Examples of languages in P

• Recall: positive integers \( x, y \) are relatively prime if their Greatest Common Divisor (GCD) is 1.

• will show the following language is in P:
  \[ \text{RELPRIME} = \{<x, y>: x \text{ and } y \text{ are relatively prime}\} \]

• what is the running time of the algorithm that tries all divisors up to \( \min(x, y) \)?

Euclid’s Algorithm

• possibly earliest recorded algorithm
  
  on input \( <x, y> \):
  
  • repeat until \( y = 0 \)
  
  • set \( x = x \mod y \)
  
  • swap \( x, y \)

  • \( x \) is the GCD(\( x, y \)). If \( x = 1 \), accept; otherwise reject

Example run on input \( <24, 5> \):

\[
\begin{align*}
x, y &= 24, 5 \\
x, y &= 5, 4 \\
x, y &= 4, 1 \\
x, y &= 1, 0 \\
	ext{accept}
\end{align*}
\]

Claim: value of \( x \) reduced by \( \frac{1}{2} \) at every execution of (2) except possibly first one.

Proof:

• after (2) \( x < y \)

• after (3) \( x > y \)

• if \( x/2 \geq y \), then \( x \mod y < y \times x/2 \)

• if \( x/2 < y \), then \( x \mod y = x - y < x/2 \)
A puzzle

- Find an efficient algorithm to solve the following problem:
- Input: sequence of pairs of symbols e.g. (A, b), (E, D), (d, C), (B, a)
- Goal: determine if it is possible to circle at least one symbol in each pair without circling upper and lower case of same symbol.

2SAT

- This is a disguised version of the language 2SAT = \{ formulas in Conjunctive Normal Form with 2 literals per clause for which there exists a satisfying truth assignment \}
- CNF = “AND of ORs”
  \[(A, b), (E, D), (d, C), (b, a)\]
  \[(x_1 \lor \neg x_2) \land (x_2 \lor x_3) \land (\neg x_3 \lor x_4) \land (\neg x_2 \lor \neg x_1)\]
- satisfying truth assignment = assignment of TRUE/FALSE to each variable so that whole formula is TRUE

Algorithm for 2SAT

- Build a graph with separate nodes for each literal.
  - add directed edge \((x, y)\) iff formula includes clause \((\neg x \lor y)\) (equiv. to \(x \Rightarrow y\))
  \[n\]
  e.g. \((x_1 \lor \neg x_2) \land (x_5 \lor x_4) \land (\neg x_3 \lor x_4) \land (\neg x_2 \lor \neg x_1)\]
Algorithm for 2SAT

- **Proof (⇒)**
  - to construct a satisfying assign. (if no x with a path from x to ¬x and a path from ¬x to x):
    - pick unassigned literal s with no path from s to ¬s
    - assign it TRUE, as well as all nodes reachable from it; assign negations of these literals FALSE
    - note: path from s to t and s to ¬t implies path from ¬t to ¬s and t to ¬s
    - note: path s to t (assigned FALSE) implies path from ¬t to ¬s implies path from ¬l (assigned TRUE) to ¬s, so s already assigned at that point.

Algorithm for 2SAT

- **Algorithm:**
  - build derived graph
  - for every pair x, ¬x check if there is a path from x to ¬x and from ¬x to x in the graph

Running time of algorithm (input length n):
- O(n) to build graph
- O(n) to perform each check
- O(n) checks
- running time O(n²). 2SAT ∈ P.

Another puzzle

- Find an efficient algorithm to solve the following problem.
- **Input:** sequence of triples of symbols e.g. (A, b, C), (E, D, b), (d, A, C), (c, b, a)
- **Goal:** determine if it is possible to circle at least one symbol in each triple without circling upper and lower case of same symbol.

3SAT

- This is a disguised version of the language 3SAT = {formulas in Conjunctive Normal Form with 3 literals per clause for which there exists a satisfying truth assignment} e.g. (A, b, C), (E, D, b), (d, A, C), (c, b, a) (x₁V ¬x₂Vx₃) ∧ (x₂Vx₃V ¬x₄) ∧ (¬x₃Vx₂Vx₄) ∧ (¬x₂V ¬x₅ ∧ ¬x₁)

- observe that this language is in TIME(2ⁿ)

Time Complexity

**Key definition:** “P” or “polynomial-time” is P = Uₖ ≥ 1 TIME(nᵏ)

**Definition:** “EXP” or “exponential-time” is EXP = Uₖ ≥ 1 TIME(2ⁿᵏ)

- EXP ⊇ P
- We have seen 3SAT ∈ EXP.
  - does not rule out possibility that it is in P
- Is P different from EXP?
Theorem: for every proper complexity function \( f(n) \geq n \):
\[ \text{TIME}(f(n)) \subseteq \text{TIME}(f(2n)^3) \).

- Note: \( \text{P} \subseteq \text{TIME}(2^n) \not\subseteq \text{TIME}(2^{2n^3}) \not\subseteq \text{EXP} \)
- Most natural functions (and \( 2^n \) in particular) are proper complexity functions. We will ignore this detail in this class.

Proof idea:
- Use diagonalization to construct a language that is not in \( \text{TIME}(f(n)) \).
- Constructed language comes with a TM that decides it and runs in time \( f(2n)^3 \).

Proof of Time Hierarchy Theorem

- Proof:
  - \( \text{SIM} \) is TM deciding language \( \{ <M, x> : M \text{ accepts } x \text{ in } f(|x|) \text{ steps} \} \)
  - Claim: \( \text{SIM} \) runs in time \( g(n) = f(n)^3 \).
  - Define new TM \( D \): on input \( <M> \)
    - if \( \text{SIM} \) accepts \( <M, <M>> \), reject
    - if \( \text{SIM} \) rejects \( <M, <M>> \), accept
  - \( D \) runs in time \( g(2n) \)

- Proof (continued):
  - Suppose \( M \) in \( \text{TIME}(f(n)) \) decides \( L(D) \)
    - \( M(<M>) = \text{SIM}(<M, <M>>) \neq D(<M>) \)
    - but \( M(<M>) = D(<M>) \)
  - Contradiction.
Proof of Time Hierarchy Theorem

• Claim: there is a TM SIM that decides \( \{<M, x> : M \text{ accepts } x \text{ in } \leq f(|x|) \text{ steps}\} \) and runs in time \( g(n) = f(n)^3 \).

• Proof sketch: SIM has 4 work tapes
  • contents and "virtual head" positions for M's tapes
  • M's transition function and state
  • \( f(|x|) \) "+"s used as a clock
  • scratch space

Proof of Time Hierarchy Theorem

• Proof sketch (continued): 4 work tapes
  • contents and "virtual head" positions for M's tapes
  • M's transition function and state
  • \( f(|x|) \) "+"s used as a clock
  • scratch space
  – initialize tapes
  – simulate step of M, advance head on tape 3; repeat.
  – can check running time is as claimed.

So far…

• We have defined the complexity classes P (polynomial time), EXP (exponential time)