

CS5371

Theory of Computation

Lecture 18: Complexity III
(Two Classes: P and NP)

Objectives

- Define what is the class P
- Examples of languages in P
- Define what is the class NP
- Examples of languages in NP

The Class P

Definition: **P** is the class of languages that are decidable in polynomial time on a single-tape DTM. In other words,

$$\bigcup_{k=1} \text{TIME}(n^k)$$

- **P** is invariant for all computation models that are **polynomially** equivalent to the single-tape DTM, and
- **P** roughly corresponds to the class of problems that are realistically solvable

Further points to notice

- When we describe an algorithm, we usually describe it with **stages**, just like a step in the TM, except that each stage may actually consist of many TM steps
- Such a description allows an easier (and clearer) way to analyze the running time of the algorithm

Further points to notice (2)

- So, when we analyze an algorithm to show that it runs in poly-time, we usually do:
 1. Give a polynomial upper bound on the number of stages that the algorithm uses when its input is of length n
 2. Ensure that each stage can be implemented in polynomial time on a **reasonable** deterministic model
- When the two tasks are done, we can say the algorithm runs in poly-time (why??)

Further points to notice (3)

- Since time is measured in terms of n , we have to be careful how to encode a string
- We continue to use the notation $\langle \rangle$ to indicate a **reasonable** encoding
- E.g., the graph encoding in (V,E) , DFA encoding in (Q,Σ,δ,q_0,F) , are reasonable
- E.g., to encode a number in unary, such as using 11111111111111111 to represent 17, is not reasonable since it is exponentially than any base- k encoding, for any $k > 1$

Examples of Languages in P

Let **PATH** be the language

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

Theorem: **PATH** is in P.

How to prove??

... Find a decider for **PATH** that runs in polynomial time

PATH is in P

Proof: A polynomial time decider M for $PATH$ operates as follows:

M = "On input $\langle G, s, t \rangle$,

1. Mark node s
2. Repeat until no new nodes are marked
 - i. Scan all edges of G to find an edge that has exactly one marked node.
Mark the other node
3. If t is marked, **accept**. Else, **reject**."

PATH is in P (2)

What is the running time for M ?

- Let m be the number of nodes in G
- Stages 1 and 3 each involves $O(1)$ scan of the input
- Stage 2 has at most m runs, each run checks at most all the edges of G . Thus, each run involves at most $O(m^2)$ scans of the input \rightarrow Stage 2 involves $O(m^3)$ scans
- Since $m = O(n)$, where n = input length, the total time is polynomial in n

RELPRIME is in P

Let RELPRIME be the language

$\{ \langle x, y \rangle \mid x \text{ and } y \text{ are integers, } \gcd(x, y) = 1 \}$

Theorem: RELPRIME is in P.

How to prove??

... Let's try this ...

RELPRIME is in P (2)

Proof (?): Let M be the following decider for RELPRIME:

M = "On input $\langle x, y \rangle$,

1. Let $z = \min \{x, y\}$

2. Repeat for $k = 2, 3, 4, \dots, z$

if k divides both x and y , reject;

3. If no k can divide both x and y , accept"

Quick Quiz: Does M run in polynomial time?

... No, so the proof is not correct...

RELPRIME is in P (3)

Proof: Let E (Euclidean algorithm) be the following decider for RELPRIME:

E = "On input $\langle x, y \rangle$,

1. If $x < y$, exchange x and y
2. Repeat until $y = 0$
 - i. Assign $x \leftarrow x \bmod y$
 - ii. Exchange x and y
3. If $x = 1$, accept. Else, reject."

Question: What is the running time of E ?

RELPRIME is in P (4)

- Stage 1 and Stage 3 is run once
- Each run of Stage 2 reduces the value of x at least by half \rightarrow number of runs of Stage 2 is $O(z)$, with $z = \log x + \log y$
- Each run in the above stages requires arithmetic operations, which takes time polynomial in the encoding of operands \rightarrow polynomial in z
- Total running time is polynomial in z
- Since $z = O(n)$, RELPRIME is in P

Correctness

In the previous slides, let x_i and y_i be the values of the x and y when we run Stage 2 the i^{th} time. We claim that x_{end} becomes 1 in the end if and only if the original x_0 and y_0 are relatively prime

Proof: We show $\gcd(x_k, y_k) = \gcd(x_{k+1}, y_{k+1})$ for all $k = 0, 1, \dots, \text{end}-1$. If this is true, $\gcd(x_0, y_0) = \dots = \gcd(x_{\text{end}}, 0) = x_{\text{end}}$, so that our claim is correct.

Correctness (2)

First, any common divisor of x_k and y_k must divide both x_{k+1} [which is y_k] and y_{k+1} [which is $x_k \bmod y_k$]. This implies that

$$\gcd(x_k, y_k) \leq \gcd(x_{k+1}, y_{k+1}).$$

Second, any common divisor of x_{k+1} and y_{k+1} must divide both x_k [which is equal to $r * x_{k+1} + y_{k+1}$ for some r] and y_k [which is x_{k+1}]. This implies

$$\gcd(x_k, y_k) \geq \gcd(x_{k+1}, y_{k+1}).$$

Every CFL is in P

Theorem: Every context-free language is in P.

How to prove??

... Let's recall an old idea for deciding a particular CFL ...

Every CFL is in P (2)

Proof(?): Let C be the CFL and G be the CFG in Chomsky Normal form that generates C . Define M as follows:

M = "On input $w = w_1 w_2 \dots w_n$,

1. Construct all possible derivations in G with $2n-1$ steps
2. If any derivation generates w , accept. Else, reject."

Quick Quiz: Does M run in polynomial time?

Every CFL is in P (3)

Proof: Let C be the CFL and $G = (V, T, S, R)$ be the CFG in Chomsky Normal form that generates C . Define D as follows:

- D = "On input $w = w_1 w_2 \dots w_n$,
1. If $w = \varepsilon$ and $S \rightarrow \varepsilon$ is a rule, accept
 2. Repeat for $k = 1, 2, \dots, n$
 - i. For each substring w' of w of length k , find all variables that generate w'
 3. If S generates w , accept. Else, reject."

Every CFL is in P (4)

More on Stage 2:

Repeat for $k = 1, 2, \dots, n$

- i. For each substring w' of w of length k , find all variables that generate w'

In order to perform this stage efficiently, we use the dynamic programming idea:

- For $k = 1$, we do this by brute force
- For each $k = 2, 3, \dots, n$, we do this based on the results up to length $k-1$

Every CFL is in P (5)

We shall store an $n \times n$ table such that the entry (i,j) stores the possible variables that can generate $w_i w_{i+1} \dots w_j$

When $k = 1$, we do:

For each substring w' of w of length 1, find all variables that generate w'

So, for each i , we scan the rules in R of the form $A \rightarrow b$ to fill in the entry (i,i)

Every CFL is in P (6)

When $k = 2, 3, \dots, n$, we do:

For each substring w' of w of length k , find all variables that generate w' (based on the result of length $1, 2, \dots, k-1$)

So, for each i , we scan the rules in R of the form $A \rightarrow BC$, and see if there exists x (between i and $i+k-1$) with B is in (i, x) and C is in $(x+1, i+k-1)$. If so, add A in the entry $(i, i+k-1)$

Every CFL is in P (7)

What is the running time for Stage 2?

- Let v and r be the number of variables and number of rules of G , which are both fixed constant independent of the input w
 - We need to compute $n \times n$ entries in the table (each entry has at most v variables)
 - Each entry is computed by scanning all the rules, and for each rule, scanning the table at most $O(n)$ times
- Total scans to complete table = $O(n \times n \times r \times n \times v) = O(n^3)$

Every CFL is in P (7)

- As each scan (either the table or the rules) takes time polynomial to the input, Stage 2 takes polynomial time
- Also, the other stages take polynomial time (constant number of scans)

Thus we can decide $C = L(G)$ in polynomial time, so that C is in P

The Class NP

Definition: A **verifier** for a language **A** is an algorithm **V**, where

$$A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \}$$

A **polynomial-time verifier** is a verifier that runs in time polynomial in the length of the input **w**.

The Class NP

A language A is polynomially verifiable if it has a polynomial time verifier.

Definition: NP is the class of language that is polynomially verifiable.

Examples of Languages in NP

Let **HAMILTON** be the language

$\{ \langle G \rangle \mid G \text{ is a Hamiltonian graph} \}$

Theorem: **HAMILTON** is in NP.

How to prove?? ... Define a polynomial time verifier V , and for each $\langle G \rangle$ in **HAMILTON**, define a string c , and show $\{ \langle G \rangle \mid V \text{ accepts } \langle G, c \rangle \} = \text{HAMILTON}$

HAMILTON is in NP

Proof: Define a TM V as follows:

V = "On input $\langle G, c \rangle$,

1. If c is a cycle in G that visits each vertex once, **accept**
2. Else, **reject**."

- Note: V runs in time polynomial in length of $\langle G \rangle$ (why?)
- To show **HAMILTON** is in NP, it remains to show V is a verifier for **HAMILTON**

HAMILTON is in NP (2)

To show V is a verifier, we let $H = \{ \langle G \rangle \mid V \text{ accepts } \langle G, c \rangle \}$, and show $H = \text{HAMILTON}$

For every $\langle G \rangle$ in H , there is some c that V accepts $\langle G, c \rangle$. This implies $\langle G \rangle$ is a Hamiltonian graph, and $H \subseteq \text{HAMILTON}$

For every $\langle G \rangle$ in HAMILTON , let c be one of the hamilton cycle in the graph. Then, V accepts $\langle G, c \rangle$, and so $\text{HAMILTON} \subseteq H$

Examples of Languages in NP (2)

Let **COMPOSITE** be the language

$\{ x \mid x \text{ is a composite number} \}$

Theorem: **COMPOSITE** is in NP.

How to prove?? ... Define a polynomial time verifier V , and for each x in **COMPOSITE**, define a string c , and show that $\{ x \mid V \text{ accepts } \langle x, c \rangle \} = \text{COMPOSITE}$

COMPOSITE is in NP

Proof: Define a TM V as follows:

V = "On input $\langle x, c \rangle$,

1. If c is not 1 or x , and c divides x ,
accept
2. Else, reject."

- Note: V runs in time polynomial in length of $\langle x \rangle$ (why?)
- To show COMPOSITE is in NP, it remains to show V is a verifier for COMPOSITE

COMPOSITE is in NP (2)

To show V is a verifier, we let $C = \{ x \mid V \text{ accepts } \langle x, c \rangle \}$, and show $C = \text{COMPOSITE}$

For every x in C , there is some c that V accepts $\langle x, c \rangle$. This implies x is a composite number, and $C \subseteq \text{COMPOSITE}$

For every x in COMPOSITE , let c be one of the divisor of x with $1 < c < x$. Then, V accepts $\langle x, c \rangle$, and so $\text{COMPOSITE} \subseteq C$

Next Time

- More on NP
- The class NP-Complete
 - Containing the “most difficult” problems in NP
- Proving a problem is in NP-Complete