Algorithms: Design and Analysis, Part II

NP-Completeness

Reductions and Completeness

Reductions

Conjecture: [Edmonds '65] there is no polynomial-time algorithm that solves the TSP. [equivalent to $P \neq NP$]

Really good idea: amass evidence of intractability via relative difficulty – TSP "as hard as" lots of other problems.

Definition: [a little informal] problem Π_1 reduces to problem Π_2 if: given a polynomial-time subroutine for Π_2, can use it to solve Π_1 in polynomial time.

Quiz

Which of the following statements are true?

(A) computing the median reduces to sorting

(B) detecting a cycle reduces to depth-first search

(C) all pairs shortest paths reduces to single-source shortest paths

(D) all of the above

Completeness

Suppose Π_1 reduces to Π_2.

Contrapositive: if Π_1 is not in P, then neither is Π_2.

That is: Π_2 is at least as hard as Π_1.

Definition: Let $\mathcal{C}$ = a set of problems.

The problem Π is $\mathcal{C}$-complete if:

(1) $\Pi \in \mathcal{C}$ (2) everything in $\mathcal{C}$ reduces to Π.

That is: Π is the hardest problem in all of $\mathcal{C}$.

Choice of the Class C?

Idea: show TSP is $\mathcal{C}$-complete for a REALLY BIG set $\mathcal{C}$.

How about: show this where $\mathcal{C}$ = ALL problems.

Halting Problem: given a program and an input for it, will it eventually halt?

Fact: [Turing '36] no algorithm, however slow, solves the Halting Problem.

Contrast: TSP definitely solvable in finite time (via brute-force search).

Refined idea: TSP as hard as all brute-force-solvable problems.