CS21
Decidability and Tractability

Lecture 18

February 14, 2022

Outline

- Examples of problems in P
- 2-SAT, 3-SAT
- The complexity class EXP
- Time Hierarchy Theorem
- hardness and completeness
  - an EXP-complete problem

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:

  RELPRIME = {<x, y> : x and y 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 <10, 22>:

x, y = 10, 22

x, y = 22, 10

x, y = 10, 2

x, y = 2, 0

reject

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>:

x, y = 24, 5

x, y = 5, 4

x, y = 4, 1

x, y = 1, 0

accept

Euclid's Algorithm

on input <x, y>:
- (1) repeat until y = 0
  - (2) set x = x mod y
  - (3) swap x, y
- x is the GCD(x, y). If x = 1, accept; otherwise reject

- every 2 times through loop, (x, y) each reduced by ½
- loops ≤ $2\max\{\log_2 x, \log_2 y\}$ = O(n = |<x, y>|); poly time for each loop

Claim: value of x reduced by ½ at every execution of (2) except possibly first one.

Proof:
- after (2) x < y
- after (3) x > y
- if x/2 ≥ y, then x mod y < y ≤ 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.

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)

  $(x_1 \lor \neg x_2) \land (x_5 \lor x_4) \land (\neg x_4 \lor x_3) \land (\neg x_2 \lor \neg x_1)$
  - satisfying truth assignment = assignment of TRUE/FALSE to each variable so that whole formula is TRUE

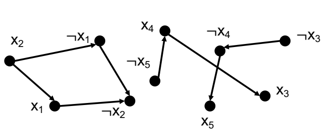
2SAT

Theorem: There is a polynomial-time algorithm deciding 2SAT ("2SAT $\in$ P").

Proof: algorithm described on next slides.

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 $x \Rightarrow y$)



e.g. $(x_1 \lor \neg x_2) \land (x_5 \lor x_4) \land (\neg x_4 \lor x_3) \land (\neg x_2 \lor \neg x_1)$

Algorithm for 2SAT

Claim: formula is unsatisfiable iff there is some variable x with a path from x to $\neg x$ and a path from $\neg x$ to x in derived graph.

- Proof ($\Leftarrow$)
  - edges represent implication $\Rightarrow$. By transitivity of $\Rightarrow$, a path from x to $\neg x$ means $x \Rightarrow \neg x$, and a path from $\neg x$ to x means $\neg x \Rightarrow x$.

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, implies path from s to ¬s
    - note: path s to t (assigned FALSE) implies path from ¬t (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 $O(n^2)$. 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)

  $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_5 \vee x_4 \vee \neg x_2) \wedge (\neg x_4 \vee x_1 \vee x_3) \wedge (\neg x_3 \vee \neg x_2 \vee \neg x_1)$

- observe that this language is in $\text{TIME}(2^n)$

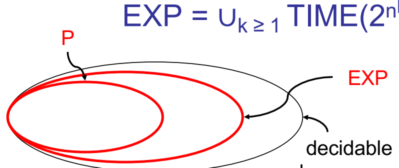
Time Complexity

Key definition: "P" or "polynomial-time" is

$$P = \cup_{k \geq 1} \text{TIME}(n^k)$$

Definition: "EXP" or "exponential-time" is

$$\text{EXP} = \cup_{k \geq 1} \text{TIME}(2^{n^k})$$



EXP

$$P = \cup_{k \geq 1} \text{TIME}(n^k)$$

$$\text{EXP} = \cup_{k \geq 1} \text{TIME}(2^{n^k})$$

- Note: P ⊆ EXP.
- We have seen 3SAT ∈ EXP.
  - **does not rule out possibility that it is in P**
- Is P different from EXP?

Time Hierarchy Theorem

Theorem: for every proper complexity function $f(n) \geq n$:

$TIME(f(n)) \subsetneq TIME(f(2n)^3)$.

- Note: $P \subseteq TIME(2^n) \subsetneq TIME(2^{(2n)^3}) \subseteq EXP$
- Most natural functions (and 2^n in particular) are proper complexity functions. We will ignore this detail in this class.

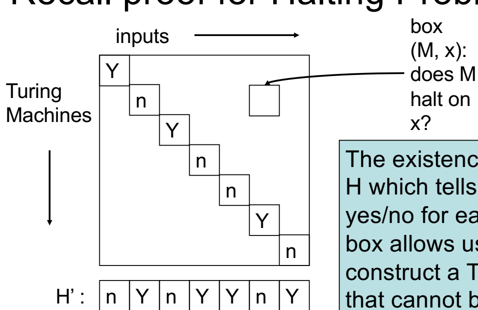
Time Hierarchy Theorem

Theorem: for every proper complexity function $f(n) \geq n$:

$TIME(f(n)) \subsetneq TIME(f(2n)^3)$.

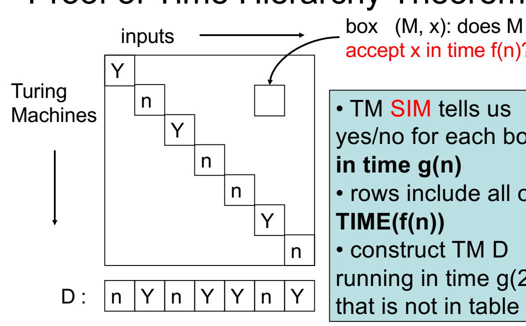
- Proof idea:
  - use diagonalization to construct a language that is not in TIME(f(n)).
  - constructed language comes with a TM that decides it and runs in time $f(2n)^3$.

Recall proof for Halting Problem



The existence of H which tells us yes/no for each box allows us to construct a TM H' that cannot be in the table.

Proof of Time Hierarchy Theorem



- TM SIM tells us yes/no for each box **in time g(n)**
- rows include all of **TIME(f(n))**
- construct TM D running in time g(2n) that is not in table

Proof of Time Hierarchy Theorem

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

Proof of Time Hierarchy Theorem

- Proof (continued):
  - suppose M in **TIME(f(n))** decides L(D)
    - M(<M>) = SIM(<M, <M>>) ≠ D(<M>)
    - but M(<M>) = D(<M>)
  - contradiction.

Proof of Time Hierarchy Theorem

- Claim: there is a TM SIM that decides

  {<M, x> : M accepts x in ≤ f(|x|) 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)