THOMSON
COURSE TECHNOLOGY

Contains 120-day trial of Maple™ software!

DISCRETE MATHEMATICAL STRUCTURES:
Theory and Applications

D.S. MALIK AND M.K. SEN

Discrete Mathematical Structures: Theory and Applications

D.S. Malik
M.K. Sen

THOMSON
COURSE TECHNOLOGY

Australia • Canada • Mexico • Singapore • Spain • United Kingdom • United States

THOMSON

COURSE TECHNOLOGY™

Discrete Mathematical Structures: Theory and Applications

by D.S. Malik and M.K. Sen

Product Manager:
Alyssa Pratt

Managing Editor:
Jennifer Muroff

Senior Acquisitions Editor:
Amy Yarnevich

Development Editor:
Laurie Brown

Senior Production Editor:
Aimee Poirier

Associate Product Manager:
Mirella Misiaszek

Editorial Assistant:
Amanda Piantedosi

Senior Manufacturing Coordinator:
Trevor Kallop

Cover Designer:
Betsy Young

Compositor:
Techsetters, Inc.

Printed in the United States of America

1 2 3 4 5 6 7 8 9 QWT 08 07 06 05 04

For more information, contact Course Technology, 25 Thomson Place, Boston, Massachusetts, 02210.

Or find us on the World Wide Web at: www.course.com

ISBN 0-619-21285-3 with software
ISBN 0-619-21558-5 without software

Preface

Discrete Mathematical Structures: Theory and Applications is an innovative text that introduces a new way of teaching the Discrete Structures course. A course in discrete structures is an integral part of the computer science curriculum. The class may consist of both math and computer science majors and can be taught either by the mathematics department or by the computer science department. Therefore, it is important that a course in discrete structures present a balance of theoretical concepts as well as their relevant applications.

Approach

The approach that we have taken in this book is a culmination of many years of experience. Our main objective is to make the learning of discrete mathematics easier and fun. Typically, in computer science, a course in discrete mathematics is taken just after programming courses. In many programs, this course becomes a prerequisite of other higher-level courses. In *Discrete Mathematical Structures: Theory and Applications*, we want to give students a solid foundation of theoretical concepts and their applications.

We have been teaching the discrete structures course for a number of years. The textbooks that we have come across tend to be either theory oriented or applications oriented. We do not believe in simply presenting the statement of a theorem and then showing its proof. Showing proof after proof is the surest way to discourage many students. On the other hand, showing application after application without the reinforcement of theoretical results is like following a cook book.

In *Discrete Mathematical Structures: Theory and Applications*, we show why theory is important and how theory connects with applications. Over the years, we have learned that giving an example before and after presenting a theoretical result makes learning easier and effective. Before writing a proof, we usually present examples to show the relevance of the concept. Moreover, we do not just show a proof, we show how the proof is constructed. The same methodology is followed when we present an algorithm. Before and/or after presenting an algorithm, we show how the algorithm works.

This book is written exclusively with students in mind. The language is user-friendly and conducive to learning. Very often we hear statements from students such as "How do I solve problems and write proofs?" To bridge this extremely important gap, we present a set of fully Worked-Out Exercises at the end of each section. These Worked-Out Exercises teach students how to solve problems as well as write proofs—they prepare students to do exercises on their own.

The book contains a rich collection of exercises. Furthermore, at the end of each chapter we include a set of Programming Exercises. Students are encouraged to solve these exercises in the programming language of their choice, such as Maple, C++, or Java.

Although this book is intended for a one-semester course, the book contains more material than could possibly be covered in this time frame. This gives the instructor flexibility in determining topic coverage. The book contains thirteen chapters, and they can be studied out of order depending on individual preference.

Organization and Coverage

Chapter 1 covers the basics of set theory, logic, and algorithms. We present the basic terminology used in set theory and various results used throughout the book. In the logic section, after presenting the basic material, such as statements and rules of inference, we show various proof techniques. Finally, in the algorithm section, we set the syntax used to write algorithms throughout the book.

Chapter 2 is concerned with the properties of integers and principles of induction. Integers are by far the most important source of examples. We cover basic properties of integers and then show how integers are represented in computer memory. Next, we cover the principles of induction in detail, giving various examples and then discussing how induction is used to prove the correctness of programs, especially loops.

In Chapters 3 and 4 we cover relations, posets, and matrices in detail. We show how graphs and matrices are used to represent relations. Moreover, we use matrices to determine the transitive closure of relations on a finite set. Warshall's algorithm is covered in detail to find the transitive closure. We also show how relations are used in the design of relational databases.

Chapter 5 covers functions in detail. Other than covering various types of functions, we show the relationship between functions and strings.

Chapter 6 is concerned with congruences and their various applications. We focus on how congruences are used in the construction of ISBNs, UPCs, credit card numbers, the scheduling of round robin tournaments, hashing, and code words. This chapter can be studied after Chapter 3, and it is not a prerequisite for the remaining chapters in the book.

Chapter 7 focuses on counting techniques. More specifically, we discuss basic counting principles—the addition and the multiplication principle, pigeonhole principles, permutations, combinations, binomial coefficients, and discrete probability. We also give various algorithms to generate permutations, combinations, and binomial coefficients.

Chapter 8 is concerned with advanced counting techniques using recurrence relations. Following this, we focus on solving linear homogenous recurrence relations and certain linear nonhomogenous recurrence relations. We are especially interested in linear nonhomogenous recurrence relations as they frequently appear in the analysis of algorithms that use divide and conquer techniques. We present enough results so that we can analyze the various algorithms given in Chapter 9.

Chapter 9 focuses on the algorithms and their complexity. We start with showing why algorithm analysis is important and then develop theoretical concepts such as Big-O and theta notations. In the second half of this chapter, we present and analyze various searching and sorting algorithms, as well as discuss algorithms to multiply matrices and an effective way to determine the order in which a sequence of matrices can be multiplied.

Chapter 10 covers graphs in detail. Starting with basic graph theory definitions and terminology, we discuss topics such as subgraphs, walks, paths, circuits, isomorphism of graphs, planer graphs, and graph coloring. Also covered is a way to represent graphs in computer memory as well as various graph algorithms.

Chapter 11 focuses on trees, special types of trees, and determining spanning and minimal spanning trees. We close this chapter with a discussion of the transport network and present an algorithm to determine a maximal flow in a network.

Chapter 12 is concerned with Boolean algebra and its applications in the design of electric circuits.

Chapter 13 presents an introduction to automata theory and languages.

The Web site accompanying this book contains the following additional material: applications of Boolean algebra in the design of switching circuits, characterization of regular languages by right congruences, nondeterministic finite automata with lambda transitions, and generating functions.

The chapter dependency diagram in Figure 1 shows the dependency of chapters. A dotted line means that that the chapter is not necessarily a prerequisite for the subsequent chapter.

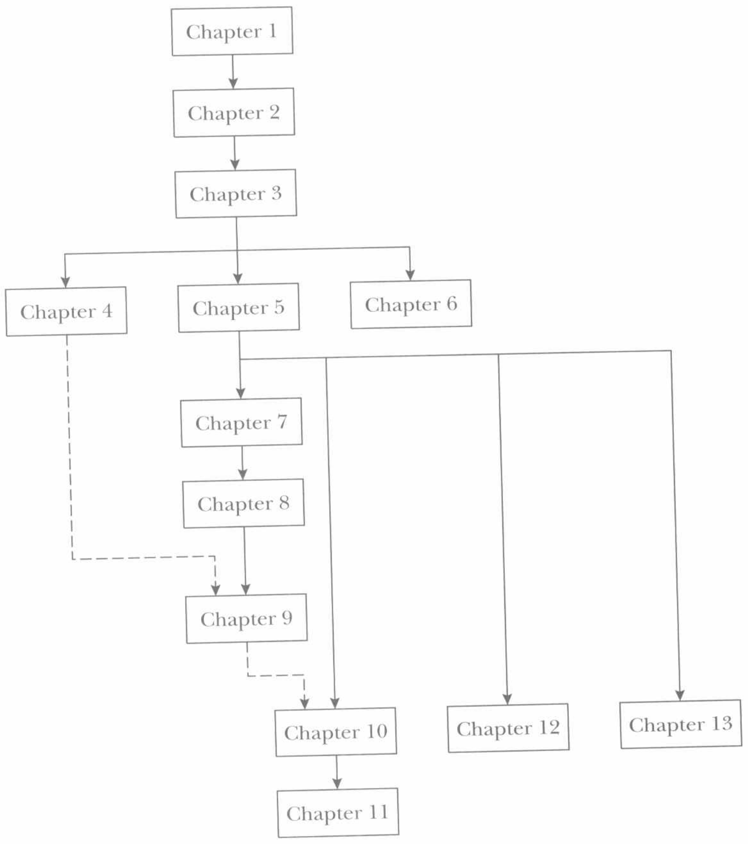


FIGURE 1

As shown in Figure 1, Chapters 1, 2, and 3 should be studied in sequence. After studying these chapters there are various choices. In Chapter 9, we describe certain algorithms related to matrices. The basic concept and basic operations of matrices, such as addition and multiplication, are needed to understand these algorithms. Therefore, only these parts from Chapter 4 are need for Chapter 9.

In Chapter 9, we present various notations used in algorithm analysis, such as Big-O and theta. In Chapter 10, other than discussing theoretical concepts related to graphs, we also discuss the matrix representation of graphs and applications of graphs in computer science such as shortest path algorithm and topological sort. Moreover, these algorithms are described in detail. Only the concept of theta notation from Chapter 9 is needed for the analysis of the shortest path algorithm. Other than this, Chapter 9 is not a prerequisite for Chapter 10. Similarly, only the basic properties of matrices are needed to study Chapter 10.

Syllabus Planning

Some of the ways the chapters can be studied are (Chapter 6 can be studied any time after Chapter 3. Therefore, we do not list Chapter 6 in the following sequences.):

1. Study all the chapters in sequence.
2. Study the chapters in the sequence: 1, 2, 3, 4, 5, 10, 11, 7, 8, 9, 12, 13.
3. Study the chapters in the sequence: 1, 2, 3, 5, 4, 10, 11, 7, 8, 9, 12, 13.
4. Study the chapters in the sequence: 1, 2, 3, 5, 4,10, 11, 12, 13, 6, 7, 8, 9.

Features

Every chapter in this book includes the following features. These features are both conducive to learning and allow students to learn the material at their own pace.

- *Learning Objectives* offer an outline of the concepts discussed in detail in the chapter.
- *Remarks* highlight important facts about the concepts introduced in the chapter.
- More than 450 visual diagrams, both extensive and exhaustive, illustrate difficult concepts.
- *Numbered Examples* illustrate the key concepts.
- *Worked-Out Exercises* is a set of more than 325 *fully* Worked-Out Exercises at the end of each chapter. These exercises teach how to solve problems and write proofs. We strongly recommend that students study these Worked-Out Exercises very carefully in order to learn problem-solving techniques.
- *Section Review* offers a summary of the concepts covered in the chapter.
- *Exercises* further reinforce learning and ensure that students have, in fact, learned the material.
- *Programming Exercises* challenge students to write programs with a specified outcome.

Student Resources

Maple Software. We are pleased to offer a 120-day trial version of Maple software with every saleable copy of this text. The CD in the back of the book provides access to a fully functional version of the latest release of Maple.

Student Online Companion Web Site. In the front of this text, you will find a scratch-off card with a key code that provides full access to a robust Web site, located at `www.course.com/malikdiscrete`. This site includes the following features:

- **Student Solutions Manual** that supplies answers and detailed explanations to the odd-numbered problems in the text.
- **Student Guide to Maple Software** that contains chapter-by-chapter suggestions for using Maple to illustrate concepts in the text.
- **Review Questions** that allow the opportunity to think critically about the material in each chapter.

- **Practice Tests** for each chapter that provide extra practice in an interactive format.
- **Useful Web Links** that offer additional information about the ideas discussed in each chapter.

Teaching Tools

Discrete Mathematical Structures: Theory and Applications includes teaching tools to support instructors in the classroom. The ancillaries that accompany the textbook include an Instructor's Manual, Solutions, Test Banks, and Test Engine, PowerPoint presentations, and Figure Files. All teaching tools available with this book are provided to the instructor on a single CD-ROM and are also available on the Web at `www.course.com`.

Electronic Instructor's Manual. The Instructor's Manual that accompanies this textbook includes:

- Additional instructional material to assist in class preparation, including suggestions for lecture topics
- Solutions to all the exercises, including the Programming Exercises

ExamView® This objective-based test generator lets the instructor create paper, LAN, or Web-based tests from testbanks specifically designed for this Course Technology text. Instructors can use the QuickTest Wizard to create tests in fewer than five minutes by taking advantage of Course Technology's question banks—or create customized exams.

Solutions. The solution files for all programming exercises in C++ are available at `www.course.com`, and are also available on the Teaching Tools CD-ROM.

PowerPoint Presentations. Microsoft PowerPoint slides are included for each chapter. Instructors might use the slides in a variety of ways, including as teaching aids during classroom presentations or as printed handouts for classroom distribution. Instructors can add their own slides for additional topics introduced to the class.

Figure Files. Figure files allow instructors to create their own presentations using figures taken directly from the text.

Acknowledgements

There are many people that we must thank who, in one way or another, contributed to the success of this book. First, we would like to thank Dr. S.C. Cheng for his support and making suggestions to improve the text. We must also thank students who, during the preparation, were spontaneous in telling us if certain portions needed to be reworded for better understanding and clearer reading. We must thank Lee I. Fenicle, Director, Office of Technology Transfer, Creighton University, Dr. Randall L. Crist, and Dr. Ratish Basu Roy for their involvement, support, and for providing encouraging words when we needed them. We would like to acknowledge the feedback provided by Sunil Kumar Maity and Madhumita Mukherjee.

We owe a great deal to the following reviewers who patiently read each page of every chapter of the current version and made critical comments to improve on the book: Jim Ball, Indiana State University; Jose Cordova, University of Louisiana at Monroe; Joseph Klerlein, Western Carolina University; and Catherine Yan, Texas

A&M University. The reviewers will recognize that their criticisms have not been overlooked and, in fact, made this a better book. Thanks to Development Editor Laurie Brown for carefully editing and promptly returning each chapter. All this would not have been possible without the planning of Managing Editor Jennifer Muroff and Product Manager Alyssa Pratt. Our sincere thanks to Jennifer Muroff and Alyssa Pratt, as well as to Aimee Poirier, Senior Production Editor, and also to the QA department of Course Technology for carefully testing the code. We would especially like to thank Burt LaFountain, QA Tester, for carefully reading the manuscript, solutions to all the exercises, and programming exercises. We would also like to thank Kate Deibel, University of Washington, for providing answers to the programming exercises in Chapters 1–8 and 11, and Jim Bishop, Bryant College, for his help with the biographies.

We are thankful to our parents for their blessings.

Finally, we are thankful for the support of our wives Sadhana and Monisha and our children Shelly, Nilanjan, Debanjan, and Shubhashree. They cheered us whenever we were overwhelmed during the writing of this book. We welcome any comments concerning the text. In spite of our diligent efforts there may still be room for improvement. Comments may be forwarded to the following e-mail address: `malik@creighton.edu` or `senmk@cal3.vsnl.net.in`.

D.S. Malik
M.K. Sen

Contents

chapter 1

Foundations: Sets, Logic, and Algorithms

The objectives of this chapter are to:

- Learn about sets
- Explore various operations on sets
- Become familiar with Venn diagrams
- Learn how to represent sets in computer memory
- Learn about statements (propositions)
- Learn how to use logical connectives to combine statements
- Explore how to draw conclusion using various argument forms
- Become familiar with quantifiers and predicates
- Learn various proof techniques
- Explore what an algorithm is

This chapter sets the stage for all that follows and also serves as an appropriate place for codifying certain technical terminologies used throughout the text. In the first section, we discuss sets and their basic properties. We then study mathematical logic in some detail. In this book, the focus is not just on theory but also on applications. When theoretical concepts are presented, we give various examples to clarify the concepts as well as to prove theoretical results, wherever appropriate. Therefore, after discussing sets, we study mathematical logic and describe various proof techniques.

Over the years a revolution in computer technology has changed the ways in which we live and communicate. Computer programs have made tedious computations easy to handle and have enabled us to achieve results quickly and to a great degree of precision. Therefore, throughout the book we discuss various algorithms that can be implemented in a variety of programming languages, such as C++ and Java. In the last section of this chapter, we introduce algorithms and describe the syntax of the pseudocode used to describe algorithms in this book.

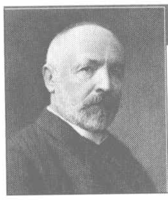
Natural numbers, integers, rational numbers, and real numbers are a great source of examples. We assume that the reader is familiar with these number systems.

1.1 SETS

The mathematical theory of sets grew out of the German mathematician Georg Cantor's study of trigonometric series and series of real numbers. The language of sets has since become an important tool for all branches of mathematics, serving as a basis for the precise description of higher concepts and for mathematical reasoning.

Let us begin with the question, what is a *set*? It is fascinating that the answer to this very basic and apparently simple question once jeopardized the very foundation of set theory. In this text, however, we adopt a naive and intuitive point of view and introduce the definition of a **set** according to his definition. According to his definition, *a set is a well-defined collection of distinct objects* of our perception or of our thoughts, to be conceived as a whole.

Historical Notes



Georg Cantor (1845–1918) Although considered one of the great German mathematicians, Cantor was born in St. Petersburg, Russia, in the winter of 1845 to a wealthy Danish merchant. At the age of 11, he moved with his family to Germany where he continued his education, earning a doctorate degree from the University of Berlin in 1867. In 1869 Cantor accepted a post at the University of Halle, an undistinguished school for women. His provocative ideas regarding concepts of infinity had put him in bad standing with his contemporaries, and many of them opposed his appointment to the prestigious University of Berlin. Cantor suffered fits of depression due in large part to stress related to his work. He spent the better part of his later years in and out of mental hospitals and ultimately died in a sanatorium.

Cantor is considered to be the founder of set theory, and he established its relation to transfinite numbers. He explored paradoxes that had existed in mathematics for centuries and even stumbled upon one of his own, now known as Cantor's paradox. Although his theories were vehemently disputed by his peers, including Leopold Kronecker, his mentor at the University of Berlin, modern mathematicians completely accept Cantor's work.

To develop a perfectly balanced working idea of sets, it is sufficient for a beginner to concentrate on the first part (the italicized part) of this definition. Note that here *well-defined* is an adjective to the noun *collection* and not to the distinct objects that are to be collected to form a set. What this means is that there should be no ambiguity whatsoever regarding the membership of such a collection; *well-defined* means that we can tell for certain whether an object is a member of the collection or not. These objects are called *members* or *elements* of the set.

For example, we can talk about the set of all positive integers, even though no one really knows all of them. But a collection of *some* positive integers is *not* a set because it is not clear whether a particular positive integer, say 5, is a member of this collection or not. For another example, the collection of students taking the discrete mathematics course in your school is a set. On the other hand, the collection of best cars in a city cannot be a set because there is no well-defined notion of best.

We use italic uppercase letters, $A, B, C, \ldots, X, Y, Z$, to denote sets. A set can be described in various ways, but the main point of any description is to specify the elements of the set in some unambiguous way. One common way, called the **roster method**, to describe a set is to list the elements of the set and enclose them within curly braces. For example, if A is a set of vowels, then we write

$$A = \{a, e, i, o, u\}.$$

For another example, we can describe the set B of all positive integers less than 11 as

$$B = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}.$$

Let X be a set. If x is an element of X, then we write $x \in X$ and say that x belongs to X. The symbol $\in$ stands for *belongs to*, which, like many other notations, was introduced in 1889 by the Italian mathematician Giuseppe Peano (1858–1932) and is believed to be a stylized form of the Greek epsilon. If x is not an element of X, then we write $x \notin X$ and say that x is not an element of X. The symbol $\notin$ stands for *does not belong to*.

EXAMPLE 1.1.1 Let A be the set

$$A = \{1, 2, 3, 4, 5\}.$$

Then $2 \in A$ and $5 \in A$. Also, $6 \notin A$.

EXAMPLE 1.1.2 Let B be the set of first 10 positive odd integers. Then

$$B = \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19\}.$$

It follows that $9 \in B$ and $2 \notin B$.

We also describe sets in the following manner. Let S be a set. The notation

$$A = \{x \mid x \in S, P(x)\}$$

or

$$A = \{x \in S \mid P(x)\}$$

means that A is the set of all elements x of S such that x satisfies the property P. This way of describing a set is called the **set-builder method**.

For example, if $\mathbb{Z}$ denotes the set of integers, then

$$\mathbb{N} = \{x \mid x \in \mathbb{Z},\ x > 0\}$$

or

$$\mathbb{N} = \{x \in \mathbb{Z} \mid x > 0\}.$$

Here the property $P(x)$ is

$$P(x) : x > 0.$$

EXAMPLE 1.1.3 In set-builder notation, the set B of Example 1.1.2 can be described as

$$B = \{x \in \mathbb{Z} \mid x \text{ is odd and } 1 \leq x \leq 19\}.$$

EXAMPLE 1.1.4 Let $A = \{2, -2\}$. Because 2 and -2 are the only integers that satisfy the equation $x^2 - 4 = 0$, we can also write A as

$$A = \{x \mid x \in \mathbb{Z},\ x^2 - 4 = 0\}$$

or

$$A = \{x \in \mathbb{Z} \mid x^2 - 4 = 0\}.$$

Here the property $P(x)$ is

$$P(x) : x^2 - 4 = 0.$$

EXAMPLE 1.1.5 Let A be the set described in set-builder form as:

$$A = \{x \mid x \text{ is a complex number and } x^4 = 1\}.$$

Now the equation $x^4 = 1$, i.e., $x^4 - 1 = 0$, can be factored as

$$(x + 1)(x - 1)(x - i)(x + i) = 0,$$

where $i^2 = -1$ or $i = \sqrt{-1}$. This implies that the solutions of the equation $x^4 = 1$, where x is a complex number, are $x = 1, -1, i, -i$. Therefore, using the roster form, the set A can be written as

$$A = \{1, -1, i, -i\}.$$

Throughout the book, we will use numbers to provide examples. Therefore, we would like to standardize the symbols to denote various sets of numbers as follows.

$\mathbb{N}$: The set of all natural numbers (i.e., all positive integers)

$\mathbb{Z}$: The set of all integers

$\mathbb{Z}^*$: The set of all nonzero integers

$\mathbb{E}$: The set of all even integers

$\mathbb{Q}$: The set of all rational numbers

$\mathbb{Q}^*$: The set of all nonzero rational numbers

$\mathbb{Q}^+$: The set of all positive rational numbers

$\mathbb{R}$: The set of all real numbers

$\mathbb{R}^*$: The set of all nonzero real numbers

$\mathbb{R}^+$: The set of all positive real numbers

$\mathbb{C}$: The set of all complex numbers

$\mathbb{C}^*$: The set of all nonzero complex numbers

We know that every integer is a real number; that is, every element of $\mathbb{Z}$ is an element of $\mathbb{R}$. Similarly, every vowel is a letter in the set of English letters. In other words, if $A = \{a, e, i, o, u\}$ and B is the set of all English letters, then every element of A is an element of B. When every element of a set, say A, is also an element of a set, say B, we say that A is a subset of B. More formally, we have the following definition.

DEFINITION 1.1.6 ▶ Let X and Y be sets. Then X is said to be a **subset** of Y, written $X \subseteq Y$, if every element of X is an element of Y. If X is not a subset of Y, then we write $X \nsubseteq Y$.

EXAMPLE 1.1.7

(i) Let $X = \{0, 2, 4, 6, 8\}$, $Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, and $Z = \{1, 2, 3, 4, 5\}$. Then $X \subseteq Y$ because every element of X is an element of Y. However, because $0 \in X$ and $0 \notin Z$, we have $X \nsubseteq Z$.

Notice that we used the fact that $0 \in X$ and $0 \notin Z$ to conclude that $X \nsubseteq Z$. We could have also used the fact that $6 \in X$ and $6 \notin Z$ or $8 \in X$ and $8 \notin Z$ to conclude that $X \nsubseteq Z$. In other words, the elements 6 and 8 also prevent X from being a subset of Z.

(ii) Let $A = \{a, b, c\}$ and $B = \{a, c, b\}$. Now every element of A is also an element of B and so $A \subseteq B$. Also notice that $B \subseteq A$.

(iii) Let $A = \{\text{Basic, Fortran, C++}\}$ and $B = \{\text{Basic, Fortran, Pascal, C++, Java}\}$. Then $A \subseteq B$.

Note: For every set X, we have $X \subseteq X$.

Let X and Y be sets. If $X \subseteq Y$, we also say that X is contained in Y, or Y contains X, or Y is a **superset** of X (written $Y \supseteq X$).

Notice that in Example 1.1.7(i), every element of X is an element of Y. However, there are some elements in Y that are not in X. Such a set X is called a proper subset of Y.

DEFINITION 1.1.8 ▶ Let X and Y be sets. Then X is a **proper subset** of Y, written $X \subset Y$, if X is a subset of Y and there exists at least one element in Y that is not in X.

EXAMPLE 1.1.9

Let $A = \{a, b\}$ and $B = \{a, b, c\}$. Because every element of A is an element of B, we have $A \subseteq B$. Now $c \in B$ and $c \notin A$. Therefore, there exists an element in B that is not in A. It now follows that A is a proper subset of B, i.e., $A \subset B$.

EXAMPLE 1.1.10

The set of all even integers is a proper subset of the set of all integers. In set notation, $\{2n \mid n \in \mathbb{Z}\} \subset \mathbb{Z}$.

Note that $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$.