

Theory of Computing

Introduction and Preliminaries

(Based on [Sipser 2006, 2013])

Yih-Kuen Tsay

Department of Information Management
National Taiwan University

What It Is

- 🌐 The central question:

What are the fundamental capabilities and limitations of computers?





- 🌐 Three main areas:

- ☀ *Automata Theory*
- ☀ *Computability Theory*
- ☀ *Complexity Theory*

Complexity Theory

- Some problems are easy and some hard.
For example, sorting is easy and scheduling is hard.
- The central question of complexity theory:
What makes some problems computationally hard and others easy?
- We don't have the answer to it.
- However, researchers have found a scheme for **classifying** problems according to their computational difficulty.
- One practical application: cryptography/security.

Options for dealing with a computationally hard problem:

-  Try to simplify it (the hard part of the problem might be unnecessary).
-  Settle for an approximate solution.
-  Find a solution that usually runs fast.
-  Consider alternative types of computation (such as randomized computation).

Computability Theory

- 🌐 **Alan Turing**, among other mathematicians, discovered in the 1930s that certain basic problems cannot be solved by computers.
- 🌐 One example is the problem of determining whether a mathematical statement is true or false.
- 🌐 Theoretical models of computers developed at that time eventually lead to the construction of actual computers.
- 🌐 The theories of computability and complexity are closely related.
- 🌐 *Complexity theory* seeks to classify problems as easy ones and hard ones, while in *computability theory* the classification is by whether the problem is solvable or not.

Automata Theory

- 🌐 The theories of computability and complexity require a **precise, formal definition** of a *computer*.
- 🌐 *Automata theory* deals with the definitions and properties of mathematical models of computation.
- 🌐 Two basic and practically useful models:
 - ☀️ *Finite-state*, or simply *finite, automaton*
 - ☀️ *Context-free grammar* (pushdown automaton)

Why You Should Learn the Subject

- 🌐 It will certainly broaden your knowledge of what computing is fundamentally.
- 🌐 Below are a few things you may find particularly useful or interesting:
 - ☀️ **Regular expressions**, in their original simplest form, for describing patterns of strings/words.
 - ☀️ **Context-free grammars** for describing the syntax of a (programming) language.
 - ☀️ The so-called **Turing machines**, as the most commonly used model for a computer.
 - ☀️ Exemplar **undecidable problems**, which cannot be (perfectly) solved by computers.
 - ☀️ **A proof of SAT being NP-hard**, where every NP problem is shown to be polynomially reducible to SAT.

- 🌐 Set, element (member), subset, proper subset
- 🌐 Multiset
- 🌐 Description of a set
- 🌐 The empty set (\emptyset)
- 🌐 Finite set, infinite set
- 🌐 Union, intersection, complement
- 🌐 Power set
- 🌐 Venn diagram

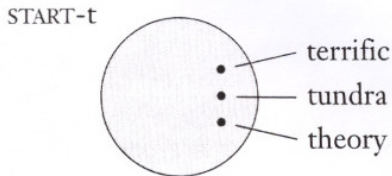


FIGURE 0.1

Venn diagram for the set of English words starting with “t”

Source: [Sipser 2006]

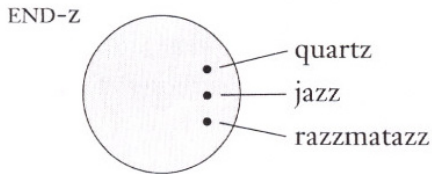


FIGURE 0.2
Venn diagram for the set of English words ending with “z”

Source: [Sipser 2006]

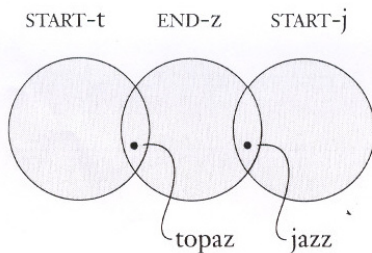


FIGURE 0.3
Overlapping circles indicate common elements

Source: [Sipser 2006]

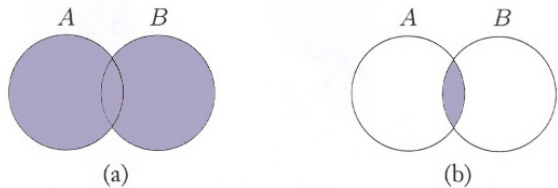


FIGURE 0.4
Diagrams for (a) $A \cup B$ and (b) $A \cap B$

Source: [Sipser 2006]

Sequences and Tuples

- 🌐 A *sequence* of objects is a list of these objects in some order. Order is essential and repetition is also allowed.
- 🌐 Finite sequences are often called *tuples*. A sequence with k elements is a k -tuple; a 2-tuple is also called a *pair*.
- 🌐 The *Cartesian product*, or cross product, of A and B , written as $A \times B$, is the set of all pairs (x, y) such that $x \in A$ and $y \in B$.
- 🌐 Cartesian products generalize to k sets, A_1, A_2, \dots, A_k , written as $A_1 \times A_2 \times \dots \times A_k$. Every element in the product is a k -tuple.
- 🌐 A^k is a shorthand for $A \times A \times \dots \times A$ (k times).

Functions

- 🌐 A *function* sets up an *input-output* relationship, where the same input always produces the same output.
- 🌐 If f is a function whose output is b when the input is a , we write $f(a) = b$.
- 🌐 A function is also called a *mapping*; if $f(a) = b$, we say that f maps a to b .

Functions (cont.)

- 🌐 The set of possible inputs to a function is called its *domain*; the outputs come from a set called its *range*.
- 🌐 A function is *onto* if it uses all the elements of the range (it is *one-to-one* if ...).
- 🌐 The notation $f : D \longrightarrow R$ says that f is a function with domain D and range R .
- 🌐 More notions and terms: *k-ary function*, *unary function*, *binary function*, *infix notation*, *prefix notation*

Relations

- 🌐 A *predicate*, or property, is a function whose range is $\{\text{TRUE}, \text{FALSE}\}$.
- 🌐 A predicate whose domain is $A_1 \times A_2 \times \dots \times A_k$ is called a *k-ary relation* on A_1, A_2, \dots, A_k . When the A_i 's are the same set A , it is simply called a *k-ary relation* on A .
- 🌐 A 1-ary relation is usually called a *unary relation* and a 2-ary relation is called a *binary relation*.

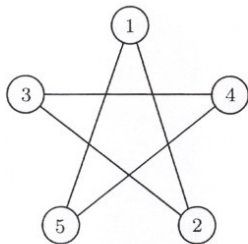
Equivalence Relations

- 🌐 An *equivalence relation* is a special type of binary relation that captures the notion of two objects being *equal* in some sense.
- 🌐 A binary relation R on A is an equivalence relation if
 1. R is *reflexive* (for every x in A , xRx),
 2. R is *symmetric* (for every x and y in A , xRy if and only if yRx),
and
 3. R is *transitive* (for every x , y , and z in A , xRy and yRz implies xRz).

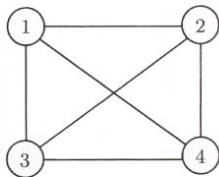
Graphs

- 🌐 Undirected graph, node (vertex), edge (link), degree
- 🌐 Description of a graph: $G = (V, E)$
- 🌐 Labeled graph
- 🌐 Subgraph, induced subgraph
- 🌐 Path, simple path, cycle, simple cycle
- 🌐 Connected graph
- 🌐 Tree, root, leaf
- 🌐 Directed graph, outdegree, indegree
- 🌐 Strongly connected graph

Graphs (cont.)



(a)



(b)

FIGURE 0.12
Examples of graphs

Source: [Sipser 2006]

Graphs (cont.)

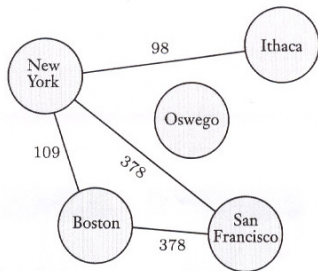


FIGURE 0.13
Cheapest nonstop air fares between various cities

Source: [Sipser 2006]

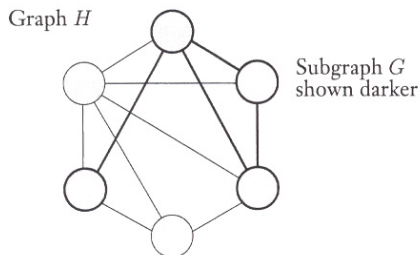


FIGURE 0.14
Graph G (shown darker) is a subgraph of H

Source: [Sipser 2006]

Graphs (cont.)

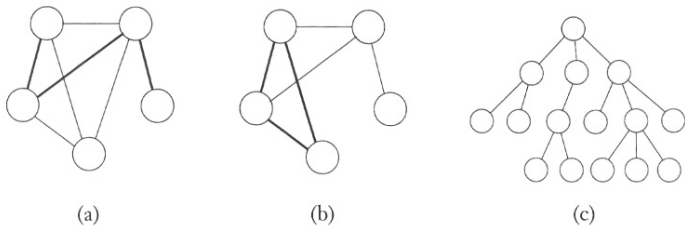


FIGURE 0.15

(a) A path in a graph, (b) a cycle in a graph, and (c) a tree

Source: [Sipser 2006]

Graphs (cont.)

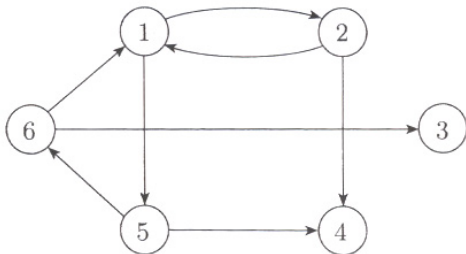


FIGURE 0.16
A directed graph

Source: [Sipser 2006]

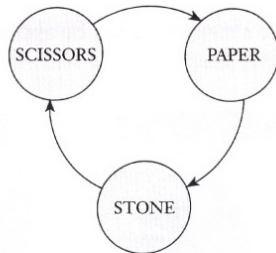




FIGURE 0.18
The graph of the relation *beats*

Source: [Sipser 2006]

Strings and Languages

- 🌐 An *alphabet* is any finite set of *symbols*.
- 🌐 A *string* over an alphabet is a finite sequence of symbols from that alphabet.
- 🌐 The *length* of a string w , written as $|w|$, is the number of symbols that w contains.
- 🌐 The string of length 0 is called the *empty string*, written as ϵ .
- 🌐 The *concatenation* of x and y , written as xy , is the string obtained from appending y to the end of x .
- 🌐 A *language* is a set of strings.
- 🌐 More notions and terms: *reverse*, *substring*, *lexicographic ordering*.

Boolean Logic

-  *Boolean logic* is a mathematical system built around the two *Boolean values* TRUE (1) and FALSE (0).
-  Boolean values can be manipulated with *Boolean operations*: *negation* or NOT (\neg), *conjunction* or AND (\wedge), *disjunction* or OR (\vee).

$$0 \wedge 0 \triangleq 0$$

$$0 \vee 0 \triangleq 0$$

$$\neg 0 \triangleq 1$$

$$0 \wedge 1 \triangleq 0$$

$$0 \vee 1 \triangleq 1$$


$$\neg 1 \triangleq 0$$

$$1 \wedge 0 \triangleq 0$$

$$1 \vee 0 \triangleq 1$$

$$1 \wedge 1 \triangleq 1$$

$$1 \vee 1 \triangleq 1$$

-  Unknown Boolean values are represented symbolically by *Boolean variables or propositions*, e.g., P , Q , etc.

Boolean Logic (cont.)

- 🌐 Additional Boolean operations: *exclusive or* or XOR (\oplus), *equality/equivalence* (\leftrightarrow or \equiv), *implication* (\rightarrow).

$$0 \oplus 0 \stackrel{\Delta}{=} 0$$

$$0 \leftrightarrow 0 \stackrel{\Delta}{=} 1$$

$$0 \rightarrow 0 \stackrel{\Delta}{=} 1$$

$$0 \oplus 1 \stackrel{\Delta}{=} 1$$

$$0 \leftrightarrow 1 \stackrel{\Delta}{=} 0$$

$$0 \rightarrow 1 \stackrel{\Delta}{=} 1$$

$$1 \oplus 0 \stackrel{\Delta}{=} 1$$

$$1 \leftrightarrow 0 \stackrel{\Delta}{=} 0$$

$$1 \rightarrow 0 \stackrel{\Delta}{=} 0$$

$$1 \oplus 1 \stackrel{\Delta}{=} 0$$

$$1 \leftrightarrow 1 \stackrel{\Delta}{=} 1$$

$$1 \rightarrow 1 \stackrel{\Delta}{=} 1$$

- 🌐 All in terms of conjunction and negation:

$$P \vee Q \equiv \neg(\neg P \wedge \neg Q)$$

$$P \rightarrow Q \equiv \neg P \vee Q$$

$$P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$$

$$P \oplus Q \equiv \neg(P \leftrightarrow Q)$$

Logical Equivalences and Laws

- 🌐 Two logical expressions/formulae are *equivalent* if each of them implies the other, i.e., they have the same truth value.
- 🌐 Equivalence plays a role analogous to equality in algebra.
- 🌐 Some laws of Boolean logic:
 - ☀ (Distributive) $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$
 - ☀ (Distributive) $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
 - ☀ (De Morgan's) $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
 - ☀ (De Morgan's) $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$

Definitions, Theorems, and Proofs

- 🌐 *Definitions* describe the objects and notions that we use. Precision is essential to any definition.
- 🌐 After we have defined various objects and notions, we usually make *mathematical statements* about them. Again, the statements must be precise.
- 🌐 A *proof* is a convincing logical argument that a statement is true. The only way to determine the truth or falsity of a mathematical statement is with a mathematical proof.
- 🌐 A *theorem* is a mathematical statement proven true. *Lemmas* are proven statements for assisting the proof of another more significant statement.
- 🌐 *Corollaries* are statements seen to follow easily from other proven ones.

Finding Proofs

🌐 Find proofs isn't always easy; no one has a recipe for it.

🌐 Below are some helpful general strategies:

1. Carefully read the statement you want to prove.
2. Rewrite the statement in your own words.
3. Break it down and consider each part separately.
For example, $P \iff Q$ consists of two parts: $P \rightarrow Q$ (the forward direction) and $Q \rightarrow P$ (the reverse direction).
4. Try to get an intuitive feeling of why it should be true.

Tips for Producing a Proof

- 🌐 A well-written proof is a sequence of statements, wherein each one follows by simple reasoning from previous statements in the sequence.
- 🌐 Tips for producing a proof:
 - ☀️ *Be patient.* Finding proofs takes time.
 - ☀️ *Come back to it.* Look over the statement, think about it, leave it, and then return some time later.
 - ☀️ *Be neat.* Use simple, clear text and/or pictures; make it easy for others to understand.
 - ☀️ *Be concise.* Emphasize high-level ideas, but be sure to include enough details of reasoning.

An Example Proof

Theorem

For any two sets A and B , $\overline{A \cup B} = \overline{A} \cap \overline{B}$.

Proof. We show that every element of $\overline{A \cup B}$ is also an element of $\overline{A} \cap \overline{B}$ and vice versa.

Forward ($x \in \overline{A \cup B} \rightarrow x \in \overline{A} \cap \overline{B}$):

$$x \in \overline{A \cup B}$$

$\rightarrow x \notin A \cup B$, def. of complement

$\rightarrow x \notin A$ and $x \notin B$, def. of union

$\rightarrow x \in \overline{A}$ and $x \in \overline{B}$, def. of complement

$\rightarrow x \in \overline{A} \cap \overline{B}$, def. of intersection

Reverse ($x \in \overline{A} \cap \overline{B} \rightarrow x \in \overline{A \cup B}$): ...

Another Example Proof

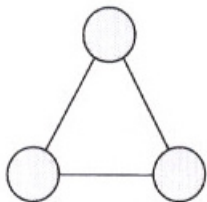
Theorem

In any graph G , the sum of the degrees of the nodes of G is an even number.

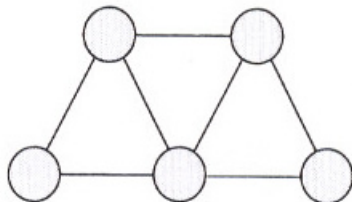
Proof.

- 🌐 Every edge in G connects two nodes, contributing 1 to the degree of each.
- 🌐 Therefore, each edge contributes 2 to the sum of the degrees of all the nodes.
- 🌐 If G has e edges, then the sum of the degrees of the nodes of G is $2e$, which is even.

Another Example Proof (cont.)



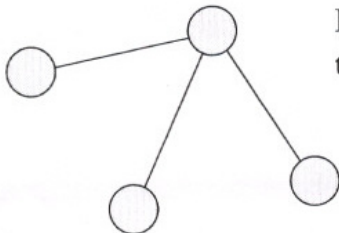
$$\begin{aligned} \text{sum} &= 2+2+2 \\ &= 6 \end{aligned}$$



$$\begin{aligned} \text{sum} &= 2+3+4+3+2 \\ &= 14 \end{aligned}$$

Source: [Sipser 2006]




Another Example Proof (cont.)



Every time an edge is added,
the sum increases by 2.

Source: [Sipser 2006]

Types of Proof

-  *Proof by construction:*
prove that a particular type of object exists, by showing how to construct the object.
-  *Proof by contradiction:*
prove a statement by first assuming that the statement is false and then showing that the assumption leads to an obviously false consequence, called a contradiction.
-  *Proof by induction:*
prove that all elements of an infinite set have a specified property, by exploiting the inductive structure of the set.

Proof by Construction

Theorem

For each even number n greater than 2, there exists a 3-regular graph with n nodes.

Proof. Construct a graph $G = (V, E)$ with $n (= 2k > 2)$ nodes as follows.

Let V be $\{0, 1, \dots, n - 1\}$ and E be defined as

$$E = \{\{i, i + 1\} \mid \text{for } 0 \leq i \leq n - 2\} \cup \{\{n - 1, 0\}\} \cup \{\{i, i + n/2\} \mid \text{for } 0 \leq i \leq n/2 - 1\}.$$

Proof by Contradiction

Theorem

$\sqrt{2}$ is irrational.

Proof. Assume toward a contradiction that $\sqrt{2}$ is rational, i.e., $\sqrt{2} = \frac{m}{n}$ for some integers m and n , which *cannot both be even*.

$\sqrt{2} = \frac{m}{n}$, from the assumption
$n\sqrt{2} = m$, multipl. both sides by n
$2n^2 = m^2$, square both sides
m is even	, m^2 is even
$2n^2 = (2k)^2 = 4k^2$, from the above two
$n^2 = 2k^2$, divide both sides by 2
n is even	, n^2 is even

Now both m and n are even, a contradiction.

Example: Home Mortgages

P : the *principle* (amount of the original loan).

I : the yearly *interest rate*.

Y : the monthly payment.

M : the *monthly multiplier* $= 1 + I/12$.

P_t : the amount of loan outstanding after the t -th month; $P_0 = P$
and $P_{k+1} = P_k M - Y$.

Theorem

For each $t \geq 0$,

$$P_t = PM^t - Y \left(\frac{M^t - 1}{M - 1} \right).$$

Proof by Induction

Theorem

For each $t \geq 0$,

$$P_t = PM^t - Y\left(\frac{M^t - 1}{M - 1}\right).$$

Proof. The proof is by induction on t .

 *Basis:* When $t = 0$, $PM^0 - Y\left(\frac{M^0 - 1}{M - 1}\right) = P = P_0$.

Proof by Induction (cont.)

🌐 *Induction step:* When $t = k + 1$ ($k \geq 0$),

$$\begin{aligned}
 & P_{k+1} \\
 = & \quad \{\text{definition of } P_t\} \\
 & P_k M - Y \\
 = & \quad \{\text{the induction hypothesis}\} \\
 & (PM^k - Y(\frac{M^k - 1}{M - 1}))M - Y \\
 = & \quad \{\text{distribute } M \text{ and rewrite } Y\} \\
 & PM^{k+1} - Y(\frac{M^{k+1} - M}{M - 1}) - Y(\frac{M - 1}{M - 1}) \\
 = & \quad \{\text{combine the last two terms}\} \\
 & PM^{k+1} - Y(\frac{M^{k+1} - 1}{M - 1})
 \end{aligned}$$

Structural Induction

- 🌐 Structural induction is a generalization of mathematical induction on the natural numbers.
- 🌐 It is used to prove that some proposition $P(x)$ holds for all x of some sort of recursively/inductively defined structure such as binary trees.

Structural Induction

- 🌐 **Structural induction** is a generalization of mathematical induction on the natural numbers.
- 🌐 It is used to prove that some proposition $P(x)$ holds for all x of some sort of **recursively/inductively defined structure** such as binary trees.
- 🌐 Proof by structural induction:
 1. Base case: the proposition holds for all the minimal structures.
 2. Inductive step: if the proposition holds for the immediate substructures of a certain structure S , then it also holds for S .