Theory of Computation – Lecture 13

MARCH 1, 2021

- The Class NP (Section 7.3)
- NP Completeness (Section 7.4)

Some material from slides by M. Sipser
Announcements

- Homework 3 due Wednesday March 17, 2021 at 11:59pm (NY Time)
A TM runs in time $t(n)$ if M always halts within $t(n)$ steps on all inputs of length $n$.

$\text{TIME}(t(n)) = \{ B \mid \text{some deterministic single-tape TM decides } B \text{ in time } O(t(n)) \}$
- E.g., $\text{TIME}(n^2)$ is the set of all languages that can be decided by a deterministic single-tape Turing machine in $O(n^2)$.

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine,

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

P is invariant for all reasonable deterministic models of computation.
- If language B is decidable on a deterministic multi-tape TM in $n$ steps, then B is decidable on a deterministic single-tape TM in $O(n^2)$.
How to prove a problem is in P?

- The Path problem is in P
  - \( \text{PATH} = \{(G,s,t) \mid G \text{ is a directed graph with a path from } s \text{ to } t \} \)
  - We proved this by constructing a deterministic single tape TM that decides \( \text{PATH} \) in polynomial time

- Be careful with selection of the input encoding
  - Make sure the length of the encoding is polynomial in the “size” of the input data structure

- We can also prove a problem is in P by reducing it (in polynomial time) to a problem that we already know is in P
  - More on this later
In a nondeterministic TM that is a decider, all computation paths halt on all inputs.

\[ \text{NTIME}(t(n)) = \{ B \mid \text{some nondeterministic single tape TM decides } B \text{ and runs in time } O(t(n)) \} \]

NP is the class of languages that are decidable in polynomial time on a nondeterministic single tape Turing machine

\[ NP = \bigcup_{k} \text{NTIME}(n^{k}) \]

NP is invariant for all reasonable nondeterministic models.
How to Prove a Problem is in NP

- The HAMPATH problem is in NP
  - \( \text{HAMPATH} = \{(G,s,t) \mid G \text{ is a directed graph with a path from } s \text{ to } t \text{ and the path goes through every node of } G \} \)
  - We proved this problem is in NP by constructing a nondeterministic single tape TM that decides \( \text{HAMPATH} \) in polynomial time
  - Idea: nondeterministically select a permutation of the \( n \) vertices and check whether it is a Hamiltonion path
    - The TM accepts if at least one computation accepts

- We used a similar approach to prove that \( \text{COMPOSITES} \) is in NP

- One important takeaway - for these problems, there are exponentially many possibly computation branches
  - Not trivial to convert nondeterministic solution to run on a deterministic TM in polynomial time
  - This does not mean the problems are not in \( P \), just that we cannot use the general procedure for constructing an equivalent deterministic TM to prove they are in \( P \)
How to Prove a Problem is in NP?

- Another key takeaway: for the proof, each computation path in the nondeterministic TM must terminate in a polynomial number of steps.
  - Thus, the entire TM has polynomial running time.

- We can consider each computation path as a processing a candidate solution \(c\) to the problem
  - A single computation path in the TM verifies whether \(c\) is a solution in polynomial time

- We can construct a deterministic single tape TM that performs an equivalent verification for such a candidate \(c\)
  - This TM is called a verifier
A **verifier** for a language $A$ is an algorithm (TM) $V$ where $A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \}$

- A verifier uses the extra information in $c$ to check that $w$ is a member of $A$.
- $c$ is called the **certificate** or the **proof**.

A **polynomial time verifier** runs in time polynomial in the length of $w$.

A language $A$ is **polynomially verifiable** if it has a polynomial time verifier.
NP and Polynomial Time Verifiers

- Definition 1: NP is the class of languages that are decidable by a nondeterministic single tape TM in polynomial time.

- Definition 2: NP is the class of languages that have polynomial time verifiers.

These definitions are not two separate criteria for the class NP. They are equivalent.

- **Theorem:** A language has a polynomial time verifier if and only if it is decided by some nondeterministic polynomial time TM.
Theorem: A language has a polynomial time verifier if and only if it is decided by some nondeterministic polynomial time Turing machine.
Theorem: A language has a polynomial time verifier if and only if it is decided by some nondeterministic polynomial time Turing machine.
The Clique Problem

- A **clique** is a subgraph of an undirected graph in which every pair of nodes is connected by an edge.
- A $k$-clique is a clique that contains $k$ nodes.

- The **clique problem** is to determine whether a graph contains a clique of a specified size.
  - $CLIQUE = \{ \langle G, k \rangle | G$ is an undirected graph with a $k$-clique $\}$
- $\text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique} \}$
- **Theorem:** $\text{CLIQUE} \in \text{NP}$
P vs NP

- NP = class of languages that are solvable in polynomial time on a nondeterministic TM
  = class of languages that can be verified in polynomial time (on a deterministic TM)
  = class of languages for which membership can be verified quickly

- P = class of languages that are solvable in polynomial time on a deterministic TM
  = class of languages for which membership can be decided quickly

- As of yet, we are unable to prove the existence of a single language in NP that is not in P

- What we do know: $NP \subseteq EXPTIME = \bigcup_k TIME(2^{n^k})$
NP Completeness

SIPSER 7.4
In the early 1970s, Stephen Cook and Leonid Levin discovered certain problems in NP such that, if a polynomial time algorithm exists for any of these problems, then $P = NP$

- These problems are called **NP-complete** problems.

If a researcher wants to show that $P \neq NP$, then they may focus on a single NP-complete problem.

If one believes $P \neq NP$, and they prove a problem is NP-complete, they may not want to waste time looking for a polynomial time algorithm for it.
The Satisfiability Problem

- A **Boolean formula** $\phi$ has Boolean variables (True/False values) and the Boolean operations: And ($\land$), Or ($\lor$), and Not ($\lnot$).

- $\phi$ is **satisfiable** if $\phi$ evaluates to TRUE for some assignment to its variables.

$$SAT = \{\langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \}$$

**Theorem:** $SAT \in P$ if and only if $P = \text{NP}$
- SAT is NP-complete
Polynomial Time Reducibility

- A function $f: \Sigma^* \rightarrow \Sigma^*$ is a **polynomial time computable function** there exists a polynomial time Turing machine $M$ that, on every input $w$, halts with just $f(w)$.

- Language $A$ is **polynomial time mapping reducible** (also called **polynomially time reducible**) to language $B$ (written $A \leq_P B$) if there is a polynomial time computable function $f$ where $w \in A$ if and only if $f(w) \in B$.

  The function $f$ is called a **polynomial time reduction** of $A$ to $B$. 
Theorem: If $A \leq_B B$ and $B \in P$ then $A \in P$
Example Polynomial Time Reduction

- A Boolean formula $\phi$ is in **Conjunctive Normal Form** (CNF) if it has the form
  $$\phi = (x \lor \bar{y} \lor z) \land (\bar{x} \lor \bar{s} \lor z \lor u) \land \cdots \land (\bar{z} \lor \bar{u})$$

- A **3cnf-formula** is a cnf-formula where all clauses have exactly three literals.

- $3SAT = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula} \}$

- We will show $3SAT$ is polynomial time reducible to $CLIQUE$

  $$CLIQUE = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique} \}$$
$3SAT = \{(\phi) | \phi \text{ is a satisfiable 3cnf-formula}\}$  

$\text{CLIQUE} = \{(G, k) | G \text{ is an undirected graph with a } k\text{-clique}\}$

**Theorem:** $3SAT$ is polynomial time reducible to $\text{CLIQUE}$

$$(x_1 \lor x_1 \lor x_2) \land (x_1 \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2)$$
3SAT = \{ (\phi) | \phi \text{ is a satisfiable 3cnf-formula} \} \quad CLIQUE = \{ (G, k) | G \text{ is an undirected graph with a } k\text{-clique} \} \\

**Theorem:** 3SAT is polynomial time reducible to CLIQUE \\

\[(x_1 \lor x_1 \lor x_2) \land (x_1 \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2)\]
A language $B$ is **NP-complete** if it satisfies two conditions:

1. $B$ is in NP and
2. Every $A$ in NP is polynomial time reducible to $B$

**Theorem:** If $B$ is NP-complete and $B$ is in P, then $P = NP$

Proof follows directly from theorem: If $A \leq_P B$ and $B \in P$ then $A \in P$

**Theorem:** If $B$ is NP-complete and $B \leq_P C$ for $C$ in NP, then $C$ is NP-complete.
The Cook-Levin Theorem

- $SAT = \{\langle \phi \rangle | \phi$ is a satisfiable Boolean formula}\}

- **Theorem:** $SAT$ is NP-complete

- Will prove this in the next lecture.
  - For now, just assume it is true.

- We can show a problem $C$ is NP-complete by showing $SAT \leq_P C$
$3SAT = \{\langle \phi \rangle | \phi \text{ is a satisfiable 3cnf-formula} \}$ \hspace{1cm} $SAT = \{\langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \}$

**Theorem:** $3SAT$ is NP-complete.