Acknowledgement

• Eric Torng
• Moon Jung Chung
• Charles Ofria

Outline

• Problems and Problem Instances
  – Decision vs. Optimization Problems
• What is an Algorithm?
• Models of Computation
  – Deterministic vs. Non-deterministic
• Problem Formulation: Encoding
• Hardness: Absolute vs. Relative
Problems (Recap)

- Problem:
  - A mapping/relation between a set of input instances (domain) and an output set (range)
  - A general question including a set of parameters (free variables/inputs) and a set of constraints that should be satisfied by an answer

- Problem instance:
  - An instantiation of the general problem definition for a specific set of parameters values

- Example: Traveling Salesperson Problem (TSP)
  - Input:
    - Parameters:
      - A set of cities $\{c_1, \ldots, c_n\}$
      - For each pair $c_i$ and $c_j$ in $C$, a distance $d(c_i, c_j)$
    - Output (Answer): a tour starting from some city $c_k$ back to $c_k$ such that the total summation of distances is minimized

Problems – cont’d

- An instance of TSP:

![Traveling Salesperson Problem Instance](image)

Decision Problems

- We restrict our attention to decision problems
- Almost all natural problems can be converted into an equivalent decision problem without changing the complexity of the problem
  - One technique: add an extra input variable that represents a specific solution for the original problem
- Key characteristic: 2 types of inputs
  - Yes input instances
  - No input instances
Optimization vs. Decision

- Example using clique problem
  - Optimization Problem
    - Input: Graph G=(V,E)
    - Task: Output size of maximum clique in G
  - Task 2: Output a maximum sized clique of G
  - Decision Problem
    - Input: Graph G=(V,E), integer k ≤ |V|
    - Y/N Question: Does G contain a clique of size k?

- Your assignment
  - Show that if we can solve decision clique in polynomial-time, then we can solve the optimization clique problems in polynomial-time.

What is an Algorithm?

According to Wikipedia:

- "an algorithm is a procedure (a finite set of well-defined instructions) for accomplishing some task which, given an initial state, will terminate in a defined end-state"

- Notice: An algorithm is supposed to find a solution for any given instance of a problem
  - “The word “algorithm” stems from the name Al-Khwarizmi, a Persian mathematician, astronomer, astrologer and geographer (lived 780-850)”
  - “considered to be the father of algebra, a title he shares with Diophantus”
  - “Algoritmi de numero Indorum, the Latin translation of his major work on the Indian numerals, introduced the positional number system and the number zero to the Western world in the 12th century”

Computation Models
Turing Machines

A Turing machine has a finite-state-control (its program), a two way infinite tape (its memory) and a read-write head (its program counter).

Finite State Control

Head

Tape

....10100011011....

Nondeterministic Running Time

- We measure running time by looking at height of computation tree, not number of nodes explored.
- Both computation have same height 4 and thus same running time.

ND Computation Returning Yes

- If any leaf node returns yes, we consider the input to be a yes input.
- If all leaf nodes return no, then we consider the input to be a no input.
- In reality, we cannot have an arbitrary non-deterministic machine; finite resources!
Problem Formulation: Encoding

Encoding Inputs

- Encoding:
  - a mapping from a problem $\Pi$ to an instance of $\Pi$ represented as a binary string
- We assume the input is encoded as a string of 0’s and 1’s
- Example: input is an undirected graph
  - First specify the number of nodes in binary
  - Then use an adjacency matrix to represent edges

Hardness: Absolute vs. Relative
Proof of Hardness?!

- Let’s say I am given a problem to solve and I fail to find an efficient algorithm in a specific amount of time (e.g., 6 months, 1 year, etc.)
- What are my excuses?
  - I cannot find an efficient algorithm because
    - I am too dumb!
    - no such algorithm exists!
    - neither can all the famous scientists! (unlikely to exist)
- Would it not be nice to back my excuse by some kind of formal proof?

Complexity Class P

Fundamental Setting

- When faced with a new problem $\Pi$, we alternate between the following two goals
  1. Find a “good” algorithm for solving $\Pi$
     - Use algorithm design techniques
  2. Prove a “hardness result” for problem $\Pi$
     - No “good” algorithm exists for problem $\Pi
Polynomial-Time Algorithms

- An algorithm with worst-case time complexity \( n^k \), where \( n \) is the input size and \( k \) is a constant
- Example: (Euler tour)
  - Input: a connected digraph \( G = \langle V, A \rangle \)
  - Output:
    - Does there exist a cycle in \( G \) that traverses each arc in \( A \) exactly once?
    - Can you find an algorithm in order of \( O(|A|) \)?
- Hamiltonian Cycle:
  - Input: a connected digraph \( G = \langle V, A \rangle \)
  - Output:
    - Does there exist a cycle in \( G \) that meets each vertex in \( A \) exactly once?

Polynomial-time Solvable

- Class P:
  - Set of all problems for which there exists a polynomial-time (efficient) algorithm for an instance \( I \) in encoding \( e(I) \)
- A problem \( \Pi \) is polynomial-time solvable if and only if \( \Pi \in P \)
- \( O(n^{100}) \) is also polynomial; is it an efficient algorithm?
- In addition to polynomial-time complexity, what else is interesting about class P?
  - If you find a polynomial algorithm for a problem, then often you can improve the efficiency of your algorithm

Polynomially Computable Functions

- Function: Take a string in \( \{0, 1\}^* \) as input, generate a string in \( \{0, 1\}^* \) as solution (output)
- A function \( f : \{0, 1\}^* \rightarrow \{0, 1\}^* \) is polynomial-time computable iff for any string \( s \) in \( \{0, 1\}^* \) there exists a polynomial-time algorithm that generates \( f(s) \)
- Why not use other encodings? (e.g., alphabet)
  - Does it affect the complexity?
- Two encodings \( e_1 \) and \( e_2 \) are polynomially related if there exist two polynomial-time computable functions \( f_{12} \) and \( f_{21} \) s.t.
  - for any problem instance \( I \) in \( \{0, 1\}^* \) we have \( f_{21}(e_1(I)) = e_2(I) \) and \( f_{12}(e_2(I)) = e_1(I) \)
Complexity Class P

- P is the set of problems that can be solved using a polynomial-time algorithm
  - Sometimes we focus only on decision problems
  - The task of a decision problem is to answer a yes/no question
- If a problem belongs to P, it is considered to be “efficiently solvable”
- If a problem is not in P, it is generally considered to be NOT “efficiently solvable”
- Our goals are to:
  1. Prove that Π belongs to P
  2. Prove that Π does not belong to P

Absolute Hardness Results

- Fuzzy Definition
  - A hardness result for a problem Π without reference to another problem
- Examples
  - Solving the clique problem requires Ω(2^n) time in the worst-case?
  - Solving the clique problem requires Ω(n) time in the worst-case?
  - The clique problem is not in P?
Proof Techniques

• Diagonalization
  – We don’t cover, but can be used to prove superpolynomial times required for some problems
• Information-Theoretic arguments
  – \( \Omega(n \log n) \) lower bound for sorting
  – Typically not a superpolynomial lower bounds
• “Size of input” argument
  – Prove that solving the graph connectivity problem requires \( \Omega(V^2) \) time
  – Prove that solving the maximum clique problem requires \( \Omega(V^2) \) time
  – Typically not a superpolynomial lower bound

Status

• Many natural problems can be shown to be in P
  – Graph connectivity
  – Shortest Paths
  – Minimum Spanning Tree
• Very few natural problems have been proven to NOT be in P
  – Variants of halting problem are one example
• Many natural problems cannot be placed in or out of P
  – Satisfiability
  – Longest Path Problem
  – Hamiltonian Cycle
  – Traveling Salesperson

Relative Hardness Results
Relative Hardness Results

- Fuzzy Definition
  - A hardness result for a problem Π with reference to another problem
- Examples
  - SAT (Satisfiability): Given a Boolean formula φ, does there exist a truth-value assignment to variables of φ such that φ evaluates to true?
  - SAT is at least as hard as Hamiltonian Cycle to solve
  - If SAT is unsolvable, then Hamiltonian Cycle is unsolvable.
  - If SAT is in P, then Hamiltonian Cycle is in P
  - If Hamiltonian Cycle is not in P, then SAT is not in P

Important Observation

- We are interested in relative hardness results BECAUSE of our inability to prove absolute hardness results
- That is, if we could prove strong absolute hardness results, we would not be as interested in relative hardness results
- Example
  - If I could prove “SAT is not in P”, then I would be less interested in proving “If Hamiltonian Cycle is not in P, then SAT is not in P”.

Relative Hardness Proof Technique

- We show that Π₂ is at least as hard as Π₁ in the following way
- An alternative way of saying it:
  Π₁ is no harder than Π₂
- Informal:
  - We show how to solve problem Π₁ using a procedure P₂ that solves Π₂ as a subroutine
Examples

• Multiplication and Squaring
  – square(x) = mult(x, x)
  • Proves multiplication is at least as hard as squaring
  – mult(x, y) = (square(x + y) − square(x − y))/4
  • Prove squaring is at least as hard as multiplication
  • Assumes that addition, subtraction, and division by 4 can be done with no substantial increase in complexity
  • Specific complexity of multiplication may be higher as there are two calls to square, but the difference is polynomially bounded

Polynomial-Time Reduction

Polynomial-time Reduction

Technique

• Consider two problems $\Pi_1$ and $\Pi_2$, and suppose I want to show
  – If $\Pi_1$ is in P, then $\Pi_2$ is in P
  – If $\Pi_1$ is not in P, then $\Pi_2$ is not in P
• The basic idea
  – Develop a function (reduction) $R$ that maps input instances of problem $\Pi_1$ to input instances of problem $\Pi_2$
  – The function $R$ should be computable in polynomial time
  – $x$ is a yes input to $\Pi_1$ $\implies R(x)$ is a yes input to $\Pi_2$
• Notation: $\Pi_1 \leq_p \Pi_2$
What $\Pi_1 \leq_p \Pi_2$ Means

If $R$ exists, then:
- If $\Pi_2$ is in $P$, then $\Pi_1$ is in $P$
- If $\Pi_1$ is not in $P$, then $\Pi_2$ is not in $P$

Showing $\Pi_1 \leq_p \Pi_2$

- For any $x$ input for $\Pi_1$, specify what $R(x)$ will be
- Show that $R(x)$ has polynomial size relative to $x$
  - You should show that $R$ runs in polynomial time; we only require the size requirement above
- Show that if $x$ is a YES instance for $\Pi_1$, then $R(x)$ is a YES instance for $\Pi_2$
- Show that if $x$ is a NO instance for $\Pi_1$, then $R(x)$ is a NO instance for $\Pi_2$
  - Often done by showing that if $R(x)$ is a YES instance for $\Pi_2$, then $x$ must have been a YES instance for $\Pi_1$