BU CS 332 – Theory of Computation

Lecture 22:
- P Examples
- NP

Reading:
Sipser Ch 7.2-7.3

Mark Bun
November 29, 2022

https://forms.gle/kYjWMSiDbtAGjCGy5
Complexity class **P**

**Definition:** $P$ is the class of languages decidable in polynomial time on a basic single-tape (deterministic) TM

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

- Class doesn’t change if we substitute in another reasonable deterministic model (Extended Church-Turing)
- **Cobham-Edmonds Thesis:** Roughly captures class of problems that are feasible to solve on computers
Examples of languages in $P$

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

Idea: Breadth-first search

Assume $G$ presented as adjacency matrix

```
\begin{array}{c|c}
\text{Input Length:} & \text{Total run time:} \\
111^2 + \log 111 + \log 111 & 111 \cdot O(111^3) = O(111^3) \\
\end{array}
```

"On input $\langle G, s, t \rangle$:

1. Mark start vertex $s = 111^2 \cdot 2 \log 111 = \infty$

2. For $i = 1, 2, \ldots, |V|$: runs for $111$ steps

3. Mark all neighbors of currently marked vertices $O(111^2)$

4. If $t$ is marked, accept. Else, reject."

Correctness: If $G$ an $s$-$t$ path in $G$, $\exists$ such a path using $\leq 111$ vertices

Then $t$ gets marked after $111$ steps of BFS $\Rightarrow$ alg. accepts.

If no $s$-$t$ path, then $t$ will never get marked, so alg. rejects.
Examples of languages in P

\[ E_{DFA} = \{ (D) \mid D \text{ is a DFA that recognizes the empty language} \}\]

\[ E_{DFA} \in P \]

Because BFS solves problem in polynomial time

Can also reduce to PATH:

For all accept states \( t \) of \( D \):

Check if \( (D, s, t) \in \text{PATH} \). If any are \( \text{reject} \), \( \text{reject} \). If all are not, \( \text{accept} \).
Examples of languages in $\mathbf{P}$

- $RELPRIME = \{ \langle x, y \rangle \mid x \text{ and } y \text{ are relatively prime} \}$
  
  $\text{Euclidean algorithm}$

- $PRIMES = \{ \langle x \rangle \mid x \text{ is prime} \}$

2006 Gödel Prize citation

The 2006 Gödel Prize for outstanding articles in theoretical computer science is awarded to Manindra Agrawal, Neeraj Kayal, and Nitin Saxena for their paper "PRIMES is in $\mathbf{P}$."

In August 2002 one of the most ancient computational problems was finally solved....
A polynomial-time algorithm for \textit{PRIMES}?

Consider the following algorithm for \textit{PRIMES}:

On input $\langle x \rangle$: $x$ is written in binary $n = 1 < x > 1$

For $b = 2, 3, 4, 5, \ldots, \sqrt{x}$:
- Try to divide $x$ by $b$
- If $b$ divides $x$, reject

If all $b$ fail to divide $x$, accept

How many divisions does this algorithm require in terms of $n = |\langle x \rangle|$?

a) $O(\sqrt{n})$  
 b) $O(n)$  
 c) $2^{O(\sqrt{n})}$  
 d) $2^{O(n)}$

\[ \text{Number of divisions} \approx \sqrt{2^n} = (2^n)^{\frac{1}{2}} = 2^{\frac{n}{2}} \]
Beyond polynomial time

Definition: EXP is the class of languages decidable in exponential time on a basic single-tape (deterministic) TM

$$\text{EXP} = \bigcup_{k=1}^{\infty} \text{TIME}(2^{n^k})$$

$$= \text{TIME}(2^n) \cup \text{TIME}(2^{n^2}) \cup \text{TIME}(2^{n^3}) \cup \ldots$$
Why study P?

Criticism of the Cobham-Edmonds Thesis:
- Algorithms running in time $n^{100}$ aren’t really efficient
  
  **Response:** Runtimes improve with more research

- Does not capture some physically realizable models using randomness, quantum mechanics

  **Response:** Randomness may not change P, useful principles

"Hardness vs. randomness": If “hard” functions exist, then all randomized algs. can be “de-randomized” w/ only slowdown

$TIME(n) \text{ vs. } TIME(n^2)$  

$P \text{ vs. } EXP$

decidable vs. undecidable
Nondeterministic Time and NP
Extended Church-Turing Thesis

Every “reasonable” (physically realizable) model of computation can be simulated by a basic, single-tape TM with only a polynomial slowdown.

E.g., doubly infinite TMs, multi-tape TMs, RAM TMs
Does not include nondeterministic TMs (not reasonable)
Nondeterministic time

Let \( t: \mathbb{N} \rightarrow \mathbb{N} \)

A NTM \( M \) runs in time \( t(n) \) if on every input \( w \in \Sigma^n \), \( M \) halts on \( w \) within at most \( t(n) \) steps on every computational branch.
Deterministic vs. nondeterministic time
Deterministic vs. nondeterministic time

**Theorem:** Let \( t(n) \geq n \) be a function. Every NTM running in time \( t(n) \) has an equivalent single-tape TM running in time \( 2^{O(t(n))} \)

**Proof:** Simulate NTM by 3-tape TM

---

Finite control

Input \( w \) to \( N \) (read-only)

Simulation tape (run \( N \) on \( w \) using nondeterministic choices from tape 3)

Address in computation tree
Counting leaves

What is an upper bound on the maximum number of nodes in a tree with branching factor $b$ and depth $t$?

a) $bt$

b) $b^t$

c) $t^b$

d) $2^t$

\[
\begin{align*}
&1 + b + b^2 + \ldots + b^{t-1} \\
&= \frac{b^t - 1}{b-1} \leq b^t
\end{align*}
\]
Deterministic vs. nondeterministic time

**Theorem:** Let \( t(n) \geq n \) be a function. Every NTM running in time \( t(n) \) has an equivalent single-tape TM running in time \( 2^{O(t(n))} \)

**Proof:** Simulate NTM by 3-tape TM

- # nodes: \( \leq b^{t(n)} \) where \( b = \max \) # of nondet. choices NTM can make in any step.

Running time:

To simulate one root-to-node path:

\[ \leq O(t(n)) \]

Total time:

\[ b^{t(n)} \cdot O(t(n)) = 2^{t(n) \log b + O(\log t(n))} = 2^{O(t(n))} \]
Deterministic vs. nondeterministic time

Theorem: Let $t(n) \geq n$ be a function. Every NTM running in time $t(n)$ has an equivalent single-tape TM running in time $2^{O(t(n))}$

Proof: Simulate NTM by 3-tape TM in time $2^{O(t(n))}$

We know that a 3-tape TM can be simulated by a single-tape TM with quadratic overhead, hence we get running time

$$2^{O(t(n))} \cdot 2^{O(t(n))} = 2^{2 \cdot O(t(n))} = 2^{O(t(n))}$$

for some $f(n) = O(n)$
Difference in time complexity

Extended Church-Turing Thesis:
At most polynomial difference in running time between all (reasonable) deterministic models

At most exponential difference in running time between deterministic and nondeterministic models
Nondeterministic time

Let $f : \mathbb{N} \to \mathbb{N}$

A NTM $M$ runs in time $f(n)$ if on every input $w \in \Sigma^n$, $M$ halts on $w$ within at most $f(n)$ steps on every computational branch.

$\text{NTIME}(f(n))$ is a class (i.e., set) of languages:

A language $A \in \text{NTIME}(f(n))$ if there exists an NTM $M$ that

1) Decides $A$, and
2) Runs in time $O(f(n))$
NTIME explicitly

A language $A \in \text{NTIME}(f(n))$ if there exists an NTM $M$ such that, on every input $w \in \Sigma^*$

1. Every computational branch of $M$ halts in either the accept or reject state within $f(|w|)$ steps

2. If $w \in A$, then there exists an accepting computational branch of $M$ on input $w$

3. If $w \notin A$, then every computational branch of $M$ rejects on input $w$
Complexity class NP

**Definition:** NP is the class of languages decidable in polynomial time on a nondeterministic TM

\[ NP = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k) \]

Which of the following are definitely true about NP?

- a) \( P \subseteq NP \)
- b) \( NP \subseteq P \)
- c) \( NP \not\subseteq P \)
- d) \( NP \subseteq \text{EXP} \)
- e) \( \text{EXP} \subseteq NP \)
Hamiltonian Path

$HAMPATH = \{ \langle G, s, t \rangle \mid G$ is a directed graph and there is a path from $s$ to $t$ that passes through every vertex exactly once $\}$
**HAMPATH ∈ NP**

The following nondeterministic algorithm decides HAMPATH in polynomial time:

- Let $k = |V|$ (Input length: $k^2$)

On input $\langle G, s, t \rangle$: (Vertices of $G$ are numbers 1, ..., $k$)

1. **Nondeterministically** guess a sequence $c_1, c_2, \ldots, c_k$ of numbers 1, ..., $k$
2. Check that $c_1, c_2, \ldots, c_k$ is a permutation: Every number 1, ..., $k$ appears exactly once
3. Check that $c_1 = s$, $c_k = t$, and there is an edge from every $c_i$ to $c_{i+1}$
4. Accept if all checks pass, otherwise, reject.
Analyzing the algorithm

Need to check:

1) Correctness

If \((G,s,t) \in \text{HAMPATH}\), \(\exists\) a permutation \(c_1, \ldots, c_n\) forming a path from \(s\) to \(t\). So branch of computation on which \(c\) was guessed leads to accept.

If \((G,s,t) \notin \text{HAMPATH}\), no sequence \(c_1, \ldots, c_n\) represents a valid Hamiltonian path \(\Rightarrow\) alg. rejects.

2) Running time

\[
\begin{align*}
\text{Step 1: } & \quad O(n \log n) \quad \text{ (guess \(k\) \((\log k)\)-bit numbers)} \\
\text{Step 2: } & \quad O((n \log n)^2) \quad \text{ (repeatedly scan)} \\
\text{Step 2.5: } & \quad O((n \log n) \cdot n^2) \\
\text{Total time: } & \quad O(n^3 \log n) \quad \text{ (polynomial in input length \(n^2\))}
\end{align*}
\]
An alternative characterization of \textbf{NP}:

“Languages with polynomial-time verifiers”

How did we design an NTM for HAMPATH?

- Given a candidate path, it is easy (poly-time) to check whether this path is a Hamiltonian path.
- We designed a poly-time NTM by nondeterministically guessing this path and then checking it.
- Lots of problems have this structure (CLIQUE, 3-COLOR, COMPOSITE,...). They might be hard to solve, but a candidate solution is easy to check.
An alternative characterization of \textbf{NP}

“Languages with polynomial-time verifiers”

A \textbf{verifier} for a language \( L \) is a \textbf{deterministic} algorithm \( V \) such that \( w \in L \) iff there \textbf{exists} a string \( c \) such that \( V(\langle w, c \rangle) \) accepts

Running time of a verifier is only measured in terms of \(|w|\)

\( V \) is a \textbf{polynomial-time verifier} if it runs in time polynomial in \(|w|\) on every input \( \langle w, c \rangle \)

(Without loss of generality, \(|c|\) is polynomial in \(|w|\), i.e., \(|c| = O(|w|^k)\) for some constant \( k \))
**HAMPATH** has a polynomial-time verifier

Certificate $c$:

Verifier $V$:

On input $\langle G, s, t; c \rangle$: (Vertices of $G$ are numbers 1, ..., $k$)

1. Check that $c_1$, $c_2$, ..., $c_k$ is a permutation: Every number 1, ..., $k$ appears exactly once

2. Check that $c_1 = s$, $c_k = t$, and there is an edge from every $c_i$ to $c_{i+1}$

3. **Accept** if all checks pass, otherwise, **reject**.
NP is the class of languages with polynomial-time verifiers

**Theorem:** A language $L \in \text{NP}$ iff there is a polynomial-time verifier for $L$