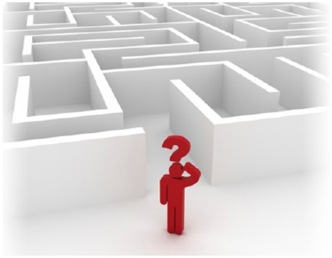


NP-Completeness

P: Polynomial-Time Solvable Problems

Algorithms: Design and Analysis, Part II

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 key strokes 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 $\log W$]

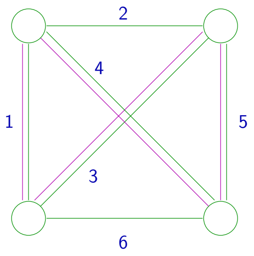
Both problems are NP-complete

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].



$OPT = 13$

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

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