Monday

Recall Definition (Sipser 7.1): For M a deterministic decider, its **running time** is the function $f : \mathbb{N} \to \mathbb{N}$ given by

$$f(n) = \text{max number of steps } M \text{ takes before halting, over all inputs of length } n$$

Recall Definition (Sipser 7.7): For each function $t(n)$, the **time complexity class** $TIME(t(n))$, is defined by

$$TIME(t(n)) = \{L \mid L \text{ is decidable by a Turing machine with running time in } O(t(n))\}$$

Recall Definition (Sipser 7.12) : P is the class of languages that are decidable in polynomial time on a deterministic 1-tape Turing machine

$$P = \bigcup_k TIME(n^k)$$

Definition (Sipser 7.9): For N a nodeterministic decider. The **running time** of N is the function $f : \mathbb{N} \to \mathbb{N}$ given by

$$f(n) = \text{max number of steps } N \text{ takes on any branch before halting, over all inputs of length } n$$

Definition (Sipser 7.21): For each function $t(n)$, the **nondeterministic time complexity class** $NTIME(t(n))$, is defined by

$$NTIME(t(n)) = \{L \mid L \text{ is decidable by a nondeterministic Turing machine with running time in } O(t(n))\}$$

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

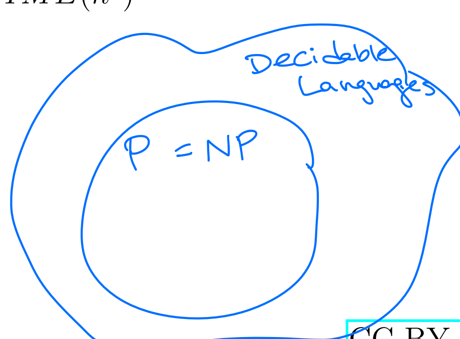
True or **False**: $TIME(n^2) \subseteq NTIME(n^2)$

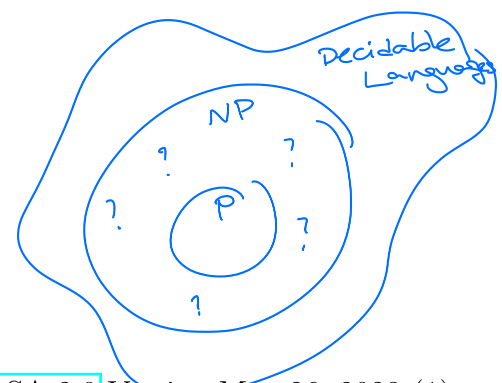
if have deterministic machine that is guaranteed to halt in $O(n^2)$ steps. it also witnesses that the language is in $NTIME(n^2)$.

True or **False**: $NTIME(n^2) \subseteq DTIME(n^2)$

???

Big Question





Examples in P

informally

Can't use nondeterminism; Can use multiple tapes; Often need to be "more clever" than naïve / brute force approach

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is digraph with } n \text{ nodes there is path from s to t}\}$$

Use breadth first search to show in P

$$RELPRIME = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime integers}\}$$

Use Euclidean Algorithm to show in P (long division over and over)

$$L(G) = \{w \mid w \text{ is generated by } G\}$$

(where G is a context-free grammar). Use dynamic programming to show in P.

Examples in NP

"Verifiable" i.e. NP, Can be decided by a nondeterministic TM in polynomial time, best known deterministic solution may be brute-force, solution can be verified by a deterministic TM in polynomial time.

$HAMPATH = \{\langle G, s, t\rangle \mid G$ is digraph with n nodes, there is path from s to t that goes through every node exa

$$VERTEX-COVER = \{\langle G, k\rangle \mid G \text{ is an undirected graph with } n \text{ nodes that has a } k\text{-node vertex cover}\}$$

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with } n \text{ nodes that has a } k\text{-clique}\}$$

$$SAT = \{\langle X\rangle \mid X \text{ is a satisfiable Boolean formula with } n \text{ variables}\}$$

any language in P is also in NP.

Every problem in NP is decidable with an exponential-time algorithm

Nondeterministic approach: guess a possible solution, verify that it works.

Brute-force (worst-case exponential time) approach: iterate over all possible solutions, for each one, check if it works.

Problems in P	**Problems in** NP
(Membership in any) regular language	Any problem in P
(Membership in any) context-free language	
A_{DFA}	SAT
E_{DFA}	$CLIQUE$
EQ_{DFA}	$VERTEX-COVER$
$PATH$	$HAMPATH$
$RELPRIME$	$\ldots$
$\ldots$	

Million-dollar question: Is $P = NP$?

One approach to trying to answer it is to look for *hardest* problems in NP and then (1) if we can show that there are efficient algorithms for them, then we can get efficient algorithms for all problems in NP so $P = NP$, or (2) these problems might be good candidates for showing that there are problems in NP for which there are no efficient algorithms.

Review: Week 10 Monday

Please complete the review quiz questions on Gradescope about complexity (P and NP)

Wednesday

Before: mapping reductions.

Definition (Sipser 7.29) Language A is **polynomial-time mapping reducible** to language B, written $A \leq_P B$, means there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that for every $x \in \Sigma^*$

$$x \in A \quad \text{iff} \quad f(x) \in B.$$

The function f is called the polynomial time reduction of A to B.

f witnesses the polynomial time reduction.

Theorem (Sipser 7.31): If $A \leq_P B$ and $B \in P$ then $A \in P$.

Reference result

if $A \leq_m B$ and B is decidable then A is decidable.

WTS

Proof:

Let A, B be languages.

Assume $A \leq_p B$, so there's witnessing function $f: \Sigma^* \to \Sigma^*$ computed by TM F, with $x \in A$ iff $f(x) \in B$. in polynomial time.

Assume $B \in P$, so there's witnessing decider M_B that halts in polynomial time and $L(M_B) = B$.

Define = "On input x

1. Compute (using F) $f(x)$. polynomial time
2. Run M_B on $f(x)$. polynomial time
3. If M_B accepts $f(x)$, accept x. polynomial time
4. If M_B rejects $f(x)$, reject x." polynomial time

polynomial time

Claim: this TM is a polynomial time decider for A.

Definition (Sipser 7.34; based in Stephen Cook and Leonid Levin's work in the 1970s): A language B is **NP-complete** means (1) B is in NP **and** (2) every language A in NP is polynomial time reducible to B.

$B \in NP$ — for each $A \in NP$ $(A \leq_p B)$.

Theorem (Sipser 7.35): If B is NP-complete and $B \in P$ then $P = NP$.

Proof: Assume B is NP-complete and $B \in P$.

We already know $P \subseteq NP$.
It remains to prove that $NP \subseteq P$.

Let A be an arbitrary set in NP.
By definition of B being NP-complete,
$A \leq_p B$.
By thm 7.31 and assumption that $B \in P$, $A \in P$ also, as required. ∎

Example of NP complete problem

3SAT: A literal is a Boolean variable (e.g. x) or a negated Boolean variable (e.g. $\bar{x}$). A Boolean formula is a **3cnf-formula** if it is a Boolean formula in conjunctive normal form (a conjunction of disjunctive clauses of literals) and each clause has three literals.

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

collection of strings encoding satisfiable proposition formulas in conjunctive normal form with 3 literals in each term.

Example strings in $3SAT$

$\langle (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee y \vee z) \rangle \in$ 3-SAT.

$(\vee \vee) \wedge (\vee \vee) \wedge (\vee \vee)$

e.g.

Example strings not in $3SAT$

$(x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee y \vee z)$

satisfiable, satisfied by truth assignment

$x = T$
$y = T$
$z = T$

Cook-Levin Theorem: $3SAT$ is NP-complete.

"first ever NP-complete" problem

Are there other NP-complete problems? To prove that X is NP-complete

- *From scratch*: prove X is in NP and that all NP problems are polynomial-time reducible to X.
- *Using reduction*: prove X is in NP and that $\underbrace{\text{a known-to-be } NP\text{-complete problem}}_{Y}$ is polynomial-time reducible to X.

Given A in NP, WTS $A \leq_p X$

If Y is known to be NP-complete, know that

$A \leq_p Y$

so as soon as we have $Y \leq_p X$

CLIQUE: A k**-clique** in an undirected graph is a maximally connected subgraph with k nodes.

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

subset of vertices that are pairwise adjacent.

Example strings in $CLIQUE$

Example strings not in $CLIQUE$

Theorem (Sipser 7.32):

$$3SAT \leq_P CLIQUE$$

Given a Boolean formula in conjunctive normal form with k clauses and three literals per clause, we will map it to a graph so that the graph has a clique if the original formula is satisfiable and the graph does not have a clique if the original formula is not satisfiable.

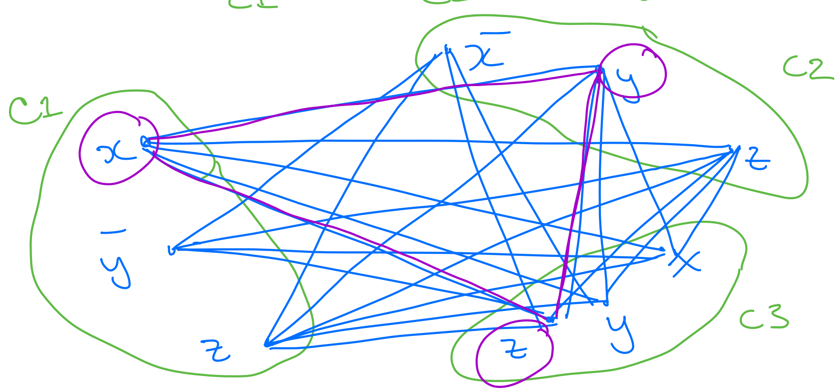
The graph has $3k$ vertices (one for each literal in each clause) and an edge between all vertices except

- vertices for two literals in the same clause
- vertices for literals that are negations of one another

Example: $\underbrace{(x \vee \bar{y} \vee \bar{z})}_{C1} \wedge \underbrace{(\bar{x} \vee y \vee z)}_{C2} \wedge \underbrace{(x \vee y \vee z)}_{C3}$ $K = 3$

Satisfiable!

sample satisfying assignment is $x=T$, $y=T$, $z=T$.



has 3-clique! {x from C1, y from C2, z from C3}

Review: Week 10 Wednesday

Please complete the review quiz questions on Gradescope about complexity (NP-completeness)

What can('t) computers do?

Friday

	Model of Computation	**Class of Languages**
Ch 1	**Deterministic finite automata**: formal definition, how to design for a given language, how to describe language of a machine? **Nondeterministic finite automata**: formal definition, how to design for a given language, how to describe language of a machine? **Regular expressions**: formal definition, how to design for a given language, how to describe language of expression? *Also*: converting between different models. RegExp: Concise DFA: implement	**Class of regular languages**: what are the closure properties of this class? which languages are not in the class? using **pumping lemma** to prove nonregularity. some example operations: complement, union, Kleene star, concatenation, ...
Ch 2	**Push-down automata**: formal definition, how to design for a given language, how to describe language of a machine? **Context-free grammars**: formal definition, how to design for a given language, how to describe language of a grammar? Ex context-free, nonregular languages: $\{0^n 1^n \mid n \geq 0\}$ $\{1^n 0^m \mid m \geq n \geq 0\}$ $\{ww^R \mid w \in \Sigma^*\}$	**Class of context-free languages**: what are the closure properties of this class? which languages are not in the class? recognizable by a PDA or generatable by a CFG.
Ch 7	Turing machines that always halt in polynomial time Nondeterministic Turing machines that always halt in polynomial time	P NP polynomial time reductions NP-complete.
Ch 3-4 Ch 5	**Deciders** (Turing machines that always halt): formal definition, how to design for a given language, how to describe language of a machine?	**Class of decidable languages**: what are the closure properties of this class? which languages are not in the class? using diagonalization and mapping reduction to show undecidability $A_{TM} = \{\langle M, w \rangle \mid M \text{ TM } w \text{ str}, w \in L(M)\}$
Ch 3 Ch 5	**Turing machines** formal definition, how to design for a given language, how to describe language of a machine?	**Class of recognizable languages**: what are the closure properties of this class? which languages are not in the class? using closure and mapping reduction to show unrecognizability

Closure properties: to prove that a class of languages is not closed under an operation, need an example language that is in the class but for which applying this operation results in a language no longer in the class

Given a language, prove it is regular

Strategy 1: construct DFA recognizing the language and prove it works.

Strategy 2: construct NFA recognizing the language and prove it works.

Strategy 3: construct regular expression recognizing the language and prove it works.

"Prove it works" means ...

Example: $L = \{w \in \{0,1\}^* \mid w \text{ has odd number of 1s or starts with } 0\}$ Extra practice

Using NFA

Using regular expressions

Example: Select all and only the options that result in a true statement: "To show a language A is not regular, we can..."

a. Show A is finite

b. Show there is a CFG generating A

c. Show A has no pumping length

d. Show A is undecidable

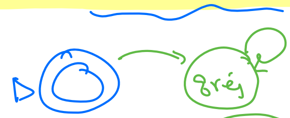
Extra practice

Example: What is the language generated by the CFG with rules

$$S \to aSb \mid bY \mid Ya$$
$$Y \to bY \mid Ya \mid \varepsilon$$

Extra practice

Example: Prove that the language $T = \{\langle M \rangle \mid M \text{ is a Turing machine and } L(M) \text{ is infinite}\}$ is undecidable.

Example string in T: $\langle$  $\rangle$

state diagram for a TM

To show a language is undecidable can find some set that is undecidable and that mapping reduces to it.

* $A_{TM} = \{\langle M, w\rangle \mid M \text{ TM } w \text{ string, } w \in L(M)\}$. *

$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ TM } w \text{ string } M \text{ halts on } w\}$.

$\overline{A_{TM}}$

$\overline{HALT_{TM}}$

reference undecidable languages

Goal: Prove that $A_{TM} \leq_m T = \{\langle M\rangle \mid M \text{ is TM } L(M) \text{ infinite}\}$.

We need a computable function $f : \Sigma^* \to \Sigma^*$

where $x \in A_{TM}$ iff $f(x) \in T$.

i.e. for each $x \in \Sigma^*$

if $x \neq \langle M, w\rangle$, M TM w str want $f(x) \notin T$.

if $x = \langle M, w\rangle$, M TM w str. want $f(x) \in T$

M accepts w

if $x = \langle M, w\rangle$ M TM w str. want $f(x) \notin T$.

M rejects w
or
M loops on w

use $L(\_) = \Sigma^*$
$L(\_) = \emptyset$
as proxy.

Define f as the function computed by the TM

$F =$ "On input x

1. If $x = \langle M, w \rangle$ for some M TM, w str. go to step 3.
2. If not, output $\langle \triangleright q_{rej} \rangle$
3. $x = \langle M, w \rangle$ for TM M, str w.
   Define
   M'_x = "On input y:
   1. Run M on w
   2. If accepts, accept.
   3. If rejects, reject."
4. Output $\langle M'_x \rangle$"

Claim: This TM computes function witnessing $A_{TM} \leq_m T$.

Pf of claim: Extra practice

Example: Prove that the class of decidable languages is closed under concatenation.

Extra practice

Review: Week 10 Friday

Please complete the review quiz questions on Gradescope giving feedback on the quarter. Have a great summer!