6.006	Lecture 23	Dec. 8, 2011

TODAY: Computational Complexity

- P, EXP, R
- most problems are uncomputable
- NP
- hardness & completeness
- reductions

$\underline{P}$ = {problems solvable in polynomial time} ↗ n^c

(what this class is all about)

$\underline{EXP}$ = {problems solvable in exponential time} ↳ 2^{n^c}

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

↳ "recursive" [Turing 1936; Church 1941]

$\boxed{P \subsetneq EXP \subsetneq R}$

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 ($\notin R$): 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 $\underbrace{\text{binary strings}}_{\approx \text{ nonneg. integers}}$ to $\underbrace{\{\text{YES}, \text{NO}\}}_{\approx \{0, 1\}}$

$\approx$ infinite sequence of bits $\approx$ real number $\in \mathbb{R}$

- $|\mathbb{N}| \ll |\mathbb{R}|$: 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

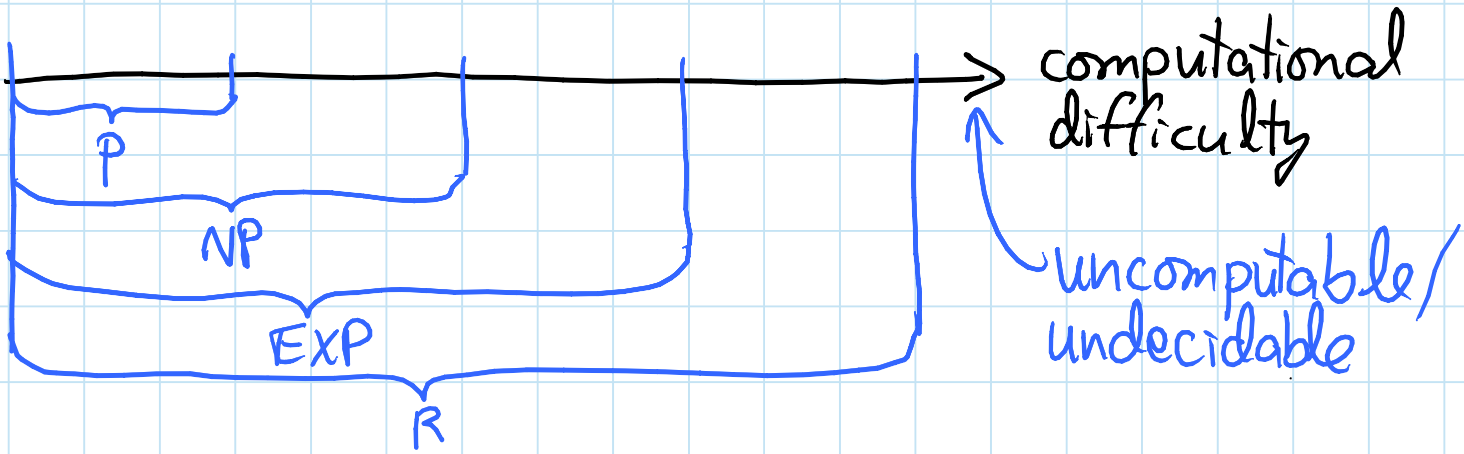
NP = {decision problems solvable in poly. time via a "lucky" algorithm}

↳ 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)

= {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 ∈ 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

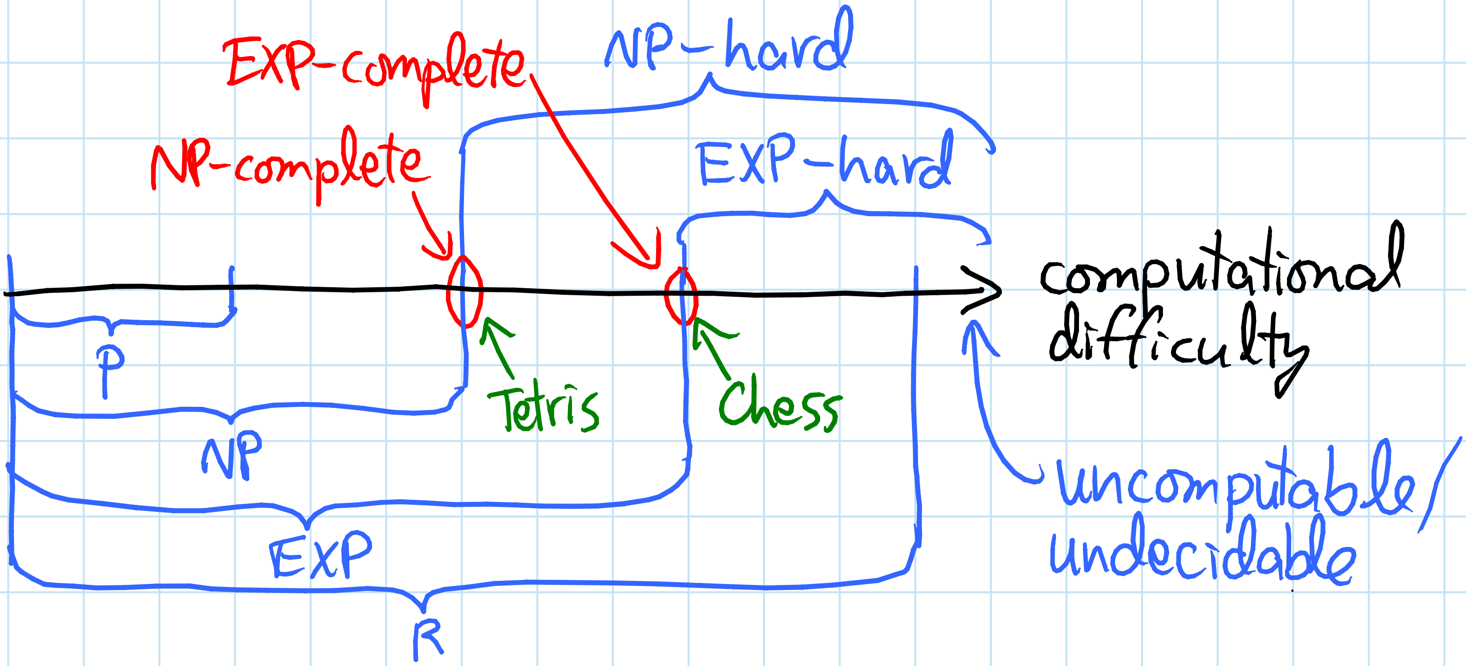
Claim: if $P \neq NP$, then Tetris $\in NP \setminus P$

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

Why? Tetris is NP-hard

= "as hard as" every problem $\in NP$

— in fact NP-complete = NP ∩ NP-hard



Similarly: Chess is EXP-complete

= EXP ∩ EXP-hard

as hard as every problem in EXP

⇒ if $\underbrace{NP \neq EXP}_{\text{also open, but less famous/"important"}}$, then Chess $\notin EXP \setminus NP$

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 [PS6-1]
- longest path → shortest path
  negate weights [Quiz 2, P1k]
- shortest ordered tour → shortest path
  k copies of the graph [Quiz 2, P5]
- cheapest leaky-tank path → shortest path
  graph reduction [Quiz 2, P6]

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?
- 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
- Minesweeper, Sudoku, & most puzzles
- SAT: given a Boolean formula (and, or, not), is it ever true? x and not $x \rightarrow$ NO
- shortest paths amidst obstacles in 3D
- 3-coloring a given graph
- find largest clique in a given graph