NP-Completeness

Jessica Guo and Audrey Wei
Mentor: Kerri Lu

MIT PRIMES Circle

May 27, 2022

Introduction

- Time complexity in computational complexity theory: measures running time of algorithms
- NP-complete problems: hardest problems that can be verified quickly
- P versus NP problem: can any problem whose solution can be verified in polynomial time also be decided in polynomial time?

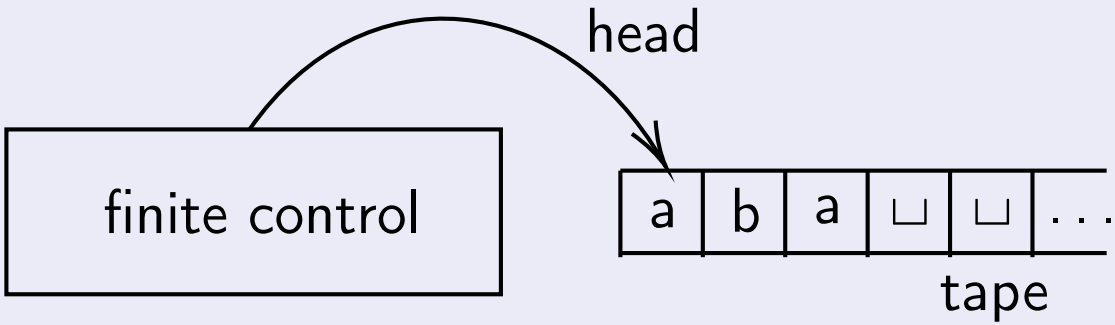
Turing Machines

A Turing machine (TM) is an idealized computer used to simulate an algorithm.

Turing Machines

A Turing machine (TM) is an idealized computer used to simulate an algorithm.

Turing Machine



Turing Machines

A Turing machine halts when it enters an "accept" or "reject" state. If the TM does not halt, it will loop infinitely instead.

The language a TM recognizes is the set of input strings the TM accepts.

A TM decides a language if it recognizes the language and the TM always halts.

Turing Machines

Example

We describe a Turing machine M that recognizes $A = \{0^k 1^k | k \geqslant 0\}$ as follows:

$M =$ "On input string w:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat step 3 if both 0s and 1s remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off or if 1s still remain after all of the 0s are crossed off, reject. Otherwise, if no 0s or 1s remain on the tape, accept."

Time Complexity

Definition of Running Time

Suppose that a halting deterministic TM uses at most $f(n)$ steps on any input of length n. We call $f(n)$ the running time of our TM.

Time Complexity

Definition of Running Time

Suppose that a halting deterministic TM uses at most $f(n)$ steps on any input of length n. We call $f(n)$ the running time of our TM.

Example

- Step 1: M scans the tape: n steps, the head moves back to the left end: n steps. Total: $2n$ or $O(n)$ steps.
- Step 2 and 3: M scans the tape: n steps, at most $n/2$ times (M crosses off 2 symbols each scan). Total: $(n/2)O(n) = O(n^2)$ steps.
- Step 4: M scans the tape: n or $O(n)$ steps.

Total running time of M is $O(n) + O(n^2) + O(n) = O(n^2)$.

Polynomial Time

Definition of TIME

Let $t(n)$ be a function which takes in natural numbers and outputs nonnegative real numbers. The time complexity class $\mathrm{TIME}(t(n))$ is the collection of languages that are decidable by an $O(t(n))$ TM.

Polynomial Time

Definition of TIME

Let $t(n)$ be a function which takes in natural numbers and outputs nonnegative real numbers. The time complexity class $\mathrm{TIME}(t(n))$ is the collection of languages that are decidable by an $O(t(n))$ TM.

Definition of P

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

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

Polynomial Time

Example: $\mathrm{PATH} \in \mathrm{P}$

$\mathrm{PATH} =$
$\{\langle G, s, t\rangle \mid G$ is a directed graph with a directed path from s to t$\}$.
M = "On input $\langle G, s, t\rangle$ where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the next step until no additional nodes are marked:
3. Scan all of the edges of G. If an edge (x, y) where x is a marked node and y is an unmarked node, mark node y.
4. If t is marked, accept. Otherwise reject."

Non-Deterministic Polynomial Time

P=NP problem still unsolved.

Non-Deterministic Polynomial Time

P=NP problem still unsolved.

Polynomial Verifiablility

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

$$A = \{w | \text{N accepts } \langle w, c \rangle \text{ for some string } c\}.$$

A language A is a polynomially verifiable if it has a polynomial time verifier.

Non-Deterministic Polynomial Time

P=NP problem still unsolved.

Polynomial Verifiablility

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

$$A = \{w | \text{N accepts } \langle w, c \rangle \text{ for some string } c\}.$$

A language A is a polynomially verifiable if it has a polynomial time verifier.

Nondeterministic Turing Machine

A nondeterministic TM (NTM) has the additional ability of "guessing," such that there are multiple possibilities for the next configuration on different "branches" of the machine's computation. A NTM accepts iff at least one of its branches accepts.

Non-Deterministic Polynomial Time

Definition of NTIME

$\text{NTIME}(t(n)) = \{L | L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic TM}\}$.

Non-Deterministic Polynomial Time

Definition of NTIME

NTIME($t(n)$) = $\{L | L$ is a language decided by an $O(t(n))$ time nondeterministic TM$\}$.

Definition of NP

NP is the class of languages decided by polynomial time NTMs. The mathematical notation of NP is

$$NP = \bigcup_k NTIME(n^k).$$

Non-Deterministic Polynomial Time

Theorem

A language is in NP iff it is polynomially verifiable.

Example $SAT \in NP$

Definition of Boolean Formula and Satisfiability

A Boolean formula ϕ is an expression of *Boolean variables* (1s and 0s) manipulated by *Boolean operations* AND($\wedge$), OR($\vee$), and NOT($\neg$). ϕ is satisfiable if ϕ evaluates to 1 for some assignment of variables.

Example $SAT \in NP$

Definition of Boolean Formula and Satisfiability

A Boolean formula ϕ is an expression of *Boolean variables* (1s and 0s) manipulated by *Boolean operations* AND($\wedge$), OR($\vee$), and NOT($\neg$). ϕ is satisfiable if ϕ evaluates to 1 for some assignment of variables.

Example

Let $\phi = x \wedge (\overline{y} \vee z)$. The assignment $x = 1, y = 1, z = 1$ makes ϕ evaluate to 1, so ϕ is satisfiable.

Example $SAT \in NP$

Definition of Boolean Formula and Satisfiability

A Boolean formula ϕ is an expression of *Boolean variables* (1s and 0s) manipulated by *Boolean operations* AND($\wedge$), OR($\vee$), and NOT($\neg$). ϕ is satisfiable if ϕ evaluates to 1 for some assignment of variables.

Example

Let $\phi = x \wedge (\overline{y} \vee z)$. The assignment $x = 1, y = 1, z = 1$ makes ϕ evaluate to 1, so ϕ is satisfiable.

SAT (satisfiability problem) tests if ϕ is satisfiable.

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

Example $SAT \in NP$

SAT is in NP because given a satisfying assignment (which is the certificate), a verifier plugs it into ϕ and checks if it satisfies ϕ.

A NTM can also guess an assignment to the boolean formula and accept if the assignment satisfies the formula.

NP-Completeness

NP-complete problems: problems in NP whose complexity is linked to complexity of all problems in NP.

NP-Completeness

NP-complete problems: problems in NP whose complexity is linked to complexity of all problems in NP.

Definition of Polynomial Time Reducible

Language A is polynomial time reducible to language L ($A \leqslant_p L$) if there exists a polynomial time computable function f such that for any input w,

$$w \in A \Leftrightarrow f(w) \in L.$$

NP-Completeness

NP-complete problems: problems in NP whose complexity is linked to complexity of all problems in NP.

Definition of Polynomial Time Reducible

Language A is polynomial time reducible to language L ($A \leqslant_p L$) if there exists a polynomial time computable function f such that for any input w,

$$w \in A \Leftrightarrow f(w) \in L.$$

Definition of NP-complete

Language L is NP-complete if

1. $L \in$ NP,
2. for every $A \in$ NP, $A \leqslant_p L$.

NP-Completeness

Cook-Levin Theorem

The Cook-Levin Theorem states that SAT is NP-complete.

NP-Completeness

Cook-Levin Theorem

The Cook-Levin Theorem states that SAT is NP-complete.

To prove some language L is NP-complete, we show $\mathrm{L} \in \mathrm{NP}$ and $A \leqslant_{\mathrm{p}} \mathrm{L}$ where A is known to be NP-complete.

- By proving that SAT is NP-complete, we can use SAT as our language A to solve other NP-complete problems.

NP-Completeness

Cook-Levin Theorem

The Cook-Levin Theorem states that SAT is NP-complete.

To prove some language L is NP-complete, we show $\mathrm{L} \in \mathrm{NP}$ and $A \leqslant_{\mathrm{p}} \mathrm{L}$ where A is known to be NP-complete.

- By proving that SAT is NP-complete, we can use SAT as our language A to solve other NP-complete problems.
- Examples:
  - HAMPATH: determines whether in a directed graph G, there is a directed path from node s to node t that goes through each node exactly once.
  - CLIQUE: determines whether an undirected graph G contains a k-clique (every two nodes are connected in subgraph of k nodes).

NP-Completeness

Corollary

$SAT \in P$ if and only if $P = NP$.

Finding any algorithm in P that solves an NP-complete problem (like SAT) would mean all problems in NP can be solved in P, which would prove $P = NP$.

Acknowledgements

We would like to thank our mentor, Kerri Lu, and the PRIMES Circle program coordinators, Marisa Gaetz and Mary Stelow, for their time, support, and guidance.

References

[1] Sisper, M. (2013). *Introduction to the Theory of Computation* (3rd ed.).