Lecture 19: Complexity

Decision Problems

- **Decision problem**: assignment of inputs to YES (1) or NO (0)
- Inputs are either **NO inputs** or **YES inputs**

Problem	Decision
s-t Shortest Path	Does a given G contain a path from s to t with weight at most d?
Negative Cycle	Does a given G contain a negative weight cycle?
Longest Simple Path	Does a given G contain a **simple** path with weight at least d?
Subset Sum	Does a given set of integers A contain a subset with sum S?
Tetris	Can you survive a given sequence of pieces in given board?
Chess	Can a player force a win from a given board?
Halting problem	Does a given computer program terminate for a given input?

- **Algorithm/Program**: constant-length code (working on a word-RAM with $\Omega(\log n)$-bit words) to solve a problem, i.e., it produces correct output for every input and the length of the code is independent of the instance size
- Problem is **decidable** if there exists a program to solve the problem in finite time

Decidability

- Program is finite (constant) string of bits, i.e., a nonnegative integer $\in \mathbb{N}$.
  Problem is function $p : \mathbb{N} \to \{0, 1\}$, i.e., infinite string of bits.
- (# of programs $|\mathbb{N}|$, countably infinite) $\ll$ (# of problems $|\mathbb{R}|$, uncountably infinite)
- (Proof by Cantor's diagonalization argument, probably covered in 6.042)
- Proves that most decision problems not solvable by any program (undecidable)
- E.g., the Halting problem is undecidable (many awesome proofs in 6.045)
- Fortunately most problems we think of are algorithmic in structure and are decidable

Decidable Decision Problems

R	problems decidable in finite time	('R' comes from recursive languages)
EXP	problems decidable in exponential time $2^{n^{O(1)}}$	(most problems we think of are here)
P	problems decidable in polynomial time $n^{O(1)}$	(efficient algorithms, the focus of this class)

- These sets are distinct, i.e., **P** $\subsetneq$ **EXP** $\subsetneq$ **R** (via time hierarchy theorems, see 6.045)
- E.g., Chess is in **EXP** $\setminus$ **P**

Nondeterministic Polynomial Time (NP)

- **P** is the set of decision problems for which there is an algorithm A such that, for every input I of size n, A on I runs in poly(n) time and solves I correctly
- **NP** is the set of decision problems for which there is a **verification** algorithm V that takes as input an input I of the problem and a **certificate** bit string of length polynomial in the size of I, so that:
  - V always runs in time polynomial in the size of I;
  - if I is a YES input, then there is some certificate c so that V outputs YES on input (I, c); and
  - if I is a NO input, then no matter what certificate c we choose, V always output NO on input (I, c).
- You can think of the certificate as a **proof** that I is a YES input.
  If I is actually a NO input, then no proof should work.

Problem	Certificate	Verifier
s-t Shortest Path	A path P from s to t	Adds the weights on P and checks whether $\leq d$
Negative Cycle	A cycle C	Adds the weights on C and checks whether < 0
Longest Simple Path	A path P	Checks whether P is a **simple** path with weight $\geq d$
Subset Sum	A set of items A'	Checks whether $A' \in A$ has sum S
Tetris	Sequence of moves	Checks that the moves allow survival

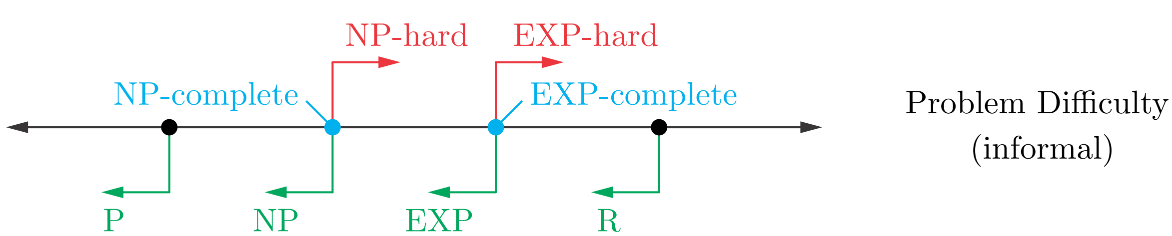
- **P** $\subseteq$ **NP**: The verifier V just solves the instance ignoring any certificate
- **NP** $\subseteq$ **EXP**: Try all possible certificates! At most $2^{n^{O(1)}}$ of them, run verifier V on all
- **Open:** Does **P** $=$ **NP**? **NP** $=$ **EXP**?
- Most people think **P** $\subsetneq$ **NP** ($\subsetneq$ **EXP**), i.e., generating solutions harder than checking
- If you prove either way, people will give you lots of money ($1M Millennium Prize)
- Why do we care? If can show a problem is hardest problem in **NP**,
  then problem cannot be solved in polynomial time if **P** $\neq$ **NP**
- How do we relate difficulty of problems? Reductions!

Reductions

- Suppose you want to solve problem A
- One way to solve is to convert A into a problem B you know how to solve
- Solve using an algorithm for B and use it to compute solution to A
- This is called a **reduction** from problem A to problem B ($A \to B$)
- Because B can be used to solve A, B is **at least as hard** as A ($A \le B$)
- General algorithmic strategy: reduce to a problem you know how to solve

A	Conversion	B
Unweighted Shortest Path	Give equal weights	Weighted Shortest Path
Integer-weighted Shortest Path	Subdivide edges	Unweighted Shortest Path
Longest Path	Negate weights	Shortest Path

- Problem A is **NP-hard** if every problem in **NP** is polynomially reducible to A
- i.e., A is at least as hard as (can be used to solve) every problem in **NP** ($X \le A$ for $X \in$ **NP**)
- **NP-complete = NP $\cap$ NP-hard**
- All **NP-complete** problems are equivalent, i.e., reducible to each other
- First **NP-complete** problem? Every decision problem reducible to satisfying a logical circuit, a problem called "Circuit SAT".
- Longest Simple Path and Tetris are **NP-complete**, so if any problem is in **NP** $\setminus$ **P**, these are
- Chess is **EXP-complete**: in **EXP** and reducible from every problem in **EXP** (so $\notin$ **P**)



Examples of NP-complete Problems

- Subset Sum from L18 ("weakly NP-complete" which is what allows a pseudopolynomial-time algorithm, but no polynomial algorithm unless **P** = **NP**)
- 3-Partition: given n integers, can you divide them into triples of equal sum? ("strongly NP-complete": no pseudopolynomial-time algorithm unless **P** = **NP**)
- Rectangle Packing: given n rectangles and a target rectangle whose area is the sum of the n rectangle areas, pack without overlap
  - Reduction from 3-Partition to Rectangle Packing: transform integer a_i into $1 \times a_i$ rectangle; set target rectangle to $n/3 \times \left(\sum_i a_i\right)/3$
- Jigsaw puzzles: given n pieces with possibly ambiguous tabs/pockets, fit the pieces together
  - Reduction from Rectangle Packing: use uniquely matching tabs/pockets to force building rectangles and rectangular boundary; use one ambiguous tab/pocket for all other boundaries
- Longest common subsequence of n strings
- Longest simple path in a graph
- Traveling Salesman Problem: shortest path that visits all vertices of a given graph (or decision version: is minimum weight $\leq d$)
- Shortest path amidst obstacles in 3D
- 3-coloring given graph (but 2-coloring $\in$ **P**)
- Largest clique in a given graph
- SAT: given a Boolean formula (made with AND, OR, NOT), is it every true?
  E.g., x AND NOT x is a NO input
- Minesweeper, Sudoku, and most puzzles
- Super Mario Bros., Legend of Zelda, Pokémon, and most video games are **NP-hard** (many are harder)