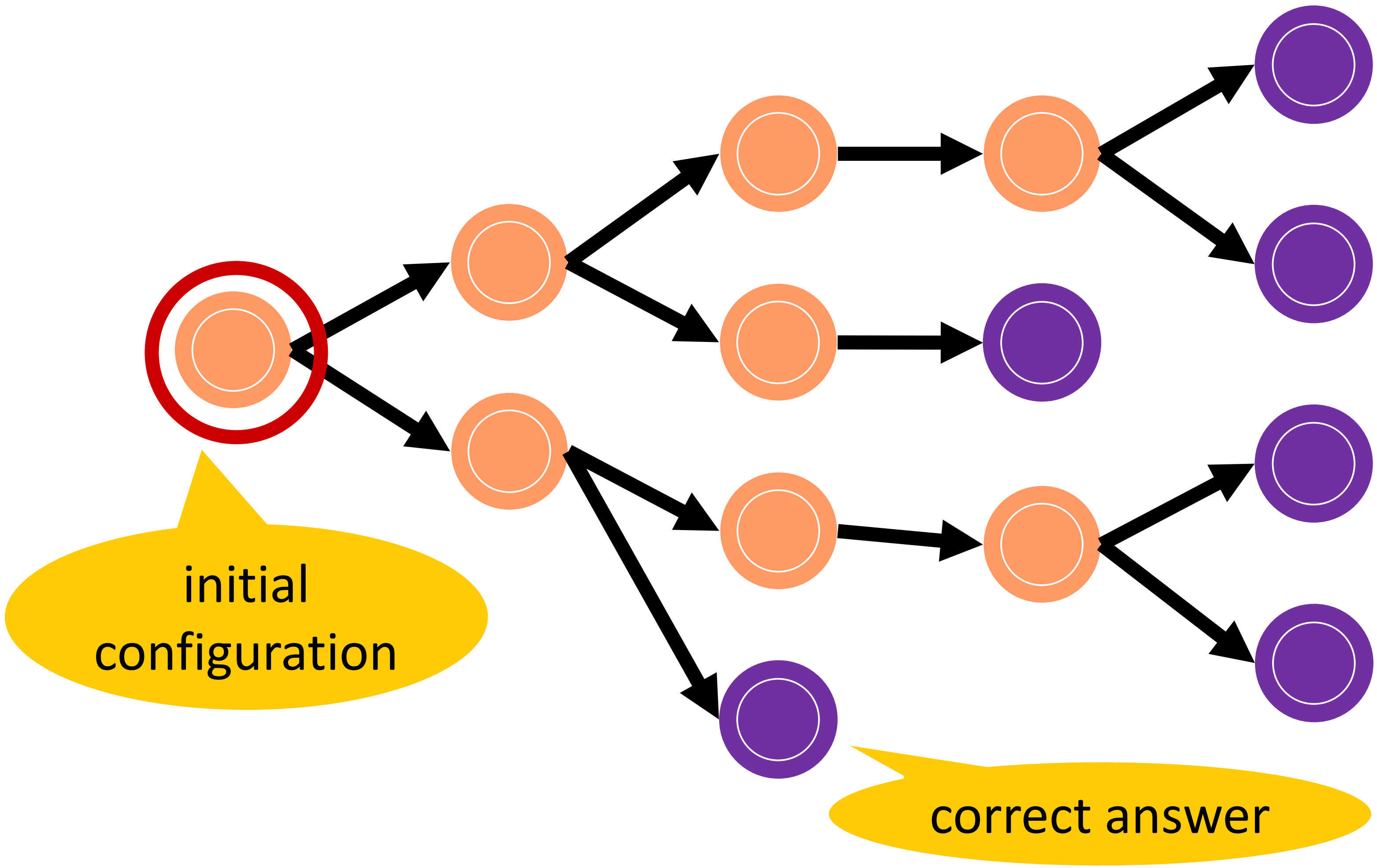
```
Non-Deterministic-Vertex-Cover(G, k)
  set S = {}
  for each vertex x of G
    non-deterministically insert x to S
  if |S| > k
    output no
  if S is not a vertex cover
    output no
  output yes
```

Algorithm Correctness

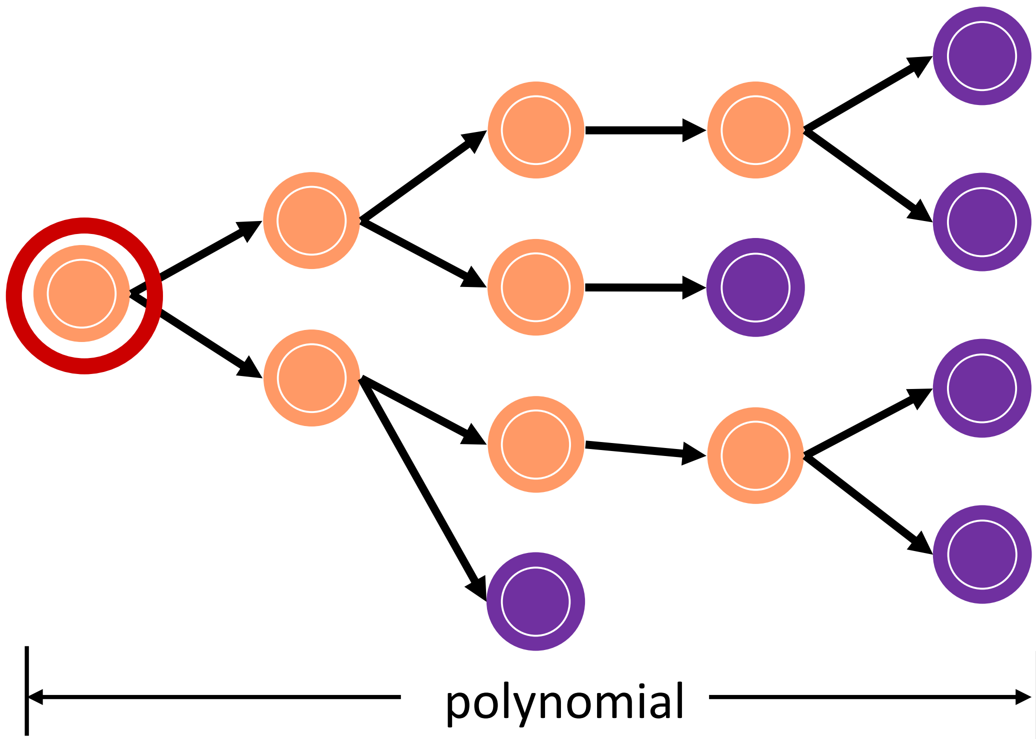
```
Non-Deterministic-Vertex-Cover(G, k)
  set S = {}
  for each vertex x of G
    non-deterministically insert x to S
  if |S| > k
    output no
  if S is not a vertex cover
    output no
  output yes
```

- If the correct answer is *yes*, then there is a computation path of the algorithm that leads to *yes*.
  - 至少有一條路是對的
- If the correct answer is *no*, then all computation paths of the algorithm lead to *no*.
  - 每一條路都是對的

Non-Deterministic Problem Solving



Non-Deterministic Polynomial



“solved” in non-deterministic polynomial time
= “verified” in polynomial time

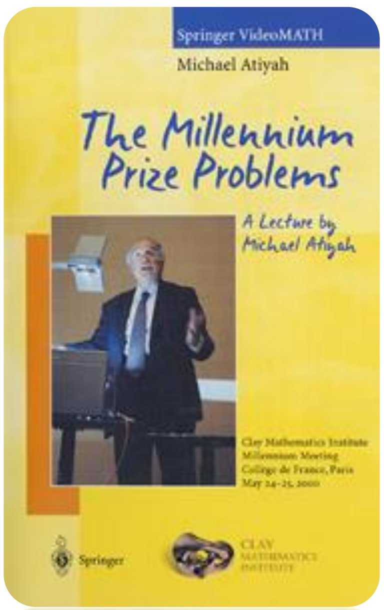
P ⊆ NP or NP ⊆ P?

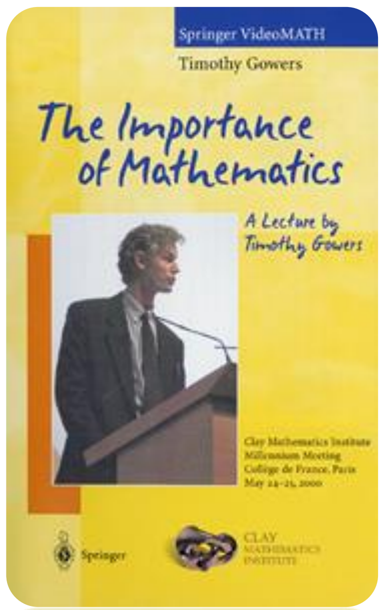
- P ⊆ NP
  - A problem solvable in polynomial time is verifiable in polynomial time as well
- Any NP problem can be solved in (deterministically) exponential time?
  - Yes

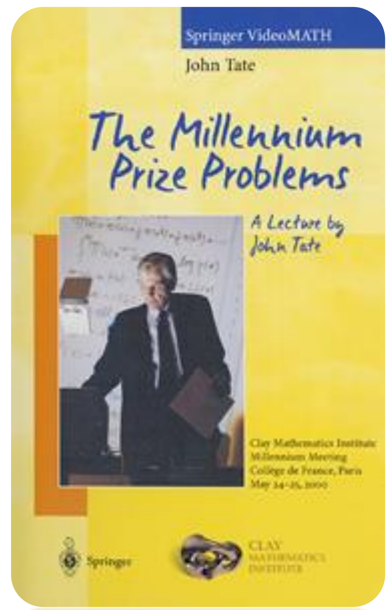
- Any NP problem can be solved in (deterministically) polynomial time?
  - Open problem

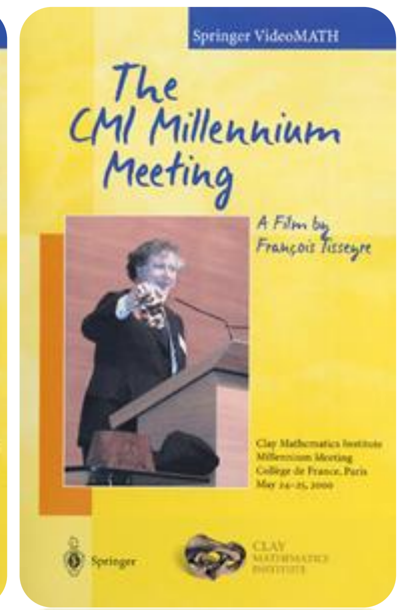
US$1,000,000 Per Problem

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



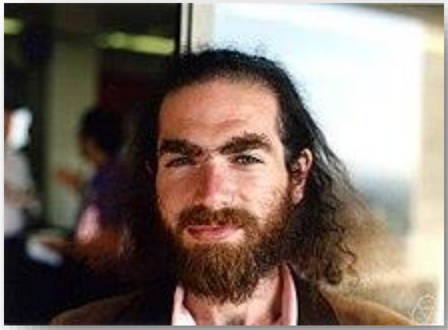






Millennium Problems

- Yang–Mills and Mass Gap
- Riemann Hypothesis
- P vs NP Problem
- Navier–Stokes Equation
- Hodge Conjecture
- Poincaré Conjecture (solved by Grigori Perelman)
- Birch and Swinnerton-Dyer Conjecture



Grigori Perelman
Fields Medal (2006), declined
Millennium Prize (2010), declined

Vinay Deolalikar

- Aug 2010 claimed a proof of P is not equal to NP.

If P = NP

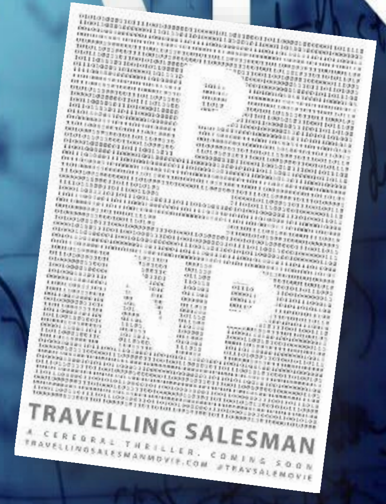
- problems that are verifiable → solvable

- public-key cryptography will be broken

"If P = NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss..." – Scott Aaronson, MIT

Widespread belief in P ≠ NP

TRAVELLING SALESMAN



Travelling Salesman (2012)

A movie about P = NP

Best Feature Film in Silicon Valley Film Festival 2012

37

NP, NP-Complete, NP-Hard

NP-Hardness

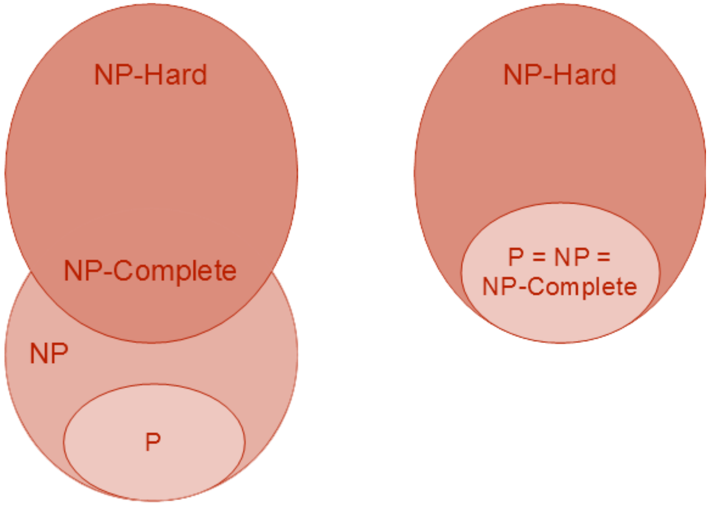
- A problem is NP-hard if it is as least as hard as all NP problems.
- In other words, a problem *X* is NP-hard if the following condition holds:
  - If *X* can be solved in (deterministic) polynomial time, then all NP problems can be solved in (deterministic) polynomial time.

NP-Completeness (NPC)

- A problem is NP-complete if
  - it is NP-hard and
  - it is in NP.
- In other words, an NP-complete problem is one of the "hardest" problems in the class NP.
- In other words, an NP-complete problem is a hardness representative problem of the class NP.
- Hardest in NP → solving one NPC can solve all NP problems ("complete")
- It is wildly believed that NPC problems have no polynomial-time solution → good reference point to judge whether a problem is in P
  - We can decide whether a problem is "too hard to solve" by showing it is as hard as an NPC problem
  - We then focus on designing approximate algorithms or solving special cases

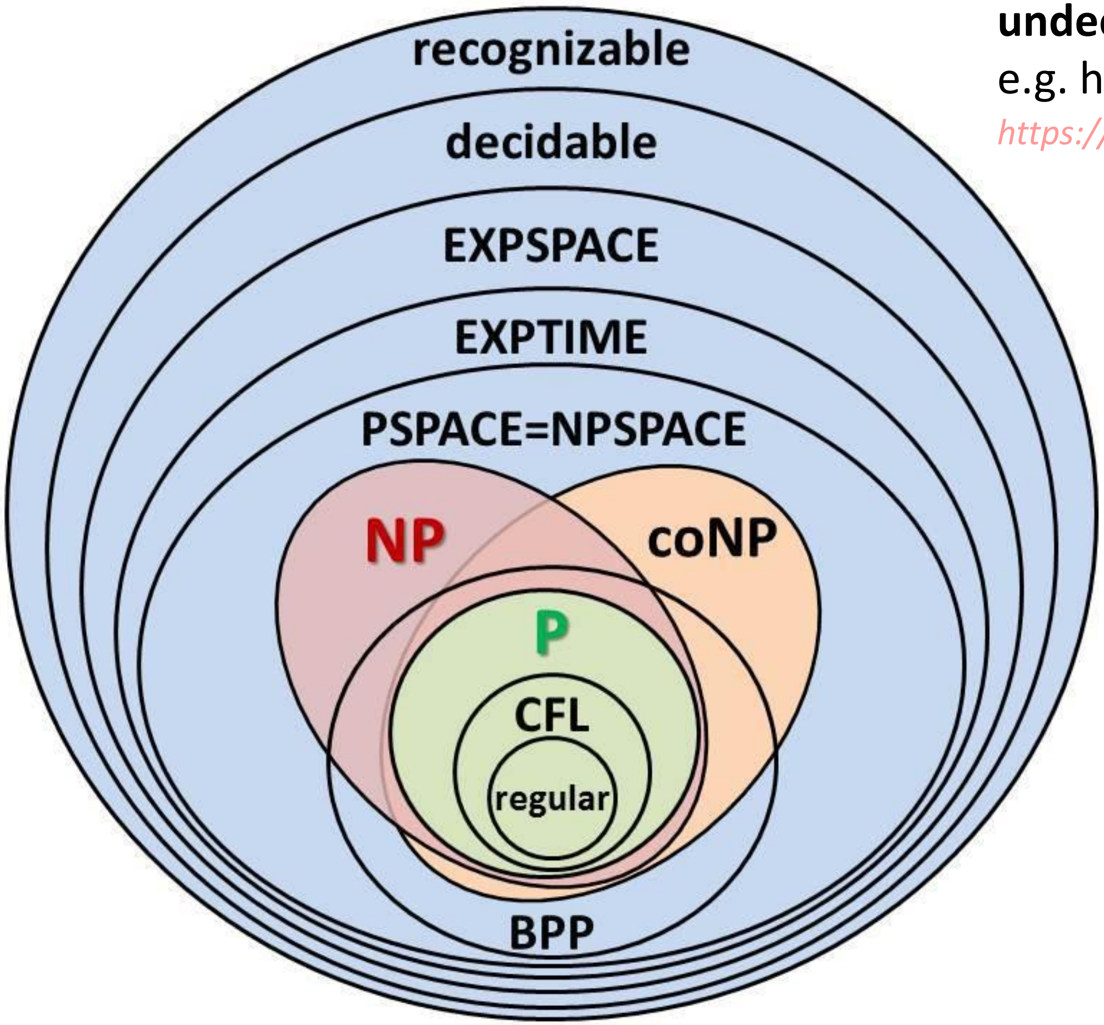
Complexity Classes

- Class P: class of problems that can be solved in $O(n^k)$
- Class NP: class of problems that can be verified in $O(n^k)$
- Class NP-hard: class of problems that are "at least as hard as all NP problems"
- Class NP-complete: class of problems in both NP and NP-hard



P ≠ NP   P = NP

More Complexity Classes



undecidable: no algorithm;
e.g. halting problem
https://www.youtube.com/watch?v=wGLQiHXHWNk

42

To Be Continued…

Question?

Important announcement will be sent to @ntu.edu.tw mailbox
& post to the course website

Course Website: http://ada.miulab.tw

Email: ada-ta@csie.ntu.edu.tw