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

$$\mathrm{P} = \bigcup_{k=1}^{\infty} \mathrm{TIME}(n^k)$$ $= \{$ languages L | $\exists$ TM M deciding L in poly time $\}$

$= \mathrm{TIME}(n) \cup \mathrm{TIME}(n^2) \cup \mathrm{TIME}(n^3) \cup \ldots$

- 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

$= \{ \text{languages decidable in poly-time} \}$

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?

$\text{VERIFY\_ADD} = \{ \langle x, y, z \rangle \mid x + y = z \} \in P$

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 $\langle\quad\rangle$ 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 V-vertex, E-edge graph…

… as an adjacency matrix? $O(|V|^2) \leftarrow$

… as an adjacency list? $O(|V| + |E|) \leftarrow$

Alg that runs in time poly in # vertices, or # edges, is a "poly-time" alg

How long is the encoding of a natural number k

$k=5$ $\quad 101$

… in binary? $|\langle k\rangle| \approx \log_2 k$

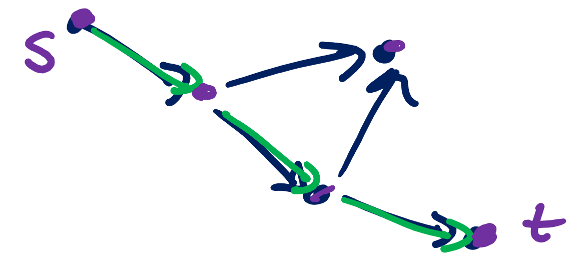
… in decimal? $|\langle k\rangle| = \log_{10} k = O(\log_2 k)$

… in unary? $\langle k\rangle = \underbrace{1111\ldots1}_{k}$ $\quad |\langle k\rangle| = k$

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.
  $\Rightarrow$ 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 <u>binary</u> unless otherwise stated.)

Examples of languages in $\mathbf{P}$



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

Idea: Breadth-first search

Assume G presented as adjacency matrix

$\approx |V|^2$ $\log|V|$ $\log|V|$ $\Rightarrow |\langle G,s,t\rangle| = O(|V|^2)$

$\| $
n

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

1. Mark start vertex s $\leftarrow O(|V|^2)$
2. For $i = 1, 2, \ldots, |V|$: $\leftarrow O(|V|)$ iterations
3. Mark all neighbors of currently marked vertices $\leftarrow O(|V|^2)$
4. If t is marked, accept. Else, reject." $O(|V|^2)$

Runtime:

$$\underbrace{O(|V|^2)}_{\text{step 1}} + \underbrace{O(|V|)\cdot O(|V|^2)}_{\text{step 2}} + \underbrace{O(|V|^2)}_{\text{step 3}}$$

$$= O(|V|^3) = O(n^{3/2})$$

Correctness: If $\langle G,s,t\rangle \in PATH$ then $\exists$ a path from s to t in G, using $\leq |V|$ vertices $\Rightarrow$ BFS marks t w/in $|V|$ iterations $\Rightarrow$ alg accepts

If $\langle G,s,t\rangle \notin PATH$, then no path from s to t, so BFS will never find t $\Rightarrow$ alg. rejects.

Examples of languages in $\mathbf{P}$

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

$E_{DFA} \in P$ because BFS solves it!

Another way: reduce to PATH

On input $\langle D \rangle$:

For each accept state q in D:

Check if $\langle G, q_0, q \rangle \in$ PATH. If yes <u>reject</u>.

graph underlying D

start state

candidate accept state

<u>Accept</u>.

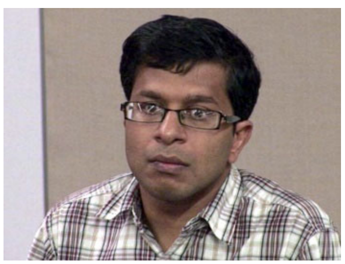
Examples of languages in $\mathbf{P}$

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

i.e. $\gcd(x,y) = 1$

Euclidean algorithm solves in poly time

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


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 $PRIMES$

$n = |\langle x \rangle|$ ← binary

$\Rightarrow$ x could be as big as $\approx 2^n$

On input $\langle x \rangle$:

For $b = 2, 3, 4, 5, \ldots, \sqrt{x}$:

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

If all b fail to divide x, accept

divisions $= \sqrt{x} \approx \sqrt{2^n} = 2^{n/2}$

$= 2^{O(n)}$

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

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

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

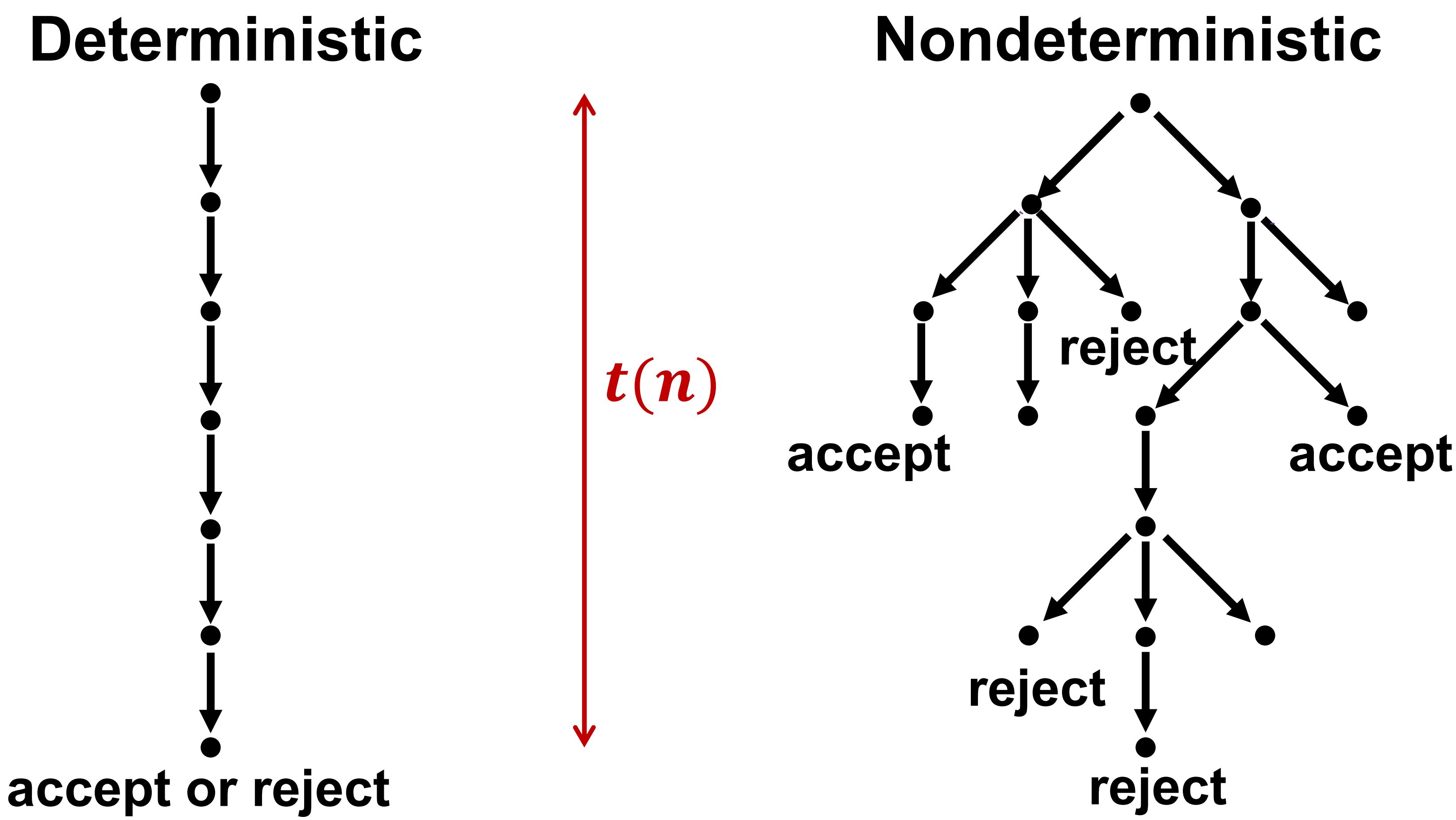
input length ↓ ← run time

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

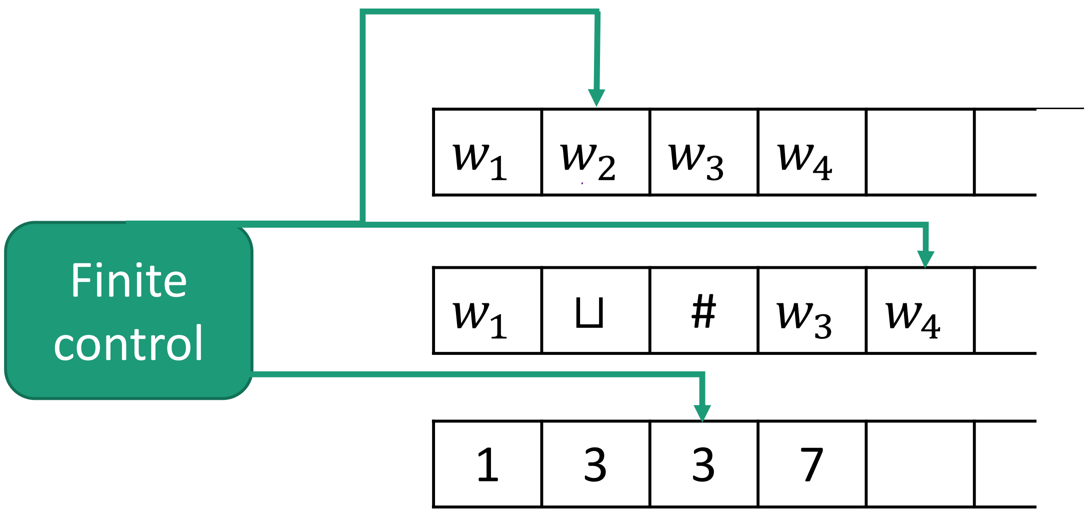
Deterministic vs. nondeterministic time



Deterministic 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



Input w to N (read-only)

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

Address in computation tree

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

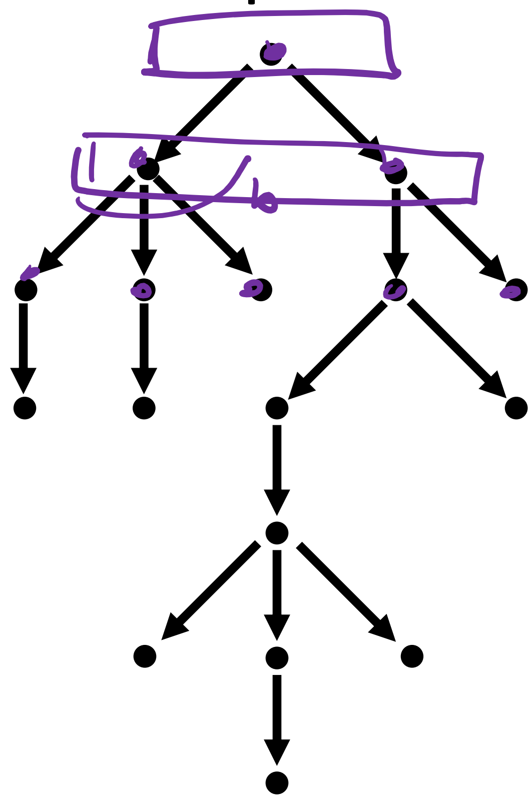
c) t^b

d) 2^t

$$\underbrace{1}_{\text{layer 1}} + \underbrace{b}_{\text{layer 2}} + \underbrace{b^2}_{\text{layer 3}} + \dots + \underbrace{b^{t-1}}_{\text{layer } t}$$

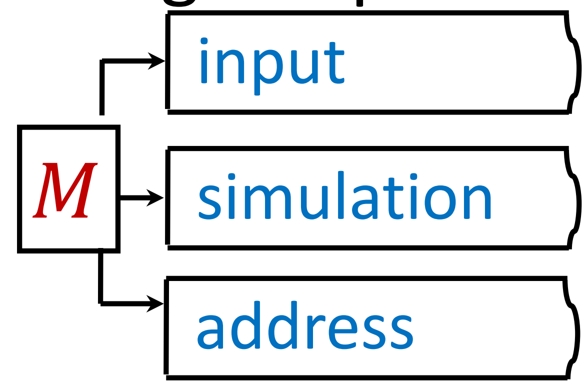
t

$$= \frac{b^t - 1}{b - 1} \le b^t$$



Deterministic vs. nondeterministic time

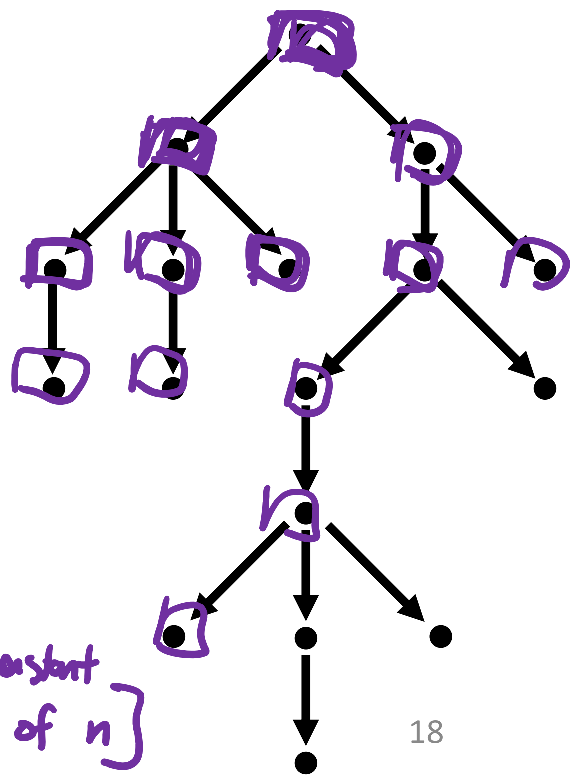
Theorem: Let $t(n) \geq n$ be a function. Every NTM running in time $t(n)$ has an equivalent deterministic single-tape TM running in time $2^{O(t(n))}$



Proof: Simulate NTM by 3-tape TM

b = max # of nondet. choices NTM makes in any steps

- # nodes: $\leq b^{t(n)}$



<u>Running time:</u>

To simulate one root-to-node path:

$\leq O(t(n))$ time

Total time: $b^{t(n)} \cdot O(t(n)) = 2^{t(n) \cdot \log b} \cdot 2^{\log t(n)}$

$= 2^{t(n) \cdot \log b + \log(t(n))}$

$= 2^{O(t(n))}$ {since b constant indep. of n}

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

$\mathrm{NTIME}(f(n))$ 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 $O(f(|w|))$ steps

   Runtime of NTM M is $O(f(n))$

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

M decides A

Complexity class NP

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

$$\mathrm{NP} = \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k) = \mathrm{NTIME}(n) \cup \mathrm{NTIME}(n^2) \cup \ldots$$

Which of the following are definitely true about NP?

a) $\mathrm{P} \subseteq \mathrm{NP}$

b) $\mathrm{NP} \subseteq \mathrm{P}$

c) $\mathrm{NP} \not\subseteq \mathrm{P}$

Win \$1 million

d) $\mathrm{NP} \subseteq \mathrm{EXP}$

$\mathrm{NTIME}(n^k) \subseteq \mathrm{TIME}(2^{O(n^k)})$

e) $\mathrm{EXP} \subseteq \mathrm{NP}$???

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

= { languages decidable in poly time on a deterministic TM }

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

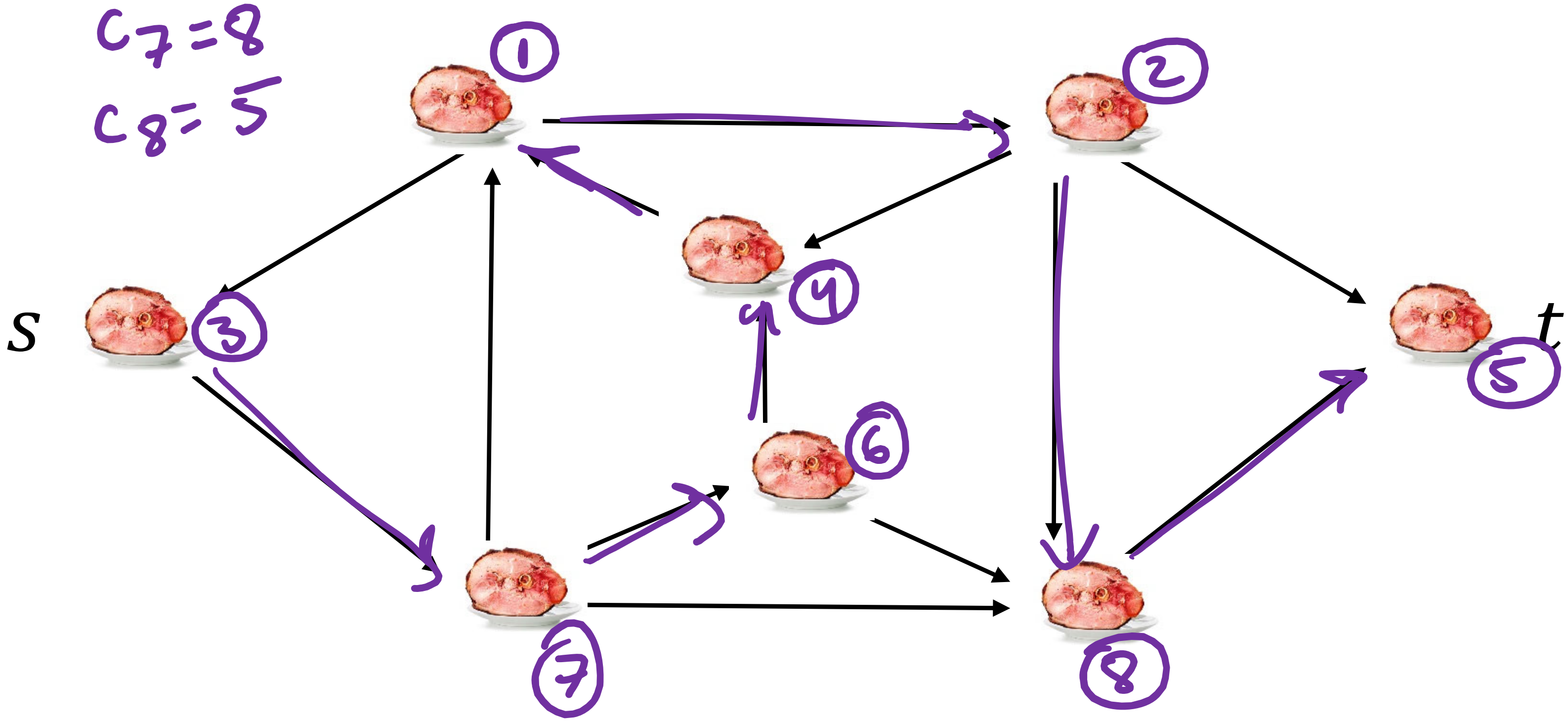
= { " exponential time }

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$\}$

Nondet.
Guess:

$c_1 = 3$
$c_2 = 7$
$c_3 = 6$
$c_4 = 4$
$c_5 = 1$
$c_6 = 2$
$c_7 = 8$
$c_8 = 5$



$HAMPATH \in \text{NP}$

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.