Lecture 3: **P, NP** and beyond

Arijit Bishnu

01.02.2010

Outline

Outline

Computing a Function and Running Time

Definition (Computing a Function and Running Time)

Let $f : \{0,1\}^* \to \{0,1\}^*$ and let $T : \mathbb{N} \to \mathbb{N}$ be some function, and let M be a Turing Machine (TM). We say that M computes f in $T(n)$-time if for every $x \in \{0,1\}^*$, if M is initialized to the start configuration on input x, then after at most $T(|x|)$ steps it halts with $f(x)$ written on its output tape.
We say that M computes f if it computes f in $T(n)$ time for some function $T : \mathbb{N} \to \mathbb{N}$.

Time-Constructible Functions

Time-Constructible Functions

We say that a function $T : \mathbb{N} \to \mathbb{N}$ is time constructible if $T(n) \geq n$ and there is a TM M that computes the function $x \mapsto (T(|x|))_b$ where $(T(|x|))_b$ denotes the binary representation of the number $T(|x|)$.

Time-Constructible Functions

Time-Constructible Functions

We say that a function $T : \mathbb{N} \to \mathbb{N}$ is time constructible if $T(n) \geq n$ and there is a TM M that computes the function $x \mapsto (T(|x|))_b$ where $(T(|x|))_b$ denotes the binary representation of the number $T(|x|)$.

Examples

n, $n \log n$, n^5, 2^n

Time-Constructible Functions

Time-Constructible Functions

We say that a function $T : \mathbb{N} \to \mathbb{N}$ is time constructible if $T(n) \geq n$ and there is a TM M that computes the function $x \mapsto (T(|x|))_b$ where $(T(|x|))_b$ denotes the binary representation of the number $T(|x|)$.

Examples

n, $n \log n$, n^5, 2^n

Remark

Functions that are much larger than exponential in n are non-time constructible. As an example, $T : \mathbb{N} \to \mathbb{N}$ such that every function computable in time $T(n)$ can also be computed in the much shorter time $\log T(n)$.

Outline

Machines as Strings and UTM

- We can write the description of any TM on paper. Hence, we can encode it using strings over $\{0, 1\}$.

Machines as Strings and UTM

- We can write the description of any TM on paper. Hence, we can encode it using strings over $\{0, 1\}$.
- The action of a TM is determined by its transition function, δ.

Machines as Strings and UTM

- We can write the description of any TM on paper. Hence, we can encode it using strings over $\{0, 1\}$.
- The action of a TM is determined by its transition function, δ.
- So, list all inputs and outputs of δ and encode it as a string over $\{0, 1\}^*$.

Machines as Strings and UTM

- We can write the description of any TM on paper. Hence, we can encode it using strings over $\{0, 1\}$.
- The action of a TM is determined by its transition function, δ.
- So, list all inputs and outputs of δ and encode it as a string over $\{0, 1\}^*$.
- Our representation scheme satisfies the following
  - Every string in $\{0, 1\}^*$ represents some TM.
  - Every TM is represented by infinitely many strings.

Machines as Strings and UTM

- We can write the description of any TM on paper. Hence, we can encode it using strings over $\{0, 1\}$.
- The action of a TM is determined by its transition function, δ.
- So, list all inputs and outputs of δ and encode it as a string over $\{0, 1\}^*$.
- Our representation scheme satisfies the following
  - Every string in $\{0, 1\}^*$ represents some TM.
  - Every TM is represented by infinitely many strings.
- Some notations:
  - For a TM M, we use M_b to denote the binary string representation of M.
  - For a string α, M_α denotes the TM represented by α.

UTM

- a UTM can simulate the execution of every other TM M given M's description as an input.

UTM

- a UTM can simulate the execution of every other TM M given M's description as an input.
- The parameters of a UTM are fixed - alphabet size, number of states, and the number of tapes.

UTM

- a UTM can simulate the execution of every other TM M given M's description as an input.
- The parameters of a UTM are fixed - alphabet size, number of states, and the number of tapes.
- The UTM encodes any other TM M's states and transition table on its tapes, and then follows along the computation step by step.

Theorem on Efficient UTM

Theorem: Efficient UTM

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

Proof.

See book for details. □

Outline

Class P

Recall that

- we deal with decision problems or languages that are nothing but boolean functions.
- we identify a boolean function with the language $L_f = \{x | f(x) = 1\}$.

Class P

Recall that

- we deal with decision problems or languages that are nothing but boolean functions.
- we identify a boolean function with the language $L_f = \{x | f(x) = 1\}$.

Definition: The Class DTIME

Let $T : \mathbb{N} \to \mathbb{N}$ be some function. We let DTIME($T(n)$) be the set of all boolean functions that are computable in $d \cdot T(n)$-time for some constant $d > 0$.

Class P

Recall that

- we deal with decision problems or languages that are nothing but boolean functions.
- we identify a boolean function with the language $L_f = \{x | f(x) = 1\}$.

Definition: The Class DTIME

Let $T : \mathbb{N} \to \mathbb{N}$ be some function. We let DTIME($T(n)$) be the set of all boolean functions that are computable in $d \cdot T(n)$-time for some constant $d > 0$.

Definition: The Class P

$\mathrm{P} = \bigcup_{c \geq 1} \mathrm{DTIME}(n^c)$.

Outline

Polynomial Time Reducibility

Definition: Polynomial Time Reduction

We say that a language $A \subseteq \{0,1\}^*$ is polynomial-time (Karp) reducible to a language $B \subseteq \{0,1\}^*$, denoted by $A \leq_P B$, if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$.

Polynomial Time Reducibility

Definition: Polynomial Time Reduction

We say that a language $A \subseteq \{0,1\}^*$ is polynomial-time (Karp) reducible to a language $B \subseteq \{0,1\}^*$, denoted by $A \leq_P B$, if there is a polynomial-time computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$.

An Intuitive Notion

If $A \leq_P B$, then B is at least as hard as A. That means, if A cannot be solved in polynomial time, then B cannot be solved in polynomial time.

Polynomial Time Reducibility

Definition: Polynomial Time Reduction

We say that a language $A \subseteq \{0,1\}^*$ is polynomial-time (Karp) reducible to a language $B \subseteq \{0,1\}^*$, denoted by $A \leq_P B$, if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$.

An Intuitive Notion

If $A \leq_P B$, then B is at least as hard as A. That means, if A cannot be solved in polynomial time, then B cannot be solved in polynomial time.

Lemma

If $A \leq_P B$, and $B \leq_P C$, then $A \leq_P C$.

Polynomial Time Reducibility

Definition: Polynomial Time Reduction

We say that a language $A \subseteq \{0,1\}^*$ is polynomial-time (Karp) reducible to a language $B \subseteq \{0,1\}^*$, denoted by $A \leq_P B$, if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$.

An Intuitive Notion

If $A \leq_P B$, then B is at least as hard as A. That means, if A cannot be solved in polynomial time, then B cannot be solved in polynomial time.

Lemma

If $A \leq_P B$, and $B \leq_P C$, then $A \leq_P C$.

Proof

Left as an exercise.

An Example of a Polynomial Time Reduction

$$\text{SAT} \leq_P \text{CLIQUE}$$

- Given an instance of SAT with m clauses and n variables, we construct a graph $G = (V, E)$ where V is the set of all occurrences of the $2n$ literals and
  $E = \{(x_i, x_j) | x_i \text{ and } x_j \text{ are in two diff. clauses and } x_i \neq \overline{x_j}\}$.

An Example of a Polynomial Time Reduction

$$\text{SAT} \leq_P \text{CLIQUE}$$

- Given an instance of SAT with m clauses and n variables, we construct a graph $G = (V, E)$ where V is the set of all occurrences of the $2n$ literals and
  $E = \{(x_i, x_j) | x_i \text{ and } x_j \text{ are in two diff. clauses and } x_i \neq \overline{x_j}\}$.

Lemma

The SAT formula f is satisfiable if and only if G has a clique of size m.

Understanding Certifier

Definition: The Class NP

A language $L \subseteq \{0,1\}^*$ is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call u a certificate for x (w.r.t. language L and machine M).

Understanding Certifier

Definition: The Class NP

A language $L \subseteq \{0,1\}^*$ is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call u a certificate for x (w.r.t. language L and machine M).

Examples

Traveling Salesperson, Subset sum, Integer Programming, Linear Programming, Graph Isomorphism, etc.

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

- M is a polynomial time algorithm (alternately, TM) that takes two input arguments x, u.

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

- M is a polynomial time algorithm (alternately, TM) that takes two input arguments x, u.
- There is a polynomial function p so that for every string x, we have $x \in L$ if and only if $\exists$ a string u such that $|u| \leq p(|x|)$ and $M(x, u) = 1$.

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

- M is a polynomial time algorithm (alternately, TM) that takes two input arguments x, u.
- There is a polynomial function p so that for every string x, we have $x \in L$ if and only if $\exists$ a string u such that $|u| \leq p(|x|)$ and $M(x, u) = 1$.

A Managerial View of M, an efficient certifier

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

- M is a polynomial time algorithm (alternately, TM) that takes two input arguments x, u.
- There is a polynomial function p so that for every string x, we have $x \in L$ if and only if $\exists$ a string u such that $|u| \leq p(|x|)$ and $M(x, u) = 1$.

A Managerial View of M, an efficient certifier

- It will not try to decide whether $x \in L$ on its own.

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

- M is a polynomial time algorithm (alternately, TM) that takes two input arguments x, u.
- There is a polynomial function p so that for every string x, we have $x \in L$ if and only if $\exists$ a string u such that $|u| \leq p(|x|)$ and $M(x, u) = 1$.

A Managerial View of M, an efficient certifier

- It will not try to decide whether $x \in L$ on its own.
- It will rather try to efficiently evaluate proposed "proofs" u that $x \in L$ - provided they are not too long.

A Brute Force Algorithm

M's use in solving L

- On an input x, try all strings u, s.t. $|u| \leq p(|x|)$, and see if $M(x, u) = 1$ for any of these strings.

A Brute Force Algorithm

M's use in solving L

- On an input x, try all strings u, s.t. $|u| \leq p(|x|)$, and see if $M(x, u) = 1$ for any of these strings.
- Existence of M does not provide an efficient solver for L.

A Brute Force Algorithm

M's use in solving L

- On an input x, try all strings u, s.t. $|u| \leq p(|x|)$, and see if $M(x, u) = 1$ for any of these strings.
- Existence of M does not provide an efficient solver for L.
- It is upto us to find a string u that will make $M(x, u) = 1$, and there are exponentially many possibilities for u.

Relation between Complexity Classes Learnt till now

Lemma

$P \subseteq NP \subseteq \bigcup_{c>1} DTIME(2^{n^c})$

Relation between Complexity Classes Learnt till now

Lemma

$\mathrm{P} \subseteq \mathrm{NP} \subseteq \bigcup_{c>1} \mathrm{DTIME}(2^{n^c})$

Proof

The first one is trivial. Left as an exercise.
For the other one, if $L \in \mathrm{NP}$, and M and p are as in the definition, then we can decide L in time $2^{O(p(n))}$ by enumerating all possible u and using M to check whether u is a valid certificate for input x. The machine accepts iff such a u is ever found. Since, $p(n) = O(n^c)$ for some $c > 1$, then the machine runs in $2^{O(n^c)}$ time.

An alternate definition using NDTM

We say that an NDTM N runs in $T(n)$ time if for every input $x \in \{0,1\}^*$ and every sequence of non-deterministic choices, N reaches either the halting state or q_{acc} within $T(|x|)$ steps.

Definition: The Class NTIME

For every function $T : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$, we say that $L \in \text{NTIME}(T(n))$ if there is a constant $c > 0$ and a $cT(n)$-time NDTM N such that for every $x \in \{0,1\}^*$, $x \in L \iff M(x) = 1$.

An alternate definition using NDTM

We say that an NDTM N runs in $T(n)$ time if for every input $x \in \{0,1\}^*$ and every sequence of non-deterministic choices, N reaches either the halting state or q_{acc} within $T(|x|)$ steps.

Definition: The Class NTIME

For every function $T : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$, we say that $L \in \text{NTIME}(T(n))$ if there is a constant $c > 0$ and a $cT(n)$-time NDTM N such that for every $x \in \{0,1\}^*$, $x \in L \iff M(x) = 1$.

The following theorem shows the equivalence between the two definitions.

An alternate definition using NDTM

We say that an NDTM N runs in $T(n)$ time if for every input $x \in \{0,1\}^*$ and every sequence of non-deterministic choices, N reaches either the halting state or q_{acc} within $T(|x|)$ steps.

Definition: The Class NTIME

For every function $T : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$, we say that $L \in \text{NTIME}(T(n))$ if there is a constant $c > 0$ and a $cT(n)$-time NDTM N such that for every $x \in \{0,1\}^*$, $x \in L \iff M(x) = 1$.

The following theorem shows the equivalence between the two definitions.

Theorem: The Class NP

$\text{NP} = \bigcup_{c \in \mathbb{N}} \text{NTIME}(n^c)$.

Proof

Left as an exercise. The idea is that the sequence of non-deterministic choices made by an accepting computation of an NDTM can be viewed as a certificate that the input is in the language, and vice versa.

NP Hardness and NP Completeness

Definition: NP Hardness

B is NP-hard if $A \leq_P B$ for every $A \in \text{NP}$.

NP Hardness and NP Completeness

Definition: NP Hardness

B is NP-hard if $A \leq_P B$ for every $A \in \text{NP}$.

Lemma

If a language A is NP-hard and $A \in \text{P}$ then $\text{P} = \text{NP}$.

NP Hardness and NP Completeness

Definition: NP Hardness

B is NP-hard if $A \leq_P B$ for every $A \in \text{NP}$.

Lemma

If a language A is NP-hard and $A \in \text{P}$ then $\text{P} = \text{NP}$.

Definition: NP Completeness

B is NP-complete if B is NP-hard and $B \in \text{NP}$.

NP Hardness and NP Completeness

Definition: NP Hardness

B is NP-hard if $A \leq_P B$ for every $A \in \text{NP}$.

Lemma

If a language A is NP-hard and $A \in \text{P}$ then $\text{P} = \text{NP}$.

Definition: NP Completeness

B is NP-complete if B is NP-hard and $B \in \text{NP}$.

Lemma

If a language A is NP-complete then $A \in \text{P}$ iff $\text{P} = \text{NP}$.

The Significance of Cook-Levin Theorem

How to prove NP-completeness?

- Show the problem $A \in \text{NP}$.

The Significance of Cook-Levin Theorem

How to prove NP-completeness?

- Show the problem $A \in \text{NP}$.
- For any problem B that is NP-hard, show $B \leq_P A$.

The Significance of Cook-Levin Theorem

How to prove NP-completeness?

- Show the problem $A \in \text{NP}$.
- For any problem B that is NP-hard, show $B \leq_P A$.
- What about the first problem that was proved to be NP-complete? That is the historical significance of Cook-Levin Theorem.

The Significance of Cook-Levin Theorem

How to prove NP-completeness?

- Show the problem $A \in \text{NP}$.
- For any problem B that is NP-hard, show $B \leq_P A$.
- What about the first problem that was proved to be NP-complete? That is the historical significance of Cook-Levin Theorem.

Cook-Levin Theorem

Let SAT be the language of all satisfiable CNF formulae. SAT is NP-complete.

The Significance of Cook-Levin Theorem

How to prove NP-completeness?

- Show the problem $A \in \text{NP}$.
- For any problem B that is NP-hard, show $B \leq_P A$.
- What about the first problem that was proved to be NP-complete? That is the historical significance of Cook-Levin Theorem.

Cook-Levin Theorem

Let SAT be the language of all satisfiable CNF formulae. SAT is NP-complete.

Proof

We will do it next day.

3SAT is NP-complete

3SAT be the language of all satisfiable 3CNF formulae. 3SAT $\in$ NP.

Lemma

SAT $\leq_P$ 3SAT

Proof

- Map a CNF formula ϕ into a 3CNF formula ψ such that ψ is satisfiable iff ϕ is.

3SAT is NP-complete

3SAT be the language of all satisfiable 3CNF formulae. 3SAT $\in$ NP.

Lemma

SAT $\leq_P$ 3SAT

Proof

- Map a CNF formula ϕ into a 3CNF formula ψ such that ψ is satisfiable iff ϕ is.
- Any clause C of size $k > 3$ can be changed to an equivalent pair of clauses C_1 of size $k-1$ and C_2 of size 3 by using an additional auxiliary variable.

3SAT is NP-complete

3SAT be the language of all satisfiable 3CNF formulae. 3SAT $\in$ NP.

Lemma

SAT $\leq_P$ 3SAT

Proof

- Map a CNF formula ϕ into a 3CNF formula ψ such that ψ is satisfiable iff ϕ is.
- Any clause C of size $k > 3$ can be changed to an equivalent pair of clauses C_1 of size $k-1$ and C_2 of size 3 by using an additional auxiliary variable.
- Say $C = \overline{x_1} \vee x_2 \vee x_3 \vee \overline{x_4}$. Let $C_1 = \overline{x_1} \vee x_2 \vee z$ and $C_2 = x_3 \vee \overline{x_4} \vee \overline{z}$. Clearly, if C is true, then there is an assignment to z that satisfies both C_1 and C_2 and vice versa.

INTEGER PROGRAMMING (IPROG) is NP-complete

For a set of linear inequalities with rational coefficients over variables $x_1, x_2, \ldots, x_n$ is there an assignment of integer numbers in $\{0, 1, \ldots\}$ to $x_1, x_2, \ldots, x_n$ that satisfies it. IPROG $\in$ NP

Lemma

SAT $\leq_P$ IPROG

INTEGER PROGRAMMING (IPROG) is NP-complete

For a set of linear inequalities with rational coefficients over variables $x_1, x_2, \ldots, x_n$ is there an assignment of integer numbers in $\{0, 1, \ldots\}$ to $x_1, x_2, \ldots, x_n$ that satisfies it. IPROG $\in$ NP

Lemma

SAT $\leq_P$ IPROG

Proof

INTEGER PROGRAMMING (IPROG) is NP-complete

For a set of linear inequalities with rational coefficients over variables $x_1, x_2, \ldots, x_n$ is there an assignment of integer numbers in $\{0, 1, \ldots\}$ to $x_1, x_2, \ldots, x_n$ that satisfies it. IPROG $\in$ NP

Lemma

SAT $\leq_P$ IPROG

Proof

- Add the constraints $0 \leq x_i \leq 1$ for every i to ensure that the feasible assignments to the variables are only 0 and 1.

INTEGER PROGRAMMING (IPROG) is NP-complete

For a set of linear inequalities with rational coefficients over variables $x_1, x_2, \ldots, x_n$ is there an assignment of integer numbers in $\{0, 1, \ldots\}$ to $x_1, x_2, \ldots, x_n$ that satisfies it. IPROG $\in$ NP

Lemma

SAT $\leq_P$ IPROG

Proof

- Add the constraints $0 \leq x_i \leq 1$ for every i to ensure that the feasible assignments to the variables are only 0 and 1.
- Now, express every clause as an inequality. As an example, the clause $\overline{x_1} \vee x_2 \vee \overline{x_3}$ can be expressed as $(1 - x_1) + x_2 + (1 - x_3) \geq 1$.