CS 6041
Theory of Computation

Complexity

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Outline

- O() and o()
- Analyzing algorithm
- Time complexity
- Class P
- Class NP
- P vs. NP, NP-hard, NP-completeness

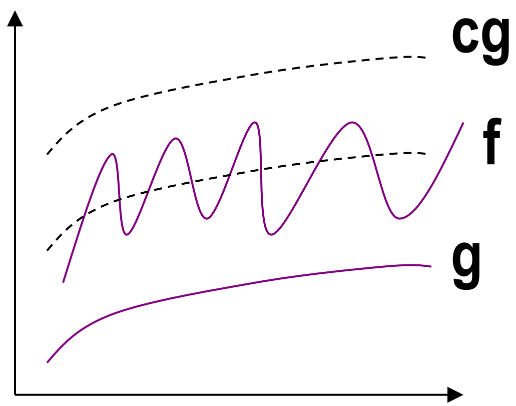
O(): Big-O notation

Suppose f and g are functions, f, g:$N \to R^+$.

If $\exists c, n_0, \forall n \geq n_0$, ***$f(n) \leq cg(n)$***

then ***$f(n)=O(g(n))$***

we say g is an ***upper bound*** for f().



Question

Let $f(n)=5n^3+2n^2+22n+6$, then

$f(n)=O(n^3)$?

True

Question

Let $f(n)=5n^3+2n^2+22n+6$, then

$f(n)=O(n^4)$?

True

Question

Let $f(n)=5n^3+2n^2+22n+6$, then

$f(n)=O(n^2)$?

False

Question

Let $f(n) = 3n\log_2 n + 5n\log_2(\log_2 n) + 2$, then

$f(n) = O(n\log n)$?

True

o(): small-O notation

- f(n)=o(g(n)) if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

- $\sqrt{n} = \mathrm{o}(\mathrm{n})$?

True $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sqrt{n}}{n} = \lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0$

Question

- n = o($nlog_2^{log_2^n}$)?

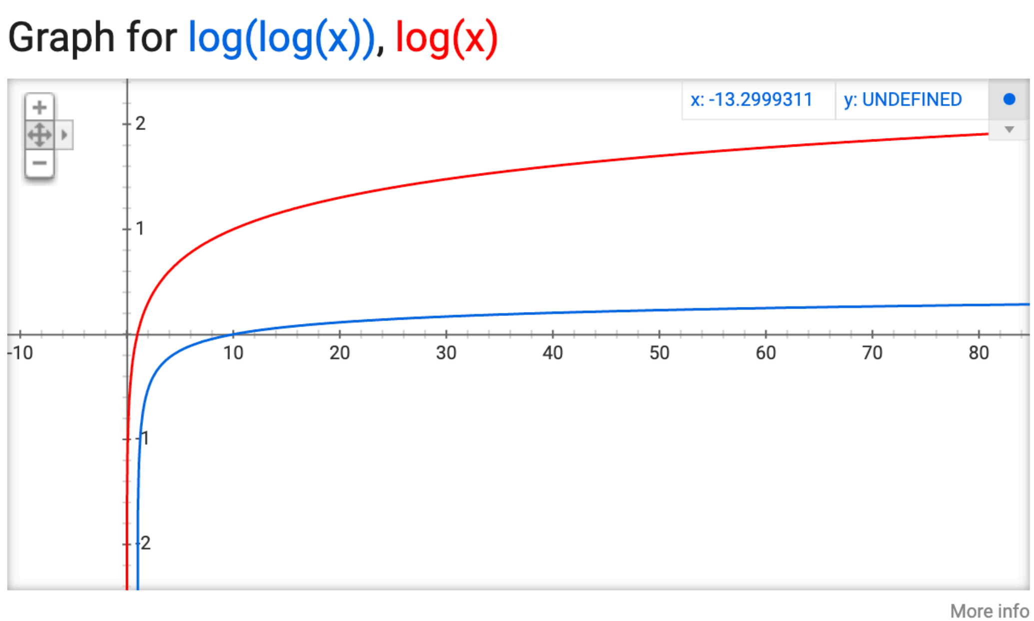
True

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{n}{nlog_2^{log_2^n}} = \lim_{n\to\infty}\frac{1}{log_2^{log_2^n}} = 0$$

Question

- $n log_2^{log_2^n}$ = o($n log_2^n$)?

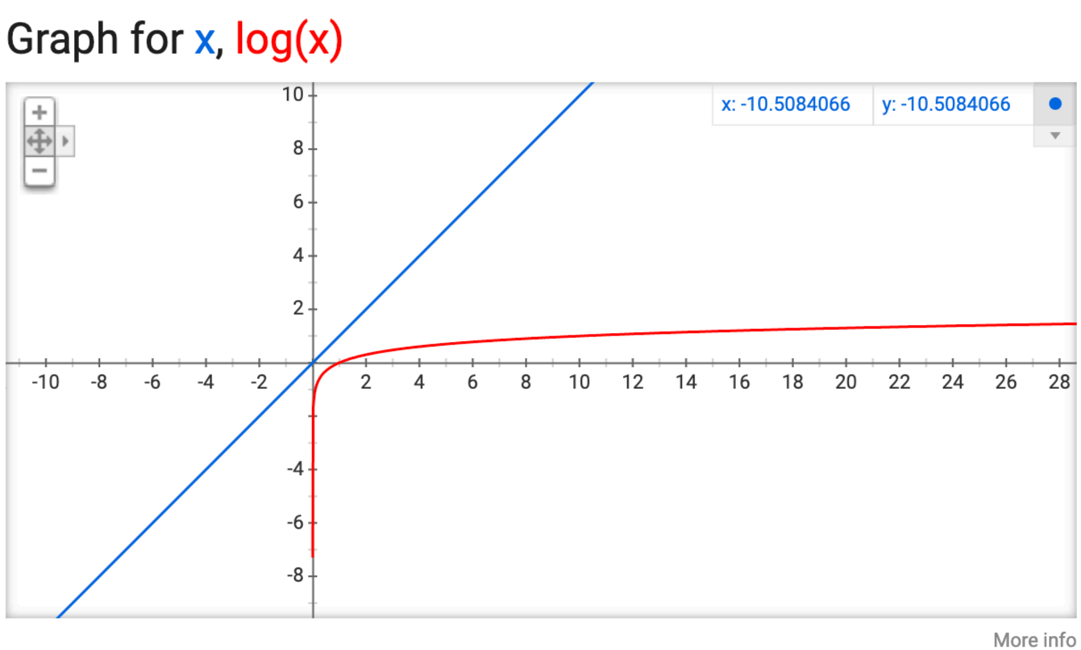
True $$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{n log_2^{log_2^n}}{n log_2^n} = \lim_{n\to\infty}\frac{log_2^{log_2^n}}{log_2^n} = 0$$



Question

- $n\log n = o(n^2)$?

True $$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{nlog_2^n}{n^2} = \lim_{n\to\infty}\frac{log_2^n}{n} = 0$$



Question

- $n^2 = o(n^3)$?

True

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{n^2}{n^3} = \lim_{n\to\infty}\frac{1}{n} = 0$$

Polynomial bounds vs. Exponential bounds

Polynomial bounds: $n^{O(1)}$

$n^{0.1}$, $n^{0.99}$, n, $n^{1.1}$, n^2, $n^{2.57}$, n^3, n^{10}, n^{100}, ……

Exponential bounds: $2^{n^{O(1)}}$

$2^{n^{0.1}}$, $2^{n^{0.5}}$, 2^{n^2}, $2^{n^{10}}$, $2^{n^{100}}$

Exponential functions grow much faster than polynomial functions

Worst case vs. Average case

- Worst case complexity

  $time_M(n)$=Maximum execution steps on TM M for input which length is n

- Average case complexity

  $time_M(n)$= Average execution steps on TM M for input which length is n

- Here we only discuss the worst case

Analyzing algorithms: $A = \{ 0^k1^k \mid k \geq 0 \}$

- Single tape DTM M_1: $O(n^2)$
- Single tape DTM M_2: $O(n\log n)$
- Double tape DTM M_3: $O(n)$

Analyzing algorithms: A = { 0^k1^k | $k \geq 0$ }

M_1="for input w:

1) scan the tape, if there exists 0 on the right of 1, reject;

O(n)

2) repeat as long as some 0s and some 1s remain on the tape:

3) scan across the tape and cross off one 0 and one 1;

(n/2)O(n)

4) if there exist 0s after all 1s are crossed off, reject; if there exist 1s after all 0s are crossed off, reject; if there are no 0s and 1s on tape, accept. "

Time: $O(n^2)$

O(n)

n/2 n/2

$\cancel{0}$	$\cancel{0}$	0	0	0	…	$\cancel{1}$	$\cancel{1}$	1	1	1	…

Analyzing algorithms: $A = \{ 0^k 1^k \mid k \geq 0 \}$

M_2="for input w:

1) scan the tape, if there exists 0 on the right of 1, reject;

$O(n)$

2) repeat as long as some 0s and some 1s remain on the tape:

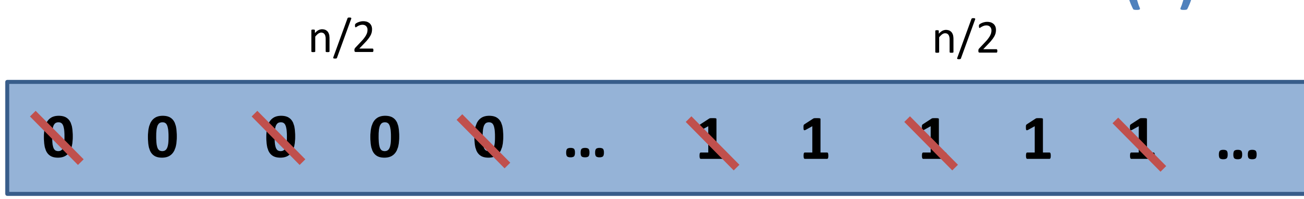
3) Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, reject;

4) Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.

$(1+\log_2 n)O(n)$

5) If no 0s and no 1s remain on the tape, accept. Otherwise, reject . "

$O(n)$



Time: $O(n\log n)$

Analyzing algorithms: $A = \{ 0^k 1^k \mid k \geq 0 \}$

M_3="for input w:

1) Scan across tape 1 and reject if a 0 is found to the right of a 1. **O(n)**

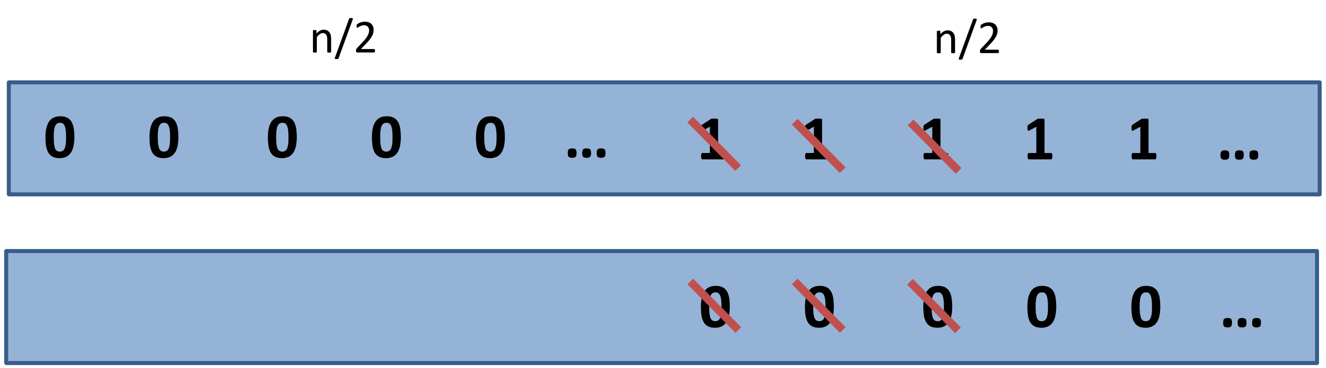
2) Scan across the 0s on tape 1 until the first 1. At the same time,

copy the 0s onto tape 2. **O(n)**

3) Scan across the 1s on tape 1 until the end of the input. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all the 1s are read, reject . **O(n)**

4. If all the 0s have now been crossed off, accept . If any 0s remain, reject ." **O(n)**

Time: O(n)



Analyzing algorithms: $A = \{ 0^k1^k \mid k \geq 0 \}$

- Single tape DTM M_1: $O(n^2)$
- Single tape DTM M_2: $O(n\log n)$
- Double tape DTM M_3: $O(n)$

- M_2 to M_1: a better algorithm
- M_3 to M_1/M_2: exchange more space for less time

Time complexity class

- TIME(t(n)) = { L | language L is decidable by an O(t(n)) time Turing machine.}

- A = { 0^k1^k | $k \geq 0$ }
  - **A∈TIME(*nlogn*)** → Single tape DTM M_2
  - **A∈TIME(n^2)** Single tape DTM M_1

Complexity relationships among models

- The choice of computational model can affect the time complexity of languages

- Single-tape TM vs. multitape TM
  - square relationship (n*n vs n)

- Deterministic TM vs. nondeterministic TM
  - exponential relationship (a^n vs n)

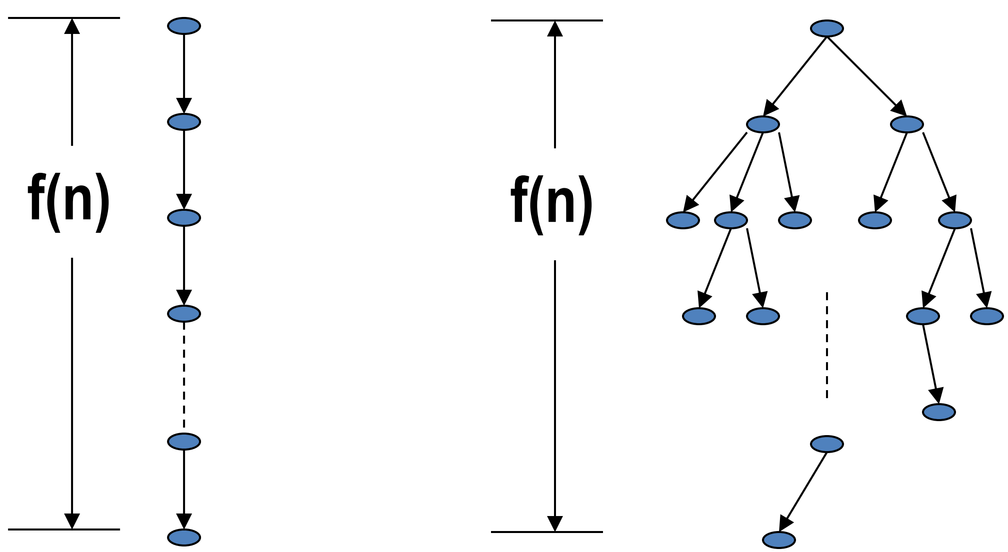
Complexity relationships among models

- Single-tape TM vs. multitape TM
  - Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape TM has an equivalent $O(t^2(n))$ time single-tape Turing machine.
  - Proof idea: simulating each step of the multitape machine uses at most $O(t(n))$ steps on the single-tape machine. Hence the total time used is $O(t^2(n))$ steps.

Complexity relationships among models

- Let N be a nondeterministic. The running time of N is the function f: N→N,

  where f(n) is the maximum number of steps that N uses on any branch of its computation on any input of length n.

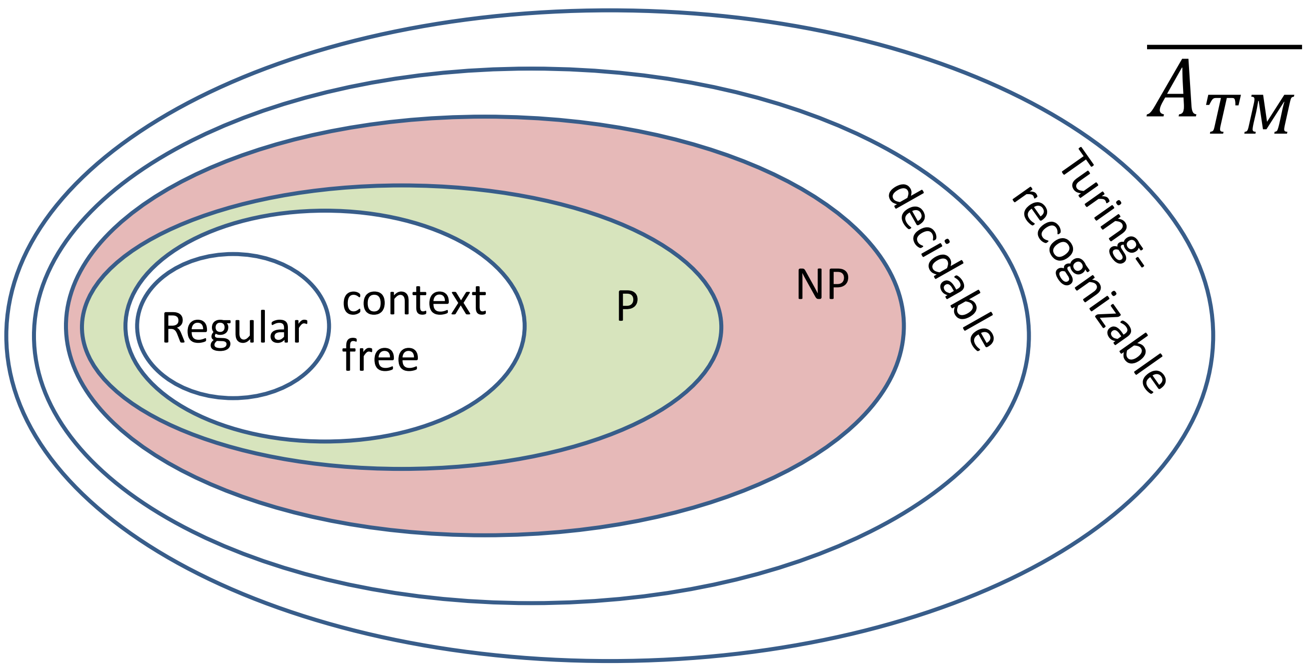


Complexity relationships among models

- Deterministic TM vs. nondeterministic TM
  - Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape TM has an equivalent $2^{O(t(n))}$ time deterministic single-tape TM.
  - Proof idea: the total number of nodes in the tree is less than twice the maximum number of leaves $O(2^{t(n)})$. The time it takes to start from the root and travel down to a node is $O(t(n))$. Therefore, the running time of D is $O(t(n)2^{t(n)}) = 2^{O(t(n)})$.

Time complexity class

- TIME(t(n)) = { L | language L is decidable by an O(t(n)) time Turing machine.}

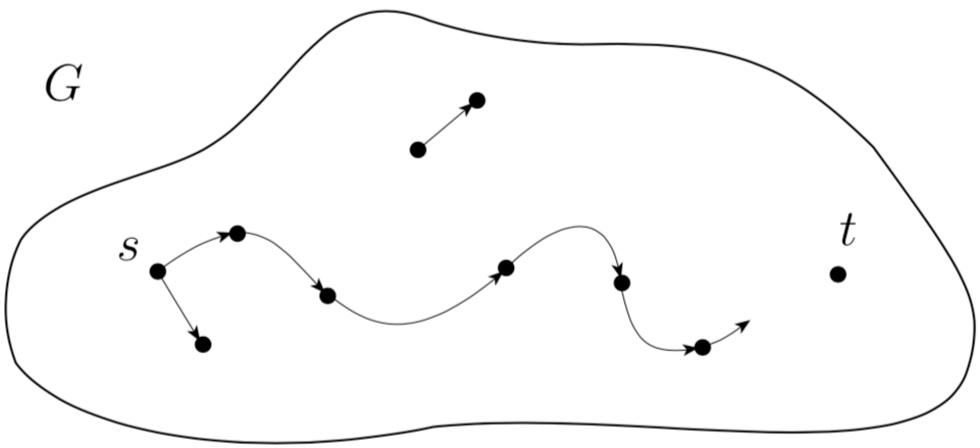


The class P

- P = $\bigcup_k$ TIME(n^k) = { L | L are languages that are <u>decidable in polynomial time</u> on <u>a single-tape DTM</u>}.
    - P is invariant for all models of computation that are polynomially equivalent to the single-tape DTM
    - P roughly corresponds to the class of problems that are realistically solvable on a computer, e.g., O(n), O(n^2),O(n^3).

The example of class P: the Path problem

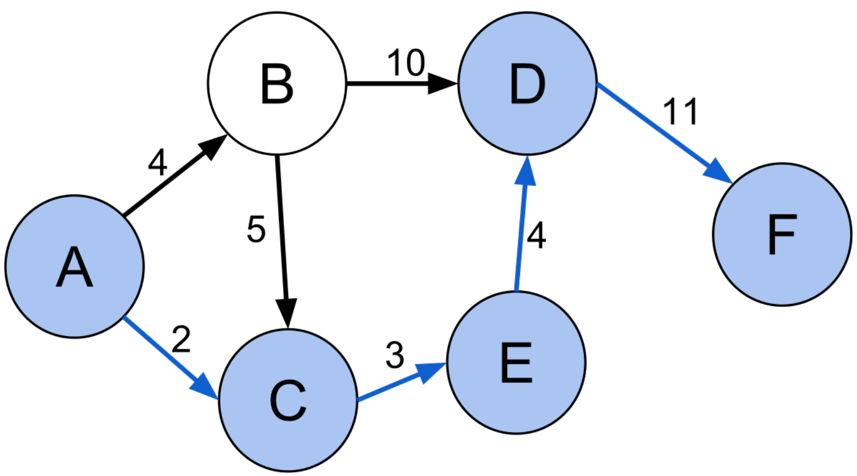
- Is there a path from s to t?



- *PATH* = {⟨G, s, t⟩| G is a directed graph that has a directed path from s to t}.

The example of class P: the Path problem

- Is there a path from s to t?
- *PATH* = {⟨G, s, t⟩| G is a directed graph that has a directed path from s to t}.



<A, F>? Yes

<D, C>? No

The example of class P: the Path problem

- Theorem: PATH $\in$ P

- Analysis:
  - Input: number of vertex: n
  - Brute-force search such potential paths: $O(n^n)$
  - Breadth-first search: $O(n)$

The example of class P: relatively prime

- Two numbers are ***relatively prime*** if 1 is the largest integer that evenly divides them both
  - 10 and 21 are relatively prime.

- *RELPRIME* is the problem of testing whether two numbers are relatively prime
  - $RELPRIME = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime}\}$.

The example of class P: relatively prime

- Theorem: RELPRIME $\in$ P

$$\underbrace{101011101}_{n}$$

- Analysis:
  - Input: length of number in binary: n
  - Brute-force: $O(2^n)$
  - Euclidean algorithm (greatest common divisor): $O(n^k)$

The example of class P: relatively prime

- Euclidean algorithm (greatest common divisor): $O(n^2)$

$$\underbrace{101011101}_{n}$$

https://www.youtube.com/watch?v=JUzYl1TYMcU

$E =$ "On input $\langle x, y\rangle$, where x and y are natural numbers in binary:

1. Repeat until $y = 0$:
2. Assign $x \leftarrow x \bmod y$.
3. Exchange x and y.
4. Output x."

every execution of stage 2 cuts the value of x by at least half

$\log_2(n)$

The class NP

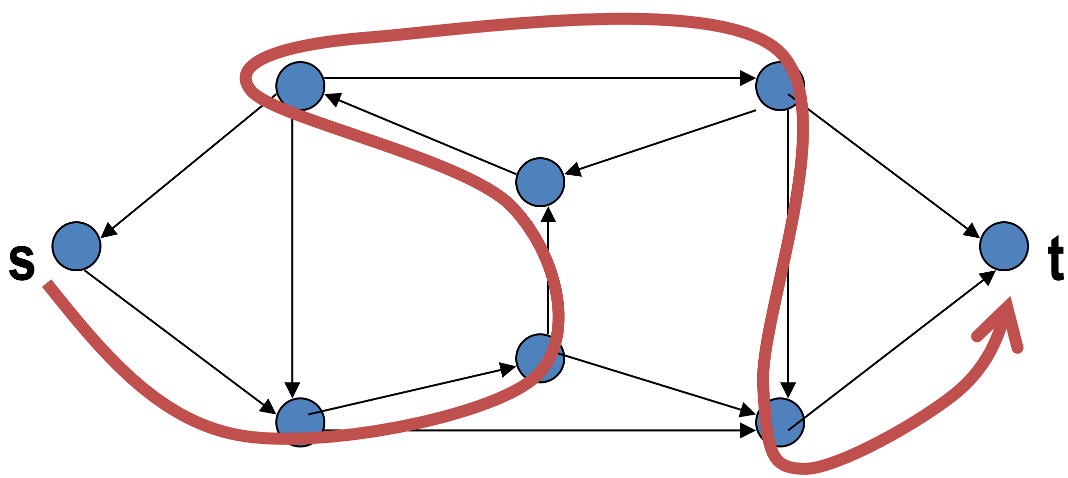
- $P = \bigcup_k TIME(n^k)$ = { L | L are languages that are decidable in polynomial time on a single-tape DTM}.

- NP = { We do not know languages L are decidable or not in polynomial time on a single-tape DTM}.

The example of class NP: Hamiltonian path

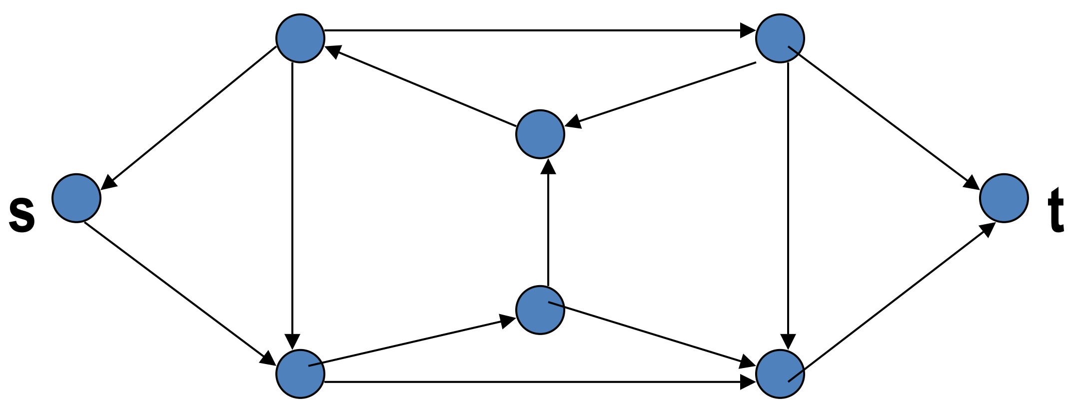
- Hamiltonian path: a directed path that goes through each node exactly once

- *HAMPATH* ={⟨G,s,t⟩|G is a directed graph with a Hamiltonian path from s to t}.



The example of class NP: Hamiltonian path

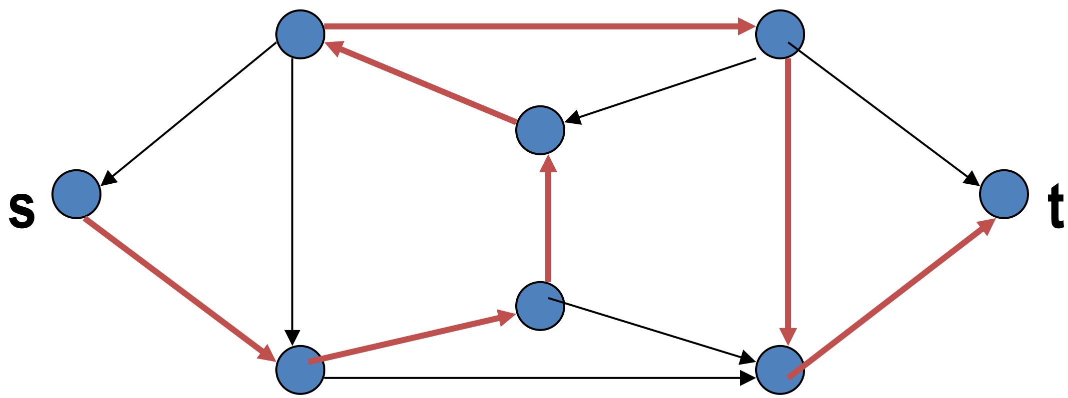
- *HAMPATH* $=\{\langle G,s,t\rangle \mid$ G is a directed graph with a Hamiltonian path from s to t$\}$.



No one knows whether HAMPATH is solvable in polynomial time.

Polynomial verifiability of Hamiltonian path

- *HAMPATH* $=\{\langle G,s,t\rangle \mid G$ is a directed graph with a Hamiltonian path from s to t$\}$.



We can ***verify*** *one HAMPATH in polynomial time.*

Verifying *something is much easier than* ***determining*** *something.*

The class NP

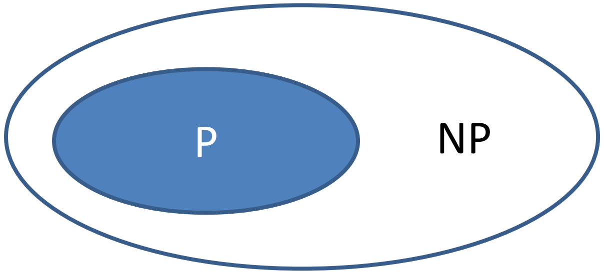
- P = $\bigcup_k \mathrm{TIME}(n^k)$ = { L | L are languages that are decidable in polynomial time on a single-tape DTM}.
- NP = { We do not know languages L are decidable or not in polynomial time on a single-tape DTM}.
- NP is the class of languages that have polynomial time verifiers

NTIME()

- $\text{NTIME}(t(n)) = \{ L \mid L$ is a language decided by an $O(t(n))$ time nondeterministic TM $\}$
- $\text{NP} = \bigcup_k \text{NTIME}(n^k)$
- A language is in NP iff it is decided by some nondeterministic polynomial time TM.

P vs. NP

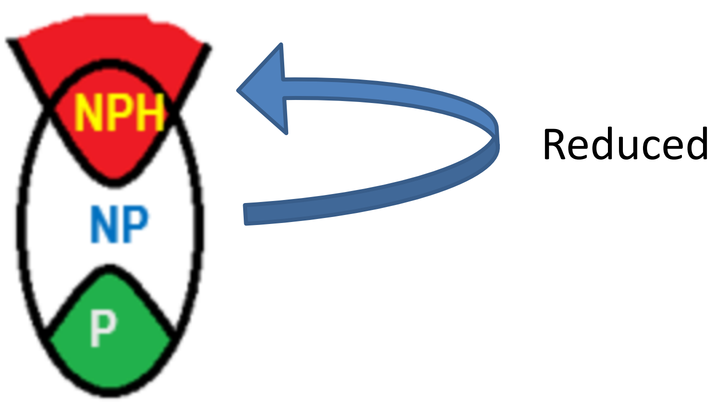
- P = the class of languages for which membership can be decided quickly
- NP = the class of languages for which membership can be verified quickly
- The question of whether P = NP is one of the greatest unsolved problems in computer science and mathematics



NP-hard

- A language B is ***NP-hard*** if:

  1. every A in NP is polynomial time reducible to B.



NP-completeness

- A language B is ***NP-complete*** if it satisfies two conditions:

  1. B is in NP, and

  2. every A in NP is polynomial time reducible to B.

Conclusion

- O() and o()
- Analyzing algorithm
- Time complexity
- Class P
- Class NP
- P vs. NP, NP-hard, NP-completeness