COL 352 Introduction to Automata and Theory of Computation

Nikhil Balaji

Bharti 420
Indian Institute of Technology, Delhi
nbalaji@cse.iitd.ac.in

April 20, 2023

Lecture 33: Computational Complexity Theory (Part 2)

Time complexity and complexity classes

Let $t : \mathbb{N} \to \mathbb{N}$.

Definition

A language $L \subseteq \Sigma^*$ is said to be in class NTIME$(t(n))$ if there exists a non-deterministic Turing machine M such that $\forall x \in \Sigma^*$,

- each run of M halts on x in time $O(t(|x|))$, where $|x|$ indicates the length of x.
- if $x \in L$ then M accepts x on at least one run.
- if $x \notin L$ then M rejects x on all runs.

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

Time complexity and complexity classes

Let $t : \mathbb{N} \to \mathbb{N}$.

Definition

A language $L \subseteq \Sigma^*$ is said to be in class NTIME$(t(n))$ if there exists a non-deterministic Turing machine M such that $\forall x \in \Sigma^*$,

each run of M halts on x in time $O(t(|x|))$, where $|x|$ indicates the length of x.

if $x \in L$ then M accepts x on at least one run.

if $x \notin L$ then M rejects x on all runs.

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

$$NEXP = \bigcup_k \text{NTIME}(2^{n^k})$$

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

P $\subseteq$ NP by definition.

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

P $\subseteq$ NP by definition.

P $\subseteq$ EXP again by definition.

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

- P $\subseteq$ NP by definition.
- P $\subseteq$ EXP again by definition.
- Similarly, NP $\subseteq$ NEXP by definition.

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

- P $\subseteq$ NP by definition.
- P $\subseteq$ EXP again by definition.
- Similarly, NP $\subseteq$ NEXP by definition.
- Finally, NP $\subseteq$ EXP due to the previous lemma.

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

P ⊆ NP by definition.

P ⊆ EXP again by definition.

Similarly, NP ⊆ NEXP by definition.

Finally, NP ⊆ EXP due to the previous lemma.

$$\begin{tikzcd}
\mathrm{P} \arrow[r] & \mathrm{NP} \\
\mathrm{EXP} & \mathrm{NEXP}
\end{tikzcd}$$

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

- P $\subseteq$ NP by definition.
- P $\subseteq$ EXP again by definition.
- Similarly, NP $\subseteq$ NEXP by definition.
- Finally, NP $\subseteq$ EXP due to the previous lemma.

$$\begin{tikzcd}
\mathsf{P} \arrow[r] \arrow[d] & \mathsf{NP} \\
\mathsf{EXP} & \mathsf{NEXP}
\end{tikzcd}$$

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

- $\mathsf{P} \subseteq \mathsf{NP}$ by definition.
- $\mathsf{P} \subseteq \mathsf{EXP}$ again by definition.
- Similarly, $\mathsf{NP} \subseteq \mathsf{NEXP}$ by definition.
- Finally, $\mathsf{NP} \subseteq \mathsf{EXP}$ due to the previous lemma.

$$\begin{tikzcd}
\mathsf{P} \arrow[r] \arrow[d] & \mathsf{NP} \\
\mathsf{EXP} \arrow[r] & \mathsf{NEXP}
\end{tikzcd}$$

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

- $\mathsf{P} \subseteq \mathsf{NP}$ by definition.
- $\mathsf{P} \subseteq \mathsf{EXP}$ again by definition.
- Similarly, $\mathsf{NP} \subseteq \mathsf{NEXP}$ by definition.
- Finally, $\mathsf{NP} \subseteq \mathsf{EXP}$ due to the previous lemma.

$$
\begin{tikzcd}
\mathsf{P} \arrow[r] \arrow[d] & \mathsf{NP} \arrow[d] \\
\mathsf{EXP} \arrow[r] & \mathsf{NEXP}
\end{tikzcd}
$$

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

P $\subseteq$ NP by definition.

P $\subseteq$ EXP again by definition.

Similarly, NP $\subseteq$ NEXP by definition.

Finally, NP $\subseteq$ EXP due to the previous lemma.

$$\begin{tikzcd}
\mathsf{P} \arrow[r] \arrow[d] & \mathsf{NP} \arrow[d] \arrow[dl] \\
\mathsf{EXP} \arrow[r] & \mathsf{NEXP}
\end{tikzcd}$$

Relationships between complexity classes

How are P, NP, EXP, and NEXP related?

- P ⊆ NP by definition.
- P ⊆ EXP again by definition.
- Similarly, NP ⊆ NEXP by definition.
- Finally, NP ⊆ EXP due to the previous lemma.

$$\begin{tikzcd}
\mathsf{P} \arrow[r] \arrow[d] & \mathsf{NP} \arrow[d] \arrow[dl, red] \\
\mathsf{EXP} \arrow[r] & \mathsf{NEXP}
\end{tikzcd}$$

P vs. NP

P vs. NP

P the class of languages where membership can be decided quickly.

P vs. NP

P the class of languages where membership can be decided quickly.

NP the class of languages where membership can be verified quickly.

Definition

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

$$A = \{w \mid V \text{ accepts } (w, c) \text{ for some string } c\}$$

P vs. NP

P the class of languages where membership can be decided quickly.

NP the class of languages where membership can be verified quickly.

Definition

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

$$A = \{w \mid V \text{ accepts } (w, c) \text{ for some string } c\}$$

NP is the class of languages that have polynomial time verifiers. c is the "certificate" or "witness" or "proof" that $w \in A$.

Examples

$\mathsf{SAT} = \{\phi \mid \phi \text{ is satisfiable}\}$.

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$.

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

On input (G, s, t), where G is a directed graph with nodes s and t:

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

On input (G, s, t), where G is a directed graph with nodes s and t:

- Place a mark on node s.
- Repeat the following until no additional nodes are marked:
- Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
- If t is marked, accept. Otherwise, reject.

3-SAT $= \{\phi \mid \phi$ is a 3-CNF and satisfiable$\}$.

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

On input (G, s, t), where G is a directed graph with nodes s and t:

- Place a mark on node s.
- Repeat the following until no additional nodes are marked:
- Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
- If t is marked, accept. Otherwise, reject.

3-SAT $= \{\phi \mid \phi$ is a 3-CNF and satisfiable$\}$. in NP (and not known to be in P)

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

On input (G, s, t), where G is a directed graph with nodes s and t:

- Place a mark on node s.
- Repeat the following until no additional nodes are marked:
- Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
- If t is marked, accept. Otherwise, reject.

3-SAT $= \{\phi \mid \phi$ is a 3-CNF and satisfiable$\}$. in NP (and not known to be in P)

2-SAT $= \{\phi \mid \phi$ is a 2-CNF and satisfiable$\}$.

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

On input (G, s, t), where G is a directed graph with nodes s and t:

- Place a mark on node s.
- Repeat the following until no additional nodes are marked:
- Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
- If t is marked, accept. Otherwise, reject.

3-SAT $= \{\phi \mid \phi$ is a 3-CNF and satisfiable$\}$. in NP (and not known to be in P)

2-SAT $= \{\phi \mid \phi$ is a 2-CNF and satisfiable$\}$. in P

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

On input (G, s, t), where G is a directed graph with nodes s and t:

- Place a mark on node s.
- Repeat the following until no additional nodes are marked:
- Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
- If t is marked, accept. Otherwise, reject.

3-SAT $= \{\phi \mid \phi$ is a 3-CNF and satisfiable$\}$. in NP (and not known to be in P)

2-SAT $= \{\phi \mid \phi$ is a 2-CNF and satisfiable$\}$. in P

Factoring $= \{(k, n) \mid n$ has a factor $\leq k\}$.

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

On input (G, s, t), where G is a directed graph with nodes s and t:

- Place a mark on node s.
- Repeat the following until no additional nodes are marked:
- Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
- If t is marked, accept. Otherwise, reject.

3-SAT $= \{\phi \mid \phi$ is a 3-CNF and satisfiable$\}$. in NP (and not known to be in P)

2-SAT $= \{\phi \mid \phi$ is a 2-CNF and satisfiable$\}$. in P

Factoring $= \{(k, n) \mid n$ has a factor $\leq k\}$. Google it!

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

On input (G, s, t), where G is a directed graph with nodes s and t:

- Place a mark on node s.
- Repeat the following until no additional nodes are marked:
- Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
- If t is marked, accept. Otherwise, reject.

3-SAT $= \{\phi \mid \phi$ is a 3-CNF and satisfiable$\}$. in NP (and not known to be in P)

2-SAT $= \{\phi \mid \phi$ is a 2-CNF and satisfiable$\}$. in P

Factoring $= \{(k, n) \mid n$ has a factor $\leq k\}$. Google it!

Clique $= \{(G, k) \mid G$ has a clique of size $\geq k\}$.

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

On input (G, s, t), where G is a directed graph with nodes s and t:

- Place a mark on node s.
- Repeat the following until no additional nodes are marked:
- Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
- If t is marked, accept. Otherwise, reject.

3-SAT $= \{\phi \mid \phi$ is a 3-CNF and satisfiable$\}$. in NP (and not known to be in P)

2-SAT $= \{\phi \mid \phi$ is a 2-CNF and satisfiable$\}$. in P

Factoring $= \{(k, n) \mid n$ has a factor $\leq k\}$. Google it!

Clique $= \{(G, k) \mid G$ has a clique of size $\geq k\}$. in NP (and not known to be in P)

Examples

SAT $= \{\phi \mid \phi$ is satisfiable$\}$. in NP (and not known to be in P)

Reach $= \{(G, s, t) \mid t$ is reachable from s in $G\}$. in P

On input (G, s, t), where G is a directed graph with nodes s and t:

- Place a mark on node s.
- Repeat the following until no additional nodes are marked:
- Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
- If t is marked, accept. Otherwise, reject.

3-SAT $= \{\phi \mid \phi$ is a 3-CNF and satisfiable$\}$. in NP (and not known to be in P)

2-SAT $= \{\phi \mid \phi$ is a 2-CNF and satisfiable$\}$. in P

Factoring $= \{(k, n) \mid n$ has a factor $\leq k\}$. Google it!

Clique $= \{(G, k) \mid G$ has a clique of size $\geq k\}$. in NP (and not known to be in P)

Lower Bounds

How do we separate NP from P?

Lower Bounds

How do we separate NP from P?

To prove	Method used

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	pumping lemma for REG

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	pumping lemma for REG
non-context-free	

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	pumping lemma for REG
non-context-free	pumping lemma or CFLs

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	pumping lemma for REG
non-context-free	pumping lemma or CFLs
not recognizable	

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	pumping lemma for REG
non-context-free	pumping lemma or CFLs
not recognizable	diagonalization

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	pumping lemma for REG
non-context-free	pumping lemma or CFLs
not recognizable	diagonalization
not decidable	

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	pumping lemma for REG
non-context-free	pumping lemma or CFLs
not recognizable	diagonalization
not decidable	Rice's theorem

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	pumping lemma for REG
non-context-free	pumping lemma or CFLs
not recognizable	diagonalization
not decidable	Rice's theorem or diagonalization and reductio

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	pumping lemma for REG
non-context-free	pumping lemma or CFLs
not recognizable	diagonalization
not decidable	Rice's theorem or diagonalization and reductio
not in P	

Lower Bounds

How do we separate NP from P?

To prove	Method used
not regular	pumping lemma for REG
non-context-free	pumping lemma or CFLs
not recognizable	diagonalization
not decidable	Rice's theorem or diagonalization and reductio
not in P	???

Finer structure inside P

Definition

A function $t : \mathbb{N} \to \mathbb{N}$ is said to be time constructible

Finer structure inside P

Definition

A function $t : \mathbb{N} \to \mathbb{N}$ is said to be time constructible if the there exists a TM that on input 1^n, it outputs $t(n)$ in time $O(t(n))$.

Finer structure inside P

Definition

A function $t : \mathbb{N} \to \mathbb{N}$ is said to be time constructible if the there exists a TM that on input 1^n, it outputs $t(n)$ in time $O(t(n))$.

Examples

Finer structure inside P

Definition

A function $t : \mathbb{N} \to \mathbb{N}$ is said to be time constructible if the there exists a TM that on input 1^n, it outputs $t(n)$ in time $O(t(n))$.

Examples

n^2

Finer structure inside P

Definition

A function $t : \mathbb{N} \to \mathbb{N}$ is said to be time constructible if the there exists a TM that on input 1^n, it outputs $t(n)$ in time $O(t(n))$.

Examples

n^2, $n \log n$.

Finer structure inside P

Definition

A function $t : \mathbb{N} \to \mathbb{N}$ is said to be time constructible if the there exists a TM that on input 1^n, it outputs $t(n)$ in time $O(t(n))$.

Examples

n^2, $n \log n$.

Theorem

Let $t : \mathbb{N} \to \mathbb{N}$ be a time constructible function. There exists a language L such that $L \in TIME(t(n)^2)$, but $L \notin TIME(o(t(n)))$.

Finer structure inside P

Definition

A function $t : \mathbb{N} \to \mathbb{N}$ is said to be time constructible if the there exists a TM that on input 1^n, it outputs $t(n)$ in time $O(t(n))$.

Examples

n^2, $n \log n$.

Theorem

Let $t : \mathbb{N} \to \mathbb{N}$ be a time constructible function. There exists a language L such that $L \in TIME(t(n)^2)$, but $L \notin TIME(o(t(n)))$.

Idea: On input x, simulate the execution of M_x on x for $|x|^{1.5}$ steps using a Universal TM. If U outputs some bit $b \in \{0, 1\}$ in this time, then output the opposite answer (i.e., output $1 - b$). Else output 0.

Efficient simulation

Efficient simulation

Theorem (Efficient universal Turing machine)

There exists a TM U such that for every $x, \alpha \in \{0,1\}^*, U(x,\alpha) = M_\alpha(x)$, where M_α denotes the TM represented by α. Moreover, if M_α halts on input x within T steps then $U(x,\alpha)$ halts within $CT \log T$ steps, where C is a number independent of $|x|$ and depending only on M_α's alphabet size, number of tapes, and number of states.

Polynomial time reductions and NP-hardness

Definition

A function $f : \Sigma^* \to \Sigma^*$ is polynomial time computable if there is a polynomial time Turing machine TM, say M, such that on any input $w \in \Sigma^*$, M stops with only $f(w)$ on its tape.

Polynomial time reductions and NP-hardness

Definition

A function $f : \Sigma^* \to \Sigma^*$ is polynomial time computable if there is a polynomial time Turing machine TM, say M, such that on any input $w \in \Sigma^*$, M stops with only $f(w)$ on its tape.

Polynomial time reductions and NP-hardness

Definition

A language L_1 is said to be polynomial time reducible to another language L_2

Polynomial time reductions and NP-hardness

Definition

A language L_1 is said to be polynomial time reducible to another language L_2, denoted as $L_1 \leq_m L_2$

Polynomial time reductions and NP-hardness

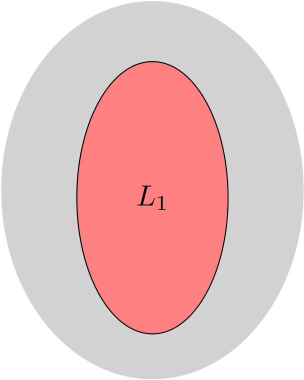
Definition

A language L_1 is said to be polynomial time reducible to another language L_2, denoted as $L_1 \leq_m L_2$, if there exists a polynomial time computable function f such that for all $w \in \Sigma^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.

Polynomial time reductions and NP-hardness

Definition

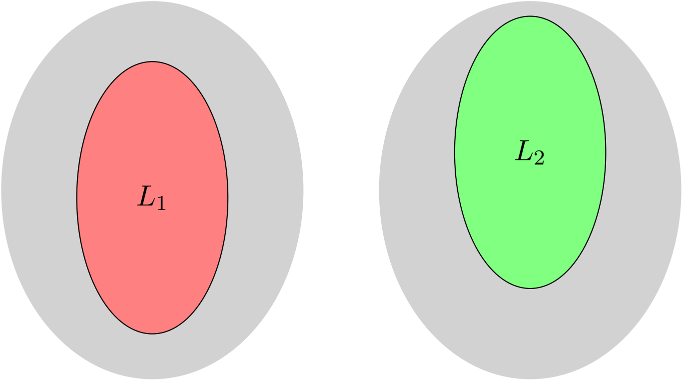
A language L_1 is said to be polynomial time reducible to another language L_2, denoted as $L_1 \leq_m L_2$, if there exists a polynomial time computable function f such that for all $w \in \Sigma^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.



Polynomial time reductions and NP-hardness

Definition

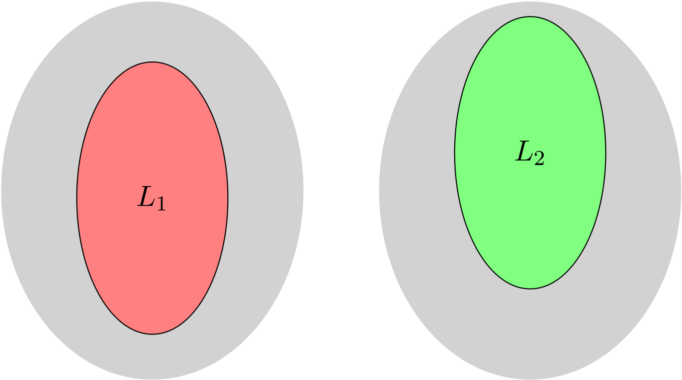
A language L_1 is said to be polynomial time reducible to another language L_2, denoted as $L_1 \leq_m L_2$, if there exists a polynomial time computable function f such that for all $w \in \Sigma^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.



Polynomial time reductions and NP-hardness

Definition

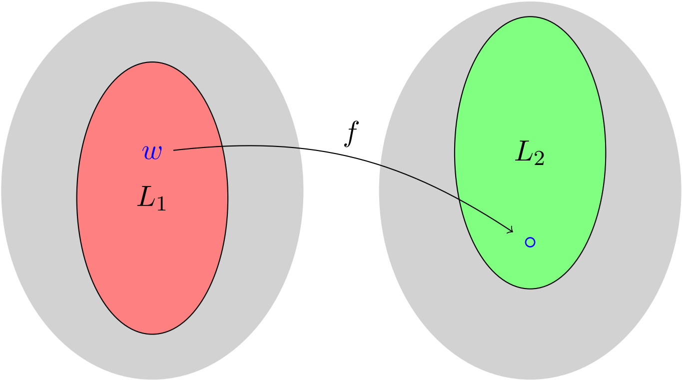
A language L_1 is said to be polynomial time reducible to another language L_2, denoted as $L_1 \leq_m L_2$, if there exists a polynomial time computable function f such that for all $w \in \Sigma^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.



Polynomial time reductions and NP-hardness

Definition

A language L_1 is said to be polynomial time reducible to another language L_2, denoted as $L_1 \leq_m L_2$, if there exists a polynomial time computable function f such that for all $w \in \Sigma^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.



Polynomial time reductions and NP-hardness

Definition

A language L_1 is said to be polynomial time reducible to another language L_2, denoted as $L_1 \leq_m L_2$, if there exists a polynomial time computable function f such that for all $w \in \Sigma^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.