BACKGROUND

- P vs NP
- NP-completeness
PROBLEM

\[ x^2 + 3xy + 7yz^3 = 1 \]

\[ x + y + z = 4 \]

\[ xyz + 2x - y - z = -4 \]

SOLUTION

\[ x = 2 \quad y = 3 \quad z = -1 \]
P vs NP: searching vs checking

- Scheduling/Assignment Problems
  - e.g., time table (courses, teachers) (times)
  - (students, rooms)
  - jobs on a single machine (release, deadline, length)
  - jigsaw puzzle
  - crossword puzzle

- Flow Problems
  - over a physical/abstract network
    - e.g., pipes, wires, roads, airlines

- Routing Problems
  - (Short) paths under various constraints
    - e.g., finding a labyrinth path

- Arithmetic/Algebraic problems
  - e.g., solving a system of equations

**EASY TO CHECK CORRECTNESS OF SOLUTIONS**

**HARD TO FIND CORRECT SOLUTIONS**
P vs NP: proving vs verifying

The notion of a PROOF presupposes that verifying validity of proofs is easier than finding them.

(Finding proofs is a search problem for which correct solutions are easy to validate.)

\[ \iff P \neq NP \]

\[ \text{(RE: TRAD. PROOFS...)} \]

EASY TO VERIFY VALIDITY OF PROOFS

\[ \text{vs} \]

HARD TO FIND CORRECT PROOFS
**Efficient vs Infeasible**

- (usually) **MUST READ THE INPUT**
  \[ \Rightarrow \text{MUST SPEND LINEAR TIME.} \]
  
  **time linear in the input length**

- **LINEAR TIME IS EFFICIENT**

- What about quadratic time?
  
  (e.g. INTEGER MULTIPLICATION via ELEMENTARY SCHOOL METHOD)

  \[ \uparrow \text{POLYNOMIAL-TIME IS EFFICIENT} \]

- **EXPONENTIAL-TIME IS INFEASIBLE**
  
  (e.g., the naive FACTORING ALGORITHM)
EFFICIENT vs INFEASIBLE

- (usually) MUST READ THE INPUT
  \[ \Rightarrow \text{MUST SPEND LINEAR TIME.} \]
  \[ \text{time linear in the input length} \]

- LINEAR TIME IS EFFICIENT

- What about quadratic time?
  (e.g., INTEGER MULTIPLICATION via ELEMENTARY SCHOOL METHOD)
  \[ \uparrow \]
  POLYNOMIAL-TIME IS EFFICIENT

<table>
<thead>
<tr>
<th>n</th>
<th>10</th>
<th>20</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n^2)</td>
<td>100</td>
<td>400</td>
<td>10,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(2^n)</th>
<th>1000</th>
<th>1M</th>
<th>1,000, \ldots, 000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>30 ZEROS</td>
</tr>
</tbody>
</table>

- EXPONENTIAL-TIME IS INFEASIBLE
  (e.g., the naive FACTORING ALGORITHM)
GRAPH THEORY

GRAPH = SET OF POINTS (vertices) + SET OF PAIRS OF POINTS (edges).

Points = \{1, 2, 3, 4, 5, 6\}
Edges = \{(2, 3), (4, 6), (4, 5), (5, 6)\}

Graphs represent various natural objects

e.g., - Networks (of roads, communication, pipes, etc.)
- (binary) Relations (e.g., who are related/friends)

Solving problems by considering their abstraction as graph problems.
(e.g., Matching, Hamiltonian cycle, etc.)
3-coloring
**NP-completeness**

Conjecture: \( P \neq NP \) (widely believed but unproven)

Phenomenon: Natural "NP Problems" have no efficient solvers (i.e., not in \( P \)).

For each \( \Pi \in NP \), we wish to show \( \Pi \notin P \) but... this would yield \( P \neq NP \)...

Instead, we can hope to show that "if \( \Pi \in P \) then \( NP = P \)."

\[ \Rightarrow P \neq NP \text{ implies that } \Pi \notin P \]

**Method:** Show that any problem in \( NP \) can be "reduced" to \( \Pi \).

\[ \text{Instance of } \Pi' \text{ (for any } \Pi' \in NP) \xrightarrow{\text{efficiently transformed}} \text{Instance of } \Pi \]

\[ \Rightarrow \Pi \text{ "encodes" all problems in } NP. \]
NP-Completeness (cont.)

Solving systems of (quadratic) equations is NP-Complete

⇒ Each problem in NP

  e.g. - scheduling
  - finding short TSP
  - factoring integers

  CAN BE "ENCODED" (reduced to solving) A SYSTEM OF EQUATIONS.

- TSP ≜ (1) Dist. between pairs of cities
  (2) Need a path that visits all cities (tour)

- Scheduling over a single processor/machine
  a seq. of jobs w. release, length & deadline.

- Factoring: Composite number ⇒ prime factorization