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

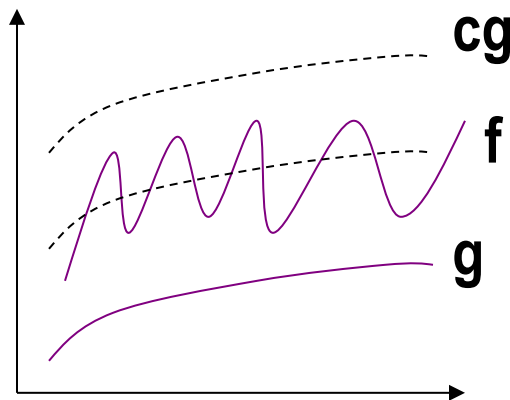- Class P

- Class NP

- P vs. NP, NP-hard, NP-completeness

O(): Big-O notation

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

If $\exists c, n_0, \forall n \geq n_0, \boldsymbol{f(n) \leq cg(n)}$

then $\boldsymbol{f(n) = O(g(n))}$

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(nlog n)?

True

o(): small-O notation

- f(n)=o(g(n)) if $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$

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

True $\qquad \lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = \lim\limits_{n\to\infty} \dfrac{\sqrt{n}}{n} = \lim\limits_{n\to\infty} \dfrac{1}{\sqrt{n}} = 0$

Question

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

True

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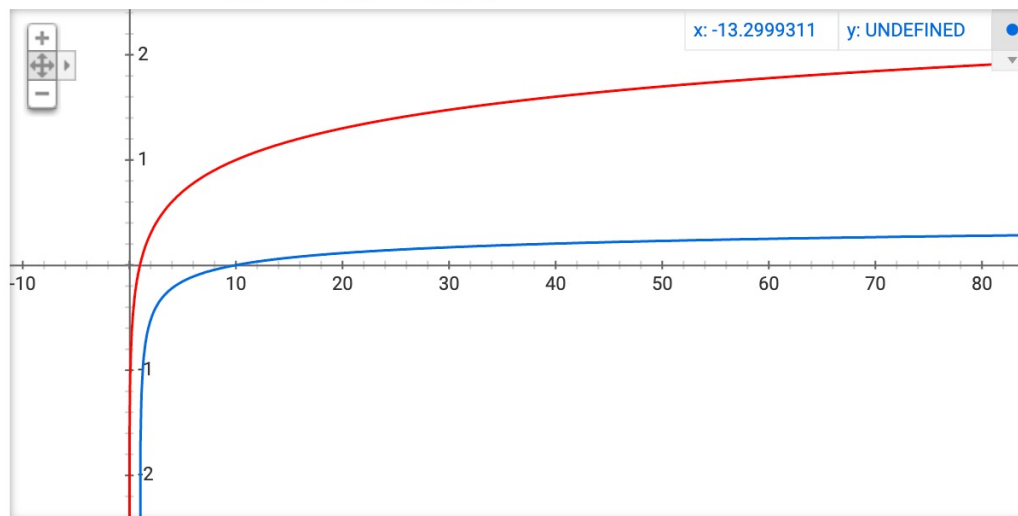
Question

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

True $\quad \lim_{n \to \infty} \dfrac{f(n)}{g(n)} = \lim_{n \to \infty} \dfrac{nlog_2^{log_2^n}}{nlog_2^n} = \lim_{n \to \infty} \dfrac{log_2^{log_2^n}}{log_2^n} = 0$
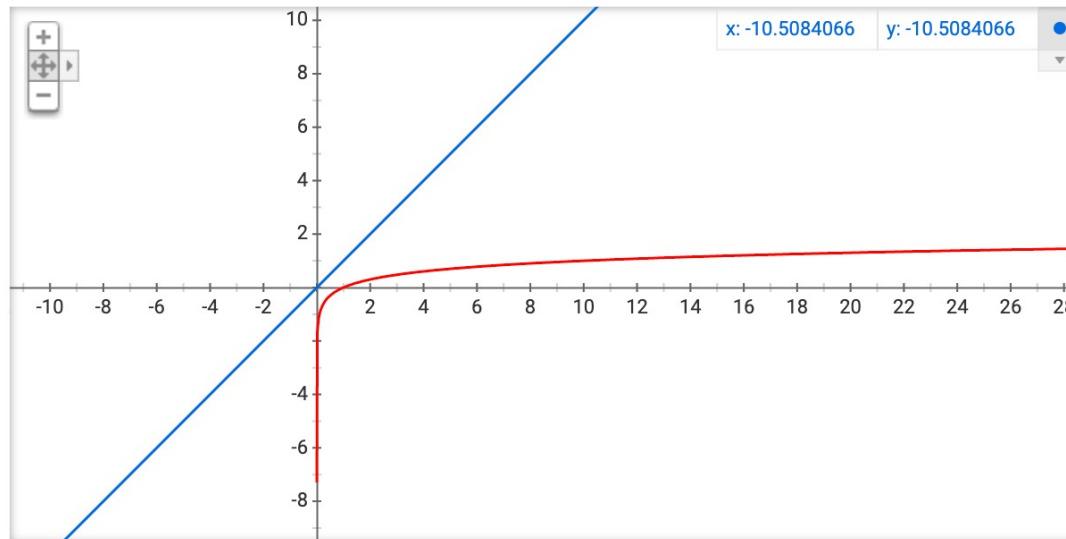
Graph for log(log(x)), log(x)

mputation

More info

Question

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

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

Graph for x, log(x)

More info

Question

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

True

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

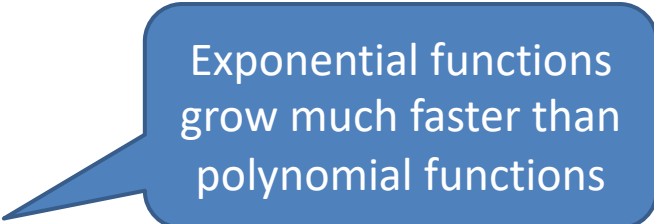# Polynomial bounds vs. Exponential bounds

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

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

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 | k≥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: $A = \{ 0^k 1^k \mid k \geq 0 \}$

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;

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. " **(n/2)O(n)**

Time: O(n²) **O(n)**

n/2 n/2

| 0̸ | 0̸ | 0 | 0 | 0 | … | 1̸ | 1̸ | 1 | 1 | 1 | … |

Analyzing algorithms: A = { 0^k1^k | k≥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)

n/2   n/2

Time: O(nlogn)

| 0̸ | 0 | 0̸ | 0 | 0̸ | ... | 1̸ | 1 | 1̸ | 1 | 1̸ | ... |

Analyzing algorithms: A = { 0^k1^k | k≥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,   **O(n)**
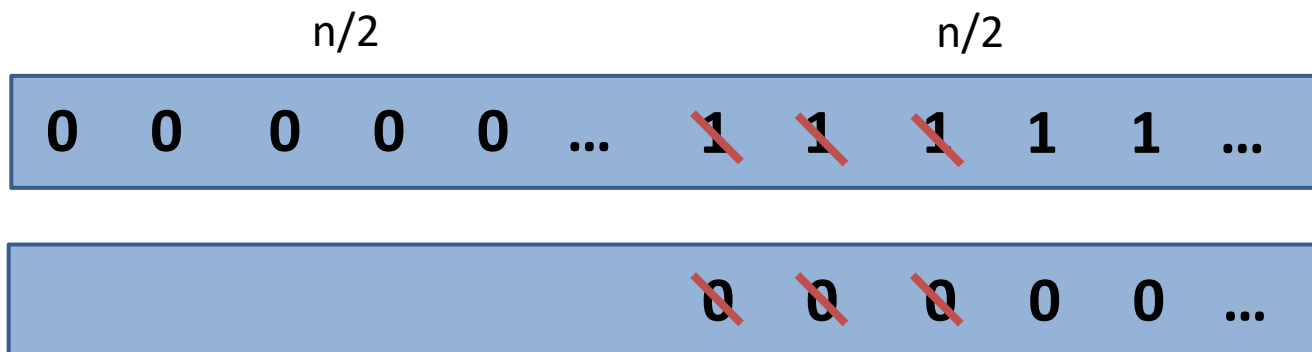
    copy the 0s onto tape 2.

    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 | k≥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

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

- 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.
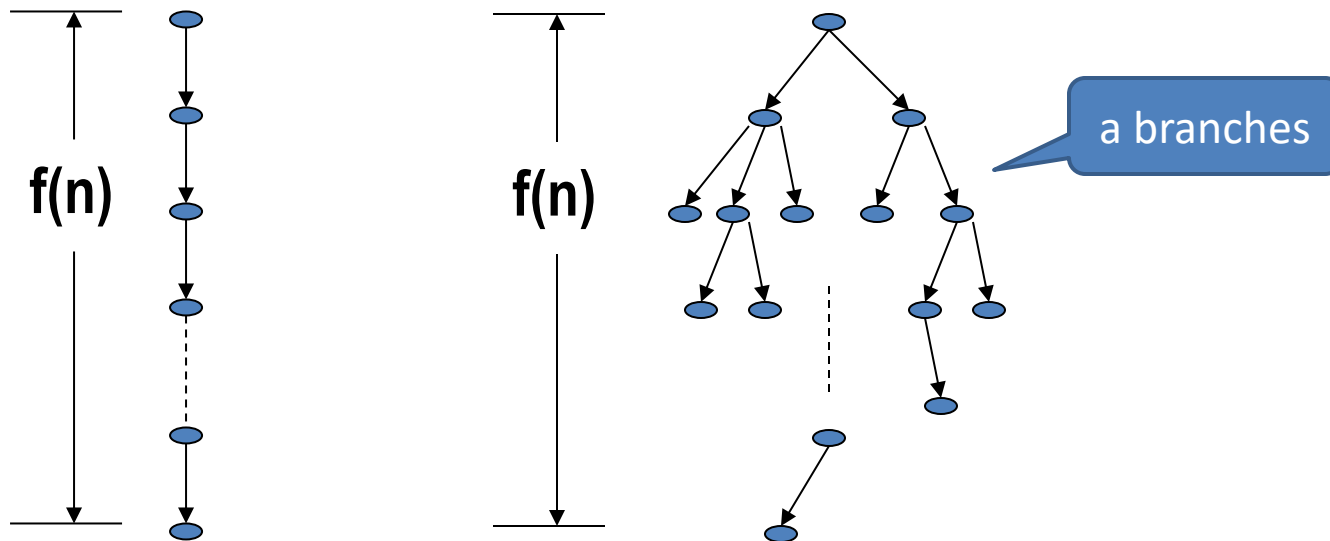
Time complexity class

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

$$\overline{A_{TM}}$$

Regular

context free

P

NP

decidable

Turing-recognizable
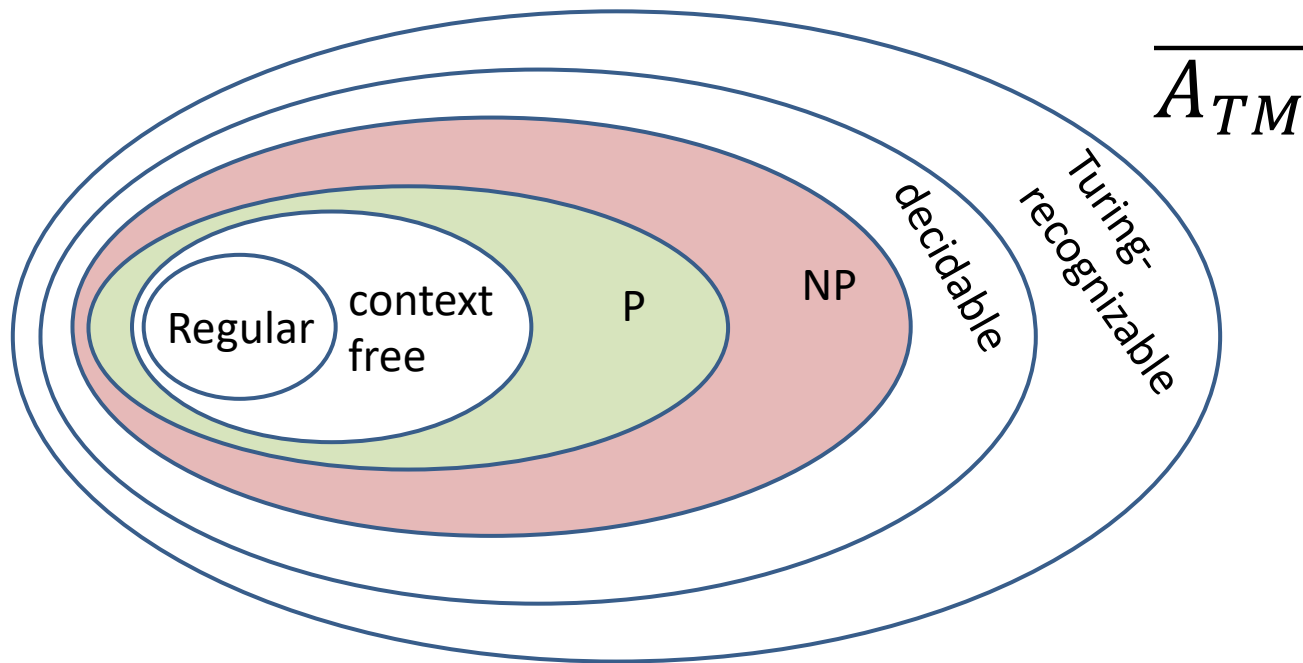
The class P

- 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>}.

  o P is invariant for all models of computation that are polynomially equivalent to the single-tape DTM

  o 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}.

<A, F>?   Yes
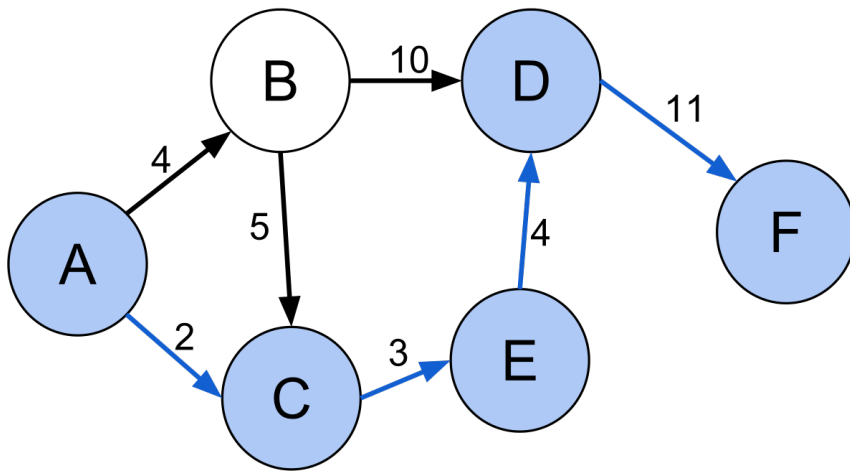
<D, C>?   No

The example of class P: the Path problem

- Theorem: PATH$\in$P

- Analysis:

   o Input: number of vertex: n

   o Brute-force search such potential paths: O(n^n)

   o Breadth-first search: O(n)
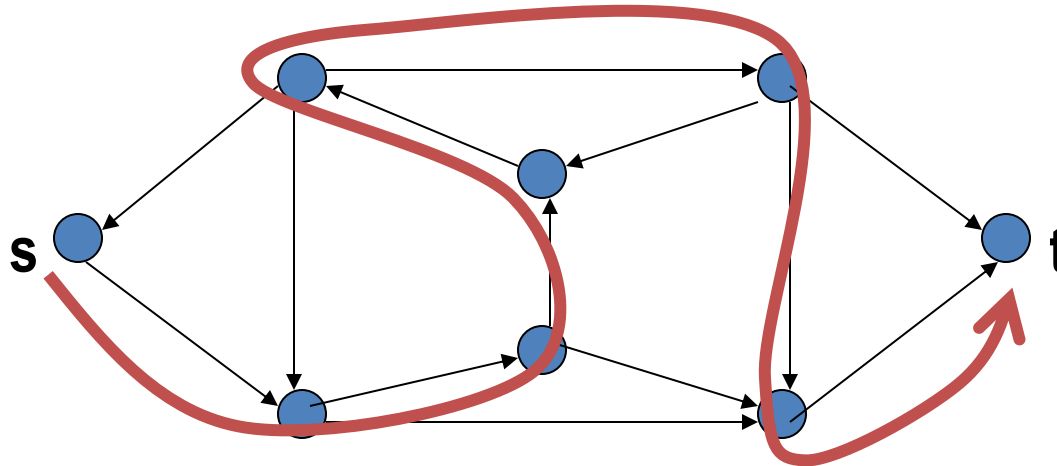
The class NP

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

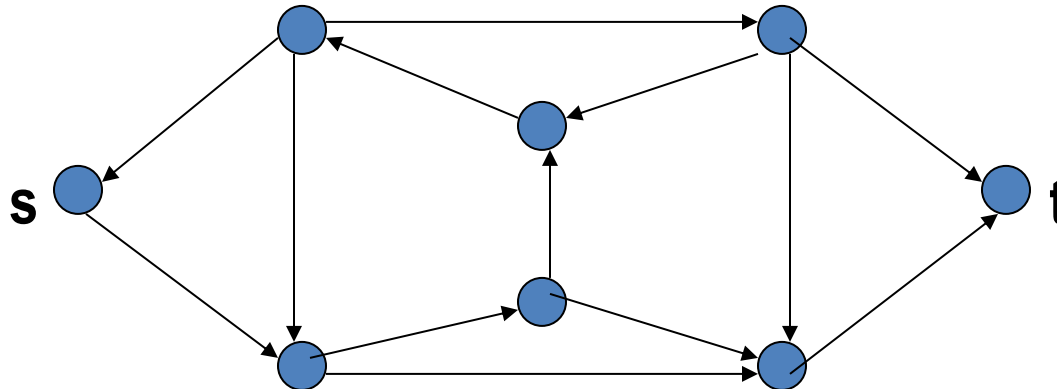- $NP = \{$ We do not know languages L are decidable or not in polynomial time on a single-tape DTM$\}$.

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* ={⟨G,s,t⟩|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* ={⟨G,s,t⟩|G is a directed graph with a Hamiltonian path from s to t}.
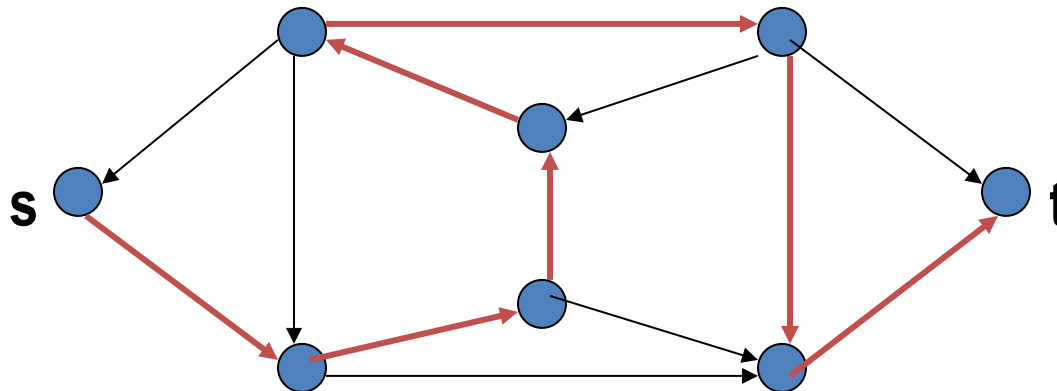
We can verify *one HAMPATH in polynomial time.*

Verifying something is much easier than determining *something.*

# The class NP

- P = $\bigcup_k$ 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

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
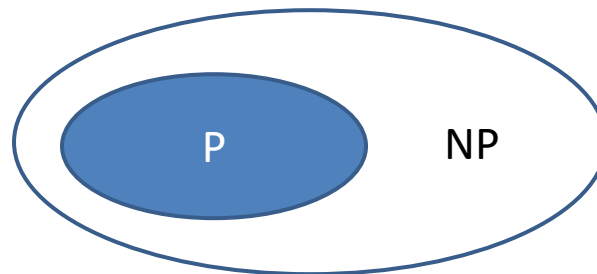
NP-hard

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

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

Reduced
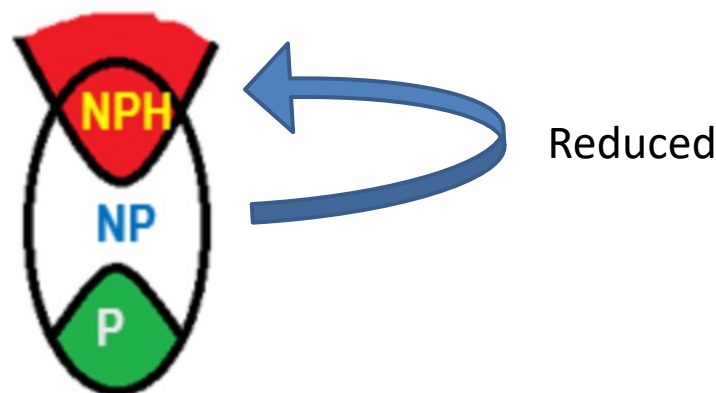
NP-completeness

- 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