Outline: Search III & NP-completeness
- BFS vs. DFS
- job scheduling
- topological sort
- intractable problems
- \( P, NP, NP\)-completeness

Reading: CLRS 22.4 & 34.1-34.3 (at high level)

Recall:
- Breadth-First Search (BFS): level by level
- Depth-First Search (DFS): backtrack as necc.
- both \( O(V+E) \) worst-case time \( \Rightarrow \) optimal
- BFS computes shortest paths (min. \# edges)
- DFS is a bit simpler & has useful properties
Job scheduling: given directed acyclic graph (DAG), where vertices represent tasks & edges represent dependencies, order tasks without violating dependencies

Source = vertex with no incoming edges = schedulable at beginning (A, G, I)

Attempt: BFS from each source:
- from A finds H, B, C, F
- from D finds C, E, F
- from G finds H

Topological sort: reverse of DFS finishing times (time at which node’s outgoing edges finished)

Exercise: prove that no contraints are violated
Intractability
- DFS & BFS are worst-case optimal if problem is really graph search (to look at graph)
- what if graph...
- is implicit?
- has special structure?
- is infinite?

Halting problem: given a computer program, does it ever halt (stop)?
- decision problem: answer is YES or NO
- UNDECIDABLE: no algorithm solves this problem correctly in finite time on all inputs

Most decision problems are undecidable:
- program \( \cong \) binary string \( \cong \) nonneg. integer \( \in \mathbb{N} \)
- decision problem = a function from binary strings to \{YES, NO\}
- nonneg. integers \( \cong \{0, 1\} \)
- infinite sequence of bits \( \cong \) 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
$n \times n \times n$ Rubik's cube:
- $n=2$ or $3$ is easy algorithmically: $O(1)$ time (in practice, $n=3$ still unsolved)
- graph size grows exponentially with $n$
- solvability decision question is easy (parity check)
- finding shortest solution: UNSOLVED

$n \times n$ Chess: given $n \times n$ board & some configuration of pieces, can WHITE force a win?
- can be formulated as $(\alpha \beta)$ graph search
- every algorithm needs time exponential in $n$: "EXPTIME-complete" [Fraenkel & Lichtenstein 1981]

$n^2 - 1$ Puzzle: given $n \times n$ grid with $n^2 - 1$ pieces, sort pieces by sliding
- similar to Rubik's cube:
- solvability decision question is easy (parity check)
- finding shortest solution: $\text{NP-COMPLETE}$ [Ratner & Warmuth 1990]

Tetris: given current board configuration & list of pieces to come, stay alive
- $\text{NP-COMPLETE}$ [Demaine, Hohenberger, Liben-Nowell 2003]
\[ P = \text{all (decision) problems solvable by a polynomial time algorithm (efficient)} \]

\[ \text{NP} = \text{all decision problems whose YES answers have short (polynomial-length) "proofs" checkable by a polynomial-time algorithm} \]

e.g.: Rubik's cube & \( n^2 - 1 \) puzzle; is there a solution of length \( \leq k^2 \)?
- \( \text{YES} \Rightarrow \text{easy-to-check short proof (moves)} \)
- \text{Tetris} \( \in \text{NP} \)
- \text{but we conjecture Chess} \( \notin \text{NP} \)
  (winning strategy is big - exponential in \( n \))

\( P \neq \text{NP} \): big conjecture (worth \$1,000,000\)
- \( \approx \) generating proofs/solutions is harder than checking them

\( \text{NP-complete} = \text{in NP & NP-hard} \)

\( \text{NP-hard} = \text{as hard as every problem in NP} \)
- every problem in NP can be efficiently converted into this problem
- \( \Rightarrow \) if this problem \( \in P \) then \( P = \text{NP} \)
  (so probably this problem \( \notin P \))