CS21
Decidability and Tractability

Lecture 17
February 14, 2025

1

Time complexity

- Example: TM M deciding $L = \{0^k 1^k : k \geq 0\}$.

On input x:	
• scan tape left-to-right, reject if 0 to right of 1	# steps?
• repeat while 0's, 1's on tape: • scan, crossing off one 0, one 1	# steps?
• if only 0's or only 1's remain, reject; if neither 0's nor 1's remain, accept	# steps?

2

Time complexity

- We do not care about fine distinctions
  - e.g. how many additional steps M takes to check that it is at the left of tape
- We care about the behavior on large inputs
  - general-purpose algorithm should be "scalable"
  - overhead for e.g. initialization shouldn't matter in big picture

3

Time complexity

- Measure time complexity using asymptotic notation ("big-oh notation")
  - disregard lower-order terms in running time
  - disregard coefficient on highest order term
- example:

$$f(n) = 6n^3 + 2n^2 + 100n + 102781$$

  - "f(n) is order n^3"
  - write $f(n) = O(n^3)$

4

Asymptotic notation

<u>Definition</u>: given functions $f,g:\mathbf{N} \to \mathbf{R}^+$, we say $f(n) = O(g(n))$ if there exist positive integers c, n_0 such that for all $n \geq n_0$

$$f(n) \leq cg(n).$$

- meaning: f(n) is (asymptotically) less than or equal to g(n)
- if $g > 0$ can assume $n_0 = 0$, by setting

$$c' = \max_{0 \leq n \leq n_0}\{c, f(n)/g(n)\}$$

5

Asymptotic notation facts

each bound asymptotically less than next

- "logarithmic": $O(\log n)$
  - $\log_b n = (\log_2 n)/(\log_2 b)$
  - so $\log_b n = O(\log_2 n)$ for any constant b; therefore suppress base when write it
- "polynomial": $O(n^c) = n^{O(1)}$
  - also: $c^{O(\log n)} = O(n^{c'}) = n^{O(1)}$
- "exponential": $O(2^{n^\delta})$ for $\delta > 0$

6

Time complexity

On input x:

- scan tape left-to-right, reject if 0 to right of 1 — O(n) steps
- repeat while 0's, 1's on tape: — ≤ n repeats
  - scan, crossing off one 0, one 1 — O(n) steps
- if only 0's or only 1's remain, reject; if neither 0's nor 1's remain, accept — O(n) steps

- total = O(n) + nO(n) + O(n) = O(n^2)

7

Time complexity

- Recall:
  - language is a set of strings
  - a complexity class is a set of languages
  - complexity classes we've seen:
    - Regular Languages, Context-Free Languages, Decidable Languages, RE Languages, co-RE languages

Definition: TIME(t(n)) = {L : there exists a TM M that decides L in time O(t(n))}

8

Time complexity

- We saw that L = {0^k1^k : k ≥ 0} is in TIME(n^2).
- Book: it is also in TIME(n log n) by giving a more clever algorithm
- Can prove: There does not exist a (single tape) TM which decides L in time (asymptotically) less than n log n
- How about on a multitape TM?

9

Time complexity

- 2-tape TM M deciding L = {0^k1^k : k ≥ 0}.

On input x:

- scan tape left-to-right, reject if 0 to right of 1 — O(n)
- scan 0's on tape 1, copying them to tape 2 — O(n)
- scan 1's on tape 1, crossing off 0's on tape 2 — O(n)
- if all 0's crossed off before done with 1's reject
- if 0's remain after done with ones, reject; otherwise accept.

total: 3*O(n) = O(n)

10

Multitape TMs

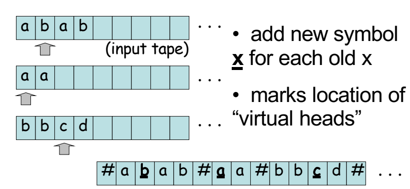
- Convenient to "program" multitape TMs rather than single ones
  - equivalent when talking about decidability
  - not equivalent when talking about time complexity

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

11

Multitape TMs

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

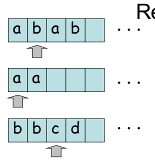


- add new symbol **x** for each old x
- marks location of "virtual heads"

12

Multitape TMs

Repeat: O(t(n)) times



- scan tape, remembering the symbols under each virtual head in the state

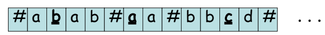
  O(k t(n)) = O(t(n))

- 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))



13

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 +, -, *, /

14

Extended Church-Turing Thesis

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

  > <u>The "extended" Church-Turing Thesis</u>
  >
  > 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

15

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"

<u>Key definition</u>: "P" or "polynomial-time" is

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

16

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)

17

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}?

18

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

19

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

20

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 ≤ $2\max\{\log_2 x, \log_2 y\} = O(n = |\langle x, y\rangle|)$; 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

21

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.

22