COMS W3261 — Lecture 11, Part 2: Time Complexity.

Recall: Asymptotic Analysis: "roughly comparing functions."

Def. Let f, g be some functions $f, g : \mathbb{N} \to \mathbb{R}^+$. (time it takes to run on certain input)

- $f(n) = O(g(n))$ if there exist positive integers n_0 and c such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

$f(n) = O(g(n)) \approx$ "In the long run, f is at most some constant times g."

"In the long run, f is ~smaller~ than g."

- $f(n) = \Omega(g(n))$ if there exist positive integers n_0 and c s.t. $f(n) \geq cg(n)$ for all $n \geq n_0$.

$(f(n) = \Omega(g(n)) \approx$ "In the long run, f is no ~smaller~ than g")

(Sipser 7.1)

Examples:

$$10 = O(\log_2(n)).$$

$$\log_2(n) = O(n).$$

$$n = \Omega(\log_2(n)).$$

$$5n = O(n) = \Omega(n).$$

$$16n^2 + n + 4 = O(n^2).$$

$$2^n = O(3^n).$$

$$\log_2(3^n) = n \cdot \log_2(3) = O(n).$$

Def. Let M be a deterministic single-tape TM that halts on all inputs. The running time or time complexity of M is a function $f : \mathbb{N} \to \mathbb{N}$ that indicates the maximum number of steps that

M takes to halt on any input of length n.

Def. We define the complexity class (set of languages) $\mathrm{TIME}(t(n))$ to be the set of all languages decided by some TM that runs in time $O(t(n))$.

Example. TMs that decide $A = \{0^k 1^k \mid k \geq 0\}$.

One approach:

M_1 = "On input w of length n:

1. Scan and reject if any 0 occurs to the right of a 1.
2. Shuttle back and forth, crossing off 0's and 1's, accept iff there are the same number of 0's and 1's."

this takes $\approx n$ steps.

shuttling: scan the tape once for every input symbol.

$\approx n$ to scan

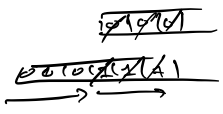
$\approx n$ input symbols

$\approx n^2$ time to shuttle.

Total time complexity: $a \cdot n$ (step 1) $+ b \cdot n^2$ (step 2) $+ c$

$= O(n^2)$.

Second approach (2-tape TM):



M_2 = "On input w:

$\approx n$ — 1. Scan for formatting (same as before.)

$O(n)$ steps — 2. Scan left to right until we see a 1, copying all 0's to the second tape.

$O(n)$ steps — 3. Scan the 1's, crossing off a 0 on the second tape for each 1. Accept/reject depending on whether the number of 0's equals the number of 1's."

Total runtime: $O(n)$ total.

Multiple tapes don't increase the set of languages we can decide, but they may increase how fast we can do it!

P: Polynomial Time.

Def. P is the class of languages that some deterministic TM decides in polynomial time. In math,

$$P = \bigcup_k \text{TIME}(n^k).$$

Why this class?

Idea: polynomial functions $\ll$ exponential functions.

At $n = 1000$, $n^3 = 1$ billion

$2^n >$ # of atoms in the universe.

Idea: polynomial · polynomial = polynomial.

(can use polynomial functions as subroutines and stay in P.)

Problem A decidable in time $n^c \in P$.

Problem B can be decided by deciding A n^d times.

(Runtime for B: now $O(n^d \cdot n^c) = O(n^{c+d})$.

Idea: "In practice," problems in P tend to have algorithms that run in 'small polynomials'.

Sipser: "All reasonable deterministic computational models are polynomially equivalent."

For example: computers (and TMs) can simulate each other with almost polynomial-time slow down.

Takeaway: If a problem is poly-time solvable on a TM, it's poly-time

solvable on all computers. 'Model independence.'

Takeaway: "P is the class of efficiently decidable languages."

Problems in P: All context-free languages.

PATH – does the input of a directed graph, s, t, contain a path between s and t?

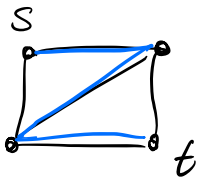
Relatively prime – $\langle x, y \rangle$, are x, y relatively prime?

Many, many, many others.

NP: Nondeterministic Polynomial Time.

Idea: A problem is verifiable if you can show me some certificate (evidence) that proves a given string is in the language.

Example. HAMPATH = $\{\langle G, s, t \rangle \mid G$ is a graph with a Hamiltonian path – a path that passes through every vertex exactly once – from s to t.$\}$



- Finding a path is hard.
- Showing a path is easy, if we can find it.

Def. A verifier for a language A is an algorithm V, where

$$A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some certificate } c. \}$$

V is a polynomial-time verifier if it runs in time polynomial in the length of w.

Def. NP is the class of efficiently verifiable languages – all languages

that have polynomial time verifiers.

Examples of Languages in NP:

Sudoku = $\{\langle P \rangle \mid P$ is a sudoku puzzle and P is solvable.$\}$

(certificate for any puzzle – valid solution.)

k-clique = $\{\langle G \rangle \mid G$ has a complete subgraph of size $k\}$

(certificate: subgraph)

Subset sum = $\{\langle S, t \rangle \mid S$ is a set of numbers with a subset that adds up to $t\}$.

(certificate: subset.)

(By the way: NP = all languages decidable by a nondeterministic TM in polynomial time.)

P and NP.

P: efficiently decidable

NP: efficiently verifiable.

$P \subseteq NP$. (Certificate for any string $w \in L$, where $L \in P$? Just show off the accepting computation of a decider on w.)

$NP \subseteq P$? Seems unlikely. Does being able to efficiently certify a solution imply that you can efficiently find one?

Conjectured that $NP \not\subseteq P$.

We don't know if P=NP.

CSOR 4231. – Algorithms.

COMS 4236 – Complexity. NP, P, and beyond:

COMS 4252 – Computational Learning Theory.

↓

Reminders: HW #6 due Monday. (8/9) Final 8/10, 8/11

Readings: 5.1 (Undecidability, reductions)
7.1 – 7.3 (Time complexity.)