Time Complexity

CS60001: Foundations of Computing Science

Pallab Dasgupta
Professor, Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur

Measuring Complexity

Definition

- Let *M* be a deterministic Turing machine that halts on all inputs. The *running time* or *time complexity* of *M* is the function $f: \mathcal{N} \rightarrow \mathcal{N}$, where $f(n)$ is the running time of *M*, we say that *M* runs in time $f(n)$ and that *M* is an $f(n)$ time Turing machine. Customarily we use n to represent the length of the input

Complexity Analysis

- Worst-case Analysis
  - Longest running time of all inputs of a particular length
- Average-case Analysis
  - Average of all the running times of inputs of a particular length

Big-O and Small-o Notations

- **Asymptotic Upper Bound (O)**
  - Let f and g be functions $f, g: \mathcal{N} \rightarrow \mathcal{R}^+$. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$

$$f(n) \leq c.g(n)$$

  - When $f(n) = O(g(n))$ we say that $g(n)$ is an upper bound for $f(n)$, or more precisely, that $g(n)$ is an asymptotic upper bound for $f(n)$, to emphasize that we are suppressing constant factors

- **Asymptotic Strict-Upper Bound (o)**
  - Let f and g be functions $f, g: \mathcal{N} \rightarrow \mathcal{R}^+$. Say that $f(n) = o(g(n))$ if

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

  - In other words, f(n) = o(g(n)) means that, for any real number $c > 0$, a number n_0 exist, where $f(n) < c.g(n)$ for all $n \geq n_0$

Analyzing Algorithms

- Let *t*: $\mathcal{N} \rightarrow \mathcal{R}^+$ be a function. Define the *time complexity class*, *TIME(t(n))*, to be the collection of all languages that are decidable by an *O(t(n))* time Turing machine

- Example
  - Analyze the TM algorithm for the language $\mathcal{A} = \{0^k1^k \mid k \geq 0\}$
  - There can be different TM constructions (M_1, M_2, M_3) deciding the language [see Sipser's Book, pp. 207-209]
  - The total-time taken by them is different
    - M_1 decides $\mathcal{A}$ in time $O(n^2)$, therefore $\mathcal{A} \in TIME(n^2)$
    - M_2 decides $\mathcal{A}$ in time $O(n\log n)$, therefore $\mathcal{A} \in TIME(n.\log n)$
    - M_3 decides $\mathcal{A}$ in time $O(n)$, therefore $\mathcal{A} \in TIME(n)$

Complexity Relationships among Models

- **Definition**
  - Let N_{TM} be a non-deterministic Turing machine that is a decider. The running time of N_{TM} is the function $f: \mathcal{N} \rightarrow \mathcal{N}$, where $f(n)$ is the maximum number of steps that N_{TM} uses on any branch of its computation on any input n

- **Theorems**
  - Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multi-tape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine
  - Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time non-deterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine

The Class P (Polynomial Time)

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

$$P = \bigcup_k TIME(n^k)$$

- The role of P in theory:
  - *P* is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape Turing machine
  - *P* roughly corresponds to the class of problems that are realistically solvable on a computer
- Examples of Problems in P
  - PATH = {<G, s, t> | G is a directed graph that has a directed path from s to t}
  - RELATIVE_PRIME = {<x, y> | x and y are relatively prime}
  - Every context-free language is a member of *P*

The Class NP (Non-deterministic Polynomial Time)

- **Definitions**
  - A *verifier* for a language $\mathcal{A}$ is an algorithm *V*, where

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

    We measure the time of a verifier only in terms of the length of *w*, so a *polynomial time verifier* runs in polynomial time in the length of *w*.
  - A language $\mathcal{A}$ is *polynomially verifiable* if it has a polynomial time verifier.
  - *NP* is the class of languages that have polynomial time verifiers

- **Examples of Problems in NP**
  - HAM_PATH = {<G, s, t> | G is a directed graph with a Hamiltonian path from s to t}
  - COMPOSITES = {x | x = pq, for integers p, q > 1}
  - CLIQUE = {<G, k> | G is an undirected graph with *k*-clique}
  - SUBSET-SUM = $\{\langle S, t\rangle \mid S = \{x_1, x_2, \ldots, x_k\}$ and for some $\{y_1, y_2, \ldots, y_l\} \subseteq \{x_1, x_2, \ldots, x_k\}$, we have $\Sigma y_i = t\}$

The Class NP (contd…)

- **Theorem**
  - A language is in *NP* if and only if it is decided by some non-deterministic polynomial time Turing machine

- **Definition**
  - Non-deterministic time complexity class is defined as,

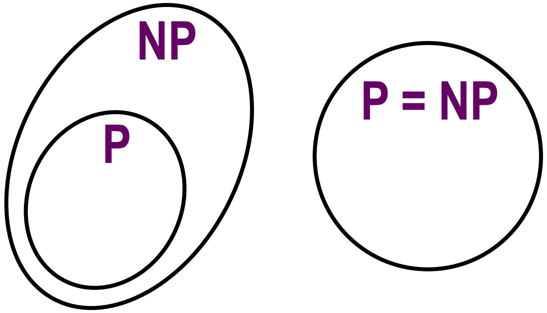
    NTIME(t(n)) = {L | L is a language decided by a O(t(n)) time non-deterministic Turing machine}

- **Corollary:** $NP = \bigcup_k NTIME(n^k)$

The P Versus NP Question

- Referring (loosely) to polynomial time solvable as solvable "quickly",
  - *P* = the class of languages for which membership can be decided quickly
  - *NP* = the class of languages for which membership can be verified quickly

- Unsolved Problem in Theoretical Computer Science
  - P = NP? OR P ≠ NP?
  - One of these two possibilities is correct ➔



- Best method known for solving languages in *NP* deterministically uses exponential time. In other words, we can prove that

$$NP \subseteq EXPTIME = \bigcup_k TIME(2^{n^k})$$

But, we do not know whether *NP* is contained in a smaller deterministic time complexity class

NP-Completeness

- **Polynomial Time Reducibility**
  - A function $f: \Sigma^* \rightarrow \Sigma^*$ is a *polynomial time computable* function if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w
  - Language $\mathcal{A}$ is *polynomial time mapping reducible*, or simply *polynomial time reducible*, to language $\mathcal{B}$, written $\mathcal{A} \leq_p \mathcal{B}$, if a polynomial time computable function $f: \Sigma^* \rightarrow \Sigma^*$ exists, where for every w,

    $$w \in \mathcal{A} \Leftrightarrow f(w) \in \mathcal{B}$$

    The function f is called the *polynomial time reduction* of A to B

- **Theorem**
  - If $\mathcal{A} \leq_p \mathcal{B}$ and $\mathcal{B} \in P$, then $\mathcal{A} \in P$
  - *3SAT* is polynomial time reducible to *CLIQUE*

NP-Completeness (contd...)

- **Definition**
  - A language $\mathcal{B}$ is *NP-complete* if it satisfies two conditions:
    - $\mathcal{B}$ is in *NP*, and
    - Every $\mathcal{A}$ in *NP* is polynomial time reducible to $\mathcal{B}$

- **Theorems**
  - If $\mathcal{B}$ is NP-complete and $\mathcal{B} \in P$, *then* $P = NP$
  - If $\mathcal{B}$ is NP-complete and $\mathcal{B} \leq_p \mathcal{C}$ for $\mathcal{C}$ in *NP*, then $\mathcal{C}$ is NP-complete

- **COOK-LEVIN's Theorem**
  - *SAT* is NP-complete (other form: $SAT \in P$ if and only if $P = NP$)
  - Corollary: *3SAT* is NP-complete

Additional NP-Complete Problems

- Examples of NP-complete Problems
  - CLIQUE = {<G, k> | G is an undirected graph with *k*-clique}
  - VERTEX-COVER = {<G, k> | G is an undirected graph that has a *k*-node vertex cover}
  - HAM_PATH = {<G, s, t> | G is a directed graph with a Hamiltonian path from s to t}
  - UHAM_PATH = Hamiltonian path in undirected graph
  - SUBSET-SUM = {<S, t> | S = $\{x_1, x_2, \ldots, x_k\}$ and for some $\{y_1, y_2, \ldots, y_l\} \subseteq \{x_1, x_2, \ldots, x_k\}$, we have $\Sigma y_i = t$}