6.006	Lecture 14	Apr. 1, 2008

Outline: Search III & NP-completeness

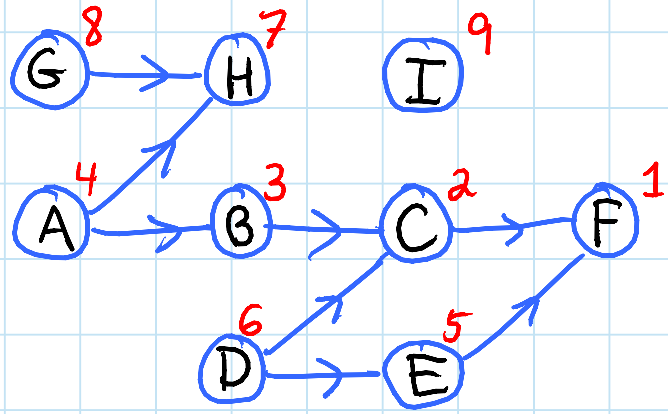
- BFS vs. DFS
- job scheduling
- topological sort
- intractable problems
- P, NP, NP-completeness

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

} need to merge $\Rightarrow$ costly

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?

} Rubik's cube

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 $\approx$ 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

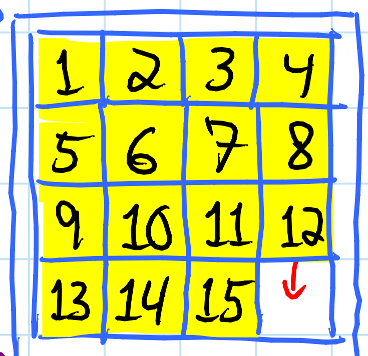
$n \times n \times n$ Rubik's cube:

- $n=2$ 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:
  NP-COMPLETE [Ratner & Warmuth 1990]

Tetris:
given current board configuration & list of pieces to come, stay alive

- NP-COMPLETE [Demaine, Hohenberger, Liben-Nowell 2003]

P = all (decision) problems solvable by a polynomial ($O(n^c)$) time algorithm (efficient)

NP = 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 $\leq k$?
YES $\Rightarrow$ easy-to-check short proof (moves)
- Tetris $\in$ NP
- but we conjecture Chess $\notin$ NP
(winning strategy is big — exponential in n)

P ≠ NP: big conjecture (worth \$1,000,000)
≈ generating proofs/solutions is harder than checking them

NP-complete = in NP & NP-hard

NP-hard = 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 = NP
(so probably this problem $\notin$ P)