15-453

FORMAL LANGUAGES, AUTOMATA AND COMPUTABILITY
TIME COMPLEXITY AND POLYNOMIAL TIME; NON DETERMINISTIC TURING MACHINES AND NP

THURSDAY Mar 20
COMPLEXITY THEORY

Studies what can and can’t be computed under limited resources such as time, space, etc

Today: Time complexity
We measure time complexity by counting the elementary steps required for a machine to halt.

Consider the language \( A = \{ 0^k1^k \mid k \geq 0 \} \)

On input of length \( n \):

1. Scan across the tape and **reject** if the string is not of the form \( 0^i1^j \)

2. Repeat the following if both 0s and 1s remain on the tape:
   - Scan across the tape, crossing off a single 0 and a single 1

3. If 0s remain after all 1s have been crossed off, or vice-versa, **reject**. Otherwise **accept**.
MEASURING TIME COMPLEXITY

We measure time complexity by counting the elementary steps required for a machine to halt.

Consider the language \( A = \{ 0^k1^k \mid k \geq 0 \} \)

On input of length \( n \):

1. Scan across the tape and reject if the string is not of the form \( 0^i1^j \)

2. Repeat the following if both 0s and 1s remain on the tape:
   - Scan across the tape, crossing off a single 0 and a single 1

3. If 0s remain after all 1s have been crossed off, or vice-versa, reject. Otherwise accept.
Definition: Let M be a TM that halts on all inputs. The running time or time-complexity of M is the function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length $n$. 
ASYMPTOTIC ANALYSIS

5n^3 + 2n^2 + 22n + 6 = O(n^3)
Let $f$ and $g$ be two functions $f, g : \mathbb{N} \to \mathbb{R}^+$. We say that $f(n) = O(g(n))$ if there exist positive integers $c$ and $n_0$ so that for every integer $n \geq n_0$

$$f(n) \leq cg(n)$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an asymptotic upper bound for $f(n)$

$f$ asymptotically NO MORE THAN $g$
Let $f$ and $g$ be two functions $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$. We say that $f(n) = O(g(n))$ if there exist positive integers $c$ and $n_0$ so that for every integer $n \geq n_0$

$$f(n) \leq cg(n)$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an asymptotic upper bound for $f(n)$

$f$ asymptotically NO MORE THAN $g$

$$5n^3 + 2n^2 + 22n + 6 = O(n^3)$$

If $c = 6$ and $n_0 = 10$, then $5n^3 + 2n^2 + 22n + 6 \leq cn^3$
\[ 2n^{4.1} + 200283n^4 + 2 = O(n^{4.1}) \]

\[ 3n \log_2 n + 5n \log_2 \log_2 n = O(n \log_2 n) \]

\[ n \log_{10} n^{78} = O(n \log_{10} n) \]
\[ 2n^{4.1} + 200283n^4 + 2 = O(n^{4.1}) \]

\[ 3n\log_2 n + 5n \log_2 \log_2 n = O(n\log_2 n) \]

\[ n\log_{10} n^{78} = O(n\log_{10} n) \]

\[ \log_{10} n = \frac{\log_2 n}{\log_2 10} \]

\[ O(n\log_2 n) = O(n\log_{10} n) = O(n\log\log n) \]
Definition: \( \text{TIME}(t(n)) = \{ L \mid L \text{ is a language decided by a } O(t(n)) \text{ time Turing Machine} \} \)

\[
A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n^2)
\]
A = \{ 0^k 1^k \mid k \geq 0 \} \in \text{TIME}(n \log n)

Cross off every other 0 and every other 1. If the # of 0s and 1s left on the tape is odd, reject

00000000000001111111111111
x0x0x0x0x0x0xx1x1x1x1x1x1x
xxx0xxx0xxx0xxxx1xxx1xxx1x
xxxxxxx0xxxxxxxxxxxx1xxxxx
xxxxxxxxxxxxxxxxxxxxxxxxx
We can prove that a TM cannot decide A in less time than $O(n \log n)$
We can prove that a TM cannot decide A in less time than $O(n\log n)$

*7.49 Extra Credit. Let $f(n) = o(n\log n)$. Then $Time(f(n))$ contains only regular languages.

where $f(n) = o(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

ie, for all $c > 0$, $\exists n_0$ such that $f(n) < cg(n)$ for all $n \geq n_0$

$f$ asymptotically LESS THAN $g$
Can $A = \{ 0^k1^k \mid k \geq 0 \}$ be decided in time $O(n)$ with a two-tape TM?

Scan all 0s and copy them to the second tape. Scan all 1s, crossing off a 0 from the second tape for each 1.
Different models of computation yield different running times for the same language!
Theorem: Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$-time multi-tape TM has an equivalent $O(t(n)^2)$ single tape TM.

Claim: Simulating each step in the multi-tape machine uses at most $O(t(n))$ steps on a single-tape machine. Hence total time of simulation is $O(t(n)^2)$.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
**Theorem:** Let \( t(n) \) be a function such that \( t(n) \geq n \). Then every \( t(n) \)-time multi-tape TM has an equivalent \( O(t(n)^2) \) single tape TM.

**Analysis:** (Note, \( k \), the \# of tapes, is fixed.)

Let \( S \) be simulator

- Put \( S \)'s tape in proper format: \( O(n) \) steps
- **Two scans** to simulate one step,
  1. to obtain info for next move \( O(t(n)) \) steps, why?
  2. to simulate it (may need to shift everything over to right possibly \( k \) times): \( O(t(n)) \) steps, why?
**Theorem:** Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$-time multi-tape TM has an equivalent $O(t(n)^2)$ single tape TM.

Analysis: (Note, $k$, the # of tapes, is fixed.)

Let $S$ be simulator

- Put $S$’s tape in proper format: $O(n)$ steps
- Two scans to simulate one step,
  1. to obtain info for next move $O(t(n))$ steps, why?
  2. to simulate it (may need to shift everything over to right possibly $k$ times): $O(t(n))$ steps, why?

Therefore, $O(n) + t(n) O(t(n)) = O(t(n)^2)$ steps in simulation.
\[ P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k) \]
NON-DETERMINISTIC TURING MACHINES AND NP
Definition: A Non-Deterministic TM is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \notin \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{L,R\})}$
- $q_0 \in Q$ is the start state
- $q_{accept} \in Q$ is the accept state
- $q_{reject} \in Q$ is the reject state, and $q_{reject} \neq q_{accept}$
NON-DETERMINISTIC TMs

...are just like standard TMs, except:

1. The machine may proceed according to several possibilities

2. The machine accepts a string if there exists a path from start configuration to an accepting configuration
Deterministic Computation:

- accept or reject

Non-Deterministic Computation:

- accept
- reject
**Definition:** Let $M$ be a NTM that is a decider, i.e. on all inputs, all branches halt (with accept or reject). The **running time or time-complexity** of $M$ is the function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any branch of its computation on any input of length $n$. 
Theorem: Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$-time nondeterministic single-tape TM has an equivalent $2^{O(t(n))}$ deterministic single tape TM.
Definition:  \( \text{NTIME}(t(n)) = \{ L \mid L \text{ is decided by a } O(t(n))-\text{time non-deterministic Turing machine} \} \)

\[ \text{TIME}(t(n)) \subseteq \text{NTIME}(t(n)) \]
BOOLEAN FORMULAS

φ = (¬x ∧ y) ∨ z

A satisfying assignment is a setting of the variables that makes the formula true

x = 1, y = 1, z = 1 is a satisfying assignment for φ
A Boolean formula is **satisfiable** if there exists a satisfying assignment for it.

\[
\text{YES:} \quad a \land b \land c \land \neg d
\]

\[
\text{NO:} \quad \neg(x \lor y) \land x
\]

\[
\text{SAT} = \{ \phi \mid \phi \text{ is a satisfiable Boolean formula} \}
\]
A 3cnf-formula is of the form:

\[(x_1 \vee \neg x_2 \vee x_3) \land (x_4 \vee x_2 \vee x_5) \land (x_3 \vee \neg x_2 \vee \neg x_1)\]

- literals
- clauses

\[\begin{align*}
(x_1 \vee \neg x_2 \vee x_1) \\
(x_3 \vee x_1) \land (x_3 \vee \neg x_2 \vee \neg x_1) \\
(x_1 \vee x_2 \vee x_3) \land (\neg x_4 \vee x_2 \vee x_1) \lor (x_3 \vee x_1 \vee \neg x_1) \\
(x_1 \vee \neg x_2 \vee x_3) \land (x_3 \land \neg x_2 \land \neg x_1)
\end{align*}\]
A 3cnf-formula is of the form:

\((x_1 \lor \neg x_2 \lor x_3) \land (x_4 \lor x_2 \lor x_5) \land (x_3 \lor \neg x_2 \lor \neg x_1)\)

The formula is satisfiable if at least one clause is true. Here are some examples:

**YES**

\((x_1 \lor \neg x_2 \lor x_1)\)

**NO**

\((x_3 \lor x_1) \land (x_3 \lor \neg x_2 \lor \neg x_1)\)

**NO**

\((x_1 \lor x_2 \lor x_3) \land (\neg x_4 \lor x_2 \lor x_1) \lor (x_3 \lor x_1 \lor \neg x_1)\)

**NO**

\((x_1 \lor \neg x_2 \lor x_3) \land (x_3 \land \neg x_2 \land \neg x_1)\)

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

**Theorem:** $\text{3SAT} \in \text{NTIME}(n^2)$

On input $\phi$:

1. Check if the formula is in 3cnf
2. For each variable, non-deterministically substitute it with 0 or 1
3. Test if the assignment satisfies $\phi$
NP = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)
Theorem: $L \in NP \iff$ if there exists a poly-time Turing machine $V(\text{erifier})$ with

$L = \{ x | \exists y(\text{witness}) \ |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$
Theorem: $L \in NP \iff$ if there exists a poly-time Turing machine $V(\text{erifier})$ with

$L = \{ x \mid \exists y \ (\text{witness}) \ |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$

Proof:

(1) If $L = \{ x \mid \exists y \ |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$
then $L \in NP$

(2) If $L \in NP$ then

$L = \{ x \mid \exists y \ |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$
Theorem: $L \in \text{NP} \iff$ if there exists a poly-time Turing machine $V(\text{erifier})$ with

$L = \{ x | \exists y (\text{witness}) |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$

Proof:

(1) If $L = \{ x | \exists y |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$ then $L \in \text{NP}$

Because we can guess $y$ and then run $V$

(2) If $L \in \text{NP}$ then

$L = \{ x | \exists y |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$
Theorem: $L \in \text{NP} \iff$ if there exists a poly-time Turing machine $V(\text{erifier})$ with

$L = \{ x \mid \exists y (\text{witness}) \ |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$

Proof:

(1) If $L = \{ x \mid \exists y \ |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$ then $L \in \text{NP}$

Because we can guess $y$ and then run $V$

(2) If $L \in \text{NP}$ then

$L = \{ x \mid \exists y \ |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$

Let $N$ be a non-deterministic poly-time TM that decides $L$ and define $V(x,y)$ to accept if $y$ is an accepting computation history of $N$ on $x$
3SAT = \{ \phi \mid \exists y \text{ such that } y \text{ is a satisfying assignment to } \phi \text{ and } \phi \text{ is in 3cnf} \}

SAT = \{ \phi \mid \exists y \text{ such that } y \text{ is a satisfying assignment to } \phi \}
A language is in NP if and only if there exist polynomial-length certificates* for membership to the language.

SAT is in NP because a satisfying assignment is a polynomial-length certificate that a formula is satisfiable.*

* that can be verified in poly-time
HAMILTONIAN PATHS
HAMILTONIAN PATHS
HAMPATH = \{ (G,s,t) \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t \} \\

**Theorem:** HAMPATH $\in$ NP \\

The Hamilton path itself is a certificate
K-CLIQUE
K-CLIQUE
CLIQUE = \{ (G,k) \mid G \text{ is an undirected graph with a } k\text{-clique} \}

**Theorem:** CLIQUE $\in$ NP

The k-clique itself is a certificate
NP = all the problems for which once you have the answer it is easy (i.e. efficient) to verify
P = NP?
$P = NP?$
**POLY-TIME REDUCIBILITY**

\[ f : \Sigma^* \rightarrow \Sigma^* \] is a polynomial time computable function if some poly-time Turing machine \( M \), on every input \( w \), halts with just \( f(w) \) on its tape.

Language \( A \) is polynomial time reducible to language \( B \), written \( A \leq_P B \), if there is a poly-time computable function \( f : \Sigma^* \rightarrow \Sigma^* \) such that:

\[ w \in A \iff f(w) \in B \]

\( f \) is called a polynomial time reduction of \( A \) to \( B \).
Theorem: If \( A \leq_P B \) and \( B \in P \), then \( A \in P \)

Proof: Let \( M_B \) be a poly-time (deterministic) TM that decides \( B \) and let \( f \) be a poly-time reduction from \( A \) to \( B \)

We build a machine \( M_A \) that decides \( A \) as follows:

On input \( w \):

1. Compute \( f(w) \)
2. Run \( M_B \) on \( f(w) \)
Definition: A language $B$ is NP-complete if:

1. $B \in \text{NP}$
2. Every $A$ in NP is poly-time reducible to $B$ (i.e. $B$ is NP-hard)
Suppose B is NP-Complete

So, if B is NP-Complete and $B \in P$ then $NP = P$. Why?
Theorem (Cook-Levin): SAT is NP-complete

Corollary: SAT ∈ P if and only if P = NP
Read Chapter 7.3 of the book for next time