Introduction to the Theory of Computation

- Computation Model
- Finite Automaton, Context-free Grammar
- Turing Machine (Algorithm)
- Computability
- Complexity Class (eg. P, NP, PSPACE, EXP, L, NL...)
- NP Completeness, Reduction

Big-O Notation

Def 1.1 Let

$$f, g : \mathbb{N} \to \mathbb{R}$$

1. Write

$$f = O(g) \Leftrightarrow (\exists c > 0)(\exists N)(\forall n \geq N : |f(n)| \leq c \cdot g(n))$$

2. Write

$$f = \Omega(g) \Leftrightarrow (\exists c > 0)(\exists N)(\forall n \geq N : |f(n)| \geq c \cdot g(n))$$

3. Write

$$f = \Theta(g) \Leftrightarrow f = O(g)\ and\ f = \Omega(g)$$

4. Write

$$f = o(g) \Leftrightarrow (\forall \epsilon > 0)(\exists N)(\forall n \geq N : |f(n)| \leq \epsilon \cdot g(n))$$

eg. $f(n) = 6n^4 - 2n^3 + 5 = O(n^4) = \Omega(n^4) = \Theta(n^4)$

Prop 1.2

1. $f_1 = O(g_1), f_2 = O(g_2) \Rightarrow f_1 \cdot f_2 = O(g_1 \cdot g_2)$
2. $f \cdot O(g) = O(f \cdot g)$
3. $f_1 = O(g_1), f_2 = O(g_2) \Rightarrow f_1 + f_2 = O(max(g_1, g_2))$
4. $f_1 = O(g), f_2 = O(g) \Rightarrow f_1 + f_2 = O(g)$
5. $f = O(g) \Rightarrow kf = O(g)\ where\ k\ is\ a\ constant$

Constant	$O(1)$
Double Logarithmic	$O(loglogn)$

Constant	$O(1)$
Logarithmic	$O(logn)$
Poly-logarithmic	$O(log^c n) = O(log^{O(1)} n), c > 0$
Linear	$O(n)$
Quasilinear	$O(nlog^c n), c > 0$
Quadratic	$O(n^2)$
Polynomial	$O(n^c), c > 0$
Exponential	$O(c^n), c > 1$
Factorial	$O(n!)$

Def 1.3

$$f = \omega(g) \Leftrightarrow (\forall c > 0)(\exists N)(\forall n \geq N : f(n) \geq c \cdot g(n))$$

$$f = \theta(g) \Leftrightarrow (\forall \epsilon > 0)(\exists N)(\forall n \geq N : |f(n) - g(n)| < \epsilon \cdot g(n))$$

Alphabets & Languages

Def 1.4 **An Alphabet is a set of symbols.**

Def 1.5 **A String (over an alphabet) is a finite sequence of symbols from the alphabet.**

- $\Sigma^* = \cup_{n \geq 0} \Sigma^n$
- ϵ : Empty string

eg. $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, \dots\}$

Def 1.6 **Two strings over the same alphabet can be combined by the operation of concatenation. The concatenation of "x" and "y" is denoted by "xy".**

Def 1.7 **A string "v" is a substring of "w"**$\Leftrightarrow \exists x, y \in \Sigma^*, w = xvy$

- If $w = xv$, then v is a **suffix** of w.
- If $w = vy$, then v is a **prefix** of w.

Def 1.8 **The string "w^i" is defined by:**

$$w^0 = \epsilon, w^{i+1} = w^i \cdot w, i \geq 0$$

Def 1.9 **The reversal of a string w (denoted by w^R), is the string "spelled backwards".**

Def 1.10 **A Language is a set of strings over an alphabet.**

Def 1.11 **Let L be a language. The complement of L, denoted by $\bar{L}$, is $\Sigma^* - L$. The concatenation of L_1 and L_2 is defined by:**

$$L_1L_2 = \{w \in \Sigma^* | w = xy \ for \ some \ x \in L_1 \ and \ y \in L_2\}$$

Def 1.12 **The Kleene Star of L, denoted by L^*, is the set of strings obtained by concatenating zero or more strings from L.**

$$L^* = \{w \in \Sigma^* | w = w_1w_2 \dots w_k, k \geq 0, w_1, \dots, w_k \in L\}$$

- **Write L^+ for the language LL^*. Equivalently,**

$$L^+ = \{w \in \Sigma^* | w = w_1 \dots w_k, k \geq 1, w_1, \dots, w_k \in L\}$$

Encoding of Problems

eg.

1. **Integer Multiplication**: Given 2 nonnegative integers x and y, compute xy.
2. **Primality Testing**: Given $n \in N$, decide if n is a prime number.
3. **Hamiltonian Cycle**: Given an undirected graph G, test if G has a Hamiltonian Cycle.

- **By encoding, any decision problem is a language, and any computation problem is a function from Σ^* to Σ^*.**
- **Usually, the way of encoding does not matter. By preprocessing, one can switch between encodings.**
- **eg. Adjacency matrix $\leftrightarrow$ Adjacency list**