Monday May 30

No class in observance of Memorial Day holiday.

Wednesday June 1

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

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

For each function $t(n)$, the **time complexity class** $TIME(t(n))$, is defined by

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

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

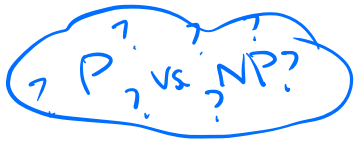
$$f(n) = \text{max number of steps } N \text{ takes on any branch before halting, over all inputs of length } n$$

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

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

$TIME(n^2) \subseteq P$
$NTIME(n^2) \subseteq NP$

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



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

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

DTIME emphasizing Turing machines deterministic

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.

Examples in P

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

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

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

efficient algorithms.

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.

graphs.

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

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

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

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

logic

3-SAT = {⟨X⟩ | X is satisfiable Boolean formula w/ n variables in 3-CNF}

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}	$VERTEX - COVER$
$PATH$	$HAMPATH$
$RELPRIME$	$\ldots$
$\ldots$	

Notice $NP \subseteq \{L \mid L \text{ is decidable}\}$

So $A_{TM} \notin NP$

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.

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 $x \in \Sigma^*$

There is a TM that, on input x, halts within polynomial time and outputs f(x).

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

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

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

Proof:

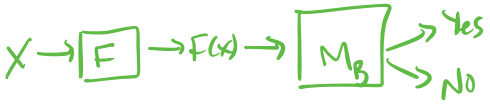
There is a TM that efficiently decides B

There is a TM that efficiently decides A.

WTS there is a TM that efficiently decides A

Given a TM, M_B, that efficiently decides B and given a TM F that efficiently computes witness function to $A \leq_p B$.

Define M_A = On input x (is it in A?)
1. Calculate F(x). Polynomial time
2. Run M_B on F(x)
if M_B accepts F(x), accept x.
if M_B rejects F(x), reject x"



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 is NP-hard"

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

a hardest problem in NP.

Proof:

WTS for every $A \in NP$, $A \in P$ too
Could use the previous result to translate efficient deciders for B to efficient deciders for A.

Notice: NP-complete problems are all decidable.

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.

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

Example strings in $3SAT$

Example strings not in $3SAT$

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

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 a known-to-be NP-complete problem is polynomial-time reducible to X.

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

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

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

There are lots of NP-complete problems an efficient alg for any one of them gives efficient algs for all of them.

Review: Week 10 Wednesday

Please complete the review quiz questions on Gradescope about complexity (P, NP, and NP-completeness.)

Friday June 3 What can(t) computers do?

Model of Computation	**Class of Languages**
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.	**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.
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?	**Class of context-free languages**: what are the closure properties of this class? which languages are not in the class?
Turing machines that always halt in polynomial time Nondeterministic Turing machines that always halt in polynomial time	P NP
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
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

Which example to start with?

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 ...

extra practice

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

Using NFA

extra practice

Using regular expressions

extra practice

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.

Goal: Show $A_{TM} \leq_m T$.

$A_{TM} = \{\langle M, w \rangle \mid M \text{ TM } w \text{ string } M \text{ accepts } w\}$.

By theorem in book (- - -), since A_{TM} is undecidable and we will show $A_{TM} \leq_m T$, can conclude T is undecidable.

Need TM that computes function so that for all strings $x \in \Sigma^*$

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

$\langle M, w \rangle$ $w \in L(M)$?

$\langle \tilde{M} \rangle$ $L(\tilde{M})$ is infinite. use as proxy Σ^* $\emptyset$

Build F = "On input x

1. If $x \neq \langle M, w \rangle$ for M TM, w string, output $\langle \triangleright$ (stuck) $\circlearrowleft \rangle$
2. Otherwise $x = \langle M, w \rangle$ for M TM, w string.
3. Define M'_x = "On input y
   1. Run M on w.
   2. If M accepts w, accept y.
   3. If M rejects w, reject y."
4. Output $\langle M'_x \rangle$."

Claim the function computed by F witnesses $A_{TM} \leq_m T$.

Proof: Since this function is computable, it remains to show that for all $x \in \Sigma^*$, $x \in A_{TM}$ iff $F(x) \in T$.

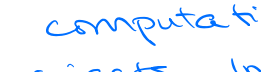
Let $x \in \Sigma^*$ be arbitrary.

Case (1) $x \in A_{TM}$. Then $x = \langle M, w \rangle$ for some TM M, string w, and $w \in L(M)$. Tracing the definition of F, on input x, the conditional in step 1 is false so the computation moves to steps 2 and 3, in which M'_x is defined.

Notice that $L(M'_x) = \Sigma^*$ because, on any string y, the computation of M'_x on y simulates M on w (in step 1), which (by our

case assumption) is accepting so M'_x accepts y in step 2. $F(x) = \langle M'_x \rangle$ and since $L(M'_x) = \Sigma^*$ is infinite, $F(x) \in T$, as required.

Case (2) $x \notin A_{TM}$. We consider two subcases

Case (2a) $x \neq \langle M, w \rangle$ for any TM M, string w. In this case, step 1 of computing $F(x)$ gives $F(x) = \langle$  $\rangle$. Notice that $L($  $) = \emptyset$ because the start state is the reject state so each computation halts immediately and rejects. In particular, $L($  $)$ is finite so $F(x) \notin T$, as required.

Case (2b) $x = \langle M, w \rangle$ for some TM M, string w and $w \notin L(M)$. Tracing the definition of F, on input x, the conditional in step 1 is false so the computation moves to steps 2 and 3, in which M'_x is defined. Notice that $L(M'_x) = \emptyset$ because, on any string y, the computation of M'_x on y simulates M on w (in step 1), which (by our case assumption) either rejects or loops. If M loops on w then M'_x loops in step 1 for any y so $L(M'_x) = \emptyset$. If M rejects w then M'_x rejects (any) y in step 3 so $L(M'_x) = \emptyset$. Notice that no matter what, in this case $L(M'_x) = \emptyset$ is a finite set and $F(x) = \langle M'_x \rangle$ so $F(x) \notin T$, as required.

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

(c) Consider the following construction of a Turing machine: $w \in L_1 \circ L_2$?

"On input w

1. Let $n = |w|$.
2. Create a two dimensional array of strings $s_{m,j}$ where $0 \leq m \leq n$ and $0 \leq j \leq 1$.
3. For each $0 \leq m \leq n$, initialize $s_{m,0}$ to be the prefix of w of length m and $s_{m,1}$ to be the suffix of w of length $n - m$. In other words, $w = s_{m,0}s_{m,1}$ and $|s_{m,0}| = m$, $|s_{m,1}| = n - m$. $s_{m,0}$ $s_{m,1}$
4. For $i = 1, 2, \ldots$ ε, w
5. For $k = 0, \ldots, i$ one char of w, rest of w
6. Run M_1 on $s_{\min(k,n),0}$ for (at most) i steps.
7. Run M_2 on $s_{\min(k,n),1}$ for (at most) i steps.
8. If both simulations in steps 6 and 7 accept, accept."

Can this construction be used to prove that the class of Turing-recognizable languages is closed under concatenation? Briefly justify your answer.

Solution: Yes. Given L_1 and L_2 recognizable by Turing machines M_1 and M_2, apply the above construction to obtain Turing machine M. We will show M recognizes $L_1 \circ L_2$ and hence, $L_1 \circ L_2$ is Turing-recognizable. $\forall w \in L_1 \circ L_2$, by definition $w = xy$ for some $x \in L_1$ and $y \in L_2$. Furthermore, suppose M_1 accepts x in a_1 steps and M_2 accepts y in a_2 steps. We want to show that w is accepted by M. When $i = max(a_1, a_2, |w|)$, then for the value $k = |x|$ we have $\min(k, n) = \min(k, |w|) = k$ since $k \leq i \leq |w|$. Thus, in line 6 M runs M_1 on $s_{\min(k,n),0} = s_{k,0} = s_{|x|,0} = x$ for $i \geq a_1$ steps. Similarly, in line 7 M runs M_2 on y $s_{\min(k,n),1} = s_{k,1} = s_{|x|,1} = y$ for $i \geq a_2$ steps. Both machines simulations run long enough to accept. Hence, M accepts w. Conversely, $\forall w \in L(M)$.

To prove that a class of languages is closed under concatenation, need to show is that for any languages, L_1, L_2, in this class their concatenation $L_1 \circ L_2$ is also in this class.

Then, there exists k and i such that $w = s_{k,0}s_{k,1}$, M_1 accepts $s_{k,0}$ in at most i steps, and M_2 accepts $s_{k,1}$ in at most i steps. This implies $w \in L_1 \circ L_2$. Therefore M recognizes $L_1 \circ L_2$.

To prove that the class of decidable languages is closed under concatenation let's start with arbitrary decidable languages L_1, L_2. Call the TMs that decide L_1 and L_2 D_1 and D_2 (respectively).

WTS $L_1 \circ L_2$ is also decidable.

We will do this by defining a TM D

and proving for all $x \in \Sigma^*$

if $x \in L_1 \circ L_2$, D accepts x.

if $x \notin L_1 \circ L_2$, D rejects x.

(Notice that this guarantees D decides $L_1 \circ L_2$)

Idea: Given x 

Consider all splits of $x = s_{m,0}\, s_{m,1}$

* Ask if $s_{m,0} \in L_1$ and if $s_{m,1} \in L_2$? *

Build TM

D = "On input x.

1. Let $n = |x|$
2. Initialize a 2D array of strings $s_{m,j}$ where $0 \le m \le n$ and $0 \le j \le 1$.
3. Define $s_{m,0}$ to be prefix of x of length m and $s_{m,1}$ to be suffix of x of length n-m for each m, $0 \le m \le n$.
4. For $m = 0$ to n
5. Run D_1 on $s_{m,0}$ and record result.
6. Run D_2 on $s_{m,1}$ and record result.
7. If both D_1 and D_2's computations accept, accept (and break/return).
8. If either D_1 and D_2's computations reject, go to next loop iteration
9. If loop finishes and haven't accepted, reject"

Now we prove that D witnesses that $L_1 \circ L_2$ is decidable.

i.e. D is decider and $L(D) = L_1 \circ L_2$.

We want to show, for all $x \in \Sigma^*$

if $x \in L_1 \circ L_2$, D accepts x.

if $x \notin L_1 \circ L_2$, D rejects x.

halt.

Let x be arbitrary string.

Case ① $x \in L_1 \circ L_2$. Then, by definition of setwise concatenation, there are strings y, z such that $x = yz$, $y \in L_1$, $z \in L_2$. We take y, z to be witnesses where y has shortest length among all possible witnesses. Tracing the computation of D on x, n is set to $|x|$ and in steps 2 and 3 we'll have $s_{|y|,0} = y$, $s_{|y|,n} = z$. In the for loop, for $n = 0, \ldots, |y|-1$, in steps 5 and 6 D_1 and D_2's computations will halt (because D_1 and D_2 are deciders) and at least one will reject (because we assumed y has shortest length among witnesses and $L(D_1) = L_1$, $L(D_2) = L_2$ so if both computations in these steps accepted we'd have a witness with a shorter prefix). So, the loop execution continues until $n = |y|$. At this point the computations in steps 5 and 6 both accept because $s_{m,0} = s_{|y|,0} = y \in L_1 = L(D_1)$ and $s_{m,1} = s_{|y|,1} = z \in L_2 = L(D_2)$. Thus, D accepts x in step 7, as required.

$L_1 = \{ w \mid w \text{ even length strings} \}$

$L_2 = \{ w \mid w \text{ starts with } 1 \}$

$\in L_1 \circ L_2$.

not even length

$\in L_1 \circ L_2$.

Case ② extra practice.

Review: Week 10 Friday

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