香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF DATA SCIENCE
數據科學學院

CSC 6011 - Theory of Computation

Lecture 11 - P and NP, SAT

Time: Mon & Wed, 4:00 - 5:20 PM

Location: Teaching A, 107

Instructor: Tao LIN (林涛)

TA: Yefan GAO (高叶繁)

Fall 2026

Lecture 11 - P and NP, SAT

Last time:

- $\mathrm{TIME}(t(n))$
- $\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$
- $\mathrm{PATH} \in \mathrm{P}$
- $\mathrm{HAMPATH} \in \mathrm{P}$?

Today:

- $\mathrm{NTIME}(t(n))$
- NP
- P vs NP problem
- Satisfiability Problem

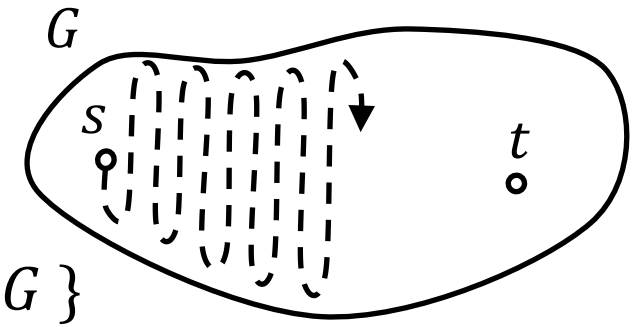
Quick Review

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

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

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

Theorem: $\mathrm{PATH} \in \mathsf{P}$



$\mathrm{HAMPATH} = \{\langle G, s, t\rangle |\ G$ is a directed graph with a path from s to t that goes through every node exactly once in $G\ \}$

$\mathrm{HAMPATH} \in \mathsf{P}$? Unsolved Problem

[connection to many other problems, e.g., 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.

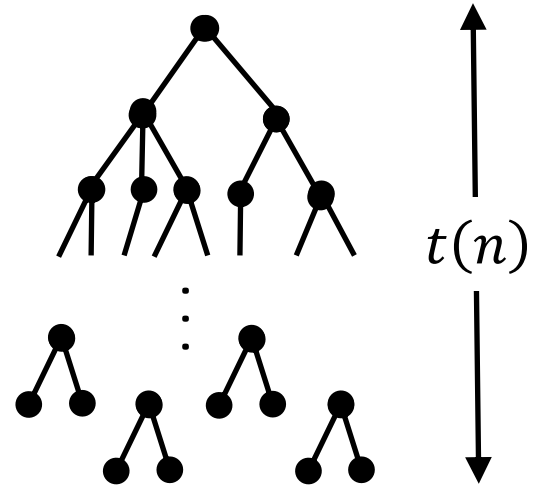
Defn: $\mathrm{NTIME}(t(n)) = \{B \mid$ some 1-tape NTM decides B and runs in time $O(t(n))\}$

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

$=$ 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 ∈ NP

Theorem: HAMPATH ∈ NP

Proof:

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

1. Nondeterministically write a sequence

   v_1, v_2, … , v_m of m nodes.

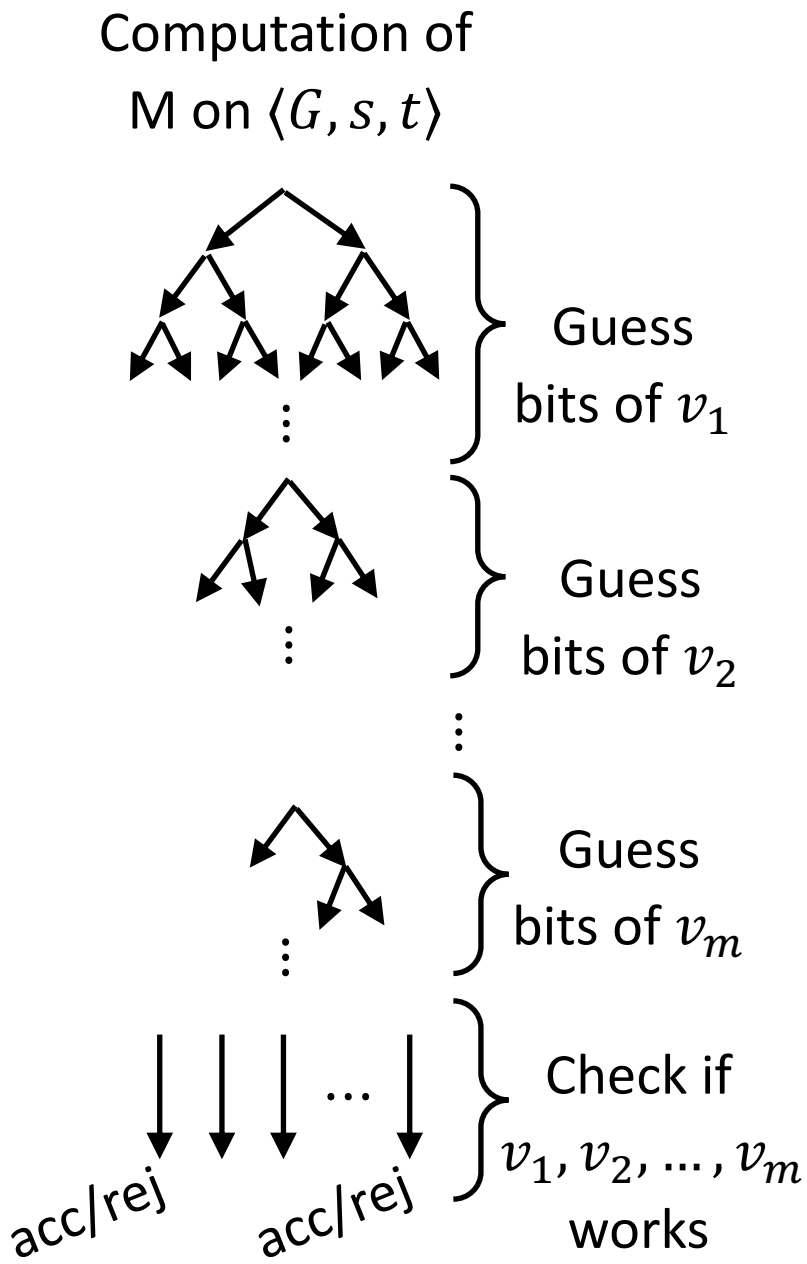
2. *Accept* if $v_1 = s$

   $v_m = t$

   each (v_i, v_{i+1}) is an edge

   and no v_i repeats.

3. *Reject* if any condition fails."



COMPOSITES ∈ NP

Note: The length of input is $n = |x| = O(\log x)$

Defn: $\text{COMPOSITES} = \{x |\ x \text{ is not a prime number (written in decimal or binary)}\}$

$= \{x |\ x = yz \text{ for integers } y, z > 1\}$

Theorem: COMPOSITES ∈ 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."

Long division takes $O(|x|^2) = O(n^2)$ time, hence in NP.

Note: Deterministically trying every $1 < y < x$ is not polynomial time:

- There are $x - 2 = O(2^n)$ possibilities

Surprising result: COMPOSITES ∈ P (Agrawal, Kayal, and Saxena, 2002)

They check whether x is prime without trying every y. (We won't cover their proof.)

NP as a Projection

Theorem: $L \in$ NP iff there are a deterministic polynomial-time decider V (verifier) and a polynomial p such that

$$L = \{x \mid \exists y,\ |y| \leq p(|x|) \text{ and } \langle x, y \rangle \text{ is accepted by } V\}.$$

Here y is a short certificate for $x \in L$.

Examples:

- $L = \text{HAMPATH}$:
  $x = \langle G, s, t \rangle$, $y =$ a Hamiltonian path.
- $L = \text{COMPOSITES}$:
  $x =$ a number, $y =$ a factor.

Recall (Lecture 06):

C is T-recognizable iff there is a T-decidable D such that

$$C = \{x \mid \exists y,\ \langle x, y \rangle \in D\}.$$

Here y need not be short, and the decider for D need not run in polynomial time.

NP requires both polynomial-length certificates and polynomial-time verification.

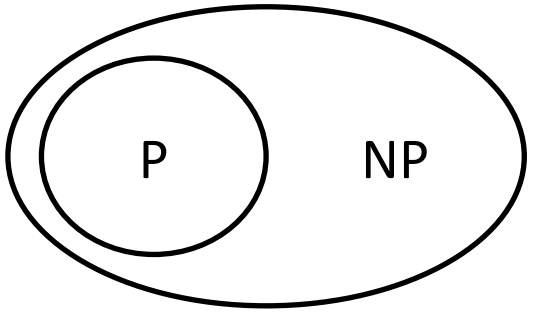
P vs NP

NP = All languages where we can verify "membership" quickly (given a "short certificate")

$(x \in L)$ (y)

P = All languages where we can decide "membership" quickly

By definition, P ⊆ NP



Question: P = NP? Famous unsolved problem (Cook 1971).

Conjecture: P ≠ NP. Some problems are in NP but not in P.

Why is this conjecture hard to prove?

One reason: polynomial-time algorithms are powerful.

(We will see an example.)

Is A_{CFG} in NP?

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

Assume G is in Chomsky Normal Form (CNF):

$A \rightarrow BC$

$B \rightarrow b$

Recall (Lecture 07): For a CFG G in CNF, every nonempty $w \in L(G)$ has a derivation of $2|w| - 1$ steps.

Question:

Is A_{CFG} in NP?

(a) Yes

(b) No.

Theorem: $A_{\text{CFG}} \in$ 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. "

In fact, $A_{\text{CFG}} \in$ P

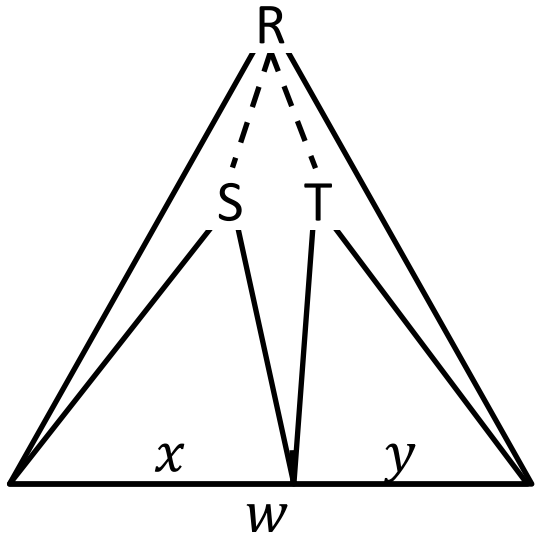
Theorem: $A_{\text{CFG}} \in$ P

Proof attempt:

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

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

1. For each way to divide $w = xy$ and for each rule R → ST
   2. Use C 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.

C is a correct algorithm, but it takes non-polynomial time.
(Each recursion makes $O(n^2)$ calls and the depth is $O(\log n)$ or larger.)

Issue: The recursion solves sub-problems repeatedly.

Fix: Use recursion + memory, also known as *Dynamic Programming* (DP)

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

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

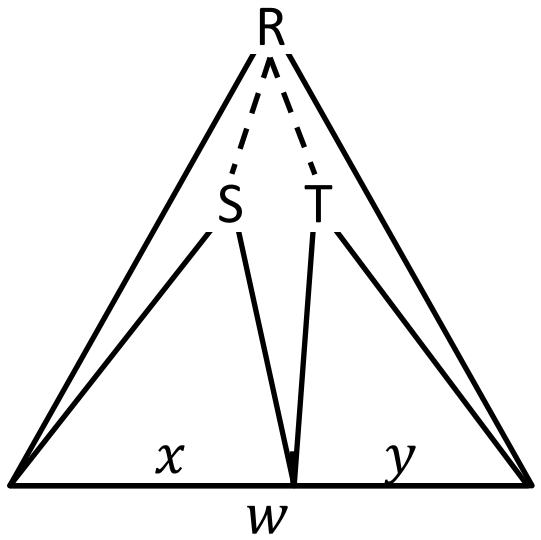
Proof : Use DP (Dynamic Programming) = recursion + memory.

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

0. If previously solved $\langle G, w, \text{R} \rangle$ then return the answer, else continue.
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."

} same as before

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



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

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

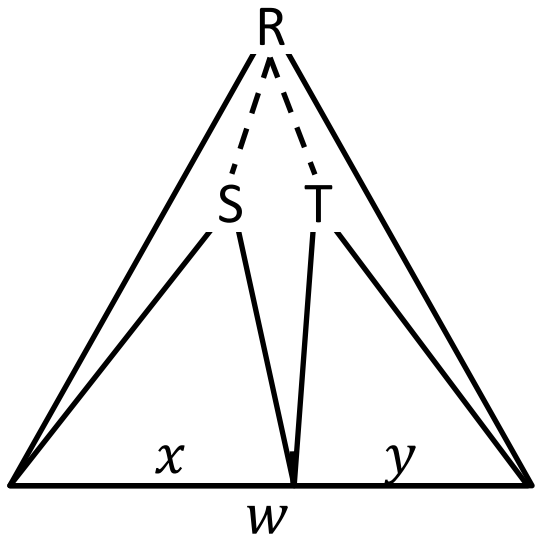
Proof : Use DP (Dynamic Programming) = recursion + memory.

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

0. If previously solved $\langle G, w, \text{R} \rangle$ then return the answer, else continue.
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."

} same as before

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



Mini-Quiz 11.3: Let $n = |\langle G, w \rangle|$.
What's the total number of sub-problems solved by this algorithm?
(Hint: Each sub-problem is indexed by $\langle G, x, \text{S} \rangle$)

(a) $O(n)$ (b) $O(n^2)$ (c) $O(n^3)$ (d) More than $O(n^3)$

Therefore, the algorithm runs in polynomial time.

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

Alternatively, we can solve all sub-problems first, from small to large ("bottom-up")

Input: CFG G in CNF with start variable S_0, string $w = w_1 \cdots w_n$

Bottom-up DP algorithm:

Construct a table M to record, for each variable R and substring $w_i \cdots w_j$:

$M[R, i, j] = 1$ if R generates $w_i \cdots w_j$, and 0 otherwise.

1. Set all entries $M[R, i, j]$ to 0.
2. For each $i = 1, \ldots, n$ and variable R, set $M[R, i, i] = 1$ if $R \to w_i$ is a rule.
3. For $\ell = 2, \ldots, n$ and every interval $i, \ldots, j$ of length ℓ:
   For each rule $R \to ST$ and each split $i \leq k < j$,
   if $M[S, i, k] = 1$ and $M[T, k+1, j] = 1$, set $M[R, i, j] = 1$.

   } Fill in the $M[R, i, j]$ entries in the order of increasing substring length
4. Accept if $M[S_0, 1, n] = 1$. Otherwise, reject."

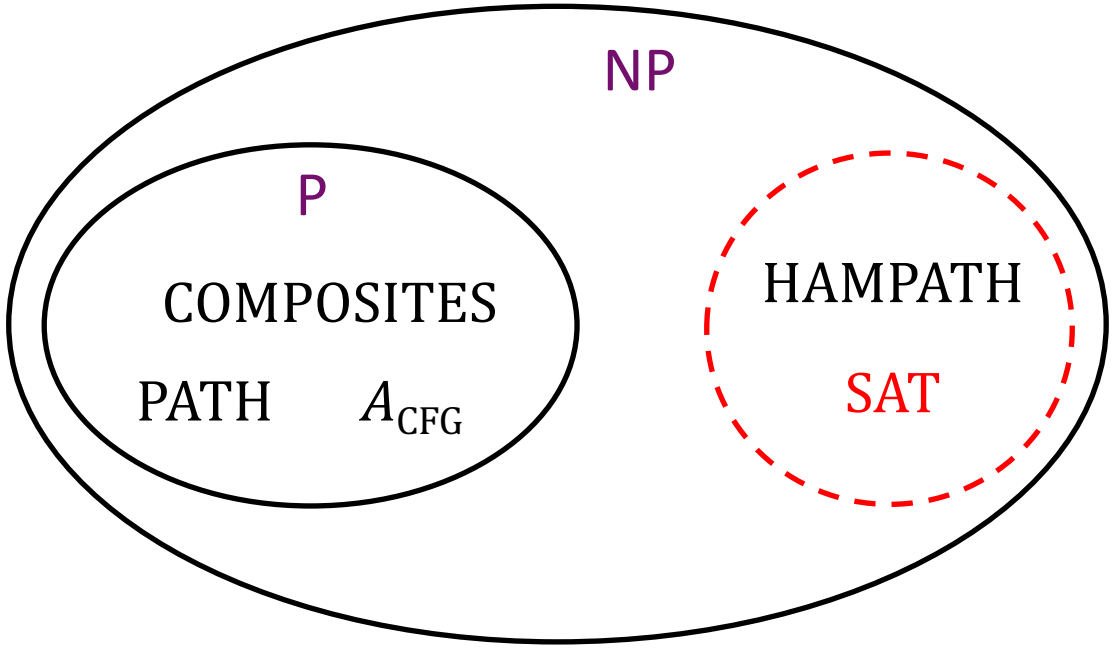
Time: $O(|G|n^3) = O(n^2)$ intervals × $O(|G|)$ rules × $O(n)$ splits.

P vs NP

NP = All languages where we can <u>verify</u> membership quickly (given a short certificate)

$(x \in L)$ (y)

P = All languages where we can <u>decide</u> membership quickly



The “most difficult” problems in NP

Satisfiability Problem (SAT)

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.

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

- $x = 1, y = 1$: $1 \quad 1 \quad 0 \quad 0 \Rightarrow \phi = 0$ (Use 1 for True and 0 for False)
- $x = 1, y = 0$: $1 \quad 0 \quad 0 \quad 1 \Rightarrow \phi = 1$

So, ϕ is satisfiable

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

Observation: $\mathrm{SAT} \in$ NP.

Theorem (Cook, Levin 1971): $\mathrm{SAT} \in$ P $\rightarrow$ P = NP
(If SAT can be solved in poly time, then all NP problems can be solved in poly time.)

Proof idea: show that any problem in NP can be reduced to SAT (in poly time).

Quick review of today

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

These slides are adapted from Michael Sipser, *18.404J Theory of Computation*, Fall 2020. Massachusetts Institute of Technology:

- Source: MIT OpenCourseWare https://ocw.mit.edu

Except for separately identified third-party material, this adapted presentation is licensed under CC BY-NC-SA 4.0.