

BU CS 332 – Theory of Computation

Lecture 19:

Reading:

- Time bounded computation Sipser Ch 7.1-7.3
- The Classes P, NP

Ran Canetti

November 17, 2020

Where we are in CS 332

Automata & Formal Languages	Computability	Complexity
-----------------------------	---------------	------------

Previous unit: **Computability theory**

What kinds of problems can / can't computers solve?

Final unit: **Complexity theory**

What kinds of problems can / can't computers solve under **constraints on their computational resources?**






Running time analysis

Time complexity of a TM (algorithm) = maximum number of steps it takes on a worst-case input

Formally: Let $f : \mathbb{N} \rightarrow \mathbb{N}$. A TM M runs in time $f(n)$ if on every input $w \in \Sigma^*$, M halts on w within at most $f(n)$ steps

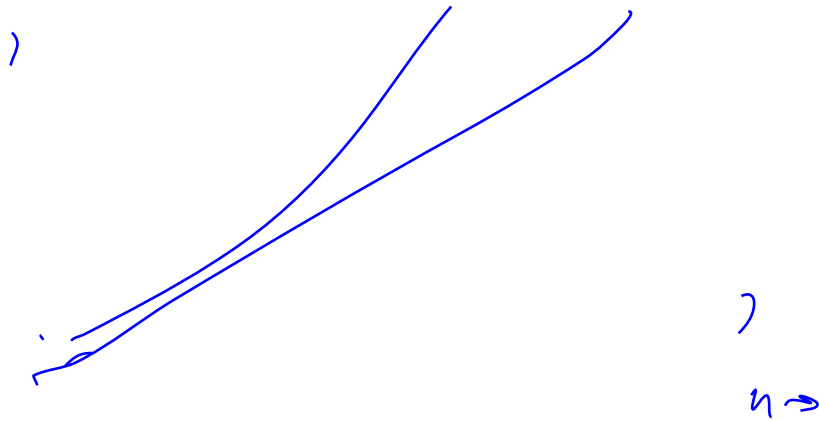
- Focus on worst-case running time: Upper bound of $f(n)$ must hold for all inputs of length n
- Exact running time $f(n)$ does not translate well between computational models / real computers. Instead focus on **asymptotic complexity**.

A handy-dandy chart

Notation	... means ...	Think...	Example	$\lim_{n \leftarrow \infty} \frac{f(n)}{g(n)}$
$f(n)=O(g(n))$ 	$\exists c>0, n_0>0, \forall n > n_0 :$ $f(n) < cg(n)$	Upper bound	$100n^2$ $= O(n^3)$	If it exists, it is $< \infty$
$f(n)=\Omega(g(n))$ 	$\exists c>0, n_0>0, \forall n > n_0 :$ $cg(n) < f(n)$	Lower bound	2^n $= \Omega(n^{100})$	If it exists, it is > 0
$f(n)=\Theta(g(n))$ 	both of the above: $f=\Omega(g)$ and $f=O(g)$	Tight bound	$\log(n!)$ $= \Theta(n \log n)$	If it exists, it is > 0 and $< \infty$
$f(n)=o(g(n))$ 	$\forall c>0, \exists n_0>0, \forall n > n_0 :$ $f(n) < cg(n)$	Strict upper bound	$n^2 = o(2^n)$	Limit exists, $=0$
$f(n)=\omega(g(n))$ 	$\forall c>0, \exists n_0>0, \forall n > n_0 :$ $cg(n) < f(n)$	Strict lower bound	n^2 $= \omega(\log n)$	Limit exists, $=\infty$

How fast do functions grow?

$$o(1) < \log n < n^{\frac{1}{c}} < n < \boxed{n^c} < n^{\log n} < 2^n$$



Time complexity classes

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

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

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

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

How much does the runtime depend on the model?

- In reality: The actual computer used can make a difference, but not an “essential” one.
 - In our mathematical model:
 - Number of tapes matters:
 - With two tapes can recognize $\{0^n 1^n\}$ in linear time $O(n)$
 - With one tape, any non-regular language requires $\Omega(n \log n)$ time
 - But doesn't matter too much:
 - Can simulate a multi-tape TM on a single tape one with quadratic overhead.
(in fact can be done with logarithmic overhead)
- $t(n) \rightarrow t^2(n)$
- $t(n) \rightarrow t(n) \log(t(n))$

Extended Church-Turing Thesis

Every “reasonable” 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)

Possible counterexamples? Randomized computation, parallel computation, DNA computing, quantum computation

Extended Church-Turing Thesis

Every “reasonable” 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)

Possible counterexamples? Randomized computation, parallel computation, DNA computing, quantum computation

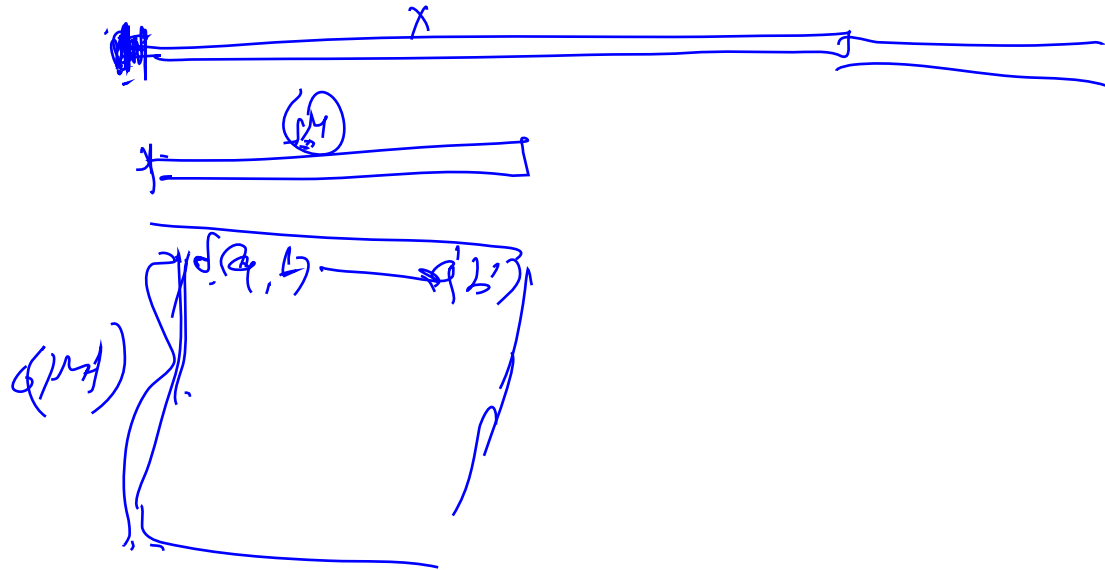
The overhead of universal simulation

- We saw: There exists a TM U such that

$$\forall M, x \quad \underline{U(\langle M, x \rangle)} = \underline{M(x)}$$

- If M halts on x within t steps, How many steps does U(⟨M, x⟩) take to halt?

U CHS X



1, X

polynomial slow down

Complexity class P

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

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

$$O(n^{\log n}) \notin P$$

$$O(n^{\log n})$$

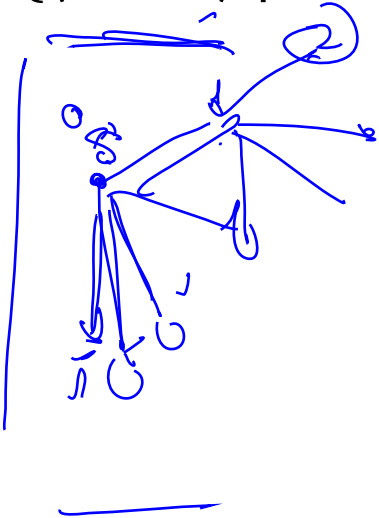
- Class doesn't change if we substitute in another reasonable deterministic model (Extended Church-Turing)
- **Cobham-Edmonds Thesis:** Captures class of problems that are feasible to solve on computers

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.)

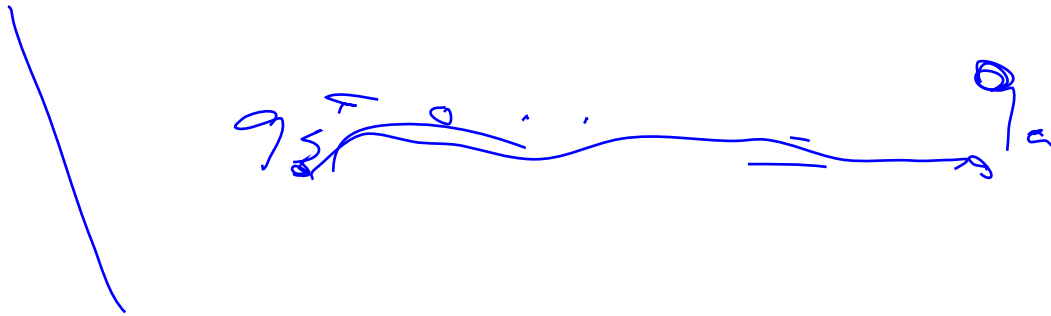
Examples of languages in P

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



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

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

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

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 $\langle x \rangle$:

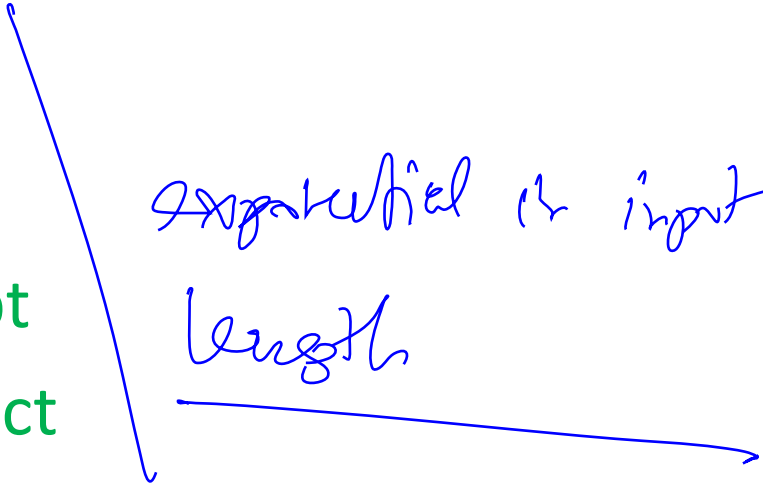
For $b = \underline{2}, 3, 5, \dots, \underline{\sqrt{x}}$:

Try to divide x by b

If b divides x , **accept**

If all b fail to divide x , **reject**

exponential in input
length



A polynomial-time algorithm for *PRIMES*?

Consider the following algorithm for *PRIMES*

On input $\langle x \rangle$:

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

 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 $n = |\langle x \rangle|$?

Examples of languages in P

Problem	Description	Algorithm	Yes	No
MULTIPLE	Is x a <u>multiple of y</u> ?	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

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

Nondeterministic Time and NP

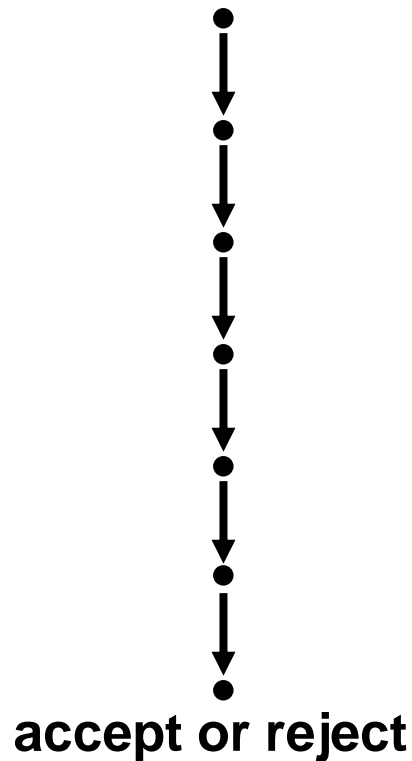
Nondeterministic time

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

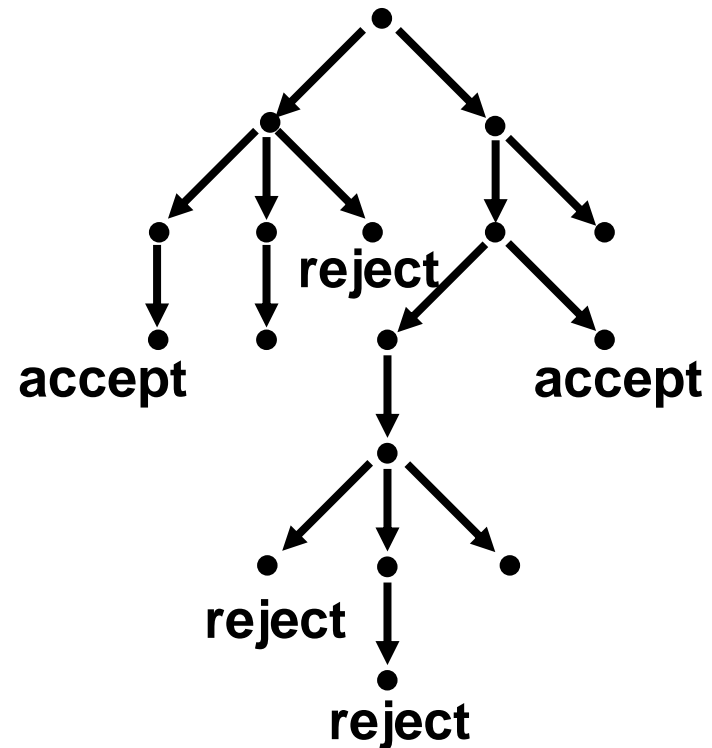
Deterministic vs. nondeterministic time

Deterministic



$t(n)$

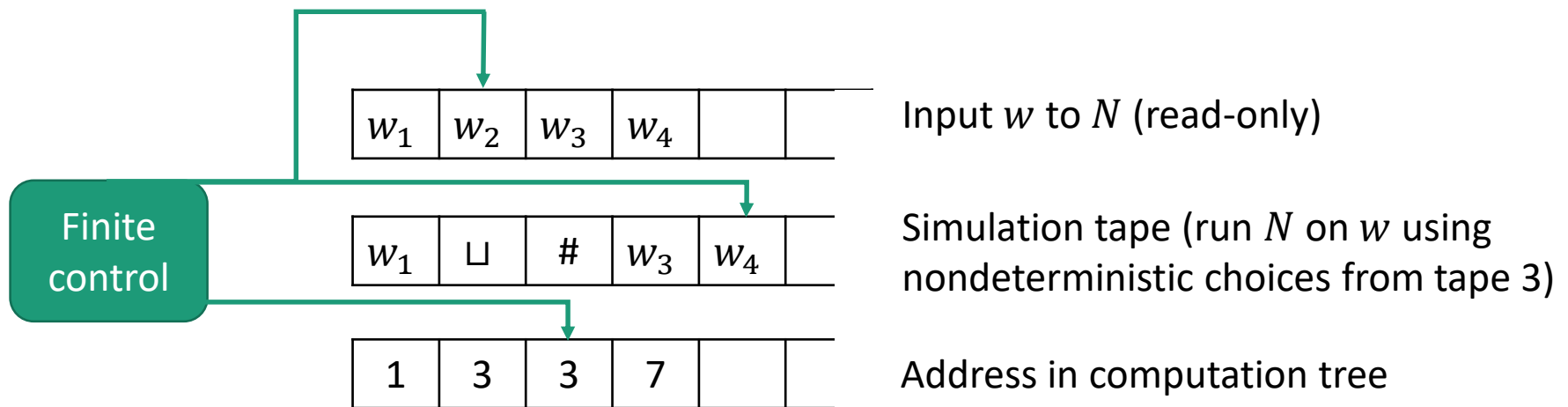
Nondeterministic



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



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

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 $2^{O(t(n))}$

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

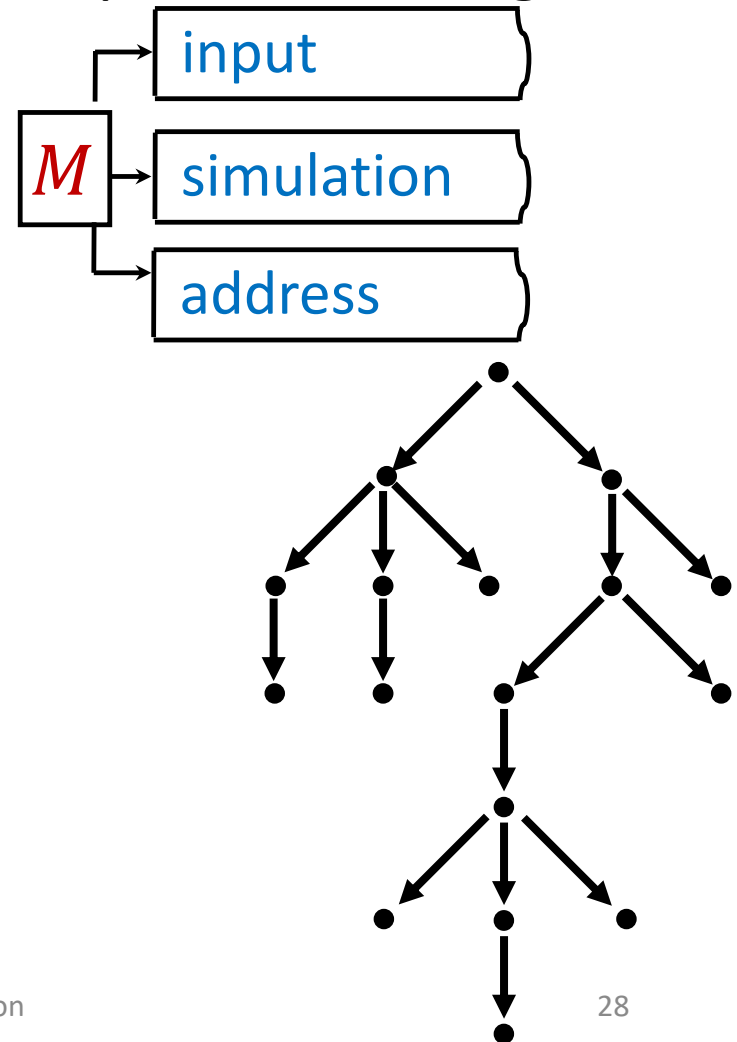
- # leaves:

- # 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

$\text{NTIME}(f(n))$ 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 $x \in \Sigma^*$

1. Every computational branch of M halts in either the accept or reject state within $f(|x|)$ steps
2. $wx \in A$ iff **there exists** an accepting computational branch of M on input w
3. $x \notin A$ iff **every** computational branch of M rejects on input x (or dies with no applicable transitions)

Complexity class NP

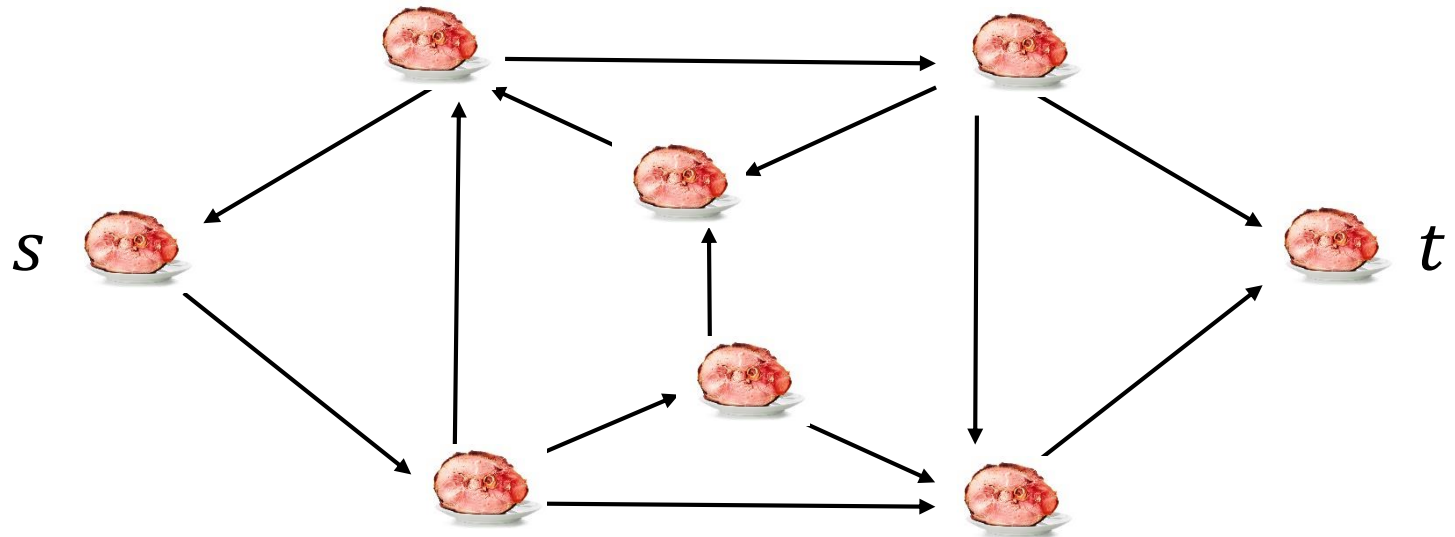
Definition: NP is the class of languages decidable in polynomial time on a nondeterministic TM

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



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 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 $1, \dots, k$)

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**.

An alternative characterization of NP

“Languages with polynomial-time verifiers”

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

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

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

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

Proof: