P and NP

Peter Mayr

Computability Theory, April 5, 2021

Complexity of problems

Our definition of the complexity of TMs (that always halt) can be extended to their (computable) languages.

Definition

$$\mathrm{DTIME}(t(n)) := \{L(M) \;:\; M \text{ is a DTM with running time } O(t(n))\}$$

$$\mathrm{NTIME}(t(n)) := \{L(M) \;:\; M \text{ is a non-deterministic TM with running time } O(t(n))\}$$

Common complexity classes:

Definition

$\mathrm{P} := \mathrm{DTIME}(n^{O(1)}) = \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n^k)$

$\mathrm{NP} := \mathrm{NTIME}(n^{O(1)}) = \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(n^k)$

$\mathrm{EXPTIME} := \mathrm{DTIME}(2^{n^{O(1)}}) = \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(2^{n^k})$

$\mathrm{NEXPTIME} := \mathrm{NTIME}(2^{n^{O(1)}}) = \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(2^{n^k})$

Graphs

Definition

- A **directed graph** (**digraph**) $G = (V, E)$ is a set $V = \{1, \ldots, n\}$ of **vertices** with a binary relation E (**edges**) on V.
- The **adjacency matrix** of G is the $n \times n$-matrix $(a_{ij})_{1 \le i,j \le n}$ with
$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E, \\ 0 & \text{else.} \end{cases}$$
- $v_0 \to v_1 \to \cdots \to v_k$ is a (directed) **path** in G from vertices v_0 to v_k if $(v_i, v_{i+1}) \in E$ for all $i \in \{0, \ldots, k-1\}$.

Example

Digraph $G = (\{1,2,3\}, \{(1,2),(2,3),(1,3)\})$ has adjacency matrix
$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Examples in P

Reachability (Path)

Input: digraph $G = (V, E)$ with vertices $\{1, \dots, n\}$ by its adjacency matrix
Question: Is there a path in G from 1 to n?

Brute force: Enumerating all paths in G is $O(n^n)$.

Algorithm (Enumerate all vertices reachable from 1)

1. $R := \{1\}$ . . . vertices reachable from 1
   $B := \{1\}$ . . . boundary of the currently reachable set
2. For $i \in B$ do
3. $\quad B := B \setminus \{i\}$
4. $\quad$ For $j \in \{1, \dots, n\}$ with $(i, j) \in E$ do
5. $\quad\quad$ If $j \notin R$, then $R := R \cup \{j\}, B := B \cup \{j\}$.
6. Return $n \in R$.

Correctness: The algorithm enumerates all vertices that are reachable from 1 into the set R. Hence it returns the correct answer in 6.

Input size: n^2 for the adjacency matrix

Running time:

- Loops in 2. and 4. are executed at most n times each.
- Updating R, B in 5. is polynomial in n.
- Hence the total running time is polynomial in n.

Question
What's the space complexity of the previous algorithm?

Theorem
Reachability is in P.

Regular languages

Theorem

Every regular language is in P.

Proof.

Any regular language is decided by some DFA whose running time is equal to the length of the input. □

Examples in NP

Hamiltonian cycle (Traveling Salesman)

Input: digraph $G = (V, E)$ with vertices $\{1, \ldots, n\}$ by adjacency matrix
Question: Is there a cyclic path in G visiting each vertex exactly once?

Brute force: Enumerating all cycles in G and checking whether one is Hamiltonian is $O(n^n)$.

Non-deterministic TM N (with several tapes)

1. **Guess:** N non-deterministically writes n numbers from $\{1, \ldots, n\}$ (on tape 2).
2. **Verify:** N checks whether these numbers represent a Hamiltonian cycle (on tape 3).

Correctness:

- If G has a Hamiltonian cycle, then one computational branch of N will find it in 1. and accept in 2.
- If G has no Hamiltonian cycle, then all computational branches of N will reject.

Input size: n^2 for the adjacency matrix

Running time:

- In 1. a list of n numbers is written in $O(n \log(n))$ steps.
- In 2. check that
  - any given vertex i has not appeared before
  - any i and its successor j are connected by E.
- Hence the total running time is polynomial in n.

Theorem

Hamiltonian cycle is in NP.

Note

- Not known whether Hamiltonian cycle is in P.
- Deterministic algorithm using dynamic programming runs in $O(n^2 2^n)$ (Bellman, Held, Karp 1962).

Verification

Guessing and verifying is the typical structure of a non-deterministic algorithm.

Definition

A **verifier** for a language L is a DTM V such that

$$L = \{x \ : \ \mathrm{V} \text{ accepts } (x, c) \text{ for some string } c\}.$$

Here c is a **certificate** (witness, proof of membership) that allows to verify $x \in L$.
A **polynomial time verifier** is a DTM that runs in polynomial time in $|x|$.

Note

- For a polynomial time verifier V we may assume that the certificate c for any x has polynomial length in $|x|$ since V cannot access more of c anyway.
- If $x \in L$, the verifier V does need to accept (x, c) for all c.
- A verifier V for L does not need to verify $x \notin L$.

Example

- A certificate c for a digraph G having a Hamiltonian cycle is just the sequence of vertices forming a Hamiltonian cycle.
- Clearly such a c is polynomial in $|G|$ and can be verified in polynomial time.
- What is a certificate to show that G does not have a Hamiltonian cycle?

Theorem

NP is the class of languages that have polynomial time verifiers.

Proof.

$\subseteq$: Let $L \in \mathrm{NP}$ be decided by non-deterministic polytime N.
Construct a polytime verifier V:

- If $x \in L$, let c denote the sequence of choices of N in an accepting branch for x (such c of polynomial size must exist).
- On input (x, c), V simulates N's computation on the branch c (runs in polytime in $|x|$).
- V accepts (x, c) if N accepts x on the branch c; else V rejects.

$\supseteq$: Assume L has a verifier V running in time $\leq |x|^k$.
Construct a non-deterministic N that decides L in polytime:

- On input x, N guesses a certificate c of length $\leq |x|^k$.
- Run V on input (x, c) and accept if V accepts; else N rejects.

Note: The existence of k suffices to prove the existence of N (we don't need to know the actual value). $\square$