CS21
Decidability and Tractability

Lecture 17
February 15, 2019

Outline

- Extended Church-Turing Thesis
- The complexity class P
- Examples of problems in P

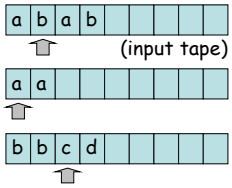
Multitape TMs

- Convenient to "program" multitape TMs rather than single ones
  - equivalent when talking about decidability
  - not equivalent when talking about time complexity

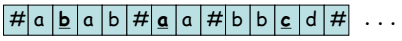
Theorem: Let $t(n)$ satisfy $t(n) \geq n$. Every multi-tape TM running in time $t(n)$ has an equivalent TM running in time $O(t(n)^2)$.

Multitape TMs

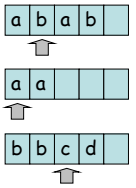
simulation of k-tape TM by single-tape TM:



- add new symbol **x** for each old x
- marks location of "virtual heads"



Multitape TMs



Repeat: O(t(n)) times

- scan tape, remembering the symbols under each virtual head in the state
  O(k t(n)) = O(t(n))
- make changes to reflect 1 step of M; if hit #, shift to right to make room.
  O(k t(n)) = O(t(n))

when M halts, erase all but 1st string
O(t(n))

Multitape TMs

- Moral: feel free to use k-tape TMs, but be aware of slowdown in conversion to TM
  - note: if $t(n) = O(n^c)$ then $t(n)^2 = O(n^{2c}) = O(n^{c'})$
  - note: if $t(n) = O(2^{n^\delta})$ for $\delta > 0$ then $t(n)^2 = O(2^{2n^\delta}) = O(2^{n^{\delta'}})$ for $\delta' > 0$
- high-level operations you are used to using can be simulated by TM with only polynomial slowdown
  - e.g., copying, moving, incrementing/decrementing, arithmetic operations +, -, *, /

Extended Church-Turing Thesis

- the belief that TMs formalize our intuitive notion of an efficient algorithm is:

> The "extended" Church-Turing Thesis
>
> everything we can compute in time t(n) on a physical computer can be computed on a Turing Machine in time $t(n)^{O(1)}$ (polynomial slowdown)

- quantum computers challenge this belief

Time Complexity

- interested in a coarse classification of problems. For this purpose,
  - treat any polynomial running time as "efficient" or "tractable"
  - treat any exponential running time as inefficient or "intractable"

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

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

Time Complexity

- Why polynomial-time?
  - insensitive to particular deterministic model of computation chosen
  - closed under modular composition
  - empirically: qualitative breakthrough to achieve polynomial running time is followed by quantitative improvements from impractical (e.g. n^{100}) to practical (e.g. n^3 or n^2)

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)

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

  – satisfying truth assignment = assignment of TRUE/FALSE to each variable so that whole formula is TRUE

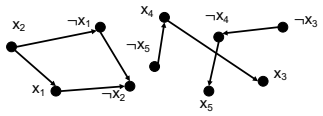
2SAT

Theorem: There is a polynomial-time algorithm deciding 2SAT ("2SAT ∈ 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 $(\neg x \vee y)$ (equiv. to $x \Rightarrow y$)



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

Algorithm for 2SAT

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

- Proof (⇐)
  - edges represent implication ⇒. By transitivity of ⇒, a path from x to ¬x means x ⇒ ¬x, and a path from ¬x to x means ¬x ⇒ 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

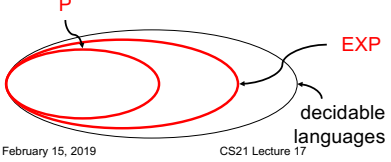
- Find an efficient algorithm to solve the following problem.
- Input: sequence of *triples* of symbols
  e.g. (A, b, C), (E, D, b), (d, A, C), (c, b, a)
- Goal: determine if it is possible to circle at least one symbol in each *triple* without circling upper and lower case of same symbol.

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 $TIME(2^n)$

Time Complexity

Key definition: "P" or "polynomial-time" is
$P = \cup_{k \geq 1} TIME(n^k)$
Definition: "EXP" or "exponential-time" is
$EXP = \cup_{k \geq 1} TIME(2^{n^k})$



EXP

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

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

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