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

Today:
- $\text{NTIME}(t(n))$
- $NP$
- P vs NP problem
- Dynamic Programming
- Polynomial-time reducibility

Posted:
- Midterm & solutions, Problem Set 3 solutions, Problem Set 4
Quick Review

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

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

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

**Theorem:** \( \text{PATH} \in P \)

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

\( \text{HAMPATH} \in P \)?

[connection to factoring]
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:** $\text{NTIME}(t(n)) = \{B | \text{some 1-tape NTM decides } B$ and runs in time $O(t(n)) \}$

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

- = nondeterministic polynomial time decidable languages
- Invariant for all reasonable nondeterministic models
- Corresponds roughly to easily verifiable problems
Theorem: \( \text{HAMPATH} \in \text{NP} \)

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. \textbf{Accept} if \( v_1 = s \) \( \quad v_m = t \)
   each \( (v_i, v_{i+1}) \) is an edge and no \( v_i \) repeats.

3. \textbf{Reject} if any condition fails."
\textbf{COMPOSITES} \in \textbf{NP}

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

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

\textbf{Proof:} “On input } x 
1. Nondeterministically write } y \text{ where } 1 < y < x.
2. \text{Accept if } y \text{ divides } x \text{ with remainder } 0.
\text{Reject if not.”}

\textbf{Note:} Using base 10 instead of base 2 wouldn’t matter because can convert in polynomial time.

\textbf{Bad encoding:} write number } k \text{ in unary: } 1^k = \underbrace{111 \cdots 1}_{k}, \text{ exponentially longer.}

\textbf{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 \( (G, s, t) \in HAMPATH \) 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 $2|w| - 1$.
3. Accept if any generate $w$. Reject if not.

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 $A_{CFG} \in P$

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

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

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

1. For each way to divide $w = xy$ and for each rule $R \rightarrow ST$
2. Use $C$ 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 $A_{CFG}$ 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.
DP shows $A_{CFG} \in P$

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

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

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

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:** \( A_{CFG} \in P \)

**Proof:** Use bottom-up DP.

\( D = \) "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 \( k = 2, ..., n \) 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 \( \phi = (x \lor y) \land (\neg x \lor \neg y) \) (Notation: \( \neg x \) means \( \neg x \))

Then \( \phi \) is satisfiable (\( x=1, y=0 \))

**Defn:** \( SAT = \{ \phi | \phi \) 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.
**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 \(A \in P\).

Idea to show \(SAT \in P \rightarrow P = NP\)

Analogy with \(A_{TM}\)
Quick review of today

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