Today: Computational Complexity
- P, EXP, R
- most problems are uncomputable
- NP
- hardness & completeness
- reductions

$P = \{ \text{problems solvable in polynomial time}\}$
(what this class is all about)

$EXP = \{ \text{problems solvable in exponential time}\}$

$R = \{ \text{problems solvable in finite time}\}$

“recursive” [Turing 1936; Church 1941]

Examples:
- negative-weight cycle detection $\in P$
- $n \times n$ Chess $\in EXP$ but $\notin P$ [who wins from given board config.]
- Tetris $\in EXP$ but don’t know whether $\in P$ [survive given pieces from given board]
Halting problem: given a computer program, does it ever halt (stop)?

- uncomputable ($\notin \mathbb{R}$): no algorithm solves it (correctly in finite time on all inputs)
- decision problem: answer is YES or NO

Most decision problems are uncomputable:
- program $\sim$ binary string $\sim$ nonneg. integer $\in \mathbb{N}$
- decision problem = a function from binary strings to $\{\text{YES, NO}\}$
- $\sim$ nonneg. integers $\sim \{0, 1\}$
- infinite sequence of bits $\sim$ real number $\in \mathbb{R}$
- $|\mathbb{N}| < |\mathbb{R}|$: no assignment of unique nonneg. integers to real numbers ($\mathbb{R}$ uncountable)
- not nearly enough programs for all problems
- each program solves only one problem
- almost all problems cannot be solved
\( NP = \{ \text{decision problems solvable in poly. time} \}
\]

- Can make lucky guesses, always "right", without trying all options
- Nondeterministic model: algorithm makes guesses & then says YES or NO
- Guesses guaranteed to lead to YES outcome if possible (no otherwise)

= \{ \text{decision problems with solutions that can be "checked" in polynomial time} \}

- When answer = YES, can "prove" it & poly-time algorithm can check proof

Example: Tetris \(\in\) NP
- Nondeterministic alg:
  - Guess each move
  - Did I survive?
- Proof of YES: list what moves to make (rules of Tetris are easy)
**P=NP**: big conjecture (worth $1,000,000)

- can't engineer luck
- generating (proofs of) solutions can be harder than checking them

**Claim**: if P ≠ NP, then Tetris ∈ NP \ P

[Deukelaar, Demaine, Hohenberger, Hoogeboom, Kusters, Liben-Nowell, 2004]

**Why?** Tetris is **NP-hard**

= "as hard as" every problem ∈ NP
  
  - in fact **NP-complete** = NP ∩ NP-hard

**Similarly**: Chess is **EXP-complete**

= EXP ∩ EXP-hard

= "as hard as every problem in EXP"

⇒ if NP ≠ EXP, then Chess ∈ EXP \ NP

(also open, but less famous/"important")
Reductions: convert your problem into a problem you already know how to solve (instead of solving from scratch)
- most common algorithm design technique
- unweighted shortest path ⇒ weighted
  set weights = 1
- min-product path ⇒ shortest path
  take logs [PS5-1]
- longest path ⇒ shortest path
  negate weights
- k-bad-road path ⇒ shortest path
  k copies of the graph [PS4-1]

These are all:
**One-call reductions:** A problem ⇒ B problem
  cooler
  A solution ← B solution
**Multicall reductions:** solve A using free calls to B
  in this sense, every algorithm reduces problem ⇒ model of computation

- NP-complete problems are all interreducible using polynomial-time reductions (same difficulty)
  ⇒ can use reductions to prove NP-hardness
e.g. 3-Partition ⇒ Tetris
Examples of NP-complete problems:

- Knapsack (pseudopoly, not poly)
- 3-Partition: given $n$ integers, can you divide them into triples of equal sum?
  (not even pseudopoly possible if $P \neq NP$)

$\rightarrow$ rectangle packing

$\rightarrow$ jigsaw puzzles
- ambiguous
- unique

[Demaine & Demaine 2007]

- Minesweeper, Sudoku, & most puzzles
- Super Mario Bros., Legend of Zelda, Pokémon, ...
  [Aloupis, Demaine, Guo 2012]
- longest simple path in a graph
  $\rightarrow$ deciding whether you've already won in Settler's of Catan is NP-complete!
- Traveling Salesman Problem: shortest path that visits all vertices of a given graph
  - decision version: is min weight $\leq x$?
- longest common subsequence of $k$ strings
- SAT: given a Boolean formula (and, or, not), is it ever true?

$\rightarrow$ shortest paths amidst obstacles in 3D
- 3-coloring a given graph
- find largest clique in a given graph