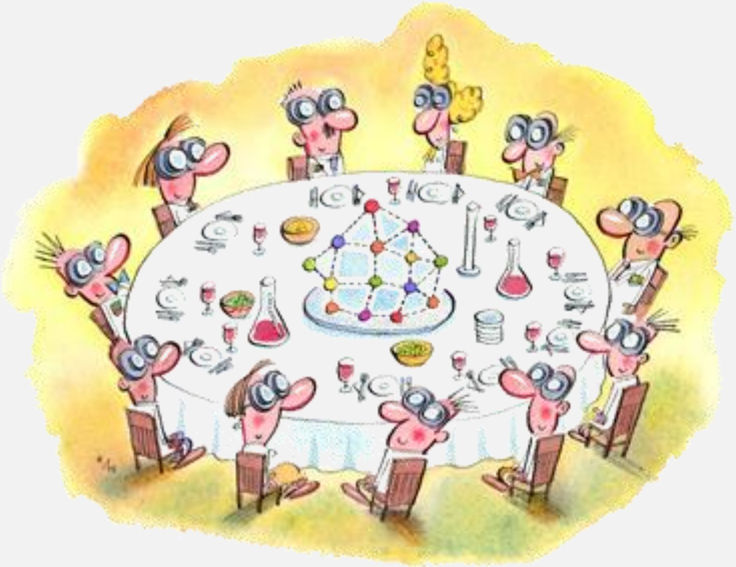
NP

Monday November 23, 2020



CREDIT: HAL MAYFORTH

HW questions?

Announcements

- HW10 released
    - Note extended due date: Sun Dec 6 11:59pm EST
- HW7 and 8 resubmissions due Mon Nov 30 11:59pm EST

Recap: The *PATH* Problem

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

- The **search** problem:
  - Exponential time (brute force) algorithm:
    - Check all possible paths and see if any connects s and t
  - Polynomial time algorithm:
    - Do a breadth-first search (roughly), marking “seen” nodes as we go

Verifying a *PATH*

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

- The **verification** problem:
  - Given some path p in G, check that it is a path from s to t
  - Let m = longest possible path = # edges in G
  - <u>Verifier</u> V = On input $<G, s, t, p>$, where p is some set of edges:
    1. Check some edge in p has "from" node s; mark and set it as "current" edge
       - <u>Max steps</u> = $O(\mathbf{m})$
    2. While there remains unmarked edges in p:
       a) Find the "next" edge in p, whose "from" node is the "to" node of "current" edge
       b) If found, then mark that edge and set it as "current", else reject
       - <u>Max steps</u> of each loop iteration $O(\mathbf{m})$
       - <u>Loop iterates</u> at most $\mathbf{m}$ times; total looping time = $O(\mathbf{m}^2)$
    3. Check "current" edge has "to" node t; if yes accept, else reject
  - Total time = $O(\mathbf{m}) + O(\mathbf{m}^2) = O(\mathbf{m}^2)$ = polynomial in m

PATH can be verified in polynomial time

Verifiers, Formally

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

DEFINITION 7.18

A ***verifier*** for a language A is an algorithm V, where

$$A = \{w |\ V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}.$$

certificate, or ***proof***

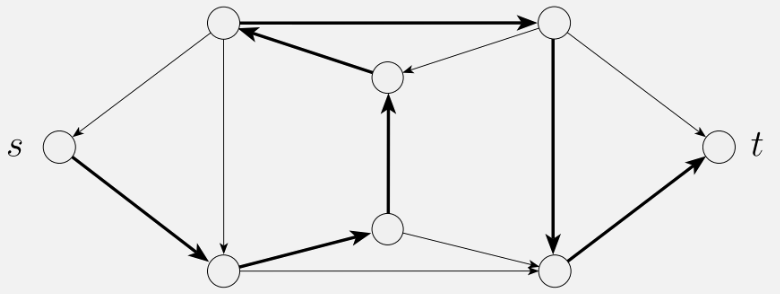
We measure the time of a verifier only in terms of the length of w, so a ***polynomial time verifier*** runs in polynomial time in the length of w. A language A is ***polynomially verifiable*** if it has a polynomial time verifier.

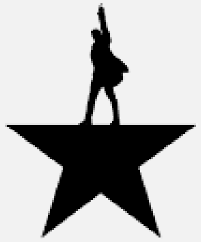
- NOTE: a cert c must be at most length n^k, where n = length of w
  - Why?
- *PATH* is polynomially verifiable

The *HAMPATH* Problem

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

- A Hamiltonian path goes through every node in the graph





- The **Search** problem:
  - Exponential time (brute force) algorithm:
    - Check all possible paths and see if any connect s and t using all nodes
  - Polynomial time algorithm:
    - We don't know if there is one!!!
- The **Verification** problem:
  - Still $O(\mathbf{m^2})$!
  - *HAMPATH* is polynomially verifiable, but <u>not</u> polynomially decidable

The class **NP**

DEFINITION 7.19

NP is the class of languages that have polynomial time $\boxed{\text{verifiers}}$.

- *PATH* is in **NP**, and **P**
- *HAMPATH* is in **NP**, but <u>not</u> **P**

NP = Nondeterministic polynomial time

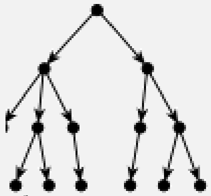
DEFINITION 7.19

NP is the class of languages that have polynomial time verifiers.

THEOREM 7.20

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

- **=>** If a lang L is in NP, then it has a poly time verifier V
- Create NTM deciding L: on input w =
  - Nondeterministically run V with w and all possible certs c
- **<=** If L has NTM decider N,
  - then let the cert denote one accepting path in N
  - Then create poly time verifier that runs N for only that path



P vs NP

DEFINITION 7.7

Let $t\colon \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. Define the ***time complexity class***, $\textbf{TIME}(\boldsymbol{t(n)})$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

DEFINITION 7.12

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k).$$

DEFINITION 7.21

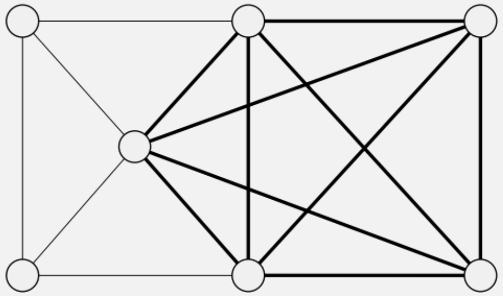
$$\textbf{NTIME}(\boldsymbol{t(n)}) = \{L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine}\}.$$

COROLLARY 7.22

$$\mathrm{NP} = \bigcup_k \mathrm{NTIME}(n^k).$$

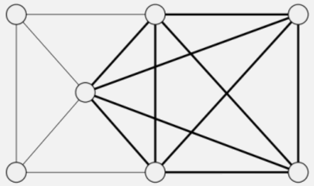
More NP Problems

- $CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$
  - A clique is a subgraph where every two nodes are connected
  - A k-clique contains k nodes



- $$\begin{aligned} SUBSET\text{-}SUM = \{\langle S, t\rangle \mid\ & S = \{x_1, \ldots, x_k\}, \text{ and for some} \\ & \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\} \end{aligned}$$
  - Some subset of a set of numbers sums to some total
  - e.g., $\langle\{4, 11, 16, 21, 27\}, 25\rangle \in SUBSET\text{-}SUM$

Theorem: *CLIQUE* is in NP



$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

PROOF IDEA The clique is the certificate.

PROOF The following is a verifier V for *CLIQUE*.

$V =$ "On input $\langle\langle G, k\rangle, c\rangle$:

1. Test whether c is a subgraph with k nodes in G. $O(\mathbf{k})$
2. Test whether G contains all edges connecting nodes in c. $O(\mathbf{k}^2)$
3. If both pass, *accept*; otherwise, *reject*."

DEFINITION 7.18

A ***verifier*** for a language A is an algorithm V, where

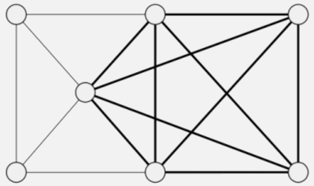
$$A = \{w \mid V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}.$$

We measure the time of a verifier only in terms of the length of w, so a ***polynomial time verifier*** runs in polynomial time in the length of w. A language A is ***polynomially verifiable*** if it has a polynomial time verifier.

DEFINITION 7.19

NP is the class of languages that have polynomial time verifiers.

Proof 2: *CLIQUE* is in NP



$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

$N =$ "On input $\langle G, k\rangle$, where G is a graph:

1. Nondeterministically select a subset c of k nodes of G.

2. Test whether G contains all edges connecting nodes in c.

3. If yes, *accept*; otherwise, *reject*."

"try all subgraphs"

$O(\mathbf{k^2})$

THEOREM 7.20

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

Theorem: *SUBSET-SUM* is in NP

$$SUBSET\text{-}SUM = \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\}$$

PROOF IDEA The subset is the certificate.

PROOF The following is a verifier V for *SUBSET-SUM*.

V = "On input $\langle\langle S, t\rangle, c\rangle$:

1. Test whether c is a collection of numbers that sum to t.
2. Test whether S contains all the numbers in c.
3. If both pass, *accept*; otherwise, *reject*."

ALTERNATIVE PROOF We can also prove this theorem by giving a nondeterministic polynomial time Turing machine for *SUBSET-SUM* as follows.

N = "On input $\langle S, t\rangle$:

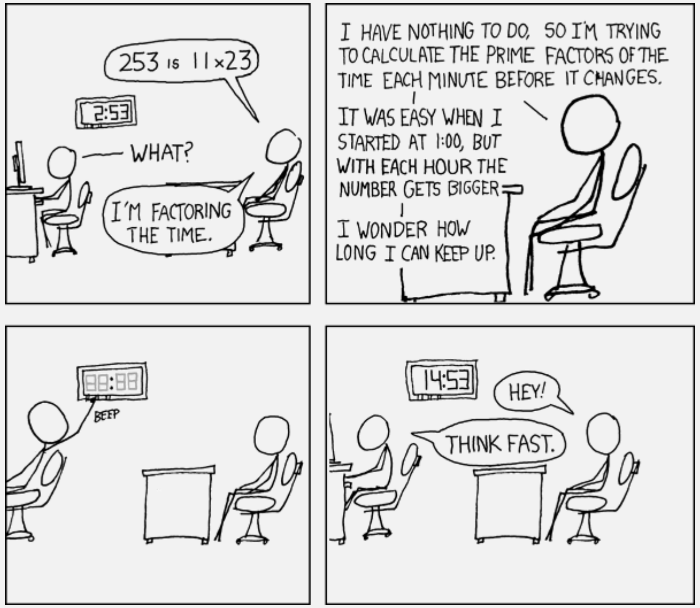
1. Nondeterministically select a subset c of the numbers in S.
2. Test whether c is a collection of numbers that sum to t.
3. If the test passes, *accept*; otherwise, *reject*."

$$COMPOSITES = \{x \mid x = pq, \text{ for integers } p, q > 1\}$$

- A composite number is <u>not</u> prime

- *COMPOSITES* is polynomially verifiable

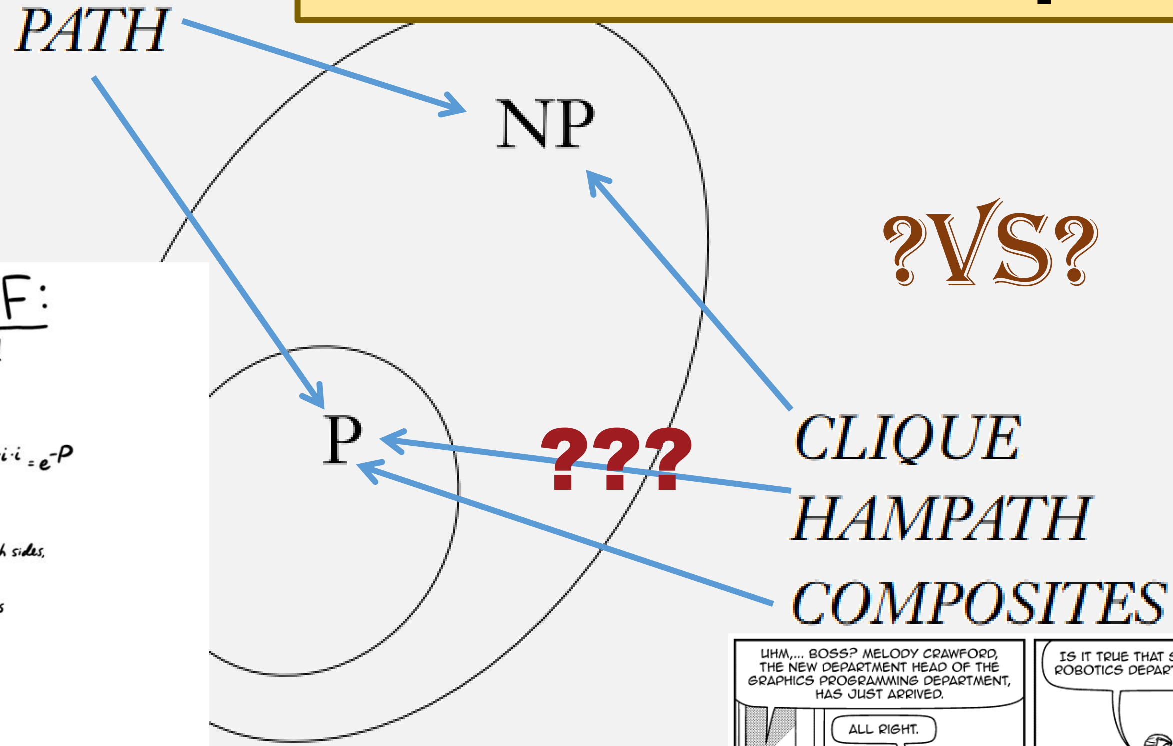
- A certificate could be:
  - Some factor that is not 1



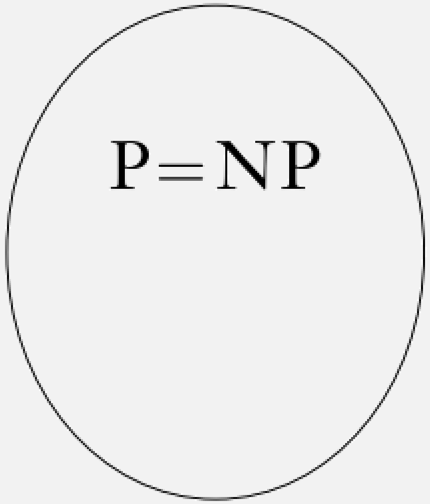
- Checking existence of factors (or not, i.e., testing primality) …
  - … is also poly time
  - But only discovered recently (2002)

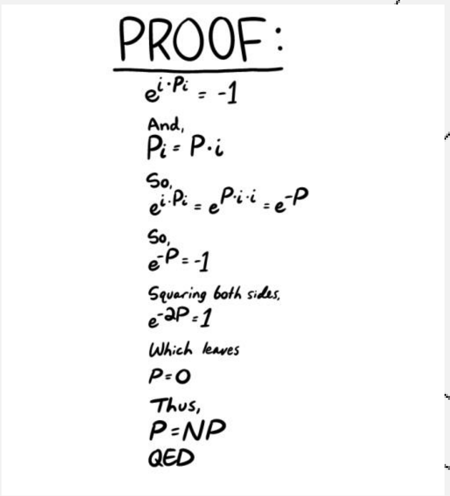
Does P = NP?

One of the greatest unsolved mysteries in math and computer science

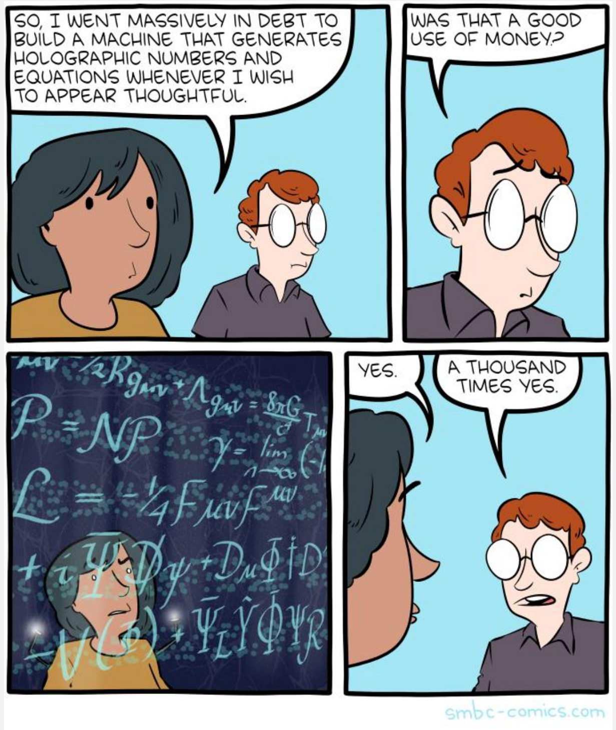


?VS?





It's hard to prove that something doesn't exist



Sandra and Woo by Oliver Knörzer (writer) and Powree (artist) – www.sandraandwoo.com

Check-in Quiz 11/25

On gradescope

End of Class Survey 11/25

See course website