BU CS 332 – Theory of Computation

https://forms.gle/XcoaktUeaBdWrWLr5

Lecture 20:

- P Examples
- NP

Reading:

Sipser Ch 7.2-7.3

Final: Monday May 5
3-5 PM, CAS B12
(this room)

Mark Bun

April 14, 2025

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 | ∃ a TM deciding L in polynomial time }

= TIME(n) ∪ TIME(n^2) ∪ TIME(n^3) ∪ ...

- 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

Decision problems: a yes/NO question that we can ask about arbitrary length input strings

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?

ADD-VERIFICATION = $\{\langle x, y, z\rangle \mid x + y = z\}$

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? $\approx |V|^2$

… as an adjacency list? $\approx |V| + |E|$

> related by a polynomial factor
>
> $\Rightarrow$ poly-time under one representation stays poly-time under other

How long is the encoding of a natural number k

… in binary? $k = 7 \quad \langle k \rangle = 111$

$|\langle k \rangle| \approx \log_2 k$

… in decimal?

$|\langle k \rangle| \approx \log_{10} k = O(\log_2 k)$

… in unary?

$k = 7 \quad \langle k \rangle = 1111111$

$|\langle k \rangle| = k$ Exponentially worse encoding than binary/decimal

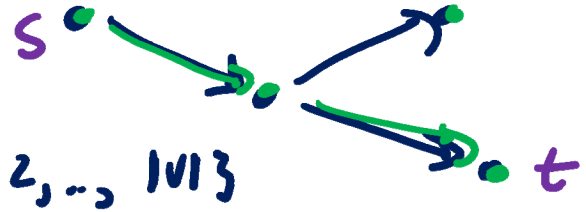
Describing and analyzing polynomial-time algorithms

- 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.

  $\Rightarrow$ 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 binary unless otherwise stated.)

Examples of languages in P



$s \in \{1, 2, \ldots, |V|\}$ $t \in \{1, 2, \ldots, |V|\}$

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

Idea: Breadth-first search

Assume G presented as adjacency matrix

1) 0 1 0 1 0 1 1
2) 1 0 1 1 1 0

$|\langle G, s, t\rangle| \approx \underbrace{|V|^2}_{G} + \underbrace{\log_2 |V|}_{s} + \underbrace{\log_2 |V|}_{t}$

$\approx |V|^2 = n$

$\Leftrightarrow |V| = n^{1/2}$

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

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

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

1 · 2 outer loop · 2 inner loop

Examples of languages in P

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

BFS on state diagram of $\langle D \rangle$ solves E_{DFA} in poly-time

Reduction to PATH:

On input $\langle D \rangle$: // description of DFA

1. Let G = state diagram of D
   Let s = start state of D
2. For each accept state t of D:
   - Determine whether $\langle G, s, t \rangle \in$ PATH
   - If so, reject
3. Accept.

Examples of languages in P

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

Euclidean algorithm

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

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 $PRIMES$

written in binary

On input $\langle x \rangle$:

NOT A POLY-TIME ALGORITHM

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: $\sqrt{x}$

$n = \log_2 x \iff x = 2^n$

divisions: $\sqrt{2^n} = 2^{n/2} = 2^{O(n)}$

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

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

$= \{ L \mid L$ decidable in exponential time $\}$

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

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)

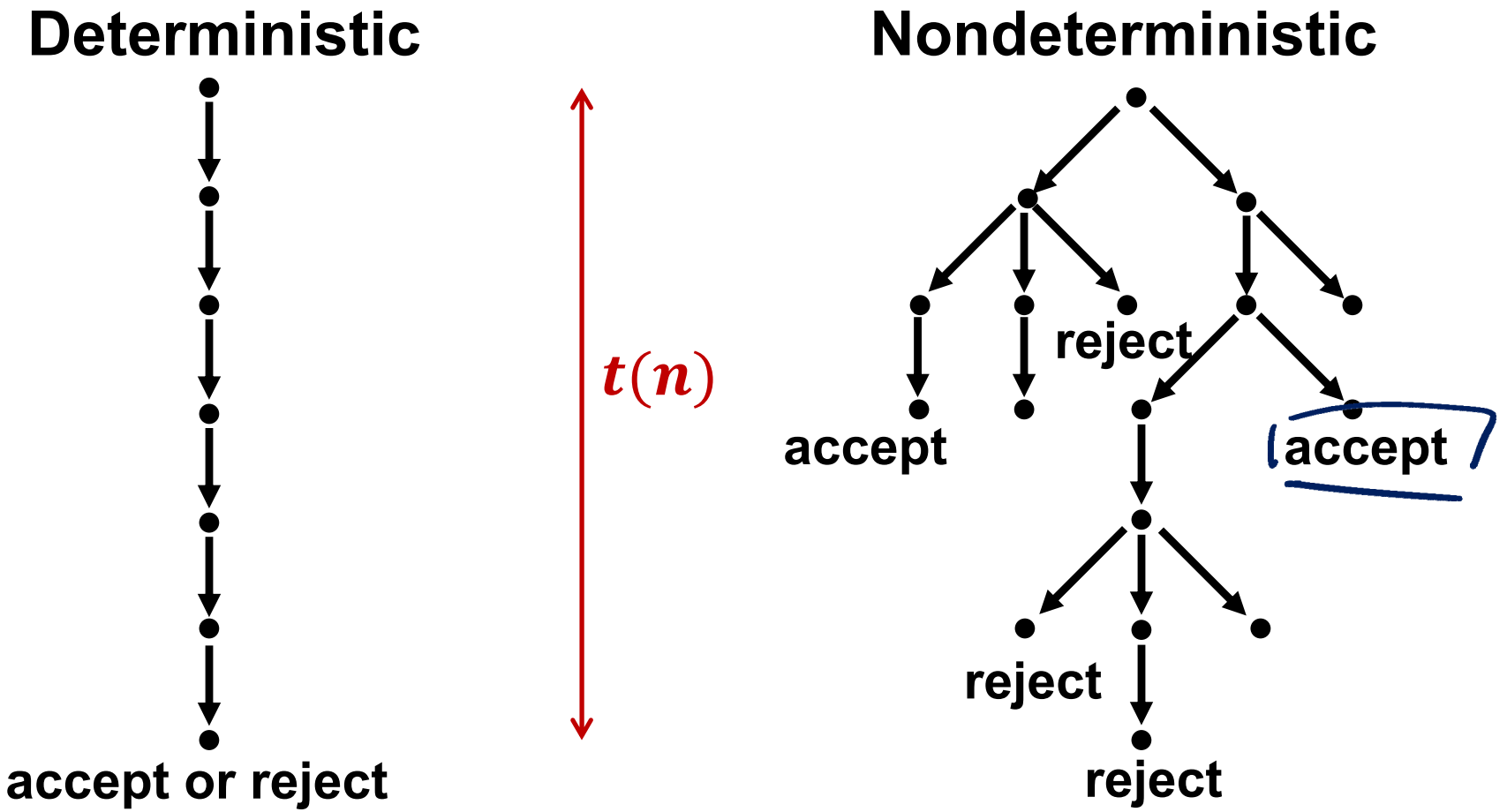
Nondeterministic time

Let t: $\mathbb{N} \rightarrow \mathbb{N}$ (input length → runtime)

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

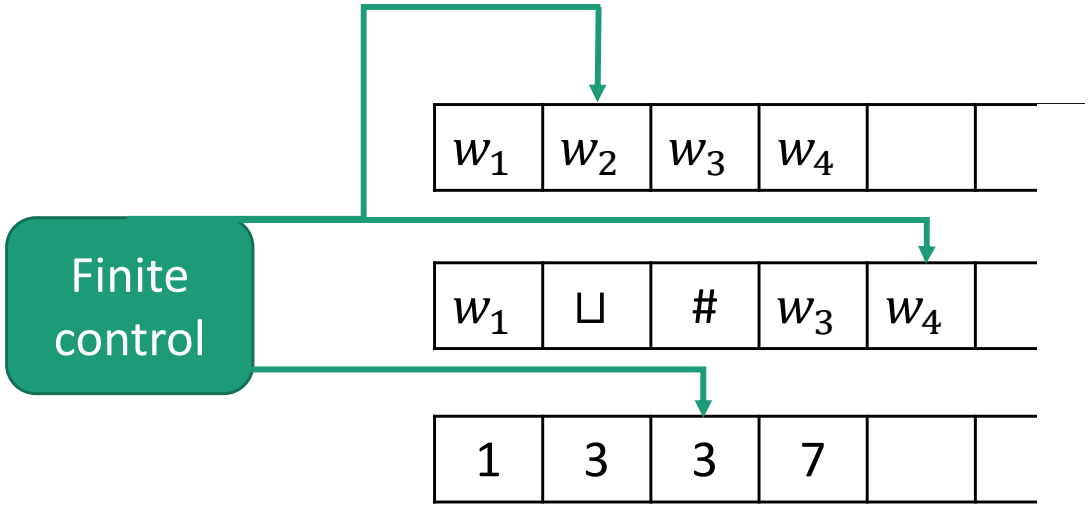
Deterministic vs. nondeterministic time



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

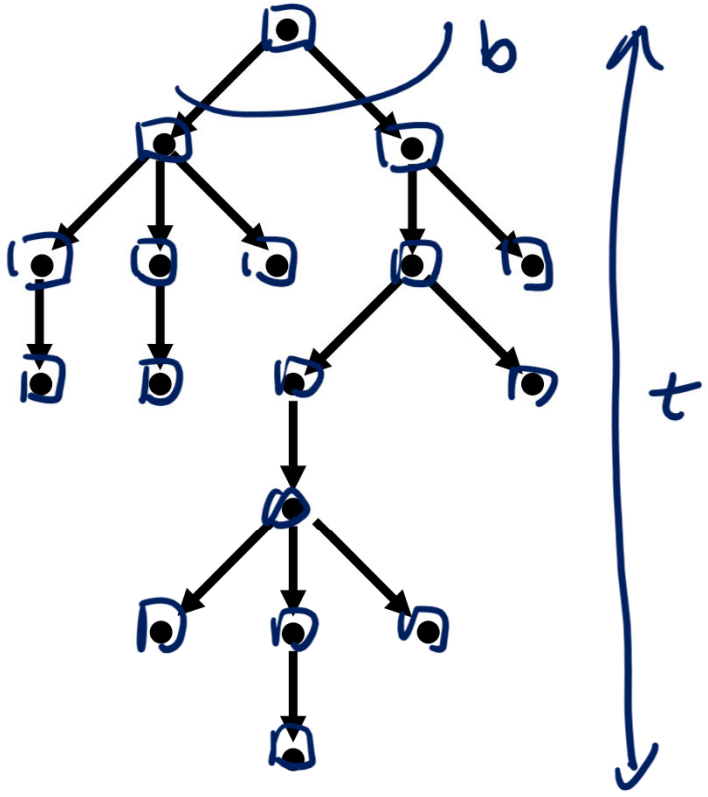
c) t^b

d) 2^t

$1 + b + b^2 + \ldots + b^{t-1}$

level 1, level 2, level t

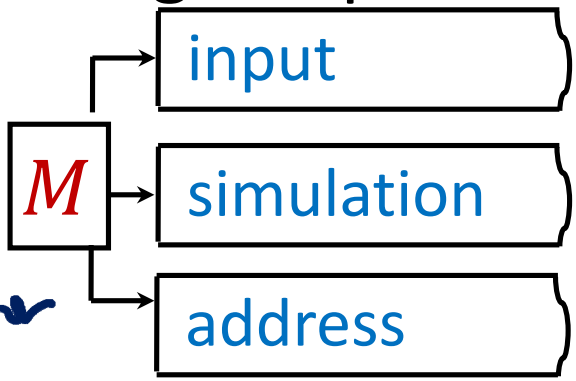
$= \frac{b^t - 1}{b - 1} \leq b^t$



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



- # nodes: $b^{t(n)}$ where b = NTM's branching factor

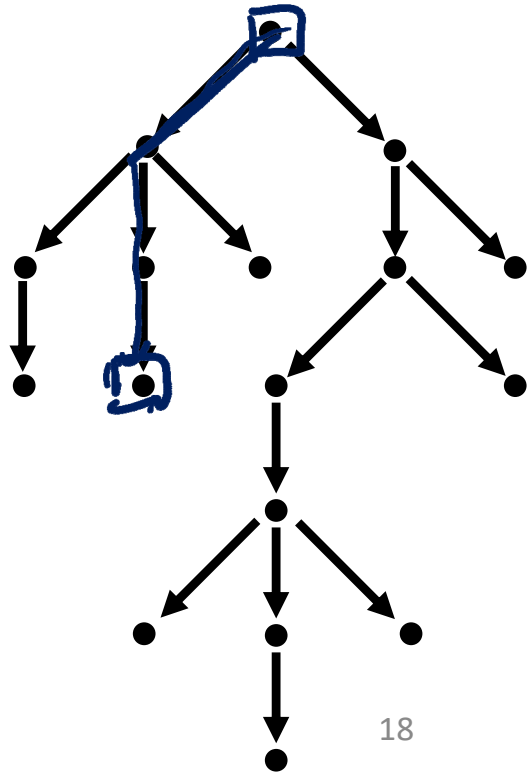
Running time:

To simulate one root-to-node path: $O(t(n))$

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

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

$= 2^{O(t(n)\cdot \log b)}$ ← constant $= 2^{O(t(n))}$



Deterministic vs. nondeterministic time

Theorem: Let $t(n) \geq n$ be a function. Every NTM running in time $t(n)$ has an equivalent 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 $f : \mathbb{N} \to \mathbb{N}$

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:

$= \{L \mid L$ decidable in the $O(f(n))$ on an NTM$\}$

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 $\mathrm{O}(f(|w|))$ 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

NTM M decides A

Complexity class NP

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

$$\mathrm{NP} = \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k) = \{ \text{languages } L \mid L \text{ decidable in poly-time on an NTM} \}$$

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

Which of the following are definitely true about NP?

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

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

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

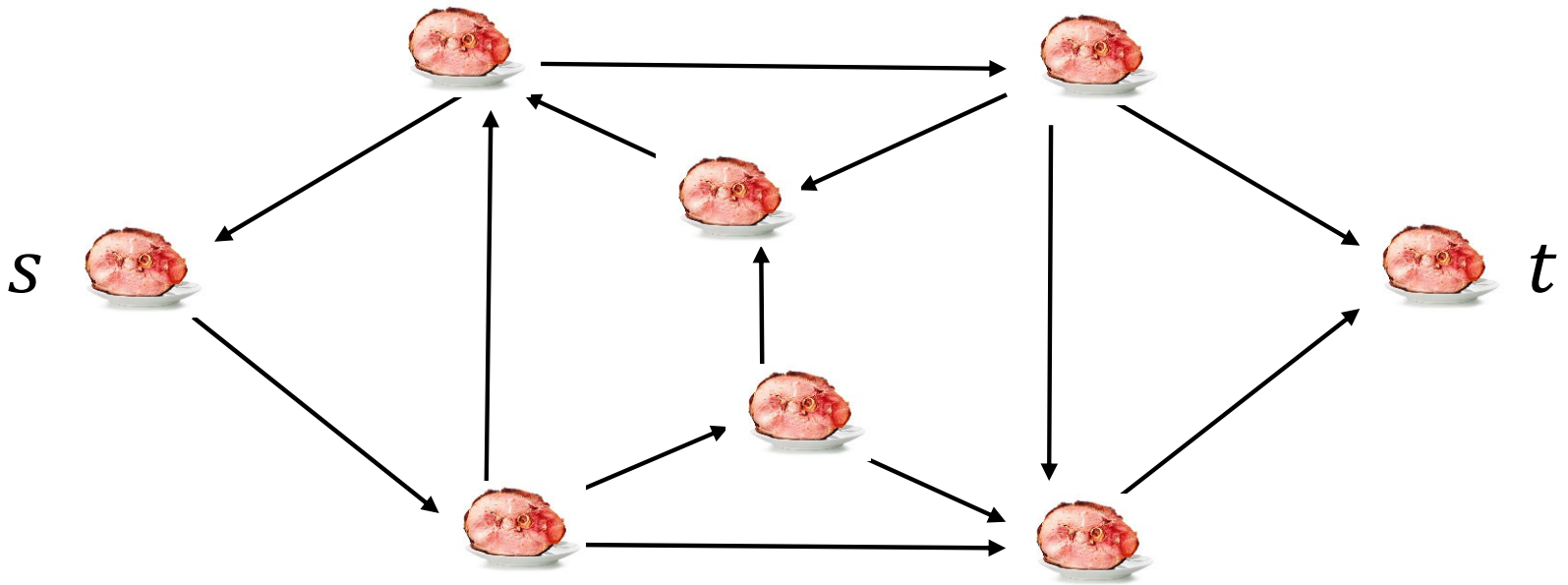
??? \$1 million prize

d) $\mathrm{NP} \subseteq \mathrm{EXP}$ $\quad \forall\, t(n)$ $\quad \mathrm{NTIME}(t(n)) \subseteq \mathrm{TIME}(2^{O(t(n))})$

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

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 algorithm decides $HAMPATH$ in **nondeterministic** polynomial time:

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.

Analyzing the algorithm

Need to check:

1) Correctness

2) Running time