Algorithms: Design and Analysis, Part II

NP-Completeness

P: Polynomial-Time Solvable Problems

Ubiquitous Intractability

Focus of this course (+ Part I): practical algorithms + supporting theory for fundamental computational problems.

Sad fact: many important problems seem impossible to solve efficiently.

Next: How to formalize computational intractability using NP-completeness.

Later: algorithmic approaches to NP-complete problems.

Polynomial-Time Solvability

Question: how to formalize (in)tractability?

Definition: a problem is polynomial-time solvable if there is an algorithm that correctly solves it in $O(n^k)$ time, for some constant k.

[where n = input length = # of keystrokes needed to describe input]

[yes, even $k = 10{,}000$ is sufficient for this definition]

Comment: will focus on deterministic algorithms, but to first order doesn't matter.

The Class P

Definition: P = the set of poly-time solvable problems.

Examples: everything we've seen in this course except:

- cycle-free shortest paths in graphs with negative cycles
- Knapsack [running time of our algorithm was $\Theta(nW)$, but input length proportional to $\log W$]

both problems are NP-complete

Interpretation: rough litmus test for "computational tractability".

Traveling Salesman Problem

Input: Complete undirected graph with nonnegative edge costs.

Output: a min-cost tour [i.e., a cycle that visits every vertex exactly once].

Conjecture: [Edmonds '65] there is no polynomial-time algorithm for TSP.

[as we'll see, equivalent to $P \neq NP$]