Lecture 22:

- Nondeterministic time, NP
- NP-completeness

Reading:
Sipser Ch 7.3-7.4

Mark Bun
April 14, 2021

OH: 6-7 pm today
Big-Oh, formally

\( f(n) = O(g(n)) \) means:

There exist constants \( c > 0, n_0 > 0 \) such that

\[ f(n) \leq c g(n) \text{ for every } n \geq n_0 \]

Example: Show that \( 7n^2 \cdot 3^n = 2^{O(n)} \)

\[
\log \left( 7n^2 \cdot 3^n \right) = \log \left( 2^{g(n)} \right) \quad \text{where } g(n) = 0(n)
\]

\[
\log \left( 7n^2 \cdot 3^n \right) = O(n)
\]

\[
\begin{align*}
\text{LHS} & = (\log 3 + 2 + \log 7) \cdot n \\
& \leq (\log 3) \cdot n + 2 \cdot n + (\log 7) \cdot n \\
& \leq 2 \cdot n + (\log 7) n \\
\end{align*}
\]

Meas: \( \exists g(n) \text{ s.t. } g(n) = O(n) \) and \( 7n^2 \cdot 3^n = 2^{g(n)} \)
Big-Oh, formally

$f(n) = O(g(n))$ means:

There exist constants $c > 0, n_0 > 0$ such that $f(n) \leq cg(n)$ for every $n \geq n_0$

Example: Show that $7n^2 \cdot 3^n = 2^{O(n)}$

Proof:

Set $c = \log 7 + 2 + \log 3$, $n_0 = 1$

Set $g(n) = \log (7n^2 \cdot 3^n)$

It suffices to show that $g(n) = O(n)$ [since $7n^2 \cdot 3^n = 2^{O(n)} \iff g(n) = O(n)$]

$g(n) = \log (7n^2 \cdot 3^n) = \log 7 + 2 \log n + n \log 3 \\ \leq c \cdot n$ (for $n \geq n_0 = 1$) \quad \Box
Nondeterministic time

Let $f : \mathbb{N} \rightarrow \mathbb{N}$

A NTM $M$ runs in time $f(n)$ if on every input $w \in \Sigma^n$, $M$ halts on $w$ within at most $f(n)$ steps on every computational branch

NTIME($f(n)$) is a class (i.e., set) of languages:

A language $A \in$ NTIME($f(n)$) if there exists an NTM $M$ that

1) Decides $A$, and
2) Runs in time $O(f(n))$
Complexity class $\mathbf{NP}$

**Definition:** $\mathbf{NP}$ is the class of languages decidable in polynomial time on a nondeterministic TM

$$\mathbf{NP} = \bigcup_{k=1}^{\infty} \mathbf{NTIME}(n^k)$$
**HAMPATH ∈ NP**

\[ HAMPATH = \{ (G, s, t) \mid \text{G is a directed graph and there is a path from } s \text{ to } t \text{ that passes through every vertex exactly once} \} \]

On input \( (G, s, t) \):

1. **Nondeterministically** guess a sequence of vertices
2. Check that the guess forms a Hamiltonian path from \( s \) to \( t \)
An alternative characterization of \textbf{NP}

“Languages with polynomial-time verifiers”

A \textit{verifier} for a language \( L \) is a \textit{deterministic} algorithm \( V \) such that \( w \in L \) iff there \textit{exists} a string \( c \) such that \( V(\langle w, c \rangle) \) accepts

Running time of a verifier is only measured in terms of \(|w|\)

\( V \) is a \textit{polynomial-time verifier} if it runs in time polynomial in \(|w|\) on every input \( \langle w, c \rangle \)

(Without loss of generality, \(|c|\) is polynomial in \(|w|\), i.e., \(|c| = O(|w|^k)\) for some constant \( k \))
HAMPATH has a polynomial-time verifier

Verifier V:

| Certificate c: | \( c_1, \ldots, c_k \in [l,w] \) |

On input \( (G, s, t; c) \): (Vertices of \( G \) are numbers 1, \( \ldots, k \))

1. Check that \( c_1, c_2, \ldots, c_k \) is a permutation: Every number 1, \( \ldots, k \) appears exactly once

2. Check that \( c_1 = s, c_k = t \), and there is an edge from every \( c_i \) to \( c_{i+1} \)

3. Accept if all checks pass, otherwise, reject.
NP is the class of languages with polynomial-time verifiers

**Theorem:** A language \( L \in \text{NP} \) iff there is a polynomial-time verifier for \( L \)

**Proof:** \( \Leftarrow \) Let \( L \) have a poly-time verifier \( V(\langle w, c \rangle) \)

**Idea:** Design NTM \( N \) for \( L \) that nondeterministically guesses a certificate

\[
N = \text{"on input } w:\ 
\begin{align*}
1) \text{Nondeterministically guess certificate } c \text{ (of appropriate length)} \\
2) \text{Run } V(\langle w, c \rangle), \text{ return result} \\
\end{align*}
\]

**Correctness:** \( w \in L \Rightarrow \exists c \text{ s.t. } V(\langle w, c \rangle) \text{ accepts } \Rightarrow N \text{ accepts } w \)

**Only run time:** 1) \( c \) has to have poly \((|w|)\) length for \( V \) to be efficient

2) \( V \) itself runs in poly time \( \Rightarrow \) poly time NTM
NP is the class of languages with polynomial-time verifiers

⇒ Let $L$ be decided by an NTM $N$ making up to $b$ nondeterministic choices in each step, running $T(n)$.

Idea: Design verifier $V$ for $L$ where certificate is sequence of “good” nondeterministic choices

$V = \text{" accept if accept, reject otherwise "}$

1. Simulate $N$ on $w$ using $c$ as the sequence of nondet. choices

Need to show correctness, i.e. $w \in L \Rightarrow \exists c$ s.t. $V(<w,c>)$ accepts.

Need to show poly-time: $V$ runs in the poly$(1,\nu_1)$.
Alternative proof of \( \text{NP} \subseteq \text{EXP} \)

One can prove \( \text{NP} \subseteq \text{EXP} \) as follows. Let \( V \) be a verifier for a language \( L \) running in time \( T(n) \). We can construct a \( 2^{O(T(n))} \) time algorithm for \( L \) as follows.

a) On input \( \langle w, c \rangle \), run \( V \) on \( \langle w, c \rangle \) and output the result \( \checkmark \)

b) On input \( w \), run \( V \) on all possible \( \langle w, c \rangle \), where \( c \) is a certificate. Accept if any run accepts.

c) On input \( w \), run \( V \) on all possible \( \langle w, c \rangle \), where \( c \) is a certificate of length at most \( T(|w|) \). Accept if any run accepts.

\[ \text{running time} \sim T(n) \cdot 2^{T(n)} \text{ exp} \quad \text{poly} \quad \text{where} \quad T \text{-is poly} \]

d) On input \( w \), run \( V \) on all possible \( \langle x, c \rangle \), where \( x \) is a string of length \( |w| \) and \( c \) is a certificate of length at most \( T(|w|) \). Accept if any run accepts.
WARNING: Don’t mix-and-match the NTM and verifier interpretations of NP

To show a language $L$ is in NP, do exactly one:

1) Exhibit a poly-time NTM for $L$
   
   $N = \text{“On input } x:\n   <\text{Do some nondeterministic stuff}>..."$

OR

2) Exhibit a poly-time (deterministic) verifier for $L$
   
   $V = \text{“On input } x \text{ and certificate } c:\n   <\text{Do some deterministic stuff}>..."$

instance $x$ to problem specified by $L$

Guess a string $(c_1, \ldots, c_k) \in \Sigma^k$ in time $O(n)$
Examples of NP languages: SAT

“Is there an assignment to the variables in a logical formula that make it evaluate to true?”

• **Boolean variable**: Variable that can take on the value true/false (encoded as 0/1)  
  \[ \exists \ x_1, x_2, x, y, z \]

• **Boolean operations**: \( \land \) (AND), \( \lor \) (OR), \( \neg \) (NOT)

• **Boolean formula**: Expression made of Boolean variables and operations.  
  \[ (x_1 \lor \overline{x_2}) \land x_3 = \varphi \]

• An assignment of 0s and 1s to the variables satisfies a formula \( \varphi \) if it makes the formula evaluate to 1  
  Assignment: \( x_1 = 0, x_2 = 1, x_3 = 1 \) does not satisfy \( \varphi \), but \( x_1 = 1, x_2 = 1, x_3 = 0 \) does satisfy \( \varphi \)

• A formula \( \varphi \) is satisfiable if there exists an assignment that satisfies it
Examples of \textbf{NP} languages: SAT

Ex: $(x_1 \lor \overline{x_2}) \land x_3$

Assignment $x_1 = 1, x_2 = 1, x_3 = 1$ satisfies it

Ex: $(x_1 \lor x_2) \land \overline{x_1} \land \overline{x_2}$

Not satisfiable

$SAT = \{ \langle \varphi \rangle | \varphi \text{ is a satisfiable formula} \}$

Claim: $SAT \in \text{NP}$

Verifier for SAT:

\begin{enumerate}
  \item On input $\varphi(x_1, \ldots, x_n)$,
    \begin{enumerate}
      \item Nondet. guess $\hat{x}_1, \ldots, \hat{x}_n \in \{0, 1\}^n$
      \item If $\varphi(\hat{x}_1, \ldots, \hat{x}_n) = 1$, accept.
      \item Else, reject.
    \end{enumerate}
\end{enumerate}

4/14/2021
Examples of NP languages: Traveling Salesperson

“Given a list of cities and distances between them, is there a ‘short’ tour of all of the cities?”

More precisely: Given

• A number of cities \( m \)

• A function \( D: \{1, \ldots, m\}^2 \rightarrow \mathbb{N} \) giving the distance between each pair of cities

• A distance bound \( B \)

\[
TSP = \{\langle m, D, B \rangle | \exists \text{ a tour visiting every city with length } \leq B \}\]
**P vs. NP**

**Question:** Does $P = NP$?

Philosophically: Can every problem with an efficiently verifiable solution also be solved efficiently?

A central problem in mathematics and computer science

---

**EXP**

**NP**

If $P \neq NP$

**EXP**

**P = NP**

If $P = NP$
A world where P = NP

• Many important decision problems can be solved in polynomial time (HAMPATH, SAT, TSP, etc.)

• Many search problems can be solved in polynomial time (e.g., given a natural number, find a prime factorization)

• Many optimization problems can be solved in polynomial time (e.g., find the lowest energy conformation of a protein)
A world where $P = NP$

- Secure cryptography becomes impossible
  An NP search problem: Given a ciphertext $c$, find a plaintext $m$ and encryption key $k$ that would encrypt to $c$

- AI / machine learning become easy: Identifying a consistent classification rule is an NP search problem

- Finding mathematical proofs becomes easy: NP search problem: Given a mathematical statement $S$ and length bound $k$, is there a proof of $S$ with length at most $k$?

General consensus: $P \neq NP$