Introduction to Algorithms
Lecture 23
Prof. Piotr Indyk

© Piotr Indyk

Have seen so far

- Algorithms for many interesting problems:
  - Running times \(O(nm^2), O(n^3), O(n \log n), O(n), \ldots\)
  - I.e., polynomial in the input size
- Can we solve all (or most of) interesting problems in polynomial time?
- Not really…

© Piotr Indyk

Example difficult problem

- Traveling Salesperson Problem (TSP)
  - Input: undirected graph with lengths on edges
  - Output: shortest tour that visits each vertex exactly once
- Best known algorithm: \(O(n 2^n)\) time.

© Piotr Indyk

Another difficult problem

- Clique:
  - Input: undirected graph \(G = (V,E)\)
  - Output: largest subset \(C\) of \(V\) such that every pair of vertices in \(C\) has an edge between them
- Best known algorithm: \(O(n 2^n)\) time

© Piotr Indyk

What can we do?

- Spend more time and money designing efficient algorithms for those problems
  - People tried for a few decades, no luck
  - Outstanding $1000,000 prize for finding one
  - It seems very likely that such algorithms do not exist
- Prove there is no polynomial time algorithm for those problems
  - Would be great
  - Seems really difficult
- Best lower bounds for “natural” problems:
  - \(\Omega(n^3)\) for restricted computational models
  - \(\Omega(n)\) for unrestricted computational models

© Piotr Indyk

P vs NP
(interconnectedness of all things)

- A whole course by itself
- We’ll do just two lectures
- More in 6.045, 6.840J, etc.
What else can we do?

- Show that those hard problems are essentially equivalent.
  I.e., if we can solve one of them in poly time, then all others can be solved in poly time as well.
- Works for at least few thousand hard problems.

The benefits of equivalence

- Combines research efforts
- If one problem has polytime solution, then all of them do.

A more realistic scenario

- Once an exponential lower bound is shown for one problem, it holds for all of them.
- But someone is happy…

Summing up

- If we show that a problem ∏ is equivalent to a few thousand other well-studied problems without efficient algorithms, then we get a very strong evidence that ∏ is hard.
- We need to:
  1. Identify the class of problems of interest
  2. Define the notion of equivalence
  3. Prove the equivalence(s)

1. Class of problems
   (informally)

- Decision problems: answer YES or NO. E.g., “is there a tour of length ≤ K?”
- Solvable in non-deterministic polynomial time:
  - Intuitively: the solution can be verified in polynomial time.
  - E.g., if someone gives a tour T, we can verify if T is a tour of length ≤ K.
- Therefore, TSP is in NP.

1. Class of problems: NP

- A problem ∏ is solvable in poly time (or ∏ ∈ P), if there is a poly time algorithm V(·) such that for any input x:
  \[ V(x) = \text{YES} \iff \text{there exists certificate } y \text{ of size } \text{poly}(|x|) \text{ such that } V(x,y) = \text{YES} \]
- A problem ∏ is solvable in non-deterministic poly time (or ∏ ∈ NP), if there is a poly time algorithm V(·) such that for any input x:
  \[ V(x) = \text{YES} \iff \text{there exists certificate } y \text{ of size } \text{poly}(|x|) \text{ such that } V(x,y) = \text{YES} \]
Examples of problems in NP

- Is “Does there exist a clique in G of size ≥k” in NP?
  - Yes: \( V(x,y) \) interprets \( x \) as a graph \( G \), \( y \) as a set \( C \), and checks if all vertices in \( C \) are adjacent and if \( |C| \geq k \)
- Is Sorting in NP?
  - No, not a decision problem.
- Is “Sortedness” in NP?
  - Yes: ignore \( y \), and check if the input \( x \) is sorted.

2. Reductions:

2. Reductions (formally)

- \( \Pi' \) is poly time reducible to \( \Pi \) ( \( \Pi' \leq \Pi \) ) iff there is a poly time function \( f \) that maps inputs \( x' \) to \( \Pi' \) into inputs \( x \) of \( \Pi \), such that for any \( x' \)
  \( \Pi'(x') = \Pi(f(x')) \)

- Fact 1: if \( \Pi \in \text{P} \) and \( \Pi' \leq \Pi \) then \( \Pi' \in \text{P} \)
- Fact 2: if \( \Pi \in \text{NP} \) and \( \Pi' \leq \Pi \) then \( \Pi' \in \text{NP} \)
- Fact 3: if \( \Pi' \leq \Pi \) and \( \Pi'' \leq \Pi' \) then \( \Pi'' \leq \Pi \)