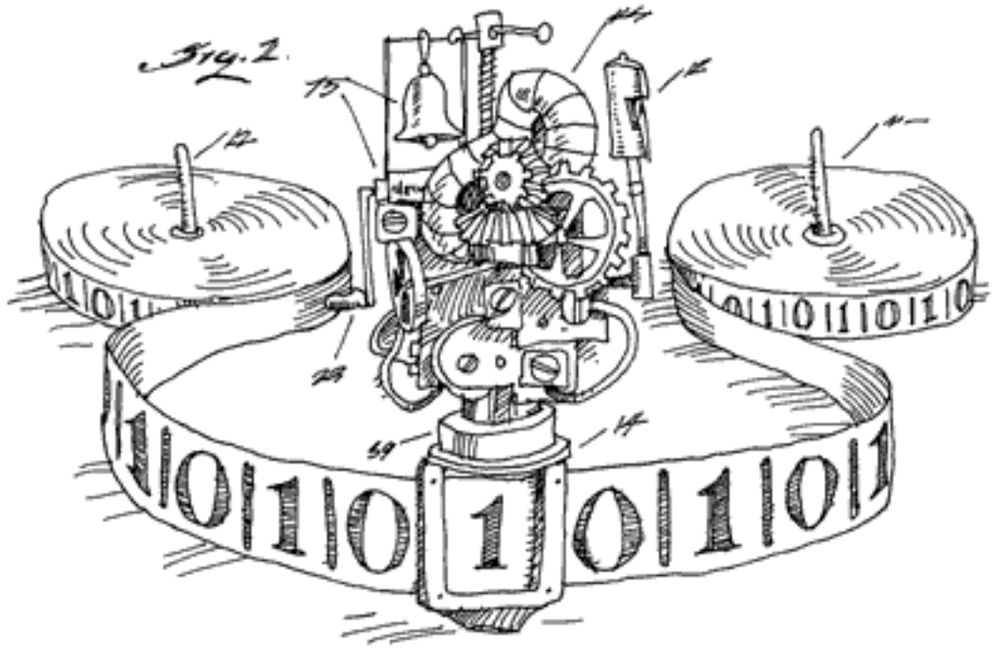
CSC4010: Introduction to Theoretical Computer Science



Lecture 12

How large is P?

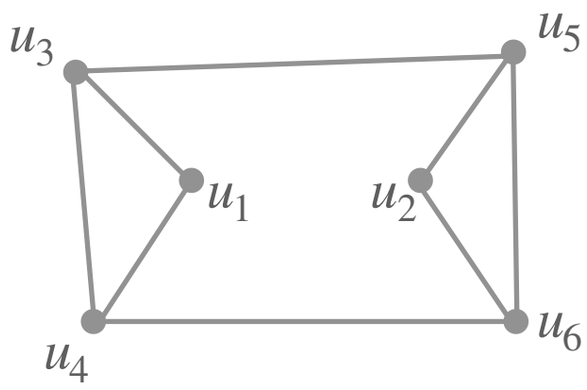
- Languages in P:
  - PALINDROMES, REACHABILITY…
- But last time: TIME(n) $\subsetneq$ TIME(n^2) $\subsetneq$ TIME(n^3) $\subsetneq$ $\cdots$
  - And similarly: $P \subsetneq$ TIME(2^n)
- How large is P? What is the boundary of class P? What are the class of problems which are almost in P but for which we do not have a polynomial time algorithm?

CLIQUE

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

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?

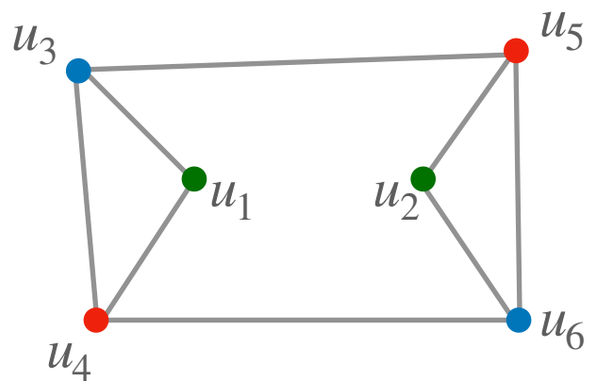


Graph G

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



Graph G

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

NP: A game between a Prover and a Verifier

Verifier



poly-time
skeptical

Prover



omniscient
unreliable

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

Verifier



poly-time
skeptical

Prover



omniscient
unreliable

“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.

Verifier

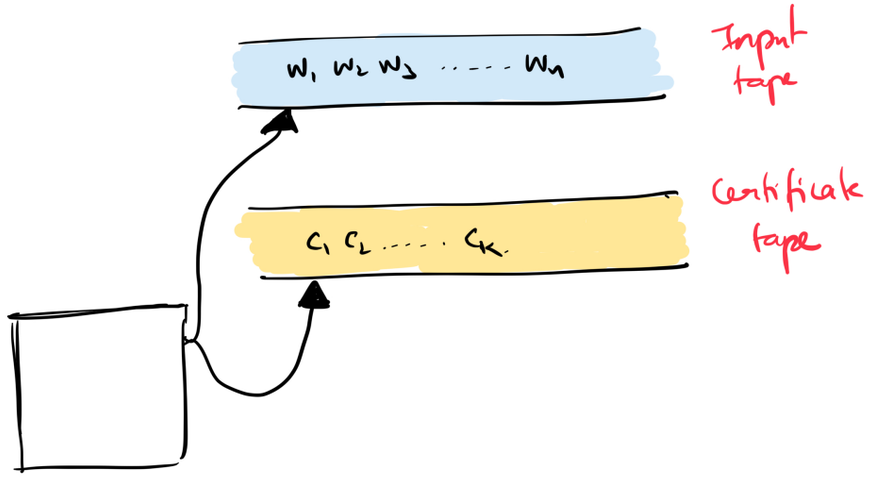


poly-time skeptical

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: 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)

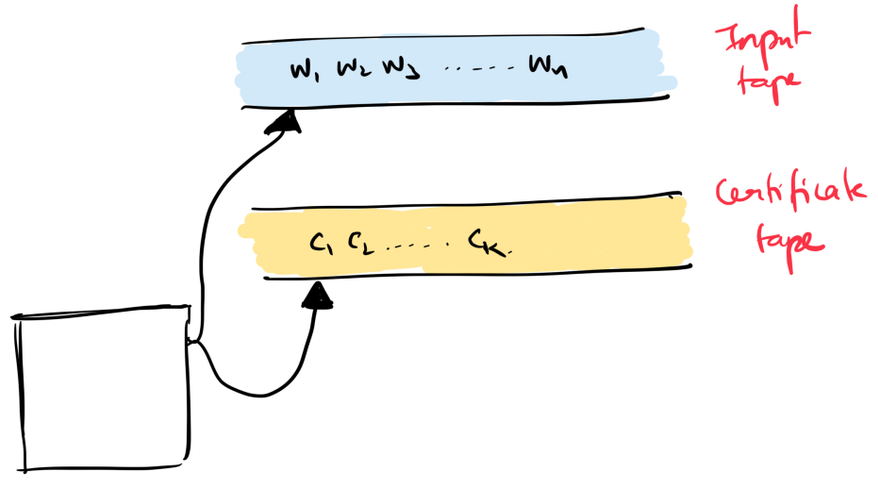
"Completeness"

"Soundness"

Verifier



poly-time skeptical



Verifier: As a Turing Machine

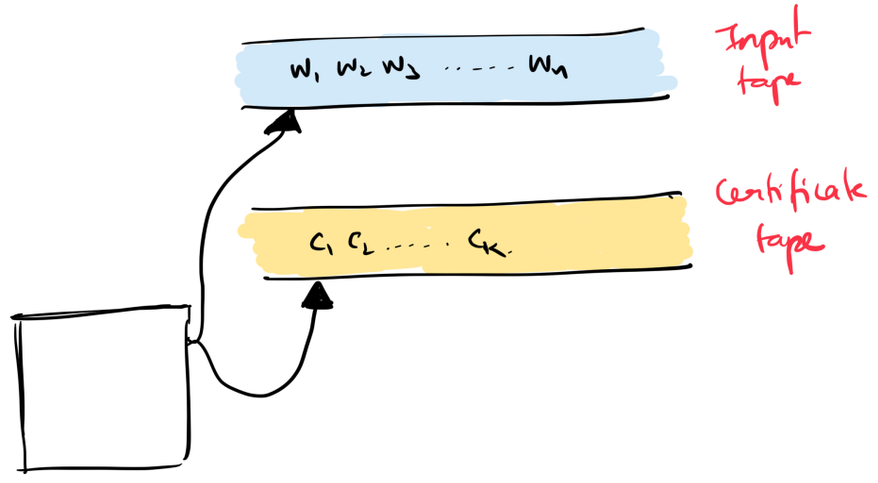
Definition: A polynomial-time verifier is a verifier that decides its language and has time cost $O(n^c)$ for some constant $c \geq 0$.

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



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 $c \geq 0$.

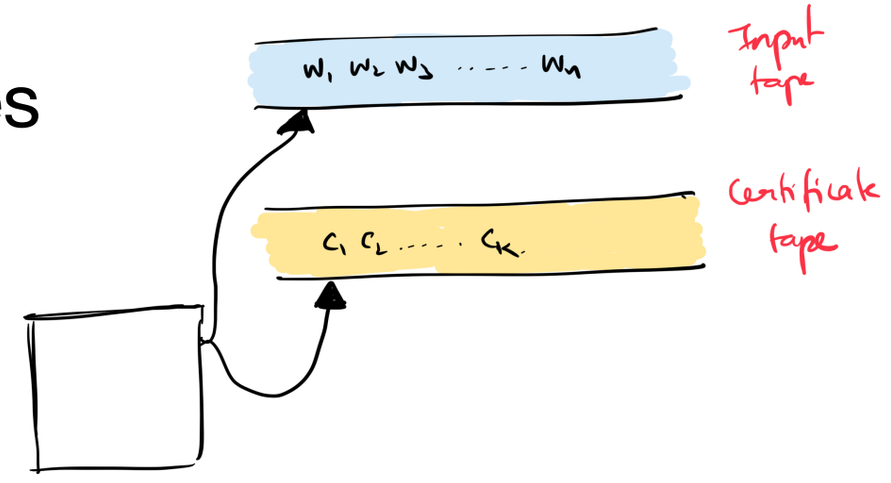
Thus, for any polynomial-time verifier V and $x \in L(V)$, there must be a certificate of length polynomial in $|x|$ that causes V to accept.

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

Verifier



poly-time skeptical



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.

Example: SUBSET-SUM

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.

NP-Completeness

Polynomial-time Reducibility

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:

- 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.

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.

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.

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.