Theory of Computation
Fall 2017

Wonhong Nam

Konkuk University

September 1, 2017

Class Information

Instructor: Prof. Wonhong Nam

- Web page: http://home.konkuk.ac.kr/∼wnam
- Office: New Millennium Building 1301
- Email: wnam@konkuk.ac.kr, Tel: 2049-6128
- Office hours: Tue 2–3pm and Thur 3–4pm

Textbook: Introduction to the Theory of Computation

- Michael Sipser (MIT), 2nd Edition (2006) or 3rd Edition (2013)
- Cover to Ch.4 (p.214) and approximately 15 pages in every week.

- Students are expected to study 6–9 hours in a week.
- Course web page: http://home.konkuk.ac.kr/∼wnam/class/tc2017/
- Grade – attendance: 10%, homework: 20%, midterm exam: 35%, final exam: 35%.

Preface

"The theory of computation" comprises the fundamental mathematical properties of computer hardware, software, and certain applications.

- We seek to determine what can and cannot be computed, how quickly, with how much memory, and on which type of computational model.
- Theoretical computer science does have many fascinating big ideas, but it also has many small and sometimes dull details that can be tiresome.
- My primary objective is to expose you to the exciting aspects of computer theory, without getting bogged down in the drudgery.
- Imitation game: https://www.youtube.com/watch?v=nuPZUUED5uk
- Alan Turing: https://en.wikipedia.org/wiki/Alan_Turing
- https://ko.wikipedia.org/wiki/%EC%95%A8%EB%9F%B0_%ED%8A%9C%EB%A7%81

Preface

Theory shows you a new, simpler, and more elegant side of computers.

- The best computer designs and applications are conceived with elegance in mind.
- A theoretical course can heighten your aesthetic sense and help you build more beautiful systems.

Theory is good for you because studying it expands your mind.

- Computer technology changes quickly.
- Specific technical knowledge, though useful today, becomes outdated in just a few years.
- Consider instead the abilities to think, to express yourself clearly and precisely, to solve problems, and to know when you haven't solved a problem.
- These abilities have lasting value.

Chapter 0. Introduction
0.1 Automata, Computability, and Complexity

What are the fundamental capabilities and limitations of computers?

- This question goes back to the 1930s when mathematical logicians first began to explore the meaning of computation.
- In each of the three areas—automata, computability, and complexity—this question is interpreted differently, and the answers vary according to the interpretation.

Complexity Theory

- Computer problems come in different varieties; some are easy, and some are hard.
- For example, the sorting problem is an easy one.
- Scheduling problem: If you have just a thousand classes, finding the best (optimal) schedule may require centuries, even with a supercomputer.
- What makes some problems computationally hard and others easy?

0.1 Automata, Computability, and Complexity

Computability Theory

- During the first half of the 20th century, mathematicians such as Kurt Gödel, Alan Turing, and Alonzo Church discovered that certain basic problems cannot be solved by computers.
- In complexity theory, the objective is to classify problems as easy ones and hard ones.
- In computability theory the classification of problems is by those that are solvable and those that are not.

Automata Theory

- Automata theory deals with the definitions and properties of mathematical models of computation.
- These models play a role in several applied areas of computer science.
  - One model, called the finite automaton, is used in text processing, compilers, and hardware design.
  - Another model, called the context-free grammar, is used in programming languages and artificial intelligence.

0.2 Mathematical Notions and Terminology

Sets

- **set**: $\{7, 21, 57\}$
- **elements** or **members**: $7 \in \{7, 21, 57\}, 8 \notin \{7, 21, 57\}$
- **subset** $A \subseteq B$, **proper subset** $A \subset B$
- **multiset**: $\{7\}$ and $\{7, 7\}$ are different as multisets but identical as sets.
- **infinite set**: $\{1, 2, 3, \dots\}$
- **empty set**: $\emptyset$
- **union** $A \cup B$, **intersection** $A \cap B$, **complement** $\overline{A}$
- **Venn diagram**

Sequences and Tuples

- A **sequence** of objects is a list of these objects in some order.
- (7, 21, 57) is not the same as (57, 7, 21).
- (7, 7, 21, 57) is different from both of the other sequences, whereas the set $\{7, 21, 57\}$ is identical to the set $\{7, 7, 21, 57\}$.
- Finite sequences often are called **tuples**. A sequence with k elements is a **k-tuple**. Thus (7,21, 57) is a 3-tuple. A 2-tuple is also called an **ordered pair**.
- The **power set** of A is the set of all subsets of A.
- If A and B are two sets, the **Cartesian product** or **cross product** of A and B, written $A \times B$, is the set of all pairs wherein the first element is a member of A and the second element is a member of B.
- Example 0.5: If $A = \{1, 2\}$ and $B = \{x, y, z\}$, $A \times B = \{(1, x), (1, y), (1, z), (2, x), (2, y), (2, z)\}$.
- Example 0.6: If A and B are as in Example 0.5, $A \times B \times A = ?$

Functions and Relations

- A **function** takes an input and produces an output. $f(a) = b$
- A function also is called a **mapping**, and, if $f(a) = b$, we say that f maps a to b.
- The set of possible inputs to the function is called its **domain**.
- The outputs of a function come from a set called its **range**. $f : D \to R$.
- When the domain of a function f is $A_1 \times \cdots \times A_k$ for some sets $A_1, \ldots, A_k$, the input to f is a k-tuple $(a_1, a_2, \ldots, a_k)$ and we call the a_i the **arguments** to f.
- A function with k arguments is called a **k-ary function**, and k is called the **arity** of the function.
- If k is 1, f has a single argument and f is called a **unary function**. If k is 2, f is a **binary function**.
- A **predicate** or **property** is a function whose range is {TRUE, FALSE}. e.g., *even*().
- A property whose domain is a set of k-tuples $A_1 \times \cdots \times A_k$ is called a **relation**, a **k-ary relation**.

Functions and Relations

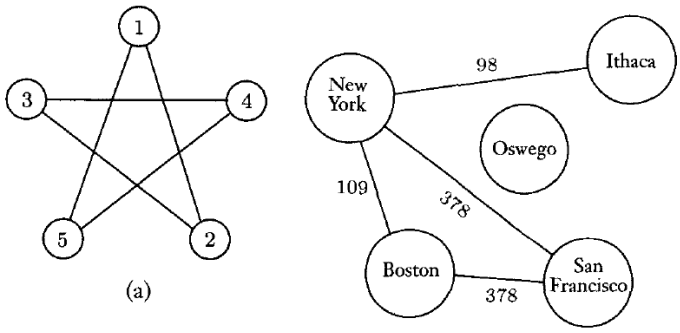
- Example 0.10 Scissors-Paper-Stone game
- In a children's game called Scissors-Paper-Stone, the two players simultaneously select a member of the set {SCISSORS, PAPER, STONE} and indicate their selections with hand signals.
- If the two selections are the same, the game starts over.
- If the selections differ, one player wins, according to the relation beats.

beats	SCISSORS	PAPER	STONE
SCISSORS	FALSE	TRUE	FALSE
PAPER	FALSE	FALSE	TRUE
STONE	TRUE	FALSE	FALSE

- Sometimes describing predicates (relations) with sets instead of functions is more convenient.
- The predicate $P : D \to \{\text{TRUE}, \text{FALSE}\}$ may be written (D, S), where $S = \{a \in D \mid P(a) = \text{TRUE}\}$, or simply S if the domain D is obvious from the context.
- Hence the relation beats may be written {(SCISSORS, PAPER), (PAPER, STONE), (STONE, SCISSORS)}.

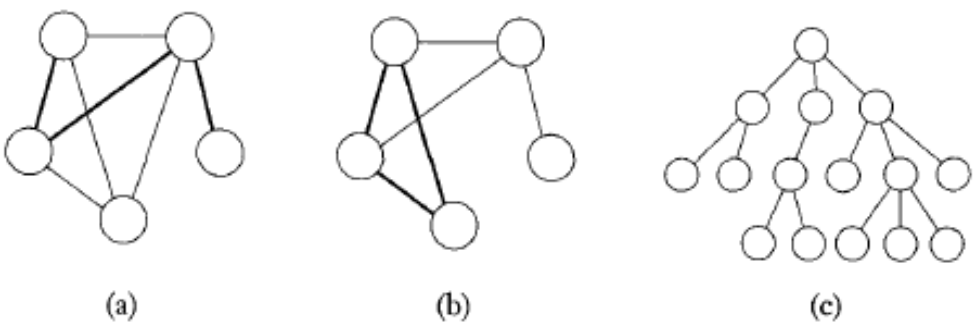
Graphs

- An **undirected graph**, or simply a **graph**, is a set of points with lines connecting some of the points.
- The points are called **nodes** or **vertices**, and the lines are called **edges**.
- The number of edges at a particular node is the **degree** of that node.
- In a graph G that contains nodes i and j, the pair (i, j) represents the edge that connects i and j.
- If V is the set of nodes of G and E is the set of edges, we say $G = (V, E)$.
- A formal description of the graph in Figure 0.12(a) is $(\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1)\})$.
- Sometimes, for convenience, we label the nodes and/or edges of a graph, which then is called a **labeled graph**.



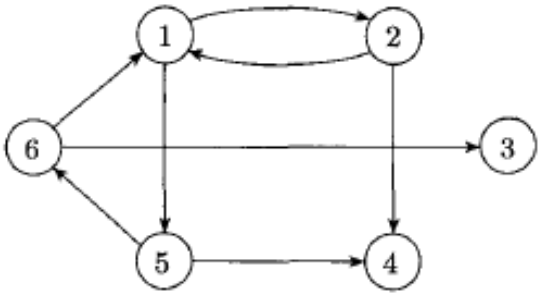
Graphs

- We say that graph G is a **subgraph** of graph H if the nodes of G are a subset of the nodes of H, and the edges of G are the edges of H on the corresponding nodes.
- A **path** in a graph is a sequence of nodes connected by edges.
- A **simple path** is a path that doesn't repeat any nodes.
- A graph is **connected** if every two nodes have a path between them.
- A path is a **cycle** if it starts and ends in the same node.
- A **simple cycle** is one that contains at least three nodes and repeats only the first and last nodes.
- A graph is a **tree** if it is connected and has no simple cycles.



Graphs

- If it has arrows instead of lines, the graph is a **directed graph**.
- The number of arrows pointing from a particular node is the **outdegree** of that node, and the number of arrows pointing to a particular node is the **indegree**.
- The formal description of the graph in Figure 0.16 is $(\{1,2,3,4,5,6\}, \{(1,2),(1,5),(2,1),(2,4),(5,4),(5,6),(6,1),(6,3)\})$.
- A path in which all the arrows point in the same direction as its steps is called a **directed path**.
- A directed graph is **strongly connected** if a directed path connects every two nodes.



Strings and Languages

- Strings of characters are fundamental building blocks in computer science.
- We define an **alphabet** to be any nonempty finite set.
- The members of the alphabet are the **symbols** of the alphabet.
- $\Sigma_1 = \{0,1\}$,
  $\Sigma_2 = \{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z\}$,
  $\Gamma = \{0,1,x,y,z\}$
- A **string over an alphabet** is a finite sequence of symbols from that alphabet.
- If $\Sigma_1 = \{0,1\}$, then 01001 is a string over Σ_1.
- If $\Sigma_2 = \{a,b,c,\ldots,z\}$, then abracadabra is a string over Σ_2.
- If w is a string over Σ, the **length** of w, written $|w|$, is the number of symbols that it contains.
- The string of length zero is called the **empty string** and is written ε.

Strings and Languages

- If w has length n, we can write $w = w_1 w_2 \cdots w_n$ where each $w_i \in \Sigma$.
- The reverse of w, written $w^{\mathcal{R}}$, is the string obtained by writing w in the opposite order (i.e., $w_n w_{n-1} \cdots w_1$).
- String z is a **substring** of w if z appears consecutively within w.
- If we have string x of length m and string y of length n, the **concatenation** of x and y, written xy, is the string obtained by appending y to the end of x.
- The **lexicographic ordering** of strings is the same as the familiar dictionary ordering.
- A **language** is a set of strings.

Boolean Logic

- **Boolean logic** is a mathematical system built around the two values TRUE and FALSE.
- The values TRUE and FALSE are called the **Boolean values** and are often represented by the values 1 and 0.
- **Boolean operations**: **negation**, NOT, $\neg$, **conjunction**, AND, $\wedge$, **disjunction**, OR, $\vee$.
- $P \wedge Q$: P and Q are called the **operands** of the operation.
- **exclusive or**, XOR, $\oplus$, **equality**, $\leftrightarrow$, **implication**, $\rightarrow$.

$0 \oplus 0 = 0$	$0 \leftrightarrow 0 = 1$	$0 \rightarrow 0 = 1$
$0 \oplus 1 = 1$	$0 \leftrightarrow 1 = 0$	$0 \rightarrow 1 = 1$
$1 \oplus 0 = 1$	$1 \leftrightarrow 0 = 0$	$1 \rightarrow 0 = 0$
$1 \oplus 1 = 0$	$1 \leftrightarrow 1 = 1$	$1 \rightarrow 1 = 1$