Last time:
- $\text{TIME}(t(n))$
- $P = \bigcup_k \text{TIME}(n^k)$
- $PATH \in P$

Today:  (Sipser §7.2 – §7.3)
- $\text{NTIME}(t(n))$
- $NP$
- $P$ vs $NP$ problem
- Dynamic Programming
- Polynomial-time reducibility
**Quick Review**

**Defn:** \( \text{TIME}(t(n)) = \{ B \mid \text{some deterministic 1-tape TM } M \text{ decides } B \) and \( M \) runs in time \( O(t(n)) \}\)

**Defn:** \( P = \bigcup_k \text{TIME}(n^k) \)

\( = \) polynomial time decidable languages

\( PATH = \{ \langle G, s, t \rangle \mid G \text{ is a directed graph with a path from } s \text{ to } t \} \)

**Theorem:** \( PATH \in P \)

\( HAMPATH = \{ \langle G, s, t \rangle \mid G \text{ is a directed graph with a path from } s \text{ to } t \) that goes through every node of } G \} \)

\( HAMPATH \in P ? \)  
[connection to factoring]
Nondeterministic Complexity

In a nondeterministic TM (NTM) decider, all branches halt on all inputs.

**Defn:** An NTM runs in time $t(n)$ if all branches halt within $t(n)$ steps on all inputs of length $n$.

**Defn:** $\text{NTIME}(t(n)) = \{ B \mid \text{some 1-tape NTM decides } B$ and runs in time $O(t(n)) \} \}

**Defn:** $\text{NP} = \bigcup_k \text{NTIME}(n^k)$

$= \text{nondeterministic polynomial time decidable languages}$

- Invariant for all reasonable nondeterministic models
- Corresponds roughly to easily verifiable problems
**Theorem:** \( \textsc{Hampath} \in \text{NP} \)

**Proof:**

"On input \( \langle G, s, t \rangle \) (Say \( G \) has \( m \) nodes.)

1. Nondeterministically write a sequence \( \langle v_1, v_2, \ldots, v_m \rangle \) of \( m \) nodes.
2. **Accept** if \( v_1 = s \)
   - \( v_m = t \)
   - each \( (v_i, v_{i+1}) \) is an edge
   - and no \( v_i \) repeats.
3. **Reject** if any condition fails."

Computation of \( M \) on \( \langle G, s, t \rangle \)

- Guess bits of \( v_1 \)
- Guess bits of \( v_2 \)
- Guess bits of \( v_m \)
- Check \( v_1, v_2, \ldots, v_m \) works

\[ \text{acc/rej acc/rej} \]
**COMPOSITES ∈ NP**

**Defn:** \( \text{COMPOSITES} = \{ x \mid x \text{ is not prime and } x \text{ is written in binary} \} \\
= \{ x \mid x = yz \text{ for integers } y, z > 1, \ x \text{ in binary} \} \)

**Theorem:** \( \text{COMPOSITES} \in \text{NP} \)

**Proof:** "On input \( x \)

1. Nondeterministically write \( y \) where \( 1 < y < x \).
2. Accept if \( y \) divides \( x \) with remainder 0.
   Reject if not."

**Note:** Using base 10 instead of base 2 wouldn’t matter because can convert in polynomial time.
**Bad encoding:** write number \( k \) in unary: \( 1^k = \underbrace{111 \ldots 1}_k \), exponentially longer.

**Theorem (2002):** \( \text{COMPOSITES} \in \text{P} \)
We won’t cover this proof.
Intuition for P and NP

NP = All languages where can verify membership quickly
P = All languages where can test membership quickly

Examples of quickly verifying membership:
- **HAMPATH**: Give the Hamiltonian path.
- **COMPOSITES**: Give the factor.

The Hamiltonian path and the factor are called *short certificates* of membership.

### Check-in 14.1

Let $\overline{HAMPATH}$ be the complement of $HAMPATH$.

So $(G, s, t) \in \overline{HAMPATH}$ if $G$ does not have a Hamiltonian path from $s$ to $t$.

Is $HAMPATH \in NP$?

(a) Yes, we can invert the accept/reject output of the NTM for $HAMPATH$.
(b) No, we cannot give a short certificate for a graph not to have a Hamiltonian path.
(c) I don’t know.
Recall $A_{\text{CFG}}$

Recall: $A_{\text{CFG}} = \{ \langle G, w \rangle | G \text{ is a CFG and } w \in L(G) \}$

**Theorem:** $A_{\text{CFG}}$ is decidable

**Proof:** $D_{A-\text{CFG}} =$ "On input $\langle G, w \rangle$"
1. Convert $G$ into Chomsky Normal Form.
2. Try all derivations of length $2|w| - 1$.

Chomsky Normal Form (CNF):
- $A \rightarrow BC$
- $B \rightarrow b$

Let’s always assume $G$ is in CNF.

**Theorem:** $A_{\text{CFG}} \in \text{NP}$

**Proof:** "On input $\langle G, w \rangle$"
1. Nondeterministically pick some derivation of length $2|w| - 1$.
2. *Accept* if it generates $w$. *Reject* if not.
Attempt to show $A_{\text{CFG}} \in P$

**Theorem:** $A_{\text{CFG}} \in P$

Proof attempt:

Recursive algorithm $C$ tests if $G$ generates $w$, starting at any specified variable $R$.

$C =$ “On input $\langle G, w, R \rangle$ 

1. For each way to divide $w = xy$ and for each rule $R \rightarrow ST$
2. Use $C$ to test $\langle G, x, S \rangle$ and $\langle G, y, T \rangle$
3. Accept if both accept
4. Reject if none of the above accepted.”

Then decide $A_{\text{CFG}}$ by starting from $G$’s start variable.

$C$ is a correct algorithm, but it takes non-polynomial time. 
(Each recursion makes $O(n)$ calls and depth is roughly $\log n$.)

**Fix:** Use recursion + memory called *Dynamic Programming (DP)*

**Observation:** String $w$ of length $n$ has $O(n^2)$ substrings $w_i \cdots w_j$ therefore there are only $O(n^2)$ possible sub-problems $\langle G, x, S \rangle$ to solve.
**Theorem:** $A_{CFG} \in P$

**Proof:** Use DP (Dynamic Programming) = recursion + memory.

$D = \text{"On input } \langle G, w, R \rangle$

1. For each way to divide $w = xy$ and for each rule $R \rightarrow ST$
2. Use $D$ to test $\langle G, x, S \rangle$ and $\langle G, y, T \rangle$
3. Accept if both accept
4. Reject if none of the above accepted.”

Then decide $A_{CFG}$ by starting from $G$’s start variable.

Total number of calls is $O(n^2)$ so time used is polynomial.

Alternately, solve all smaller sub-problems first: “bottom up”

**Check-in 14.2**
Suppose $B$ is a CFL. Does that imply that $B \in P$?
(a) Yes
(b) No.
**Theorem:** $A_{\text{CFG}} \in \mathcal{P}$ & Bottom-up DP

Proof: Use bottom-up DP.

$D = “\text{On input } \langle G, w \rangle$}

1. For each $w_i$ and variable $R$
   Solve $\langle G, w_i, R \rangle$ by checking if $R \rightarrow w_i$ is a rule.

2. For $k = 2, \ldots, n$ and each substring $u$ of $w$ where $|u| = k$ and variable $R$
   Solve $\langle G, u, R \rangle$ by checking for each $R \rightarrow ST$ and each division $u = xy$
   if both $\langle G, x, S \rangle$ and $\langle G, y, T \rangle$ were positive.

3. Accept if $\langle G, w, S \rangle$ is positive where $S$ is the original start variable.
   4. Reject if not.”

Total number of calls is $O(n^2)$ so time used is polynomial.

Often, bottom-up DP is shown as filling out a table.
Satisfiability Problem

**Defn:** A *Boolean formula* $\phi$ has Boolean variables (TRUE/FALSE values) and Boolean operations AND ($\land$), OR ($\lor$), and NOT ($\neg$).

**Defn:** $\phi$ is *satisfiable* if $\phi$ evaluates to TRUE for some assignment to its variables. Sometimes we use 1 for True and 0 for False.

**Example:** Let $\phi = (x \lor y) \land (\overline{x} \lor \overline{y})$ (Notation: $\overline{x}$ means $\neg x$)
Then $\phi$ is satisfiable ($x=1$, $y=0$)

**Defn:** $SAT = \{ \langle \phi \rangle | \phi$ is a satisfiable Boolean formula $\}$

**Theorem (Cook, Levin 1971):** $SAT \in P \rightarrow P = NP$
**Proof method:** polynomial time (mapping) reducibility

**Check-in 14.3**
Is $SAT \in NP$?
(a) Yes.
(b) No.
(c) I don’t know.
(d) No one knows.
**Polynomial Time Reducibility**

**Defn:** $A$ is polynomial time reducible to $B$ ($A \leq_P B$) if $A \leq_m B$ by a reduction function that is computable in polynomial time.

**Theorem:** If $A \leq_P B$ and $B \in P$ then $A \in P$. 

![Diagram showing reducibility and containment between complexity classes](image)

Idea to show $SAT \in P \rightarrow P = NP$

Analogy with $A_{TM}$

12
Quick review of today

1. $\text{NTIME}(t(n))$ and NP
2. $\text{HAMPATH}$ and $\text{COMPOSITES} \in \text{NP}$
3. P versus NP question
4. $A_{\text{CFG}} \in \text{P}$ via Dynamic Programming
5. The Satisfiability Problem $\text{SAT}$
6. Polynomial time reducibility