### 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

 $f(n) = \max$  number of steps M 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

 $f(n) = \max$  number of steps N 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**:  $TIME(n^2) \subseteq NTIME(n^2)$ 

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

#### 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.

#### 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.

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

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

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

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

#### 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	

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

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  $x \in \Sigma^*$ 

$$x \in A$$
 iff  $f(x) \in B$ .

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:

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.

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

Proof:



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

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.



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

Example strings in CLIQUE

Example strings not in CLIQUE

Theorem (Sipser 7.32):

$$3SAT <_{P} CLIQUE$$

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:  $(x \lor \bar{y} \lor \bar{z}) \land (\bar{x} \lor y \lor z) \land (x \lor y \lor z)$ 

Review: Week 10 Wednesday						
Please complete the review quiz questions on Gradescope about complexity $(NP\text{-completeness})$						

# Friday

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
<b>Deciders</b> (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
<b>Turing machines</b> 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

Given	a	language,	prove	it	ic	regui	ar
Given	а	language,	prove	ւլ	$\mathbf{IS}$	regu	aı

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

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

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

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



Example:	Prove t	that the	class of d	lecidable	language	s is closed	d under co	oncatenati	ion.	



## Review: Week 10 Friday

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