Notes on Computability & Complexity

XIN CHEN   chenxin_hello@outlook.com $Q_{uality} = \int(K, P, t)$

latest update: December 22, 2024

May the force of P *and* NP *be with you.*

Contents

Citation testing: [1]

One of the important scientific advances in the first half of the twentieth century was that the notion of "computation" received a much more precise definition.

At roughly 50 years (1970s – 2020s), complexity theory is still an infant science, and many important results are less than 30 years old.

1 Basic concepts

1.1 logarithms

Definition 1.1 (Logarithms) Let a, b be two positive real numbers, and $a \neq 1$. *The logarithm of b to the base a*, denoted by $\log_a b$, is the unique real number x such that $a^x = b$. That is,

$$\log_a b = x \quad \text{iff} \quad a^x = b.$$

$\dashv$

The logarithm function is the inverse of the exponential function. The logarithm function is defined for $a > 0$, $a \neq 1$, and $b > 0$. The most commonly used bases are $a = 2$, $a = e$, and $a = 10$. The base $a = 2$ is used in computer science, $a = e$ is used in calculus, and $a = 10$ is used in engineering and science. The base $a = 10$ is called the *common logarithm*, and is denoted by $\lg b$.

Table 1: Commonly used bases for logarithms

Base	Notation
2	$\log b$
e	$\ln b$
10	$\lg b$

$\log_a b \quad \rightsquigarrow \quad a^{\square} = b$

$e \approx 2.718$

Table 2: Identities of logarithms

Identity	Formula
Product	$\log_b(xy) = \log_b x + \log_b y$
Quotient	$\log_b(\frac{x}{y}) = \log_b x - \log_b y$
Power	$\log_b x^r = r \cdot \log_b x$
Root	$\log_b \sqrt[r]{x} = \frac{1}{r} \cdot \log_b x$

1.2 Strings

If S is a *finite* set, called *alphabet set*, then a *string* over S is a finite ordered tuple of elements from S.

We will typically consider the *binary* alphabet $2 = \{0, 1\}$.

$S^0 = \{\epsilon\}$

$S^* = \bigcup_{n \geq 0} S^n$ is the set of all strings over S.

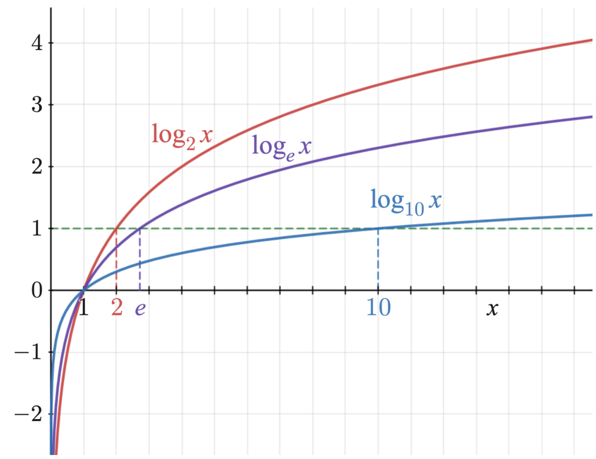

Figure 1: Plots of logarithm functions, with three commonly used bases, from wikipedia

The *concatenation* of strings x, y is denoted by $x^\frown y$, $x \circ y$, or simply xy.

x^k denotes the concatenation of k copies of x for $k \geq 1$. For example, 1^3 is '111'.

The length of a string x is denoted by $|x|$.

1.3 Representations

we implicitly identify any function f whose domain and range are not strings with the function

$$g\colon \{0,1\}^* \to \{0,1\}^*$$

that given a representation of an object x as input, outputs the representation of $f(x)$.

1.4 Big–Oh notation

Definition 1.2 (Big–Oh notation) If f, g are two functions over $\mathbb{N}$, then we say that

(1) $f = O(g)$ if there exist a constant c such that $f(n) \leq c \cdot g(n)$ for every sufficient large n.

(2) $f = \Omega(g)$ if $g = O(f)$.

(3) $f = \Theta(g)$ if $f = O(g)$ and $g = O(f)$.

(4) $f = o(g)$ if for $\forall \kappa > 0$, $f(n) \leq \kappa \cdot g(n)$ for every sufficient large n.

(5) $f = \omega(g)$ if $g = o(f)$.

To emphasize the input parameter, we often write $f(n) = O(g(n))$ instead of $f = O(g)$, and use similar notation for $o, \Theta, \Omega, \omega$. $\dashv$

Example 1.3 Here are some examples of big–Oh notation:

(1) If $f(n) = 100n \log n$ and $g(n) = n^2$, then $f = O(g)$.

(2) If $f(n) = 100n^2 + 24n + 2\log n$ and $g(n) = n^2$, then $f = O(g)$ and $g = O(f)$.

$\dashv$

2 Computational models

This section introduces some of the most important computational models in the history of computer science and explains *why it doesn't matter.*

Uncomputability has an intimate connection to Gödel's famous Incompleteness Theorem.

2.1 Turing machines

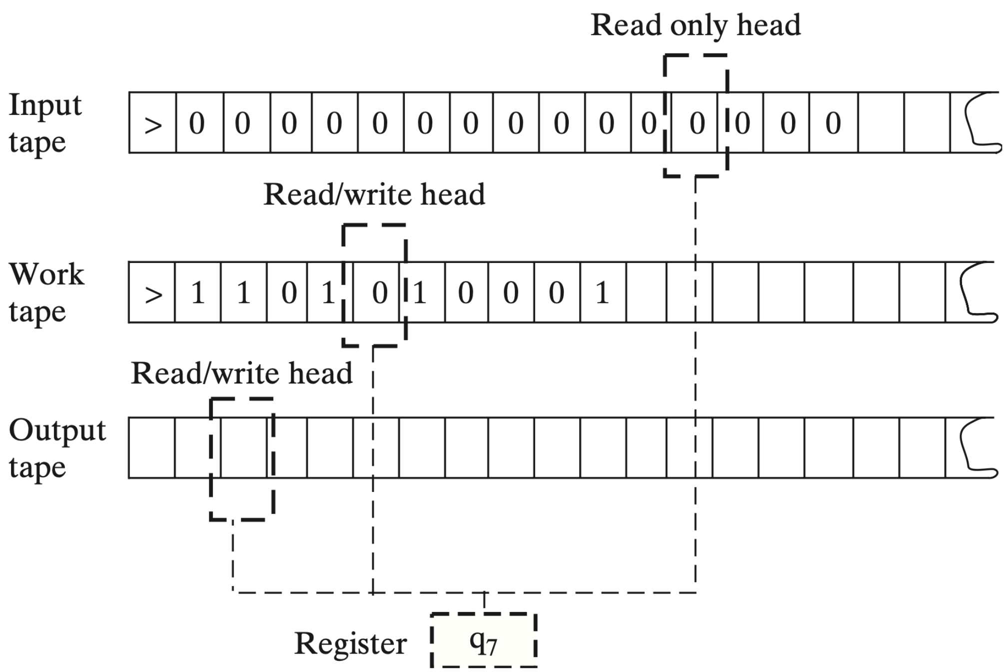

Figure 2: A snapshot of the execution of a three–tape Turing machine M with an input tape, a work tape, and an output tape, see [1, p. 11]

A *tape* is an infinite one–directional line of cells, each of which can store a symbol from a finite set called *alphabet*. Each tape is equipped with a *tape head* that can potentially read or write symbols to the tape one cell at a time.

alphabet $\{0,1\} \cup \{\sqcup, \triangleright, \}$

state set Q, contains two distinguished states: the start state q_{start} and the halting state q_{halt}

Example 2.1 (palindrome [回文]) A *palindrome* is a string that reads the same forwards and backwards. The language of palindromes over the binary alphabet is

$$PAL = \{w \in \{0,1\}^* \mid w = w^R\}$$

where w^R denotes the reverse of w. For example, 00100 is a palindrome, and clearly $\{\epsilon, 0, 1\} \subseteq PAL$.

A Turing machine that computes PAL within less than $3|x|$ steps for any input x.

Our TM M will use three tapes (the input, work and output tape) and the alphabet $\{\sqcup, \triangleright, 0, 1\}$:

Input x (where $|x| = n$)
Copy x to the work tape (n steps)
Move the input–tape head the begining of x (n steps)
Move the input-tape head to the right while moving the work-tape head to the left.
if at any moment the machine observes two different symbols **then**
  reject and output 0
else
  accept and output 1
end if ($\leq n$ steps)

$\dashv$

2.2 Running Time

Any nontrivial computational task requires at least reading the entire input, we count the number of basic steps as a function of the input length.

Definition 2.2 (Computing a function and running time) Given $f\colon \{0,1\}^* \to \{0,1\}^*$, $T\colon \mathbb{N} \to \mathbb{N}$ and a Turing machine M, we say that

(1) *M computes f* if for every $x \in \{0,1\}^*$, whenever M is initialized to the start configuration on input x, then it halts with $f(x)$ on the output tape.

(2) *M computes f in $T(n)$–time* if its computation on every input x requires at most $T(|x|)$ steps.

$\dashv$

2.3 Machines as Strings and the Universal Turing Machine

It is almost obvious that *we can represent a Turing machine as a string*: Just write the description of the TM on paper, and encode this description as a sequence of zeros and ones. This string can be given as input to another Turing machine.

2.4 Variants of Turing machines

2.4.1 Turing machines with alphabet $\{0, 1, \sqcup, \triangleright\}$

2.4.2 Single tape Turing machines

2.4.3 Bidirectional single tape Turing machines

2.5 Other computational models*

2.5.1 URM: the unlimited register machine

3 Uncomputability

There exist functions that cannot be computed within any finite amount of time.

4 Complexity Classes

4.1 Hardness and Completeness

For any complexity class $\mathcal{C}$, a decision problem D is said to be $\mathcal{C}$*–hard* if

$$\forall F \in \mathcal{C}, F \leq_p D.$$

If, in addition $D \in \mathcal{C}$ then D is said to be $\mathcal{C}$*–complete*.

So the class of NP-complete problems are those problems in NP to which any other problem in NP can be polynomially reduced.

(1) deciding if a propositional formula has a model (SAT).

(2) deciding if a graph has a path that contains every vertex exactly once (a variant of the so-called Travelling Salesperson Problem).

(3) deciding if a given argument is acceptable w.r.t. Dung's stable semantics.

5 The Class P

A *complexity class* is a set of functions that can be computed within given resource bounds.

Definition 5.1 (The class DTIME) The class $\mathsf{DTIME}(f(n))$ is the set of all functions that can be computed by a deterministic Turing machine in $O(f(n))$ steps. ⊣

The D in the notation DTIME refers to "deterministic".

Definition 5.2 (The class P) The class P is the union of all complexity classes **DTIME**(n^c) for all c. In other words,

$$\mathsf{P} = \bigcup_{c \geq 1} \mathsf{DTIME}(n^c).$$

⊣

P: polynomial.

Example 5.3 (Graph connectivity) Graph connectivity problem:

input: a graph G and two vertices s, t,

inquiry: whether there is a path from s to t.

Using *depth–first search* (DFS), we can solve this problem in $O(|V| + |E|)$ steps. ⊣

. . . .

A problem is viewed as having an efficient algorithmic solution if it can be placed into the class P (*polynomial–time*) of all problems that have a polynomial–time algorithm, i.e., an algorithm that for each instance x (of size $|x|$) produces its answer after at most $|x|^k$ steps, for a fixed constant k.

P is a rather coarse–grained class, an important subclass we will consider is L (*logarithmic space*), which consists of the problems that can be solved in logarithmic space (not counting input and output) and polynomial–time.

We consider problems in the classes L and P to be computationally *tractable*.

6 The Class NP, coNP and DP

. . . .

Quantified boolean formula (QBF)

$$Q_1 x_1 Q_2 x_2 \cdots Q_k x_k \varphi(x_1, \ldots, x_k)$$

where $Q_i \in \{\forall, \exists\}$ and $\exists$ is followed by $\forall$

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge: Cambridge University Press, 2009. xxiv+579. ISBN: 978-0-511-53381-5. URL: `https://doi.org/10.1017/CBO9780511804090` (cit. on pp. 2, 6).