P, NP, Reductions

Exposition by William Gasarch—U of MD

P and EXP

Definition

1. $\mathrm{P} = \mathrm{DTIME}(n^{O(1)})$.
2. $\mathrm{EXP} = \mathrm{DTIME}(2^{n^{O(1)}})$.
3. PF is the set of functions that are computable in poly time.

NP

Definition A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x \mid (\exists y)[|y| = p(|x|) \wedge (x, y) \in B]\}.$$

NP

Definition A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x \mid (\exists y)[|y| = p(|x|) \land (x, y) \in B]\}.$$

Intuition. Let $A \in \mathrm{NP}$.

- If $x \in A$ then there is a SHORT (poly in $|x|$) proof of this fact, namely y, such that x can be VERIFIED in poly time. So if I wanted to convince you that $x \in L$, I could give you y. You can verify $(x, y) \in B$ easily and be convinced.

NP

Definition A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x \mid (\exists y)[|y| = p(|x|) \wedge (x, y) \in B]\}.$$

Intuition. Let $A \in \mathrm{NP}$.

- If $x \in A$ then there is a SHORT (poly in $|x|$) proof of this fact, namely y, such that x can be VERIFIED in poly time. So if I wanted to convince you that $x \in L$, I could give you y. You can verify $(x, y) \in B$ easily and be convinced.
- If $x \notin A$ then there is NO proof that $x \in A$.

Examples of Sets in NP: SAT

$$\mathrm{SAT} = \{\phi : (\exists \vec{y})[\phi(\vec{y}) = T]\}$$

There is a satisfying assignment for boolean formula ϕ.

Examples of Sets in NP: SAT

$$\mathrm{SAT} = \{\phi : (\exists \vec{y})[\phi(\vec{y}) = T]\}$$

There is a satisfying assignment for boolean formula ϕ.

1. y is $\vec{y}$. Note that $|y| < |\phi|$.

Examples of Sets in NP: SAT

$$\mathrm{SAT} = \{\phi : (\exists \vec{y})[\phi(\vec{y}) = T]\}$$

There is a satisfying assignment for boolean formula ϕ.

1. y is $\vec{y}$. Note that $|y| < |\phi|$.
2. Formally $B = \{(\phi, \vec{y}) : \phi(\vec{y}) = T\}$.

Examples of Sets in NP: SAT

$$\mathrm{SAT} = \{\phi : (\exists \vec{y})[\phi(\vec{y}) = T]\}$$

There is a satisfying assignment for boolean formula ϕ.

1. y is $\vec{y}$. Note that $|y| < |\phi|$.
2. Formally $B = \{(\phi, \vec{y}) : \phi(\vec{y}) = T\}$.

Do we think SAT is in P?

Examples of Sets in NP: SAT

$$\mathrm{SAT} = \{\phi : (\exists\vec{y})[\phi(\vec{y}) = T]\}$$

There is a satisfying assignment for boolean formula ϕ.

1. y is $\vec{y}$. Note that $|y| < |\phi|$.
2. Formally $B = \{(\phi, \vec{y}) : \phi(\vec{y}) = T\}$.

Do we think SAT is in P? No—we will later see its NP-complete.

Examples of Sets in NP: SAT

$$\mathrm{SAT} = \{\phi : (\exists \vec{y})[\phi(\vec{y}) = T]\}$$

There is a satisfying assignment for boolean formula ϕ.

1. y is $\vec{y}$. Note that $|y| < |\phi|$.
2. Formally $B = \{(\phi, \vec{y}) : \phi(\vec{y}) = T\}$.

Do we think SAT is in P? No—we will later see its NP-complete.

Note SAT only asks if **there exists** Satisfying assignment.

Examples of Sets in NP: SAT

$$\mathrm{SAT} = \{\phi : (\exists \vec{y})[\phi(\vec{y}) = T]\}$$

There is a satisfying assignment for boolean formula ϕ.

1. y is $\vec{y}$. Note that $|y| < |\phi|$.
2. Formally $B = \{(\phi, \vec{y}) : \phi(\vec{y}) = T\}$.

Do we think SAT is in P? No—we will later see its NP-complete.

Note SAT only asks if **there exists** Satisfying assignment.
It is not asking to **find one**.

Examples of Sets in NP: Graph Coloring

$$3\mathrm{COL} = \{G : G \text{ is 3-colorable } \}$$

One can 3-color the vertices of G such that no edge connects two nodes of the same color.

Examples of Sets in NP: Graph Coloring

$$\mathrm{3COL} = \{G : G \text{ is 3-colorable } \}$$

One can 3-color the vertices of G such that no edge connects two nodes of the same color.

1. y is the 3-coloring . $|y| < |B|$.

Examples of Sets in NP: Graph Coloring

$$3\mathrm{COL} = \{G : G \text{ is 3-colorable } \}$$

One can 3-color the vertices of G such that no edge connects two nodes of the same color.

1. y is the 3-coloring . $|y| < |B|$.
2. $B = \{(G, \rho) : \rho \text{ is a 3-coloring of } G\}$

Examples of Sets in NP: Graph Coloring

$$\mathrm{3COL} = \{G : G \text{ is 3-colorable }\}$$

One can 3-color the vertices of G such that no edge connects two nodes of the same color.

1. y is the 3-coloring . $|y| < |B|$.
2. $B = \{(G, \rho) : \rho \text{ is a 3-coloring of } G\}$

Do we think 3COL is in P?

Examples of Sets in NP: Graph Coloring

$$3\mathrm{COL} = \{G : G \text{ is 3-colorable }\}$$

One can 3-color the vertices of G such that no edge connects two nodes of the same color.

1. y is the 3-coloring . $|y| < |B|$.
2. $B = \{(G, \rho) : \rho \text{ is a 3-coloring of } G\}$

Do we think 3COL is in P? No—we will later see its NP-complete.

Examples of Sets in NP: Graph Coloring

$$3\mathrm{COL} = \{G : G \text{ is 3-colorable } \}$$

One can 3-color the vertices of G such that no edge connects two nodes of the same color.

1. y is the 3-coloring . $|y| < |B|$.
2. $B = \{(G, \rho) : \rho \text{ is a 3-coloring of } G\}$

Do we think 3COL is in P? No—we will later see its NP-complete.

Note 3COL only asks if **there exists** a 3-coloring.

Examples of Sets in NP: Graph Coloring

$$3\mathrm{COL} = \{G : G \text{ is 3-colorable } \}$$

One can 3-color the vertices of G such that no edge connects two nodes of the same color.

1. y is the 3-coloring . $|y| < |B|$.
2. $B = \{(G, \rho) : \rho \text{ is a 3-coloring of } G\}$

Do we think 3COL is in P? No—we will later see its NP-complete.

Note 3COL only asks if **there exists** a 3-coloring.
It is not asking to **find one**.

Examples of Sets in NP: Graph Coloring

$$\mathrm{CLIQ} = \{(G, k) : G \text{ has a clique of size } k\}$$

A clique is a set of vertices that are all pairwise connected.

Examples of Sets in NP: Graph Coloring

$$\mathrm{CLIQ} = \{(G, k) : G \text{ has a clique of size } k\}$$

A clique is a set of vertices that are all pairwise connected.

1. y is the set of k vertices. $|y| < |G|$.

Examples of Sets in NP: Graph Coloring

$$\mathrm{CLIQ} = \{(G, k) : G \text{ has a clique of size } k\}$$

A clique is a set of vertices that are all pairwise connected.

1. y is the set of k vertices. $|y| < |G|$.
2. $B = \{(G, A) : A \text{ is a set of } k \text{ vertices that form a Clique}\}$

Examples of Sets in NP: Graph Coloring

$$\mathrm{CLIQ} = \{(G, k) : G \text{ has a clique of size } k\}$$

A clique is a set of vertices that are all pairwise connected.

1. y is the set of k vertices. $|y| < |G|$.
2. $B = \{(G, A) : A \text{ is a set of } k \text{ vertices that form a Clique}\}$

Do we think $\mathrm{CLIQ} \in P$?

Examples of Sets in NP: Graph Coloring

$$\mathrm{CLIQ} = \{(G, k) : G \text{ has a clique of size } k\}$$

A clique is a set of vertices that are all pairwise connected.

1. y is the set of k vertices. $|y| < |G|$.
2. $B = \{(G, A) : A \text{ is a set of } k \text{ vertices that form a Clique}\}$

Do we think $\mathrm{CLIQ} \in P$? No—we will later see its NP-complete.

Examples of Sets in NP: Graph Coloring

$$\mathrm{CLIQ} = \{(G, k) : G \text{ has a clique of size } k\}$$

A clique is a set of vertices that are all pairwise connected.

1. y is the set of k vertices. $|y| < |G|$.
2. $B = \{(G, A) : A \text{ is a set of } k \text{ vertices that form a Clique}\}$

Do we think $\mathrm{CLIQ} \in P$? No—we will later see its NP-complete.

Note CLIQ only asks if **there exists** a k-cliq.

Examples of Sets in NP: Graph Coloring

$$\mathrm{CLIQ} = \{(G, k) : G \text{ has a clique of size } k\}$$

A clique is a set of vertices that are all pairwise connected.

1. y is the set of k vertices. $|y| < |G|$.
2. $B = \{(G, A) : A \text{ is a set of } k \text{ vertices that form a Clique}\}$

Do we think $\mathrm{CLIQ} \in P$? No—we will later see its NP-complete.

Note CLIQ only asks if **there exists** a k-cliq.
It is not asking to **find one** or **find the size of the largest clique.**

Finding the Size of the Largest Clique

$\mathrm{NCLIQ}(G)$ is the size of largest clique. It's a **function**, not a **set**.

Finding the Size of the Largest Clique

$\mathrm{NCLIQ}(G)$ is the size of largest clique. It's a **function**, not a **set**.

We show $\mathrm{CLIQ} \in \mathrm{P}$ implies $\mathrm{NCLIQ} \in \mathrm{PF}$.

Finding the Size of the Largest Clique

$\mathrm{NCLIQ}(G)$ is the size of largest clique. It's a **function**, not a **set**.

We show $\mathrm{CLIQ} \in \mathrm{P}$ implies $\mathrm{NCLIQ} \in \mathrm{PF}$.

We know $1 \le \mathrm{NCLIQ}(G) \le n$.

Finding the Size of the Largest Clique

$\mathrm{NCLIQ}(G)$ is the size of largest clique. It's a **function**, not a **set**.

We show $\mathrm{CLIQ} \in \mathrm{P}$ implies $\mathrm{NCLIQ} \in \mathrm{PF}$.

We know $1 \leq \mathrm{NCLIQ}(G) \leq n$.

By asking CLIQ we do binary search to find k such that
$(G, k) \in \mathrm{CLIQ}$ and $(G, k+1) \notin \mathrm{CLIQ}$.
Hence $\mathrm{NCLIQ}(G) = k$.

Finding the Size of the Largest Clique

$\mathrm{NCLIQ}(G)$ is the size of largest clique. It's a **function**, not a **set**.

We show $\mathrm{CLIQ} \in \mathrm{P}$ implies $\mathrm{NCLIQ} \in \mathrm{PF}$.

We know $1 \leq \mathrm{NCLIQ}(G) \leq n$.

By asking CLIQ we do binary search to find k such that
$(G, k) \in \mathrm{CLIQ}$ and $(G, k+1) \notin \mathrm{CLIQ}$.
Hence $\mathrm{NCLIQ}(G) = k$.

This algorithm took $\log n$ queries to CLIQ.

Finding the Largest Clique

$\mathrm{FCLIQ}(G)$ returns largest clique. It's a **function**, not a **set**.

Finding the Largest Clique

$\mathrm{FCLIQ}(G)$ returns largest clique. It's a **function**, not a **set**.

We show $\mathrm{CLIQ} \in \mathrm{P}$ implies $\mathrm{FCLIQ} \in \mathrm{PF}$.

Finding the Largest Clique

$\mathrm{FCLIQ}(G)$ returns largest clique. It's a **function**, not a **set**.

We show $\mathrm{CLIQ} \in \mathrm{P}$ implies $\mathrm{FCLIQ} \in \mathrm{PF}$.

Algorithm that will, given (G, k), return a clique of size k OR say NO there isn't one.

Finding the Largest Clique

$\mathrm{FCLIQ}(G)$ returns largest clique. It's a **function**, not a **set**.

We show $\mathrm{CLIQ} \in \mathrm{P}$ implies $\mathrm{FCLIQ} \in \mathrm{PF}$.

Algorithm that will, given (G, k), return a clique of size k OR say NO there isn't one.
$\mathrm{HELPFCLIQ}$:

1. Input (G, k)
2. Reduce the problem as follows: Let v be a vertex. Let $G' = G - \{v\}$. Test $(G', k) \in \mathrm{CLIQ}$.
    - If YES then find $\mathrm{HELPFCLIQ}(G', k)$ since we don't need v.
    - If NO then find $A = \mathrm{HELPFCLIQ}(G', k-1)$ and return $A \cup \{v\}$ since we know we NEED v.

Finishing Up CLIQ and FCLIQ

FCLIQ:

1. Input G
2. Find $k = NCLIQ(G)$.
3. Call $\text{HELPFCLIQ}(G, k)$.

Other Set–Function Issues

In the problems we will look at, the SET version (e.g., CLIQ) can always be used to find the FUNCTION version (e.g., FCLIQ).

We will not discuss this anymore in class, though it may be on some HWs.

Examples of Sets in NP: HAM

$$\mathrm{HAM} = \{G : \ G \text{ has a Hamiltonian Cycle } \}$$

A cycle is Hamiltonian (HAM) if it visits every **vertex** once.

1. y is the cycle itself.
2. $B = \{(G, C) : C \text{ is a HAM cycle of } G\}$

Examples of Sets in NP: HAM

$$\mathrm{HAM} = \{G : \text{ } G \text{ has a Hamiltonian Cycle } \}$$

A cycle is Hamiltonian (HAM) if it visits every **vertex** once.

1. y is the cycle itself.
2. $B = \{(G, C) : C \text{ is a HAM cycle of } G\}$

Do we think HAM is in P?

Examples of Sets in NP: HAM

$$\mathrm{HAM} = \{G : \ G \text{ has a Hamiltonian Cycle } \}$$

A cycle is Hamiltonian (HAM) if it visits every **vertex** once.

1. y is the cycle itself.
2. $B = \{(G, C) : C \text{ is a HAM cycle of } G\}$

Do we think HAM is in P? NO—it is NP-complete

Examples of Sets in NP: HAM

$$\mathrm{HAM} = \{G : \ G \text{ has a Hamiltonian Cycle } \}$$

A cycle is Hamiltonian (HAM) if it visits every **vertex** once.

1. y is the cycle itself.
2. $B = \{(G, C) : C \text{ is a HAM cycle of } G\}$

Do we think HAM is in P? NO—it is NP-complete

Note HAM only asks if **there exists** a HAM cycle.

Examples of Sets in NP: HAM

$$\mathrm{HAM} = \{G : \ G \text{ has a Hamiltonian Cycle } \}$$

A cycle is Hamiltonian (HAM) if it visits every **vertex** once.

1. y is the cycle itself.
2. $B = \{(G, C) : C \text{ is a HAM cycle of } G\}$

Do we think HAM is in P? NO—it is NP-complete

Note HAM only asks if **there exists** a HAM cycle.
It is not asking to **find one**.

Examples of Sets in NP: EUL

$$\mathrm{EUL} = \{G : \ G \text{ has an Eulerian Cycle } \}$$

A cycle is Eulerian (EUL) if it visits every **edge** once.

1. y is the cycle itself.
2. $B = \{(G, C) : C \text{ is an EUL cycle of } G\}$

Examples of Sets in NP: EUL

$$\mathrm{EUL} = \{G : \ G \text{ has an Eulerian Cycle } \}$$

A cycle is Eulerian (EUL) if it visits every **edge** once.

1. y is the cycle itself.
2. $B = \{(G, C) : C \text{ is an EUL cycle of } G\}$

Do we think EUL is in P?

Examples of Sets in NP: EUL

$$\mathrm{EUL} = \{G : \ G \text{ has an Eulerian Cycle } \}$$

A cycle is Eulerian (EUL) if it visits every **edge** once.

1. y is the cycle itself.
2. $B = \{(G, C) : C \text{ is an EUL cycle of } G\}$

Do we think EUL is in P?
YES—known that G has an Euler Cycle iff every degree is even.

Examples of Sets in NP: EUL

$$\mathrm{EUL} = \{G : \ G \text{ has an Eulerian Cycle }\}$$

A cycle is Eulerian (EUL) if it visits every **edge** once.

1. y is the cycle itself.
2. $B = \{(G, C) : C \text{ is an EUL cycle of } G\}$

Do we think EUL is in P?
YES—known that G has an Euler Cycle iff every degree is even.

Note EUL only asks if **there exists** an EUL cycle.

Examples of Sets in NP: EUL

$$\mathrm{EUL} = \{G : \ G \text{ has an Eulerian Cycle } \}$$

A cycle is Eulerian (EUL) if it visits every **edge** once.

1. y is the cycle itself.
2. $B = \{(G, C) : C \text{ is an EUL cycle of } G\}$

Do we think EUL is in P?
YES—known that G has an Euler Cycle iff every degree is even.

Note EUL only asks if **there exists** an EUL cycle.
It is not asking to **find one**.

History: HAM and EUL

1736 Euler solves the Konigsberg bridge problem by proving, in modern terms,
A graph is EUL iff every vertex has even degree

History: HAM and EUL

1736 Euler solves the Konigsberg bridge problem by proving, in modern terms,
A graph is EUL iff every vertex has even degree

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

History: HAM and EUL

1736 Euler solves the Konigsberg bridge problem by proving, in modern terms,
A graph is EUL iff every vertex has even degree

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

Note Mathematicians wanted a **characterization of HAM graphs similar to the characterization of EUL graphs**.

History: HAM and EUL

1736 Euler solves the Konigsberg bridge problem by proving, in modern terms,
A graph is EUL iff every vertex has even degree

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

Note Mathematicians wanted a **characterization of HAM graphs similar to the characterization of EUL graphs**.
They didn't have the language of algorithms to state what they wanted more rigorously.

History: HAM and EUL

1736 Euler solves the Konigsberg bridge problem by proving, in modern terms,
A graph is EUL iff every vertex has even degree

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

Note Mathematicians wanted a **characterization of HAM graphs similar to the characterization of EUL graphs**.
They didn't have the language of algorithms to state what they wanted more rigorously.

The theory of NP-completeness enabled mathematicians to **state** what they wanted rigorously ($\mathrm{HAM} \in \mathrm{P}$) and also gave the basis for proving likely it cannot be done (since HAM is NP-Complete).

Examples of Sets in NP: ShortPath

$$\mathrm{SP} = \{(G, v_1, v_2, c) : \text{ there is a path } v_1 \to v_2 \text{ in } G \text{ of length} \leq c\}$$

1. y is the path itself.
2. $B = \{((G, v_1, v_2, c), p) :$
   p is a path in G from v_1 to v_2 with $\leq c$ edges$\}$

Examples of Sets in NP: ShortPath

$\mathrm{SP} = \{(G, v_1, v_2, c) :$ there is a path $v_1 \to v_2$ in G of length $\leq c\}$

1. y is the path itself.
2. $B = \{((G, v_1, v_2, c), p) :$
   p is a path in G from v_1 to v_2 with $\leq c$ edges$\}$

Do we think SP is in P?

Examples of Sets in NP: ShortPath

$$\mathrm{SP} = \{(G, v_1, v_2, c) : \text{ there is a path } v_1 \to v_2 \text{ in } G \text{ of length} \leq c\}$$

1. y is the path itself.
2. $B = \{((G, v_1, v_2, c), p) :$
   p is a path in G from v_1 to v_2 with $\leq c$ edges$\}$

Do we think SP is in P?
YES—Dijkstra's algorithm computes the shortest path.

Examples of Sets in NP: 4-square

$$\mathrm{FOURSQ} = \{n : n \text{ can be written as the sum of 4 squares}\}$$

(We allow 0: $2 \in \mathrm{FOURSQ}$ since $2 = 1^2 + 1^2 + 0^2 + 0^2$.)

Examples of Sets in NP: 4-square

$$\mathrm{FOURSQ} = \{n : n \text{ can be written as the sum of 4 squares}\}$$

(We allow 0: $2 \in \mathrm{FOURSQ}$ since $2 = 1^2 + 1^2 + 0^2 + 0^2$.)

1. The 4 numbs whose sqs add to n is witness. Clearly shorter than $|n|$. (Note $|n| \sim \lg_2(n)$).
2. $B = \{(n; w, x, y, z) : n = w^2 + x^2 + y^2 + z^2\}$

Length of Input n is in binary, so $|n| = \log_2(n)$.

Examples of Sets in NP: 4-square

$$\text{FOURSQ} = \{n : n \text{ can be written as the sum of 4 squares}\}$$

(We allow 0: $2 \in \text{FOURSQ}$ since $2 = 1^2 + 1^2 + 0^2 + 0^2$.)

1. The 4 numbs whose sqs add to n is witness. Clearly shorter than $|n|$. (Note $|n| \sim \lg_2(n)$).
2. $B = \{(n; w, x, y, z) : n = w^2 + x^2 + y^2 + z^2\}$

Length of Input n is in binary, so $|n| = \log_2(n)$.

Do we think FOURSQ is in P?

Examples of Sets in NP: 4-square

$$\mathrm{FOURSQ} = \{n : n \text{ can be written as the sum of 4 squares}\}$$

(We allow 0: $2 \in \mathrm{FOURSQ}$ since $2 = 1^2 + 1^2 + 0^2 + 0^2$.)

1. The 4 numbs whose sqs add to n is witness. Clearly shorter than $|n|$. (Note $|n| \sim \lg_2(n)$).
2. $B = \{(n; w, x, y, z) : n = w^2 + x^2 + y^2 + z^2\}$

Length of Input n is in binary, so $|n| = \log_2(n)$.

Do we think FOURSQ is in P?
YES: Thm: $(\forall n)(\exists w, x, y, z)[n = w^2 + x^2 + y^2 + z^2]$.

Examples of Sets in NP: 4-square

$$\mathrm{FOURSQ} = \{n : n \text{ can be written as the sum of 4 squares}\}$$

(We allow 0: $2 \in \mathrm{FOURSQ}$ since $2 = 1^2 + 1^2 + 0^2 + 0^2$.)

1. The 4 numbs whose sqs add to n is witness. Clearly shorter than $|n|$. (Note $|n| \sim \lg_2(n)$).
2. $B = \{(n; w, x, y, z) : n = w^2 + x^2 + y^2 + z^2\}$

Length of Input n is in binary, so $|n| = \log_2(n)$.

Do we think FOURSQ is in P?
YES: Thm: $(\forall n)(\exists w, x, y, z)[n = w^2 + x^2 + y^2 + z^2]$.

Note FOURSQ only asks if **there exists** those four numbers. And there always do. But FOURSQ does not ask to find them.

Examples of Sets in NP: 4-square

$$\mathrm{FOURSQ} = \{n : n \text{ can be written as the sum of 4 squares}\}$$

(We allow 0: $2 \in \mathrm{FOURSQ}$ since $2 = 1^2 + 1^2 + 0^2 + 0^2$.)

1. The 4 numbs whose sqs add to n is witness. Clearly shorter than $|n|$. (Note $|n| \sim \lg_2(n)$).
2. $B = \{(n; w, x, y, z) : n = w^2 + x^2 + y^2 + z^2\}$

Length of Input n is in binary, so $|n| = \log_2(n)$.

Do we think FOURSQ is in P?
YES: Thm: $(\forall n)(\exists w, x, y, z)[n = w^2 + x^2 + y^2 + z^2]$.

Note FOURSQ only asks if **there exists** those four numbers. And there always do. But FOURSQ does not ask to find them.
A polyalg to find them is known but difficult (paper on course website).

Points About The Examples

1. Being in NP **does not** mean a problem is hard. We discuss NP-completeness which probably means a problem is hard.

Points About The Examples

1. Being in NP **does not** mean a problem is hard. We discuss NP-completeness which probably means a problem is hard.
2. Being in NP **does not** mean a problem is easy.

Points About The Examples

1. Being in NP **does not** mean a problem is hard. We discuss NP-completeness which probably means a problem is hard.
2. Being in NP **does not** mean a problem is easy.
3. In problems involving numbers we represent the numbers in binary. Be careful: usually n is the length of the input, but if our problem involves numbers and the input is n, then $\lceil \log_2(n) \rceil$ is the length of the input.

Points About The Examples

1. Being in NP **does not** mean a problem is hard. We discuss NP-completeness which probably means a problem is hard.
2. Being in NP **does not** mean a problem is easy.
3. In problems involving numbers we represent the numbers in binary. Be careful: usually n is the length of the input, but if our problem involves numbers and the input is n, then $\lceil \log_2(n) \rceil$ is the length of the input.
4. NP is a set of **sets**. Hence I have comments like:

Points About The Examples

1. Being in NP **does not** mean a problem is hard. We discuss NP-completeness which probably means a problem is hard.
2. Being in NP **does not** mean a problem is easy.
3. In problems involving numbers we represent the numbers in binary. Be careful: usually n is the length of the input, but if our problem involves numbers and the input is n, then $\lceil \log_2(n) \rceil$ is the length of the input.
4. NP is a set of **sets**. Hence I have comments like:

   Note 3COL only asks if **there exists** a 3-coloring.

Points About The Examples

1. Being in NP **does not** mean a problem is hard. We discuss NP-completeness which probably means a problem is hard.
2. Being in NP **does not** mean a problem is easy.
3. In problems involving numbers we represent the numbers in binary. Be careful: usually n is the length of the input, but if our problem involves numbers and the input is n, then $\lceil \log_2(n) \rceil$ is the length of the input.
4. NP is a set of **sets**. Hence I have comments like:

   Note 3COL only asks if **there exists** a 3-coloring.
   It is not asking to **find one**.

Reductions

Definition Let X, Y be languages. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

We express this by writing $X \leq Y$.

Reductions

Definition Let X, Y be languages. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

We express this by writing $X \leq Y$.

Reductions are transitive.
Easy Lemma (on Final?) If $X \leq Y$ and $Y \in \mathrm{P}$ then $X \in \mathrm{P}$.

Reductions

Definition Let X, Y be languages. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

We express this by writing $X \leq Y$.

Reductions are transitive.
Easy Lemma (on Final?) If $X \leq Y$ and $Y \in \mathrm{P}$ then $X \in \mathrm{P}$.

Contrapositive If $X \leq Y$ and $X \notin \mathrm{P}$ then $Y \notin \mathrm{P}$.

Definition of NP-Complete

Definition A language Y is NP-complete

- $Y \in \mathrm{NP}$
- If $X \in \mathrm{NP}$ then $X \leq Y$.

Definition of NP-Complete

Definition A language Y is NP-complete

- $Y \in \mathrm{NP}$
- If $X \in \mathrm{NP}$ then $X \leq Y$.

Easy Lemma If Y is NP-complete and $Y \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

Definition of NP-Complete

Definition A language Y is NP-complete

- $Y \in \mathrm{NP}$
- If $X \in \mathrm{NP}$ then $X \leq Y$.

Easy Lemma If Y is NP-complete and $Y \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

Honesty When I first saw the definition of NP-completeness I thought (1) there are no NP-complete sets or (2) there are no natural NP-complete sets.

Definition of NP-Complete

Definition A language Y is NP-complete

- $Y \in \mathrm{NP}$
- If $X \in \mathrm{NP}$ then $X \leq Y$.

Easy Lemma If Y is NP-complete and $Y \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

Honesty When I first saw the definition of NP-completeness I thought (1) there are no NP-complete sets or (2) there are no natural NP-complete sets.

The condition:

$$\text{for EVERY } X \in \mathrm{NP},\ X \leq Y?$$

seemed very hard to meet.

An Unnatural NP-complete set

Theorem Define language Y via:

$$Y = \left\{ \langle M, x, 1^t \rangle \;\middle|\; \begin{array}{c} M \text{ is a non-deterministic T.M.} \\ \text{which accepts } x \text{ within } t \text{ steps} \end{array} \right\}.$$

An Unnatural NP-complete set

Theorem Define language Y via:

$$Y = \left\{ \langle M, x, 1^t \rangle \mid \begin{array}{c} M \text{ is a non-deterministic T.M.} \\ \text{which accepts } x \text{ within } t \text{ steps} \end{array} \right\}.$$

Then Y is NP-complete.

An Unnatural NP-complete set

Theorem Define language Y via:

$$Y = \left\{ \langle M, x, 1^t \rangle \mid \begin{array}{c} M \text{ is a non-deterministic T.M.} \\ \text{which accepts } x \text{ within } t \text{ steps} \end{array} \right\}.$$

Then Y is NP-complete.

An Unnatural NP-complete set

Theorem Define language Y via:

$$Y = \left\{ \langle M, x, 1^t \rangle \mid \begin{array}{c} M \text{ is a non-deterministic T.M.} \\ \text{which accepts } x \text{ within } t \text{ steps} \end{array} \right\}.$$

Then Y is NP-complete.

Not that interesting since Y is not a natural set.

Variants of SAT

We define several variants of SAT:

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable.

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.
3. k-SAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of exactly k literals.

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.
3. k-SAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of exactly k literals.
4. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of literals.

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.
3. k-SAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of exactly k literals.
4. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of literals.
5. k-DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of exactly k literals.

SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed:

CNF-SAT is NP-complete

SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed:

CNF-SAT is $\mathbf{NP}$-complete

Thoughts on this:

SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed:

CNF-SAT is NP-complete

Thoughts on this:

1. The proof is not hard, but it involves looking at actual TMs. We will prove it next lecture. SAT was the **first** NP-complete problem. You could not use some other problem.

SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed:

CNF-SAT is NP-complete

Thoughts on this:

1. The proof is not hard, but it involves looking at actual TMs. We will prove it next lecture. SAT was the **first** NP-complete problem. You could not use some other problem.
2. Once we have SAT is NP-complete we will NEVER use TMs again. To show Y NP-complete: (1) $Y \in NP$, (2) $\mathrm{SAT} \le Y$.

SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed:

CNF-SAT is NP-complete

Thoughts on this:

1. The proof is not hard, but it involves looking at actual TMs. We will prove it next lecture. SAT was the **first** NP-complete problem. You could not use some other problem.
2. Once we have SAT is NP-complete we will NEVER use TMs again. To show Y NP-complete: (1) $Y \in NP$, (2) $\mathrm{SAT} \le Y$.
3. Thousands of problems are NP-complete. If any are in P then they are all in P.

What Do Theorists Think of P vs NP?

What Do Theorists Think of P vs NP?

I have done three polls of what theorists think of P vs NP and other issues.

What Do Theorists Think of P vs NP?

I have done three polls of what theorists think of P vs NP and other issues.
First I'll poll you, then I'll show you what the polls said.

What Do Theorists Think of P vs NP?

I have done three polls of what theorists think of P vs NP and other issues.
First I'll poll you, then I'll show you what the polls said.
Poll of 452 students: Do you think P vs NP?

What Do Theorists Think of P vs NP?

I have done three polls of what theorists think of P vs NP and other issues.

First I'll poll you, then I'll show you what the polls said.

Poll of 452 students: Do you think P vs NP?

	$P \neq NP$	$P = NP$	Ind	DK	other
2002	61 (61%)	9 (9%)	4 (4%)	22 (22%)	7 (7%))
2012	126 (83%)	12 (9%)	5 (3%)	1 (0.66%)	8 (5.1%)
2019	109 (88%)	15 (12%)	0	0	0