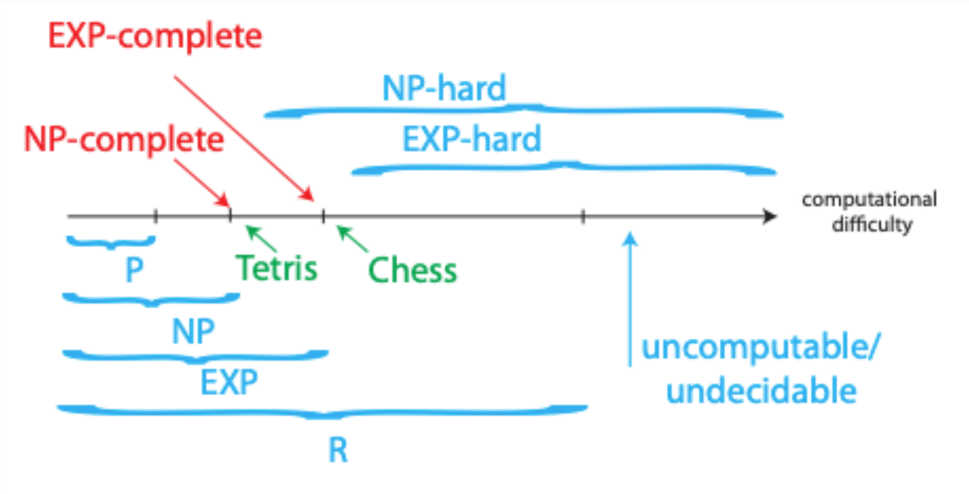
UNIT 8: ADVANCED TOPICS

L23. COMPUTATIONAL COMPLEXITY



DEFINITIONS

- $\mathrm{P} = \{$ problems solvable in polynomial (n^c) time $\}$ (what this class is all about)
- $\mathrm{EXP} = \{$ problems solvable in exponential $\left(2^{n^c}\right)$ time $\}$
- $\mathrm{R} = \{$ problems solvable in finite time $\}$ "recursive" (Turing 1936; Church 1941)

MOST DECISION PROBLEMS ARE UNCOMPUTABLE

- program $\approx$ (finite?) binary string $\approx$ nonneg. integer $\in N$
- decision problem $=$ a function from binary strings ($\approx$ nonneg. integers) to $\{\mathrm{YES}(1), \mathrm{NO}(0)\}$
- $\approx$ 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)
- $\implies$ not nearly enough programs for all problems
- each program solves only one problem
- $\implies$ almost all problems cannot be solved

NP

Non-deterministic Polynomial

NP = {Decision problems solvable in polynomial time via a "lucky" algorithm}. The "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)

In other words, NP = {decision problems with solutions that can be "checked" in polynomial time}. This means that when answer = YES, can "prove" it & polynomial-time algorithm can check proof

$\mathbf{P} \neq \mathbf{NP}$

Big conjecture (worth $\$1,000,000$)

- $\approx$ cannot engineer luck
- $\approx$ generating (proofs of) solutions can be harder than checking them

HARDNESS AND COMPLETENESS

Tetris is $\underline{\textbf{NP-hard}}$ $=$ "as hard as" every problem $\in$ NP. In fact $\underline{\textbf{NP-complete}}$ $= \textbf{NP} \cap \textbf{NP}$-hard.

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) (PS6-1)
- 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)

All the above are One-call reductions: A problem → B problem → B solution → A 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). This implies that we can use reductions to prove NP-hardness — such as in 3-Partition → Tetris

THINKING

这一讲快速介绍了可计算性和计算复杂性。几乎是纯数学的领域。其中大多数决策问题不可计算的证明如果有什么实际意义的话，会指向很有趣的讨论。

R23. COMPUTATIONAL COMPLEXITY

NP-COMPLETE

Read CLRS on NP-complete, know these problems that are NP-complete, and avoid reduction to these problems in the future.

SAT

THINKING

可计算性和计算复杂度在实际应用中有重要的意义：对于需要在大数据集上运算的情况，避免无法解决或解决速度很慢（≥NP-complete）的问题。

L24. TOPICS IN ALGORITHMS RESEARCH

Parallel Processor Architecture & Algorithms

Geometric Folding Algorithms

Data Structures

(Almost) Planar Graphs

Recreational Algorithms

R24. FINAL EXAM REVIEW

BLOOM FILTER

$\prod_{i=0}^{k}$