18.404/6.840 Lecture 14

(midterm replaced lecture 13)

Last time:

- TIME$\big(t(n)\big)$
- P $= \bigcup_k$ TIME(n^k)
- $PATH \in$ P

Today: (Sipser §7.2 – §7.3)

- NTIME$\big(t(n)\big)$
- NP
- P vs NP problem
- Dynamic Programming
- Polynomial-time reducibility

Quick Review

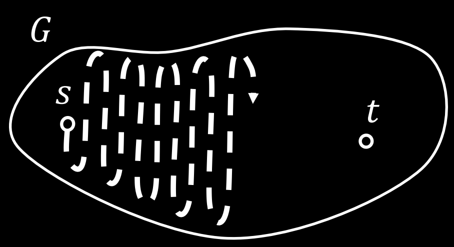
Defn: $\text{TIME}\big(t(n)\big) = \{B \mid$ some deterministic 1-tape TM M decides B and M runs in time $O\big(t(n)\big)\}$

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

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

Theorem: $PATH \in \text{P}$

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



$HAMPATH \in \text{P}$?

[connection to factoring]

Nondeterministic Complexity

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

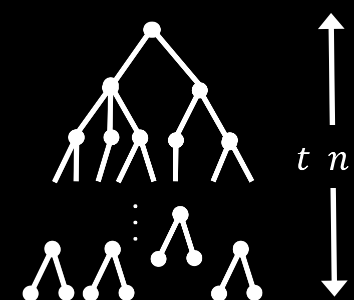
Defn: An NTM runs in time $t(n)$ if all branches halt within $t(n)$ steps on all inputs of length n.

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

Defn: **NP** $= \bigcup_k \text{NTIME}(n^k)$
$=$ nondeterministic polynomial time decidable languages

- Invariant for all reasonable nondeterministic models
- Corresponds roughly to easily verifiable problems

Computation tree for NTM on input w.



all branches halt within $t(n)$ steps

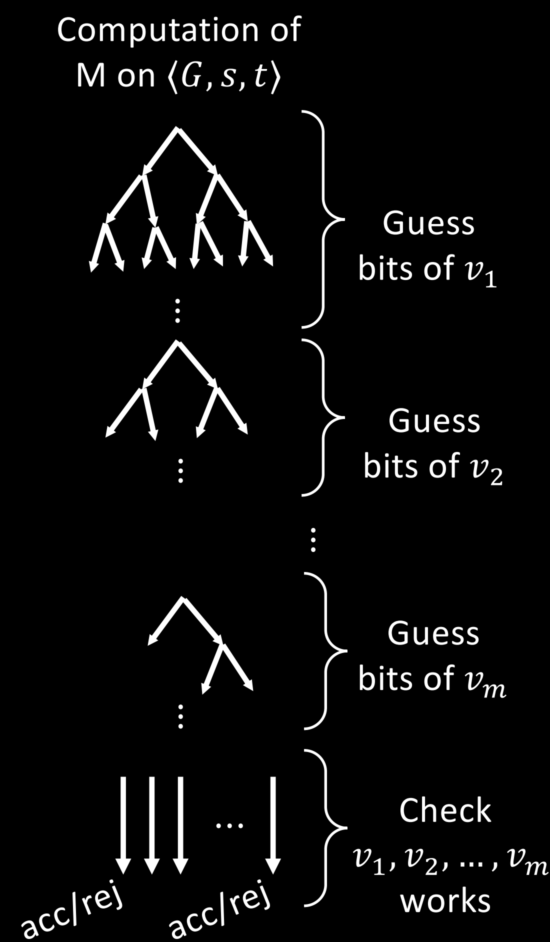
$HAMPATH \in$ NP

Theorem: $HAMPATH \in$ NP

Proof:

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

1. Nondeterministically write a sequence $v_1, v_2, \dots, v_m$ 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."



$COMPOSITES \in$ NP

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

Theorem: $COMPOSITES \in$ 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: $1^k = \overbrace{111 \cdots 1}^{k}$, exponentially longer.

Theorem (2002): $COMPOSITES \in$ P

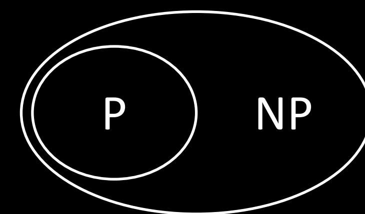
We won't cover this proof.

Intuition for P and NP

NP = All languages where can <u>verify</u> membership quickly
P = All languages where can <u>test</u> membership quickly

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

The <u>Hamiltonian path</u> and the <u>factor</u> are called ***short certificates*** of membership.



Check-in 14.1

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

So $\langle G, s, t\rangle \in \overline{HAMPATH}$ if G does <u>not</u> have a Hamiltonian path from s to t.

Is $\overline{HAMPATH} \in$ 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.

Check-in 14.1

Recall A_{CFG}

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

Theorem: A_{CFG} is decidable

Proof: $D_{\mathrm{A-CFG}} =$ "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 → BC

B → b

Let's always assume G is in CNF.

Theorem: $A_{\mathrm{CFG}} \in \mathrm{NP}$

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.

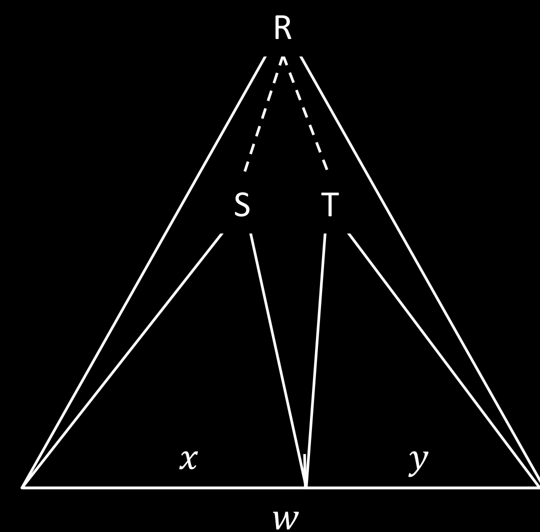
Attempt to show $A_{\mathrm{CFG}} \in \mathrm{P}$

Theorem: $A_{\mathrm{CFG}} \in \mathrm{P}$
Proof attempt:
Recursive algorithm C tests if G generates w, starting at any specified variable R.
$C =$ "On input $\langle G, w, \mathrm{R}\rangle$
1. For each way to divide $w = xy$ and for each rule $\mathrm{R} \rightarrow \mathrm{ST}$
2. Use C to test $\langle G, x, \mathrm{S}\rangle$ and $\langle G, y, \mathrm{T}\rangle$
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 $\langle G, x, \mathrm{S}\rangle$ to solve.



DP shows $A_{\text{CFG}} \in \text{P}$

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

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

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

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

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

} same as before

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 $B \in \text{P}$?

(a) Yes

(b) No.

$A_{\text{CFG}} \in$ P & Bottom-up DP

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

Proof : Use bottom-up DP.

$D =$ "On input $\langle G, w \rangle$

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

   } Solve for substrings of length 1

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

   } Solve for substrings of length k by using previous answers for substrings of length $< k$.

3. *Accept* if $\langle G, w, \text{S} \rangle$ 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*** ϕ has Boolean variables (TRUE/FALSE values) and Boolean operations AND ($\wedge$), OR ($\vee$), and NOT ($\neg$).

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

Example: Let $\phi = (x \vee y) \wedge (\overline{x} \vee \overline{y})$ (Notation: $\overline{x}$ means $\neg x$)
Then ϕ 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.

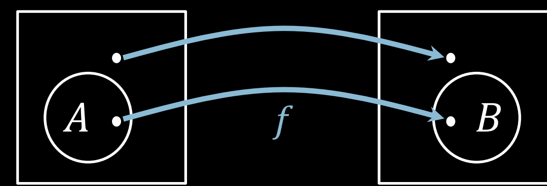
(d) No one knows.

Check-in 14.3

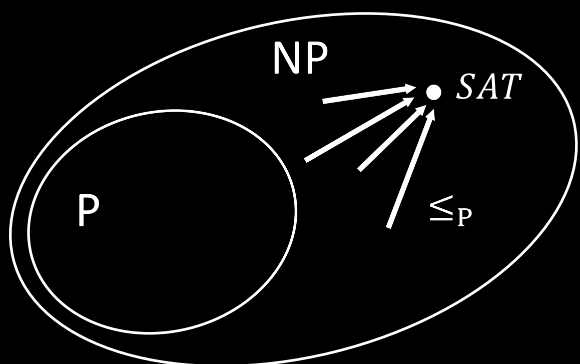
Polynomial Time Reducibility

Defn: A is polynomial time reducible to B ($A \leq_{\mathrm{P}} B$) if $A \leq_{\mathrm{m}} B$ by a reduction function that is computable in polynomial time.

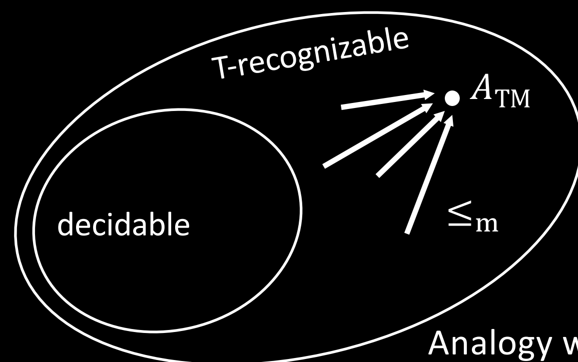
Theorem: If $A \leq_{\mathrm{P}} B$ and $B \in \mathrm{P}$ then $A \in \mathrm{P}$.



f is computable in polynomial time



Idea to show $SAT \in \mathrm{P} \rightarrow \mathrm{P} = \mathrm{NP}$



Analogy with A_{TM}

Quick review of today

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