Lecture 15

Reminder: Midterm review session Wed Feb 9 4:30 pm

CSE 431 Intro to Theory of Computation

Last time:

M has running time T, for $T: \mathbb{N} \to \mathbb{N}$, when

$T(n) = \max\{$ # of steps M takes in the worst case over all inputs $w \in \Sigma^*$ and over all computation choices if M is an NTM $\}$

$NTIME(T(\cdot)) = \{A :$ some multitape NTM with running time $O(T(n))$ decides $A\}$

A: language "set of strings" "yes/no question"

polynomial time $P = \bigcup_k TIME(n^k)$

non-deterministic polynomial time $NP = \bigcup_k NTIME(n^k)$

Note:
- $P \subseteq NP$ (every TM is an NTM)
- P is a set of languages (decision problems)

FP is the set of polynomial-time computable functions

PATH = $\{\langle G, s, t\rangle$: G is a directed graph with a path from s to $t\}$

Thm PATH $\in P$

Proof BFS, DFS, (or any general Graph Search) are all polynomial time

Note: Doesn't matter if graph is given by

input size n

poly time to convert between	adjacency matrix	$\Theta(N^2)$	N vertex
	adjacency lists	$\Theta((N+M)\log N)$	M edge
	edge lists	$\Theta(M \log N)$	graph

need $\Theta(\log N)$ bits to represent each vertex names

Encoding $\langle\ \rangle$: For computability it didn't matter much. For complexity, we assume efficient use of an alphabet of size ≥ 2 (w.l.o.g. binary)

RELPRIME = $\{\langle a, b\rangle$: a, b are integers with $\gcd(a,b)=1\}$

($\langle a, b\rangle$: binary encoding)

Thm RELPRIME $\in P$

Proof: Claim: Euclid's Algorithm is polynomial time

Let's recall how Euclid's algorithm works

$a_0 = a \qquad a_1 = b$

Repeatedly Compute $a_i = q_i a_{i-1} + a_{i+1}$ for $0 \le a_{i+1} < a_{i-1}$, q_i, a_{i+1} integers

long division with remainder

Computable in time $O(n^2)$ using grade school algorithm

$a_0 = q_1 a_1 + a_2$

$a_1 = q_2 a_2 + a_3$

$\vdots$

$a_{t-1} = q_{t-1} a_t + a_{t+1}$ $= \gcd(a,b)$ must be 1 for RELPRIME

$a_t = q_t a_{t+1} + 0$ (a_{t+2})

How many steps t?

Recall Fibonacci numbers

$F_0 = 0 = a_{t+2}$

$F_1 = 1 = a_{t+1}$ $\qquad F_i = F_{i-1} + F_{i-2}$

$a_{t-1} \ge a_{t+1} + a_{t+2} = F_1 + F_0 = F_2$

since $q_{t-1} \ge 1$

generally,

$a_i \ge a_{i+1} + a_{i+2}$

So $a_1 \ge F_t$ t-th Fibonacci #

Recall from CSE 311: $2^t \ge F_t \ge 2^{t/2} - 1$ for $t \ge 2$

Actually $F_t = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^t - \left(\frac{1-\sqrt{5}}{2}\right)^t\right)$

$\frac{1+\sqrt{5}}{2}$ = golden ratio $\approx 1.618\ldots$ ↑ exponential in t

$\left|\frac{1-\sqrt{5}}{2}\right| < 1$ ↑ basically rounded to near integer

$\therefore$ F_t takes $\Omega(t)$ bits

$\therefore$ input size in bits n is $\Omega(t)$

$\therefore$ # steps t is $O(n)$

$\therefore$ # of steps is linear in n & total runtime is $O(n^3)$ ∎

By contrast consider

FACTOR = $\{\langle N, k, \ell\rangle$: integer N has an integer factor m with $k \le m \le \ell\}$

Is FACTOR $\in$ P?

Note: $\langle N, 2, N-1\rangle \notin$ FACTOR iff N is prime

iff $\langle N, 2, \lceil\sqrt{N}\rceil\rangle \notin$ FACTOR

Can actually decide this in P!!

Surprising algorithm given in 2002 by Agarwal, Kayal, Saxena

(Randomized alg in text is more practical)

If FACTOR ∈ P can use binary search to actually find the factors of N and break RSA and break SSL and secure commerce

The question of whether FACTOR ∈ P is open

But FACTOR ∈ NP:

Nondeterministic algorithm:

polynomial time (On input <N, k, l>
Guess m between k and l
and check whether N is a multiple of m.

Thm Every context-free language is in P

Proof: see slides

Language Classes so far: