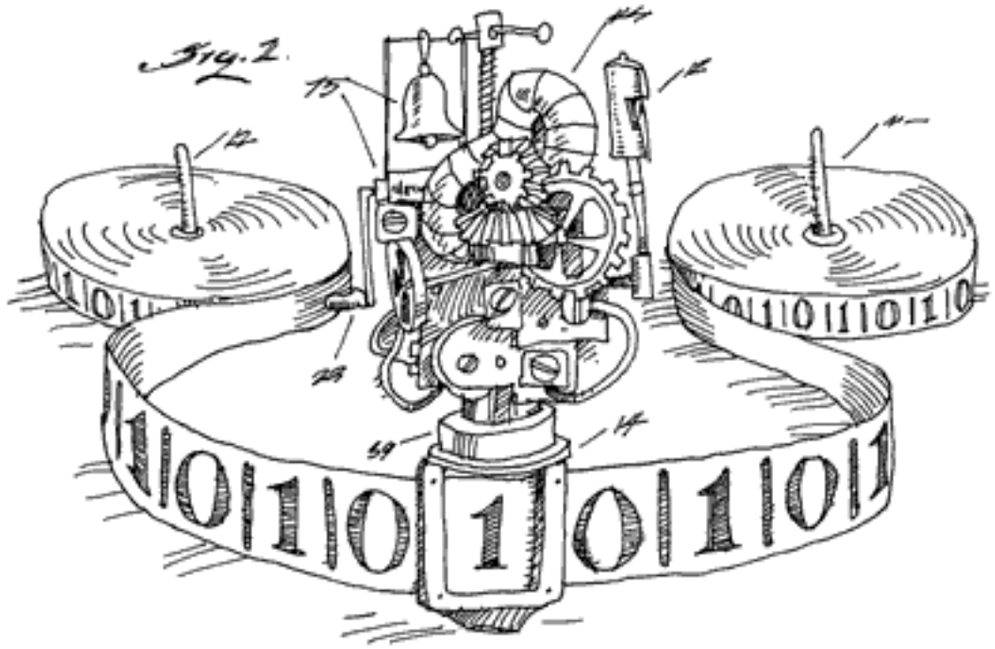
CSC4010: Introduction to Theoretical Computer Science



Lecture 12

How large is P?

= set of all languages that are solvable in polynomial time.

- Languages in P:
  - PALINDROMES, REACHABILITY…



u, s, t, is more a path from s to t.

- But last time: $\text{TIME}(n) \subsetneq \text{TIME}(n^2) \subsetneq \text{TIME}(n^3) \subsetneq \cdots$
  - And similarly: $P \subsetneq \text{TIME}(2^n)$

$\text{EXP} = \bigcup_{k \geq 1} \text{TIME}(2^{n^k})$ $\quad P \subsetneq \text{EXP}$.

- How large is P? What is the boundary of class P? What are the class of problems which are almost in P but for which we do not have a polynomial time algorithm?

↳ NP = Non-deterministic polynomial time -

CLIQUE

The trivial algorithm: $\binom{k}{2} \cdot n^k$. ⇒ iterating through all subsets.

$|S| \geq k$. $S \subseteq V$ (the set of vertices) s.t. $\forall u, v \in S$ $\exists$ an edge b/w u, v.

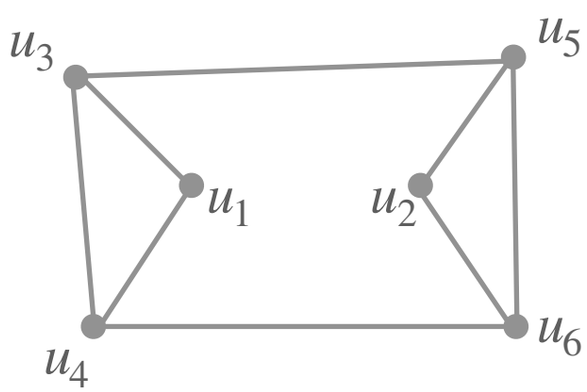
$k \approx \frac{n}{2}$ then time complexity for this problem is $2^{n/2}$.

$$\text{CLIQUE} = \{\langle G, k\rangle \mid G \text{ has a } k\text{-clique}\}.$$

I give you a graph G and a number k. You claim that $\langle G, k\rangle \in$ CLIQUE.

How would you convince me of this fact?

"If someone gives me a proof the fact that $\langle G, k\rangle \in$ CLIQUE, then I can verify in time k^2".



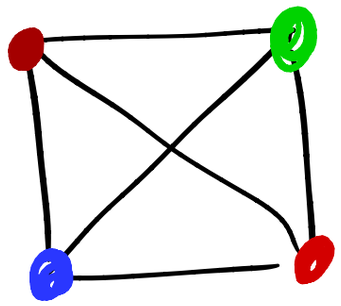
Graph G

	u_1	u_2	u_3	u_4	u_5	u_6
u_1	0	0	1	1	0	0
u_2	0	0	0	0	1	1
u_3	1	0	0	1	1	0
u_4	1	0	1	0	0	0
u_5	0	1	1	0	0	1
u_6	0	1	0	0	1	0

Natural Problems with Short Certificates

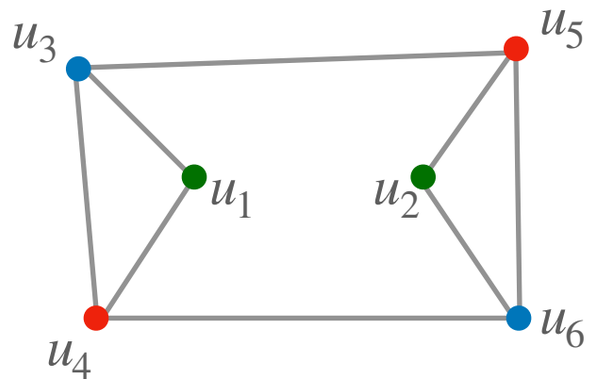
- 3-COL = $\{\langle G \rangle \mid G$ can be 3 colored$\}$.

→ assign colors to the vertices of the graph from a palette of size 3.
s.t. every pair of adjacent vertices have distinct colors.

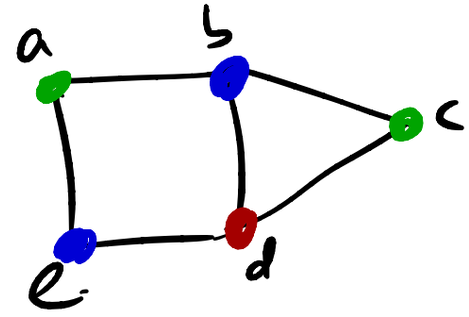


Then at least 2 vertices get the same color.

(not 3-colorable)



Graph G



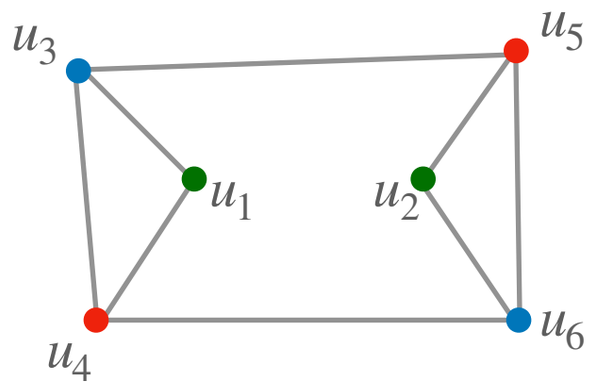
→ A proof of $\langle G \rangle \in$ 3-COL is just a valid 3-coloring of G.

→ If $\langle G \rangle \notin$ 3-COL then is no valid 3-coloring.

This certificate is easy to check: → check edges and make sure the endpts are distinctly colored.
→ $O(n^2)$.

Natural Problems with Short Certificates

- 3-COL $= \{\langle G \rangle \mid G \text{ can be 3 colored}\}$.



Graph G

- SUBSET-SUM $= \{\langle a_1, a_2, \cdots, a_n, T \rangle \mid a_1, \cdots, a_n, T \in \mathbb{N}, \exists I \subset \{1, \cdots, n\} \sum_{i \in I} a_i = T\}$

↳ target T.

Is there a subset of $\{a_1, \ldots, a_n\}$ s.t. that these numbers sum to T.

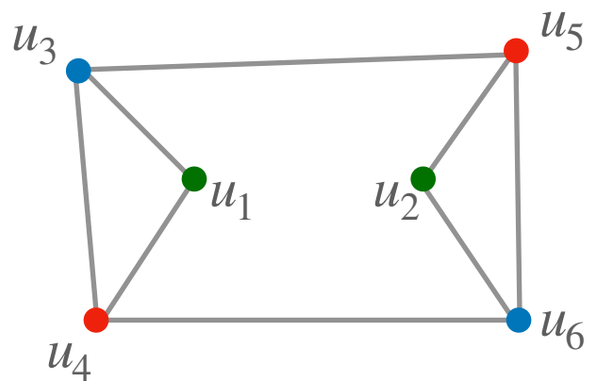
Trivial: Check all subsets $I \subseteq n$.

2^n time.

Natural Problems with Short Certificates

CLIQUE, 3-COL, SUBSET-SUM $\in$ NP.

- 3-COL $= \{\langle G \rangle \mid G \text{ can be 3 colored}\}$.



Graph G

If $\langle a_1, a_2, \ldots, a_n, T \rangle \in$ SUBSET-SUM, then $\exists I$.

- SUBSET-SUM $= \{\langle a_1, a_2, \cdots, a_n, T \rangle \mid a_1, \cdots, a_n, T \in \mathbb{N}, \exists I \subset \{1, \cdots, k\} \sum_{i \in I} a_i = T\}$

Suppose $\langle a_1, a_2, \ldots, a_n, T \rangle \in$ SUBSET-SUM, then a certificate would be a valid I s.t. $\sum_{i \in I} a_i = T$.

Verified in time $O(n)$ if the certificate is valid.

NP: A game between a Prover and a Verifier

given a language L, me prover and verifie

decide:
1) Format of the certificate
2) what will the verifier do w/ the input + certificate.

Verifier



poly-time
skeptical

Prover



omniscient
unreliable

⇒ unlimited computation power.

Is it makes mistakes.

Suppose we have a language L, $\} x \in L$?

Given a string $x \in \{0,1\}^*$

Prover wants to convince **Verifier** that $x \in L$.

Prover cooks up a "certificate" or a "proof string" u and sends it to the **Verifier**.

Verifier, in polynomial time, should be able to tell if the proof is legit.

in polynomial time, check if u is a legit proof for $x \in L$.

NP: A game between a Prover and a Verifier

Verifier



poly-time
skeptical

L = CLIQUE.

Then,

1) The format of the certificate: a list of n-vertices that form a clique.

2) checks if every pair in the list has an edge in the given graph.

Prover



omniscient
unreliable

$x \notin L$.

Then it is possible that there is no small proof of this fact

Given a string $x \in \{0,1\}^*$

Prover wants to convince **Verifier** that $x \in L$.

Prover cooks up a "certificate" or a "proof string" u and sends it to the **Verifier**.

Verifier, in polynomial time, should be able to tell if the proof is legit.

NP: A game between a Prover and a Verifier

SUBSET-SUM:
1) certificate: $I \subseteq [n]$
2) Checking if $\sum_{i \in I} a_i = T$.

Verifier



poly-time
skeptical

"sound": because for any valid instance, $\exists I$!

"complete": because for any invalid instance, there is no such I.

Prover



omniscient
unreliable

The certificate and information should satisfy the following properties:

"Completeness"

If $x \in L$ there must be some proof u that convinces the **Verifier.**

"Soundness"

If $x \notin L$ no matter what "proof" **Prover** gives, the **Verifier** should detect the lie.

Verifier: As a Turing Machine

Definition: A verifier is a Turing machine that in addition to its usual tape also has a special certificate tape. The input to the verifier is a pair (x, c) with x, the input string written on the usual tape, and c, the certificate written on the certificate tape.

Definition: The language $L(V)$ recognised by the verifier V is

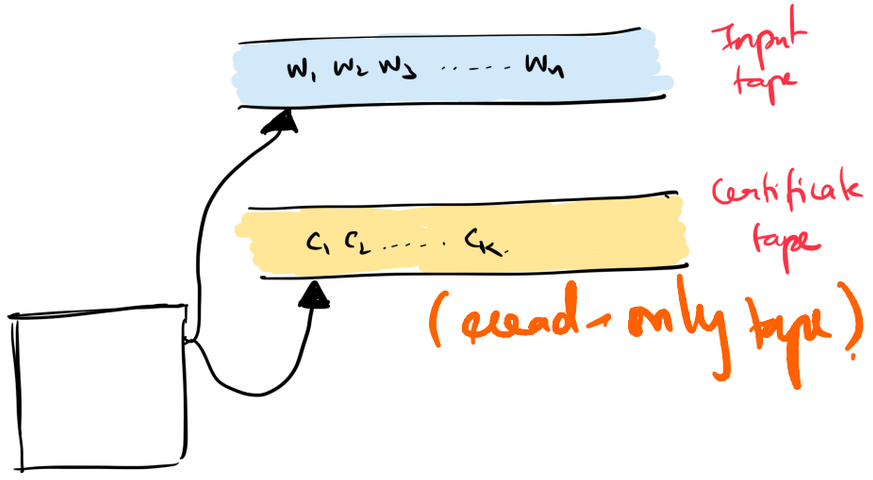
$$L(V) = \{x \in \{0,1\}^* : \exists c \in \{0,1\}^* \text{ s.t. } V \text{ accepts } (x, c)\}$$

The verifier decides $L(V)$ if it halts on all inputs (x, c)

Verifier



poly-time skeptical



Verifier: As a Turing Machine

Definition: The language $L(V)$ recognised by the verifier V is

$$L(V) = \{x \in \{0,1\}^* : \exists c \in \{0,1\}^* \text{ s.t. } V \text{ accepts } (x, c)\}$$

The verifier decides $L(V)$ if it halts on all inputs (x, c)

Verifier



poly-time skeptical

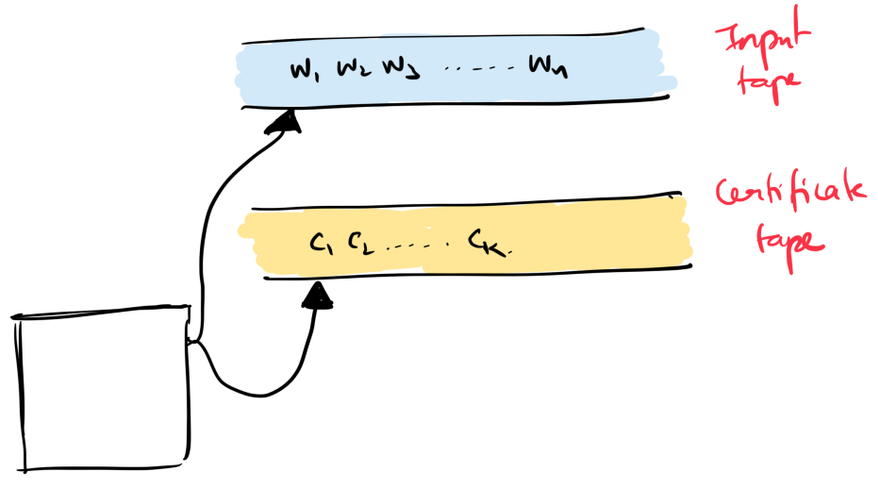
"Completeness"

" If $x \in L(V)$, $\exists c$ s.t. V accepts (x, c)"

"Soundness"

If $x \notin L(V)$ then, $\nexists c$ s.t. V accepts (x, c)

so $\forall c$, V rejects (x, c).



Verifier: As a Turing Machine

Definition: A polynomial-time verifier is a verifier that decides its language and has time cost $O(n^c)$ for some constant $c \geq 0$.

Thus, for any polynomial-time verifier V and $x \in L(V)$, there must be a certificate of length polynomial in $|x|$ that causes V to accept.

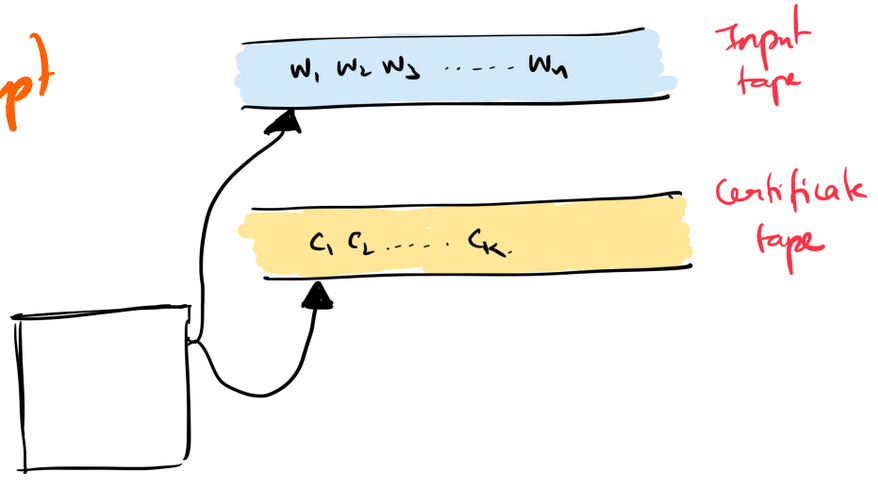
↳ $\forall x \in L(V)$ there must be a certificate c s.t. $c = |x|^k$. that causes V to accept (x, c)

$V(x, c)$ runs in polynomial time,

Verifier



poly-time skeptical



Verifier: As a Turing Machine

Definition: A polynomial-time verifier is a verifier that decides its language and has time cost $O(n^c)$ for some constant $c \geq 0$.

Thus, for any polynomial-time verifier V and $x \in L(V)$, there must be a certificate of length polynomial in $|x|$ that causes V to accept.

If V is polynomial time verifier.

$\Rightarrow \forall x \in L(V)$, $\exists c$ s.t. $V(x,c)$ is accepted & in polynomial time.

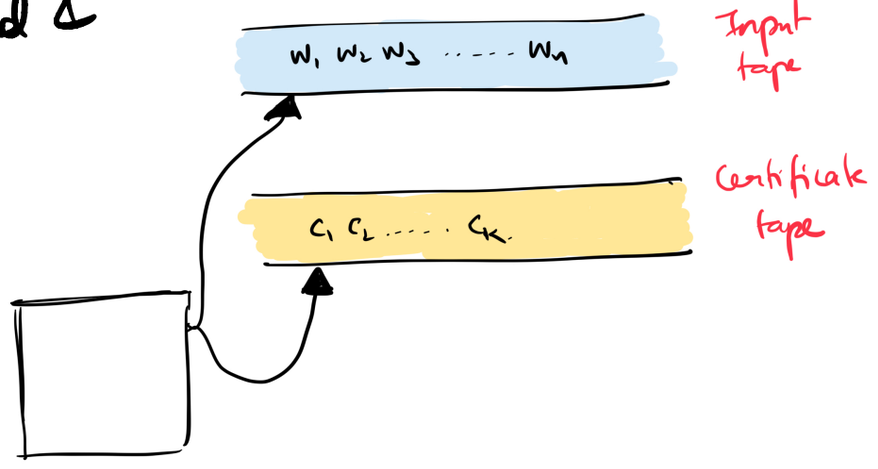
$\Rightarrow |c| \leq |x|^k$ for some constant k

($|c|$ is polynomial in $|x|$).

Verifier



poly-time skeptical



Verifier: As a Turing Machine

Definition: A polynomial-time verifier is a verifier that decides its language and has time cost $O(n^c)$ for some constant $c \geq 0$.

Thus, for any polynomial-time verifier V and $x \in L(V)$, there must be a certificate of length polynomial in $|x|$ that causes V to accept.

Verifier

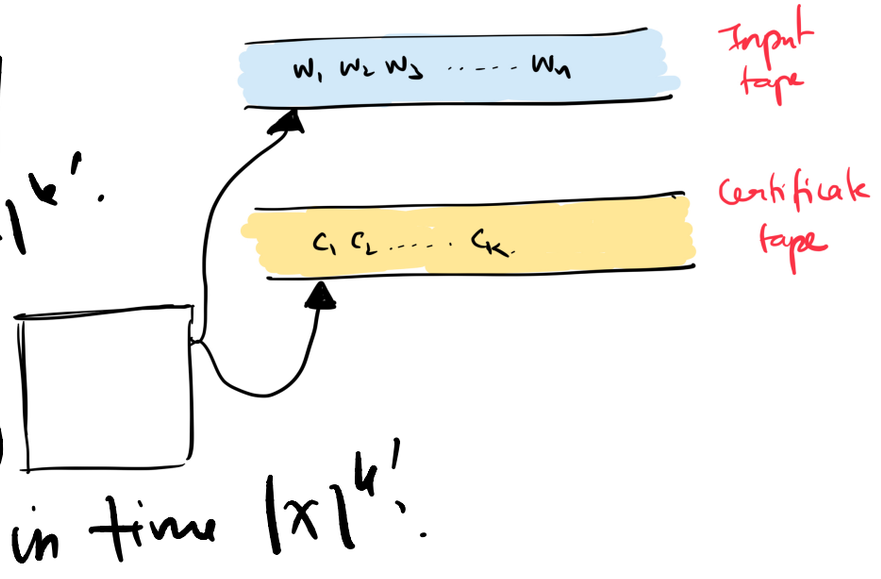


poly-time skeptical

A poly time verifier is a TM that for some constants k, k' does the following:

(1) $\forall w \in L(V)$, $\exists c$ s.t. $|c| \leq |x|^k$, and $V(x,c) = 1$ in time $|x|^{k'}$.

(2) $\forall w \notin L(V)$, $\forall c \in \{0,1\}^*$, $V(x,c) = 0$ in time $|x|^{k'}$.



Verifier: As a Turing Machine

Definition: A polynomial-time verifier is a verifier that decides its language and has time cost $O(n^c)$ for some constant $c \geq 0$.

Thus, for any polynomial-time verifier V and $x \in L(V)$, there must be a certificate of length polynomial in $|x|$ that causes V to accept.

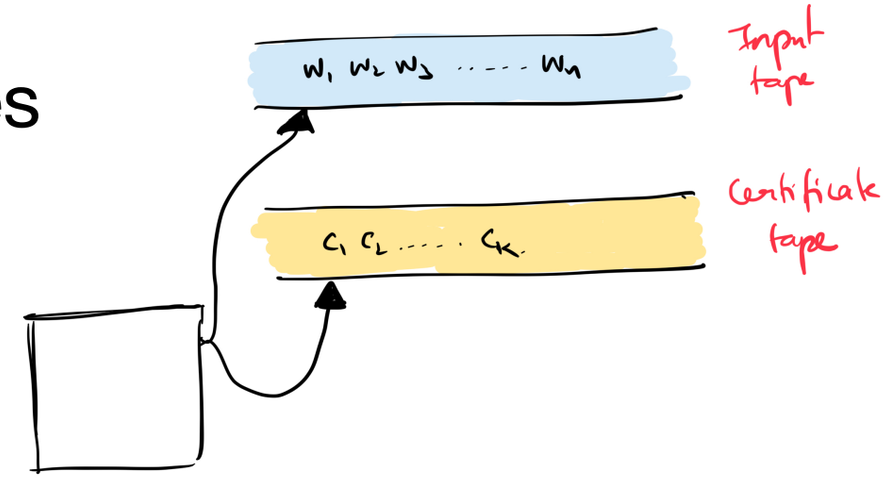
Verifier



poly-time skeptical

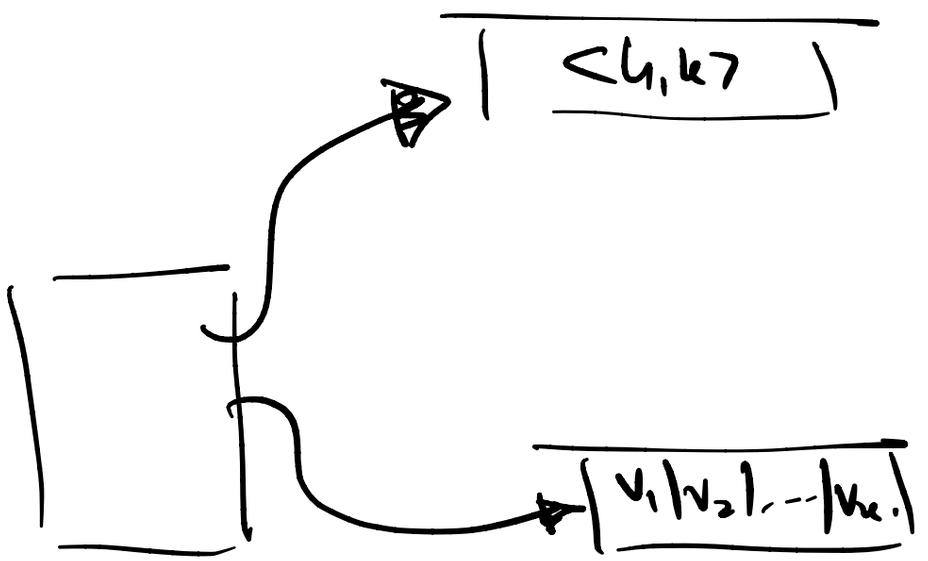
Definition: NP is the is the set of all languages that can be decided by polynomial-time verifiers.

NP is not a model of computability, but its a way of thinking about computation.



Example: CLIQUE

Definition: There exists a polynomial time verifier for CLIQUE.



Verifier: Given $\langle G, k, c \rangle$:

- Check if c encodes a k-clique in G.
- If yes, accept, if no, reject.

(a) $|\langle v_1, \ldots, v_k \rangle| = O(|\langle G, k \rangle|)$
$= O(n^2)$ size.

(b) Verifier takes time $O(k^2)$, which is polynomial in input.

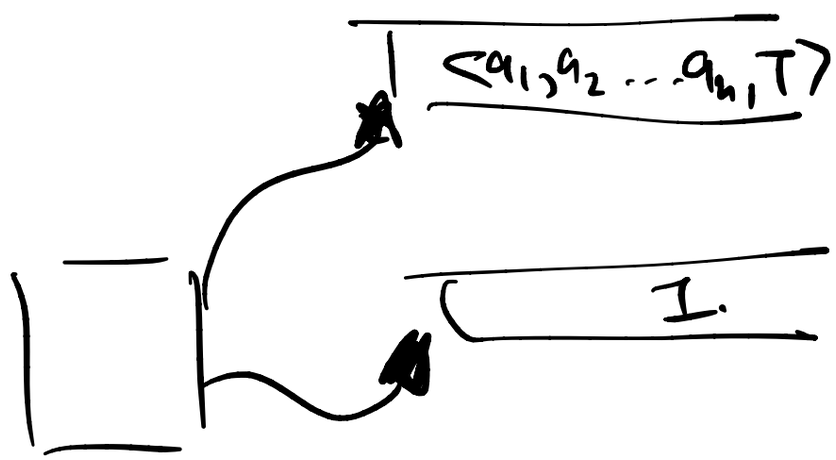
(1) Completeness: If $\langle G, k \rangle \in$ CLIQUE then, $\exists$ a list of such vertices

(2) Soundness: There is no such certificate

CLIQUE $\in$ NP.

Example: SUBSET-SUM

SUBSET-SUM $\in$ NP.



Definition: There exists a polynomial time verifier for SUBSET-SUM.

Verifier: Given $\langle a_1, a_2, \cdots, a_n, T, I \rangle$:

- Check if c encodes $I \subseteq [n]$ such that $\sum_{i \in I} a_i = T$
- If yes, accept, if no, reject.

(1) Completeness:
$\langle a_1, \cdots, a_n, T \rangle \in$ SUBSET-SUM.
Then, $\exists I$.

(2) Soundness:
If $\langle a_1, a_2 \cdots a_n, T \rangle \notin$ SUBSET-SUM
Then, $\nexists I$.

(1) Polytime:

(a) $|I| \leq O(n \log n) = |\langle a_1, a_2 \cdots, a_n, T \rangle|^k$.

(b) The time taken to check if $\sum_{i \in I} a_i = T$
$= n^{k'}$ for some constant k'.

NP-Completeness

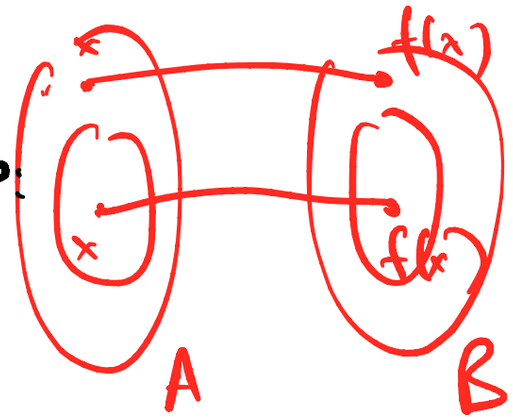
NP = consists of problems which are efficiently checkable.

P = consists of problems which are efficiently solvable.

P = NP ? Is the set of problems that are efficiently checkable = the set of problems that are efficiently solvable.

Polynomial-time Reducibility

Mapping reductions:



Definition: A language $A \subseteq \{0,1\}^*$ is polynomial-time reducible to the language B, denoted $A \leq_P B$ if and only if there exists a function $f : \{0,1\}^* \to \{0,1\}^*$ such that:

$x \notin A \Rightarrow f(x) \notin B$.

$x \in A \Rightarrow f(x) \in B$

- For every $x \in \{0,1\}^*$, we have $x \in A \iff f(x) \in B$,
- There is a polynomial time Turing machine F, that on any input $x \in \{0,1\}^*$, replaces it with $f(x)$ when it halts.

A mapping reduction that can be accomplished in polynomial time.

f: computable

∃ a decider F: which on i/p x, halts with f(x) written on the tape

Poly-time reducibility: That in addition F halt in time $|x|^k$ for some constant k.

Polynomial-time Reducibility

Theorem: For every two languages $A, B \subseteq \{0,1\}^*$ that satisfy $A \leq_P B$:
- If $B \in P$, then $A \in P$ as well, and
- If $B \in NP$, then $A \in NP$ as well.

Proof: $A \leq_p B$, and $B \in P$ then, $A \in P$ as well.

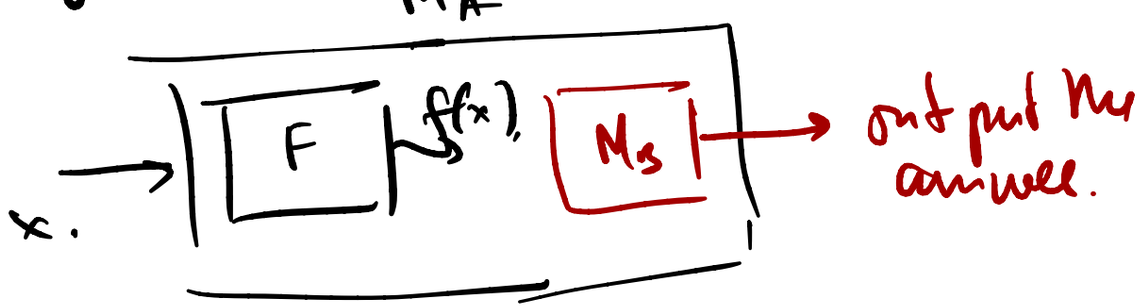
↓

(1) $\exists$ s.t. $x \in \{0,1\}^*$ to F, then, F halts in polytime & writes $f(x)$ on the tape

(2) $f(x) \in B \Leftrightarrow x \in A$.

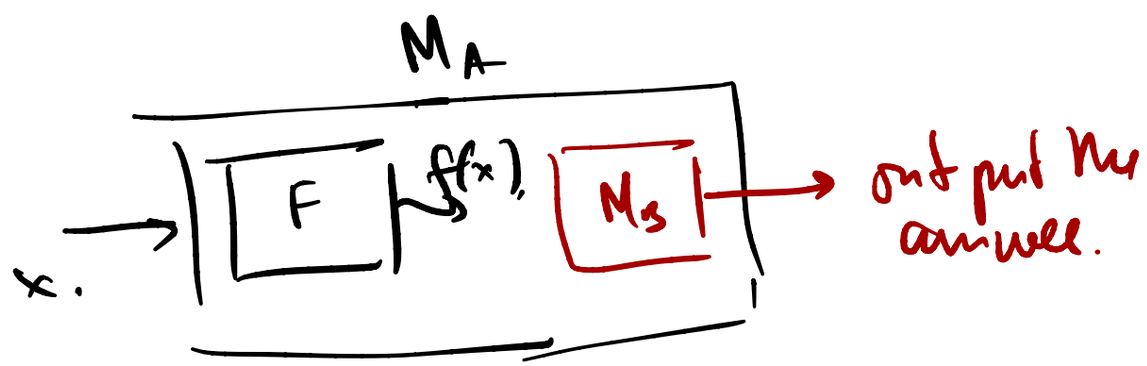
a) M_B that decides B in polynomial time: $O(n^k)$

b) Use this to come with a polynomial decider for A, M_A.



Polynomial-time Reducibility

Theorem: For every two languages $A, B \subseteq \{0,1\}^*$ that satisfy $A \leq_P B$:
- If $B \in P$, then $A \in P$ as well, and
- If $B \in NP$, then $A \in NP$ as well.



$x \in A \Leftrightarrow f(x) \in B$

$\Rightarrow M_B$ is a decider for B,

$\Rightarrow M_A$ is also a decider for A.

Runtime of M_A:

(1) Running $F(x)$: $|x|^k$ for some constant k.

(2) $M_B(f(x))$

$\Downarrow$

Runtime of M_B is $O(n^{k'})$

$\Rightarrow$ on $f(x)$: $|f(x)|^{k'}$

$\leq |x|^{k+k'}$.

$A \leq_P B$. $\Leftarrow$ polynomial time taken.

$\Rightarrow A \in P$.

Polynomial-time Reducibility

Corollary: For every two languages $A, B \subseteq \{0,1\}^*$ that satisfy $A \leq_P B$:

- If $A \notin P$, then $B \notin P$ as well, and
- If $A \notin NP$, then $B \notin NP$ as well.

Next class: SAT is NP-hard
(Cook-Levin Theorem)

B is called NP-hard.

lets say now in a language B:
$\forall A \in NP$:
$A \leq_p B$
$B \in P$, then, $A \in P$ as well.

But we don't know if any NP hard language is in P.

$P \subseteq NP$

a) $L \in P$, solve the problem, and give the solution as certificate to verifier.

NP-Hardness

Definition: The language L is NP-hard if every language $A \in NP$ satisfies $A \leq_P L$. The language L is NP-complete if $L \in NP$ and L is NP-hard.

Proposition: If any NP-hard language L is in P, then $P = NP$.
Equivalently, if $P \neq NP$ and L is NP-hard, then $L \notin P$.

A First NP Complete Language

Theorem: There exists an NP-complete language.