

# BU CS 332 – Theory of Computation

<https://forms.gle/SAavsmjbSvm6gnSj6>



## Lecture 20:

- P Examples
- NP

Reading:

Sipser Ch 7.2-7.3

Mark Bun

April 17, 2024

# Complexity class P

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

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

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

# Check your type checker: P



Consider the following computational problem: Given two numbers  $x, y$  (written in binary), output their sum  $x + y$  (in binary). Which of the following is true?

- a) This is a problem in P
- b) This problem is not in P because it cannot be solved by a Turing machine (i.e., it's undecidable)
- c) This problem is not in P because it cannot be solved in polynomial time
- d) This problem is not in P because it is not a decision problem (i.e., does not correspond to a language)

# A note about encodings

We'll still use the notation  $\langle \ \rangle$  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  $V$ -vertex,  $E$ -edge 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**:  $\text{poly}(n)$  executions of  $\text{poly}(n)$ -time subroutines run on  $\text{poly}(n)$ -size inputs gives an algorithm running in  $\text{poly}(n)$  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 P

*PATH* =

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

Idea: Breadth-first search

Assume  $G$  presented as adjacency matrix

“On input  $\langle G, s, t \rangle$ :

1. Mark start vertex  $s$
2. For  $i = 1, 2, \dots, |V|$ :
3. Mark all neighbors of currently marked vertices
4. If  $t$  is marked, **accept**. Else, **reject**.”

# Examples of languages in $P$

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

# 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  $b = 2, 3, 4, 5, \dots, \sqrt{x}$ :

- Try to divide  $x$  by  $b$
- If  $b$  divides  $x$ , **reject**

If all  $b$  fail to divide  $x$ , **accept**

How many divisions does this algorithm require in terms of  $n = |\langle x \rangle|$ ?    a)  $O(\sqrt{n})$     b)  $O(n)$     c)  $2^{O(\sqrt{n})}$     d)  $2^{O(n)}$

# Beyond polynomial time

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

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

# Why study P ?

Criticism of the Cobham-Edmonds Thesis:

- Algorithms running in time  $n^{100}$  aren't really efficient

**Response:** Runtimes improve with more research

- Does not capture some physically realizable models using randomness, quantum mechanics

**Response:** Randomness may not change P, useful principles



$TIME(n)$  vs.  $TIME(n^2)$



$P$  vs.  $EXP$



decidable vs.  
undecidable

# Nondeterministic Time and NP

# Extended Church-Turing Thesis

Every “reasonable” (physically realizable) model of computation can be simulated by a basic, single-tape TM with only a **polynomial** slowdown.

E.g., doubly infinite TMs, multi-tape TMs, RAM TMs

Does **not** include nondeterministic TMs (not reasonable)

# Nondeterministic time

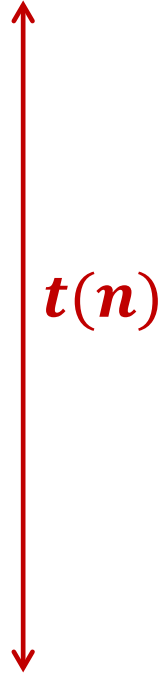
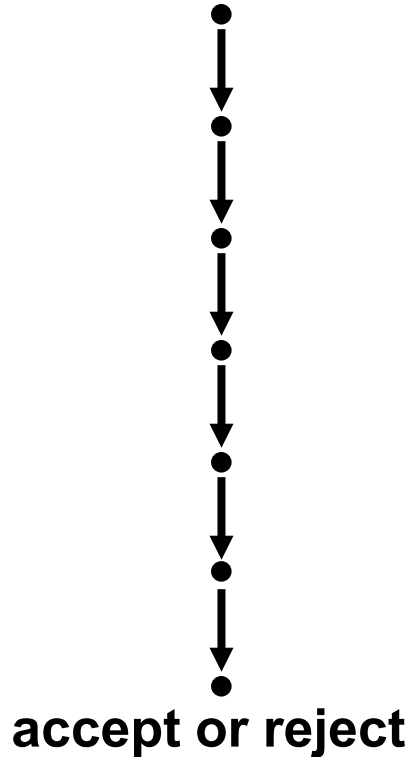
Let  $t: \mathbb{N} \rightarrow \mathbb{N}$

NTM  $M$  runs in time  $t(n)$  if:

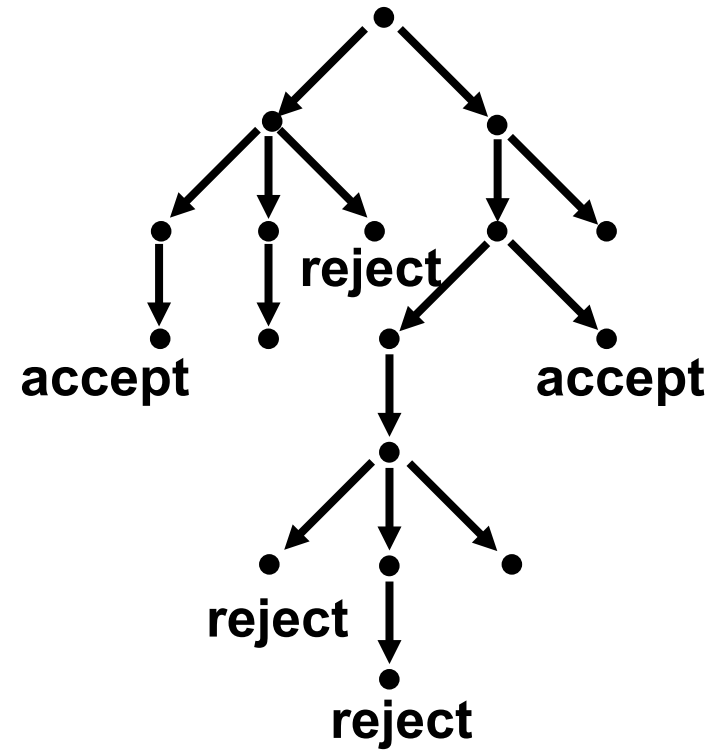
For every  $n$  and **every** input  $w \in \Sigma^n$ ,  $M$  halts on  $w$  within at most  $t(n)$  steps on **every computational branch**

# Deterministic vs. nondeterministic time

## Deterministic



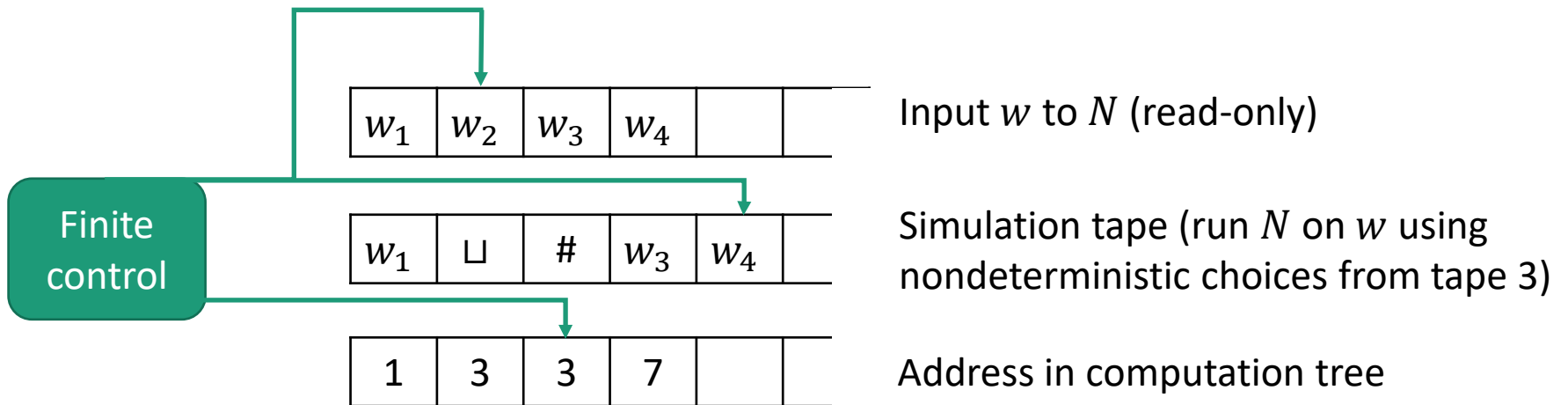
## Nondeterministic



# Deterministic vs. nondeterministic time

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

**Proof:** Simulate NTM by 3-tape TM



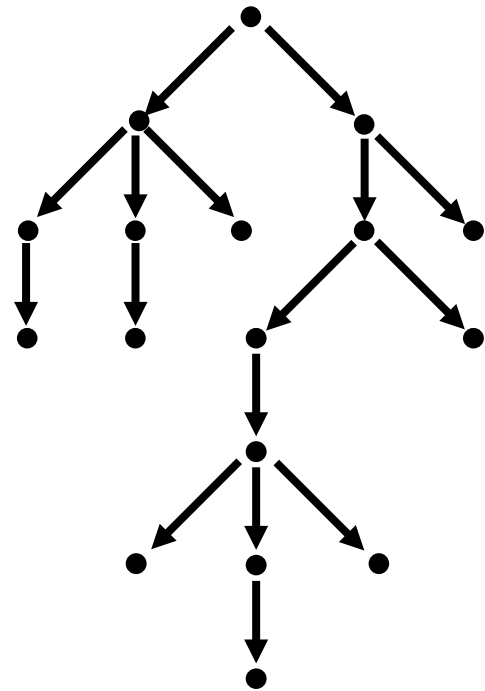


# Counting leaves



What is an upper bound on the maximum number of nodes in a tree with branching factor  $b$  and depth  $t$ ?

- a)  $bt$
- b)  $b^t$
- c)  $t^b$
- d)  $2^t$



# Deterministic vs. nondeterministic time

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

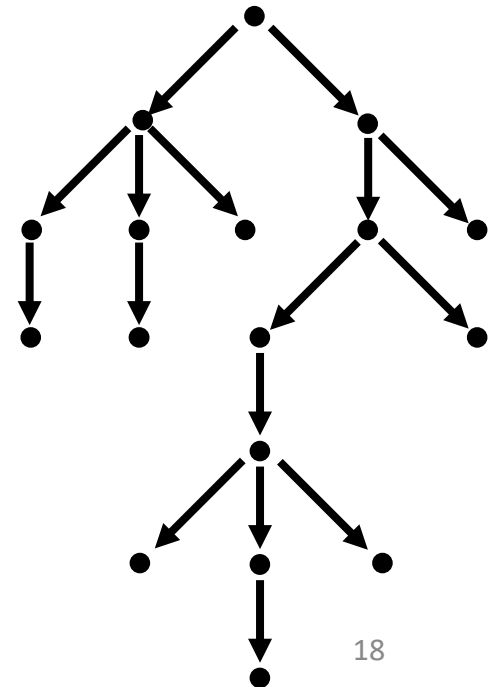
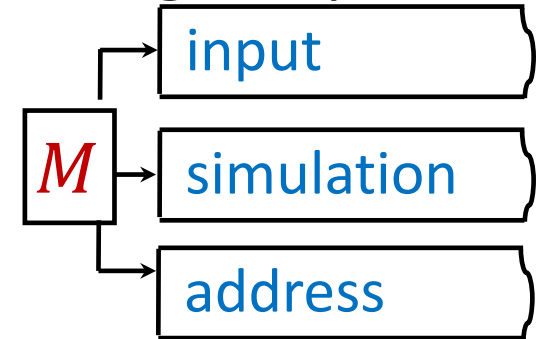
**Proof:** Simulate NTM by 3-tape TM

- # 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 deterministic 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  $f : \mathbb{N} \rightarrow \mathbb{N}$

NTM  $M$  runs in time  $f(n)$  if:

For every  $n$  and **every** input  $w \in \Sigma^n$ ,  $M$  halts on  $w$  within at most  $f(n)$  steps on **every computational branch**

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

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

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

# NTIME explicitly

A language  $A \in \text{NTIME}(f(n))$  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. If  $w \in A$ , then **there exists** an accepting computational branch of  $M$  on input  $w$
3. If  $w \notin A$ , then **every** computational branch of  $M$  rejects on input  $w$

# Complexity class NP



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

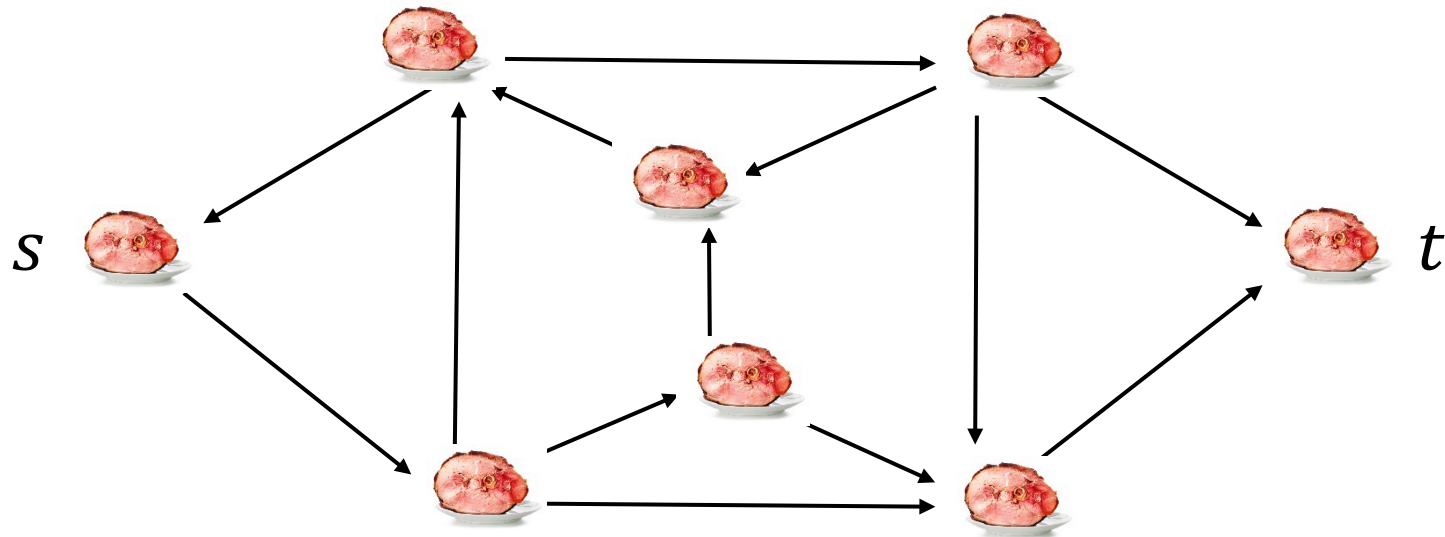
$$\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k)$$

Which of the following are definitely true about NP?

- a)  $P \subseteq \text{NP}$
- b)  $\text{NP} \subseteq P$
- c)  $\text{NP} \not\subseteq P$
- d)  $\text{NP} \subseteq \text{EXP}$
- e)  $\text{EXP} \subseteq \text{NP}$

# Hamiltonian Path

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





# *HAMPATH* $\in$ NP

The following nondeterministic algorithm decides *HAMPATH* in polynomial time:

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

1. **Nondeterministically** guess a sequence  $c_1, c_2, \dots, c_k$  of numbers  $1, \dots, k$
2. Check that  $c_1, c_2, \dots, c_k$  is a permutation: Every number  $1, \dots, k$  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**.

# Analyzing the algorithm

Need to check:

1) Correctness

2) Running time

# An alternative characterization of NP

“Languages with polynomial-time verifiers”

How did we design an NTM for HAMPATH?

- Given a candidate path, it is easy (poly-time) to check whether this path is a Hamiltonian path
- We designed a poly-time NTM by nondeterministically guessing this path and then checking it
- Lots of problems have this structure (CLIQUE, 3-COLOR, COMPOSITE,...). They might be hard to solve, but a candidate solution is easy to check.

# 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  $1, \dots, k$ )

1. Check that  $c_1, c_2, \dots, c_k$  is a permutation: Every number  $1, \dots, k$  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  $L \in \text{NP}$  iff there is a polynomial-time verifier for  $L$