Lecture Overview: Search 3 of 3 & NP-completeness

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

Readings
CLRS, Sections 22.4 and 34.1-34.3 (at a high level)

Recall:
- Breadth-First Search (BFS): level by level
- Depth-First Search (DFS): backtrack as necc.
- both $O(V + E)$ worst-case time $\implies$ 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

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 - costly

Figure 1: Dependence Graph

Figure 2: BFS-based Scheduling
Topological Sort and NP-Completeness

Topological Sort
Reverse of DFS finishing times (time at which node’s outgoing edges finished)
Exercise: prove that no constraints 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?

The first 2 characteristics (implicitness and special structure) apply to the Rubik’s Cube problem.
The third characteristic (infiniteness) applies to the Halting Problem.

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 ≈ binary string ≈ nonneg. integer ∈ ℤ
- decision problem = a function from binary strings to {YES, NO}. Binary strings refer to ≈ nonneg. integers while {YES, NO} ≈ {0, 1}
- ≈ infinite sequence of bits ≈ real number ∈ ℝ
- ℤ ⊆ ℝ: non assignment of unique nonneg. integers to real numbers (ℝ uncountable)
- ⇒ not nearly enough programs for all problems & each program solves only one problem
- ⇒ 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 (see Figure 3).

- 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, NP, NP-completeness

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 and \(n^2 - 1\) puzzle:
is there a solution of length \(\leq k\)?
YES \(\implies\) easy-to-check short proof(moves)
Tetris \(\in\) NP
but we conjecture Chess not NP (winning strategy is big- exponential in \(n\))

P ≠ NP: Big conjecture (worth $1,000,000) \approx\) 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
\(\implies\) if this problem \(\epsilon\) P then P = NP (so probably this problem not in P)