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

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

\[
\text{EXP} = \{ \text{problems solvable in exponential time} \}
\]

\[
R = \{ \text{problems solvable in finite time} \} \quad \text{"recursive" \ [Turing 1936; Church 1941]}
\]

Examples:
- negative-weight cycle detection ∈ P
- n×n Chess ∈ EXP but \$P
  - who wins from given board config.?
- Tetris ∈ EXP but don't know whether ∈ P
  - \$survive given pieces from given board
**Halting problem**: given a computer program, does it ever halt (stop)?
- **uncomputable** ($\mathcal{R}$): no algorithm solves it correctly in finite time on all inputs
- **decision problem**: answer is YES or NO

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

- 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)
\( P \neq NP: \) big conjecture (worth \$1,000,000)  
\( \approx \) can't engineer luck  
\( \approx \) generating (proofs of) solutions can be harder than checking them

Claim: if \( P \neq NP \), then Tetris \( \in NP \setminus P \)

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

Why? Tetris is \( \overline{NP\text{-hard}} \)

\( = \) "as hard as" every problem \( \in NP \)

\( = \) in fact \( \overline{NP\text{-complete}} = NP \cap \overline{NP\text{-hard}} \)

Similarly: Chess is \( \overline{EXP\text{-complete}} \)

\( = \overline{EXP} \cap \overline{EXP\text{-hard}} \)

\( \Rightarrow \) if \( NP \neq EXP \), then Chess \( \in EXP \setminus NP \)

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 $\Rightarrow$ weighted set weights $= 1$
- min-product path $\Rightarrow$ shortest path take logs $[PSG-1]$
- longest path $\Rightarrow$ shortest path negate weights $[Quiz \ 2, \ P1k]$
- shortest ordered tour $\Rightarrow$ shortest path $k$ copies of the graph $[Quiz \ 2, \ P5]$
- cheapest leaky-tank path $\Rightarrow$ shortest path graph reduction $[Quiz \ 2, \ P6]$

these are all:

**One-call reductions**: A problem $\Rightarrow$ B problem cooler
A solution $\Leftarrow$ B solution

**Multicall reductions**: solve A using free calls to B
- in this sense, every algorithm reduces problem $\Rightarrow$ model of computation

- NP-complete problems are all interreducible using polynomial-time reductions (same difficulty)
$\Rightarrow$ can use reductions to prove NP-hardness e.g. 3-Partition $\Rightarrow$ Tetris
Examples of NP-complete problems:
- Knapsack (pseudopoly, not poly)
- 3-Partition: given n integers, can you divide them into triples of equal sum?
- Traveling Salesman Problem: shortest path that visits all vertices of a given graph
  - decision version: is min weight ≤ x?
- longest common subsequence of k strings
- Minesweeper, Sudoku, & most puzzles
- SAT: given a Boolean formula (and, or, not), is it ever true?
  \[ x \text{ and not } x \rightarrow \text{NO} \]
- shortest paths amidst obstacles in 3D
- 3-coloring a given graph
- find largest clique in a given graph