Foundations of Computing

Lecture 26 – Final Exam Review

Arkady Yerukhimovich

April 25, 2024

Outline

1. Lecture 25 Review
2. Complexity Theory
   - $\mathcal{P}$
   - $\mathcal{NP}$
   - Poly-time Reductions and $\mathcal{NP}$-Completeness
   - Interactive Proofs
   - Zero-Knowledge Proofs
3. Computability
   - Turing Machines and Decidable Languages
   - Languages Recognized by TMs
   - Undecidable Languages
   - Proofs by Reduction
4. Automata and Languages

Lecture 25 Review

- Zero-Knowledge Proofs
- Where's Waldo
- Puppy and Panda
- Graph Isomorphism
- 3-Coloring

We Are Done!

Welcome to the last lecture of CS 3313!!!

- Complete course evaluation form for 5 points on final exam

Outline

Complexity Classes

Complexity Classes

- $\mathcal{P}$

Complexity Classes

- $\mathcal{P}$
- $\mathcal{NP}$

Complexity Classes

- $\mathcal{P}$
- $\mathcal{NP}$
- co-$\mathcal{NP}$

Complexity Classes

- $\mathcal{P}$
- $\mathcal{NP}$
- co-$\mathcal{NP}$
- $\mathcal{IP}$

Complexity Classes

- $\mathcal{P}$
- $\mathcal{NP}$
- co-$\mathcal{NP}$
- $\mathcal{IP}$

Important

Make sure you know the definitions and relationships between these complexity classes.

Asymptotic Notation – Big-O

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) = O(g(n))$ if

- There exist positive integers c, n_0 s.t. for all $n \geq n_0$

$$f(n) \leq cg(n)$$

Asymptotic Notation – Big-O

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) = O(g(n))$ if

- There exist positive integers c, n_0 s.t. for all $n \geq n_0$

$$f(n) \leq cg(n)$$

Example

$$f(n) = 5n^3 + 3n^2 + 10n + 8$$

Asymptotic Notation – Big-O

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) = O(g(n))$ if

- There exist positive integers c, n_0 s.t. for all $n \geq n_0$

$$f(n) \leq cg(n)$$

Example

$$f(n) = 5n^3 + 3n^2 + 10n + 8$$

- $f(n) = O(n^3)$

Asymptotic Notation – Big-O

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) = O(g(n))$ if

- There exist positive integers c, n_0 s.t. for all $n \geq n_0$

$$f(n) \leq cg(n)$$

Example

$$f(n) = 5n^3 + 3n^2 + 10n + 8$$

- $f(n) = O(n^3)$
- For every $n \geq 6$, $f(n) \leq 6n^3$
- I.e., $n_0 = 6, c = 6$

Asymptotic Notation – Big-O

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) = O(g(n))$ if

- There exist positive integers c, n_0 s.t. for all $n \geq n_0$

$$f(n) \leq cg(n)$$

Example

$$f(n) = 5n^3 + 3n^2 + 10n + 8$$

- $f(n) = O(n^3)$
- For every $n \geq 6$, $f(n) \leq 6n^3$
- I.e., $n_0 = 6, c = 6$
- Note that $f(n) = O(n^4)$

Outline

Complexity Class $\mathcal{P}$

Definition

$\mathcal{P}$ is the class of languages decidable in polynomial time on a 1-tape deterministic TM.

Complexity Class $\mathcal{P}$

Definition

$\mathcal{P}$ is the class of languages decidable in polynomial time on a 1-tape deterministic TM.

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

Complexity Class $\mathcal{P}$

Definition

$\mathcal{P}$ is the class of languages decidable in polynomial time on a 1-tape deterministic TM.

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

- $\mathcal{P}$ corresponds to the class of "efficiently-solvable" problems

Complexity Class $\mathcal{P}$

Definition

$\mathcal{P}$ is the class of languages decidable in polynomial time on a 1-tape deterministic TM.

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

- $\mathcal{P}$ corresponds to the class of "efficiently-solvable" problems
- $\mathcal{P}$ is invariant for all models of computation polynomially-equivalent to 1-tape TM

Complexity Class $\mathcal{P}$

Definition

$\mathcal{P}$ is the class of languages decidable in polynomial time on a 1-tape deterministic TM.

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

- $\mathcal{P}$ corresponds to the class of "efficiently-solvable" problems
- $\mathcal{P}$ is invariant for all models of computation polynomially-equivalent to 1-tape TM
- $\mathcal{P}$ has nice closure properties

Problems in $\mathcal{P}$

- PATH
- RELPRIME
- Anything you saw in algorithms class

Outline

The Class $\mathcal{NP}$

Definition

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

The Class $\mathcal{NP}$

Definition

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

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\mathcal{P} \subseteq \mathcal{NP}$

The Class $\mathcal{NP}$

Definition

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

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\mathcal{P} \subseteq \mathcal{NP}$

Intuition

- $\mathcal{P}$ is the class of problems where you can find a solution in poly-time

The Class $\mathcal{NP}$

Definition

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

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\quad \mathcal{P} \subseteq \mathcal{NP}$

Intuition

- $\mathcal{P}$ is the class of problems where you can find a solution in poly-time
- $\mathcal{NP}$ is the class of problems where you can verify a solution in poly-time

The Class $\mathcal{NP}$

Definition

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

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\mathcal{P} \subseteq \mathcal{NP}$

Intuition

- $\mathcal{P}$ is the class of problems where you can find a solution in poly-time
- $\mathcal{NP}$ is the class of problems where you can verify a solution in poly-time
- Question: $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent – A language L is poly-time verifiable if and only if it is decided by a poly-time NTM.

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent – A language L is poly-time verifiable if and only if it is decided by a poly-time NTM.

Proof Idea:

- Need to prove both directions
- An NTM simulates the verifier by guessing the witness w

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent – A language L is poly-time verifiable if and only if it is decided by a poly-time NTM.

Proof Idea:

- Need to prove both directions
- An NTM simulates the verifier by guessing the witness w
- A verifier simulates the NTM by using the accepting branch as the witness

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{NP}$

$L \in \mathcal{NP}$ if there exists poly-time DTM V s.t.

- for $x \in L$, there exists a witness w s.t. $V(x, w) = 1$
- for $x \notin L$, for all w, $V(x, w) = 0$

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{NP}$

$L \in \mathcal{NP}$ if there exists poly-time DTM V s.t.

- for $x \in L$, there exists a witness w s.t. $V(x, w) = 1$
- for $x \notin L$, for all w, $V(x, w) = 0$

co-$\mathcal{NP}$

$L \in$ co-$\mathcal{NP}$ if there exists poly-time DTM V s.t.

- for $x \notin L$, there exists a witness w s.t. $V(x, w) = 1$
- for $x \in L$, for all w, $V(x, w) = 0$

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{NP}$

$L \in \mathcal{NP}$ if there exists poly-time DTM V s.t.

- for $x \in L$, there exists a witness w s.t. $V(x, w) = 1$
- for $x \notin L$, for all w, $V(x, w) = 0$

co-$\mathcal{NP}$

$L \in$ co-$\mathcal{NP}$ if there exists poly-time DTM V s.t.

- for $x \notin L$, there exists a witness w s.t. $V(x, w) = 1$
- for $x \in L$, for all w, $V(x, w) = 0$

Question:

Is $\mathcal{P} = \mathcal{NP} =$ co-$\mathcal{NP}$?

Problems in $\mathcal{NP}$

- CLIQUE

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity
- Satisfiability

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity
- Satisfiability
- 3-SAT

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity
- Satisfiability
- 3-SAT
- Vertex cover

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity
- Satisfiability
- 3-SAT
- Vertex cover
- Independent set

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity
- Satisfiability
- 3-SAT
- Vertex cover
- Independent set
- and many more

Important

Make sure you know how to prove $L \in \mathcal{NP}$

Outline

Mapping Reductions

Mapping Reduction

Language A is mapping reducible to language B ($A \leq_m B$) if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every x,

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

Mapping Reductions

Mapping Reduction

Language A is mapping reducible to language B ($A \leq_m B$) if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every x,

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

Poly-time Mapping Reduction

Language A is poly-time mapping reducible to language B ($A \leq_P B$) if there is a poly-time computable function $f : \Sigma^* \to \Sigma^*$, where for every x,

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

Mapping Reductions

Mapping Reduction

Language A is mapping reducible to language B ($A \leq_m B$) if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every x,

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

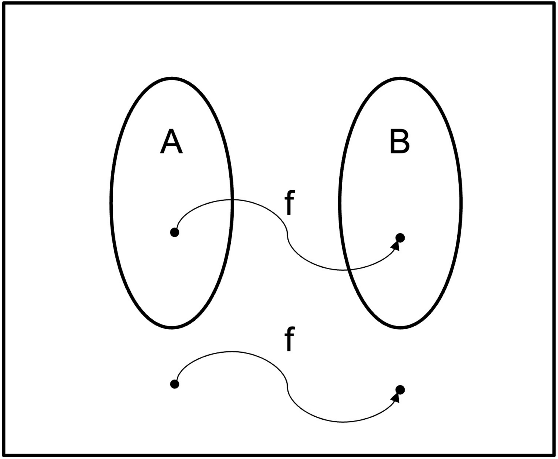
Poly-time Mapping Reduction

Language A is poly-time mapping reducible to language B ($A \leq_P B$) if there is a poly-time computable function $f : \Sigma^* \to \Sigma^*$, where for every x,

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

- Poly-time reductions give an efficient way to convert membership testing in A to membership testing in B
- If B has a poly-time solution so does A

Poly-time Mapping Reductions



f runs in time $poly(|x|)$ on all inputs x

Why Poly-Time Reductions

Theorem

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let M be the poly-time TM deciding B
- Let f be the poly-time reduction from A to B
- Can construct M' deciding A:
  M' = On input x:
  1. Compute $f(x)$
  2. Run $M(f(x))$ and output whatever M outputs

Why Poly-Time Reductions

Theorem

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let M be the poly-time TM deciding B
- Let f be the poly-time reduction from A to B
- Can construct M' deciding A:
  M' = On input x:
  1. Compute $f(x)$
  2. Run $M(f(x))$ and output whatever M outputs
     - If $x \in A$, $f(x) \in B$ so M accepts
     - If $x \notin A$, $f(x) \notin B$, so M rejects
     - Since both f and M are poly-time, $M(f(x))$ is also poly-time

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k

3SAT $\leq_P$ CLIQUE

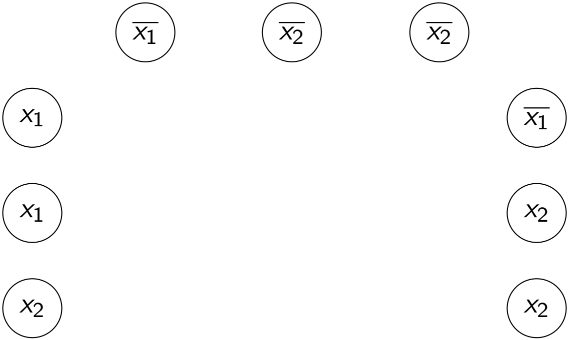
- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k
  - If ϕ is not satisfiable, G has no clique of size k

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k
  - If ϕ is not satisfiable, G has no clique of size k
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

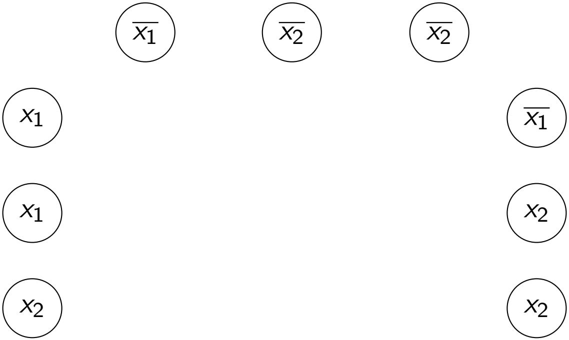
3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k
  - If ϕ is not satisfiable, G has no clique of size k
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$



3SAT $\leq_P$ CLIQUE

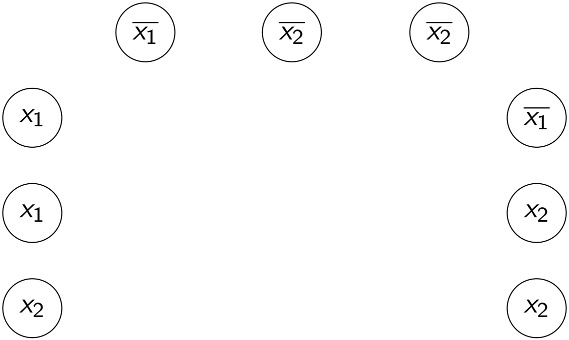
- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k
  - If ϕ is not satisfiable, G has no clique of size k
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$



- If ϕ is satisfiable then G has a k-clique

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k
  - If ϕ is not satisfiable, G has no clique of size k
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$



- If ϕ is satisfiable then G has a k-clique
- If G has a k-clique then ϕ is satisfiable

$\mathcal{NP}$-Completeness

Definition

A language B is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

$\mathcal{NP}$-Completeness

Definition

A language B is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- B is "as hard" as any language in $\mathcal{NP}$

$\mathcal{NP}$-Completeness

Definition

A language B is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- B is "as hard" as any language in $\mathcal{NP}$
- To study hardness of $\mathcal{NP}$, enough to study hardness of some $\mathcal{NP}$-complete problem

$\mathcal{NP}$-Completeness

Definition

A language B is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- B is "as hard" as any language in $\mathcal{NP}$
- To study hardness of $\mathcal{NP}$, enough to study hardness of some $\mathcal{NP}$-complete problem

Theorem

If B is $\mathcal{NP}$-complete and $B \in \mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$

$\mathcal{NP}$-Completeness

Definition

A language B is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- B is "as hard" as any language in $\mathcal{NP}$
- To study hardness of $\mathcal{NP}$, enough to study hardness of some $\mathcal{NP}$-complete problem

Theorem

If B is $\mathcal{NP}$-complete and $B \in \mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$

Theorem

If B is $\mathcal{NP}$-complete and $B \leq_P C$ for $C \in \mathcal{NP}$, then C is $\mathcal{NP}$-complete

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover
5. Vertex Cover $\leq_P$ Independent Set

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover
5. Vertex Cover $\leq_P$ Independent Set
6. 3-SAT $\leq_P$ 3-Color

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover
5. Vertex Cover $\leq_P$ Independent Set
6. 3-SAT $\leq_P$ 3-Color
7. More on the HW

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover
5. Vertex Cover $\leq_P$ Independent Set
6. 3-SAT $\leq_P$ 3-Color
7. More on the HW

Important

Make sure you remember what direction the reduction should go.

Outline

The Class $\mathcal{IP}$

Definition

$L \in \mathcal{IP}$ if there exist a pair of interactive algorithms (P, V) with V being poly-time (in $|x|$) s.t.

The Class $\mathcal{IP}$

Definition

$L \in \mathcal{IP}$ if there exist a pair of interactive algorithms (P, V) with V being poly-time (in $|x|$) s.t.

1. (Completeness) If $x \in L$, then $\Pr[\langle P, V\rangle(x) = 1] = 1$

The Class $\mathcal{IP}$

Definition

$L \in \mathcal{IP}$ if there exist a pair of interactive algorithms (P, V) with V being poly-time (in $|x|$) s.t.

1. (Completeness) If $x \in L$, then $\Pr[\langle P, V\rangle(x) = 1] = 1$
2. (Soundness) If $x \notin L$, then for any (possibly unbounded) P^*, we have $\Pr[\langle P^*, V\rangle(x) = 1] \leq 1/2$

The Class $\mathcal{IP}$

Definition

$L \in \mathcal{IP}$ if there exist a pair of interactive algorithms (P, V) with V being poly-time (in $|x|$) s.t.

1. (Completeness) If $x \in L$, then $\Pr[\langle P, V\rangle(x) = 1] = 1$
2. (Soundness) If $x \notin L$, then for any (possibly unbounded) P^*, we have $\Pr[\langle P^*, V\rangle(x) = 1] \leq 1/2$

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0,1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0,1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0,1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
    - Then G^* can only be isomorphic to one of the two graphs
    - P can perfectly determine which one this is

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0,1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs
   - P can perfectly determine which one this is
   - So $\Pr[b' = b] = 1$

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
    - Then G^* can only be isomorphic to one of the two graphs
    - P can perfectly determine which one this is
    - So $\Pr[b' = b] = 1$
2. (Soundness) Suppose that G_0 and G_1 are isomorphic

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
    - Then G^* can only be isomorphic to one of the two graphs
    - P can perfectly determine which one this is
    - So $\Pr[b' = b] = 1$
2. (Soundness) Suppose that G_0 and G_1 are isomorphic
    - Then G^* is isomorphic to both G_0 and G_1

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs
   - P can perfectly determine which one this is
   - So $\Pr[b' = b] = 1$
2. (Soundness) Suppose that G_0 and G_1 are isomorphic
   - Then G^* is isomorphic to both G_0 and G_1
   - P has no way to tell which one V started from

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs
   - P can perfectly determine which one this is
   - So $\Pr[b' = b] = 1$
2. (Soundness) Suppose that G_0 and G_1 are isomorphic
   - Then G^* is isomorphic to both G_0 and G_1
   - P has no way to tell which one V started from
   - Thus, $\Pr[b' = b] = 1/2$

Another Example – Polynomial Identity Testing

PIT Problem

Another Example – Polynomial Identity Testing

PIT Problem

- Prover P has a degree d polynomial f and wants to prove that

$$\forall x, f(x) = 0$$

Another Example – Polynomial Identity Testing

PIT Problem

- Prover P has a degree d polynomial f and wants to prove that

$$\forall x, f(x) = 0$$

- V is allowed to query $f(x)$ at points x of its choice – but, P knows V's strategy

Another Example – Polynomial Identity Testing

PIT Problem

- Prover P has a degree d polynomial f and wants to prove that

$$\forall x, f(x) = 0$$

- V is allowed to query $f(x)$ at points x of its choice – but, P knows V's strategy

Question: What should V do?

Another Example – Polynomial Identity Testing

PIT Problem

- Prover P has a degree d polynomial f and wants to prove that

$$\forall x, f(x) = 0$$

- V is allowed to query $f(x)$ at points x of its choice – but, P knows V's strategy

Question: What should V do?

- Suppose that V is deterministic:

Another Example – Polynomial Identity Testing

PIT Problem

- Prover P has a degree d polynomial f and wants to prove that

$$\forall x, f(x) = 0$$

- V is allowed to query $f(x)$ at points x of its choice – but, P knows V's strategy

Question: What should V do?

- Suppose that V is deterministic:
- What if you allow V to be randomized:

Languages in $\mathcal{IP}$

- $\mathcal{P} \subseteq \mathcal{IP}$
- $\mathcal{NP} \subseteq \mathcal{IP}$
- Graph Non-Isomorphism $\in \mathcal{IP}$

Outline

1. Lecture 25 Review
2. Complexity Theory
   - $\mathcal{P}$
   - $\mathcal{NP}$
   - Poly-time Reductions and $\mathcal{NP}$-Completeness
   - Interactive Proofs
   - Zero-Knowledge Proofs
3. Computability
   - Turing Machines and Decidable Languages
   - Languages Recognized by TMs
   - Undecidable Languages
   - Proofs by Reduction
4. Automata and Languages

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle (x)$$

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle (x)$$

Define V's view of this interaction by:

$$VIEW_V(\langle P, V \rangle (x))$$

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle(x)$$

Define V's view of this interaction by:

$$VIEW_V(\langle P, V \rangle(x))$$

This includes:

- V's randomness
- Any messages that V receives

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle (x)$$

Define V's view of this interaction by:

$$VIEW_V(\langle P, V \rangle (x))$$

This includes:

- V's randomness
- Any messages that V receives

Zero-Knowledge Proof

A proof $\langle P, V \rangle (x)$ for a language L is *zero-knowledge* if

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle(x)$$

Define V's view of this interaction by:

$$VIEW_V(\langle P, V \rangle(x))$$

This includes:

- V's randomness
- Any messages that V receives

Zero-Knowledge Proof

A proof $\langle P, V \rangle(x)$ for a language L is *zero-knowledge* if

- For any (possibly malicious) poly-time verifier V^*

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle (x)$$

Define V's view of this interaction by:

$$VIEW_V(\langle P, V \rangle (x))$$

This includes:

- V's randomness
- Any messages that V receives

Zero-Knowledge Proof

A proof $\langle P, V \rangle (x)$ for a language L is *zero-knowledge* if

- For any (possibly malicious) poly-time verifier V^*
- There exists a poly-time *Simulator* S s.t.

$$\forall x \in L, \qquad VIEW_{V^*}(\langle P, V^* \rangle (x)) = S(x)$$

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

Graph Isomorphism

Input: $x = (G_0, G_1)$

Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

The Proof

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

The Proof

1. P chooses $b \leftarrow \{0, 1\}$ and a random permutation σ and sends $H = \sigma(G_b)$ to V

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

The Proof

1. P chooses $b \leftarrow \{0,1\}$ and a random permutation σ and sends $H = \sigma(G_b)$ to V
2. V chooses $b' \leftarrow \{0,1\}$ and sends it to P

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

The Proof

1. P chooses $b \leftarrow \{0,1\}$ and a random permutation σ and sends $H = \sigma(G_b)$ to V
2. V chooses $b' \leftarrow \{0,1\}$ and sends it to P
3. P sends V the permutation π' mapping $G_{b'}$ to H

$$\pi' = \begin{cases} \sigma & \text{if } b = b' \\ \sigma\pi^{-1} & \text{if } b = 0, b' = 1 \\ \sigma\pi & \text{if } b = 1, b' = 0 \end{cases}$$

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

The Proof

1. P chooses $b \leftarrow \{0,1\}$ and a random permutation σ and sends $H = \sigma(G_b)$ to V
2. V chooses $b' \leftarrow \{0,1\}$ and sends it to P
3. P sends V the permutation π' mapping $G_{b'}$ to H

$$\pi' = \begin{cases} \sigma & \text{if } b = b' \\ \sigma\pi^{-1} & \text{if } b = 0, b' = 1 \\ \sigma\pi & \text{if } b = 1, b' = 0 \end{cases}$$

4. V accepts iff $H = \pi'(G_{b'})$

Outline

Outline

Outline

1. Lecture 25 Review
2. Complexity Theory
   - $\mathcal{P}$
   - $\mathcal{NP}$
   - Poly-time Reductions and $\mathcal{NP}$-Completeness
   - Interactive Proofs
   - Zero-Knowledge Proofs
3. Computability
   - Turing Machines and Decidable Languages
   - Languages Recognized by TMs
   - Undecidable Languages
   - Proofs by Reduction
4. Automata and Languages

Outline

Outline

1. Lecture 25 Review
2. Complexity Theory
   - $\mathcal{P}$
   - $\mathcal{NP}$
   - Poly-time Reductions and $\mathcal{NP}$-Completeness
   - Interactive Proofs
   - Zero-Knowledge Proofs
3. Computability
   - Turing Machines and Decidable Languages
   - Languages Recognized by TMs
   - Undecidable Languages
   - Proofs by Reduction
4. Automata and Languages

Outline

Exam

Exam Details:

- Tuesday, May 7, 10:20-12:20
- In the classroom
- 2 sheets (back-and-front) of notes are allowed

See you all there!