Monash University
Faculty of Information Technology

FIT2014 Theory of Computation

Lecture 25
Nondeterministic Polynomial time, and the class NP

slides by Graham Farr

Overview

- Deciding and Verifying
- Certificate
- The class NP
- Proving membership of NP
- Examples of languages in NP
- P $\subseteq$ NP
- The P-versus-NP problem
- Deciders for languages in NP
- Nondeterministic Polynomial-time Turing machines

Deciding and Verifying

Deciding if a string belongs to a language or not

versus

Verifying that a string belongs to a language (if it does)

Deciding and Verifying

P is intended to contain languages which are *efficiently decidable*

- i.e., you can efficiently decide whether something is in the language or not

Recall:

A **decider** for a language L is a TM that takes a string x and

- always halts
- if $x \in L$, the TM accepts
- if $x \notin L$, the TM rejects.

P is the set of languages for which there is a polynomial-time decider.

Deciding and Verifying

Consider:

$$\{\,\text{people who can kick a football}\,\}$$

How do you verify that a person can kick a football?

Give them a ball and get them to try to kick it.

This procedure is like a decider.

It enables you to *decide* whether or not they can kick a football.

Deciding and Verifying

Now consider:

$$\{ \text{university graduates} \}$$

How do you verify that a person is a graduate?

Can't do it just by meeting them, testing abilities etc.

There is no efficient decider for this set.

But you can verify it if you have their degree certificate.

Hard to verify that someone is not a graduate.

Deciding and Verifying

A **verifier** for a language L is a TM that takes, as input, two strings x and y,

- always halts
- if x is in L, there exists y such that the TM accepts
- if x is not in L, every y makes the machine reject.

y is called a **certificate**.

x is accepted if and only if it has a certificate which can be verified.

A **polynomial-time verifier** is a verifier with time complexity polynomial in n, where $n = |x|$.

- i.e., $O(n^k)$, for some fixed k.

NP

NP is the set of languages for which there is a polynomial-time *verifier*.

NP stands for

Non-deterministic Polynomial time

(for reasons to be given later).

NP is intended to contain languages for which membership can be efficiently verified, with the aid of an appropriate certificate.

Proving membership of NP

To show a language is in NP, you need to:

- specify the *certificate*
- give a polynomial-time verifier (as an *algorithm*)
- *prove that it is a verifier* for the language
- prove that it is *polynomial time*.

Proving membership of NP

Proof that { 3-colourable graphs } is in NP.

Given: graph G
Certificate: a function $f : V(G) \rightarrow \{\text{Red}, \text{White}, \text{Black}\}$

Verification:

```
For each edge $uv$ of $G$
{
    Look up $f(u)$ and $f(v)$.
    // ... these are the colours given to the endpoints $u, v$ of this edge
    Check that $f(u) \neq f(v)$.
    If so, continue. If not, Reject and halt.
    // ... endpoints must get different colours
}
If loop completes with no edge rejected, then Accept and halt.
```

Proving membership of NP

Claim 1:

This is a verifier for $\{$3-colourable graphs$\}$.

Proof:

G is in $\{$ 3-colourable graphs$\}$
if and only if
there exists a function $f : V(G) \to \{\text{Red}, \text{White}, \text{Black}\}$ such that,
for each edge uv, we have $f(u) \neq f(v)$
if and only if
there exists a certificate such that our verifier accepts G.

End of proof of Claim 1.

Proving membership of NP

Claim 2:
Verifier takes polynomial time, in size of input.

Proof:

Main loop: $\#$ iterations $=$ $\#$ edges $=$ m, say.

For each edge: look up each endpoint in the certificate.
Suppose certificate is given as a list of colours, one for each vertex.
The vertex gives the position in the list.

Looking up the colour of each endpoint takes $O(n)$ time, where $n := \#$ vertices.
Checking whether $f(u) \neq f(v)$ takes constant time.
So, total time $\leq m \cdot n \cdot \text{constant} = O(mn)$.
So it takes polynomial time, in size of G.

End of proof of Claim 2.

$\square$

Proving membership of NP

So we have proved that {3-colourable graphs} is in NP.

Remarks:

- Some of these time estimates are loose upper bounds.
- Better estimates are often possible.
  (E.g., how long does it take to look something up in an array of size n?)
- But if our objective is to show that something is in NP, then all we need to show is that the time complexity of verification is bounded above by a polynomial (i.e., $O(n^k)$, for some fixed k).

Some languages in NP

For each of the examples we give, ask:

- What is the certificate?
- How do you verify it?

Examples:

- the set of 2-colourable graphs
- the set of 3-colourable graphs
- $\{(G, k) : G$ is a k-colourable graph$\}$

Some languages in NP

- the set of composite numbers

$$\{\, x \in \mathbb{N} \;:\; \exists y, z \in \mathbb{N} \text{ such that } 1 < y < x,\ 1 < z < x, \text{ and } x = y \cdot z \,\}$$

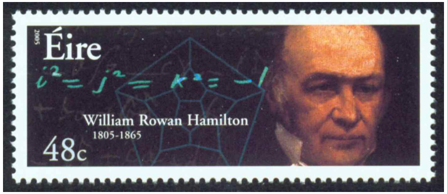
- SATISFIABILITY:
  the set of satisfiable Boolean expressions in Conjunctive Normal Form
- 2-SAT
  - exactly two literals in each clause
  - see the end of the previous lecture
- 3-SAT
  - exactly three literals in each clause

Some languages in NP

- the set of Eulerian graphs
  - An **Euler tour** in a graph *G* is a closed walk along edges that uses each edge exactly once.
  - A graph is **Eulerian** if it contains an Euler tour.

Leonhard Euler (1707–1783)
https://mathshistory.st-andrews.ac.uk/Biographies/Euler/

- the set of Hamiltonian graphs
  - A **Hamiltonian circuit** in a graph *G* is a circuit which includes each vertex exactly once. (note: a circuit doesn't repeat any vertex or edge)
  - A graph is **Hamiltonian** if it contains a Hamiltonian circuit.



William Rowan Hamilton (1805–1865)
https://mathshistory.st-andrews.ac.uk/Biographies/Hamilton/

Some languages in NP

GRAPH ISOMORPHISM:

$$\{(G, H) : G \text{ is isomorphic to } H\}$$

G is **isomorphic** to H if there is a bijection $f : V(G) \to V(H)$ such that,
for all $u, v \in V(G)$,

$$u \text{ is adjacent to } v \text{ in } G \quad \text{if and only if} \quad f(u) \text{ is adjacent to } f(v) \text{ in } H.$$

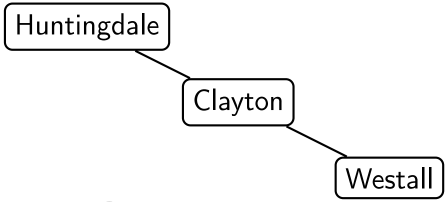
We write: $G \cong H$

Such a bijection is an **isomorphism**.

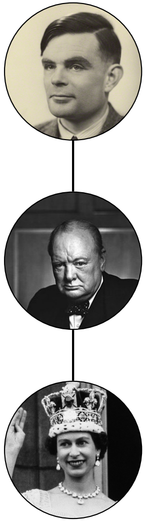
Informally: G and H are the same, apart from renaming vertices.

GRAPH ISOMORPHISM (continued)

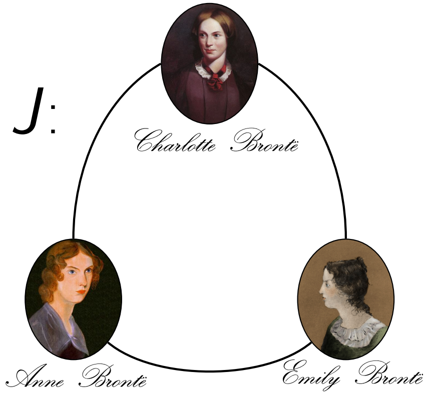
G:



H:



J:



$$G \cong H$$

$$G \ncong J$$
$$H \ncong J$$

Some languages in NP

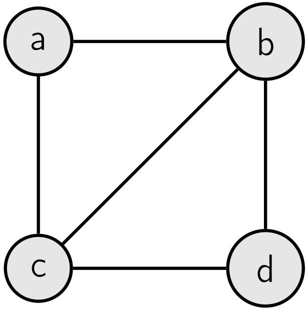
VERTEX COVER $:=$ $\{(G, k) : G$ has a vertex cover of size $\leq k\}$

A **vertex cover** in a graph $G = (V, E)$ is a set X of vertices such that every edge has at least one endpoint in X.

$$\forall uv \in E \;\; (u \in X) \vee (v \in X)$$

In this graph:

{a, b, c}	is a vertex cover
{b, c}	is a vertex cover
{a, b, c, d}	is a vertex cover
{a, b}	is NOT a vertex cover



Some languages in NP

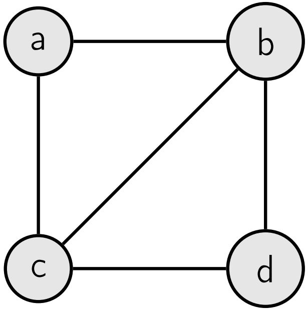
INDEPENDENT SET $:=$ $\{(G, k) : G \text{ has an independent set of size} \geq k\}$

An **independent set** in a graph $G = (V, E)$ is a set X of vertices such that no edge has both endpoints in X.

$$\forall uv \in E \;\; (u \notin X) \vee (v \notin X)$$

In this graph:

$\emptyset$	is an independent set
{b}	is an independent set
{a, d}	is an independent set
{c, d}	is NOT an independent set



What is the relationship between vertex covers and independent sets?

Some languages in NP

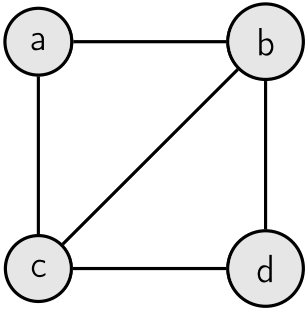
$$\text{CLIQUE} \quad := \quad \{(G, k) : G \text{ has a clique of size} \geq k\}$$

A **clique** in a graph $G = (V, E)$ is a set X of vertices such that every pair of vertices in X are adjacent.

$$\forall u \in X \ \forall v \in X : \ uv \in E$$

In this graph:

$\emptyset$	is a clique
{a}	is a clique
{a, b}	is a clique
{a, b, c}	is a clique
{a, b, d}	is NOT a clique



What is the relationship between independent sets and cliques?

P and NP

Theorem. P $\subseteq$ NP.

Proof. (outline)
For any L in P, there is a polynomial-time decider for L.
Turn this into a verifier:

```
Input:  string $x$
Certificate:  any other string $y$.      // ...but ignore it!
Run the $L$-decider on $x$.
If the decider accepts:  Accept.
If the decider rejects:  Reject.
```

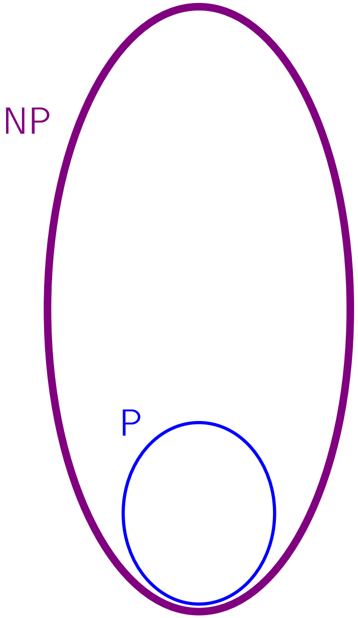
Then explain why this is a polynomial-time verifier for L. $\square$

P and NP

Conjecture. P $\neq$ NP

- The biggest open problem in Computer Science.
- One of the biggest open problems in Mathematics.
- Fame and glory await the solver . . .
- . . . and a Millennium Prize ($US 1 million) from the Clay Mathematics Institute `http://www.claymath.org/millennium-problems`
- But be careful: many false solutions have appeared, and continue to appear.

P and NP



In NP, not known to be in P:

SATISFIABILITY, 3-SAT,
HAMILTONIAN CIRCUIT,
3-COLOURABILITY,
VERTEX COVER,
INDEPENDENT SET, . . .

GRAPH ISOMORPHISM,

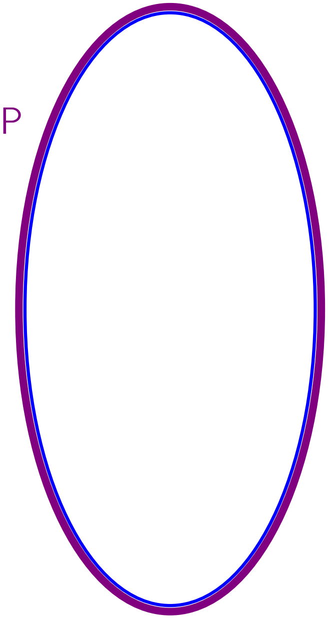
INTEGER FACTORISATION, . . .

In P:

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
CONNECTED GRAPHS,
SHORTEST PATH,
PRIMES,
Invertible matrices,
. . . ,
All Context-Free Languages,
All Regular Languages.

P and NP

If P = NP



SATISFIABILITY, 3-SAT,
HAMILTONIAN CIRCUIT,
3-COLOURABILITY,
VERTEX COVER,
INDEPENDENT SET, . . .

GRAPH ISOMORPHISM,

INTEGER FACTORISATION, . . .

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
CONNECTED GRAPHS,
SHORTEST PATH,
PRIMES,
Invertible matrices,
. . . ,
All Context-Free Languages,
All Regular Languages.

P and NP

Theorem.
Any language in NP can be decided in time $O(2^{n^K})$ for some constant K.

Idea of Proof.

Let L be any language in NP.

It has a polynomial-time verifier, V.

Construct from this a decider for L. How?

Decider does an exhaustive search of all possible certificates,
to see if one of them gets the input accepted.

Prove it's a decider for L, and has the claimed time complexity.

P and NP

The decider for L in detail:

```
Input:    $x$
For each certificate $y$:
    Call verifier $V$ on input $x$, certificate $y$.
    If $V$ accepts, then Accept, else continue.
Reject.               // ... since, if we reach here, no certificate leads to Accept.
```

This algorithm:

- accepts x if the verifier accepts for some y;
- rejects x if the verifier rejects for every y.

It's a decider for L (using definition of a verifier).

Time complexity?
If verifier has time complexity $O(n^k)$,
then decider's time complexity is: $O((\# \text{ certificates}) \cdot n^k)$.

P and NP

So: how many certificates?

At first sight: looks like infinitely many!

BUT in t steps, a Turing machine can examine at most t symbols in the certificate.

Our verifier has time complexity $O(n^k)$, which is $\leq c \cdot n^k$ (for sufficiently large n).

So this verifier sees $\leq c \cdot n^k$ symbols in the certificate.
Any symbols beyond that are ignored.

Assuming our usual alphabet $\{\mathtt{a}, \mathtt{b}\}$,
the number of certificates *that need to be checked* is $\leq 2^{cn^k}$.

So, decider's total time complexity is $O(2^{cn^k} n^k)$.

P and NP

So, decider's total time complexity is $O(2^{cn^k} n^k)$.

This is dominated by the exponential part.
In fact you can find a constant K a bit larger than k such that the time complexity is $O(2^{n^K})$.

So any language in NP can be decided in exponential time. $\square$

Nondeterministic Turing machines

All Turing machines so far have been deterministic,

- i.e., for each state and symbol, there is just one transition.
- So, for each state and for each symbol, the next action is completely determined: there is a specific next state, new symbol and direction.
  In fact, the entire computation is completely determined by the input.

In a nondeterministic Turing machine (NDTM):
for a given state and symbol, there may be more than one possible transition.

- briefly mentioned in Lecture 18.

One input may lead to many possible computations.

Deterministic TMs are also NDTMs!

Nondeterministic Turing machines

The **language accepted** by a NDTM M is the set of input strings for which *some* computation leads to an Accept state.

A NDTM M is a **nondeterministic decider** for a language L if

- M halts on all inputs, and
- the language accepted by M is L.

A **polynomial-time NDTM** is a NDTM with time complexity $O(n^k)$, for some fixed k.

- As usual, $n =$ length of input string.
- Time complexity is maximum time taken over
  - all inputs of length n *and*
  - all possible computations for each of those inputs.

Nondeterministic Turing machines

Theorem.
L is in NP if and only if
some polynomial-time NDTM is a nondeterministic decider for L.

Proof. (outline)
$(\Longrightarrow)$

Suppose L has a verifier with time complexity $\leq c\, n^k$.

Construct a NDTM M as follows.

On input x, M generates a string y of length $c\, n^k$, nondeterministically,
and then just executes the verifier on x, y.

Nondeterministic Turing machines

$(\Longleftarrow)$

Let M be a polynomial-time NDTM that decides L.

Set up a way of encoding, as a string, the sequence of choices made at the nondeterministic steps of a computation.

Use this string as a certificate . . .

$\square$

NP stands for <u>N</u>ondeterministic <u>P</u>olynomial time.

Contrast with Finite Automata, where DFAs and NFAs define the same class of languages.

Revision

Things to think about:

- Does $\overline{\text{2-SAT}}$, the complement of 2-SAT, belong to P?
- Does $\overline{\text{3-SAT}}$, the complement of 3-SAT, belong to NP?
- Can you define co-NP, by analogy with co-r.e.?
- What is the class of languages that have a verifier?

  (. . . with no requirement for it to be *polynomial time*)

Reading:

- Sipser, sections 7.2–7.3.
- M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., San Francisco, 1979: §2.3, §2.4.

Picture credits: