BU CS 332 – Theory of Computation

Lecture 19:

- More on P
- Nondeterministic time, NP

Reading:

Sipser Ch 7.2-7.3

Mark Bun

April 8, 2020

First topic: Time complexity

Last time: Answering the basic questions

1. How do we measure complexity? (as in CS 330)

2. Asymptotic notation (as in CS 330)

3. How robust is the TM model when we care about measuring complexity?

4. How do we mathematically capture our intuitive notion of “efficient algorithms”?

Running time and time complexity classes

Let $f : \mathbb{N} \to \mathbb{N}$

A TM M runs in time $f(n)$ if on every input $w \in \Sigma^n$,

M halts on w within at most $f(n)$ steps

$\mathrm{TIME}(f(n))$ is a class (i.e., set) of languages:

A language $A \in \mathrm{TIME}(f(n))$ if there exists a basic single-tape (deterministic) TM M that

1) Decides A, and

2) Runs in time $O(f(n))$

Complexity Class $\mathbf{P}$

Complexity class P

Definition: P is the class of languages decidable in polynomial time on a basic single-tape (deterministic) TM

$$\mathrm{P} = \bigcup_{k=1}^{\infty} \mathrm{TIME}(n^k)$$

- Class doesn't change if we substitute in another reasonable deterministic model (Extended Church-Turing)
- Cobham-Edmonds Thesis: Captures class of problems that are feasible to solve on computers

A note about encodings

We'll still use the notation $\langle \quad \rangle$ for "any reasonable" encoding of the input to a TM…but now we have to be more careful about what we mean by "reasonable"

How long is the encoding of a k-vertex graph…

… as an adjacency matrix?

… as an adjacency list?

How long is the encoding of a natural number k

… in binary?

… in decimal?

… in unary?

Describing and analyzing polynomial-time algorithms

- Due to Extended Church-Turing Thesis, we can still use high-level descriptions on multi-tape machines
- Polynomial-time is robust under composition: $\mathrm{poly}(n)$ executions of $\mathrm{poly}(n)$-time subroutines run on $\mathrm{poly}(n)$-size inputs gives an algorithm running in $\mathrm{poly}(n)$ time.

  => Can freely use algorithms we've seen before as subroutines if we've analyzed their runtime

- Need to be careful about size of inputs! (Assume inputs represented in binary unless otherwise stated.)

Examples of languages in $\mathbf{P}$

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

- $E_{\text{DFA}} = \{\langle D\rangle \mid D \text{ is a DFA that recognizes the empty language}\}$

Examples of languages in P

- $RELPRIME = \{\langle x, y \rangle \mid x \text{ and } y \text{ are relatively prime}\}$

- $PRIMES = \{\langle x \rangle \mid x \text{ is prime}\}$

2006 Gödel Prize citation



The 2006 Gödel Prize for outstanding articles in theoretical computer science is awarded to Manindra Agrawal, Neeraj Kayal, and Nitin Saxena for their paper "PRIMES is in P."

In August 2002 one of the most ancient computational problems was finally solved….

A polynomial-time algorithm for $PRIMES$?

Consider the following algorithm for $PRIMES$

On input $\langle x \rangle$:

For $b = 2, 3, 5, \ldots, \sqrt{x}$:

Try to divide x by b

If b divides x, accept

If all b fail to divide x, reject

How many divisions does this algorithm require in terms of $n = |\langle x \rangle|$?

CFG Generation

Theorem: Every context-free language is in P

Chomsky Normal Form for CFGs:

- Can have a rule $S \to \varepsilon$
- All remaining rules of the form $A \to BC$ or $A \to a$
- Cannot have S on the RHS of any rule

Lemma: Any CFG can be converted into an equivalent CFG in Chomsky Normal Form

Lemma: If G is in Chomsky Normal Form, any nonempty string w that can be derived from G has a derivation with at most $2|w| - 1$ steps

CFG Generation

Theorem: Every context-free language is in P

Proof attempt: Let A be a CFL with a CFG G in Chomsky Normal Form. Define a TM M recognizing A:

On input w: (Let $n = |w|$)

1. Enumerate all strings derivable in $\leq 2n - 1$ steps
2. If any of these strings equal w, accept. Else, reject.

What is the running time of this algorithm?

A better idea: Use dynamic programming

- Identify subproblems
- Record solutions to subproblems in a table
- Solve bigger subproblems by combining solutions to smaller subproblems

Examples of languages in P

Problem	Description	Algorithm	Yes	No
MULTIPLE	Is x a multiple of y?	Grade school division	51, 17	51, 16
RELPRIME	Are x and y relatively prime?	Euclid (300 BCE)	34, 39	34, 51
PRIMES	Is x prime?	AKS (2002)	53	51
all CFLs (e.g. the language of balanced parentheses and brackets)	Is a given string in a fixed CFL? (E.g., is the string of parentheses and brackets balanced?)	Dynamic programming	Depends on the language; e.g. (([])[])	Depends on the language; e.g. ([)], (()
LSOLVE	Is there a vector x that satisfies Ax = b?	Gauss-Edmonds elimination	$\begin{bmatrix}0 & 1 & 1\\ 2 & 4 & -2\\ 0 & 3 & 15\end{bmatrix}, \begin{bmatrix}2\\ 4\\ 36\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0\\ 1 & 1 & 1\\ 0 & 1 & 1\end{bmatrix}, \begin{bmatrix}1\\ 1\\ 1\end{bmatrix}$

Beyond polynomial time

Definition: EXP is the class of languages decidable in exponential time on a basic single-tape (deterministic) TM

$$\mathrm{EXP} = \bigcup_{k=1}^{\infty} \mathrm{TIME}(2^{n^k})$$

Nondeterministic Time and NP

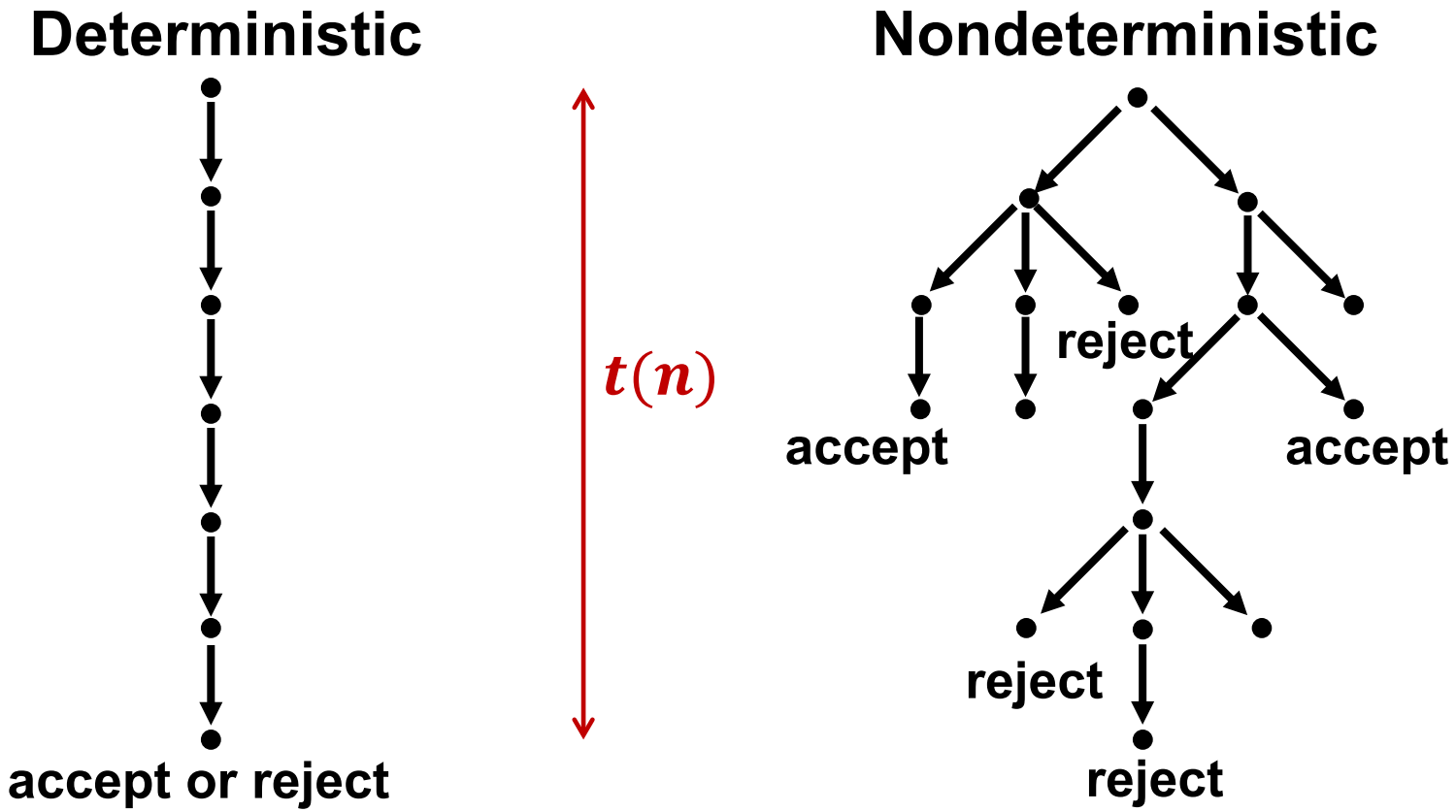
Nondeterministic time

Let $f : \mathbb{N} \rightarrow \mathbb{N}$

A NTM M runs in time $f(n)$ if on every input $w \in \Sigma^n$,

M halts on w within at most $f(n)$ steps on every computational branch

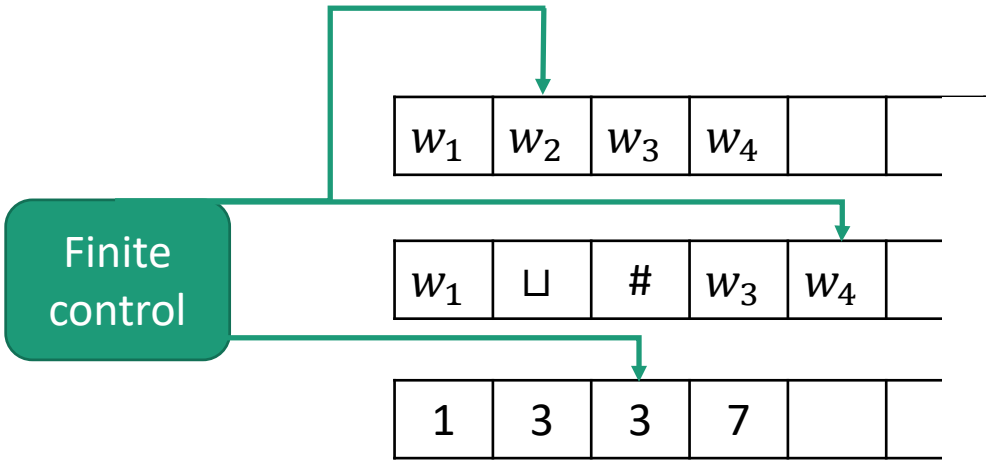
Deterministic vs. nondeterministic time



Deterministic vs. nondeterministic time

Theorem: Let $t(n) \geq n$ be a function. Every NTM running in time $t(n)$ has an equivalent single-tape TM running in time $2^{O(t(n))}$

Proof: Simulate NTM by 3-tape TM



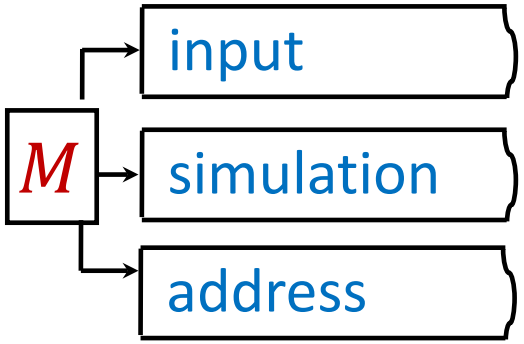
Input w to N (read-only)

Simulation tape (run N on w using nondeterministic choices from tape 3)

Address in computation tree

Deterministic vs. nondeterministic time

Theorem: Let $t(n) \geq n$ be a function. Every NTM running in time $t(n)$ has an equivalent single-tape TM running in time $2^{O(t(n))}$



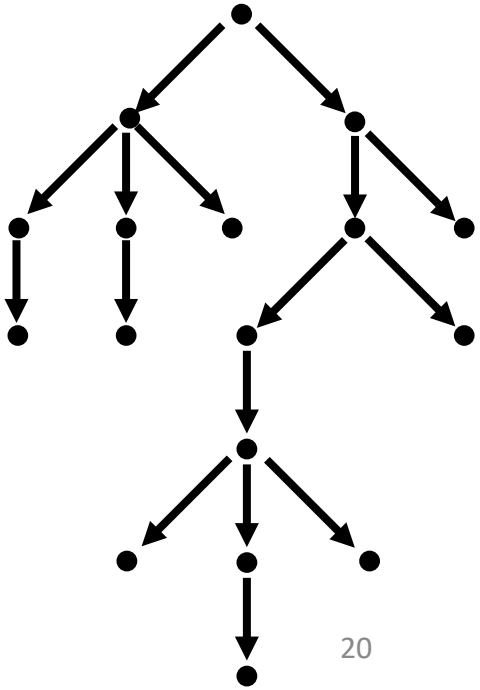
Proof: Simulate NTM by 3-tape TM

- # leaves:
- # nodes:

Running time:

To simulate one root-to-node path:

Total time:



Deterministic vs. nondeterministic time

Theorem: Let $t(n) \geq n$ be a function. Every NTM running in time $t(n)$ has an equivalent single-tape TM running in time $2^{O(t(n))}$

Proof: Simulate NTM by 3-tape TM in time $2^{O(t(n))}$

We know that a 3-tape TM can be simulated by a single-tape TM with quadratic overhead, hence we get running time

$$(2^{O(t(n))})^2 = 2^{2 \cdot O(t(n))} = 2^{O(t(n))}$$

Difference in time complexity

Extended Church-Turing Thesis:

At most polynomial difference in running time between all (reasonable) deterministic models

At most exponential difference in running time between deterministic and nondeterministic models

Nondeterministic time

Let $f : \mathbb{N} \rightarrow \mathbb{N}$

A NTM M runs in time $f(n)$ if on every input $w \in \Sigma^n$,

M halts on w within at most $f(n)$ steps on every computational branch

$\mathrm{NTIME}(f(n))$ is a class (i.e., set) of languages:

A language $A \in \mathrm{NTIME}(f(n))$ if there exists an NTM M that

1) Decides A, and
2) Runs in time $O(f(n))$

NTIME explicitly

A language $A \in \mathrm{NTIME}(f(n))$ if there exists an NTM M such that, on every input $w \in \Sigma^*$

1. Every computational branch of M halts in either the accept or reject state within $f(|w|)$ steps

2. $w \in A$ iff there exists an accepting computational branch of M on input w

3. $w \notin A$ iff every computational branch of M rejects on input w (or dies with no applicable transitions)

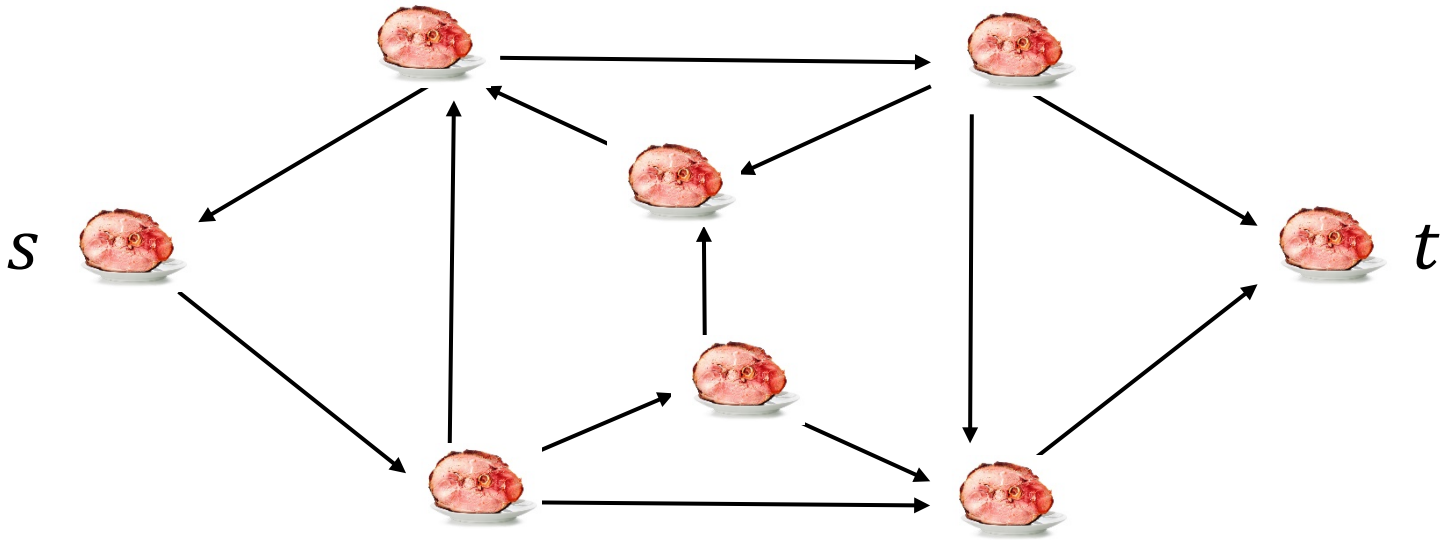
Complexity class NP

Definition: NP is the class of languages decidable in polynomial time on a nondeterministic TM

$$\mathrm{NP} = \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k)$$

Hamiltonian Path

$HAMPATH = \{\langle G, s, t\rangle \mid G$ is a directed graph and there is a path from s to t that passes through every vertex exactly once$\}$



$HAMPATH \in$ NP

The following nondeterministic algorithm accepts in time $O(n^3)$, where $n = |\langle G, s, t \rangle|$, adjacency matrix encoding

On input $\langle G, s, t \rangle$: (Vertices of G are numbers $1, \ldots, k$)

1. **Nondeterministically** guess a sequence $c_1, c_2, \ldots, c_k$ of numbers $1, \ldots, k$
2. Check that $c_1, c_2, \ldots, c_k$ is a permutation: Every number $1, \ldots, k$ appears exactly once
3. Check that $c_1 = s$, $c_k = t$, and there is an edge from every c_i to c_{i+1}
4. Accept if all checks pass, otherwise, reject.

An alternative characterization of NP

"Languages with polynomial-time verifiers"

A verifier for a language L is a deterministic algorithm V such that $w \in L$ iff there exists a string c such that $V(\langle w, c \rangle)$ accepts

Running time of a verifier is only measured in terms of $|w|$

V is a polynomial-time verifier if it runs in time polynomial in $|w|$ on every input $\langle w, c \rangle$

(Without loss of generality, $|c|$ is polynomial in $|w|$, i.e., $|c| = O(|w|^k)$ for some constant k)

$HAMPATH$ has a polynomial-time verifier

Certificate c:

Verifier V:

On input $\langle G, s, t, c \rangle$: (Vertices of G are numbers $1, \ldots, k$)

1. Check that $c_1, c_2, \ldots, c_k$ is a permutation: Every number $1, \ldots, k$ appears exactly once
2. Check that $c_1 = s$, $c_k = t$, and there is an edge from every c_i to c_{i+1}
3. Accept if all checks pass, otherwise, reject.

NP is the class of languages with polynomial-time verifiers

Theorem: A language $L \in \mathrm{NP}$ iff there is a polynomial-time verifier for L

Proof: