DISCRETE MATHEMATICS

A.JEYANTHI

DISCRETE MATHEMATICS

A.JEYANTHI

NAIRJC INTERNATIONAL BOOK PUBLISHERS

PULWAMA, JAMMU AND KASHMIR, INDIA

DISCRETE MATHEMATICS

First Edition

Author

A.JEYANTHI,

Visiting Faculty in the Department of Mathematics, Anna University, Regional Campus, Madurai, Tamil Nadu, India.

Publishers:

Nairjc Publications

221, Pulwama, Jammu & Kashmir, India.

Ph.: 01933-212815, 7298754556

E-mail: *info@nairjc.com*

Website: *www.nairjc.com*

ISBN: 978-81-932604-1-8

First Edition: 2016

Price: 270

Printers:

Nairjc Offset

Address: Pulwama, Jammu & Kashmir, India -192301

ABOUT THE AUTHOR

A.Jeyanthi: She is working as a Visiting Faculty in the Department of Mathematics, Anna University, Regional Campus, Madurai. She has Eleven years of experience of teaching in Engineering Mathematics at Anna University. She has finished her Ph.D. in October, 24-2014 of Automata Theory in the area of Theoretical computer science. She has published thirty papers in international journals. Her teaching motivates the young students in engineering Mathematics. Her guidance induces an interest to the student's project of research area of Automata Theory.

ACKNOWLEDGEMENT

Foremost, I want to offer this endeavour to our Almighty God for the wisdom he bestowed upon us, the strength, peace of mind and good health in order to complete this book successfully.

I would like to express my gratitude towards my family for the encouragement which helped me in the completion of this book. My beloved parents, my husband who supported and encouraged me through everything and my lovable children who served as an inspiration to pursue this undertaking.

This book has become a reality with the kind support and help of many individuals. I would like to extend my sincere thanks to all of them.

To all relatives, friends and others who in one way or another share their support either morally financially.

I would also like to express my sincere gratitude to my publisher NAIRJC, International book publishers.

The completion of this book gives us much Pleasure with Satisfaction.

I am very thankful to Mrs.V.Kalpana Visiting Faculty in Master of computer Applications, Anna University, Regional Campus, Madurai, for her aspiring guidance, invaluably constructive criticism and friendly advice during written my book Discrete Mathematics.

I am sincerely grateful to every one of them who are involved and sharing their truthful and illuminating views on a number of issues related to this book. Thank you.

CONTENTS

CHAPTER 1

FUNCTIONS

1.1 INTRODUCTION

A relation is any association or link between elements of one set, called the domain or (less formally) the set of inputs, and another set, called the range or set of outputs. Some people mistakenly refer to the range as the co-domain (range), but as we will see, that really means the set of all possible outputs—even values that the relation does not actually use. (Beware: some authors do not use the term co-domain (range), and use the term range instead for this purpose. Those authors use the term image for what we are calling range. So while it is a mistake to refer to the range or image as the co-domain (range), it is not necessarily a mistake to refer to co-domain as range.)

For example, if the domain is a set Fruits = {apples, oranges, bananas} and the co-domain (range) is a set Flavors = {sweetness, tartness, bitterness}, the flavors of these fruits form a relation: we might say that apples are related to (or associated with) both sweetness and tartness, while oranges are related to tartness only and bananas to sweetness only. (We might disagree somewhat, but that is irrelevant to the topic of this book.) Notice that "bitterness", although it is one of the possible Flavors (co-domain) (range), is not really used for any of these relationships; so it is not part of the range (or image) {sweetness, tartness}.

Another way of looking at this is to say that a relation is a subset of ordered pairs drawn from the set of all possible ordered pairs (of elements of two other sets, which we normally refer to as the Cartesian product of those sets). Formally, R is a relation if $R \subseteq X \times Y = \{(x, y) \mid x \in X, y \in Y\}$ for the domain X and co-domain (range) Y. The inverse relation of R, which is written as R^{-1}, is what we get when we interchange the X and Y values: $R^{-1} = \{(y, x) \mid (x, y) \in R\}$

Using the example above, we can write the relation in set notation: {(apples, sweetness), (apples, tartness), (oranges, tartness), (bananas, sweetness)}. The inverse relation, which we could describe as "fruits of a given flavor", is

{(sweetness, apples), (sweetness, bananas), (tartness, apples), (tartness, oranges)}. (Here, as elsewhere, the order of elements in a set has no significance.)

One important kind of relation is the function. A function is a relation that has exactly one output for every possible input in the domain. (The domain does not necessarily have to include all possible objects of a given type. In fact, we sometimes intentionally use a restricted domain in order to satisfy some desirable property.) The relations discussed above (flavors of fruits and fruits of a given flavor) are **not** functions: the first has two possible outputs for the input "apples" (sweetness and tartness); and the second has two outputs for both "sweetness" (apples and bananas) and "tartness" (apples and oranges).

The main reason for not allowing multiple outputs with the same input is that it lets us apply the same function to different forms of the same thing without changing their equivalence. That is, if f is a function with a (or b) in its domain, then a = b implies that f(a) = f(b). For example, z - 3 = 5 implies that z = 8 because f(x) = x + 3 is a function defined for all numbers x.

One of the most important concepts in mathematics is that of a function. The terms “map,” “mapping,” “transformation,” and many others mean the same thing; the choice of which word to use in a given situation is usually determined by tradition and the mathematical background of the person using the term. Related to the notion of a function is that of an algorithm. The notation for presenting an algorithm and a discussion of its complexity is also covered in this chapter.

1.2 FUNCTIONS

A function is a relationship between two sets of numbers. We may think of this as a mapping; a function maps a number in one set to a number in another set. Notice that a function maps values to one and only one value. Two values in one set could map to one value, but one value must never map to two values: that would be a relation, not a function. Suppose that to each element of a set A we assign a unique element of a set B; the collection of such assignments is called a function from A into B. The set A is called the domain of the function, and the set B is called the target set or co-domain.

Functions are ordinarily denoted by symbols. For example, let f denote a function from A into B. Then we

Write f: A→B, which is read: "f is a function from A into B," or "f takes (or maps) A into B." If a ∈ A, then f (a) (read: "f of a") denotes the unique element of B which f assigns to a; it is called the image of a under f, or the value of f at a. The set of all image values is called the range or image of f. The image of f: A→B is denoted by Ran (f), Im(f) or f (A). Frequently, a function can be expressed by means of a mathematical formula. For example, consider the function which sends each real number into its square. We may describe this function by writing $f(x) = x^2$ or $x \mapsto x^2$ or $y = x^2$

In the first notation, x is called a variable and the letter f denotes the function. In the second notation, the barred arrow ↦ is read "goes into." In the last notation, x is called the independent variable and y is called the dependent variable since the value of y will depend on the value of x.

Remark: Whenever a function is given by a formula in terms of a variable x, we assume, unless it is otherwise stated, that the domain of the function is R (or the largest subset of R for which the formula has meaning) and the co-domain is

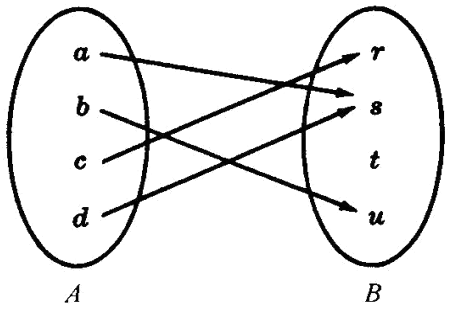


Fig. 1-1

EXAMPLE 1.1

(a) Consider the function $f(x) = x^3$, i.e., f assigns to each real number its cube. Then the image of 2 is 8, and so we may write f (2) = 8.

(b) Figure 1-1 defines a function f from A = {a, b, c, d} into B = {r, s, t, u} in the

obvious way. Here

$$f(a) = s, \quad f(b) = u, \quad f(c) = r, \quad f(d) = s$$

The image of f is the set of image values, {r, s, u}. Note that t does not belong to the image of f because t is not the image of any element under f.

(c) Let A be any set. The function from A into A which assigns to each element in A the element itself is called the identity function on A and it is usually denoted by 1_A, or simply 1. In other words, for every $a \in A$,

$$1_A(a) = a.$$

(d) Suppose S is a subset of A, that is, suppose $S \subseteq A$. The inclusion map or embedding of S into A, denoted by $i: S \hookrightarrow A$ is the function such that, for every $x \in S$,

$$i(x) = x$$

The restriction of any function $f : A \to B$, denoted by $f|_S$ is the function from S into B such that, for any $x \in S$,

$$f|_S(x) = f(x)$$

Functions as Relations

There is another point of view from which functions may be considered. First of all, every function $f : A \to B$ gives rise to a relation from A to B called the graph of f and defined by

$$\text{Graph of } f = \{(a, b) \mid a \in A, b = f(a)\}$$

Two functions $f : A \to B$ and $g : A \to B$ are defined to be equal, written f = g, if f (a) = g(a) for every $a \in A$; that is, if they have the same graph. Accordingly, we do not distinguish between a function and its graph. Now, such a graph relation has the property that each a in A belongs to a unique ordered pair (a, b) in the relation. On the other hand, any relation f from A to B that has this property gives rise to a function $f : A \to B$, where f (a) = b for each (a, b) in f.

Consequently, one may equivalently define a function as follows:

DEFINITION1.1: A function f : A → B is a relation from A to B (i.e., a subset of A × B) such that each a ∈ A belongs to a unique ordered pair (a, b) in f.

Although we do not distinguish between a function and its graph, we will still use the terminology "graph of f " when referring to f as a set of ordered pairs. Moreover, since the graph of f is a relation, we can draw its picture as was done for relations in general, and this pictorial representation is itself sometimes called the graph of f. Also, the defining condition of a function, that each a ∈ A belongs to a unique pair (a, b) in f, is equivalent to the geometrical condition of each vertical line intersecting the graph in exactly one point.

EXAMPLE 1.2

(a) Let f : A → B be the function defined in Example 1.1 (b). Then the graph of f is as follows:

$$\{(a, s), (b, u), (c, r), (d, s)\}$$

(b) Consider the following three relations on the set A = {1, 2, 3}:

$$f = \{(1, 3), (2, 3), (3, 1)\}, \quad g = \{(1, 2), (3, 1)\}, \quad h = \{(1, 3), (2, 1), (1, 2), (3, 1)\}$$

f is a function from A into A since each member of A appears as the first coordinate in exactly one ordered pair in f; here f (1) = 3, f (2) = 3, and f (3) = 1. g is not a function from A into A since 2 ∈ A is not the first coordinate of any pair in g and so g does not assign any image to 2. Also *h* is not a function from *A* into *A* since 1 ∈ A appears as the first coordinate of two distinct ordered pairs in h, (1, 3) and (1, 2). If h is to be a function it cannot assign both 3 and 2 to the element 1 ∈ A.

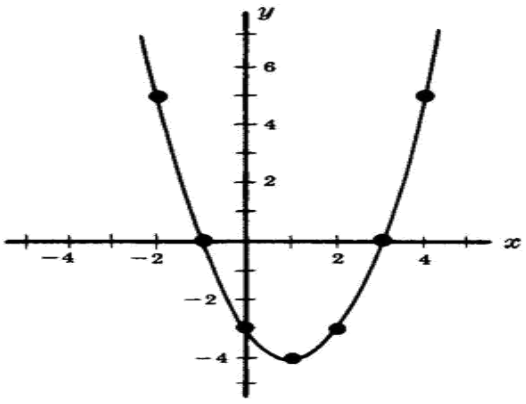
(c) By a real polynomial function, we mean a function f :

$$\mathbf{R} \to \mathbf{R} \text{ of the form } f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

Where the a_i are real numbers. Since **R** is an infinite set, it would be

impossible to plot each point of the graph. However, the graph of such a function can be approximated by first plotting some of its points and then drawing a smooth curve through these points. The points are usually obtained from a table where various values are assigned to x and the corresponding values of f (x) are computed. Figure 1-2 illustrates this technique using the function $f(x) = x^2 - 2x - 3$.

x	$f(x)$
−2	5
−1	0
0	−3
1	−4
2	−3
3	0
4	5



Graph of $f(x) = x^2 - 2x - 3$

Fig. 1-2

Composition Function

Consider functions f : A → B and g: B → C; that is, where the co-domain of f is the domain of g. Then we may define a new function from A to C, called the composition of f and g and written g◦f , as follows:

$$(g \circ f)(a) \equiv g(f(a))$$

That is, we find the image of a under f and then find the image of f (a) under g. This definition is not really new. If we view f and g as relations, then this function is the same as the composition of f and g as relations except that here we use the functional notation g◦f for the composition of f and g instead of the notation f ◦g which was used for relations.

Consider any function f : A → B. Then

$$f \circ 1_A = f \quad \text{and} \quad 1_B \circ f = f$$

Where 1_A and 1_B are the identity functions on A and B, respectively.

1.3 TYPES OF FUNCTIONS

A function f : A → B is said to be one-to-one (written 1-1) if different elements in the domain A have distinct images. Another way of saying the same thing is that f is one-to-one if f (a) = f (a) implies a = a .

A function f : A → B is said to be an onto function if each element of B is the image of some element of A. In other words, f : A → B is onto if the image of f is the entire co-domain, i.e., if f (A) = B. In such a case we say that f is a function from A onto B or that f maps A onto B.

A function f : A → B is invertible if its inverse relation f^{-1} is a function from B to A. In general, the inverse relation f^{-1} may not be a function. The following theorem gives simple criteria which tells us when it is.

THEOREM 1.1: A function f : A → B is invertible if and only if f is both one-to-one and onto.

If f : A → B is one-to-one and onto, then f is called a one-to-one correspondence between A and B. This terminology comes from the fact that each element of A will then correspond to a unique element of B and vice versa.

Some texts use the terms injective for a one-to-one function, surjective for an onto function, and bijective for a one-to-one correspondence.

EXAMPLE 1.3 Consider the functions f_1: A → B, f_2: B → C, f_3: C → D and f_4: D → E defined by the diagram of Fig. 1-3. Now f_1 is one-to-one since no element of B is the image of more than one element of A. Similarly, f_2 is one-to-one. However, neither f_3 nor f_4 is one-to-one since $f_3(r) = f_3(u)$ and $f_4(v) = f_4(w)$

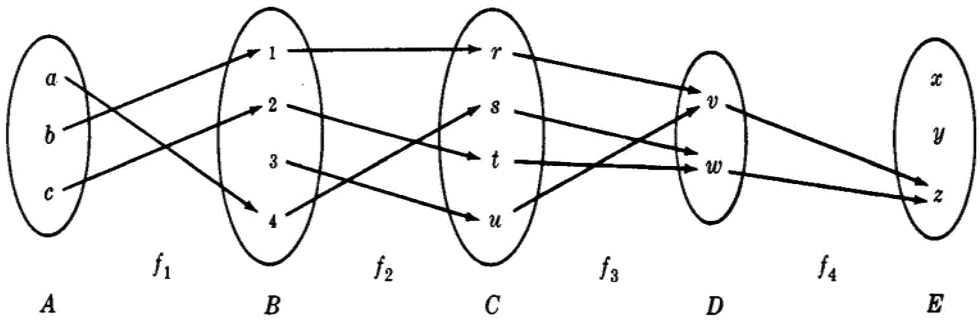


Fig. 1.3

As far as being onto is concerned, f_2 and f_3 are both onto functions since every element of C is the image under f_2 of some element of B and every element of D is the image under f_3 of some element of C, $f_2(B) = C$ and $f_3(C) = D$. On the other hand, f_1 is not onto since $3 \in B$ is not the image under f_4 of any element of A. and f_4 is not onto since $x \in E$ is not the image under f_4 of any element of D.

Thus f_1 is one-to-one but not onto, f_3 is onto but not one-to-one and f_4 is neither one-to-one nor onto. However, f_2 is both one-to-one and onto, i.e., is a one-to-one correspondence between A and B. Hence f_2 is invertible and f_2^{-1} is a function from C to B.

Geometrical Characterization of One-to-One and Onto Functions

Consider now functions of the form $f : \mathbf{R} \to \mathbf{R}$. Since the graphs of such functions may be plot-ted in the Cartesian plane $\mathbf{R}^2$ and since functions may be identified with their graphs, we might wonder, whether the concepts of being one-to-one and onto have some geometrical meaning. The answer is yes. Specifically:

(1) $f : \mathbf{R} \to \mathbf{R}$ is one-to-one if each horizontal line intersects the graph of f in at most one point.

(2) $f : \mathbf{R} \to \mathbf{R}$ is an onto function if each horizontal line intersects the graph of f at one or more points.

Accordingly, if f is both one-to-one and onto, i.e. invertible, then each horizontal line will intersect the graph of f at exactly one point.

EXAMPLE 1.4: Consider the following four functions from **R** into **R**:

$$f_1(x) = x^2, \quad f_2(x) = 2^x, \quad f_3(x) = x^3 - 2x^2 - 5x + 6, \quad f_4(x) = x^3$$

The graphs of these functions appear in Fig. 1-4. Observe that there are horizontal lines which intersect the graph of f_1 twice and there are horizontal lines which do not intersect the graph of f_1 at all; hence f_1 is neither one-to-one nor onto. Similarly, f_2 is one-to-one but not onto, f_3 is onto but not one-to-one and f_4 is both one-to-one and onto. The inverse of f_4 is the cube root function. i.e.,$f_4^{-1}(x) = \sqrt[3]{x}$

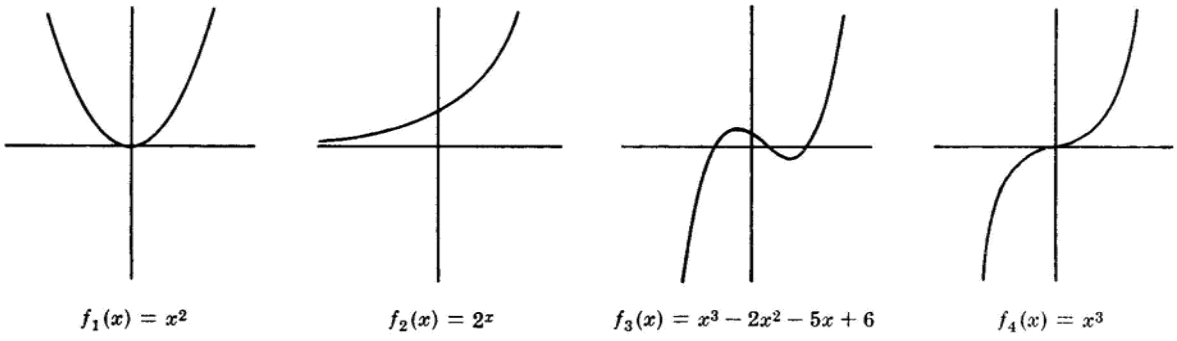


Fig. 1-4

1.4 Permutations

An invertible (bijective) function σ: X → X is called a permutation on X. The composition and inverses of permutations on X and the identity function on X are also permutations on X.

Suppose X = {1, 2, . . . , n}. Then a permutation σ on X is frequently denoted by

$$\sigma = \begin{pmatrix} 1 & 2 & 3..... & n \\ j_1 & j_2 & j_3..... & j_n \end{pmatrix}$$

where $j_1 = \sigma(i)$. The set of all such permutations is denoted by S_n, and there are n!

$= n(n-1) \cdots 3 \cdot 2 \cdot 1$ of them. For example, $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 6 & 2 & 5 & 1 & 3 \end{pmatrix}$ and

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 4 & 3 & 1 & 1 & 3 \end{pmatrix}$$

are permutations in S_6, and there are 6! = 720 of them. Sometimes, we only write the second line of the permutation, that is, we denote the above permutations by writing σ = 462513 and τ = 643125.

1.5 MATHEMATICAL FUNCTIONS, EXPONENTIAL AND LOGARITHMIC FUNCTIONS

This section presents various mathematical functions which appear often in the analysis of algorithms, and in computer science in general, together with their notation. We also discuss the exponential and logarithmic functions, and their relationship.

Floor and Ceiling Functions

Let x be any real number. Then x lies between two integers called the floor and the ceiling of x. Specifically,

[x], called the floor of x, denotes the greatest integer that does not exceed x.

[x], called the ceiling of x, denotes the least integer that is not less than x.

If x is itself an integer, then [x] = [x] ; otherwise [x] + 1 = [x] . For example,

$$[3.14] = 3, \quad [\sqrt{5}\,] = 2, \quad [-8.5] = -9, \quad [7] = 7 \quad [-4] = -4,$$

$$[3.14] = 4, \quad [\sqrt{5}\,] = 3, \quad [8.5] = -8, \quad [7] = 7, \quad [-4] = -4$$

Integer and Absolute Value Functions

Let x be any real number. The integer value of x, written INT(x), converts x into an integer by deleting (truncating) the fractional part of the number. Thus

$$\text{INT}(3.14) = 3, \text{INT}(5) = 2, \text{INT}(-8.5) = -8, \text{INT}(7) = 7$$

Observe that INT(x) = [x], or INT(x) = [x], according to whether x is positive or negative.

The absolute value of the real number x, written ABS(x) or $|x|$, is defined as the greater of x or − x. Hence ABS(0) = 0, and, for $x \neq 0$, ABS(x) = x or ABS(x) = − x, depending on whether x is positive or negative. Thus

$$|-15| = 15, \quad |7| = 7, \quad |-3.33| = 3.33, \quad |4.44| = 4.44, \quad |-0.075| = 0.075$$

We note that $|x| = |-x|$ and, for $x \neq 0$, $|x|$ is positive.

Remainder Function and Modular Arithmetic

Let k be any integer and let M be a positive integer. Then

$$k \pmod M$$

(read: k modulo M) will denote the integer remainder when k is divided by M. More exactly, k (mod M) is the unique integer r such that

$$k = Mq + r \quad \text{where} \quad 0 \leq r < M$$

When k is positive, simply divide k by M to obtain the remainder r. Thus

$$25 \pmod 7 = 4, \quad 25 \pmod 5 = 0, \quad 35 \pmod{11} = 2, \quad 3 \pmod 8 = 3$$

If k is negative, divide $|k|$ by M to obtain a remainder r'; then k (mod M) = M − r' when r' =0. Thus

$$-26 \pmod 7 = 7 - 5 = 2, \quad -371 \pmod 8 = 8 - 3 = 5, \quad -39 \pmod 3 = 0$$

The term "mod" is also used for the mathematical congruence relation, which is denoted and defined as follows:

$$a \equiv b \pmod M \quad \text{if any only if} \quad M \text{ divides } b - a$$

M is called the modulus, and $a \equiv b \pmod M$ is read "a is congruent to b modulo M". The following aspects of the congruence relation are frequently useful:

$$0 \equiv M \pmod M \quad \text{and} \quad a \pm M \equiv a \pmod M$$

Arithmetic modulo M refers to the arithmetic operations of addition, multiplication, and subtraction where the arithmetic value is replaced by its equivalent value in the set

$$\{0, 1, 2, \ldots, M-1\} \quad \text{or in the set} \quad \{1, 2, 3, \ldots, M\}$$

For example, in arithmetic modulo 12, sometimes called "clock" arithmetic,

$$6 + 9 \equiv 3, \quad 7 \times 5 \equiv 11, \quad 1 - 5 \equiv 8, \quad 2 + 10 \equiv 0 \equiv 12$$

(The use of 0 or M depends on the application.)

Exponential Functions

Recall the following definitions for integer exponents (where m is a positive integer):

$$a^m = a \cdot a \cdot a \ldots\ldots a(m \text{ times}), \quad a^0 = 1, \quad a^{-m} = \frac{1}{a^m}$$

Exponents are extended to include all rational numbers by defining, for any rational number m/n,

$$a^{m/n} = \sqrt[n]{a^m} = (\sqrt[n]{a})^m$$

For an example, $2^4 = 16, 2^{-4} = \frac{1}{2^4} = \frac{1}{16}, 125^{2/3} = 5^2 = 25$

In fact, exponents are extended to include all real numbers by defining, for any real number x,

$$a^x = \lim_{r \to x} a^r$$

Accordingly, the exponential function $f(x) = a^x$ is defined for all real numbers.

Logarithmic Functions

Logarithms are related to exponents as follows. Let b be a positive number. The logarithm of any positive number x to be the base b, written

$$\log_b x$$

represents the exponent to which b must be raised to obtain x. That is, $y = \log_b x$ and $b^y = x$ are equivalent statements. Accordingly,

$\log_2 8 = 3$ since $2^3 = 8$; $\log_{10} 100 = 2$ since $10^2 = 100$

$\log_2 64 = 6$ since $2^6 = 64$; $\log_{10} 0.001 = -3$ since $10^{-3} = 0.001$

Furthermore, for any base b, we have $b^0 = 1$ and $b^1 = b$; hence

$$\log_b 1 = 0 \quad \text{and} \quad \log_b b = 1$$

The logarithm of a negative number and the logarithm of 0 are not defined.

Frequently, logarithms are expressed using approximate values. For example, using tables or calculators, one obtains $\log_{10} 300 = 2.4771$ and $\log_e 40 = 3.6889$ as approximate answers. (Here e = 2.718281....)

Three classes of logarithms are of special importance: logarithms to base 10, called common logarithms; logarithms to base e, called natural logarithms; and logarithms to base 2, called binary logarithms. Some texts write

$$\ln x \text{ for } \log_e x \text{ and } \quad \lg x \text{ or } \log x \text{ for } \log_2 x$$

The term log x, by itself, usually means $\log_{10} x$; but it is also used for $\log_e x$ in advanced mathematical texts and for $\log_2 x$ in computer science texts.

Frequently, we will require only the floor or the ceiling of a binary logarithm. This can be obtained by looking At the powers of 2. For example,

$$[\log_2 100] = 6 \quad \text{since} \quad 2^6 = 64 \quad \text{and} \quad 2^7 = 128$$
$$[\log_2 1000] = 9 \quad \text{since} \quad 2^8 = 512 \quad \text{and} \quad 2^9 = 1024 \quad \text{and so an.}$$

Relationship between the Exponential and Logarithmic Functions

The basic relationship between the exponential and the logarithmic functions

$$f(x) = b^x \quad \text{and} \quad g(x) = \log_b x$$

is that they are inverses of each other; hence the graphs of these functions are related geometrically. This relation-ship is illustrated in Fig. 1-5 where the graphs of the exponential function f (x) = 2^x , the logarithmic function g(x) = $\log_2 x$, and the linear function h(x) = x appear on the same coordinate axis. Since f (x) = 2^x and g(x) = $\log_2 x$ are inverse functions, they are symmetric with respect to the linear function h(x) = x or, in other words, the line y = x.

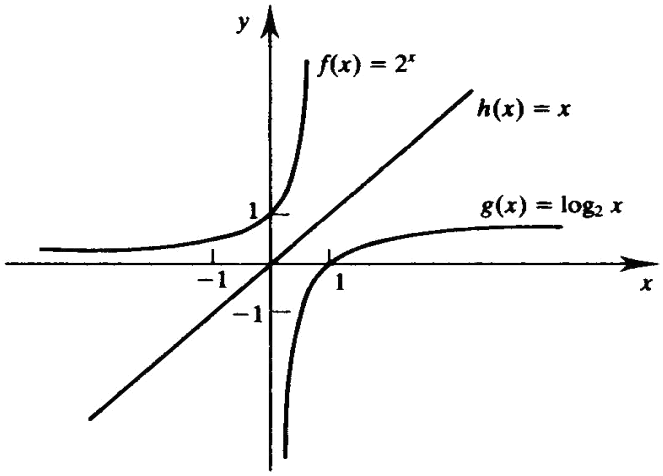


Fig. 1-5

Figure 1-5 also indicates another important property of the exponential and logarithmic functions. Specifically, for any positive c, we have

$$g(c) < h(c) < f(c), \quad \text{that is,} \quad g(c) < c < f(c)$$

In fact, as c increases in value, the vertical distances h(c) − g(c) and f (c) − g(c) increase in value. Moreover, the logarithmic function g(x) grows very slowly compared with the linear function h(x), and the exponential function f (x) grows very quickly compared with h(x).

1.6 SEQUENCES, INDEXED CLASSES OF SETS

Sequences and indexed classes of sets are special types of functions with their own notation. We discuss these objects in this section. We also discuss the summation notation here.

A sequence is a function from the set **N** = {1, 2, 3, . . .} of positive integers into a

set A. The notation a_n is used to denote the image of the integer n. Thus a sequence is usually denoted by

$$a_1, a_2, a_3, \ldots \quad \text{or} \quad \{a_n : n \in \mathbf{N}\} \quad \text{or} \quad \text{simply} \quad \{a_n\}$$

Sometimes the domain of a sequence is the set {0, 1, 2, . . .} of nonnegative integers rather than **N**. In such a ease we say n begins with 0 rather than 1.

A finite sequence over a set A is a function from {1, 2, . . . , m} into A, and it is usually denoted by $a_1, a_2, \ldots, a_m$

Such a finite sequence is sometimes called a list or an m-tuple.

EXAMPLE 1.5

(a) The following are two familiar sequences:

(i) $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$ which may be defined by $a_n = \frac{1}{n}$;

(ii) $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, , \ldots$ which may be defined by $b_n = 2^{-n}$

Note that the first sequence begins with n = 1 and the second sequence begins with n = 0.

(b) The important sequence 1, −1, 1, −1, . . . may be formally defined by

$$a_n = (-1)^{n+1} \quad \text{or, equivalently, by} \quad b_n = (-1)^n$$

where the first sequence begins with n = 1 and the second sequence begins with n = 0.

(c) Strings Suppose a set A is finite and A is viewed as a character set or an alphabet. Then a finite sequence over A is called a string or word, and it is usually written in the form $a_1a_2 \ldots a_m$, that is, without parentheses. The number m of characters in the string is called its length. One also views the set with zero characters as a string; it is called the empty string or null string. Strings over an alphabet A and certain operations on these strings will be

discussed in detail in last unit.

Summation Symbol, Sums

Here we introduce the summation symbol, Σ (the Greek letter sigma). Consider a sequencea1, a2, a3, ...

Then we define the following:

$$\sum_{j=1}^{n} a_j = a_1 + a_2 + ... + a_n and \sum_{j=m}^{n} a_j = a_m + a_{m+1} + ... + a_n$$

The letter j in the above expressions is called a dummy index or dummy variable. Other letters frequently used as dummy variables are i, k, s, and t.

EXAMPLE 1.6

$$\sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + ... + a_n b_n$$

$$\sum_{j=2}^{5} j^2 = 2^2 + 3^2 + 4^2 + 5^2 = 4 + 9 + 16 + 25 = 54$$

$$\sum_{j=1}^{n} j = 1 + 2 + ... + n$$

The last sum appears very often. It has the value n(n+1)/2. That is

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} \text{ for example } 1 + 2 + 3 + \cdots + 50 = \frac{50(51)}{2} = 1275$$

Indexed Classes of Sets

Let I be any nonempty set, and let S be a collection of sets. An indexing function from I to S is a function f : I → S. For any i ∈ I , we denote the image f (i) by A_i . Thus the indexing function f is usually denoted by
$\{A_i \mid i \in I\}$ or $\{A_i\}_{i \in I}$ or simply $\{A_i\}$

The set I is called the indexing set, and the elements of I are called indices. If f is

one-to-one and onto, we say that S is indexed by I.

The concepts of union and intersection are defined for indexed classes of sets as follows:

$$\cup_{i\in I} A_i = \{x \mid x \in A_i \text{ for some } i \in I\} \quad \text{and} \quad \cap_{i\in I} A_i = \{x \mid x \in A_i \text{ for all } i \in I\}$$

In the case that I is a finite set, this is just the same as our previous definition of union and intersection. If I is **N**, we may denote the union and intersection, respectively, as follows:

$$A_1 \cup A_2 \cup A_3 \cup \ldots \quad \text{and} \quad A_1 \cap A_2 \cap A_3 \cap \ldots$$

EXAMPLE 1.7 Let I be the set **Z** of integers. To each n ∈ **Z**, we assign the following infinite interval in **R**:

$$A_n = \{x \mid x \le n\} = (-\infty, n]$$

For any real number a, there exists integers n_1 and n_2 such that $n_1 < a < n_2$; so $a \in A_{n2}$ but $a \notin A_{n1}$. Hence

$$a \in \bigcup_n A_n \qquad \text{But} \quad a \notin \bigcap_n A_n$$

Accordingly,

$$\bigcup_n A_n = R \text{ but} \bigcap_n A_n = \phi$$

1.7 RECURSIVE FUNCTIONS

A function is said to be recursively defined if the function definition refers to itself. In order for the definition not to be circular, the function definition must have the following two properties:

(1) There must be certain arguments, called base values, for which the function does not refer to itself.

(2) Each time the function does refer to itself, the argument of the function

must be closer to a base value.

A recursive function with these two properties is said to be well-defined.

The following examples should help clarify these ideas.

Factorial Function

The product of the positive integers from 1 to n, inclusive, is called "n factorial" and is usually denoted by n!. That is,

$$n! = n(n-1)(n-2)\cdots 3\cdot 2\cdot 1$$

It is also convenient to define 0! = 1, so that the function is defined for all nonnegative integers. Thus:

$0! = 1,\quad 1! = 1,\quad 2! = 2\cdot 1 = 2,\qquad 3! = 3\cdot 2\cdot 1 = 6,\ 4! = 4\cdot 3\cdot 2\cdot 1 = 24$

$5! = 5\cdot 4\cdot 3\cdot 2\cdot 1 = 120,\qquad 6! = 6\cdot 5\cdot 4\cdot 3\cdot 2\cdot 1 = 720$

And so on. Observe that

$$5! = 5\cdot 4! = 5\cdot 24 = 120 \text{ and } 6! = 6\cdot 5! = 6\cdot 120 = 720$$

This is true for every positive integer n; that is,

$$n! = n\cdot(n-1)!$$

Accordingly, the factorial function may also be defined as follows:

DEFINITION 1.2 (Factorial Function):

(a) If n = 0, then n! = 1.

(b) If n > 0, then $n! = n\cdot(n-1)!$

Observe that the above definition of n! is recursive, since it refers to itself when it uses (n − 1)!. However:

(1) The value of n! is explicitly given when n = 0 (thus 0 is a base value).

(2) The value of n! for arbitrary n is defined in terms of a smaller value of n which is closer to the base value 0.

Accordingly, the definition is not circular, or, in other words, the function is well-defined.

EXAMPLE 1.8 Figure 1-6 shows the nine steps to calculate 4! using the recursive definition. Specifically:

Step 1. This defines 4! in terms of 3!, so we must postpone evaluating 4! until we evaluate 3. This postponement is indicated by indenting the next step.

Step 2. Here 3! is defined in terms of 2!, so we must postpone evaluating 3! until we evaluate 2!.

Step 3. This defines 2! in terms of 1!.

Step 4. This defines 1! in terms of 0!.

Step 5. This step can explicitly evaluate 0!, since 0 is the base value of the recursive definition.

Steps 6 to 9. We backtrack, using 0! to find 1!, using 1! to find 2!, using 2! to find 3!, and finally using 3! to find 4!. This backtracking is indicated by the "reverse" indention.

Observe that we backtrack in the reverse order of the original postponed evaluations.

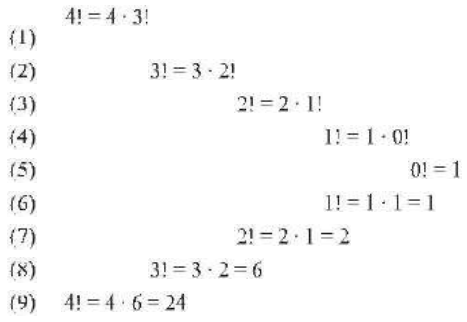


Fig. 1-6

Level Numbers

Let P be a procedure or recursive formula which is used to evaluate f (X) where f is a recursive function and X is the input. We associate a level number with each execution of P as follows. The original execution of P is assigned level 1; and each time P is executed because of a recursive call, its level is one more than the level of the execution that made the recursive call. The depth of recursion in evaluating f (X) refers to the maximum level number of P during its execution.

Consider, for example, the evaluation of 4! Example 3.8, which uses the recursive formula n! = n(n − 1)!. Step 1 belongs to level 1 since it is the first execution of the formula. Thus:

Step 2 belongs to level 2; Step 3 to level 3, . . . ; Step 5 to level 5.

On the other hand, Step 6 belongs to level 4 since it is the result of a return from level 5. In other words, Step 6 and Step 4 belong to the same level of execution. Similarly,

Step 7 belongs to level 3; Step 8 to level 2; and Step 9 to level 1.

Accordingly, in evaluating 4!, the depth of the recursion is 5.

Fibonacci Sequence

The celebrated Fibonacci sequence (usually denoted by F_0, F_1, F_2, . . .) is as follows:

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ \ldots$$

That is, $F_0 = 0$ and $F_1 = 1$ and each succeeding term is the sum of the two preceding terms. For example, the next two terms of the sequence are

$$34 + 55 = 89 \quad \text{and} \quad 55 + 89 = 144$$

A formal definition of this function follows:

DEFINITION 1.3 (Fibonacci Sequence):

(a) If $n = 0$, or $n = 1$, then $F_n = n$.
(b) If $n > 1$, then $F_n = F_{n-2} + F_{n-1}$.

This is another example of a recursive definition, since the definition refers to itself when it uses F_{n-2} and

However:

(1) The base values are 0 and 1.
(2) The value of F_n is defined in terms of smaller values of n which are closer to the base values.

Accordingly, this function is well-defined.

Ackermann Function

The Ackermann function is a function with two arguments, each of which can be assigned any nonnegative interger, that is, 0, 1, 2, ... This function is defined as:

DEFINITION 1.4 (Ackermann function):

(a) If m = 0, then A(m, n) = n + 1.

(b) If m ≠ 0 but n = 0, then A(m, n) = A(m − 1, 1).

(c) If m ≠ 0 and n ≠ 0, then A(m, n) = A(m − 1, A(m, n − 1)).

Once more, we have a recursive definition, since the definition refers to itself in parts (b) and (c). Observe that A(m, n) is explicitly given only when m = 0. The base criteria are the pairs

$$(0, 0),\quad (0, 1),\quad (0, 2),\quad (0, 3),\quad \ldots, (0, n),\quad \ldots$$

Although it is not obvious from the definition, the value of any A(m, n) may eventually be expressed in terms of the value of the function on one or more of the base pairs.

The value of A(1, 3) is calculated in Problem 1.21. Even this simple case requires 15 steps. Generally speaking, the Ackermann function is too complex to evaluate on any but a trivial example. Its importance comes from its use in mathematical logic. The function is stated here mainly to give another example of a classical recursive function and to show that the recursion part of a definition may be complicated.

1.8 CARDINALITY

Two sets A and B are said to be equipotent, or to have the same number of elements or the same cardinality, written A B, if there exists a one-to-one correspondence f : A → B. A set A is finite if A is empty or if A has the same cardinality as the set {1, 2, . . . , n} for some positive integer n. A set is infinite if it is not finite. Familiar examples of infinite sets are the natural numbers **N**, the integers **Z**, the rational numbers **Q**, and the real numbers **R**.

We now introduce the idea of "cardinal numbers". We will consider cardinal numbers simply as symbols assigned to sets in such a way that two sets are assigned the same symbol if and only if they have the same cardinality. The

cardinal number of a set A is commonly denoted by $|A|$, n(A), or card (A). We will use $|A|$.

The obvious symbols are used for the cardinality of finite sets. That is, 0 is assigned to the empty set $\emptyset$, and n is assigned to the set $\{1, 2, \ldots, n\}$. Thus $|A| = n$ if and only if A has n elements. For example,

$$|\{x, y, z\}| = 3 \quad \text{and} \quad |\{1, 3, 5, 7, 9\}| = 5$$

The cardinal number of the infinite set **N** of positive integers is $\aleph_0$ ("aleph-naught"). This symbol was introduced by Cantor. Thus $|A| = \aleph_0$ if and only if A has the same cardinality as **N**.

EXAMPLE 1.9 Let $E = \{2, 4, 6, \ldots\}$, the set of even positive integers. The function $f : \mathbf{N} \to E$ defined by $f(n) = 2n$ is a one-to-one correspondence between the positive integers **N** and E. Thus E has the same cardinality as **N** and so we may write

$$|E| = \aleph_0$$

A set with cardinality $\aleph_0$ is said to be denumerable or countably infinite. A set which is finite or denumerable is said to be countable. One can show that the set **Q** of rational numbers is countable. In fact, we have the following theorem (proved in Problem 3.13) which we will use subsequently.

THEOREM 1.2: A countable union of countable sets is countable.

That is, if $A_1, A_2, \ldots$ are each countable sets, then the following union is countable:

$$A_1 \cup A_2 \cup A_3 \cup \ldots$$

An important example of an infinite set which is uncountable, i.e., not countable, is given by the following theorem which is proved in Problem 1.14.

THEOREM 1.3: The set **I** of all real numbers between 0 and 1 is uncountable.

Inequalities and Cardinal Numbers

One also wants to compare the size of two sets. This is done by means of an inequality relation which is defined for cardinal numbers as follows. For any sets A and B, we define $|A| \leq |B|$ if there exists a function $f : A \to B$ which is one-to-one. We also write

$$|A| < |B| \quad \text{if} \quad |A| \leq |B| \quad \text{but} \quad |A| \neq |B|$$

For example, $|\mathbf{N}| < |\mathbf{I}|$, where $\mathbf{I} = \{x : 0 \leq x \leq 1\}$, since the function $f : \mathbf{N} \to \mathbf{I}$ defined by $f(n) = 1/n$ is one-to-one, but $|\mathbf{N}| = |\mathbf{I}|$ by Theorem 1.3.

Cantor's Theorem, which follows and which we prove in Problem 1.25, tells us that the cardinal numbers are unbounded.

Theorem 1.4 (Cantor): For any set A, we have $|A| < |\text{Power}(A)|$ (where Power(A) is the power set of A, i.e., the collection of all subsets of A).

The next theorem tells us that the inequality relation for cardinal numbers is anti-symmetric.

Theorem 1.5: (Schroeder-Bernstein): Suppose A and B are sets such that

$$|A| \leq |B| \quad \text{and} \quad |B| \leq |A|$$

Then $|A| = |B|$.

We prove an equivalent formulation of this theorem in Problem 1.26.

ALGORITHMS

An algorithm M is a finite step-by-step list of well-defined instructions for solving a particular problem, say, to find the output f (X) for a given function f with input X. (Here X may be a list or set of values.) Frequently, there may be more than one way to obtain f (X), as illustrated by the following examples. The particular choice of the algorithm M to obtain f (X) may depend on the "efficiency" or "complexity" of the algorithm; this question of the complexity of an algorithm M is formally discussed in the next section.

EXAMPLE 1.10 (Polynomial Evaluation). Suppose, for a given polynomial f (x) and value x = a, we want to find f (a), say,

$$f(x) = 2x^3 - 7x^2 + 4x - 15 \quad \text{and} \quad a = 5$$

This can be done in the following two ways.

(a) (**Direct Method**)**:** Here we substitute a = 5 directly in the polynomial to obtain

$$f(5) = 2(125) - 7(25) + 4(5) - 7 = 250 - 175 + 20 - 15 = 80$$

Observe that there are 3 + 2 + 1 = 6 multiplications and 3 additions. In general, evaluating a polynomial of degree n directly would require approximately

$$n + (n-1) + \cdots + 1 = \frac{n(n+1)}{2} \text{ multiplications and n additions.}$$

(b) (**Horner's Method or Synthetic Division**)**:** Here we rewrite the polynomial by successively factoring out x (on the right) as follows:

$$f(x) = (2x^2 - 7x + 4)x - 15 = ((2x - 7)x + 4)x - 15$$

Then

$$f(5) = ((3)5 + 4)5 - 15 = (19)5 - 15 = 95 - 15 = 80$$

For those familiar with synthetic division, the above arithmetic is equivalent to the following synthetic division:

5	2	−	7	+	4	− 15
			10	+	15	+ 95
	2	+	3	+	19	+ 80

Observe that here there are 3 multiplications and 3 additions. In general, evaluating a polynomial of degree n by Horner's method would require approximately

n multiplications and n additions

Clearly Horner's method (b) is more efficient than the direct method (a)

EXAMPLE 1.11 (Greatest Common Divisor) Let a and b be positive integers with, say, b < a; and suppose we want to find d = GCD (a, b), the greatest common divisor of a and b. This can be done in the following two ways.

(a) (**Direct Method**)**:** Here we find all the divisors of a, say by testing all the numbers from 2 to a/2, and all the divisors of b. Then we pick the largest common divisor. For example, suppose a = 258 and b = 60. The divisors of a and b follow:

a = 258; divisors : 1, 2, 3, 6, 86, 129, 258

b = 60; divisors : 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60

Accordingly, d = GCD (258, 60) = 6.

(b) (**Euclidean Algorithm**)**:** Here we divide a by b to obtain a remainder r_1. (Note $r_1 < b$.) Then we divide b by the remainder r_1 to obtain a second remainder r_2. (Note $r_2 < r_1$.) Next we divide r_1 by r_2 to obtain a third remainder r_3. (Note $r_3 < r_2$.) We continue dividing r_k by r_{k+1} to obtain a remainder r_{k+2}. Since

$$a > b > r_1 > r_2 > r_3 \ldots \qquad (*)$$

eventually we obtain a remainder $r_m = 0$. Then r_{m-1} = GCD (a, b). For example, suppose a = 258 and b = 60. Then:

(1) Dividing a = 258 by b = 60 yields the remainder $r_1 = 18$.

(2) Dividing b = 60 by $r_1 = 18$ yields the remainder $r_2 = 6$.

(3) Dividing $r_1 = 18$ by $r_2 = 6$ yields the remainder $r_3 = 0$.

Thus $r_2 = 6$ = GCD (258, 60).

The Euclidean algorithm is a very efficient way to find the greatest common divisor of two positive integers a and b. The fact that the algorithm ends follows from (∗). The fact that the algorithm yields d = GCD (a, b) is not obvious.

1.9 COMPLEXITY OF ALGORITHMS

The analysis of algorithms is a major task in computer science. In order to compare algorithms, we must have some criteria to measure the efficiency of our algorithms. This section discusses this important topic.

Suppose M is an algorithm, and suppose n is the size of the input data. The time and space used by the algorithm are the two main measures for the efficiency of M. The time is measured by counting the number of "key operations;" for example:

(a) In sorting and searching, one counts the number of comparisons.

(b) In arithmetic, one counts multiplications and neglects additions.

Key operations are so defined when the time for the other operations is much less than or at most proportional to the time for the key operations. The space is measured by counting the maximum of memory needed by the algorithm.

The complexity of an algorithm M is the function f (n) which gives the running time and/or storage space requirement of the algorithm in terms of the size n of the input data. Frequently, the storage space required by an algorithm is simply a multiple of the data size. Accordingly, unless otherwise stated or implied, the term "complexity" shall refer to the running time of the algorithm.

The complexity function f (n), which we assume gives the running time of an algorithm, usually depends not only on the size n of the input data but also on the particular data. For example, suppose we want to search through an English short story TEXT for the first occurrence of a given 3-letter word W. Clearly, if W is the 3-letter word "the," then W likely occurs near the beginning of TEXT, so f (n) will be small. On the other hand, if W is the 3-letter word "zoo," then W may not appear in TEXT at all, so f (n) will be large.

The above discussion leads us to the question of finding the complexity function f (n) for certain cases. The two cases one usually investigates in complexity theory are as follows:

(1) Worst case: The maximum value of f (n) for any possible input.

(2) Average case: The expected value of f (n).

The analysis of the average case assumes a certain probabilistic distribution for the input data; one possible assumption might be that the possible permutations of a data set are equally likely. The average case also uses the following concept in probability theory. Suppose the numbers $n_1, n_2, \ldots, n_k$ occur with respective probabilities $p_1, p_2, \ldots, p_k$. Then the expectation or average value E is given by

$$E = n_1p_1 + n_2p_2 + \cdots + n_k p_k$$

These ideas are illustrated below.

1.10 LINEAR SEARCH

Suppose a linear array DATA contains n elements, and suppose a specific ITEM of information is given. We want either to find the location LOC of ITEM in the array DATA, or to send some message, such as LOC = 0, to indicate that ITEM does not appear in DATA. The linear search algorithm solves this problem by comparing ITEM, one by one, with each element in DATA. That is, we compare ITEM with DATA[1], then DATA[2], and so on, until we find LOC such that ITEM = DATA[LOC].

The complexity of the search algorithm is given by the number C of comparisons between ITEM and DATA[K]. We seek C(n) for the worst case and the average case.

(1) **Worst Case**: Clearly the worst case occurs when ITEM is the last element in the array DATA or is not there at all. In either situation, we have

$$C(n) = n$$

Accordingly, C(n) = n is the worst-case complexity of the linear search algorithm.

(2) **Average Case**: Here we assume that ITEM does appear in DATA, and that it is equally likely to occur at any position in the array. Accordingly, the number of comparisons can be any of the numbers 1, 2, 3, . . . , n, and each number occurs with probability p = 1/n. Then

$$C(n) = 1.\frac{1}{n} + 2.\frac{1}{n} + \ldots\ldots\ldots + n.\frac{1}{n}$$

$$= (1 + 2 + \ldots\ldots\ldots + n).\frac{1}{n}$$

$$= \frac{n(n+1)}{2}.\frac{1}{n}$$

$$= \frac{n+1}{2}$$

This agrees with our intuitive feeling that the average number of comparisons needed to find the location of ITEM is approximately equal to half the number of elements in the DATA list.

Remark: The complexity of the average case of an algorithm is usually much more complicated to analyze than that of the worst case. Moreover, the probabilistic distribution that one assumes for the average case may not actually apply to real situations. Accordingly, unless otherwise stated or implied, the complexity of an algorithm shall mean the function which gives the running time of the worst case in terms of the input size. This is not too strong an assumption, since the complexity of the average case for many algorithms is proportional to the worst case.

Rate of Growth; Big O Notation

Suppose M is an algorithm, and suppose n is the size of the input data. Clearly the complexity f (n) of M increases as n increases. It is usually the rate of increase of f (n) that we want to examine. This is usually done by comparing f (n) with some standard function, such as

$$\log n, \quad n, \quad n \log n, \quad n^2, \quad n^3, \quad 2^n$$

The rates of growth for these standard functions are indicated in Fig. 1-7, which gives their approximate values for certain values of n. Observe that the functions are listed in the order of their rates of growth: the logarithmic function

$\log_2 n$ grows most slowly, the exponential function 2^n grows most rapidly, and the polynomial functions n^c grow according to the exponent c.

n \ $g(n)$	$\log n$	n	$n \log n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^6	10^9	10^{300}

Fig. 1-7 Rate of growth of standard functions

The way we compare our complexity function f (n) with one of the standard functions is to use the functional "big O" notation which we formally define below.

DEFINITION 1.5: Let f (x) and g(x) be arbitrary functions defined on **R** or a subset of **R**. We say "f (x) is of order g(x)," written

$$f(x) = O(g(x))$$

if there exists a real number k and a positive constant C such that, for all $x > k$, we have

$$|f(x)| \leq C|g(x)|$$

In other words, f (x) = 0(g(x)) if a constant multiple of |g(x)| exceeds |f (x)| for all x greater than some real number k

.

We also write:

$$f(x) = h(x) + O(g(x)) \qquad \text{when} \quad f(x) - h(x) = O(g(x))$$

(The above is called the "big O" notation since f (x) = o(g(x)) has an entirely meaning.)m

Consider now a polynomial P (x) of degree m. We show in Problem 3.24 that P (x) = O(x

Thus, for example,

$$7x^2 - 9x + 4 = O(x^2) \quad \text{and} \quad 8x^3 - 576x^2 + 832x - 248 = O(x^3)$$

Complexity of Algorithms

Assuming f(n) and g(n) are functions defined on the positive integers, then

$$f(n) = O(g(n))$$

means that f (n) is bounded by a constant multiple of g(n) for almost all n.

To indicate the convenience of this notation, we give the complexity of certain well-known searching and sorting algorithms in computer science:

(a) Linear search: O(n)
(b) Binary search: O(log n)
(c) Bubble sort: $O(n^2)$
(d) Merge-sort: O(n log n)

Solved Problems

FUNCTIONS

1.1Let X = {1, 2, 3, 4}. Determine whether each relation on X is a function from X into X.

(a) f = {(2, 3), (1, 4), (2, 1), (3.2), (4, 4)}

(b) g = {(3, 1), (4, 2), (1, 1)}

(c) h = {(2, 1), (3, 4), (1, 4), (2, 1), (4, 4)}

Recall that a subset f of X × X is a function f : X → X if and only if each a ∈ X appears as the first coordinate in exactly one ordered pair in f.

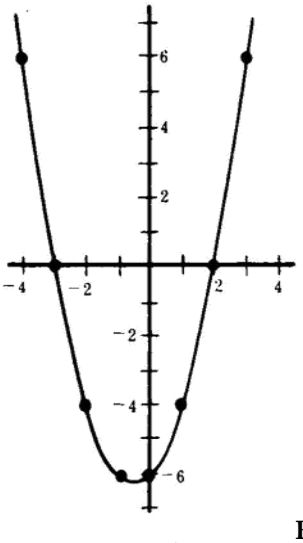
(a) No. Two different ordered pairs (2, 3) and (2, 1) in f have the same number 2 as their first coordinate. (b) No. The element 2 ∈ X does not appear as the first coordinate in any ordered pair in g.

(c) Yes. Although 2 ∈ X appears as the first coordinate in two ordered pairs in h, these two ordered pairs are equal.

1.2. Sketch the graph of: (a) $f(x) = x^2 + x - 6$; (b) $g(x) = x^3 - 3x^2 - x + 3$.

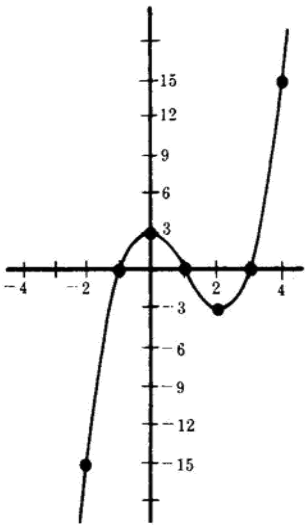
Set up a table of values for x and then find the corresponding values of the function. Since the functions are polynomials, plot the points in a coordinate diagram and then draw a smooth continuous curve through the points. in Fig. 1-8

x	$f(x)$
−4	6
−3	0
−2	−4
−1	−6
0	−6
1	−4
2	0
3	6



Graph of $f = x^2 + x - 6$

x	$g(x)$
−2	−15
−1	0
0	3
1	0
2	−3
3	0
4	15



Graph of $g = x^3 - 3x^2 - x + 3$

Fig. 1-8

1.3. Let A = {a, b, c}, B = {x, y, z}, C = {r, s, t }. Let f : A → B and g: B → C be defined by:

$$f = \{(a, y)(b, x), (c, y)\} \quad \text{and} \quad g = \{(x, s), (y, t), (z, r)\}.$$

Find: (a) composition function $g \circ f : A \to C$; (b) Im(f), Im(g), Im($g \circ f$).

(a) Use the definition of the composition function to compute:

$(g \circ f)(a) = g(f(a)) = g(y) = t$
$(g \circ f)(b) = g(f(b)) = g(x) = s$
$(g \circ f)(c) = g(f(c)) = g(y) = t$

That is $g \circ f = \{(a, t), (b, s), (c, t)\}$.

(b) Find the image points (or second coordinates):

Im(f) = {x, y}, Im(g) = {r, s, t }, Im($g \circ f$) = {s, t }

1.4 Let f : **R** → **R** and g: **R** → **R** be defined by $f(x) = 2x + 1$ and $g(x) = x^2 - 2$. Find the formula for the composition function $g \circ f$.

Compute $g \circ f$ as follows: $(g \circ f)(x) = g(f(x)) = g(2x + 1) = (2x + 1)^2 - 2 = 4x^2 + 4x - 1$.

Observe that the same answer can be found by writing

$$y = f(x) = 2x + 1 \quad \text{and} \quad z = g(y) = y^2 - 2$$

and then eliminating y from both equations:

$$z = y^2 - 2 = (2x + 1)^2 - 2 = 4x^2 + 4x - 1$$

ONE-TO-ONE, ONTO, AND INVERTIBLE FUNCTIONS

1.5 1et the functions f : A → B, g: B → C, h: C → D be defined by Fig. 1-9. Determine if each function is: (a) onto, (b) one-to-one, (c) invertible.

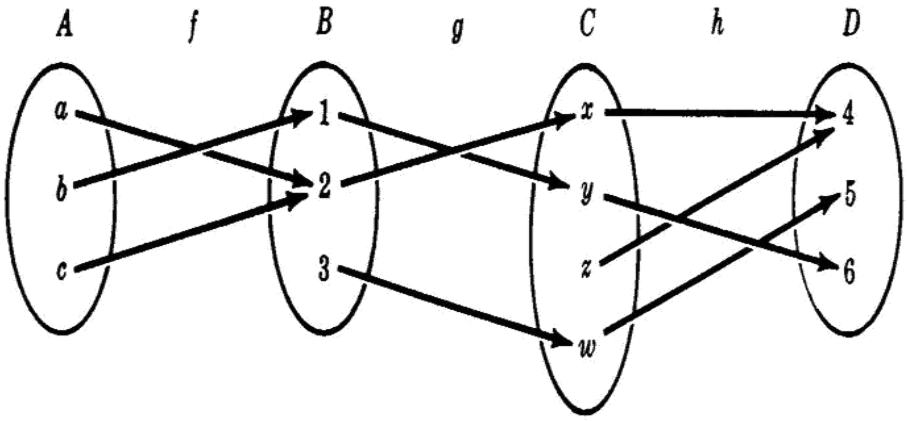


Fig. 1-9

(a) The function f : A → B is not onto since 3 ∈ B is not the image of any element in A. The function g: B → C is not onto since z ∈ C is not the image of any element in B.

(b) The function h: C → D is onto since each element in D is the image of some element of C. (b) The function f : A → B is not one-to-one since a and c have the same image 2.

(c) The function g: B → C is one-to-one since 1, 2 and 3 have distinct images. The function h: C → D is not one-to-one since x and z have the same image 4.

(d) No function is one-to-one and onto; hence no function is invertible.

1.6. Consider permutations

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 6 & 4 & 5 & 1 & 2 \end{pmatrix} \text{and } \tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 5 & 3 & 1 \end{pmatrix}$$

In S_6 Find: (a) composition $\tau \circ \sigma$; (b) σ^{-1}.

(a) Note that σ sends 1 into 3 and τ sends 3 into 6. So the composition $\tau \circ \sigma$ sends 1 into 6. I.e. $(\tau \circ \sigma)(1) = 6$. Moreover, $\tau \circ \sigma$ sends 2 into 6 into 1 that is, $(\tau \circ \sigma)(2) = 1$, Similarly,

$$(\tau \circ \sigma)(3) = 5, \quad (\tau \circ \sigma)(4) = 3, \quad (\tau \circ \sigma) = 2, \quad (\tau \circ \sigma)(6) = 4$$

Thus

$$\tau \circ \sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 6 & 5 & 3 \end{pmatrix}$$

(b)Look for 1 in the second row of σ . Note σ sends 5 into 1. Hence $\sigma^{-1}(1) = 5$. Look for 2 in the second row of σ . Note σ sends 6 into 2. Hence $\sigma^{-1}(2) = 6$. Similarly, $\sigma^{-1}(3) = 1$, $\sigma^{-1}(4) = 3$, $\sigma^{-1}(5) = 4$, $\sigma^{-1}(6) = 2$. Thus

$$\sigma^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 5 & 3 & 1 \end{pmatrix}$$

1.7 Consider functions $f : A \to B$ and $g: B \to C$. Prove the following:
(a) If f and g are one-to-one, then the composition function $g \circ f$ is one-to-one.
(b) If f and g are onto functions, then $g \circ f$ is an onto function.

(a) Suppose $(g \circ f)(x) = (g \circ f)(y)$; then $g(f(x)) = g(f(y))$. Hence $f(x) = f(y)$ because g is one-to-one. Further-more, $x = y$ since f is one-to-one. Accordingly $g \circ f$ is one-to-one.

(b) Let c be any arbitrary element of C. Since g is onto, there exists a $b \in B$ such that $g(b) = c$. Since f is onto, there exists an $a \in A$ such that $f(a) = b$. But then

$$(g \circ f)(a) = g(f(a)) = g(b) = c$$

Hence each $c \in C$ is the image of some element $a \in A$. Accordingly, $g \circ f$ is an onto function.

1.8. Let f : **R** → **R** be defined by f (x) = 2x − 3. Now f is one-to-one and onto; hence f has an inverse function f^{-1}. Find a formula for f^{-1}.

Let y be the image of x under the function f :

$$y = f(x) = 2x - 3$$

Consequently, x will be the image of y under the inverse function f^{-1}. Solve for x in terms of y in the above equation:

$$x = (y + 3)/2$$

Then $f^{-1}(y) = (y + 3)/2$. Replace y by x to obtain $f^{-1}(y) = \frac{x+3}{2}$ which is the formula for f^{-1} using the usual independent variable x.

1.9. Prove the following generalization of DeMorgan's law: For any class of sets $\{A_i\}$ we have

$$(\cup_i A_i)^c = \cap_i A^c_i$$

We have:

$$x \in (\cup_i A_i)^c \text{ iff } x \notin \cup_i A_i, \text{ iff } \forall_i \in I, x \notin A_i, \text{ iff } \forall_i \in I, x \in A^c_i, \text{ iff } x \in \cap_i A^c_i$$

Therefore, $(\cup_i A_i)^c = \cap_i A^c_i$. (Here we have used the logical notations iff for "if and only" if and ∀ for "for all.")

CARDINALITY

1.10. Find the cardinal number of each set:

(a) A = {a, b, c, . . . , y, z}, (c) C = {10, 20, 30, 40, . . .},
(b) B = {x | x ∈ **N**, $x^2 = 5$}, (d) D = {6, 7, 8, 9, . . .}.

(a) |A| = 26 since there are 26 letters in the English alphabet.
(b) |B| = 0 since there is no positive integer whose square is 5, that is, B is empty.
|C| = $\aleph_0$ because f : **N** → C, defined by $f(n) = 10_n$, is a one-to-one correspondence
© between **N** and C.
|D| = $\aleph_0$ because g: **N** → D, defined by g(n) = n + 5 is a one-to-one correspondence
(d) between **N** and D.

1.11 How that the set **Z** of integers has cardinality $\aleph_0$.

The following diagram shows a one-to-one correspondence between **N** and **Z**:

$$\begin{array}{lccccccccccc} \mathbf{N} = & 1 & & 2 & & 34 & & 56 & & 78 & \ldots \\ & \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \ldots \\ \mathbf{Z} = & 0 & & 1 & & -1 & 2 & -2 & 3 & -3 & 4 \quad \ldots \end{array}$$

That is, the following function $f : \mathbf{N} \to \mathbf{Z}$ is one-to-one and onto:

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ (1-n)/2 & \text{if } n \text{ is odd} \end{cases}$$

Accordingly, $|\mathbf{Z}| = |\mathbf{N}| = \aleph_0$.

1.12. Let $A_1, A_2, \ldots$ be a countable number of finite sets. Prove that the union $S = \cup_i A_i$ is countable.

Essentially, we list the elements of A_1, then we list the elements of A_2 which do not belong to A_1, then we list the elements of A_3 which do not belong to A_1 or A_2, i.e., which have not already been listed, and so on. Since the A_i are finite, we can always list the elements of each set. This process is done formally as follows.

First we define sets $B_1, B_2, \ldots$ where B_i contains the elements of A_i which do not belong to preceding sets, i.e., we define

$$B_1 = A_1 \quad \text{and} \quad B_k = A_k \setminus (A_1 \cup A_2 \cup \cdots \cup A_{k-1})$$

Then the B_i are disjoint and $S = \cup_i B_i$. Let $b_{i1}, b_{i2}, \ldots, b_{im}$, be the elements of B_i . Then $S = \{b_{ij}\}$. Let $f : S \to \mathbf{N}$ be defined as follows:

$$f(b_{ij}) = m_1 + m_2 + \cdots + m_{i-1} + j$$

If S is finite, then S is countable. If S is infinite then f is a one-to-one correspondence between S and **N**. Thus S is countable.

1.13. Prove Theorem 1..2: A countable union of countable sets is countable.

Suppose A_1, A_2, A_3, . . . are a countable number of countable sets. In particular, suppose a_{i1}, a_{i2}, a_{i3}, . . . are the elements of A_i . Define sets B_2, B_3, B_4, . . . as follows:

$$B_k = \{a_{ij} \mid i + j = k\}$$

For example, $B_6 = \{a_{15}, a_{24}, a_{33}, a_{12}, a_{51}\}$. Observe that each B_k is finite and

$$S = \cup_i A_i = \cup_k B_k$$

By the preceding problem $\cup_k B_k$ is countable. Hence $S = \cup_i A_i$ is countable and the theorem is proved.

1.14. Prove Theorem 1.3: The set **I** of all real numbers between 0 and 1 inclusive is uncountable.

The set **I** is clearly infinite, since it contains 1, $\frac{1}{2}$, $\frac{1}{3}$, ... Suppose **I** is denumerable. Then there exists a one-to-one correspondence f : N → **I**. Let f (1) = a_1, f (2) = a_2, . . .; that is, **I** = $\{a_1, a_2, a_3, . . .\}$. We list the elements a_1, a_2, ... in a column and express each in its decimal expansion:

$$\begin{aligned} a_1 &= 0.x_{11}x_{12}x_{13}x_{14} \ldots \\ a_2 &= 0.x_{21}x_{22}x_{23}x_{24} \ldots \\ a_3 &= 0.x_{31}x_{32}x_{33}x_{34} \ldots \\ a_4 &= 0.x_{41}x_{42}x_{43}x_{44} \ldots \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \end{aligned}$$

where $x_{ij} \in \{0, 1, 2, . . . , 9\}$. (For those numbers which can be expressed in two different decimal expansions, e.g., 0.2000000 . . . = 0.1999999 . . ., we choose the expansion which ends with nines.)

Let $b = 0.y_1y_2y_3y_4 \ldots$ be the real number obtained as follows:

$$y_i = \begin{cases} 1 & \text{if } x_{ii} \neq 1 \\ 2 & \text{if } x_{ii} = 1 \end{cases}$$

Now $b \in \mathbf{I}$. But

$b \neq a_1$ because $y_1 \neq x_{11}$
$b \neq a_2$ because $y_2 \neq x_{22}$

$b \neq a_3$ because $y_3 \neq x_{33}$
.....

Therefore b does not belong to $\mathbf{I} = \{a_1, a_2, \ldots\}$. This contradicts the fact that $b \in \mathbf{I}$. Hence the assumption that **I** is denumerable must be false, so **I** is uncountable.

SPECIAL MATHEMATICAL FUNCTIONS

1.15 Find: (a) [7.5], [−7.5], [−18]; (b)[7.5], [−7.5], [−18].

(a) By definition, [x] denotes the greatest integer that does not exceed x, hence [7.5] = 7, [−7.5] = −8, [18] = −18.

(b) By definition, [x] denotes the least integer that is not less than x, hence [7.5] = 8, [−7.5] = −7, [−18] = −18.

1.16. Find: (a) 25 (mod7); (b) 25 (mod5); (c) −35 (mod 11); (d) −3 (mod 8).

When k is positive, simply divide k by the modulus M to obtain the remainder r. Then r = k(mod M). If k is negative, divide |k| by M to obtain the remainder r . Then k(mod M) = M − r (when r _=0). Thus:

(a) 25 (mod 7) = 4 (c) −35 (mod 11) = 11 − 2 = 9
(b) 25 (mod 5) = 0 (d) −3 (mod 8) = 8 − 3 = 5

1.17 Evaluate modulo M = 15: (a) 9 + 13; (b) 7 + 11; (c) 4 − 9; (d) 2 − 10.
Use a ± M = a(mod M):

(a) 9 + 13 = 22 = 22 − 15 = 7 (c) 4 − 9 = −5 = −5 + 15 = 10
(b) 7 + 11 = 18 = 18 − 15 = 3 (d) 2 − 10 = −8 = −8 + 15 = 7

1.18 Simplify: (a) $\frac{n!}{(n-1)!}$; (b) $\frac{(n+2)!}{n!}$

(a) $\frac{n!}{(n-1)!} = \frac{n(n-1)(n-2)\ldots 3 .2 .1}{(n-1)(n-2)\ldots 3 .2 .1}$ = n or, simply, $\frac{n!}{(n-1)!} = \frac{n(n-1)!}{(n-1)!}$ = n

(b) $\frac{(n+2)!}{n!} = \frac{n(n+2)(n+1)n!}{n!}$ = (n + 2)(n - 1) = n^2 + 3n + 2

1.19. Evaluate: (a) $\log_2 8$; (b) $\log_2 64$; (c) $\log_{10} 100$; (d) $\log_{10} 0.001$.

(a) $\log_2 8 = 3$ since $2^3 = 8$ (c) $\log_{10} 100 = 2$ since $10^2 = 100$

(b) $\log_2 64 = 6$ since $2^6 = 64$ (d) $\log_{10} 0.001 = -3$ since $10^{-3} = 0.001$

RECURSIVE FUNCTIONS

1.20. Let a and b be positive integers, and suppose Q is defined recursively as follows:

$$Q(a,b) = \begin{cases} 0, & if\ a < b \\ Q(a-b,b) + 1, & if\ b \le a \end{cases}$$

(i) Q(2, 5); (ii) Q(12, 5).

(b) What does this function Q do? Find Q(5861, 7).

(a) (i) Q(2, 5) = 0 since 2 < 5.

(ii) Q(12, 5) = Q(7, 5) + 1

= [Q(2, 5) + 1] + 1 = Q(2, 5) + 2
= 0 + 2 = 2

(c) Each time b is subtracted from a, the value of Q is increased by 1. Hence Q(a, b) finds the quotient when a is divided by b. Thus Q(5861, 7) = 837.

1.21. Use the definition of the Ackermann function to find A(1, 3).

Figure 1-10 shows the 15 steps that are used to evaluate A(1, 3).

The forward indention indicates that we are postponing an evaluation and are recalling the definition, and the backward indention indicates that we are backtracking. Observe that (a) of the definition is used in Steps 5, 8, 11 and 14; (b) in Step 4; and (c) in Steps 1, 2, and 3. In the other steps we are backtracking with substitutions.

(1) $A(1,3) = A(0, A(1,2))$
(2) $A(1,2) = A(0, A(1,1))$
(3) $A(1,1) = A(0, A(1,0))$
(4) $A(1,0) = A(0,1)$
(5) $A(0,1) = 1 + 1 = 2$
(6) $A(1,0) = 2$
(7) $A(1,1) = A(0,2)$
(8) $A(0,2) = 2 + 1 = 3$
(9) $A(1,1) = 3$
(10) $A(1,2) = A(0,3)$
(11) $A(0,3) = 3 + 1 = 4$
(12) $A(1,2) = 4$
(13) $A(1,3) = A(0,4)$
(14) $A(0,4) = 4 + 1 = 5$
(15) $A(1,3) = 5$

Fig. 1-10

PROBLEMS

1.22. Find the domain D of each of the following real-valued functions of a real variable:

(a) $f(x) = \frac{1}{x-2}$ (c) $f(x) = \sqrt{25 - x^2}$

(b) $f(x) = x^{2-} - 3x - 4$ (d) x^2 where $0 \le x \le 2$

When a real-valued function of a real variable is given by a formula f (x), then the domain D consists of the largest subset of **R** for which f (x) has meaning and

is real, unless otherwise specified.

(a) f is not defined for $x - 2 = 0$, i.e., for $x = 2$; hence $D = \mathbf{R}\backslash\{2\}$.
(b) f is defined for every real number; hence $D = \mathbf{R}$.
(c) f is not defined when $25 - x^2$ is negative; hence $D = [-5, 5] = \{x \mid -5 \le x \le 5\}$.
(d) Here, the domain of f is explicitly given as $D = \{x \mid 0 \le x \le 2\}$.

1.23. For any $n \in \mathbf{N}$, let $D_n = (0, 1/n)$, the open interval from 0 to $1/n$. Find:

(a) $D_3 \cup D_4$; (b) $D_3 \cap D_{20}$; (c) $D_s \cup D_t$; (d) $D_s \cap D_t$.

(a) Since $(0, 1/3)$ is a superset of $(0, 1/7)$, $D_3 \cup D_4 = D_3$.
(b) Since $(0, 1/20)$ is a subset of $(0, 1/3)$, $D_3 \cap D_{20} = D_{20}$.
(c) Let $m = \min(s, t)$, that is, the smaller of the two numbers s and t; then D_m is equal to D_s or D_t contains the other as a subset. Hence $D_s \cap D_t = D_m$.
(c) Let $M = \max(s, t)$, that is, the larger of the two numbers s and t; then $D_s \cap D_t = D_m$.

1.24. Suppose $P(n) = a_0 + a_1n + a_2n^2 + \cdots + a_mn^2$ has degree m. Prove $P(n) = O(n^m)$.

Let $b_0 = |a_0|, b_1 = |a_1|, \ldots, b_m = |a_m|$. Then for $n \ge 1$,

$$P(n) \le b_0 + b_1\, n + b_2\, n^2 + \cdots + b_m\, n^m =$$

$$\left(\frac{b_0}{n^m} + \frac{b_1}{n^{m-1}} + \cdots + b_m\right) n^m \le$$

$$\le (b_0 + b_1 + \cdots . + b_m) n^m = M n^m$$

where $M = |a_0| + |a_1| + \cdots + |a_m|$. Hence $P(n) = O(n^m)$.

For example, $5x^3 + 3x = O(x^3)$ and $x^5 - 4000000x^2 = O(x^5)$.

1.25. Prove Theorem 1.4 (Cantor): $|A| < |\text{Power}(A)|$ (where Power (A) is the power set of A). function $g: A \to \text{Power}(A)$ defined by $g(a) = \{a\}$ is clearly one-to-one; hence $|A| \le |\text{Power}(A)|$.

If we show that $|A| \_= |\text{Power}(A)|$, then the theorem will follow. Suppose the contrary, that is, suppose $|A| = |\text{Power}(A)|$ and that $f : A \to \text{Power}(A)$ is a function which is both one-to-one and onto. Let $a \in A$ be called a "bad" element

if $a \in / f(a)$, and let B be the set of bad elements. In other words,

$$B = \{x : x \in A, x \in / f(x)\}$$

Now B is a subset of A. Since $f : A \to \text{Power}(A)$ is onto, there exists $b \in A$ such that $f(b) = B$. Is b a "bad" element or a "good" element? If $b \in B$ then, by definition of B, $b \in / f(b) = B$, which is impossible. Likewise, if $b \notin B$ then $b \in f(b) = B$, which is also impossible. Thus the original assumption that $|A| = |\text{Power}(A)|$ has led to a contradiction. Hence the assumption is false, and so the theorem is true.

1.26. Prove the following equivalent formulation of the Schroeder–Bernstein Theorem 1.5: Suppose $X \supseteq Y \supseteq X_1$ and $X \simeq X_1$. Then $X \simeq Y$.

Since $X \simeq X_1$ there exists a one-to-one correspondence (bijection) $f : X \to X_1$ Since $X \supseteq Y$, the restriction of f to Y, which we also denote by f, is also one-to-one. Let $f(Y) = Y_1$. Then Y and Y_1 are equipotent,

$$X \supseteq Y \supseteq X_1 \supseteq Y_1$$

and $f : Y \to Y_1$ is bijective. But now $Y \supseteq X_1 \supseteq Y_1$ and $Y \_ Y_1$. For similar reasons, X_1 and $f(X_1) = X_2$ are equipotent,

$$X \supseteq Y \supseteq X_1 \supseteq Y_1 \supseteq X_2$$

and $f : X_1 \to X_2$ is bijective. Accordingly, there exist equipotent sets $X, X_1, X_2, \ldots$ and equipotent sets $Y, Y_1, Y_2, \ldots$

such that

$$X \supseteq Y \supseteq X_1 \supseteq Y_1 \supseteq X_2 \supseteq Y_2 \supseteq X_3 \supseteq Y_3 \supseteq \cdots$$

and $f : X_k \to X_{k+1}$ and $f : Y_k \to Y_{k+1}$ are bijective.

Let

$$B = X \cap Y \cap X_1 \cap Y_1 \cap X_2 \cap Y_2 \cap \cdots$$

Then

$$X = (X \setminus Y) \cup (Y \setminus X_1) \cup (X_1 \setminus Y_1) \cup \cdots \cup B$$

$$Y = (Y \setminus X_1) \cup (X_1 \setminus Y_1) \cup (Y_1 \setminus X_2) \cup \cdots \cup B$$

Furthermore, $X \setminus Y$, $X_1 \setminus Y_1$, $X_2 \setminus Y_2$, . . . are equipotent. In fact, the function

$$f : (X_k \setminus Y_k) \to (X_{k+1} \setminus Y_{k+1})$$

is one-to-one and onto.

Consider the function $g: X \to Y$ defined by the diagram in Fig. 1-11. That is,

$$g(x) = \begin{cases} f(x) & \text{if } x \in X_k \setminus Y_k \text{ or } x \in X \setminus Y \\ x & \text{if } x \in Y_k \setminus X_k \text{ or } x \in B \end{cases}$$

Then g is one-to-one and onto. Therefore $X \simeq Y$.

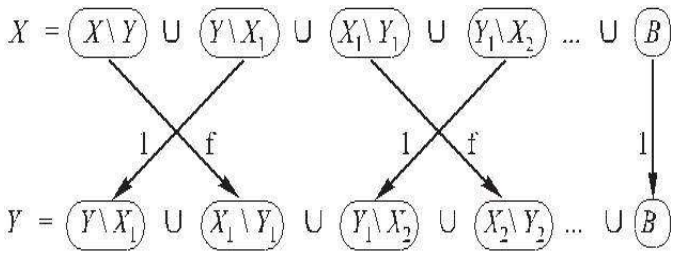


Fig. 1-11

Solved Problems

FUNCTIONS

1.27. Let W = {a, b, c, d}. Decide whether each set of ordered pairs is a function from W into W.

(a) {(b, a), (c, d), (d, a), (c, d) (a, d)}

(b) {(d, d), (c, a), (a, b), (d, b)}

(c) {(a, b), (b, b), (c, d), (d, b)}

(d) {(a, a), (b, a), (a, b), (c, d)}

1.28 Let V = {1, 2, 3, 4}. For the following functions f : V → V and g: V → V , find:

(a) f ◦g; (b), g◦f ; (c) f ◦f :

f = {(1, 3), (2, 1), (3, 4), (4, 3)} and g = {(1, 2), (2, 3), (3, 1), (4, 1)}

1.29. Find the composition function h ◦ g ◦ f for the functions in Fig. 1-9.

ONE-TO-ONE, ONTO, AND INVERTIBLE FUNCTIONS

1.30 Determine if each function is one-to-one.

(a) To each person on the earth assign the number which corresponds to his age.

(b) To each country in the world assign the latitude and longitude of its capital.

(c) To each book written by only one author assign the author.

(d) To each country in the world which has a prime minister assign its prime minister.

1.31 Let functions f, g, h from V = {1, 2, 3, 4} into V be defined by: f (n) = 6 − n, g(n) = 3, h = {(1, 2), (2, 3), (3, 4), (4, 1)}. Decide which functions are:

(a) one-to-one; (b) onto; (c) both; (d) neither.

1.32 Let functions f, g, h from **N** into **N** be defined by f (n) = n + 2, (b) $g(n) = 2^n$; h(n) = number of positive divisors of n. Decide which functions are:

(a) one-to-one; (b) onto; (c) both; (d) neither; (e) Find h (2) = {x | h(x) = 2}.

1.33. Decide which of the following functions are: (a) one-to-one; (b) onto; (c) both; (d) neither.

(1) f : $\mathbf{Z}^2 \rightarrow \mathbf{Z}$ where f (n, m) = n − m; **(3)** h: $\mathbf{Z} \times (\mathbf{Z} \backslash 0) \rightarrow \mathbf{Q}$ where h(n, m) = n/m;
(2) g: $\mathbf{Z}^2 \rightarrow \mathbf{Z}^2$ where g(n, m) = (m, n); **(4)** k: $\mathbf{Z} \rightarrow \mathbf{Z}^2$ where k(n) = (n, n).

1.34. Let $f : \mathbf{R} \to \mathbf{R}$ be defined by $f(x) = 3x - 7$. Find a formula for the inverse function $f^{-1}: \mathbf{R} \to \mathbf{R}$.

1.35. Consider permutations

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 5 & 6 & 1 & 3 & 4 \end{pmatrix}$$

$$\tau \circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 4 & 3 & 1 & 2 & 5 \end{pmatrix}$$

Find: (a) $\tau \circ \sigma$; (b) $\sigma \circ \tau$; (c) σ^2; (d) σ^{-1}; (e) τ^{-1}

PROPERTIES OF FUNCTIONS

1.36 Prove: Suppose $f : A \to B$ and $g: B \to A$ satisfy $g \circ f = 1_A$. Then f is one-to-one and g is onto.

1.37 Prove Theorem 1.1: A function $f : A \to B$ is invertible if and only if f is both one-to-one and onto.

1.38 Prove: Suppose $f : A \to B$ is invertible with inverse function $f^{-1}: B \to A$. Then $f^{-1} \circ f = 1_A$ and $f \circ f^{-1} = 1_B$ **1.39** Suppose $f : A \to B$ is one-to-one and $g : A \to B$ is onto. Let x be a subset of A.

(a) Show $f1_x$, the restriction of f to x, is one-to-one.

(b) Show $g1_x$, need not be onto.

1.40 For each $n \in \mathbf{N}$, consider the open interval $A_n = (0, 1/n) = \{x \mid 0 < x < 1/n\}$. Find:

(a) $A_2 \cup A_8$; (c) $\cup(A_i \mid i \in J)$; (e) $\cup(A_i \mid i \in K)$;

(b) $A_3 \cap A_7$; (d) $\cap(A_i \mid i \in J)$; (f) $\cap(A_i \mid i \in K)$.

where J is a finite subset of **N** and **K** is an infinite subset of **N**.

1.41 For each $n \in N$, let $D_n = \{n, 2n, 3n, \ldots\} = \{\text{multiples of } n\}$.

(a) Find: (i) $D_2 \cap D_7$; (ii) $D_6 \cap D_8$; (iii) $D_3 \cap D_{12}$; (iv) $D_3 \cup D_{12}$.

(b) Prove that $\cap(D_i \mid i \in K) = \emptyset$ where K is an infinite subset of **N**.

1.42 Consider an indexed class of sets $\{A_i \mid i \in I\}$, a set **B** and an index i_0 in I. Prove:

(a) $B \cap (\cup_i A_i) = \cup_i (B \cap A_i)$; (b) $\cap (A_i \mid i \in I) \subseteq A_{i0} \subseteq \cup(A_i \mid i \in I)$.

CARDINAL NUMBERS

1.43 Find the cardinal number of each set: (a) {x | x is a letter in "BASEBALL"};

(b) Power set of A = {a, b, c, d, e}; (c) $\{x \mid x^2 = 9, 2x = 8\}$.

1.44 Find the cardinal number of:

(a) all functions from A = {a, b, c, d} into B = {1, 2, 3, 4, 5};
(b) all functions from P into Q where $|P| = r$ and $|Q| = s$;
(c) all relations on A = {a, b, c, d};
(d) all relations on P where $|P| = r$.

1.45 Prove:

(a) Every infinite set A contains a denumerable subset D.
(b) Each subset of a denumerable set is finite or denumerable.
(c) If A and B are denumerable, then A × B is denumerable.
(e) The set **Q** of rational numbers is denumerable.

1.46 Prove: (a) $|A \times B| = |B \times A|$; (b) If $A \subseteq B$ then $|A| \leq |B|$; (c) If $|A| = |B|$ then $P(A)| = |P(B)|$.

CHAPTER 2

LOGIC AND PROPOSITIONAL CALCULUS

2.1 INTRODUCTION

Propositional calculus is the branch of mathematical logic concerned with the study of propositions that are formed by other propositions with the use of logical connectives, and how their value depends on the truth value of their components. Logical connectives are found in natural languages. In English for example, some examples are "and" (conjunction), "or" (disjunction), "not" (negation) and "if" (but only when used to denote material conditional). Propositional logic, also known as sentential logic and statement logic, is the branch of logic that studies ways of joining and/or modifying entire propositions, statements or sentences to form more complicated propositions, statements or sentences, as well as the logical relationships and properties that are derived from these methods of combining or altering statements.

In propositional logic, the simplest statements are considered as indivisible units, and hence, propositional logic does not study those logical properties and relations that depend upon parts of statements that are not themselves statements on their own, such as the subject and predicate of a statement. The most thoroughly researched branch of propositional logic is classical truth-functional propositional logic, which studies logical operators and connectives that are used to produce complex statements whose truth-value depends entirely on the truth-values of the simpler statements making them up, and in which it is assumed that every statement is either true or false and not both. However, there are other forms of propositional logic in which other truth-values are considered, or in which there is consideration of connectives that are used to produce statements whose truth-values depend not simply on the truth-values of the parts, but additional things such as their necessity, possibility or relatedness to one another.

The following is an example of a very simple inference within the scope of propositional logic:

Premise 1: If it's raining then it's cloudy.
Premise 2: It's raining.
Conclusion: It's cloudy.

Both premises and the conclusions are propositions. The premises are taken for granted and then with the application of modus ponens (an inference rule) the conclusion follows.

As propositional logic is not concerned with the structure of propositions beyond the point where they can't be decomposed anymore by logical connectives, this inference can be restated replacing those atomic statements with statement letters, which are interpreted as variables representing statements:

Many algorithms and proofs use logical expressions such as:

"IF p THEN q" or "If p_1 AND p_2, THEN q_1 OR q_2"

Therefore it is necessary to know the cases in which these expressions are TRUE or FALSE, that is, to know the "truth value" of such expressions. We discuss these issues in this chapter.

We also investigate the truth value of quantified statements, which are statements which use the logical quantifiers "for every" and "there exist."

2.2 PROPOSITIONS

A proposition (or statement) is a declarative statement which is true or false, but not both. Consider, for example, the following six sentences:

(i) Ice floats in water.	(iii) 2 + 2 = 4	(v) Where are you going?
(ii) China is in Europe.	(iv) 2 + 2 = 5	(vi) Do your homework.

The first four are propositions, the last two are not. Also, (i) and (iii) are true, but (ii) and (iv) are false.

Compound Propositions

Many propositions are composite, that is, composed of sub-propositions and various connectives discussed subsequently. Such composite propositions are called compound propositions. A proposition is said to be primitive if it cannot be broken down into simpler propositions, that is, if it is not composite.

For example, the above propositions (i) through (iv) are primitive propositions. On the other hand, the following two propositions are composite:

"Roses are red and violets are blue." and "John is smart or he studies every night."

The fundamental property of a compound proposition is that its truth value is completely determined by the truth values of its sub-propositions together with the way in which they are connected to form the compound propositions. The next section studies some of these connectives.

2.3 BASIC LOGICAL OPERATIONS

This section discusses the three basic logical operations of conjunction, disjunction, and negation which correspond, respectively, to the English words "and," "or," and "not."

Conjunction, $p \wedge q$

Any two propositions can be combined by the word "and" to form a compound proposition called the conjunction of the original propositions. Symbolically,

$$p \wedge q$$

read "p and q," denotes the conjunction of p and q. Since $p \wedge q$ is a proposition it has a truth value, and this truth value depends only on the truth values of p and q. Specifically:

DEFINITION 2.1: If p and q are true, then $p \wedge q$ is true; otherwise $p \wedge q$ is false.

The truth value of p ∧ q may be defined equivalently by the table in Fig. 2-1(a). Here, the first line is a short way of saying that if p is true and q is true, then p ∧ q is true. The second line says that if p is true and q is false, then p ∧ q is false. And so on. Observe that there are four lines corresponding to the four possible combinations of T and F for the two sub-propositions p and q. Note that p ∧ q is true only when both p and q are true.

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

(*a*) "*p* and *q*"

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

(*b*) "*p* or *q*"

p	$\neg p$
T	F
F	T

(*c*) "not *p*"

Fig. 2-1

EXAMPLE 2.1 Consider the following four statements:

(i) Ice floats in water and 2 + 2 = 4.
(ii) Ice floats in water and 2 + 2 = 5.
(iii) China is in Europe and 2 + 2 = 4.
(iv) China is in Europe and 2 + 2 = 5.

Only the first statement is true. Each of the others is false since at least one of its sub-statements is false.

Disjunction, p ∨ q

Any two propositions can be combined by the word "or" to form a compound proposition called the disjunction of the original propositions. Symbolically,

p ∨ q

read "p or q," denotes the disjunction of p and q. The truth value of p ∨ q depends only on the truth values of p and q as follows.

DEFINIITION 2.2: If p and q are false, then p ∨ q is false; otherwise p ∨ q is true.

The truth value of $p \vee q$ may be defined equivalently by the table in Fig. 2-1(b). Observe that $p \vee q$ is false only in the fourth case when both p and q are false.

EXAMPLE 2.2 Consider the following four statements:

(i) Ice floats in water or 2 + 2 = 4.
(ii) Ice floats in water or 2 + 2 = 5.
(iii) China is in Europe or 2 + 2 = 4.
(iv) China is in Europe or 2 + 2 = 5.

Only the last statement (iv) is false. Each of the others is true since at least one of its sub-statements is true.

Remark: The English word "or" is commonly used in two distinct ways. Sometimes it is used in the sense of "p or q or both," i.e., at least one of the two alternatives occurs, as above, and sometimes it is used in the sense of "p or q but not both," i.e., exactly one of the two alternatives occurs. For example, the sentence "He will go to Harvard or to Yale" uses "or" in the latter sense, called the exclusive disjunction. Unless otherwise stated, "or" shall be used in the former sense. This discussion points out the precision we gain from our symbolic language: $p \vee q$ is defined by its truth table and always means "p and/or q."

Negation, $\neg p$

Given any proposition p, another proposition, called the negation of p, can be formed by writing "It is not true that . . ." or "It is false that . . ." before p or, if possible, by inserting in p the word "not." Symbolically, the negation of p, read "not p," is denoted by

$$\neg p$$

The truth value of $\neg p$ depends on the truth value of p as follows:

DEFINITION 2.3: If p is true, then $\neg p$ is false; and if p is false, then $\neg p$ is true.

The truth value of $\neg p$ may be defined equivalently by the table in Fig. 2-1(c). Thus the truth value of the negation of p is always the opposite of the truth value

of p.

EXAMPLE 2.3 Consider the following six statements:

(a_1) Ice floats in water. (a_2) It is false that ice floats in water. (a_3) Ice does not float in water.

(b_1) 2 + 2 = 5 (b_2) It is false that 2 + 2 = 5. (b_3) 2 + 2 $\neq$ 5

Then (a_2) and (a_3) are each the negation of (a_1); and (b_2) and (b_3) are each the negation of (b_1). Since (a_1) is true, (a_2) and (a_3) are false; and since (b_1) is false, (b_2) and (b_3) are true.

Remark: The logical notation for the connectives "and," "or," and "not" is not completely standardized. For example, some texts use:

p & q, p ·q or pq	for	p ∧ q
p + q	for	p ∨ q
p', $\bar{p}$ or ~ p	for	¬ p

2.4 TRUTH TABLES

Let P (p, q, . . .) denote an expression constructed from logical variables p, q, . . ., which take on the value TRUE (T) or FALSE (F), and the logical connectives ∧, ∨, and ¬ (and others discussed subsequently). Such an expression P (p, q, . . .) will be called a proposition.

The main property of a proposition P (p, q, . . .) is that its truth value depends exclusively upon the truth values of its variables, that is, the truth value of a proposition is known once the truth value of each of its variables is known. A simple concise way to show this relationship is through a truth table. We describe a way to obtain such a truth table below.

Consider, for example, the proposition ¬(p ∧ ¬q). Figure 2-2(a) indicates how the truth table of ¬(p ∧ ¬q) is constructed. Observe that the first columns of the table are for the variables p, q, . . . and that there are enough rows in the table, to allow for all possible combinations of T and F for these variables. (For 2

variables, as above, 4 rows are necessary; for 3 variables, 8 rows are necessary; and, in general, for n variables, 2^n rows are required.) There is then a column for each "elementary" stage of the construction of the proposition, the truth value at each step being determined from the previous stages by the definitions of the connectives ∧, ∨, ¬. Finally we obtain the truth value of the proposition, which appears in the last column.

The actual truth table of the proposition ¬(p ∧ ¬q) is shown in Fig. 2-2(b). It consists precisely of the columns in Fig. 2-2(a) which appear under the variables and under the proposition; the other columns were merely used in the construction of the truth table.

p	q	$\neg q$	$p \wedge \neg q$	$\neg(p \wedge \neg q)$
T	T	F	F	T
T	F	T	T	F
F	T	F	(a) F	T
F	F	T	F	T

p	q	$\neg(p \wedge \neg q)$
T	T	T
T	F	F
F	T (b)	T
F	F	T

Fig. 2-2

Remark: In order to avoid an excessive number of parentheses, we sometimes adopt an order of precedence for the logical connectives. Specifically,

¬ has precedence over ∧ which has precedence over ∨

For example, ¬p ∧ q means (¬p) ∧ q and not ¬(p ∧ q).

Alternate Method for Constructing a Truth Table

Another way to construct the truth table for ¬(p ∧ ¬q) follows:

(a) First we construct the truth table shown in Fig. 2-3. That is, first we list all the variables and the com-binations of their truth values. Also there is a final row labeled "step." Next the proposition is written on the top row to the right of its variables with sufficient space so there is a column under each variable and under each logical operation in the proposition. Lastly (Step 1), the truth values of the variables are entered in the table under the

variables in the proposition.

(b) Now additional truth values are entered into the truth table column by column under each logical operation as shown in Fig. 2-4. We also indicate the step in which each column of truth values is entered in the table.

The truth table of the proposition then consists of the original columns under the variables and the last step, that is, the last column is entered into the table.

p	q	$\neg$	$(p$	$\wedge$	$\neg$	$q)$
T	T		T			T
T	F		T			F
F	T		F			T
F	F		F			F
Step						

Fig. 2-3

p	q	$\neg$	$(p$	$\wedge$	$\neg$	$q)$
T	T		T		F	T
T	F		T		T	F
F	T		F		F	T
F	F		F		T	F
Step			1		2	1

(a)

p	q	$\neg$	$(p$	$\wedge$	$\neg$	$q)$
T	T		T	F	F	T
T	F		T	T	T	F
F	T		F	F	F	T
F	F		F	F	T	F
Step			1	3	2	1

(b)

p	q	$\neg$	$(p$	$\wedge$	$\neg$	$q)$
T	T	T	T	F	F	T
F	T	F	T	T	T	F
F	F	T	F	F	F	T
F	F	T	F	F	T	F
Step		4	1	3	2	1

(c)

Fig. 2-4

2.5 TAUTOLOGIES AND CONTRADICTIONS

Some propositions P (p, q, . . .) contain only T in the last column of their truth tables or, in other words, they are true for any truth values of their variables. Such propositions are called tautologies. Analogously, a proposition P (p, q, . . .) is called a contradiction if it contains only F in the last column of its truth table or, in other words, if it is false for any truth values of its variables. For example, the proposition "p or not p," that is, $p \vee \neg p$, is a tautology, and the proposition "p and not p," that is, $p \wedge \neg p$, is a contradiction. This is verified by looking at their truth tables in Fig. 2-5. (The truth tables have only two rows since each proposition has only the one variable p.)

p	$\neg p$	$p \vee \neg p$
T	F	T
F	T	T

(a) $p \vee \neg p$

p	$\neg p$	$p \wedge \neg p$
T	F	F
F	T	F

(b) $p \wedge \neg p$

Fig. 2-5

Note that the negation of a tautology is a contradiction since it is always false, and the negation of a contradiction is a tautology since it is always true.

Now let P (p, q, . . .) be a tautology, and let P_1(p, q, . . .), P_2(p, q, . . .), . . . be any propositions. Since P (p, q, . . .) does not depend upon the particular truth values of its variables p, q, . . ., we can substitute P_1 for p, P_2 for q, . . . in the tautology P (p, q, . . .) and still have a tautology. In other words:

THEOREM 2.1 (Principle of Substitution): If P (p, q, . . .) is a tautology, then P (P_1, P_2, . . .) is a tautology for any propositions P_1, P_2, ...

2.6 LOGICAL EQUIVALENCE

Two propositions P (p, q, . . .) and Q(p, q, . . .) are said to be logically equivalent, or simply equivalent or equal, denoted by

$$P(p, q, \ldots) \equiv Q(p, q, \ldots)$$

if they have identical truth tables. Consider, for example, the truth tables of $\neg(p \wedge q)$ and $\neg p \vee \neg q$ appearing in Fig. 2-6. Observe that both truth tables are the same, that is, both propositions are false in the first case and true in the other three cases. Accordingly, we can write

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

In other words, the propositions are logically equivalent.

Remark: Let p be "Roses are red" and q be "Violets are blue." Let S be the statement:

"It is not true that roses are red and violets are blue."

Then S can be written in the form ¬(p ∧ q). However, as noted above, ¬(p ∧ q) ≡ ¬p ∨ ¬q. Accordingly, S has the same meaning as the statement:

"Roses are not red, or violets are not blue."

p	q	$p \wedge q$	$\neg(p \wedge q)$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

(a) $\neg(p \wedge q)$

p	q	$\neg p$	$\neg q$	$\neg p \vee \neg q$
T	T	F	F	F
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

(b) $\neg p \vee \neg q$

Fig. 2-6

2.7 ALGEBRA OF PROPOSITIONS

Propositions satisfy various laws which are listed in Table 2-1. (In this table, T and F are restricted to the truth values "True" and "False," respectively.) We state this result formally.

THEOREM 2.2: Propositions satisfy the laws of Table 2-1.

Idempotent laws:	(1a) $p \vee p \equiv p$	(1b) $p \wedge p \equiv p$
Associative laws:	(2a) $(p \vee q) \vee r \equiv p \vee (q \vee r)$	(2b) $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Commutative laws:	(3a) $p \vee q \equiv q \vee p$	(3b) $p \wedge q \equiv q \wedge p$
Distributive laws:	(4a) $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	(4b) $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Identity laws:	(5a) $p \vee F \equiv p$ (6a) $p \vee T \equiv T$	(5b) $p \wedge T \equiv p$ (6b) $p \wedge F \equiv F$
Involution law:	(7) $\neg\neg p \equiv p$	
Complement laws:	(8a) $p \vee \neg p \equiv T$ (9a) $\neg T \equiv F$	(8b) $p \wedge \neg p \equiv T$ (9b) $\neg F \equiv T$
DeMorgan's laws:	(10a) $\neg(p \vee q) \equiv \neg p \wedge \neg q$	(10b) $\neg(p \wedge q) \equiv \neg p \vee \neg q$

Table 2-1 Laws of the algebra of propositions

2.8 CONDITIONAL AND BICONDITIONAL PROPOSITIONS

Many statements, particularly in mathematics, are of the form "If p then q." Such statements are called conditional statements and are denoted by

$$p \rightarrow q$$

The conditional $p \rightarrow q$ is frequently read "p implies q" or "p only if q."

Another common statement is of the form "p if and only if q." Such statements are called bi-conditional statements and are denoted by

$$p \leftrightarrow q$$

The truth values of $p \rightarrow q$ and $p \leftrightarrow q$ are defined by the tables in Fig. 2-7(a) and (b). Observe that:

(a) The conditional p → q is false only when the first part p is true and the second part q is false. Accordingly, when p is false, the conditional p → q is true regardless of the truth value of q.

(b) The bi-conditional p ↔ q is true whenever p and q have the same truth values and false otherwise.

The truth table of ¬p ∧ q appears in Fig. 2-7(c). Note that the truth table of ¬p ∨ q and p → q are identical, that is, they are both false only in the second case. Accordingly, p → q is logically equivalent to ¬p ∨ q; that is,

$$p \rightarrow q \equiv \neg p \vee q$$

In other words, the conditional statement "If p then q" is logically equivalent to the statement "Not p or q" which only involves the connectives ∨ and ¬ and thus was already a part of our language. We may regard p → q as an abbreviation for an oft-recurring statement.

p	q	$p \rightarrow q$
T	T	**T**
T	F	F
F	T	**T**
F	F	**T**

(a) $p \rightarrow q$

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

(b) $p \leftrightarrow q$

p	q	$\neg p$	$\neg p \vee q$
T	T	F	T
T	F	F	F
F	T	**T**	T
F	F	**T**	T

(c) $\neg p \vee q$

Fig. 2-7

2.9 ARGUMENTS

An argument is an assertion that a given set of propositions $P_1, P_2, \ldots, P_n$, called premises, yields (has a consequence) another proposition Q, called the conclusion. Such an argument is denoted by

$$P_1, P_2, \ldots, P_n, \vdash Q$$

The notion of a "logical argument" or "valid argument" is formalized as follows:

DEFINITION 2.4: An argument $P_1, P_2, \ldots, P_n \vdash Q$ is said to be valid if Q is true whenever all the premises $P_1, P_2, \ldots, P_n$ are true.

An argument which is not valid is called fallacy.

EXAMPLE 2.4

(a) The following argument is valid:

$$p, p \rightarrow q \vdash q \quad (\textbf{Law of Detachment})$$

The proof of this rule follows from the truth table in Fig. 2-7(a). Specifically, p and $p \rightarrow q$ are true simultaneously only in Case (row) 1, and in this case q is true.

(b) The following argument is a fallacy:

$$p \rightarrow q, q \vdash p$$

For $p \rightarrow q$ and q are both true in Case (row) 3 in the truth table in Fig. 2-7(a), but in this case p is false.

Now the propositions $P_1, P_2, \ldots, P_n$ are true simultaneously if and only if the proposition $P_1 \wedge P_2 \wedge \ldots P_n$ is true. Thus the argument $P_1, P_2, \ldots, P_n \$ Q$ is valid if and only if Q is true whenever $P_1 \wedge P_2 \wedge \ldots \wedge P_n$ is true or, equivalently, if the proposition $(P_1 \wedge P_2 \wedge \ldots \wedge P_n) \rightarrow Q$ is a tautology. We state this result formally.

THEOREM 2.3: The argument $P_1, P_2, \ldots, P_n \$ Q$ is valid if and only if the proposition $(P_1 \wedge P_2 \ldots \wedge P_n) \rightarrow Q$ is a tautology.

We apply this theorem in the next example.

EXAMPLE 2.5 A fundamental principle of logical reasoning states:

"If p implies q and q implies r, then p implies r

p	q	r	$[(p$	$\rightarrow$	$q)$	$\wedge$	$(q$	$\rightarrow$	$r)]$	$\rightarrow$	$(p$	$\rightarrow$	$r)$
T	T	T	T	T	T	T	T	T	T	T	T	T	T
T	T	F	T	T	T	F	T	F	F	T	T	F	F
T	F	T	T	F	F	F	F	T	T	T	T	T	T
T	F	F	T	F	F	F	F	T	F	T	T	F	F
F	T	T	F	T	T	T	T	T	T	T	F	T	T
F	T	F	F	T	T	F	T	F	F	T	F	T	F
F	F	T	F	T	F	T	F	T	T	T	F	T	T
F	F	F	F	T	F	T	F	T	F	T	F	T	F
Step			1	2	1	3	1	2	1	4	1	2	1

Fig. 2-8

That is, the following argument is valid:

$$p \rightarrow q,\ q \rightarrow r \vdash\ p \rightarrow r \quad (\textbf{Law of Syllogism})$$

This fact is verified by the truth table in Fig. 2-8 which shows that the following proposition is a tautology:

$$[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$$

Equivalently, the argument is valid since the premises $p \rightarrow q$ (rows) 1, 5,7, and 8, and $q \rightarrow r$ are true simultaneously only in Cases is also true.(Observe that the truth table required and in these cases the conclusion $p \rightarrow r$ $2^3 = 8$ lines since there are three variables p, q, and r.)

EXAMPLE:

From (p implies q) and (q implies r) we conclude (p implies r). This is called the law of syllogism.

If p = The canal is open.
q = The ship can go through the canal.
r = The company can transport their goods.

If p => q and q => r, then p => r, so:

If the canal is open, then the ship can go through the canal.
If the ship can go through the canal, then the company can transport their goods.

You can use the law of syllogism to conclude that:
If the canal is open, then the company
Can transport their goods.
If a triangle has angles of 30° and 60°, then its third angle is
90°. If an angle in a triangle is 90°, then it is a right triangle.

You can use the law of syllogism to conclude that:

If a triangle has angles of 30° and 60°, then it is a
Right triangle.

We now apply the above theory to arguments involving specific statements. We emphasize that the validity of an argument does not depend upon the truth values nor the content of the statements appearing in the argument, but upon the particular form of the argument. This is illustrated in the following example.

EXAMPLE 2.6 Consider the following argument:

S_1 : If a man is a bachelor, he is unhappy.
S_2 : If a man is unhappy, he dies young.

S : Bachelors die young

Here the statement S below the line denotes the conclusion of the argument, and the statements S_1 and S_2 above the line denote the premises. We claim that the argument $S_1, S_2 \$ S$ is valid. For the argument is of the form

$$p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$$

where p is "He is a bachelor," q is "He is unhappy" and r is "He dies young;" and by Example 2.5 this argument (Law of Syllogism) is valid.

2.10 PROPOSITIONAL FUNCTIONS, QUANTIFIERS

Let A be a given set. A propositional function (or an open sentence or condition) defined on A is an expression

$$p(x)$$

which has the property that p(a) is true or false for each $a \in A$. That is, p(x) becomes a statement (with a truth value) whenever any element $a \in A$ is substituted for the variable x. The set A is called the domain of p(x), and the set T_p of all elements of A for which p(a) is true is called the truth set of p(x). In other words,

$$T_p = \{x \mid x \in A, p(x) \text{ is true}\} \quad \text{or} \quad T_p = \{x \mid p(x)\}$$

Frequently, when A is some set of numbers, the condition p(x) has the form of an equation or inequality involving the variable x.

EXAMPLE 2.7 Find the truth set for each propositional function p(x) defined on the set **N** of positive integers.

(a) Let p(x) be "x + 2 > 7." Its truth set is {6, 7, 8, . . .} consisting of all integers greater than 5.

(b) Let p(x) be "x + 5 < 3." Its truth set is the empty set . That is, p(x) is not true for any integer in **N**.

(c) Let p(x) be "x + 5 > 1." Its truth set is **N**. That is, p(x) is true for every element in **N**.

Remark: The above example shows that if p(x) is a propositional function defined on a set A then p(x) could be true for all $x \in A$, for some $x \in A$, or for no $x \in A$. The next two subsections discuss quantifiers related to such propositional functions.

Universal Quantifier

Let p(x) be a propositional function defined on a set A. Consider the expression:

$$(\forall x \in A)p(x) \quad \text{or} \quad \forall x\, p(x) \qquad (2.1)$$

which reads "For every x in A, p(x) is a true statement" or, simply, "For all x, p(x)." The symbol

$$\forall$$

which reads "for all" or "for every" is called the universal quantifier. The

statement (4.1) is equivalent to the statement

$$T_p = \{x \mid x \in A, p(x)\} = A \qquad (2.2)$$

that is, that the truth set of p(x) is the entire set A.

The expression p(x) by itself is an open sentence or condition and therefore has no truth value. However, ∀x p(x), that is p(x) preceded by the quantifier ∀, does have a truth value which follows from the equivalence of (2.1) and (2.2). Specifically:

Q_1: If $\{x \mid x \in A, p(x)\} = A$ then ∀x p(x) is true; otherwise, ∀x p(x) is false.

EXAMPLE 2.8

(a) The proposition $(\forall n \in \mathbf{N})(n + 4 > 3)$ is true since $\{n \mid n + 4 > 3\} = \{1, 2, 3, \ldots\} = \mathbf{N}$.

(b) The proposition $(\forall n \in \mathbf{N})(n + 2 > 8)$ is false since $\{n \mid n + 2 > 8\} = \{7, 8, \ldots\} \neq \mathbf{N}$.

(c) The symbol ∀ can be used to define the intersection of an indexed collection $\{A_i \mid i \in I\}$ of sets A_i as follows:

$$\cap(A_i \mid i \in I) = \{x \mid \forall_i \in I, x \in A_i\}$$

Existential Quantifier

Let p(x) be a propositional function defined on a set A. Consider the expression:

$$(\exists x \in A)p(x) \quad \text{or} \quad \exists x, p(x) \qquad (2.3)$$

which reads "There exists an x in A such that p(x) is a true statement" or, simply, "For some x, p(x)." The symbol

∃

which reads "there exists" or "for some" or "for at least one" is called the existential quantifier. Statement (2.3) is equivalent to the statement

$$T_p = \{x \mid x \in A, p(x)\} \neq \emptyset \qquad (2.4)$$

i.e., that the truth set of p(x) is not empty. Accordingly, ∃x p(x), that is, p(x) preceded by the quantifier ∃, does have a truth value. Specifically:

Q_2: If $\{x \mid p(x)\}$ then ∃x p(x) is true; otherwise, ∃x p(x) is false.

EXAMPLE2.9

(a) The proposition $(\exists n \in N)(n + 4 < 7)$ is true since $\{n \mid n + 4 < 7\} = \{1, 2\} \neq \emptyset$.

(b) The proposition $(\exists n \in N)(n + 6 < 4)$ is false since $\{n \mid n + 6 < 4\} = \emptyset$.

(c) The symbol ∃ can be used to define the union of an indexed collection $\{A_i \mid i \in I\}$ of sets A_i as follows:

$$\cup(A_i \mid i \in I) = \{x \mid \exists\, i \in I, x \mid \in A_i\}$$

2.11 NEGATION OF QUANTIFIED PROPOSITIONS

Consider the statement: "All math majors are male." Its negation reads:

"It is not the case that all math majors are male" or, equivalently,
"There exists at least one math major who is a female (not male)"

Symbolically, using M to denote the set of math majors, the above can be written as

$$\neg(\forall x \in M)(x \text{ is male}) \equiv (\exists\, x \in M)\ (x \text{ is not male})$$

or, when p(x) denotes "x is male,"

$$\neg(\forall x \in M)p(x) \equiv (\exists\, x \in M)\neg p(x) \quad \text{or} \quad \neg\forall x p(x) \equiv \exists x \neg p(x)$$

The above is true for any proposition p(x). That is:

THEOREM 2.4 (DeMorgan): $\neg(\forall x \in A)p(x) \equiv (\exists\, x \in A)\neg p(x)$.

In other words, the following two statements are equivalent:

(1) It is not true that, for all $a \in A$, $p(a)$ is true.
(2) There exists an $a \in A$ such that $p(a)$ is false.

There is an analogous theorem for the negation of a proposition which contains the existential quantifier.

THEOREM 2.5 (DeMorgan): $\neg(\exists x \in A)p(x) \equiv (\forall x \in A)\neg p(x)$.

That is, the following two statements are equivalent:

(1) It is not true that for some $a \in A$, $p(a)$ is true. (2) For all $a \in A$, $p(a)$ is false.

EXAMPLE 2.10

(a) The following statements are negatives of each other:

"For all positive integers n we have $n + 2 > 8$" "There exists a positive integer n such that $n + 2 \not> 8$ "

(b) The following statements are also negatives of each other:

"There exists a (living) person who is 150 years old" "Every living person is not 150 years old"

Remark: The expression $\neg p(x)$ has the obvious meaning:

"The statement $\neg p(a)$ is true when $p(a)$ is false, and vice versa"

Previously, $\neg$ was used as an operation on statements; here $\neg$ is used as an operation on propositional functions. Similarly, $p(x) \wedge q(x)$, read "p(x) and q(x)," is defined by:

"The statement $p(a) \wedge q(a)$ is true when $p(a)$ and $q(a)$ are true"

Similarly, $p(x) \vee q(x)$, read "$p(x)$ or $q(x)$," is defined by:

"The statement $p(a) \vee q(a)$ is true when $p(a)$ or $q(a)$ is true"

Thus in terms of truth sets:

(i) $\neg p(x)$ is the complement of $p(x)$.

(ii) $p(x) \wedge q(x)$ is the intersection of $p(x)$ and $q(x)$.

(iii) $p(x) \vee q(x)$ is the union of $p(x)$ and $q(x)$.

One can also show that the laws for propositions also hold for propositional functions. For example, we have DeMorgan's laws:

$$\neg(p(x) \wedge q(x)) \equiv \neg p(x) \vee \neg q(x) \quad \text{and} \quad \neg(p(x) \vee q(x)) \equiv \neg p(x) \wedge \neg q(x)$$

Counter example

Theorem 2.6 tells us that to show that a statement $\forall x$, $p(x)$ is false, it is equivalent to show that $\exists\, x \neg p(x)$ is true or, in other words, that there is an element x_0 with the property that $p(x_0)$ is false. Such an element x_0 is called a counterexample to the statement $\forall x$, $p(x)$.

EXAMPLE 2.11

(a) Consider the statement $\forall x \in \mathbf{R}$, $|x| \neq 0$. The statement is false since 0 is a counterexample, that is, $|0| \neq 0$ is not true.

(b) Consider the statement $\forall x \in \mathbf{R}$, $x^2 \geq x$. The statement is not true since, for example, ½ is a counterexample. Specifically, $(\frac{1}{2})^2 \geq \frac{1}{2}$ is not true, that is, $(\frac{1}{2})^2 < \frac{1}{2}$.

Consider the statement $\forall x \in \mathbf{N}$, $x^2 \geq x$. This statement is true where **N** is the set of positive integers. In other words, there does not exist a positive integer n for which $n^2 < n$.

Propositional Functions with more than One Variable

A propositional function (of n variables) defined over a product set $A = A_1 \times \cdots \times A_n$ is an expression:

$$p(x_1, x_2, \ldots, x_n)$$

which has the property that $p(a_1, a_2, \ldots, a_n)$ is true or false for any n-tuple $(a_1, \ldots a_n)$ in A. For example,

$$x + 2y + 3z < 18$$

is a propositional function on $\mathbf{N}^3 = \mathbf{N} \times \mathbf{N} \times \mathbf{N}$. Such a propositional function has no truth value. However, we do have the following:

Basic Principle: A propositional function preceded by a quantifier for each variable, for example,

$$\forall x \exists y, p(x, y) \quad \text{or}$$

This statement is true. For example, if x = 1, let y = 9; if x = 2, let y = 8, and so on.

(c) The following is also a statement:

$$\exists x \, \forall y \, \exists z, p(x, y, z)$$

$\exists y \forall x, p(x, y)$, that is, "There exists a y such that, for every x, we have x + y = 10"

No such y exists; hence this statement is false.

Note that the only difference between (a) and (b) is the order of the quantifiers. Thus a different ordering of the quantifiers may yield a different statement. We note that, when translating such quantified statements into English, the expression "such that" frequently follows "there exists."

Negating Quantified Statements with more than One Variable

Quantified statements with more than one variable may be negated by successively applying Theorems 2.5 and 2.6. Thus each ∀ is changed to ∃ and each ∃ is changed to ∀ as the negation symbol ¬ passes through the statement from left to right. For example,

$$\neg[\forall x \exists y \exists z, p(x, y, z)] \equiv \exists x \neg[\exists y \exists z, p(x, y, z)] \equiv \neg \exists z \forall y[\exists z, p(x, y, z)$$

$$\equiv \exists x \forall y \forall z, \neg p(x, y, z)$$

Naturally, we do not put in all the steps when negating such quantified statements.

EXAMPLE 2.13

(a) Consider the quantified statement:

"Every student has at least one course where the lecturer is a teaching assistant."

Its negation is the statement:

"There is a student such that in every course the lecturer is not a teaching assistant."

b) The formal definition that L is the limit of a sequence $a_1, a_2, \ldots$ follows:

$$\forall \in > 0, \exists\, n_0 \in N, \forall n > n_0 \text{ we have } | a_n - L | < \in$$

Thus L is not the limit of the sequence $a_1, a_2, \ldots$ when:

$$\exists \in > 0, \forall n_0 \in N, \exists\, n > n_0 \text{ such that } | a_n - L | \geq \in$$

Solved Problems

PROPOSITIONS AND TRUTH TABLES

2.1 Let p be "It is cold" and let q be "It is raining". Give a simple verbal sentence which describes each of the following statements: (a) ¬p; (b) p ∧ q; (c) p ∨ q; (d) q ∨ ¬p.

In each case, translate ∧, ∨, and ~ to read "and," "or," and "It is false that" or "not," respectively, and then simplify the English sentence.

(a) It is not cold.
(b) It is cold and raining.
(c) It is cold or it is raining.
(d) It is raining or it is not cold.

2. 2. Find the truth table of ¬p ∧ q.

Construct the truth table of ¬p ∧ q as in Fig. 4-9(a).

p	q	$\neg p$	$\neg p \wedge q$
T	T	F	F
T	F	F	F
F	T	T	T
F	F	T	F

$(a) \neg p \wedge q$

p	q	$p \wedge q$	$\neg (p \wedge q)$	$p \vee \neg (p \wedge q)$
T	T	T	F	T
T	F	F	T	T
F	T	F	T	T
F	F	F	T	T

$(b) p \vee \neg (p \wedge q)$

Fig. 2.9

2.3. Verify that the proposition p ∨ ¬(p ∧ q) is a tautology.

Construct the truth table of p ∨ ¬(p ∧ q) as shown in Fig. 2-9(b). Since the truth value of p ∨ ¬(p ∧ q) is T for all values of p and q, the proposition is a tautology.

2.4. Show that the propositions ¬(p ∧ q) and ¬p ∨ ¬q are logically equivalent.

Construct the truth tables for ¬(p ∧ q) and ¬p ∨ ¬q as in Fig. 2-10. Since the truth tables are the same (both propositions are false in the first case and true in the other three cases), the propositions ¬(p ∧ q) and ¬p ∨ ¬q are logically equivalent and we can write

$$\neg(p \wedge q) \equiv \neg p$$

p	q	$p \wedge q$	$\neg(p \wedge q)$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

(a) $\neg(p \wedge q)$

p	q	$\neg p$	$\neg q$	$\neg p \vee \neg q$
T	T	F	F	F
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

(b) $\neg p \vee \neg q$

Fig. 2-10

5. Use the laws in Table 4-1 to show that ¬(p ∧ q) ∨ (¬p ∧ q) ≡ ¬p.

	Statement	**Reason**
(1)	¬(p ∨ q) ∨ (¬p ∧ q) ≡ (¬p ∧ ¬q) ∨ (¬p ∧ q)	DeMorgan's law
(2)	≡ ¬p ∧ (¬q ∨ q)	Distributive law
(3)	≡ ¬p ∧ T	Complement law
(4)	≡ ¬p	Identity law

CONDITIONAL STATEMENTS

2.6. Rewrite the following statements without using the conditional:

(a) If it is cold, he wears a hat.
(b) If productivity increases, then wages rise.

Recall that "If p then q" is equivalent to "Not p or q;" that is, p → q ≡ ¬p ∨ q. Hence,

(a) It is not cold or he wears a hat.
(b) Productivity does not increase or wages rise.

2.7. Consider the conditional proposition $p \rightarrow q$. The simple propositions $q \rightarrow p$, $\neg p \rightarrow \neg q$ and $\neg q \rightarrow \neg p$ are called, respectively, the converse, inverse, and contra-positive of the conditional $p \rightarrow q$. Which if any of these propositions are logically equivalent to $p \rightarrow q$?

Construct their truth tables as in Fig. 4-11. Only the contra-positive $\neg q \rightarrow \neg p$ is logically equivalent to the original conditional proposition $p \rightarrow q$.

				Conditional	Converse	Inverse	Contrapositive
p	q	$\neg p$	$\neg q$	$p \rightarrow q$	$q \rightarrow p$	$\neg p \rightarrow \neg q$	$\neg q \rightarrow \neg p$
T	T	F	F	T	T	T	T
T	F	F	T	F	T	T	F
F	T	T	F	T	F	F	T
F	F	T	T	T	T	T	T

Fig. 2-11

2.8. Determine the contra-positive of each statement:

(a) If Erik is a poet, then he is poor.
(b) Only if Marc studies will he pass the test.
(a) The contra-positive of $p \rightarrow q$ is $\neg q \rightarrow \neg p$. Hence the contra-positive follows:

If Erik is not poor, then he is not a poet.

(b) The statement is equivalent to: "If Marc passes the test, then he studied." Thus its contra-positive is: If Marc does not study, then he will not pass the test.

2.9. Write the negation of each statement as simply as possible:

(a) If she works, she will earn money.
(b) He swims if and only if the water is warm.
(c) If it snows, then they do not drive the car.

(a) Note that ¬(p → q) ≡ p ∧ ¬q; hence the negation of the statement is:

She works or she will not earn money.

(b) Note that ¬(p ↔ q) ≡ p ↔ ¬q ≡ ¬p ↔ q ; hence the negation of the statement is either of the following:

He swims if and only if the water is not warm.
He does not swim if and only if the water is warm.

(c) Note that ¬(p → ¬q) ≡ p ∧ ¬¬q ≡ p ∧ q. Hence the negation of the statement is:

It snows and they drive the car.

ARGUMENTS

2.10. Show that the following argument is a fallacy: p → q, ¬ p **S**¬q.

Construct the truth table for [(p → q)∧ ¬p] → ¬q as in Fig. 2-12. Since the proposition [(p → q)∧ ¬p] → ¬q is not a tautology, the argument is a fallacy. Equivalently, the argument is a fallacy since in the third line of the truth table p → q and ¬p are true but ¬q is false.

p	q	$p \to q$	$\neg p$	$(p \to q) \wedge \neg p$	$\neg q$	$[(p \to q) \wedge \neg p] \to \neg p$
T	T	T	F	F	F	T
T	F	F	F	F	T	T
F	T	T	T	T	F	F
F	F	T	T	T	T	T

Fig. 2-12

2.11. Determine the validity of the following argument: p → q, ¬ p $ ¬p.

Construct the truth table for [(p → q)∧ ¬q] → ¬p as in Fig. 2-13. Since the proposition [(p → q)∧ ¬q] → ¬p is a tautology, the argument is valid.

p	q	$[(p$	$\to$	$q)$	$\wedge$	$\neg$	$q]$	$\to$	$\neg$	p
T	T	T	T	T	F	F	T	T	F	T
T	F	T	F	F	F	T	F	T	F	T
F	T	F	T	T	F	F	T	T	T	F
F	F	F	T	F	T	T	F	T	T	F
Step		1	2	1	3	2	1	4	2	1

Fig. 2-13

2.12. Prove the following argument is valid: p → ¬q, r → q, r $ ¬p.

Construct the truth table of the premises and conclusions as in Fig. 2-14(a). Now, p → ¬q, r → q, and r are true simultaneously only in the fifth row of the table, where ¬p is also true. Hence the argument is valid.

	p	q	r	$p \to \neg q$	$r \to q$	$\neg q$
1	**T**	T	T	F	T	F
2	**T**	T	F	F	T	F
3	**T**	F	T	T	F	F
4	**T**	F	F	T	T	F
5	F	T	T	T	T	**T**
6	F	T	F	T	T	**T**
7	F	F	T	T	F	**T**
8	F	F	F	T	T	**T**

(a)

p	q	$\neg q$	$p \to \neg q$	$\neg p$
T	T	F	F	F
T	F	T	T	F
F	T	F	T	T
F	F	T	T	T

(b)

Fig. 2-14

2.13. Determine the validity of the following argument:

If 7 is less than 4, then 7 is not a prime number. 7 is not less than 4.

---

7 is a prime number.

First translate the argument into symbolic form. Let p be "7 is less than 4" and q be "7 is a prime number." Then the argument is of the form

p → ¬q, ¬q $¬ q

Now, we construct a truth table as shown in Fig. 2-14(b). The above argument is shown to be a fallacy since, in the fourth line of the truth table, the premises p → ¬q and ¬p are true, but the conclusion q is false.

Remark: The fact that the conclusion of the argument happens to be a true statement is irrelevant to the fact that the argument presented is a fallacy.

2.14. Test the validity of the following argument:

If two sides of a triangle are equal, then the opposite angles are equal.
Two sides of a triangle are not equal.

The opposite angles are not equal.

First translate the argument into the symbolic form $p \rightarrow q$, $\neg p \$ \neg q$, where p is "Two sides of a triangle are equal" and q is "The opposite angles are equal." By Problem 4.10, this argument is a fallacy.

Remark: Although the conclusion does follow from the second premise and axioms of Euclidean geometry, the above argument does not constitute such a proof since the argument is a fallacy.

QUANTIFIERS AND PROPOSITIONAL FUNCTIONS

2.15. Let A = {1, 2, 3, 4, 5}. Determine the truth value of each of the following statements:

(a) $(\exists x \in A)(x+3 = 10)$

(b) $(\forall x \in A)(x + 3 \leq 7)$

(c) $(\exists x \in A)(x+3 < 5)$

(d) $(\forall x \in A)(x + 3 \leq 7)$

(a) False. For no number in A is a solution to $x + 3 = 10$
(b) True. For every number in A satisfies $x + 3 < 10$
© True. For if $x_0 = 1$, then $x_0 + 3 < 5$, i.e., 1 is a solution
(d) False. For if $x_0 = 5$, then $x_0 + 3$ is not less than or equal 7. In other words, 5 is not a solution to the given condition

2.16. Determine the truth value of each of the following statements where U={ 1,2,3} is the universal set:

(a) $\exists x \forall y, x^2 < y + 1$; (b) $\forall x \exists y, x^2 + y^2 < 12$; (c) $\forall x \forall y, x^2 + y^2 < 12$.

(a) True. For if x =1, then 1,2,3, are all solutions to $1<y+1$.
(b)True. For each x_0, let y = 1; then $x_0^2 + 1 < 12$ is a true statement
(c) False. For if $x_0 = 2$ and $y_0 = 3$, then $x_0^2 + y_0^2 < 12$ is not a true statement

2.17. Negate each of the following statements:

(a) $\exists x\, \forall y,\ p(x, y)$; (b) $\exists x\, \forall y,\ p(x, y)$; (c) $\exists y\, \exists x\, \forall z,\ p(x, y, z)$.

Use $\neg\forall x\ p(x) \equiv \exists x \neg p(x)$ and $\neg\exists x\ p(x) \equiv \forall x \neg p(x)$:

(a) $\neg(\exists x \forall y,\ p(x, y)) \equiv \forall x \exists y \neg p(x, y)$

(b) $\neg(\forall x \forall y,\ p(x, y)) \equiv \exists x \exists y \neg p(x, y)$

(c) $\neg(\exists y\ \exists x\ \forall z,\ p(x, y, z)) \equiv \forall y\ \forall x\ \exists z \neg p(x, y, z)$

2.18. Let p(x) denote the sentence "x + 2 > 5." State whether or not p(x) is a propositional function on each of the following sets: (a) **N**, the set of positive integers; (b) M = {−1, −2, −3, . . .}; (c) C, the set of complex numbers.

(a) Yes.

(b) Although p(x) is false for every element in M, p(x) is still a propositional function on M.

(c) No. Note that 2i + 2 > 5 does not have any meaning. In other words, inequalities are not defined for complex numbers.

2.19 2egate each of the following statements: (a) All students live in the dormitories. (b) All mathematics majors are males. (c) Some students are 25 years old or older.

Use Theorem 4.4 to negate the quantifiers.

(a) At least one student does not live in the dormitories. (Some students do not live in the dormitories.)

(b) At least one mathematics major is female. (Some mathematics majors are female.)

(c) None of the students is 25 years old or older. (All the students are under 25.)

PROPOSITIONS AND TRUTH TABLES

2.20 Let p denote "He is rich" and let q denote "He is happy." Write each statement in symbolic form using p and q. Note that "He is poor" and "He is unhappy" are equivalent to ¬p and ¬q, respectively.

(a) If he is rich, then he is unhappy. (c) It is necessary to be poor in order to be happy.
(b) He is neither rich nor happy. (d) To be poor is to be unhappy.

2.21 Find the truth tables for. (a) $p \vee \neg q$; (b) $\neg p \wedge \neg q$.

2.22 Verify that the proposition $(p \wedge q) \wedge \neg(p \vee q)$ is a contradiction.

ARGUMENTS

2.23. Test the validity of each argument:

(a) If it rains, Erik will be sick.
It did not rain.

Erik was not sick.

(b) If it rains, Erik will be sick.
Erik was not sick.

It did not rain.

2.24 Test the validity of the following argument:

If I study, then I will not fail mathematics. If I do not play basketball, then I will study. But I failed mathematics.

Therefore I must have played basketball.

QUANTIFIERS

2.25. Let A = {1, 2, . . . , 9, 10}. Consider each of the following sentences. If it is a statement, then determine its truth value. If it is a propositional function, determine its truth set.

(a) $(\forall x \in A)(\exists y \in A)(x + y < 14)$ (c) $(\forall x \in A)(\forall y \in A)(x + y < 14)$

(b) $(\forall y \in A)(x + y < 14)$ (d) $(\exists y \in A)(x + y < 14)$

2.26. Negate each of the following statements:

(a) If the teacher is absent, then some students do not complete their homework.
(b) All the students completed their homework and the teacher is present.
(c) Some of the students did not complete their homework or the teacher is absent.

2.27. Negate each statement in Problem 2-15.

2.28. Find a counterexample for each statement were **U** = {3, 5, 7, 9} is the universal set:

(a) $\forall x, x + 3 \geq 7$, (b) $\forall x$, x is odd, (c) $\forall x$, x is prime, (d) $\forall x, |x| = x$

CHAPTER 3

COUNTING TECHNIQUES

3.1 INTRODUCTION

A factorial, symbolized by *n!* with *n* being a number between 1 and infinite, is a product of consecutive counting numbers starting at 1 and ending at *n*. The number 0 is an exception. 0! has a value of 1. Factorials are used in many ways, from their use in equations to solving word problems. Their most useful application is through the calculation of the number of outcomes of an event. For example, it can produce the possible outcomes of the race, and how many combinations of first, second, third, and so forth of the racers.

n Factorial

For any whole number *n*, it's factorial is the product of the natural numbers that are less than or equal to it. For example:

$$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$$

The algebra behind factorials looks like this:

$$n! = n \times (n\text{-}1) \times (n\text{-}2) \times (n\text{-}3) \times \ldots \times 1$$

PERMUTATIONS

Permutations is all the possible outcomes of a given value. For example, if you were given three colors (red [r], green [g], and yellow [y]) and were asked to find possible arrangements of them, you would find 6 possible arrangements:

rgy, ryg, gyr, gry, yrg, ygr.

In a permutation, the order of the objects in the group is important. Using the same example above, we are given three different colors (red [r], green [g], and

yellow [y]). If we were just given red [r], then there would be only one way to arrange it: red. However, when green [g] is thrown in the mix, there are now two ways to arrange these two colors: **rg**or **gr**. And now, when a third color is thrown in (yellow [y]), then there are now six possible outcomes, as stated above. The amount of permutations of the 3 colors is equal to 3 x 2 x 1, which equals 6. This is also equal to the factorial of 3:

$$3! = 3 \times 2 \times 1 = 6$$

The general formula for a permutation of n items is:

$$n \times (n - 1) \times (n - 2) \times (n - 3) \times \ldots \times 1$$

As mentioned above, this is also called the *n factorial*, and is written like this: $n!$

This chapter develops some techniques for determining, without direct enumeration, the number of possible outcomes of a particular event or the number of elements in a set. Such sophisticated counting is sometimes called *combinatorial analysis*. It includes the study of permutations and combinations.

3.2 BASIC PRINCIPLES

There are two basic counting principles used throughout this chapter. The first one involves addition and the second one multiplication.

Sum Rule Principle:

Suppose some event E can occur in m ways and a second event F can occur in n ways, and suppose both events cannot occur simultaneously. Then E or F can occur in $m + n$ ways.

Product Rule Principle:

Suppose there is an event E which can occur in m ways and, independent of this event, there is a second event F which can occur in n ways. Then combinations of E and F can occur in mn ways.

The above principles can be extended to three or more events. That is, suppose an event E_1 can occur in n_1 ways, a second event E_2 can occur in n_2 ways, and, following E_2; a third event E_3 can occur in n_3 ways, and so on. Then:

Sum Rule: If no two events can occur at the same time, then one of the events can occur in:

$$n_1 + n_2 + n_3 + \cdots \text{ways.}$$

Product Rule: If the events occur one after the other, then all the events can occur in the order indicated in

$$n_1 \cdot n_2 \cdot n_3 \cdot \ldots \text{ ways.}$$

EXAMPLE 3.1 Suppose a college has 3 different history courses, 4 different literature courses, and 2 different sociology courses.

(a) The number m of ways a student can choose one of each kind of courses is:

$$m = 3(4)(2) = 24$$

(b) The number n of ways a student can choose just one of the courses is:

$$n = 3 + 4 + 2 = 9$$

There is a set theoretical interpretation of the above two principles. Specifically, suppose $n(A)$ denotes the number of elements in a set A. Then:

(1) ***Sum Rule Principle:*** Suppose A and B are disjoint sets. Then

$$n(A \cup B) = n(A) + n(B)$$

(2) ***Product Rule Principle:*** Let $A \times B$ be the Cartesian product of sets A and B.

Then

$$n(A \times B) = n(A) \cdot n(B)$$

3.3 MATHEMATICAL FUNCTIONS

We discuss two important mathematical functions frequently used in combinatorics.

Factorial Function

The product of the positive integers from 1 to n inclusive is denoted by $n!$, read "n factorial." Namely:

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-2)(n-1)n = n(n-1)(n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

Accordingly, $1! = 1$ and $n! = n(n - l\ )!$. It is also convenient to define $0! = 1$.

EXAMPLE 3.2

(a) $3! = 3 \cdot 2 \cdot 1 = 6$, $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$, $5 = 5 \cdot 4! = 5(24) = 120$.

(b) For large n, one uses Stirling's approximation (where $e = 2.7128...$):
$n! = \sqrt{2\pi n} n^n e^{-n}$

PERMUTATIONS

Any arrangement of a set of n objects in a given order is called a *permutation* of the object (taken all at a time). Any arrangement of any $r \leq n$ of these objects in a given order is called an "r-permutation" or "a permutation of the n objects taken r at a time." Consider, for example, the set of letters A, B, C, D. Then:

(i) $BDCA$, $DCBA$, and $ACDB$ are permutations of the four letters (taken all at a time).
(ii) BAD, ACB, DBC are permutations of the four letters taken three at a time.
(iii) AD, BC, CA are permutations of the four letters taken two at a time.

We usually are interested in the number of such permutations without listing them. The number of permutations of n objects taken r at a time will be denoted by

$$P(n, r)$$ (other texts may use ${}_nP_r$, $P_{n,r}$, or $(n)_r$).

The following theorem applies.

THEOREM 3.1: $P(n, r) = n(n-1)(n-2)\cdots(n-r+1) = \frac{n!}{(n-r)!}$

We emphasize that there are r factors in $n(n-1)(n-2)\cdots(n-r+1)$.

EXAMPLE 3.3. Find the number m of permutations of six objects, say, A, B, C, D, E, F, taken three at a time. In other words, find the number of "three-letter words" using only the given six letters without repetition.

Let us represent the general three-letter word by the following three positions:

$$\_\_, \_\_, \_\_$$

The first letter can be chosen in 6 ways; following this the second letter can be chosen in 5 ways; and, finally, the third letter can be chosen in 4 ways. Write each number in its appropriate position as follows:

$$6 \ , \ 5 \ , \ 4$$

By the Product Rule there are $m = 6 \cdot 5 \cdot 4 = 120$ possible three-letter words without repetition from the six letters. Namely, there are 120 permutations of 6 objects taken 3 at a time. This agrees with the formula in Theorem 3.1:

$$P(6, 3) = 6 \cdot 5 \cdot 4 = 120$$

In fact, Theorem .3.1. is proven in the same way as we did for this particular case.

Consider now the special case of P (n, r) when r = n. We get the following result.

Corollary 3.1: There are $n!$ permutations of n objects (taken all at a time).

For example, there are 3! = 6 permutations of the three letters A, B, C. These are:

$$ABC, \quad ACB, \quad BAC, \quad BCA, \quad CAB, \quad CBA.$$

3.3 PERMUTATIONS WITH REPETITIONS

Frequently we want to know the number of permutations of a multiset, that is, a set of objects some of which are alike. We will let

$$P(n; n_1, n_2, \ldots, n_r)$$

denote the number of permutations of n objects of which n_1 are alike, n_2 are alike, . . ., n_r are alike. The general formula follows:

THEOREM 3.2. $P(n; n_1, n_2, \ldots, n_r) = \frac{n!}{n_1!n_2!n_3!.....n_r!}$

We indicate the proof of the above theorem by a particular example. Suppose we want to form all possible five-letter "words" using the letters from the word "*BABBY*." Now there are 5! = 120 permutations of the objects B_1, A, B_2, B_3, Y, where the three B's are distinguished. Observe that the following six permutations

$$B_1B_2\,B_3AY\,,\quad B_2B_1\,B_3AY\,,\quad B_3B_1\,B_2AY\,,\quad B_1B_3\,B_2AY\,,\quad B_2B_3\,B_1AY\,,\quad B_3B_2\,B_1AY$$

produce the same word when the subscripts are removed. The 6 comes from the fact that there are 3! = 3 ·2 ·1 = 6 different ways of placing the three B's in the first three positions in the permutation. This is true for each set of three positions in which the B's can appear. Accordingly, the number of different five-letter words that can be formed using the letters from the word "*BABBY*" is:

$$5!$$

$$P(5; 3) = 3! = 20$$

EXAMPLE 3.4 Find the number m of seven-letter words that can be formed using the letters of the word "*BENZENE*."

We seek the number of permutations of 7 objects of which 3 are alike (the three E's), and 2 are alike (the two N's). By Theorem 3.6,

$$M = P(7,3,2) = \frac{7!}{3!2!} = \frac{7.6.5.4.3.2.1.}{3.2.1.2.1} = 420$$

Ordered Samples

Many problems are concerned with choosing an element from a set *S*, say, with *n* elements. When we choose one element after another, say, *r* times, we call the choice an *ordered sample* of size *r*. We consider two cases.

(1) Sampling with replacement

Here the element is replaced in the set *S* before the next element is chosen. Thus, each time there are *n* ways to choose an element (repetitions are allowed). The Product rule tells us that the number of such samples is:

$$n \cdot n \cdot n \cdots n \cdot n(\text{r factors}) = n$$

(2) Sampling without replacement

Here the element is not replaced in the set *S* before the next element is chosen. Thus, there is no repetition in the ordered sample. Such a sample is simply an *r*-permutation. Thus the number of such samples is:

$$P(n, r) = n(n-1)(n-2)\cdots(n-r+1) = \frac{n!}{(n-r)!}$$

3.5 COMBINATIONS WITH REPETITIONS

Consider the following problem. A bakery makes only M = 4 kinds of cookies: apple (a), banana (b), carrot (c), dates (d). Find the number of ways a person can buy r = 8 of the cookies.

Observe that order does not count. This is an example of combinations with repetitions. In particular, each combination can be listed with the a's first, then the b's, then the c's, and finally the d's. Four such combinations follow:

r_1 = aa, bb, cc, dd; r_2 = aaa, c, ddd; r_3 = bbbb, c, ddd; r_4 = aaaaa, ddd.

Counting the number m of such combinations may not be easy.

Suppose we want to code the above combinations using only two symbols, say 0

and 1. This can be done by letting 0 denote a cookie, and letting 1 denote a change from one kind of cookie to another. Then each combination will require r = 8 zeros, one for each cookie, and M − 1 = 3 ones, where the first one denotes the change from a to b, the second one from b to c, and the third one from c to d. Thus the above four combinations will be coded as follows:

$$r_1 = 00100100100, \quad r_2 = 00001101000, \quad r_3 = 10000101000, \quad r_4 = 00000111000.$$

Counting the number m of these "codewords" is easy. Each codeword contains R + M − 1 = 11 digits where r = 8 are 0's and hence M − 1 = 3 are 1's. Accordingly,

$$M = C(11, 8) = C(11, 3) = \frac{11.10.6}{3.2.1} = 165$$

A similar argument gives us the following theorem.

THEOREM 3.3: Suppose there are M kinds of objects. Then the number of combinations of r such objects is

$$C(r + M - 1, r) = C(r + M - 1, M - 1).$$

EXAMPLE 3.5. Find the number m of nonnegative integer solutions of x + y + z = 18.

We can view each solution, say x = 3, y = 7, z = 8, as a combination of r = 18 objects consisting of 3 a's, 7 b's, and 8 c's, where there are M = 3 kinds of objects, a's, b's, and c's. By Theorem 3.1,

$$m = C(r + M - 1, M - 1) = C(20, 2) = 190.$$

3.6 ORDERED AND UNORDERED PARTITIONS

Suppose a set has 7 elements. We want to find the number m of ordered partitions of S into three cells, say $[A_1, A_2, A_3]$, so they contain 2, 3, and 2 elements, respectively.

Since S has 7 elements, there are $C(7, 2)$ ways of choosing the first two elements for A_1. Following this, there are $C(5, 3)$ ways of choosing the 3 elements for A_2.

Lastly, there are $C(2, 2)$ ways of choosing the 2 elements for A_3 (or, the last 2 elements form the cell A_3). Thus:

m = C(7,2)C(5,3)C(2,2)

$= \binom{7}{2}\binom{5}{3}\binom{2}{2} = \frac{7.6}{2.1}\frac{5.4.3}{3.2.1}\frac{2.1}{.2.1.} = 210$

Observe that m= $\binom{7}{2}\binom{5}{3}\binom{2}{2}$ =

$=\frac{7!}{2!5!}\frac{5!}{3!2!}\frac{2!}{2!0!} = \frac{7!}{2!3!2!}$

since each numerator after the first is cancelled by a term in the denominator of the previous factor. The above discussion can be shown to be true in general. Namely:

THEOREM 3.4: The number m of ordered partitions of a set S with n elements into r cells $[A_1, A_2, \ldots, A_r]$ where, for each i, $n(A_i) = n_i$, follows:

$$\text{m} = \frac{n!}{n_1!n_2!n_3!\ldots\ldots n_r!}$$

Unordered Partitions

Frequently, we want to partition a set S into cells $[A_1, A_2, \ldots, A_r]$ where the cells are now unordered. The number m of such unordered partitions is obtained from the number m of ordered partitions by dividing m by each $k!$ where k of the cells have the same number of elements.

EXAMPLE 3.6. Find the number m of ways to partition 10 students into four teams so that two teams contain 3 students and two teams contain 2 students.

By Theorem 3.2, there are $m = 10!/(3!3!2!2!) = 25\,200$ such ordered partitions.

Since the teams form an unordered partition, we divide m by 2! because of the two cells with 3 elements each and 2! because of the two cells with 2 elements each.

Thus $m = 25\,200/(2!2!) = 6300$.

3.7 INCLUSION–EXCLUSION PRINCIPLE REVISITED

Let $A_1, A_2, \ldots, A_r$ be subsets of a universal set U. Suppose we let s_k denote the sum of the cardinalities of all possible k-tuple intersections of the sets, that is, the sum of all of the cardinalities

$$n(A_{i1} \cap A_{i2} \cap \cdots \cap A_{ik})$$

For example, $s_1 = \sum_i n(A_i), s_2 = \sum_{i<j} n(A_i \cap A_j), s_3 = \sum_{i_1<,i_2<i_3} n(A_{i1} \cap A_{i2} \cap A_{i3})$

The Inclusion–Exclusion Principle, which appears in Section 3.7, gave a formula for the number of elements in the union of the sets. Specifically, (Theorem 3.9) we have

$$n(A_1 \cup A_2 \cup \cdots \cup A_r) = s_1 - s_2 + s_3 - \cdots + (-1)^{r-1} s_r$$

On the other hand, using DeMorgan's law,

$$n(A^C_1 \cap A^C_2 \cap \ldots \cap A^C_r) = n([A_1 \cup A_2 \cup \ldots \cup A_r]^C) = |\mathbf{U}| - n(A_1 \cup A_2 \cup \ldots \cup A_r)$$

Accordingly, we obtain an alternate form for Theorem 3.9:

THEOREM (Inclusion–Exclusion Principle) 3.5: Let $A_1, A_2, \ldots, A_r$ be subsets of a universal set **U**. Then the number m of elements which do not appear in any of the subsets $A_1, A_2, \ldots, A_r$ of **U** is:

$$m = n(A^C_1 \cap A^C_2 \cap \ldots \cap A^C_r) = |\mathbf{U}| - s_1 + s_2 - s_3 + \cdots + (-1)^r s_r$$

EXAMPLE 3.7 Let **U** be the set of positive integers not exceeding 1000. Then |U | = 1000. Find |S| where S is the set of such integers which are not divisible by 3, 5, or 7.

Let A be the subset of integers which are divisible by 3, B which are divisible by 5, and C which are divisible by 7. Then $S = A^C \cap B^C \cap C^C$ since each element of S is not divisible by 3, 5 or 7. By integer division,

$$|A| = 1000/3 = 33, \quad |B| = 1000/5 = 200, \quad |C| = 1000/7 = 142,$$
$$|A \cap B| = 1000/15 = 66, \quad |A \cap C| = 1000/21 = 47, \quad |B \cap C| = 1000/35 = 28,$$
$$|A \cap B \cap C| = 1000/105 = 9$$

Thus, by the Inclusion–Exclusion Principle Theorem 3.3,

$$|S| = 1000 - (333 + 200 + 142) + (66 + 47 + 28) - 9 = 1000 - 675 + 141 - 9 = 457$$

Number of Onto Functions

Let A and B be sets such that $|A| = 6$ and $|B| = 4$. We want to find the number of surjective (onto) functions from A onto B.

Let b_1, b_2, b_3, b_4 be the four elements in B. Let **U** be the set of all functions from A into B. Furthermore, let F_1 be the set of functions which do not send any element of A into b_1 that is, b_1 is not in the range of any function in F_1. Similarly, let F_2, F_3, and F_4 be the sets of functions which do not send any element of A into b_2, b_3, and b_4, respectively.

We are looking for the number of functions in $S = F_1^C \cap F_2^C \cap F_3^C \cap F_4^C$, that is, those functions which do send at least one element of A into b_1, at least one element of A into b_2, and so on. We will use the Inclusion–Exclusion Principle Theorem 6.3 as follows.

(i) For each function in **U**, there are 4 choices for each of the 6 elements in A; hence $|U| = 4^6 = 4096$

(ii) There are $C(4, 1) = 4$ functions F_i . In each case, there are 3 choices for each of the 6 elements in A, hence $|F_i| = 3^6 = 729$.

(iii) There are $C(4, 2) = 6$ pairs $F_i \cap F_j$. In each case, there are 2 choices for each of the 6 elements in A, hence $|F_i \cap F_j| = 2^6 = 64$.

(iv) There are $C(4, 3) = 4$ triplets $F_i \cap F_j \cap F_k$. In each case, there is only one choice for each of the 6 elements in A. Hence $|F_i \cap F_j \cap F_k| = 1^6 = 1$.

(v) $F_1 \cap F_2 \cap F_3 \cap F_4$ has no element, that is, is empty. Hence $|F_1 \cap F_2 \cap F_3 \cap F_4| = 0$. By the Inclusion– Exclusion Principle Theorem 6.3,

$$|S| = |F_1^C \cap F_2^C \cap F_3^C \cap F_4^C| = 4^6 - C(4, 1)3^6 + C(4, 2)2^6 - C(4, 3)1^7$$

$$= 4096 - 2916 + 384 - 1 = 795$$

The above result is true in general. Namely:

THEOREM 3.6: Suppose $|A| = m$ and $|B| = n$ where $m \geq n$. Then the number N of surjective (onto) functions from A onto B is:

$$N = n^m - C(n, 1)(n - 1)^m + C(n, 2)(n - 2)^m - \cdots + (-1)^{n-1}C(n, n - 1)1^m$$

Derangements

A derangement is a permutation of objects where each object is not in its original position. For example, 453162 is not a derangement of 123456 since 3 is in its correct position, but 264531 is a derangement of 123456. (Alternately, a permutation $\sigma: X \to X$ is a derangement if $\sigma(i) \_= i$ for every $i \in X = \{1, 2, \ldots, n\}$.)

Let D_n denote the number of derangements of n objects. For example, 231 and 312 are the only derangements of 123. Hence $D_3 = 2$. The following theorem, proved in Problem 3.6, applies.

THEOREM 3.5: $D_n = n![1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}]$

The probability that a derangement of n objects occurs equals D_n divided by n!, the number of permutations of the n objects. Thus Theorem 3.5 yields:

Corollary 3.2: Let p be the probability of a derangement of n objects. Then

$$P = 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}$$

EXAMPLE 3.8. (Hat Check Problem) Suppose n = 5 people check in their hats at a restaurant and they are given back their hats at random. Find the probability p that no person receives his/her own hat.

This is an example of a derangement with n = 5. By Corollary 3.2,

$$p = 1 - 1 + 1/2 - 1/6 + 1/24 - 1/120 = 44/120 = 11/30 \approx 0.367$$

Note that the signs alternate and the terms get very, very small in Corollary 6.6. Figure 6-1 gives the values of p for the first few values of n. Note that, for n > 4 , p is very close to the following value (where e = 2.718):

$$e^{-1} = 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!} + \cdots \approx 0.368$$

n	1	2	3	4	5	6	7
$p = D_n/n!$	0.0000	0.5000	0.3333	0.3750	0.3667	0.3681	0.3679

Fig. 3-1

3.8 PIGEONHOLE PRINCIPLE REVISITED

The Pigeonhole Principle (with its generalization) is stated with simple examples in previous Section. Here we give examples of more sophisticated applications of this principle.

EXAMPLE 3.9 Consider six people, where any two of them are either friends or strangers. Show that there are three of them which are either mutual friends or mutual strangers.

Let A be one of the people. Let X consist of those which are friends of A, and Y consist of those which are strangers of A. By the Pigeonhole Principle, either X or Y has at least three people. Suppose X has three people. If two of them are friends, then the two with A are three mutual friends. If not, then X has three mutual strangers. Alternately, suppose Y has three people. If two of them are strangers, then the two with A are three mutual strangers. If not, then X has three mutual friends.

EXAMPLE 3.10 Consider five lattice points $(x_1, y_1), \ldots, (x_5, y_5)$ in the plane, that is, points with integer coor-dinates. Show that the midpoint of one pair of the points is also a lattice point.

The midpoint of points P(a, b) and Q(c, d) is ([a + c]/2, [b + d]/2). Note that

(r + s)/2 is an integer if r and s are integers with the same parity, that is, both are odd or both are even. There are four pairs of parities: (odd, odd), (odd, even), (even, odd), and (even, even). There are five points. By the Pigeonhole Principle, two of the points have the same pair of parities. The midpoint of these two points has integer coordinates.

An important application of the Pigeonhole Principle follows.

THEOREM 3.7: Every sequence of distinct $n^2 + 1$ real numbers contains a subsequence of length n + 1 which is strictly increasing or strictly decreasing.

For example, consider the following sequence of $10 = 3^2 + 1$ numbers (where n = 3): 2, 1, 8, 6, 7, 5, 9, 4, 12, 3. There are many subsequences of length n + 1 = 4 which are strictly increasing or strictly decreasing; for example,

2, 6, 9, 12; 1, 5, 9, 12; 8, 6, 5, 4; 7, 5, 4, 3.

On the other hand, the following sequence of $9 = 3^2$ numbers has no subsequence of length n + 1 = 4 which is strictly increasing or strictly decreasing:

3, 2, 1, 6, 5, 4, 9, 8, 7.

The proof of Theorem 3.7 appears in Problem 3.10.

3.9 RECURRENCE RELATIONS

Previously, we discussed recursively defined functions such as

(a) Factorial function, (b) Fibonacci sequence, (c) Ackermann function.

Here we discuss certain kinds of recursively defined sequences $\{a_n\}$ and their solution. We note that a sequence is simply a function whose domain is

$$\mathbf{N} = \{1, 2, 3, \ldots\} \quad \text{or} \quad \mathbf{N}_0 = \mathbf{N} \cup \{0\} = \{0, 1, 2, 3, \ldots]$$

We begin with some examples.

EXAMPLE 3.10. Consider the following sequence which begins with the number 3 and for which each of the following terms is found by multiplying the previous term by 2:

$$3, \quad 6, \quad 12, \quad 24, \quad 48, \quad \ldots$$

It can be defined recursively by:

$$a_0 = 3, \quad a_k = 2a_{k-1} \text{ for } k \geq 1 \quad \text{or} \quad a_0 = 3, \quad a_{k+1} = 2a_k \quad \text{for } k \geq 0$$

The second definition may be obtained from the first by setting $k = k + 1$. Clearly, the formula $a_n = 3(2^n)$ gives us the nth term of the sequence without calculating any previous term.

The following remarks about the above example are in order.

(1) The equation $a_k = 2a_{k-1}$ or, equivalently, $a_{k+1} = 2a_k$, where one term of the sequence is defined in terms of previous terms of the sequence, is called a recurrence relation.

(2) The equation $a_0 = 3$, which gives a specific value to one of the terms, is called an initial condition.

(3) The function $a_n = 3(2^n)$, which gives a formula for a_n as a function of n, not of previous terms, is called a solution of the recurrence relation.

(4) There may be many sequences which satisfy a given recurrence relation. For example, each of the following is a solution of the recurrence relation $a_k = 2a_{k-1}$.

$$1, 2, 4, 8, 16, \ldots \quad \text{and} \quad 7, 14, 28, 56, 112, \ldots$$

All such solutions form the so-called general solution of the recurrence relation.

(5) On the other hand, there may be only a unique solution to a recurrence relation which also satisfies given initial conditions. For example, the initial condition $a_0 = 3$ uniquely yields the solution 3, 6, 12, 24, . . . of the recurrence relation $a_k = 2a_{k-1}$.

This chapter shows how to solve certain recurrence relations. First we give two important sequences the reader may have previously studied.

EXAMPLE 3.11

(a) Arithmetic Progression

An arithmetic progression is a sequence of the form

$$a, a + d, a + 2d, a + 3d, \ldots$$

That is, the sequence begins with the number a and each successive term is obtained from the previous term by adding d (the common difference between any two terms). For example:

(i) a = 5, d = 3: 5, 8, 9, 11, . . .

(ii) a = 2, d = 5: 2, 7, 12, 17, . . .

(iii) a = 1, d = 0: 1, 1, 1, 1, 1, . . .

We note that the general arithmetic progression may be defined recursively by:

$a_1 = a$ and $a_{k+1} = a_k + d$ for $k \geq 1$ where the solution is $a_n = a + (n - 1)d$.

(b) Geometric Progression

A geometric progression is a sequence of the form

$$a, ar, ar^2, ar^3, \ldots$$

That is, the sequence begins with the number a and each successive term is obtained from the previous term by multiplying by r (the common ratio between any two terms) for example:

(i) a = 1, r = 3: 1, 3, 9, 27, 81, . . .

(ii) a = 5, r = 2: 5, 10, 20, 40, . . .

(iii) a = 1, r = $\frac{1}{2}$: 1, $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, . . .

We note that the general geometric progression may be defined recursively by:

$$a_1 = a \quad \text{and} \quad a_{k+1} = ra_k \quad \text{for } k \geq 1$$

where the solution is $a_{n+1} = ar^n$.

3.10 LINEAR RECURRENCE RELATIONS WITH CONSTANT COEFFICIENTS

A recurrence relation of order k is a function of the form

$$a_n = \phi(a_{n-1}, a_{n-2}, \ldots, a_{n-k}, n)$$

that is, where the nth term a_n of a sequence is a function of the preceding k terms $a_{n-1}, a_{n-2}, \ldots, a_{n-k}$ (and possibly n). In particular, a linear kth-order recurrence relation with constant coefficients is a recurrence relation of the form

$$a_n = C_1a_{n-1} + C_2a_{n-2} + \cdots + C_k a_{n-k} + f(n)$$

where $C_1, C_2, \ldots, C_k$ are constants with $C_k \neq 0$, and f (n) is a function of n. The meanings of the names linear and constant coefficients follow:

Linear: There are no powers or products of the a_j 's.

Constant coefficients: The $C_1C_2, \ldots, C_k$ are constants (do not depend on n). If f (n) = 0, then the relation is also said to be homogeneous.

Clearly, we can uniquely solve for a_n if we know the values of $a_{n-1}, a_{n-2}, \ldots, a_{n-k}$. Accordingly, by mathematical induction, there is a unique sequence satisfying the recurrence relation if we are given initial values for the first k elements of the sequence.

EXAMPLE 3.12 Consider each of the following recurrence relations.

(a) $a_n = 5a_{n-1} - 4a_{n-2} + n^2$

This is a second-order recurrence relation with constant coefficients. It is non-homogeneous because of the n^2. Suppose we are given the initial conditions

$a_1 = 1$, $a_2 = 2$. Then we can find sequentially the next few elements of the sequence:

$$a_3 = 5(2) - 4(1) + 3^2 = 15, \quad a_4 = 5(15) - 4(2) + 4^2 = 83$$

(b) $a_n = 2a_{n-1}a_{n-2} + n^2$

The product $a_{n-1}a_{n-2}$ means the recurrence relation is not linear. Given initial conditions $a_1 = 1$, $a_2 = 2$, we can still find the next few elements of the sequence:

$$a_3 = 2(2)(1) + 3^2 = 13, \quad a_4 = 2(13)(2) + 4^2 = 68$$

(c) $a_n = na_{n-1} + 3a_{n-2}$

This is a homogeneous linear second-order recurrence relation but it does not have constant coefficients because the coefficient of a_{n-1} is n, not a constant. Given initial conditions $a_1 = 1$, $a_2 = 2$, the next few elements of the sequence follow:

$$a_3 = 3(2) + 3(1) = 9, \quad a_4 = 4(9) + 3(2) = 42$$

(d) $a_n = 2a_{n-1} + 5a_{n-2} - 6a_{n-3}$

This is a homogeneous linear third-order recurrence relation with constant coefficients. Thus we need three, not two, initial conditions to yield a unique solution of the recurrence relation. Suppose we are given the initial conditions $a_1 = 1$, $a_2 = 2$, $a_3 = 1$. Then, the next few elements of the sequence follow:

$$a_4 = 2(1) + 5(2) - 6(1) = 6, \; a_5 = 2(2) + 5(1) - 6(6) = -37$$
$$a_6 = 2(1) + 5(6) - 6(-37) = 254$$

This chapter will investigate the solutions of homogeneous linear recurrence relations with constant coeffi-cients. The theory of non-homogeneous recurrence relations and recurrence relations without constant coefficients lies beyond the scope of this text.

For computational convenience, most of our sequences will begin with a rather than a. The theory is not affected at all.

3.11 SOLVING SECOND-ORDER HOMOGENEOUS LINEAR RECURRENCE RELATIONS

Consider a homogeneous second-order recurrence relation with constant coefficients which has the form

$$a_n = sa_{n-1} + ta_{n-2} \quad \text{or} \quad a_n - sa_{n-1} - ta_{n-2} = 0$$

where s and t are constants with $t \neq 0$. We associate the following quadratic polynomial with the above recurrence relation:

$$\Delta(x) = x^2 - sx - t$$

This polynomial $\Delta(x)$ is called the characteristic polynomial of the recurrence relation, and the roots of $\Delta(x)$ are called its characteristic roots.

THEOREM 3.8: Suppose the characteristic polynomial $\Delta(x) = x^2 - sx - t$ of the recurrence relation

$$a_n = sa_{n-1} + ta_{n-2}$$

has distinct roots r_1 and r_2. Then the general solution of the recurrence relation follows, where c_1 and c_2 are arbitrary constants:

$$a_n = c_1 r_1^n + c_2 r_2^n$$

We emphasize that the constants c_1 and c_2 may be uniquely computed using initial conditions. We note that the theorem is true even when the roots are not real. Such cases lie beyond the scope of this text.

EXAMPLE 3.10 Consider the following homogeneous recurrence relation:

$$a_n = 2a_{n-1} + 3a_{n-2}$$

The general solution is obtained by first finding its characteristic polynomial $\Delta(x)$

and its roots r_1 and r_2:

$$\Delta(x) = x^2 - 2x - 3 = (x - 3)(x + 1); \quad \text{roots } r_1 = 3, r_2 = -1$$

Since the roots are distinct, we can use Theorem 3.8 to obtain the general solution:

$$a_n = c_1 3^n + c_2(-1)^n$$

Thus any values for c_1 and c_2 will give a solution to the recurrence relation.

Suppose we are also given the initial conditions $a_0 = 1$, $a_1 = 2$. Using the recurrence relation we can compute the next few terms of the sequence:

$$1, \quad 2, \quad 8, \quad 28, \quad 100, 356, \quad 1268, \quad 3516, \quad \ldots$$

The unique solution is obtained by finding c_1 and c_2 using the initial conditions. Specifically:

For $n = 0$ and $a_0 = 1$, we get: $c_1 3^0 + c_2(-1)^0 = 1$ or $c_1 + c_2 = 1$
For $n = 1$ and $a_1 = 2$, we get: $c_1 3^1 + c_2(-1)^1 = 2$ or $3c_1 - c_2 = 2$

Solving the system of the two equations in the unknown's c_1 and c_2 yields:

$$c_1 = \frac{3}{4} \quad \text{and} \quad c_2 = \frac{1}{4}$$

Thus the following is the unique solution of the given recurrence relation with the given initial conditions $a_0 = 1$,

EXAMPLE 3.11 Consider the celebrated Fibonacci sequence:

$$a_n = a_{n-1} + a_{n-2}, \quad \text{with} \quad a_0 = 0, a_1 = 1$$

The first 10 terms of the sequence follow:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

Sometimes the Fibonacci sequence is defined using the initial conditions $a_0 = 1$,

$a_1 = 1$ or the initial conditions $a_1 = 1$, $a_2 = 2$. We use $a_0 = 0$, $a_1 = 1$ for computational convenience. (All three initial conditions yield the same sequence after the pair of terms 1, 2.)

Observe that the Fibonacci sequence is a homogeneous linear second-order recurrence relation, so it can be solved using Theorem 3.8. Its characteristic polynomial follows:

$$\Delta(x) = x^2 - x - 1$$

Using the quadratic formula, we obtain the roots: $r_1 = \frac{1 + \sqrt{5}}{2}$, $r_2 = \frac{1-\sqrt{5}}{2}$

By Theorem 3.8, we obtain the general solution: $a_n = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$

The initial conditions yield the following system of two linear equations in c_1 and c_2

For n=0 and $a_0 = 0$, we get: $0 = c_1 + c_2$

For n = 1 and $a_1 = 1$, we get: $1 = c_1\left(\frac{1+\sqrt{5}}{2}\right) + c_2\left(\frac{1-\sqrt{5}}{2}\right)$

The solution of the system follows $c_1 = \frac{1}{\sqrt{5}}, c_2 = -\frac{1}{\sqrt{5}}$

Accordingly, the following is the solution of the Fibonacci recurrence relation:

$$a_n = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n - c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$$

One can show that the absolute value of the above second term for a_n is always less than 1/2. Thus a_n is also the closest integer to the number

$$\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n \approx (0.4472)(1.6180)^n$$

Solution when Roots of the Characteristic Polynomial are Equal

Suppose the roots of the characteristic polynomial are not distinct. Then we have the following result.

THEOREM 3.9: Suppose the characteristic polynomial $\Delta(x) = x^2 - sx - t$ of the recurrence relation

$$a_n = sa_{n-1} + ta_{n-2}$$

has only one root r_0. Then the general solution of the recurrence relation follows, where c_1 and c_2 are arbitrary constants:

$$a_n = c_1 r_0^n + c_2 n r^n_0$$

The constants c_1 and c_2 may be uniquely computed using initial conditions.

EXAMPLE 3.12 Consider the following homogeneous recurrence relation:

$$a_n = 6a_{n-1} - 9a_{n-2}$$

The characteristic polynomial Δ (x) follows:

$$\Delta(x) = x^2 - 6x + 9 = (x - 3)^2$$

Thus Δ (x) has only the one root $r_0 = 3$. Now we use Theorem 3.9 to obtain the following general solution of the recurrence relation:

$$a_n = c_1 3^n + c_2 n 3^n$$

Thus any values for c_1 and c_2 will give a solution to the recurrence relation.

Suppose we are also given the initial conditions $a_1 = 3$, $a_2 = 27$. Using the recurrence relation we can compute the next few terms of the sequence:

$$3, \quad 27, \quad 135, \quad 567, 2187, 8109, \ \ldots$$

The unique solution is obtained by finding c_1 and c_2 using the initial conditions. Specifically:

For n = 1 and $a_1 = 3$, we get: $c_1 3^1 + c_2(1)(3)^1 = 3$ or $3c_1 + 3c_2 = 3$
For n = 2 and $a_2 = 27$, we get: $c_1 3^2 + c_2(2)(3)^2 = 27$ or $9c_1 + 18c_2 = 27$

Solving the system of the two equations in the unknown's c_1 and c_2 yields:

$$c_1 = -1 \quad \text{and } c_2 = 2$$

Thus the following is the unique solution of the recurrence relation with the given initial conditions:

$$a_n = -3^n + 2n3^n = 3^n(2n - 1)$$

EXAMPLE 3.13 Consider the following third-order homogeneous recurrence relation:

$$a_n = 11a_{n-1} - 39a_{n-2} + 45a_{n-3}$$

The characteristic polynomial Δ (x) of the recurrence relation follows:

$$\Delta(x) = x^3 - 11x^2 + 39x - 45 = (x - 3)^2(x - 5)$$

Thus _(x) has two roots, $r_1 = 3$ of multiplicity 2 and $r_2 = 5$ of multiplicity 1. Thus, by the above remarks, the following is the general solution of the recurrence relation:

$$a_n = c_1(3^n) + c_2n(3^n) + c_3(5^n) = (c_1 + c_2n)(3^n) + c_3(5^n)$$

Thus any values for c_1, c_2, c_3 will give a solution to the recurrence relation.

Suppose we are also given the initial conditions $a_0 = 5$, $a_1 = 11$, $a_3 = 25$. Using the recurrence relation we can compute the next few terms of the sequence:

$$5, \quad 11, \quad 25, \quad 71, \quad 301, \quad 1667, \quad \ldots$$

The unique solution is obtained by finding c_1, c_2, c_3 using the initial conditions. Specifically:

For n = 0 and a_0	= 5,	we get:	$c_1 + c_3 = 5$
For n = 1 and a_1	= 11,	we get:	$3c_1 + 3c_2 + 5c_3 = 11$
For n = 2 and a_2	= 25,	we get:	$9c_1 + 18c_2 + 25c_3 = 25$

Solving the system of the three equations in the unknowns c_1, c_2, c_3 yields:

$$c_1 = 4, \quad c_2 = -2, \quad c_3 = 1$$

Thus the following is the unique solution of the recurrence relation with the given initial conditions:

$$a_n = (4 - 2n)(3^n) + 5^n$$

Remark: Finding the roots of the characteristic polynomial _(x) is an important step in solving recurrence relations. Generally speaking, this may be difficult when the degree of _(x) is greater than 2. (Example B.16 indicates one way to find the roots of some polynomials of degree 3 or more.)

SOLVED PROBLEMS

COUNTING TECHNIQUES, INCLUSION–EXCLUSION

3.1 A bagel shop sells M = 5 kinds of bagels. Find the number m of ways a customer can buy: (a) 8 bagels; (b) a dozen bagels.

Use m = C(r + M − 1, r) = C(r + M − 1, M − 1), that is, Theorem 6.1, since this problem concerns combinations with repetitions.

(a) Here r = 8, so m = C(8 + 4, 4) = C(12, 4) = 494.
(b) Here r = 12, so m = C(12 + 4, 4) = C(16, 4) = 1820.

3.2. Find the number m of nonnegative solutions to x + y + z = 18 with the conditions that x ≥ 3, y ≥ 2, z ≥ 1.

Let x = x − 3, y = y − 2 and z = z − 1. Then m is also the number of nonnegative solutions to x + y + z = 12. As in Example 3.1, this second problem concerns combinations with repetitions with M = 3 and r = 12. Thus

$$m = C(12 + 2, 2) = C(14, 2) = 91.$$

3.3 Let E be the equation x + y + z = 18. Find the number m of nonnegative solutions to E with the conditions that x < 7, y < 8, z < 9.

Let S be the set of all nonnegative solutions of E. Let A be the set of solutions for which x ≥ 7, let B be the set of solutions for which y ≥ 8, and let C be the set of solutions for which z ≥ 9. Then

$$m = |A^C \cap B^C \cap C^C|$$

As in Problem 6.2, we obtain:

$$|A| = C(11 + 2, 2) = 78, \quad |A \cap B| = C(3 + 2, 2) = 10$$
$$|B| = C(10 + 2, 2) = 66, \quad |A \cap C| = C(2 + 2, 2) = 6$$
$$|C| = C(9 + 2, 2) = 55, \quad |B \cap C| = C(1 + 2, 2) = 3$$

Also, $|S| = C(18 + 2, 2) = 190$ and $|A \cap B \cap C| = 0$. By the Inclusion–Exclusion Principle,

$$m = 190 - (78 + 66 + 55) + (10 + 6 + 3) - 0 = 10$$

3.4 There are 9 students in a class. Find the number m of ways: (a) the 9 students can take 3 different tests if 3 students are to take each test; (b) the 9 students can be partitioned into 3 teams A, B, C so that each team contains 3 students,

(a) Method 1: We seek the number m of partitions of the 9 students into cells containing 3 students. By Theorem 6.2, m = 9!/(3!3!3!) = 5040.

Method 2: There are C(9, 3) to choose three students to take the first test; then there are C(6, 3) ways to choose 3 students to take the second test; and the remaining students take the third test. Thus m = C(9, 3)C(6, 3) = 5040.

(c) Each partition {A, B, C} of the students can be arranged in 3! = 6 ways as an ordered partition. By (a), there are 5040 such ordered partitions. Hence m = 5040/6 = 840.

3.5 Find the number N of ways a company can assign 7 projects to 4 people so that each person gets at least one project.

We want to find the number N of onto functions from a set with m = 7 elements onto a set with n = 4 elements. We use Theorem 3.4:

$$N = 4^2 - C(4, 1)(3^7) + C(4, 2)(2^7) - C(4, 3)(1^7)$$

$$= 4^7 - 4(3^7) + 6(2^7) - 4(1^7) = 16\,384 - 8748 + 768 - 4 = 8400$$

3.6 Prove Theorem 3.5: $D_n = n![1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}]$

Recall (Section 3.3) that S_n denotes the set of permutations on X = {1, 2, . . . ,n} and $|S_n| = n!$. For i = 1, . . . , n, let F_i denote all permutations in S_n which "fix i," that is, $F_i = \{\sigma \in S_n \mid \sigma(i) = i\}$. Then, for distinct subscripts

$$|F_r| = (n - 1)!, \quad |F_i \cap F_i| = (n - 2)!, \ldots |F_{i1} \cap F_{i2} \cap \ldots \cap F_{ir}| = (n - r)!$$

Let Y denote the set of all derangements in S_n. Then $D_n = |S_n| - s_1 + s_2 - s_3 + \cdots + (-1)^n s_n$

By the Inclusion–Exclusion principle, Where $D_n = |S_n| - s_1 + s_2 - s_3 + \cdots + (-1)^n s_n$

$$s_r = \sum_{i_1 < i_2 < i_3} |F_1 \cap F_2 \cap \ldots \ldots \cap F_{ir}| = C(n, r)(n - r)! = \frac{n!}{r!}$$

Setting $|S_n| = n!$ and $s_r = n!/r!$ in the formula for D_n gives us our theorem.

PIGEONHOLE PRINCIPLE

3.7. Suppose five points are chosen from the interior of a square S where each side has length two inches. Show that the distance between two of the points must be less than 2 inches.

Draw two lines between the opposite sides of S which partitions S into four sub-squares each whose sides have length one inch. By the Pigeonhole Principle, two of the points lie in one of the sub-squares. The diagonal of each sub-square is 2 inches, so the distance between the two points is less than 2 inches.

3.8. Let p and q be positive integers. A number r is said to satisfy the (p, q)-Ramsey property if a set of r people must have a subset of p mutual friends or a subset of q mutual strangers. The Ramsey number R(p, q) is the smallest such integer r. Show that R(3, 3) = 6.

By Example 3.5, $R(3, 3) \geq 6$. We show that $R(3, 3) > 5$. Consider five people who are sitting around a circular table, and suppose each person is only friends with the people sitting next to him/her. No three people can be strangers since two of the three people must be sitting next to each other. Also no three people can be mutual friends since they cannot be sitting next to each other. Thus $R(3, 3) > 5$. Accordingly $R(3, 3) = 6$.

3.9 Suppose a team X plays 18 games in a two-week 14-day period, and plays at least one game a day. Show that there is a period of days in which exactly 9 games were played.

Let $S = \{s_1 , s_2 , \ldots , s_{14} \}$ where s_i is the number of games X played from the first day to the ith day. Then $s_{14} = 18$, and all the s_i are distinct. Let $T = \{t_1 , t_2 , \ldots , t_{14} \}$ where $t_i = s_i + 9$. Then $t_{14} = 18 + 9 = 27$, and the t_i are distinct. Together S and T have $14 + 14 = 28$ numbers, which lie between 1 and 27. By the Pigeonhole Principle, two of the numbers must be equal. However the entries in S and the entries in T are distinct. Thus there is $s_j \in S$ and $t_n \in T$ such that $s_j = t_k = s_k + 9$. Therefore,

$$9 = s_j - s_n = \text{number of games played in days } k + 1, k + 2, \ldots , j - 1, j$$

RECURRENCE RELATION

3.10 Consider the second-order homogeneous recurrence relation $a_n = a_{n-1} + 2a_{n-2}$ with initial conditions $a_0 = 2$, $a_1 = 7$, (a) Find the next three terms of the sequence. (b) Find the general solution. (c) Find the unique solution with the given initial

conditions.

(a) Each term is the sum of the preceding term plus twice its second preceding term. Thus:

$$a_2 = 7 + 2(2) = 11, \quad a_3 = 11 + 2(7) = 25, \quad a_4 = 25 + 2(11) = 46$$

(b) First we find the characteristic polynomial $\Delta(t)$ and its roots:

$$\Delta(x) = x^2 - x - 2 = (x - 2)(x + 1); \quad \text{roots } r_1 = 2, r_2 = -1$$

Since the roots are distinct, we use Theorem 3.8 to obtain the general solution:

$$a_n = c_1 (2^n) + c_2 (-1)^n$$

(c) The unique solution is obtained by finding c_1 and c_2 using the initial conditions:

$$\text{For } n = 0,\ a_0 = 2, \text{ we get: } c_1(2^0) + c_2(-1)^0 = 2 \quad \text{Or} \quad c_1 + c_2 = 2$$
$$\text{For } n = 1,\ a_1 = 7, \text{ we get: } c_1(2) + c_2(-1) = 7 \quad \text{Or} \quad 2c_1 - c_2 = 7$$

Solving the two equations for c_1 and c_2 yields $c_1 = 3$ and $c_2 = 1$. The unique solution follows: $a_n = 3(2^n) - (-1)^n$

3.12 Consider the third-order homogeneous recurrence relation $a_n = 6a_{n-1} - 12a_{n-2} + 8a_{n-3}$

(a) Find the general solution.
(b) Find the solution with initial conditions $a_0 = 3$, $a_1 = 4$, $a_2 = 12$.

(a) First we find the characteristic polynomial

$$\Delta(x) = x^3 - 6x^2 + 12x - 8 = (x - 2)^3$$

Then $\Delta(x)$ has only one root $r_0 = 2$ which has multiplicity 3. Thus the general solution of the recurrence relation follows:

$$a_n = c_1 (2^n) + c_2\, n(2^n) + c_3\, n^2 (2^n) = (c_1 + c_2\, n + c_3\, n^2)(2^n)$$

(b) We find the values for c_1 , c_2 , and c_3 as follows:

For n = 0, a_0 = 3 we get: c_1 = 3
For n = 1, a_1 = 4 we get: $2c_1 + 2c_2 + 2c_3 = 4$
For n = 2, a_2 = 12 we get: $4c_1 + 8c_2 + `16c_2 = 12$

Solving the system of three equations in c_1 , c_2 , c_3 yields the solution

$$c_1 = 3, \quad c_2 = -2, \quad c_3 = 1$$

Thus the unique solution of the recurrence relation follows:

$$a_n = (3 - 2n + n^2)(2^n)$$

CHAPTER 4

GRAPH THEORY

4.1 INTRODUCTION

In mathematics and computer science, graph theory is the study of graphs, which are mathematical structures used to model pair wise relations between objects. A graph in this context is made up of vertices or nodes or points and edges or arcs or lines that connect them. A graph may be undirected, meaning that there is no distinction between the two vertices associated with each edge, or its edges may be directed from one vertex to another; see Graph (mathematics) for more detailed definitions and for other variations in the types of graph that are commonly considered. Graphs are one of the prime objects of study in discrete mathematics.

Refer to the glossary of graph theory for basic definitions in graph theory.

Graphs, directed graphs, trees and binary trees appear in many areas of mathematics and computer science. This and the next two chapters will cover these topics. However, in order to understand how these objects may be stored in memory and to understand algorithms on them, we need to know a little about certain data structures. We assume the reader does understand linear and two-dimensional arrays; hence we will only discuss linked lists and pointers, and stacks and queues below.

Linked Lists and Pointers

Linked lists and pointers will be introduced by means of an example. Suppose a brokerage firm maintains a file in which each record contains a customer's name and salesman; say the file contains the following data:

Customer	Adams	Brown	Clark	Drew	Evans	Farmer	Geller	Hiller	Infeld
Salesman	Smith	Ray	Ray	Jones	Smith	Jones	Ray	Smith	Ray

There are two basic operations that one would want to perform on the data:

Operation A: Given the name of a customer, find his salesman.
Operation B: Given the name of a salesman, find the list of his customers.

We discuss a number of ways the data may be stored in the computer, and the ease with which one can perform the operations A and B on the data.

Clearly, the file could be stored in the computer by an array with two rows (or columns) of nine names. Since the customers are listed alphabetically, one could easily perform operation A. However, in order to perform operation B one must search through the entire array.

One can easily store the data in memory using a two-dimensional array where, say, the rows correspond to an alphabetical listing of the customers and the columns correspond to an alphabetical listing of the salesmen, and where there is a 1 in the matrix indicating the salesman of a customer and there are 0's elsewhere. The main drawback of such a representation is that there may be a waste of a lot of memory because many 0's may be in the matrix. For example, if a firm has 1000 customers and 20 salesmen, one would need 20 000 memory locations for the data, but only 1000 of them would be useful.

We discuss below a way of storing the data in memory which uses linked lists and pointers. By a linked list, we mean a linear collection of data elements, called nodes, where the linear order is given by means of a field of pointers. Figure 4-1 is a schematic diagram of a linked list with six nodes. That is, each node is divided into two parts: the first part contains the information of the element (e.g., NAME, ADDRESS, . . .), and the second part, called the link field or next pointer field, contains the address of the next node in the list. This pointer field is indicated by an arrow drawn from one node to the next node in the list. There is

also a variable pointer, called START in Fig. 4-1, which gives the address of the first node in the list. Furthermore, the pointer field of the last node contains an invalid address, called a null pointer, which indicates the end of the list.

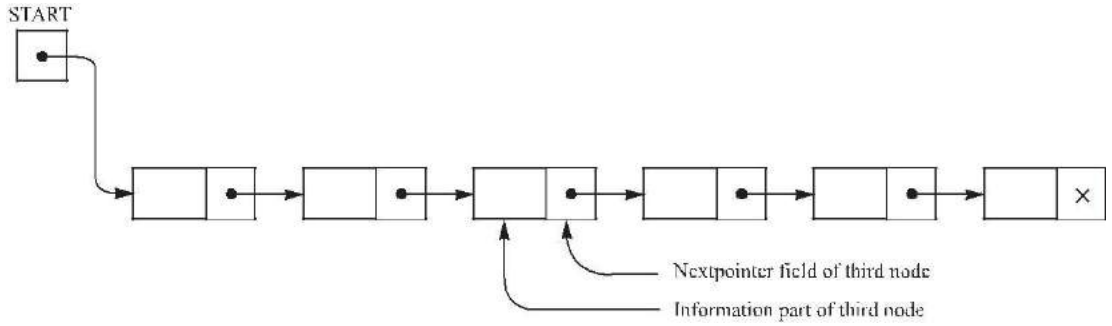


Fig. 4-1 Linked list with 6 nodes

One main way of storing the original data pictured in Fig. 4-2, uses linked lists. Observe that there are separate (sorted alphabetically) arrays for the customers and the salesmen. Also, there is a pointer array SLSM parallel to CUSTOMER which gives the location of the salesman of a customer, hence operation A can be performed very easily and quickly. Furthermore, the list of customers of each salesman is a linked list as discussed above. Specifically, there is a pointer array START parallel to SALESMAN which points to the first customer of a salesman, and there is an array NEXT which points to the location of the next customer in the salesman's list (or contains a 0 to indicate the end of the list). This process is indicated by the arrows in Fig. 8-2 for the salesman Ray.

	Customer	SLSM	NEXT
1	Adams	3	5
2	Brown	2	3
3	Clark	2	7
4	Drew	1	6
5	Evans	3	8
6	Farmer	1	0
7	Geller	2	9
8	Hill	3	0
9	Infeld	2	0

Salesman	START	
Jones	4	1
Ray	2	2
Smith	1	3

Fig. 4-2

Operation B can now be performed easily and quickly; that is, one does not need to search through the list of all customers in order to obtain the list of customers of a given salesman. Figure 4-3 gives such an algorithm (which is written in pseudo code).

Stacks, Queues, and Priority Queues

There are data structures other than arrays and linked lists which will occur in our graph algorithms. These structures, stacks, queues, and priority queues, are briefly described below.

(a) **Stack**: A stack, also called a last-in first-out (LIFO) system, is a linear list in which insertions and deletions can take place only at one end, called the "top" of the list. This structure is similar in its operation to a stack of dishes on a spring system, as pictured in Fig. 4-4(a). Note that new dishes are inserted only at the top of the stack and dishes can be deleted only from the top of the stack.

Algorithm 1 The name of a salesman is read and the list of his customers is printed.

Step 1. Read XXX.

Step 2. Find *K* such that SALESMAN[K] = XXX. [Use binary search.]

Step 3. Set PTR := START[*K*]. [Initializes pointer PTR.]

Step 4. Repeat while PTR ≠ NULL.

(*a*) Print CUSTOMER[PTR].

(*b*) Set PTR := NEXT[PTR]. [Update PTR.]

[End of loop.]

Step 5. Exit.

Fig.4-3

(b) **Queue**: A queue, also called a first-in first-out (FIFO) system, is a linear list in which deletions can only take place at one end of the list, the "front" of the list, and insertions can only take place at the other end of the list, the "rear" of the list. The structure operates in much the same way as a line of people waiting at a bus stop, as pictured in Fig. 4-4(b). That is, the first person in line is the first person to board the bus, and a new person goes to the end of the line.

(c) **Priovity queue**: Let S be a set of elements where new elements may be periodically inserted, but where the current largest element (element with the "highest priority") is always deleted. Then S is called a priority queue. The rules "women and children first" and "age before beauty" are examples of priority queues. Stacks and ordinary queues are special kinds of priority queues. Specifically, the element with the highest priority in a stack is the last element inserted, but the element with the highest priority in a queue is the first element inserted.

(*a*) Stack of dishes. (*b*) Queue waiting for a bus.

Fig. 4-4

4.2 GRAPHS AND MULTIGRAPHS

A graph G consists of two things:

(i) A set V = V (G) whose elements are called vertices, points, or nodes of G.
(ii) A set E = E(G) of unordered pairs of distinct vertices called edges of G.

We denote such a graph by G(V , E) when we want to emphasize the two parts of G.

Vertices u and v are said to be adjacent or neighbors if there is an edge e = {u, v }. In such a case, u and v are called the endpoints of e, and e is said to connect u and v. Also, the edge e is said to be incident on each of its endpoints u and v. Graphs are pictured by diagrams in the plane in a natural way. Specifically, each vertex v in V is represented by a dot (or small circle), and each edge e = $\{v_1, v_2\}$ is represented by a curve which connects its endpoints v_1 and v_2 For example, Fig.

4-5(a) represents the graph G(V , E) where:

(i) V consists of vertices A, B, C, D.
(ii) E consists of edges e_1 = {A, B}, e_2 = {B, C}, e_3 = {C, D}, e_4 = {A, C}, e_5 = {B, D}.

In fact, we will usually denote a graph by drawing its diagram rather than explicitly listing its vertices and edges.

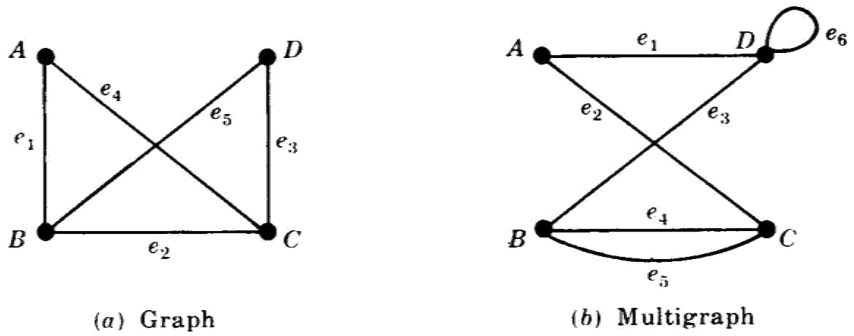


Fig. 4-5

Multi-graphs

Consider the diagram in Fig. 4-5(b). The edges e_4 and e_5 are called multiple edges since they connect the same endpoints, and the edge e_6 is called a loop since its endpoints are the same vertex. Such a diagram is called a multi-graph; the formal definition of a graph permits neither multiple edges nor loops. Thus a graph may be defined to be a multi-graph without multiple edges or loops.

Remark: Some texts use the term graph to include multi-graphs and use the term simple graph to mean a graph without multiple edges and loops.

4.3 DEGREE OF A VERTEX

The degree of a vertex v in a graph G, written deg (v), is equal to the number of edges in G which contain v, that is, which are incident on v. Since each edge is counted twice in counting the degrees of the vertices of G, we have the following simple but important result.

THEOREM 4.1: The sum of the degrees of the vertices of a graph G is equal to twice the number of edges in G.

Consider, for example, the graph in Fig. 4-5(a). We have

$$\deg(A) = 2, \quad \deg(B) = 3, \quad \deg(C) = 3, \quad \deg(D) = 2.$$

The sum of the degrees equals 10 which, as expected, is twice the number of edges. A vertex is said to be even or odd according as its degree is an even or an odd number. Thus A and D are even vertices whereas B and C are odd vertices.

Theorem 4.1 also holds for multi-graphs where a loop is counted twice toward the degree of its endpoint. For example, in Fig. 4-5(b) we have deg(D) = 4 since the edge e_6 is counted twice; hence D is an even vertex.

A vertex of degree zero is called an isolated vertex.

Finite Graphs, Trivial Graph

A multi-graph is said to be finite if it has a finite number of vertices and a finite number of edges. Observe that a graph with a finite number of vertices must automatically have a finite number of edges and so must be finite. The finite graph with one vertex and no edges, i.e., a single point, is called the trivial graph. Unless otherwise specified, the multi-graphs in this book shall be finite.

SUBGRAPHS, ISOMORPHIC AND HOMEOMORPHIC GRAPHS

This section will discuss important relationships between graphs.

Sub-graphs

Consider a graph G = G(V , E). A graph H = H (V , E) is called a sub-graph of G if the vertices and edges of H are contained in the vertices and edges of G, that is, if $V \subseteq V$ and $E \subseteq E$. In particular:

(i) A sub-graph H (V , E) of G(V , E) is called the sub-graph induced by its vertices V if its edge set E contains all edges in G whose endpoints belong to vertices in H .

(ii) If v is a vertex in G, then G − v is the sub-graph of G obtained by deleting v from G and deleting all edges in G which contain v.

(iii) If e is an edge in G, then G − e is the sub-graph of G obtained by simply deleting the edge e from G.

Isomorphic Graphs

Graphs G(V , E) and G(V *, E*) are said to be isomorphic if there exists a one-to-one correspondence f : V → V * such that {u, v} is an edge of G if and only if {f (u), f (v)} is an edge of G*. Normally, we do not distinguish between isomorphic graphs (even though their diagrams may "look different"). Figure 4-6 gives ten graphs pictured as letters. We note that A and R are isomorphic graphs. Also, F and T are isomorphic graphs, K and X are isomorphic graphs and M, S, V , and Z are isomorphic graphs.

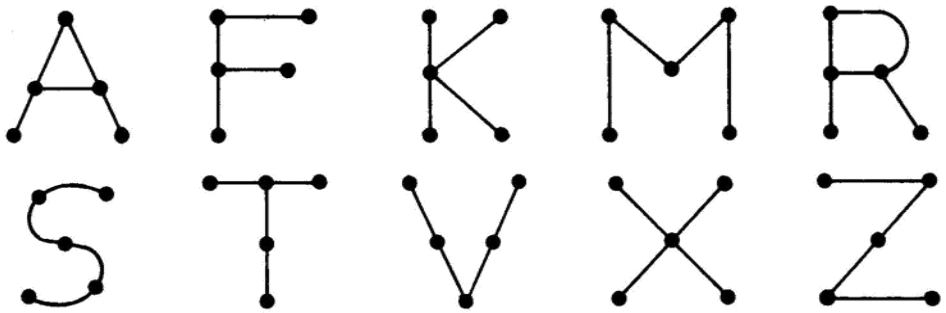


Fig. 4-6

Homeomorphic Graphs

Given any graph G, we can obtain a new graph by dividing an edge of G with additional vertices. Two graphs G and G* are said to homeomorphic if they can be obtained from the same graph or isomorphic graphs by this method. The graphs (a) and (b) in Fig. 4-7 are not isomorphic, but they are homeomorphic since they can be obtained from the graph (c) by adding appropriate vertices.

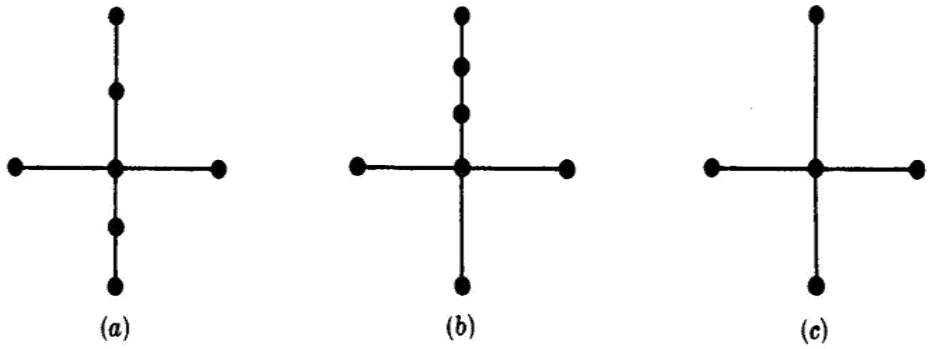


Fig. 4-7

4.4 PATHS, CONNECTIVITY

A path in a multi-graph G consists of an alternating sequence of vertices and edges of the form

$$v_0,\ e_1,\ v_1,\ e_2,\ v_2,\ \ldots,\ e_{n-1},\ v_{n-1},\ e_n,\ v_n$$

where each edge e_i contains the vertices v_{i-1} and v_i (which appear on the sides of e_i in the sequence). The number n of edges is called the length of the path. When there is no ambiguity, we denote a path by its sequence of vertices (v_0, v_1, . . . , v_n). The path is said to be closed if $v_0 = v_n$. Otherwise, we say the path is from v_0, to v_n or between v_0 and v_n, or connects v_0 to v_n.

A simple path is a path in which all vertices are distinct. (A path in which all edges are distinct will be called a trail.) A cycle is a closed path of length 3 or more in which all vertices are distinct except $v_0 = v_n$. A cycle of length k is called a k-cycle.

EXAMPLE 4.1 Consider the graph G in Fig. 8-8(a). Consider the following sequences:

$$\alpha = (P_4, P_1, P_2, P_5, P_1, P_2, P_3, P_6),\quad \beta = (P_4, P_1, P_5, P_2, P_6),$$
$$\gamma = (P_4, P_1, P_5, P_2, P_3, P_5, P_6),\qquad \delta = (P_4, P_1, P_5, P_3, P_6).$$

The sequence α is a path from P_4 to P_6; but it is not a trail since the edge $\{P_1, P_2\}$ is used twice. The sequence β is not a path since there is no edge $\{P_2, P_6\}$. The sequence γ is a trail since no edge is used twice; but it is not a simple path since the vertex P_5 is used twice. The sequence δ is a simple path from P_4 to P_6; but it is not the shortest path (with respect to length) from P_4 to P_6. The shortest path

from P_4 to P_6 is the simple path (P_4, P_5, P_6) which has length 2.

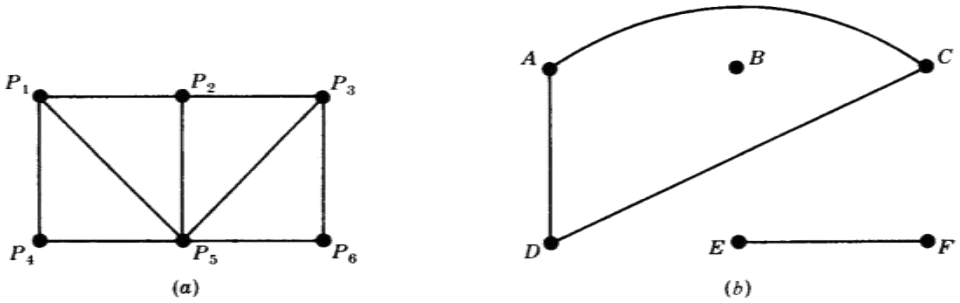


Fig.4-8

By eliminating unnecessary edges, it is not difficult to see that any path from a vertex u to a vertex v can be replaced by a simple path from u to v. We state this result formally.

THEOREM 4.2: There is a path from a vertex u to a vertex v if and only if there exists a simple path from u to v.

Connectivity, Connected Components

A graph G is connected if there is a path between any two of its vertices. The graph in Fig. 4-8(a) is connected, but the graph in Fig. 4-8(b) is not connected since, for example, there is no path between vertices D and E. Suppose G is a graph. A connected sub-graph H of G is called a connected component of G if H is not contained in any larger connected sub-graph of G. It is intuitively clear that any graph G can be partitioned into its connected components. For example, the graph G in Fig. 4-8(b) has three connected components, the sub-graphs induced by the vertex sets {A, C, D}, {E, F }, and {B}.

The vertex B in Fig. 4-8(b) is called an isolated vertex since B does not belong to any edge or, in other words, deg(B) = 0. Therefore, as noted, B itself forms a connected component of the graph.

Remark: Formally speaking, assuming any vertex u is connected to itself, the relation "u is connected to v" is an equivalence relation on the vertex set of a graph G and the equivalence classes of the relation form the connected components of G.

Distance and Diameter

Consider a connected graph G. The distance between vertices u and v in G, written d(u, v), is the length of the shortest path between u and v. The diameter of G, written diam(G), is the maximum distance between any two points in G. For example, in Fig. 4-9(a), d(A, F) = 2 and diam(G) = 3, whereas in Fig. 4-9(b), d(A, F) = 3 and diam(G) = 4.

Cutpoints and Bridges

Let G be a connected graph. A vertex v in G is called a cutpoint if G − v is disconnected. (Recall that G − v is the graph obtained from G by deleting v and all edges containing v.) An edge e of G is called a bridge if G − e is disconnected. (Recall that G − e is the graph obtained from G by simply deleting the edge e). In Fig. 4-9(a), the vertex D is a cutpoint and there are no bridges. In Fig. 4-9(b), the edge = {D, F } is a bridge. (Its endpoints D and F are necessarily cutpoints.)

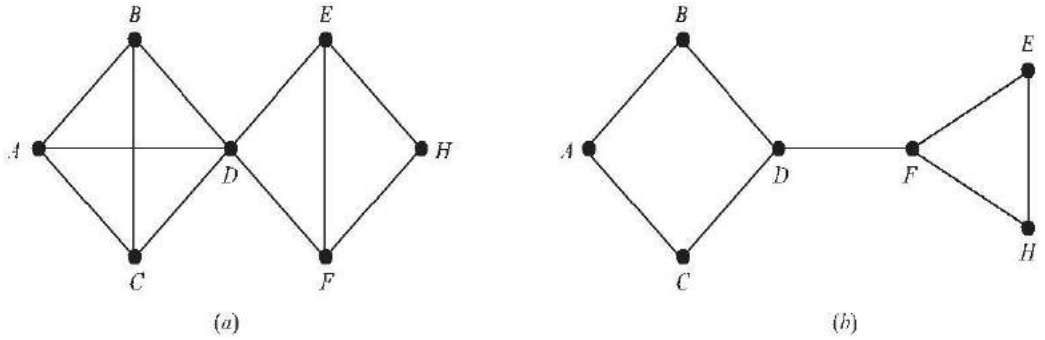
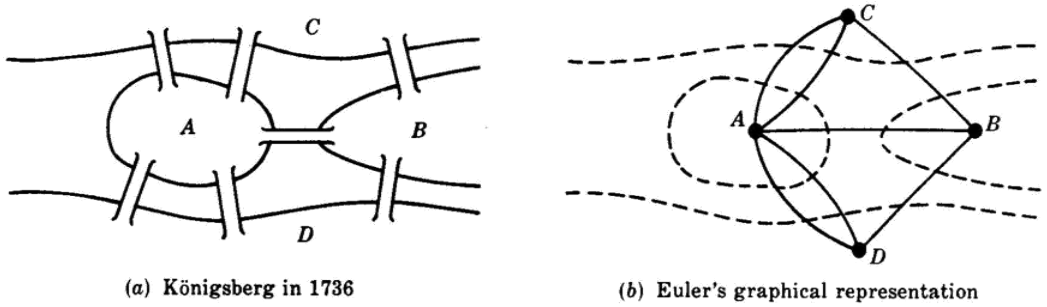


Fig. 4-9

4.5 TRAVERSABLE AND EULERIAN GRAPHS, BRIDGES OF KÖNIGSBERG

The eighteenth-century East Prussian town of Königsberg included two islands and seven bridges as shown in Fig. 4-10(a). Question: Beginning anywhere and ending anywhere, can a person walk through town crossing all seven bridges but not crossing any bridge twice? The people of Königsberg wrote to the celebrated Swiss mathematician L. Euler about this question. Euler proved in 1736 that such a walk is impossible. He replaced the islands and the two sides of the river by points and the bridges by curves, obtaining Fig. 4-10(b).

Observe that Fig. 4-10(b) is a multigraph. A multigraph is said to be traversable if it "can be drawn without any breaks in the curve and without repeating any edges," that is, if there is a path which includes all vertices and uses each edge exactly once. Such a path must be a trail (since no edge is used twice) and will be called a traversable trail. Clearly a traversable multigraph must be finite and connected.



(a) Königsberg in 1736 (b) Euler's graphical representation

Fig. 4-10

We now show how Euler proved that the multi-graph in Fig. 4-10(b) is not traversable and hence that the walk in Königsberg is impossible. Recall first that a vertex is even or odd according as its degree is an even or an odd number. Suppose a multi-graph is traversable and that a traversable trail does not begin or end at a vertex P . We claim that P is an even vertex. For whenever the traversable trail enters P by an edge, there must always be an edge not previously used by which the trail can leave P . Thus the edges in the trail incident with P must appear in pairs, and so P is an even vertex. Therefore if a vertex Q is odd, the traversable trail must begin or end at Q. Consequently, a multi-graph with more than two odd vertices cannot be traversable. Observe that the multi-graph corresponding to the Königsberg bridge problem has four odd vertices. Thus one cannot walk through Königsberg so that each bridge is crossed exactly once.

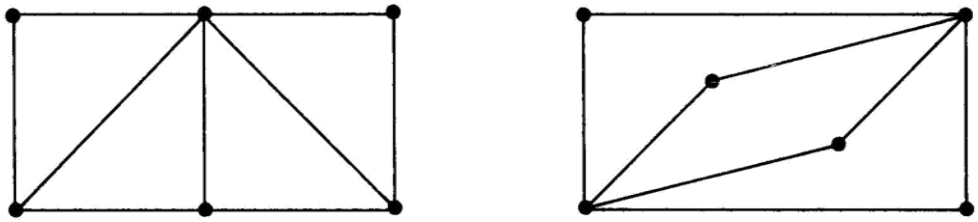
Euler actually proved the converse of the above statement, which is contained in the following theorem and corollary. (The theorem is proved in Problem 4.9.) A graph G is called an Eulerian graph if there exists a closed traversable trail, called an Eulerian trail.

THEOREM 4.3 (Euler): A finite connected graph is Eulerian if and only if each vertex has even degree.

Corollary 4.4: Any finite connected graph with two odd vertices is traversable. A traversable trail may begin at either odd vertex and will end at the other odd vertex.

HAMILTONIAN GRAPHS

The above discussion of Eulerian graphs emphasized traveling edges; here we concentrate on visiting vertices. A Hamiltonian circuit in a graph G, named after the nineteenth-century Irish mathematician William Hamilton (1803–1865), is a closed path that visits every vertex in G exactly once. (Such a closed path must be a cycle.) If G does admit a Hamiltonian circuit, then G is called a Hamiltonian graph. Note that an Eulerian circuit traverses every edge exactly once, but may repeat vertices, while a Hamiltonian circuit visits each vertex exactly once but may repeat edges. Figure 4-11 gives an example of a graph which is Hamiltonian but not Eulerian, and vice versa.



(*a*) Hamiltonian and non-Eulerian

(*b*) Eulerian and non-Hamiltonian

Fig.4-11

Although it is clear that only connected graphs can be Hamiltonian, there is no simple criterion to tell us whether or not a graph is Hamiltonian as there is for Eulerian graphs. We do have the following sufficient condition which is due to G. A. Dirac.

Theorem 4.5: Let G be a connected graph with n vertices. Then G is Hamiltonian if $n \geq 3$ and $n \leq \deg(v)$ for each vertex v in G.

4.6 LABELED AND WEIGHTED GRAPHS

A graph G is called a labeled graph if its edges and/or vertices are assigned data of one kind or another. In particular, G is called a weighted graph if each edge e of G is assigned a nonnegative number we called the weight or length of v. Figure 4-12 shows a weighted graph where the weight of each edge is given in the obvious way. The weight (or length) of a path in such a weighted graph G is defined to be the sum of the weights of the edges in the path. One important problem in graph theory is to find a shortest path, that is, a path of minimum weight (length), between any two given vertices. The length of a shortest path between P and Q in Fig. 4-12 is 14; one such path is

$$(P , A_1, A_2, A_5, A_3, A_6, Q)$$

The reader can try to find another shortest path.

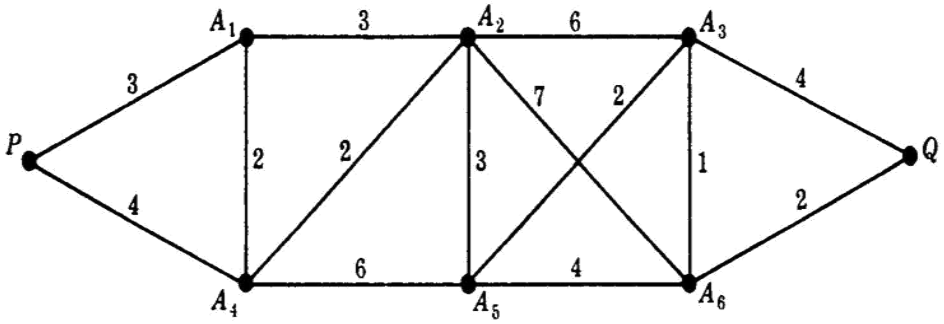


Fig. 4-12

4.7 COMPLETE, REGULAR, AND BIPARTITE GRAPHS

There are many different types of graphs. This section considers three of them: complete, regular, and bipartite graphs.

Complete Graphs

A graph G is said to be complete if every vertex in G is connected to every other vertex in G. Thus a complete graph G must be connected. The complete graph with n vertices is denoted by K_n. Figure 4-13 shows the graphs K_1 through K_6.

Regular Graphs

A graph G is regular of degree k or k-regular if every vertex has degree k. In other words, a graph is regular if every vertex has the same degree.

The connected regular graphs of degrees 0, 1, or 2 are easily described. The connected 0-regular graph is the trivial graph with one vertex and no edges. The connected 1-regular graph is the graph with two vertices and one edge connecting them. The connected 2-regular graph with n vertices is the graph which consists of a single n-cycle, in Fig. 4-14.

The 3-regular graphs must have an even number of vertices since the sum of the degrees of the vertices is an even number (Theorem 4.1). Figure 4-15 shows two connected 3-regular graphs with six vertices. In general, regular graphs can be quite complicated. For example, there are nineteen 3-regular graphs with ten vertices. We note that the complete graph with n vertices K_n is regular of degree $n - 1$.

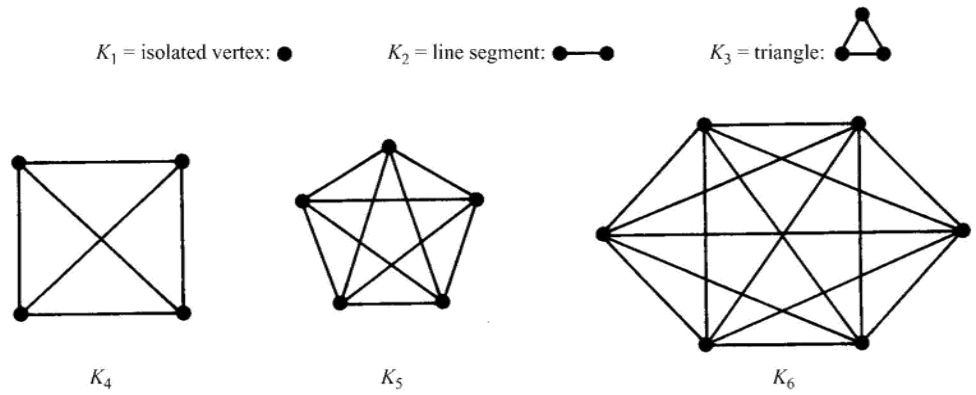


Fig. 4-13

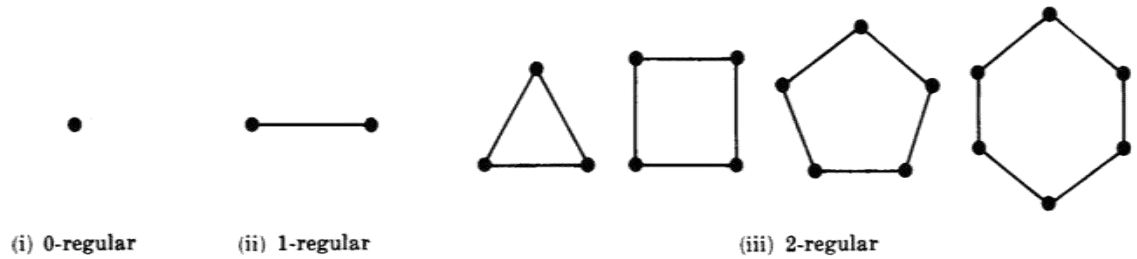


Fig. 4-14

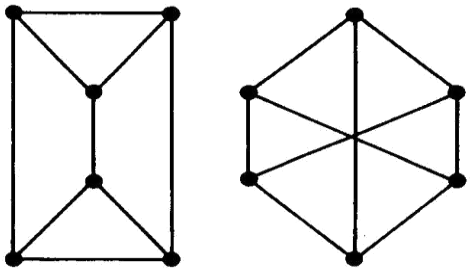


Fig. 4-15

Bipartite Graphs

A graph G is said to be bipartite if its vertices V can be partitioned into two subsets M and N such that each edge of G connects a vertex of M to a vertex of N . By a complete bipartite graph, we mean that each vertex of M is connected to each vertex of N ; this graph is denoted by $K_{m,n}$ where m is the number of vertices in M and n is the number of vertices in N , and, for standardization, we will assume $m \leq n$. Figure 8-16 shows the graphs $K_{2,3}$, $K_{3,3}$, and $K_{2,4}$, Clearly the graph $K_{m,n}$ has mn edges.

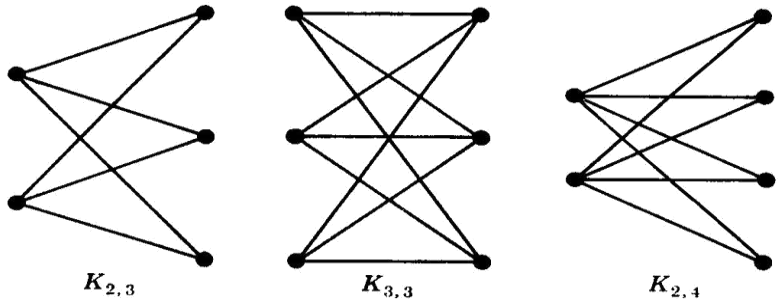


Fig. 4-16

4.8 TREE GRAPHS

A graph T is called a tree if T is connected and T has no cycles. Examples of trees are shown in Fig. 4-17. A forest G is a graph with no cycles; hence the connected components of a forest G are trees. A graph without cycles is said to be cycle-free. The tree consisting of a single vertex with no edges is called the degenerate tree.

Consider a tree T . Clearly, there is only one simple path between two vertices of T ; otherwise, the two paths would form a cycle. Also:

(a) Suppose there is no edge {u, v} in T and we add the edge e = {u, v} to T . Then the simple path from u to v in T and e will form a cycle; hence T is no longer a tree.
(b) On the other hand, suppose there is an edge e = {u, v} in T , and we delete e from T . Then T is no longer connected (since there cannot be a path from u to v); hence T is no longer a tree.

The following theorem (proved in Problem 4.14) applies when our graphs are finite.

Theorem 4.6: Let G be a graph with n > 1 vertices. Then the following are equivalent:

(i) G is a tree.
(ii) G is a cycle-free and has n − 1 edges.
(iii) G is connected and has n − 1 edges.

This theorem also tells us that a finite tree T with n vertices must have n − 1 edges. For example, the tree in Fig. 4-17(a) has 9 vertices and 8 edges, and the tree in Fig. 4-17(b) has 13 vertices and 12 edges.

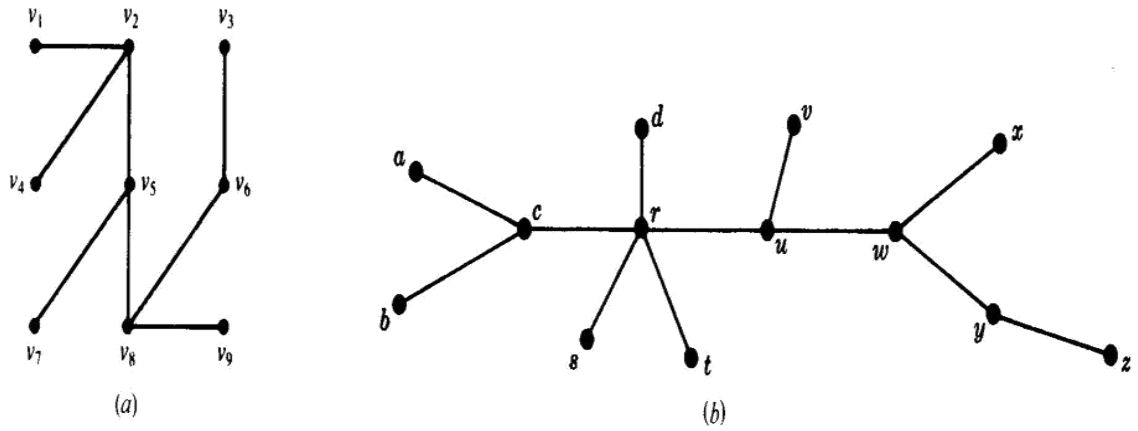


Fig. 4-17

4.9 SPANNING TREES

A sub-graph T of a connected graph G is called a spanning tree of G if T is a tree and T includes all the vertices of G. Figure 8-18 shows a connected graph G and spanning trees T_1, T_2, and T_3 of G.

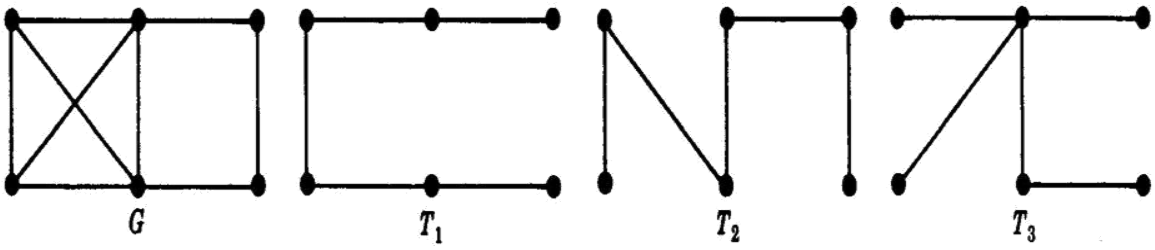


Fig. 4-18

Minimum Spanning Trees

Suppose G is a connected weighted graph. That is, each edge of G is assigned a nonnegative number called the weight of the edge. Then any spanning tree T of G is assigned a total weight obtained by adding the weights of the edges in T . A minimal spanning tree of G is a spanning tree whose total weight is as small as possible.

Algorithms 2 and 3, which appear in Fig. 4-19, enable us to find a minimal spanning tree T of a connected weighted graph G where G has n vertices. (In which case T must have n − 1 vertices.)

Algorithm 2: The input is a connected weighted graph G with n vertices.

Step 1. Arrange the edges of G in the order of decreasing weights.

Step 2. Proceeding sequentially, delete each edge that does not disconnect the graph until $n - 1$ edges remain.

Step 3. Exit.

Algorithm 3 (Kruskal): The input is a connected weighted graph G with n vertices.

Step 1. Arrange the edges of G in order of increasing weights.

Step 2. Starting only with the vertices of G and proceeding sequentially, add each edge which does not result in a cycle until $n - 1$ edges are added.

Step 3. Exit.

Fig. 4-19

The weight of a minimal spanning tree is unique, but the minimal spanning tree itself is not. Different minimal spanning trees can occur when two or more edges have the same weight. In such a case, the arrangement of the edges in Step 1 of Algorithms 2 or 3 is not unique and hence may result in different minimal spanning trees as illustrated in the following example.

EXAMPLE 4.2 Find a minimal spanning tree of the weighted graph Q in Fig. 4-20(a). Note that Q has six vertices, so a minimal spanning tree will have five edges.

(a) Here we apply Algorithm 2.

First we order the edges by decreasing weights, and then we successively delete edges without disconnecting Q until five edges remain. This yields the following data:

Edges	BC	AF	AC	BE	CE	BF	AE	DF	BD
Weight	8	7	7	7	6	5	4	4	3
Delete	Yes	Yes	Yes	No	No	Yes			

Thus the minimal spanning tree of Q which is obtained contains the edges

BE,CE, AE, DF, BD

The spanning tree has weight 24 and it is shown in Fig. 4-20(b).

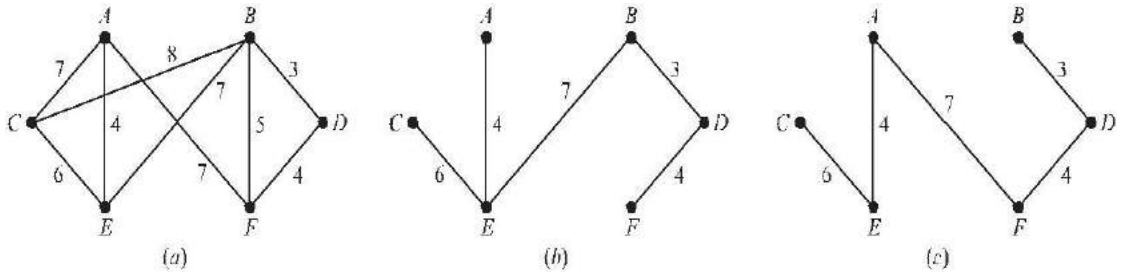


Fig. 4-20

(b) Here we apply Algorithm 3.

First we order the edges by increasing weights, and then we successively add

edges without forming any cycles until five edges are included. This yields the following data:

Edges	BD	AE	DF	BF	CE	AC	AF	BE	BC
Weight	3	4	4	5	6	7	7	7	8
Add?	Yes	Yes	Yes	No	Yes	No	Yes		

Thus the minimal spanning tree of Q which is obtained contains the edges

BD, AE, DF, CE, AF

The spanning tree appears in Fig. 4-20(c). Observe that this spanning tree is not the same as the one obtained using Algorithm 2 as expected it also has weight 24.

Remark: The above algorithms are easily executed when the graph G is relatively small as in Fig. 4-20(a). Suppose G has dozens of vertices and hundreds of edges which, say, are given by a list of pairs of vertices. Then even deciding whether G is connected is not obvious; it may require some type of depth-first search (DFS) or breadth-first search (BFS) graph algorithm. Later sections and the next chapter will discuss ways of representing graphs G in memory and will discuss various graph algorithms.4

4.10PLANAR GRAPHS

A graph or multi-graph which can be drawn in the plane so that its edges do not cross is said to be planar. Although the complete graph with four vertices K_4 is usually pictured with crossing edges as in Fig. 4-21(a), it can also be drawn with non-crossing edges as in Fig. 4-21(b); hence K_4 is planar. Tree graphs form an important class of planar graphs. This section introduces our reader to these important graphs.

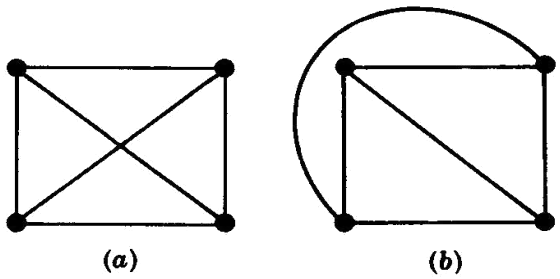


Fig. 4-21

Maps, Regions

A particular planar representation of a finite planar multi-graph is called a map. We say that the map is connected if the underlying multi-graph is connected. A given map divides the plane into various regions. For example, the map in Fig. 4-22 with six vertices and nine edges divides the plane into five regions. Observe that four of the regions are bounded, but the fifth region, outside the diagram, is unbounded. Thus there is no loss in generality in counting the number of regions if we assume that our map is contained in some large rectangle rather than in the entire plane.

Observe that the border of each region of a map consists of edges. Sometimes the edges will form a cycle, but sometimes not. For example, in Fig. 4-22 the borders of all the regions are cycles except for r_3. However, if we do move counterclockwise around r_3 starting, say, at the vertex C, then we obtain the closed path

(C, D, E, F, E, C)

where the edge {E, F } occurs twice. By the degree of a region r, written deg(r), we mean the length of the cycle or closed walk which borders r. We note that each edge either borders two regions or is contained in a region and will occur twice in any walk along the border of the region. Thus we have a theorem for regions which is analogous to Theorem 4.1 for vertices.

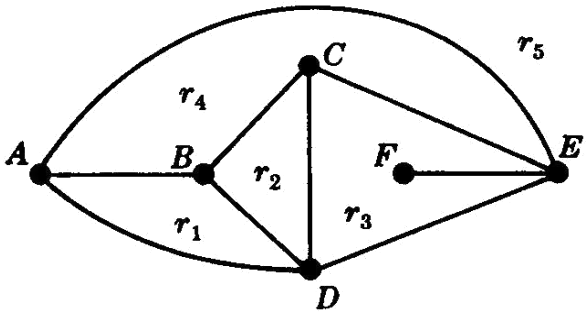


Fig. 4-22

THEOREM 4.7: The sum of the degrees of the regions of a map is equal to twice the number of edges.

The degrees of the regions of Fig. 4-22 are:

$$\deg(r_1) = 3, \quad \deg(r_2) = 3, \quad \deg(r_3) = 5, \quad \deg(r_4) = 4, \quad \deg(r_5) = 3$$

The sum of the degrees is 18, which, as expected, is twice the number of edges.

For notational convenience we shall picture the vertices of a map with dots or small circles, or we shall assume that any intersections of lines or curves in the plane are vertices.

Euler's Formula

Euler gave a formula which connects the number V of vertices, the number E of edges, and the number R of regions of any connected map. Specifically:

THEOREM 4.8 (Euler): $V - E + R = 2$.

(The proof of Theorem 8.8 appears in Problem 4.18.)

Observe that, in Fig. 4-22, V = 6, E = 9, and R = 5; and, as expected by Euler's formula.

$$V - E + R = 6 - 9 + 5 = 2$$

We emphasize that the underlying graph of a map must be connected in order for Euler's formula to hold.

Let G be a connected planar multi-graph with three or more vertices, so G is neither K_1 nor K_2. Let M be a planar representation of G. It is not difficult to see that (1) a region of M can have degree 1 only if its border is a loop, and (2) a region of M can have degree 2 only if its border consists of two multiple edges. Accordingly, if G is a graph, not a multi-graph, then every region of M must have degree 3 or more. This comment together with Euler's formula is used to prove the following result on planar graphs.

THEOREM 4.9: Let G be a connected planar graph with p vertices and q edges, where $p \geq 3$. Then $q \geq 3p - 6$.

Note that the theorem is not true for K_1 where $p = 1$ and $q = 0$, and is not true for K_2 where $p = 2$ and $q - 1$.

Proof : Let r be the number of regions in a planar representation of G. By Euler's formula, $p - q + r = 2$.

Now the sum of the degrees of the regions equals 2q by Theorem 4.7. But each region has degree 3 or more; hence $2q \geq 3r$. Thus $r \geq 2q/3$. Substituting this in Euler's formula gives

$$2 = p - q + r \leq p - q + \frac{2q}{3} or 2 \leq p - \frac{q}{3}$$

Multiplying the inequality by 3 gives $6 \leq 3p - q$ which gives us our result.

Nonplanar Graphs, Kuratowski's Theorem

We give two examples of non-planar graphs. Consider first the utility graph; that is, three houses A_1, A_2, A_3 are to be connected to outlets for water, gas and electricity, B_1, B_2, B_3, as in Fig. 8-23(a). Observe that this is the graph $K_{3,3}$ and it has $p = 6$ vertices and $q = 9$ edges. Suppose the graph is planar. By Euler's formula a planar representation has $r = 5$ regions. Observe that no three vertices are connected to each other; hence the degree of each region must be 4 or more and so the sum of the degrees of the regions must be 20 or more. By Theorem 4.7 the graph must have 10 or more edges. This contradicts the fact that the graph has $q = 9$ edges. Thus the utility graph $K_{3,3}$ is non-planar.

Consider next the star graph in Fig. 4-23(b). This is the complete graph K_5 on $p = 5$ vertices and has $q = 10$ edges. If the graph is planar, then by Theorem 4.9.

$$10 = q \leq 3p - 6 = 15 - 6 = 9$$

which is impossible. Thus K_5 is non-planar.

For many years mathematicians tried to characterize planar and non-planar graphs. This problem was finally solved in 1930 by the Polish mathematician K. Kuratowski. The proof of this result, stated below, lies beyond the scope of this text.

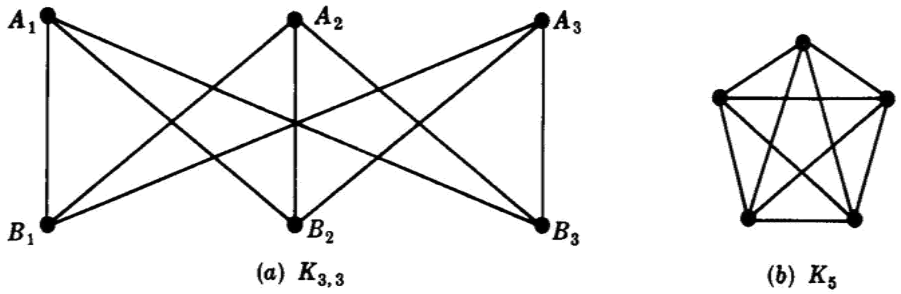


Fig. 4-23

THEOREM 4.10: (Kuratowski) A graph is non-planar if and only if it contains a sub-graph homeomorphic to $K_{3,3}$ or K_5.

4.11 GRAPH COLORINGS

Consider a graph G. A vertex coloring, or simply a coloring of G is an assignment of colors to the vertices of G such that adjacent vertices have different colors. We say that G is n-colorable if there exists a coloring of G which uses n colors. (Since the word "color" is used as a noun, we will try to avoid its use as a verb by saying, for example, "paint" G rather than "color" G when we are assigning colors to the vertices of G.) The minimum number of colors needed to paint G is called the chromatic number of G and is denoted by $\chi(G)$.

Fig. 4-24 gives an algorithm by Welch and Powell for a coloring of a graph G. We emphasize that this algorithm does not always yield a minimal coloring of G.

Algorithm 4. Welch-Powell: The input is a graph G.

Step 1. Order the vertices of G according to decreasing degrees.
Step 2. Assign the first color C_1 to the first vertex and then , in sequential order , assign C_1 to each vertex which is not adjacent to a previous vertex which was assigned C_1.
Step 3. Repeat Step 2 with second color C_2 and the subsequence of non colored

vertices.

Step 4. Repeat Step 3 with a third color C_3, then a fourth color C_4, and so on until all vertices are colored.

Step 5. Exit.

Fig. 4-24

EXAMPLE 4.3

(a) Consider the graph G in Fig. 4-25. We use the Welch-Powell Algorithm 4.4 to obtain a coloring of G. Ordering the vertices according to decreasing degrees yields the following sequence:

$$A_5,\ A_3,\ A_7,\ A_1,\ A_2,\ A_4,\ A_6,\ A_8$$

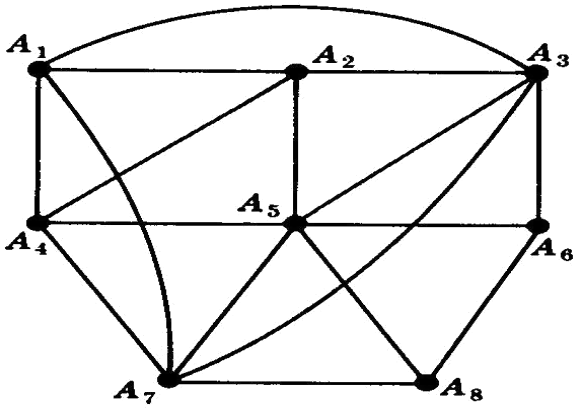


Fig. 4-25

The first color is assigned to vertices A_5 and A_1. The second color is assigned to vertices A_3, A_4, and A_8. The third color is assigned to vertices A_7, A_2, and A_6. All the vertices have been assigned a color, and so G is 3-colorable. Observe that G is not 2-colorable since vertices A_1, A_2, and A_3, which are connected to each other, must be assigned different colors. Accordingly, $\chi(G) = 3$.

(b) Consider the complete graph K_n with n vertices. Since every vertex is adjacent to every other vertex, K_n requires n colors in any coloring. Thus $\chi(K_n) = n$.

There is no simple way to actually determine whether an arbitrary graph is n-colorable. However, the following theorem (proved in Problem 4.19) gives a simple characterization of 2-colorable graphs.

THEOREM 4.11: The following are equivalent for a graph G:

(i) G is 2-colorable.
(ii) G is bipartite.
(iii) Every cycle of G has even length.

There is no limit on the number of colors that may be required for a coloring of an arbitrary graph since, for example, the complete graph K_n requires n colors. However, if we restrict ourselves to planar graphs, regardless of the number of vertices, five colors suffice. Specifically, in Problem 4.20 we prove:

THEOREM 4.12: Any planar graph is 5-colorable.

Actually, since the 1850s mathematicians have conjectured that planar graphs are 4-colorable since every known planar graph is 4-colorable. Kenneth Appel and Wolfgang Haken finally proved this conjecture to be true in 1976. That is:

Four Color Theorem (Appel and Haken): Any planar graph is 4-colorable.

We discuss this theorem in the next subsection.

Dual Maps and the Four Color Theorem

Consider a map M, say the map M in Fig. 4-26(a). In other words, M is a planar representation of a planar multi-graph. Two regions of M are said to be adjacent if they have an edge in common. Thus the regions r_2 and r_5 in Fig. 4-26(a) are adjacent, but the regions r_3 and r_5 are not. By a coloring of M we mean an assignment of a color to each region of M such that adjacent regions have different colors. A map M is n-colorable if there exists a coloring of M which uses n colors. Thus the map M in Fig. 4-26(a) is 3-colorable since the regions can be assigned the following colors:

r_1 red, r_2 white, r_3 red, r_4 white, r_5 red, r_6 blue

Observe the similarity between this discussion on coloring maps and the previous discussion on coloring graphs. In fact, using the concept of the dual map defined below, the coloring of a map can be shown to be equivalent to the vertex coloring of a planar graph.

Consider a map M. In each region of M we choose a point, and if two regions have an edge in common then we connect the corresponding points with a curve through the common edge. These curves can be drawn so that they are non-crossing. Thus we obtain a new map M^*, called the dual of M, such that each vertex of M^* corresponds to exactly one region of M. Figure 4-26(b) shows the dual of the map of Fig. 4-26(a). One can prove that each region of M^* will contain exactly one vertex of M and that each edge of M^* will intersect exactly one edge of M and vice versa. Thus M will be the dual of the map M^*

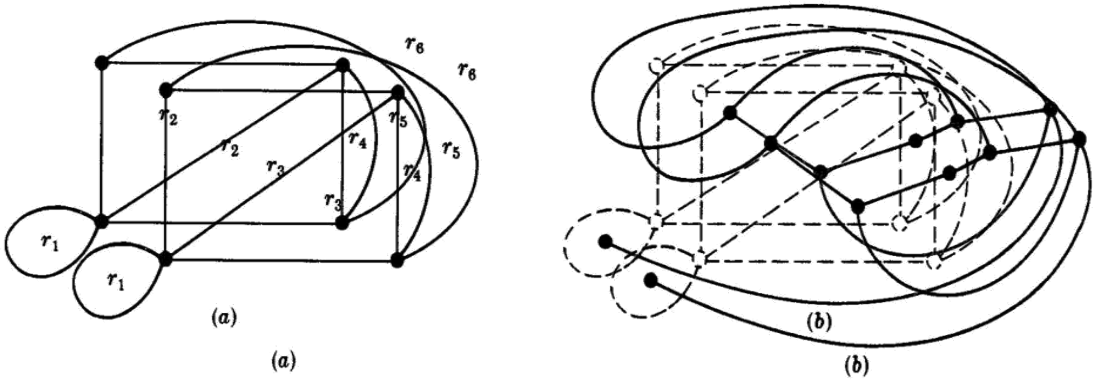


Fig. 4-26

Observe that any coloring of the regions of a map M will correspond to a coloring of the vertices of the dual map M^*. Thus M is n-colorable if and only if the planar graph of the dual map M^* is vertex n-colorable. Thus the above theorem can be restated as follows:

Four Color Theorem (Appel and Haken): If the regions of any map M are colored so that adjacent regions have different colors, then no more than four colors are required.

The proof of the above theorem uses computers in an essential way. Specifically, Appel and Haken first showed that if the four color theorem was false, then there must be a counterexample among one of approximately 2000 different types of planar graphs. They then showed, using the computer, that

none of these types of graphs has such a counterexample. The examination of each different type of graph seems to be beyond the grasp of human beings without the use of a computer. Thus the proof, unlike most proofs in mathematics, is technology dependent; that is, it depended on the development of high-speed computers.

4.12 REPRESENTING GRAPHS IN COMPUTER MEMORY

There are two standard ways of maintaining a graph G in the memory of a computer. One way, called the sequential representation of G, is by means of its adjacency matrix A. The other way, called the linked representation or adjacency structure of G, uses linked lists of neighbors. Matrices are usually used when the graph G is dense, and linked lists are usually used when G is sparse. (A graph G with m vertices and n edges is said to be dense when $m = O(n^2)$ and sparse when $m = O(n)$ or even $O(n \log n)$.)

Regardless of the way one maintains a graph G in memory, the graph G is normally input into the computer by its formal definition, that is, as a collection of vertices and a collection of pairs of vertices (edges).

Adjacency Matrix

Suppose G is a graph with m vertices, and suppose the vertices have been ordered, say, $v_1, v_2, \ldots, v_m$. Then the adjacency matrix $A = [a_{ij}]$ of the graph G is the $m \times m$ matrix defined by

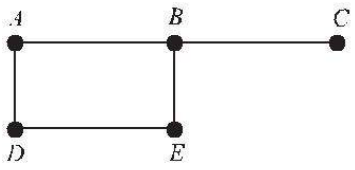
$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Figure 4-27(b) contains the adjacency matrix of the graph G in Fig. 4-27(a) where the vertices are ordered A, B, C, D, E. Observe that each edge $\{v_i , v_j \}$ of G is represented twice, by $a_{ij} = 1$ and $a_{j\,i} = 1$. Thus, in particular, the adjacency matrix is symmetric.

The adjacency matrix A of a graph G does depend on the ordering of the vertices of G, that is, a different ordering of the vertices yields a different adjacency matrix. However, any two such adjacency matrices are closely related

in that one can be obtained from the other by simply interchanging rows and columns. On the other hand, the adjacency matrix does not depend on the order in which the edges (pairs of vertics) are input into the computer.

There are variations of the above representation. If G is a multi-graph, then we usually let a_{ij} denote the number of edges $\{v_i , v_j\}$. Moreover, if G is a weighted graph, then we may let a_{ij} denote the weight of the edge $\{v_i , v_j\}$.



(a)

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	0	1
C	0	1	0	0	0
D	1	0	0	0	1
E	0	1	0	1	0

(b)

Fig. 4-27

Linked Representation of a Graph G

Let G be a graph with m vertices. The representation of G in memory by its adjacency matrix A has a number of major drawbacks. First of all it may be difficult to insert or delete vertices in G. The reason is that the size of A may need to be changed and the vertices may need to be reordered, so there may be many, many changes in the matrix A. Furthermore, suppose the number of edges is O(m) or even O(m log m), that is, suppose G is sparse. Then the matrix A will contain many zeros; hence a great deal of memory space will be wasted. Accordingly, when G is sparse, G is usually represented in memory by some type of linked representation, also called an adjacency structure, which is described below by means of an example.

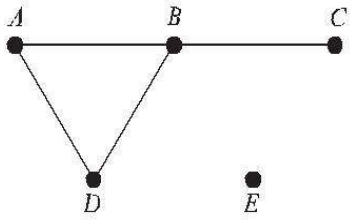
Consider the graph G in Fig. 4-28(a). Observe that G may be equivalently defined by the table in Fig. 4-28(b) which shows each vertex in G followed by its adjacency list, i.e., its list of adjacent vertices (neighbors). Here the symbol denotes an empty list. This table may also be presented in the compact form

G = [A:B, D; B:A, C, D; C:B; D:A, B; E: Ø]

where a colon ":" separates a vertex from its list of neighbors, and a semicolon

";" separates the different lists.

Remark: Observe that each edge of a graph G is represented twice in an adjacency structure; that is, any edge, say {A, B }, is represented by B in the adjacency list of A, and also by A in the adjacency list of B. The graph G in Fig. 8-28(a) has four edges, and so there must be 8 vertices in the adjacency lists. On the other hand, each vertex in an adjacency list corresponds to a unique edge in the graph G.



(a)

Vertex	Adjacency list
A	*B, D*
B	*A, C, D*
C	*B*
D	*A, B*
E	∅

(b)

Fig. 4-28

The linked representation of a graph G, which maintains G in memory by using its adjacency lists, will normally contain two files (or sets of records), one called the Vertex File and the other called the Edge File, as follows.

(a) **Vertex File:** The Vertex File will contain the list of vertices of the graph G usually maintained by an array or by a linked list. Each record of the Vertex File will have the form

VERTEX NEXT-V PTR

Here VERTEX will be the name of the vertex, NEXT-V points to the next vertex in the list of vertices in the Vertex File when the vertices are maintained by a linked list, and **PTR** will point to the first element in the adjacency list of the vertex appearing in the Edge File. The shaded area indicates that there may be other information in the record corresponding to the vertex.

(b) **Edge File:** The Edge File contains the edges of the graph G. Specifically, the

Edge File will contain all the adjacency lists of G where each list is maintained in memory by a linked list. Each record of the Edge File will correspond to a vertex in an adjacency list and hence, indirectly, to an edge of G. The record will usually have the form

Here:

(1) EDGE will be the name of the edge (if it has one).
(2) ADJ points to the location of the vertex in the Vertex File.
(3) NEXT points to the location of the next vertex in the adjacency list.

We emphasize that each edge is represented twice in the Edge File, but each record of the file corresponds to a unique edge. The shaded area indicates that there may be other information in the record corresponding to the edge.

Figure 4-29 shows how the graph G in Fig. 4-28(a) may appear in memory. Here the vertices of G are maintained in memory by a linked list using the variable START to point to the first vertex. (Alternatively, one could use a linear array for the list of vertices, and then NEXT-V would not be required.) Note that the field EDGE is not needed here since the edges have no name. Figure 4-29 also shows, with the arrows, the adjacency list [D, C, A] of the vertex B.

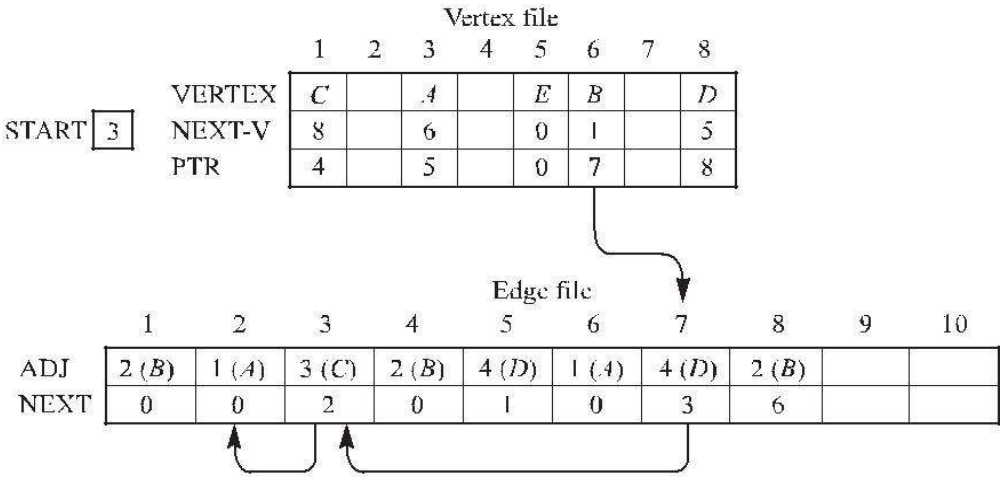
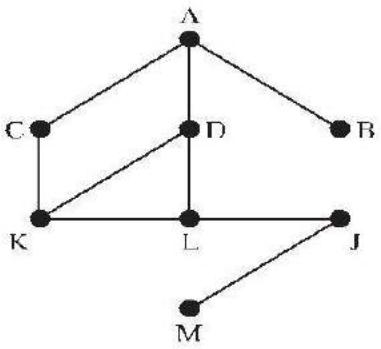


Fig. 4-29

4.13 GRAPH ALGORITHMS

This section discusses two important graph algorithms which systematically examine the vertices and edges of a graph G. One is called a depth-first search (DFS) and the other is called a breadth-first search (BFS). Other graph algorithms will be discussed in the next chapter in connection with directed graphs. Any particular graph algorithm may depend on the way G is maintained in memory. Here we assume G is maintained in memory by its adjacency structure. Our test graph G with its adjacency structure appears in Fig. 8-30 where we assume the vertices are ordered alphabetically.



(a)

Vertex	Adjacency list
A	B, C, D
B	A
C	A, K
D	A, K, L
J	L, M
K	C, D, L
L	D, J, K
M	J

(b)

Fig. 4-30

During the execution of our algorithms, each vertex (node) N of G will be in one of three states, called the status of N , as follows:

STATUS = 1: (Ready state) The initial state of the vertex N .
STATUS = 2: (Waiting state) The vertex N is on a (waiting) list, waiting to be processed.
STATUS = 3: (Processed state) The vertex N has been processed.

The waiting list for the depth-first search (DFS) will be a (modified) STACK (which we write horizontally with the top of STACK on the left), whereas the waiting list for the breadth-first search (BFS) will be a QUEUE.

Depth-first Search

The general idea behind a depth-first search beginning at a starting vertex A is as follows. First we process the starting vertex A. Then we process each vertex N along a path P which begins at A; that is, we process a neighbor of A, then a neighbor of A, and so on. After coming to a "dead end," that is to a vertex with no unprocessed neighbor, we backtrack on the path P until we can continue along another path P. And so on. The backtracking is accomplished by using a STACK to hold the initial vertices of future possible paths. We also need a field STATUS which tells us the current status of any vertex so that no vertex is processed more than once.

The depth-first search (DFS) algorithm appears in Fig. 4-31. The algorithm will process only those vertices which are connected to the starting vertex A, that is, the connected component including A. Suppose one wants to process all the vertices in the graph G. Then the algorithm must be modified so that it begins again with another vertex (which we call B) that is still in the ready state (STATE = 1). This vertex B can be obtained by traversing through the list of vertices.

Remark: The structure STACK in the above algorithm is not technically a stack since, in Step 5(b), we allow a vertex J to be deleted and then inserted in the front of the stack. (Although it is the same vertex J , it usually represents a different edge in the adjacency structure.) If we do not delete J in Step 5(b), then we obtain an alternate form of DFS

Algorithm 5 (Depth-first Search): This algorithm executes a depth-first search on a graph G beginning with a starting vertex A.

Step 1. Initialize all vertices to the ready state (STATUS = 1).

Step 2. Push the starting vertex A onto STACK and change the status of A to the waiting state (STATUS = 2).

Step 3. Repeat Steps 4 and 5 until STACK is empty.

Step 4. Pop the top vertex N of STACK. Process N, and set STATUS $(N) = 3$, the processed state.

Step 5. Examine each neighbor J of N.

(a) If STATUS $(J) = 1$ (ready state), push J onto STACK and reset STATUS $(J) = 2$ (waiting state).
(b) If STATUS $(J) = 2$ (waiting state), delete the previous J from the STACK and push the current J onto STACK.
(c) If STATUS $(J) = 3$ (processed state), ignore the vertex J.
[End of Step 3 loop.]

Step 6. Exit.

Fig. 4-31

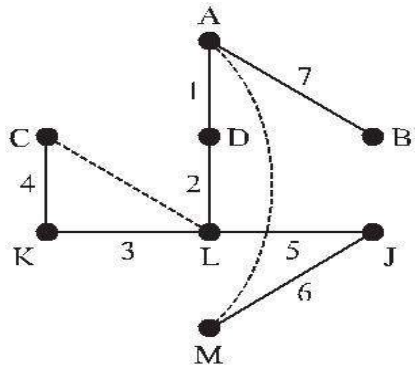
EXAMPLE 4.4 Suppose the DFS Algorithm 8.5 in Fig. 4-31 is applied to the graph in Fig. 4-30. The vertices will be processed in the following order:

A, D, L, K, C, J, M, B

Specifically, Fig. 4-32(a) shows the sequence of vertices being processed and the sequence of waiting lists in STACK. (Note that after vertex A is processed, its neighbors, B, C, and D are added to STACK in the order first B, then C, and finally D; hence D is on the top of the STACK and D is the next vertex to be processed.) Each vertex, excluding A, comes from an adjacency list and hence corresponds to an edge of the graph. These edges form a spanning tree of G which is pictured in Fig. 4-32(b). The numbers indicate the order that the edges are added to the spanning tree, and the dashed lines indicate backtracking.

STACK	Vertex
A	A
D, C, B	D
L, K, C, B	L
K, J, K, C, B	K
C, J, C, B	C
J, B	J
M, B	M
B	B
∅	

(a)



(b)

Fig. 4-32

4.14 TRAVELING-SALESMAN PROBLEM

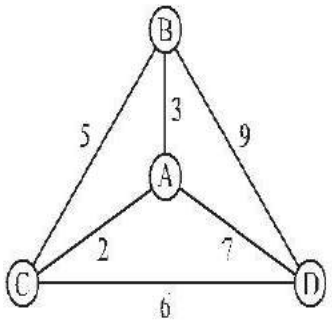
Let G be a complete weighted graph. (We view the vertices of G as cities, and the weighted edges of G as the distances between the cities.) The "traveling-salesman" problem refers to finding a Hamiltonian circuit for G of minimum weight.

First we note the following theorem, proved in Problem 4.33:

THEOREM 4.13: The complete graph K_n with $n \geq 3$ vertices has $H = (n - 1)!/2$ Hamiltonian circuits (where we do not distinguish between a circuit and its reverse).

Consider the complete weighted graph G in Fig. 4-35(a). It has four vertices, A, B, C, D. By Theorem 4.13 it has $H = 3!/2 = 3$ Hamiltonian circuits. Assuming the circuits begin at the vertex A, the following are the three circuits and their weights:

$$|ABCDA| = 3 + 5 + 6 + 7 = 21$$
$$|ACDBA| = 2 + 6 + 9 + 3 = 20$$
$$|ACBDA| = 2 + 5 + 9 + 7 = 23$$



(a)

	P	Q	R	S	T
P		18	22	15	20
Q	18		11	12	22
R	22	11		16	10
S	15	12	16		13
T	20	22	10	13	

(b)

Fig. 4-35

Thus ACDBA with weight 20 is the Hamiltonian circuit of minimum weight.

We solved the "traveling-salesman problem" for the weighted complete graph in Fig. 4-35(a) by list-ing and finding the weights of its three possible Hamiltonian circuits. However, for a graph with many ver-tices, this may be impractical or even impossible. For example, a complete graph with 15 vertices has over 40 million Hamiltonian circuits. Accordingly, for circuits with many vertices, a strategy of some kind is needed to solve or give an approximate solution to the traveling-salesman problem. We discuss one of the simplest algorithms here.

SOLVED PROBLEMS

GRAPH TERMINOLOGY

4.1. Consider the graph G in Fig. 4-36(a).

(a) Describe G formally, that is, find the set V (G) of vertices of G and the set E(G) of edges of G. (b) Find the degree of each vertex and verify Theorem 4.1 for this graph.

(a) There are five vertices so V (G) = {A, B, C, D, E}. There are seven pairs {x, y} of vertices where the vertex x is connected with the vertex y, hence

$$E(G) = [\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, E\}, \{C, D\}, \{C, E\}]$$

(b) The degree of a vertex is equal to the number of edges to which it belongs; e.g., deg(A) = 3 since A belongs to the three edges {A, B}, {A, C}, {A, D}. Similarly,

$$\deg(B) = 3, \deg(C) = 4, \deg(D) = 2, \deg(E) = 2$$

The sum of the degrees is 3 + 3 + 4 + 2 + 2 = 14 which does equal twice the number of edges.

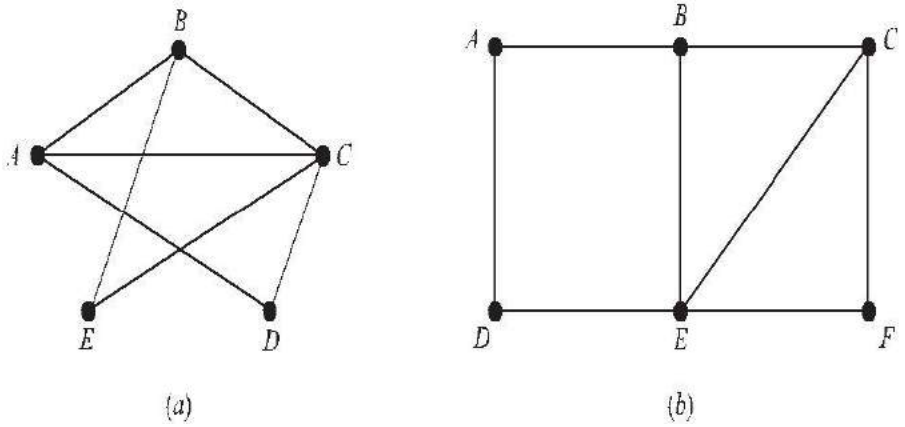


Fig. 4-36

4.2. Consider the graph G in Fig. 4-36(b). Find:

(a) all simple paths from A to F ; (d) diam(G), the diameter of G;
(b) all trails from A to F ; (e) all cycles which include vertex A;
© d(A, F), the distance from A to F ; (f) all cycles in G.

(a) simple path from A to F is a path such that no vertex, and hence no edge, is repeated. There are seven such paths, four beginning with the edges {A, B} and three beginning with the ege {A, D}:

(A, B, C, F), (A, B, C, E, F), (A, B, E, F), (A, B, E, C, F), (A, D, E, F), (A, D, E, B, C, F), (A, D, E, C, F).

(b)A trail from A to F is a path such that no edge is repeated. There are nine such trails, the seven simple paths from (a) together with

(A, D, E, B, C, E, F) and (A, D, E, C, B, E, F).

(c) There is a path, e.g., (A, B, C, F), from A to F of length 3 and no shorter path from A to F ; hence d(A, F) = 3.
(d) The distance between any two vertices is not greater than 3, and the distance from A to F is 3; hence diam(G) = 3.
(e) A cycle is a closed path in which no vertex is repeated (except the first and last). There are three cycles which include vertex A:

(A, B, E, D, A), (A, B, C, E, D, A), (A, B, C, F, E, D, A)

(f) There are six cycles in G; the three in (e) and (B, C, E, B), (C,F E,C), (B,C,F,E, B).

4.3 Consider the multi-graphs in Fig. 4-37.

(a) Which of them are connected? If a graph is not connected, find its connected components.
(b) Which are cycle-free (without cycles)?
(c) Which are loop-free (without loops)?
(d) Which are (simple) graphs?

(a) Only (1) and (3) are connected, (2) is disconnected; its connected components are {A, D, E} and {B, C}. (4) is disconnected; its connected components are {A, B, E} and {C, D}.
(b) Only (1) and (4) are cycle-free. (2) has the cycle (A, D, E, A), and (3) has the cycle (A, B, E, A).
(c) Only (4) has a loop which is {B, B}.
(d) Only (1) and (2) are graphs. Multi-graph (3) has multiple edges {A, E} and {A, E}; and (4) has both multiple edges {C, D} and {C, D} and a loop {B, B}.

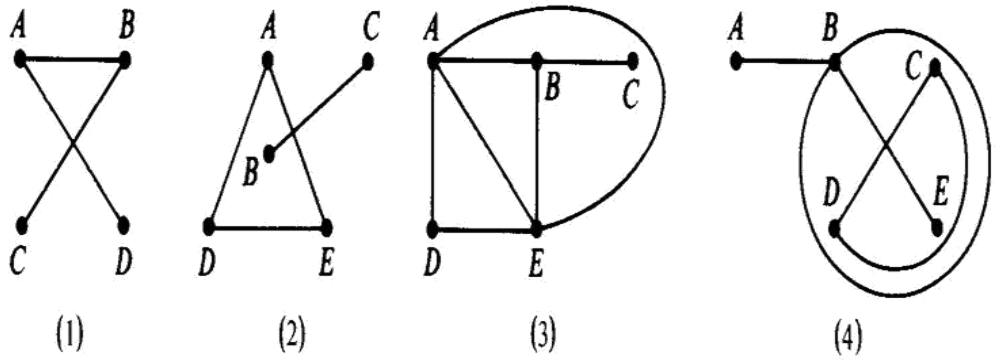


Fig.4-37

4.4. Let G be the graph in Fig. 4-38(a). Find:

(a) all simple paths from A to C;
(b) all cycles;
(c) subgraph H generated by V'' = {B, C, X, Y };
(d) G – Y ;
(e) all cut points;
(f) all bridges.

(a) There are two simple paths from A to C: (A, X, Y, C) and (A, X, B, Y, C).
(b) There is only one cycle: (B, X, Y, B).
(c)As pictured in Fig. 4-38(b), H consists of the vertices V and the set E of all edges whose endpoints bel to V' , that is, E = [{B, X}, {X, Y }, {B, Y }, {C, Y }].
(d)Delete vertex Y from G and all edges which contain Y to obtain the graph G − Y in Fig. 4-38(c). (Note Y is a cutpoint since G − Y is disconnected.)
(e) Vertices A, X, and Y are cut points.
(f)An edge e is a bridge if G − e is disconnected. Thus there are three bridges: {A, Z}, {A, X}, and {C, Y }.

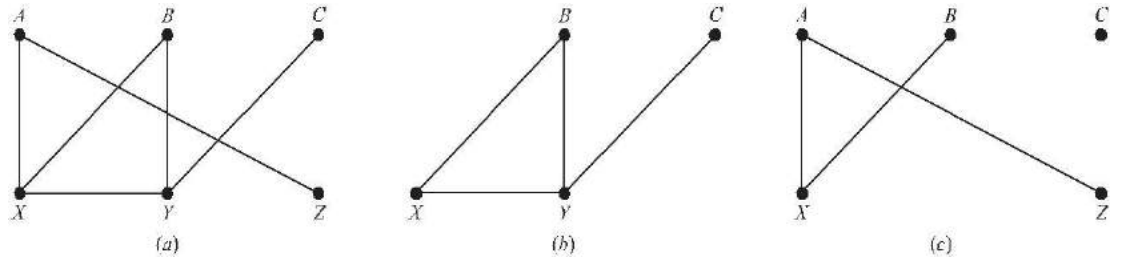


Fig. 4-38

4.5 Consider the graph G in Fig. 4-36(b). Find the sub-graphs obtained when each vertex is deleted. Does G have any cut points?

When we delete a vertex from G, we also have to delete all edges which contain the vertex. The six graphs obtained by deleting each of the vertices of G are shown in Fig. 8-39. All six graphs are connected; hence no vertex is a cut point.

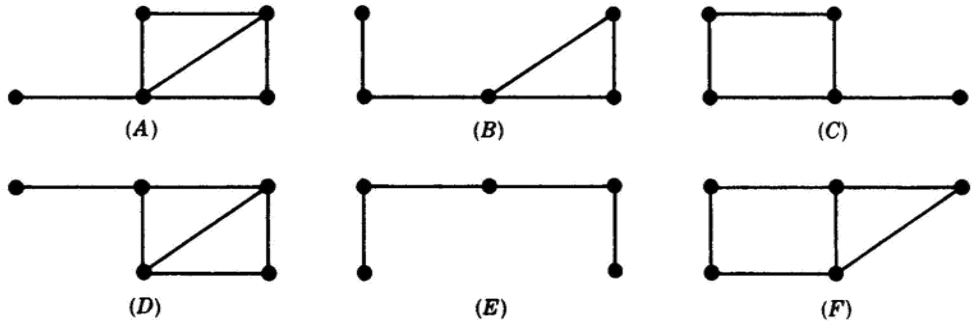


Fig. 4-39

4.6 Show that the six graphs obtained in Problem 4.5 are distinct, that is, no two of them are isomorphic. Also show that (B) and (C) are homeomorphic.

The degrees of the five vertices of any graph cannot be paired off with the degrees of any other graph, except for (B) and (C). Hence none of the graphs is isomorphic except possibly (B) and (C).

However if we delete the vertex of degree 3 in (B) and (C), we obtain distinct subgraphs. Thus (B) and (C) are also nonisomorphic; hence all six graphs are distinct. However, (B) and (C) are homeomorphic since they can be obtained from isomorphic graphs by adding appropriate vertices.

TRAVERSABLE GRAPHS, EULER AND HAMILTONIAN CIRCUITS

4.7 Consider each graph in Fig. 4-40. Which of them are traversable, that is, have Euler paths? Which are Eulerian, that is, have an Euler circuit? For those that do not, explain why.

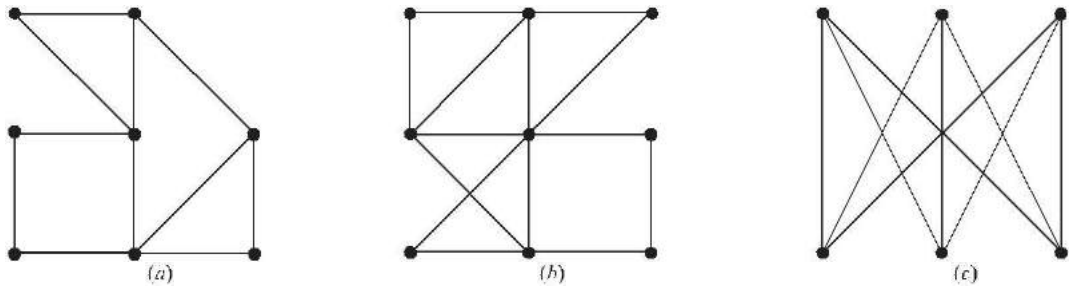


Fig. 4-40

G is traversable (has an Euler path) if only 0 or 2 vertices have odd degree, and G is Eulerian (has an Euler circuit) if all vertices are of even degree (Theorem 4.3).

(a) Traversable, since there are two odd vertices. The traversable path must begin at one of the odd vertices and will end at the other.
(b) Traversable, since all vertices are even. Thus G has an Euler circuit.
(c) Since six vertices have odd degrees, G is not traversable.

4.8. Which of the graphs in Fig. 4-40 have a Hamiltonian circuit? If not, why not?

Graphs (a) and (c) have Hamiltonian circuits. (The reader should be able to easily find one of them.) However, graph (b) has no Hamiltonian circuit. For if α is a Hamiltonian circuit, then α must connect the middle vertex with the lower right vertex, then proceed along the bottom row to the lower right vertex, then vertically to the middle right, but then is forced back to the central vertex before visiting the remaining vertices.

4.9 Prove Theorem 4.3 (Euler): A finite connected graph G is Eulerian if and only if each vertex has even degree.

Suppose G is Eulerian and T is a closed Eulerian trail. For any vertex v of G, the trail T enters and leaves v the same number of times without repeating

any edge. Hence v has even degree.

Suppose conversely that each vertex of G has even degree. We construct an Eulerian trail. We begin a trail T_1 at any edge e. We extend T_1 by adding one edge after the other. If T_1 is not closed at any step, say, T_1 begins at u but ends at $v \neq u$, then only an odd number of the edges incident on v appear in T_1 ; hence we can extend T_1 by another edge incident on v. Thus we can continue to extend T_1 until T_1 returns to its initial vertex u, i.e., until T_1 is closed. If T_1 includes all the edges of G, then T_1 is our Eulerian trail.

Suppose T_1 does not include all edges of G. Consider the graph H obtained by deleting all edges of T_1 from G. H may not be connected, but each vertex of H has even degree since T_1 contains an even number of the edges incident on any vertex. Since G is connected, there is an edge e of H which has an endpoint u in T_1 . We construct a trail T_2 in H beginning at u and using e . Since all vertices in H have even degree, we can continue to extend T_2 in H until T_2 returns to u as pictured in Fig. 4-41. We can clearly put T_1 and T_2 together to form a larger closed trail in G. We continue this process until all the edges of G are used. We finally obtain an Eulerian trail, and so G is Eulerian.

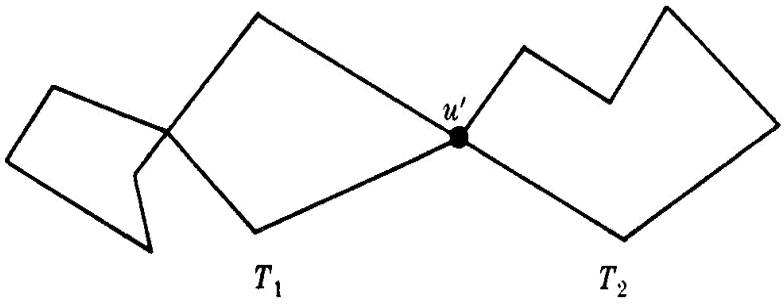


Fig. 4-41

TREES, SPANNING TREES

4.10 Draw all trees with exactly six vertices.

There are six such trees which are exhibited in Fig. 4-42. The first tree has diameter 5, the next two diameter 4, the next two diameter 3, and the last one diameter 2. Any other tree with 6 nodes is isormorphic to one of these trees.

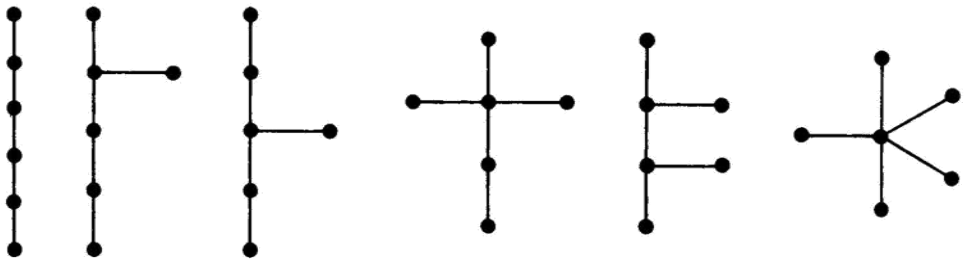


Fig. 4-42

4.11 Find all spanning trees of the graph G shown in Fig. 4-43(a).

There are eight such spanning trees as shown in Fig. 4-43(b). Each spanning tree must have 4 − 1 = 3 edges since G has four vertices. Thus each tree can be obtained by deleting two of the five edges of G. This can be done in 10 ways,

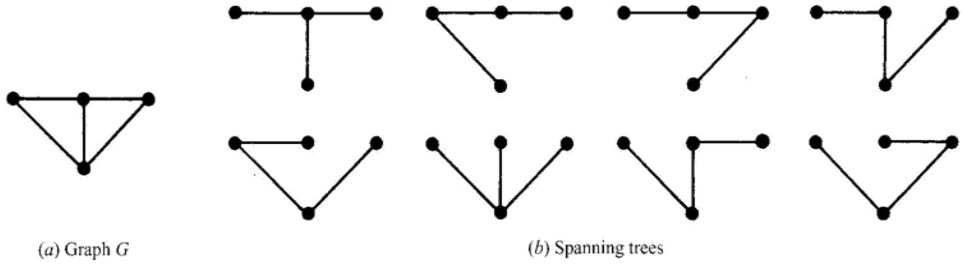


Fig. 4-43

except that two of the ways lead to disconnected graphs. Hence the above eight spanning trees are all the spanning trees of G.

4.12. Find a minimal spanning tree T for the weighted graph G in Fig. 4-44(a).

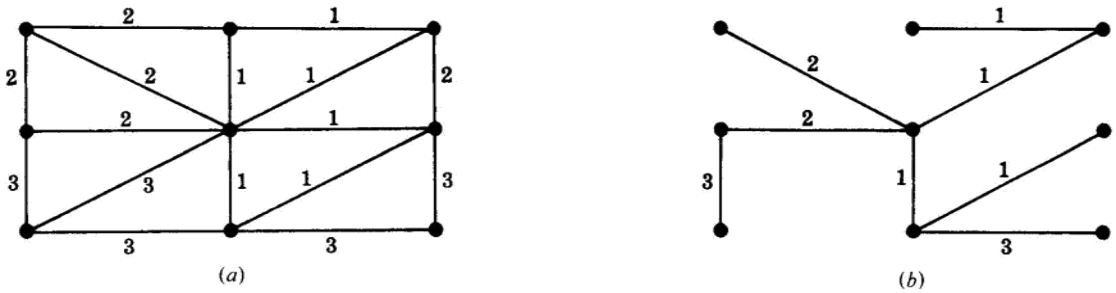


Fig. 4-44

Since G has n = 9 vertices, T must have n − 1 = 8 edges. Apply Algorithm 8.2, that is, keep deleting edges with maximum length and without disconnecting the graph until only n−1 = 8 edges remain. Alternatively, apply Algorithm 8.3, that is, beginning with the nine vertices, keep adding edges with minimum length and without forming any circle until n − 1 = 8 edges are added. Both methods give a minimum spanning tree such as that shown in Fig. 4-44(b).

4.13. Let G be a graph with more than one vertex. Prove the following are equivalent.

(i) G is a tree.
(ii) Each pair of vertices is connected by exactly one simple path.
(iii) G is connected; but G − e is disconnected for any edge e of G.
(iv) G is cycle-free, but if any edge is added to G then the resulting graph has exactly one cycle.

(i) implies (ii) Let u and v be two vertices in G. Since G is a tree, G is connected so there is at least one path between u and v. By Problem 4.37 there can only be one simple path between u and v, otherwise G will contain a cycle.

(ii) implies (iii) Suppose we delete an edge e = {u, v} from G. Note e is a path from u to v. Suppose the resulting graph G − e has a path P from u to v. Then P and e are two distinct paths from u to v, which contradicts the hypothesis. Thus there is no path between u and v in G − e, so G − e is disconnected.

G implies (iv) Suppose G contains a cycle C which contains an edge e = {u, v}. By hypothesis, G is connected but = G − e is disconnected, with u and v belonging to different components of G (Problem 4.41) This contradicts the fact that u and v are connected by the path P = C − e which lies in G . Hence G is cycle-free. Now let x and y be vertices of G and let H be the graph obtained by adjoining the edge e = {x, y} to G. Since G is connected, there is a path P from x to y in G; hence C = P e forms a cycle in H . Suppose H contains another cycle C . Since is cycle-free, C must contain the edge e, say C = P e. Then P and P are two simple

paths in G from x to y. (Fig. 4-45) By Problem 4.37, G contains a cycle, which contradicts the fact that G is cycle-free. Hence H contains only one cycle.

(iv) implies (i) Since adding any edge e = {x, y} to G produces a cycle, the vertices x and y must already be connected in G. Hence G is connected and by hypothesis G is cycle-free; that is, G is a tree.

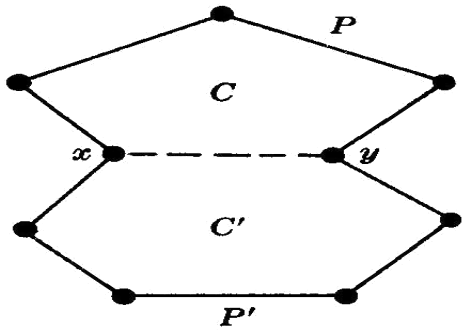


Fig. 4-45

4.14 Prove Theorem 4.6: Let G be a finite graph with n ≥ 1 vertices. Then the following are equivalent.

(i) G is a tree, (ii) G is a cycle-free and has n − 1 edges, (iii) G is connected and has n − 1 edges.

The proof is by induction on n. The theorem is certainly true for the graph with only one vertex and hence no edges. That is, the theorem holds for n = 1. We now assume that n > 1 and that the theorem holds for graphs with less than n vertices.

(i) implies (ii) Suppose G is a tree. Then G is cycle-free, so we only need to show that G has n − 1 edges. By Problem 4.38, G has a vertex of degree 1. Deleting this vertex and its edge, we obtain a tree T which has n − 1 vertices. The theorem holds for T , so T has n − 2 edges. Hence G has n − 1 edges.

(ii) implies (iii) Suppose G is cycle-free and has n − 1 edges. We only need

show that G is connected. Suppose G is disconnected and has k components, T_1 , . . . , T_k , which are trees since each is connected and cycle-free. Say T_i has n_i vertices. Note $n_i < n$. Hence the theorem holds for T_i , so T_i has $n_i - 1$ edges. Thus

$$n = n_1 + n_2 + \cdots + n_k$$

and

$$n - 1 = (n_1 - 1) + (n_2 - 1) + \cdots + (n_k - 1) = n_1 + n_2 + \cdots + n_k - k = n - k$$

Hence $k = 1$. But this contradicts the assumption that G is disconnected and has $k > 1$ components. Hence G is connected.

(iii) implies (i) Suppose G is connected and has $n - 1$ edges. We only need to show that G is cycle-free. Suppose G has a cycle containing an edge e. Deleting e we obtain the graph $H = G - e$ which is also connected. But H has n vertices and $n - 2$ edges, and this contradicts Problem 4.39. Thus G is cycle-free and hence is a tree.

PLANAR GRAPHS

4.15. Draw a planar representation, if possible, of the graphs (a), (b), and (c) in Fig. 4-46

(a) Redrawing the positions of B and E, we get a planar representation of the graph as in Fig. **4-47**(a).
(b) This is not the star graph K_5 . This has a planar representation as in Fig. 4-47(b).
(c) This graph is non-planar. The utility graph $K_{3,3}$ is a sub-graph as shown in Fig. 4-47(c) where we have redrawn the positions of C and F .

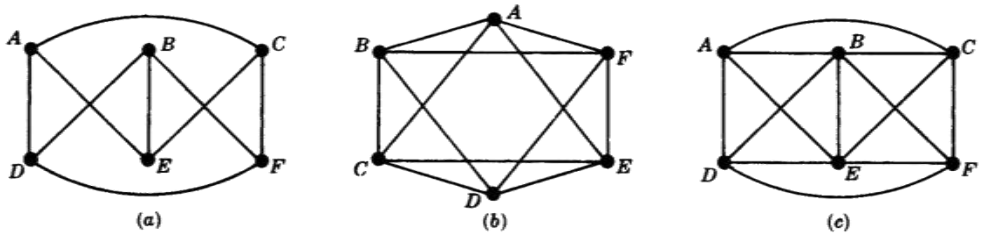


Fig.4-46

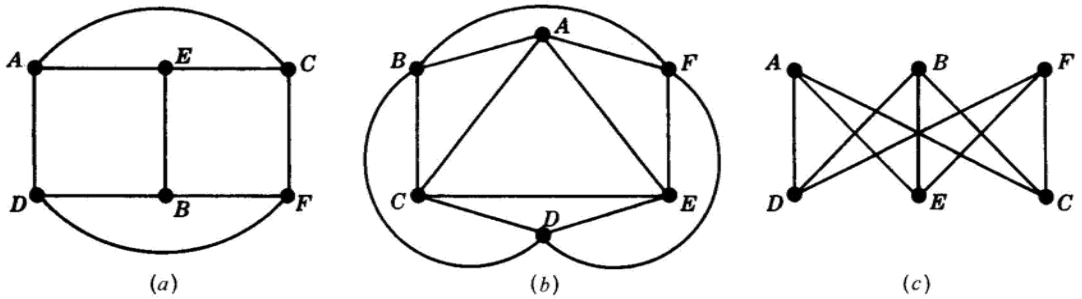


Fig. 4-47

4.16 Count the number V of vertices, the number E of edges, and the number R of regions of each map in Fig. 4-48; and verify Euler's formula. Also find the degree d of the outside region.

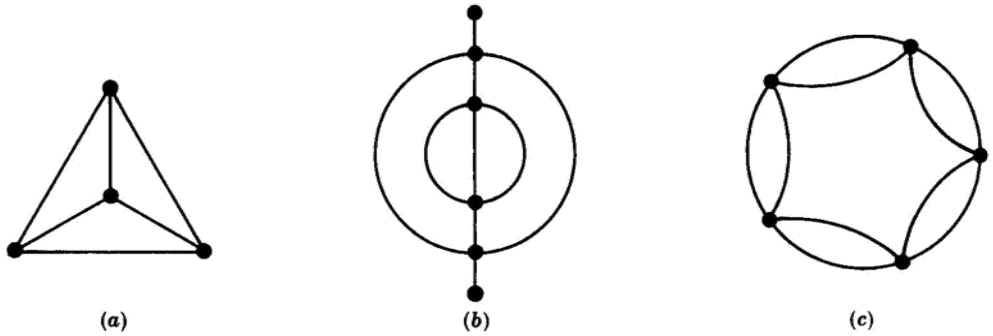


Fig.4-48

(a) V = 4, E = 6, R = 4. Hence V − E + R = 4 − 6 + 4 = 2. Also d = 3.

(b) V = 6, E = 9, R = 5; so V − E + R = 6 − 9 + 5 = 2. Here d = 6 since two edges are counted twice.

(c) V = 5, E = 10, R = 7. Hence V − E + R = 5 − 10 + 7 = 2. Here d = 5.

4.17. Find the minimum number n of colors required to paint each map in Fig. 4-48.

(a) n = 4; (b) n = 3; (c) n = 2.

4.18. Prove Theorem 4-8 (Euler): V − E + R = 2.

Suppose the connected map M consists of a single vertex P as in Fig. 4-49(a). Then V = 1, E = 0, and R = 1. Hence V − E + R = 2. Otherwise M can be built up from a single vertex by the following two constructions:

(1) Add a new vertex Q_2 and connect it to an existing vertex Q_1 by an edge which does not cross any existing edge as in Fig. 4-49(b).

(2) Connect two existing vertices Q_1 and Q_2 by an edge e which does not cross any existing edge as in Fig. 4-49(c).

Neither operation changes the value of V − E + R. Hence M has the same value of V − E + R as the map consisting of a single vertex, that is, V − E + R = 2. Thus the theorem is proved.

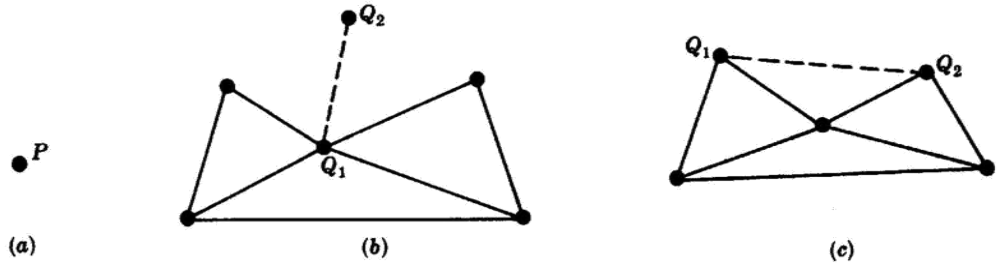


Fig. 4-49

4.19 Prove Theorem 4.11: The following are equivalent for a graph G: (i) G is 2-colorable. (ii) G is bipartite. (iii) Every cycle of G has even length.

(i) implies (ii). Suppose G is 2-colorable. Let M be the set of vertices painted the first color, and let N be the set of vertices painted the second color. Then M and N form a bipartite partition of the vertices of G since neither the vertices of M nor the vertices of N can be adjacent to each other since they are of the same color.

(ii) implies (iii). Suppose G is bipartite and M and N form a bipartite partition of the vertices of G. If a cycle begins at a vertex u of, say, M, then it will go to a vertex of N , and then to a vertex of M, and then to N and so on. Hence when the cycle returns to u it must be of even length. That is, every cycle of G will have even length.

(iii) implies (i). Lastly, suppose every cycle of G has even length. We pick a vertex in each connected component and paint it the first color, say red. We then successively paint all the vertices as follows: If a vertex is

painted red, then any vertex adjacent to it will be painted the second color, say blue. If a vertex is painted blue, then any vertex adjacent to it will be painted red. Since every cycle has even length, no adjacent vertices will be painted the same color. Hence G is 2-colorable, and the theorem is proved.

4.20 Prove Theorem 4.12: A planar graph G is 5-colorable.

The proof is by induction on the number p of vertices of G. If $p \leq 5$, then the theorem obviously holds. Suppose $p > 5$, and the theorem holds for graphs with less than p vertices. By the preceding problem, G has a vertex v such that $\deg(v) \leq 5$. By induction, the sub-graph $G - v$ is 5-colorable. Assume one such coloring. If the vertices adjacent to v use less than the five colors, than we simply paint v with one of the remaining colors and obtain a 5-coloring of G. We are still left with the case that v is adjacent to five vertices which are painted different colors. Say the vertices, moving counterclockwise about v, are $v_1, \ldots, v_5$ and are painted respectively by the colors $c_1, \ldots, c_5$. (Fig. 4-50(a).)

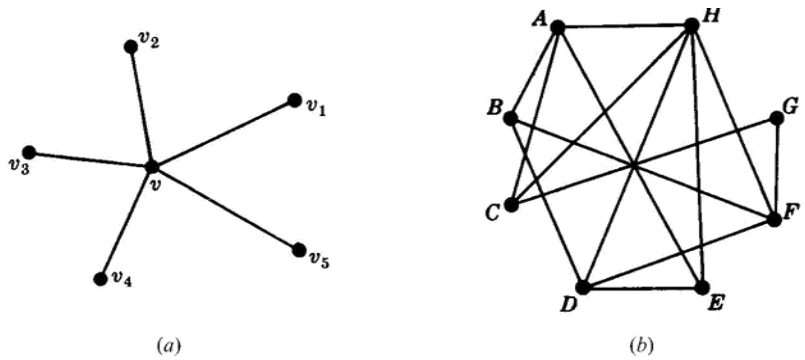


Fig. 4-50

Consider now the sub-graph H of G generated by the vertices painted c_1 and c_3. Note H includes v_1 and v_3. If v_1 and v_3 belong to different components of H , then we can interchange the colors c_1 and c_3 in the component containing v_1 without destroying the coloring of $G - v$. Then v_1 and v_3 are painted by c_3 , c_1 can be chosen to paint v, and we have a 5-coloring of G. On the other hand, suppose v_1 and v_3 are in the same component of H . Then there is a path P from v_1 to v_3 whose vertices are

painted either c_1 or c_3 . The path P together with the edges {v, v_1 } and {v, v_3 } form a cycle C which encloses either v_2 or v_4 . Consider now the subgraph K generated by the vertices painted c_3 or c_4 . Since C encloses v_2 or v_4 , but not both, the vertices v_2 and v_4 belong to different components of K. Thus we can interchange the colors c_2 and c_4 in the component containing v_2 without destroying the coloring of G − v. Then v_2 and v_4 are painted by c_4 , and we can choose c_2 to paint v and obtain a 5-coloring of G. Thus G is 5-colorable and the theorem is proved.

4.21. Use the Welch-Powell Algorithm .4 (Fig. 4-24) to paint the graph in Fig. 4-50(b).

First order the vertices according to decreasing degrees to obtain the sequence

H,A, D, F, B, C, E, G

Proceeding sequentially, we use the first color to paint the vertices H , B, and then G. (We cannot paint A, D, or F the first color since each is connected to H , and we cannot paint C or E the first color since each is connected to either H or B.) Proceeding sequentially with the unpainted vertices, we use the second color to paint the vertices A and D. The remaining vertices F , C, and E can be painted with the third color. Thus the chromatic number n cannot be greater than 3. However, in any coloring, H , D, and E must be painted different colors since they are connected to each other. Hence n = 3.

4.22 Let G be a finite connected planar graph with at least three vertices. Show that G has at least one vertex of degree 5 or less.

Let p be the number of vertices and q the number of edges of G, and suppose deg(u) ≥ 6 for each vertex u of G. But 2q equals the sum of the degrees of the vertices of G (Theorem 8.1); so 2q ≥ 6p. Therefore

$$q \geq 3p > 3p - 6$$

This contradicts Theorem 4.9. Thus some vertex of G has degree 5 or less.

SEQUENTIAL REPRESENTATION OF GRAPHS

4.23. Find the adjacency matrix A = [a_{ij}] of each graph G in Fig. 4-51.

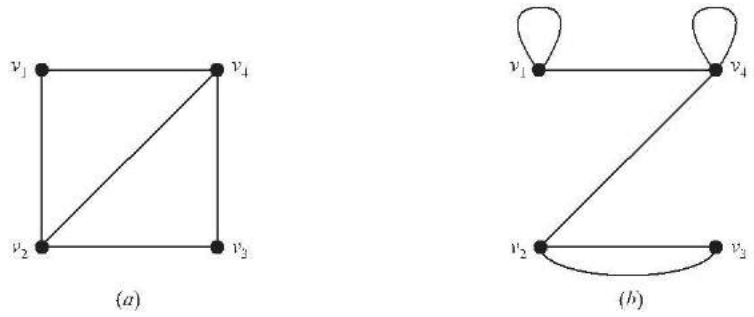


Fig.4-51

Set $a_{ij} = n$ if there are n edges $[v_i,v_j]$ and $a_{ij} = 0$ otherwise . Hence

$$\text{(a)} \quad A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}; \quad \text{(b)} \quad A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 2 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

(Since (a) has no multiple edges and no loops, the entries in A are either 0 or 1, and are 0 on the diagonal.)

4.24. Draw the graph G corresponding to each adjacency matrix:

$$\text{(a) } A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}; \quad \text{(b) } A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 2 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

(a) Since A is a 5-square matrix, G has five vertices, say, v_1 , v_2 . . . , v_5 . Draw an edge from v_i to v_j when $a_{ij} = 1$. The graph appears in Fig. 4-52(a).

(b) Since A is a 4-square matrix, G has four vertices, say, v_1 , . . . , v_4 . Draw

n edges from v_i to v_j when $a_{ij} = n$. Also, draw n loops at v_i when $a_i = n$. The graph appears in Fig.4-52(b).

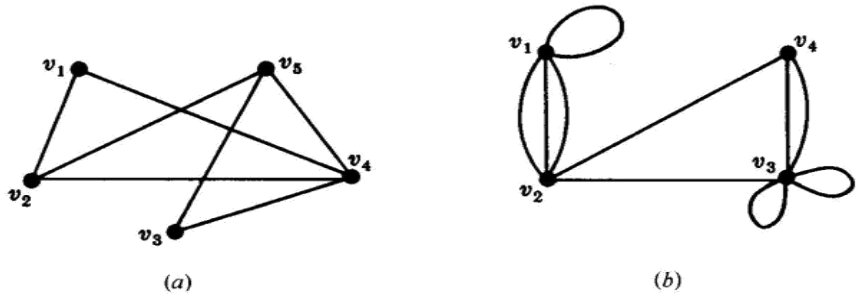


Fig. 4-52

4.25. Find the weight matrix $W = [w_{ij}]$ of the weighted graph G in Fig. 8-53(a) where the vertices are stored in the array DATA as follows: DATA: A, B, C, X, Y .

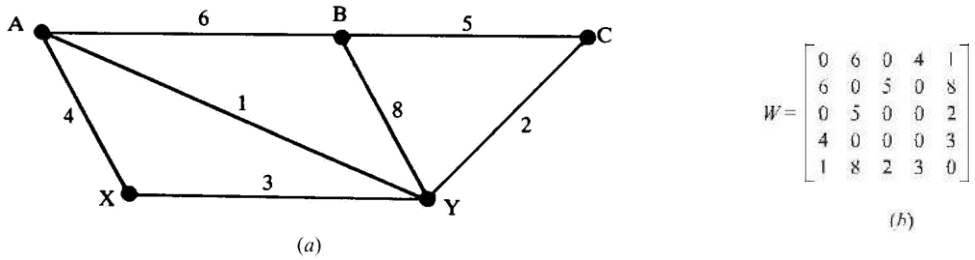


Fig. 4-53

The vertices are numbered according to the way they are stored in the array DATA; so v_1 = A, v_2 = B, . . . , v_5 = Y . Then set $W_{ij} = w$, where w is the weight of the edge from v_i to v_j . This yields the matrix W in Fig. 4-53(b).

LINKED REPRESENTATION OF GRAPHS

4.26 A graph G with vertices A, B, . . . , F is stored in memory using a linked representation with a vertex file and an edge file as in Fig. 4-54.

(a) List the vertices in the order they appear in memory.
(b) Find the adjacency list adj(v) of each vertex v of G.

(a) Since START = 4, the list begins with the vertex D. The NEXT-V tells us to go to 1(B), then 3(F), then 5(A), then 8(E), and then 7(C); that is, D, B, F, A, E, C

(b) Here adj(D) = [5(A), 1(B), 8(E)]. Specifically, PTR [4(D)] = 7 and ADJ[7] = 5(A) tells us that adj(D) begins with A. Then NEXT[7] = 3 and ADJ[3] = 1(B) tells us that B is the next vertex in adj(D). Then NEXT [3] = 10 and ADJ[10] = 8(E) tells us that E in the next vertex in adj(D). However, NEXT[10] = 0 tells us that there are no more neighbors of D. Similarly.

adj(B) = [A, D], adj(F) = [E], adj(A) = [B, D], adj(E) = [C, D, F], adj(C) = [E]

In other words, the following is the adjacency structure of G:

G = [A:B, D; B:A, D; C:E; D:A, B, E; E:C, D, F ; F :E]

Vertex file

		1	2	3	4	5	6	7	8
START 4	VERTEX	*B*		*F*	*D*	*A*		*C*	*E*
	NEXT-V	3		5	1	8		0	7
	PTR	9		4	7	6		5	12

Edge file

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ADJ	4	4	1	8	8	1	5	3	5	8	4	7		
NEXT	8	0	10	0	0	2	3	0	11	0	0	1		

Fig. 4-54

4.27 Draw the diagram of the graph G whose linked representation appears in Fig. 4-54.

Use the vertex list obtained in Problem 8.26(a) and the adjacency lists obtained in Problem 4.26(b) to draw the graph G in Fig. 8-55.

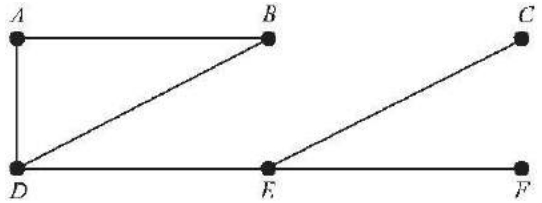


Fig. 4-55

4.28 Exhibit the adjacency structure (AS) of the graph G in: (a) Fig. 4-56(a), (b) Fig. 4-56(b).

The adjacency structure of a graph G consists of the adjacency lists of the vertices where we use a colon":" to separate a vertex from its adjacency list, and a semicolon ";" to separate the different lists. Thus:

(a) G = [A:B, C, D; B:A, C, E; C:A, B, D, E; D:A, C; E:B, C]
(b) G = [A:B, D; B:A, C, E; C:B, E, F ; D:A, E; E:B, C, D, F ; F :C, E]

GRAPH ALGORITHMS

4.29 Consider the graph G in Fig. 4-56(a) (where the vertices are ordered alphabetically).

(a) Find the adjacency structure of G.

(b) Find the order in which the vertices of G are processed using a DFS (depth-first search) algorithm beginning at vertex A.

(a) List the neighbors of each vertex as follows:

G = [A:B, C, D; B:A, J ; C:A; D:A, K; J :B, K, M; K:D, J, L; L:K, M; M:J, L]

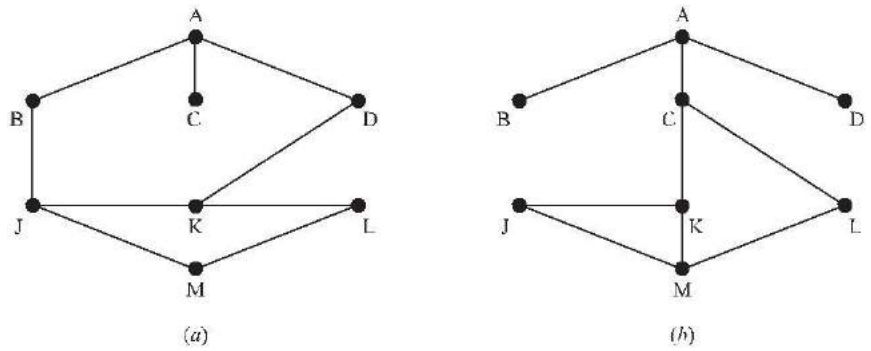


Fig. 4-56

(b) During the DFS algorithm, the first vertex N in STACK is processed and the neighbors of N (which have not been previously processed) are then pushed onto STACK. Initially, the beginning vertex A is pushed onto STACK. The

following shows the sequence of waiting lists in STACK and the vertices being processed:

STACK	A	DCB	KCB	LJ CB	MJ CB	J CB	CB	B	
Vertex	A	D	K	L	M	J	C	B	

In other words the vertices are processed in the order: A, D, K, L, M, J , C, B.

4.30. Repeat Problem 4.29 for the graph G in Fig. 4-56(b).

(a) List the neighbors of each vertex as follows:

G = [A:B, C, D; B:A; C:A, K, L; D:A; J :K, M; K:C, J, M; L:C, M; M:J, K, L]

(b) The following shows the sequence of waiting lists in STACK and the vertices being processed:

STACK	A	DCB	CB	LKB	MKB	KJ B	J B	B
Vertex	A	D	C	L	M	K	J	B

In other words the vertices are processed in the order: A, D, C, L, M, K, J, B.

4.31 Beginning at vertex A and using a BFS (breadth-first search) algorithm, find the order the vertices are processed for the graph G: (a) in Fig. 4-56(a), (b) in Fig. 4-56(b).

(a) The adjacency structure of G appears in Problem 8.29. During the BFS algorithm, the first vertex N in QUEUE is processed and the neighbors of N (which have not appeared previously) are then added onto QUEUE. Initially, the beginning vertex A is assigned to QUEUE. The following shows the sequence of waiting lists in QUEUE and the vertices being

processed:

QUEUE								
E	A	DCB	J DC	J D	KJ	MK	LM	L
Vertex	A	B	C	D	J	K	M	L

In other words the vertices are processed in the order: A, B, C, D, J , K, M, L.

(b) The adjacency structure of G appears in Problem 4.30. The following shows the sequence of waiting lists in QUEUE and the vertices being processed:

QUEUE								
E	A	DCB	DC	LKD	LK	MJ L	MJ	M
Vertex	A	B	C	D	K	L	J	M

In other words the vertices are processed in the order: A, B, C, D, K, L, J , M.

TRAVELING–SALESMAN PROBLEM

4.32 Apply the nearest-neighbor algorithm to the complete weighted graph G in Fig. 8-57 beginning at: (a) vertex A; (b) vertex D.

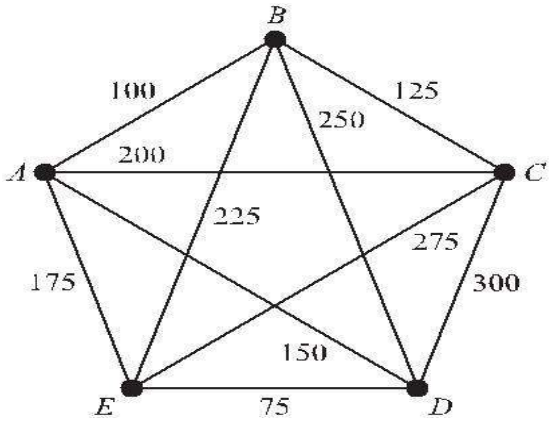


Fig. 4-57

(a) Starting at A, the closest vertex is B with distance 100; from B, the closest is C with distance 125; and from C the closest is E with distance 275. From E we must go to D with distance 75; and finally from D we must go back to A with distance 150. Accordingly, the nearest-neighbor algorithm beginning at A yields the following weighted Hamiltonian circuit:

$$|ABCEDA| = 100 + 125 + 275 + 75 + 150 = 725$$

(b) Starting at D, we must go to E, then A, then B then C and finally back to D. Accordingly, the nearest-neighbor algorithm beginning at D yields the following weighted Hamiltonian circuit:

$$|DEABCD| = 75 + 175 + 100 + 125 + 300 = 775$$

4.33 Prove Theorem 4.13. The complete graph K_n with $n \geq 3$ vertices has $H = (n - 1)!/2$ Hamiltonian circuits.

The counting convention for Hamiltonian circuits enables us to designate any vertex in a circuit as the starting point.

From the starting point, we can go to any $n - 1$ vertices, and from there to any one of $n - 2$ vertices, and so on until arriving at the last vertex and then returning to the starting point. By the basic counting principle, there are a total of

$(n - 1)(n - 2) \cdots 2 \cdot 1 = (n - 1)!$ circuits that can be formed from a starting point. For $n \geq 3$, any circuit can be paired with one in the opposite direction which determines the same Hamiltonian circuit. Accordingly, there are a total of $H = (n - 1)!/2$ Hamiltonian circuits.

PROBLEMS

GRAPH TERMINOLOGY

4.44 Consider the graph G in Fig. 4-58. Find:

(a) degree of each vertex (and verify Theorem 4.1);
(b) all simple paths from A to L;
(c) all trails (distinct edges) from B to C;
(d) d(A, C), distance from A to C;
(e) diam(G), the diameter of G.

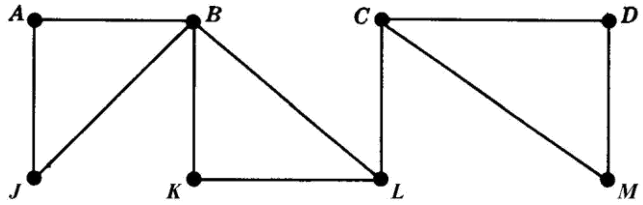


Fig. 4-58

4.35. Consider the graph in Fig. 8-58. Find (if any): (a) all cycles; (b) all cut points; (c) all bridges.

4.36. Consider the graph in Fig. 8-58. Find the sub-graph H = H (V , E) of G where V equals:

(a) {B, C, D, J, K} (b) {A, C, J, L, M} (c) {B, D, J, M} (d) {C, K, L, M}

Which of them are isomorphic and which homeomorphic?

4.37 Suppose a graph G contains two distinct paths from a vertex u to a vertex v. Show that G has a cycle.

4.38 Suppose G is a finite cycle-free graph with at least one edge. Show that G has at least two vertices of degree 1.

4.39 Show that a connected graph G with n vertices must have at least n − 1 edges.

4.40 Find the number of connected graphs with four vertices. (Draw them.)

4.41 Let G be a connected graph. Prove:

(a) If G contains a cycle C which contains an edge e, then G − e is still connected.

(b) If e = {u, v} is an edge such that G − e is disconnected, then u and v belong to different components of G − e.

4.42 Suppose G has V vertices and E edges. Let M and m denote, respectively, the maximum and minimum of the degrees of the vertices in G. Show that $m \leq 2E/V \leq M$.

4.43 Consider the following two steps on a graph G: (1) Delete an edge. (2) Delete a vertex and all edges containing that vertex. Show that every sub-graph H of a finite graph G can be obtained by a sequence consisting of these two steps.

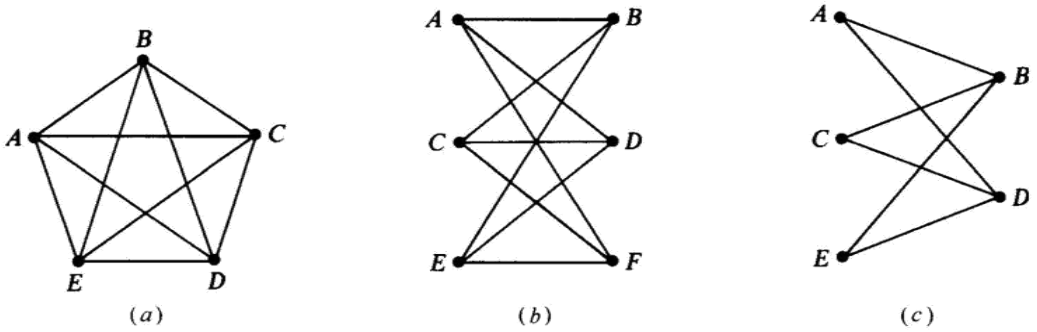
TRAVERSABLE GRAPHS, EULER AND HAMILTONIAN CIRCUITS

4.44 Consider the graphs K_5 , $K_{3,3}$ and $K_{2,3}$ in Fig. 4-59. Find an Euler (traversable) path or an Euler circuit of each graph, if it exists. If it does not, why not?

4.45 Consider each graph in Fig. 4-59. Find a Hamiltonian path or a Hamiltonian circuit, if it exists. If it does not, why not?

4.46 Show that K_n has $H = (n - 1)!/2$ Hamiltonian circuits. In particular, find the number of Hamiltonian circuits for the graph K_5 in Fig. 4-59(a).

4.47 Suppose G and G^* are homeomorphic graphs. Show that G is traversable (Eulerian) if and only if G^* is traversable (Eulerian).



(a) (b) (c)

Fig. 4-59

SPECIAL GRAPHS

4.48 Draw two 3-regular graphs with: (a) eight vertices; (b) nine vertices.

4.49 Consider the complete graph K_n.

(a) Find the diameter of K_n.
(b) Find the number m of edges in K_n.
(c) Find the degree of each vertex in K_n.
(d) Find those values of n for which K_n is: (i) traversable; (ii) regular.

4.50 Consider the complete graph $K_{m,n}$.

(b) Find the diameter of $K_{m,n}$.
(c) Find the number E of edges in $K_{m,n}$.
(d) Find those $K_{m,n}$ which are traversable.
(e) Which of the graphs $K_{m,n}$ are isomorphic and which homeomorphic?

4.51 The n-cube, denoted by $\mathbf{Q}_n$, is the graph whose vertices are the 2^n bit strings of length n, and where two vertices are adjacent if they differ in only one position. Figure 8-60(a) and (b) show the n-cubes $\mathbf{Q}_2$ and $\mathbf{Q}_3$.

(a) Find the diameter of $\mathbf{Q}_n$.
(b) Find the number m of edges in $\mathbf{Q}_n$.
(c) Find the degree of each vertex in $\mathbf{Q}_n$.
(d) Find those values of n for which $\mathbf{Q}_n$ is traversable.
(e) Find a Hamiltonian circuit (called a Gray code) for (i) $\mathbf{Q}_3$; (ii) $\mathbf{Q}_4$.

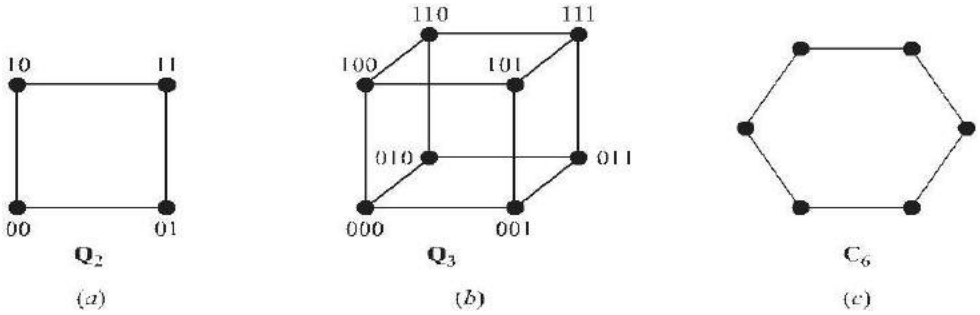


Fig. 4-60

4.52 The n-cycle, denoted by C_n, is the graph which consists of only a single cycle of length n. Figure 4-60(c) shows the 6-cycle C_6 . (a) Find the number of vertices and edges in C_n. (b) Find the diameter of C_n.

4.53 Describe those connected graphs which are both bipartite and regular.

TREES

4.54 Draw all trees with five or fewer vertices.

4.55 Find the number of trees with seven vertices.

4.56 Find the number of spanning trees in Fig. 4-61(a).

4.57 Find the weight of a minimum spanning tree in Fig. 4-61(b)

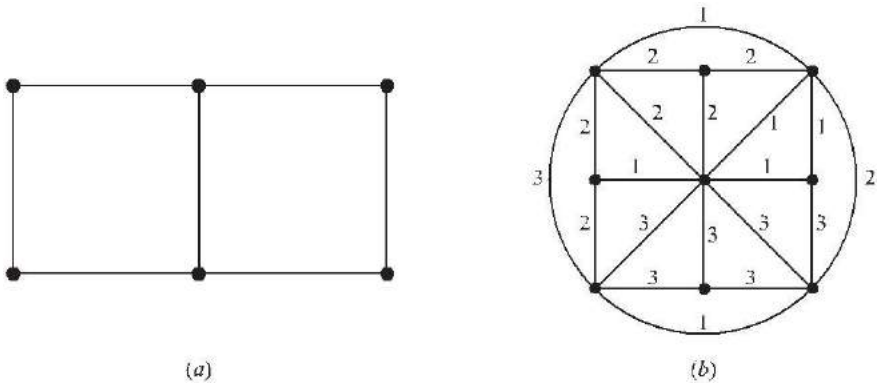


Fig. 4-61

4.58 Show that any tree is a bipartite graph.

4.59 Which complete bipartite graphs $K_{m,n}$ are trees.

PLANAR GRAPHS, MAPS, COLORINGS

4.60 Draw a planar representation of each graph G in Fig. 4-62, if possible; otherwise show that it has a sub-graph homeo-morphic to K_5 or $K_{3,3}$.

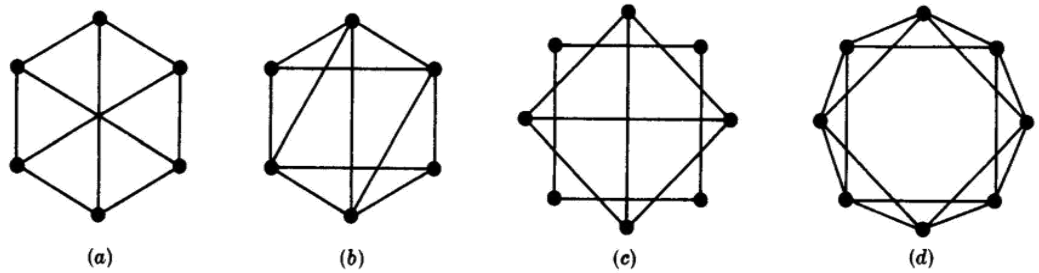


Fig. 4-62

4.61 Show that the 3-cube Q_3 (Fig. 4-60(b)) is planar.

4.62 For the map in Fig. 8-63, find the degree of each region and verify that the sum of the degrees of the regions is equal to twice the number of edges.

4.63 Count the number V of vertices, the number E of edges, and the number R of regions of each of the maps in Fig.4-64, and verify Euler's formula.

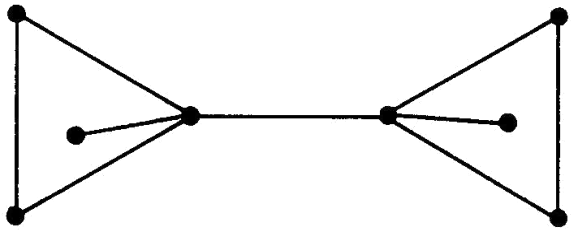


Fig.4-63

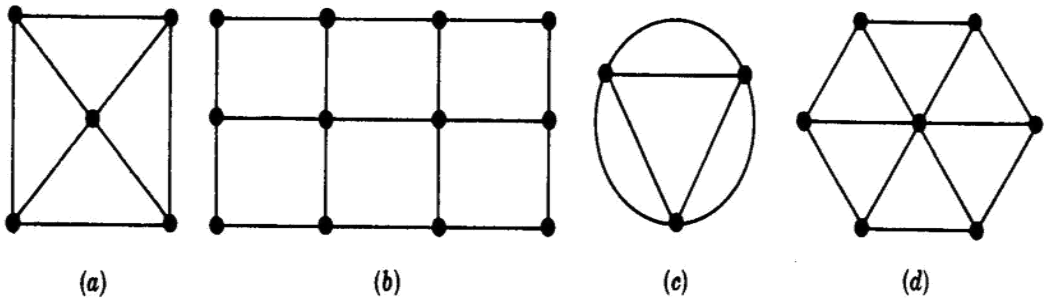


Fig. 4-64

4.64 Find the minimum number of colors needed to paint the regions of each map in Fig. 4-64.

4.65 Draw the map which is dual to each map in Fig. 4-64.

4.66 Use the Welch-Powell algorithm to paint each graph in Fig. 4-65. Find the chromatic number n of the graph.

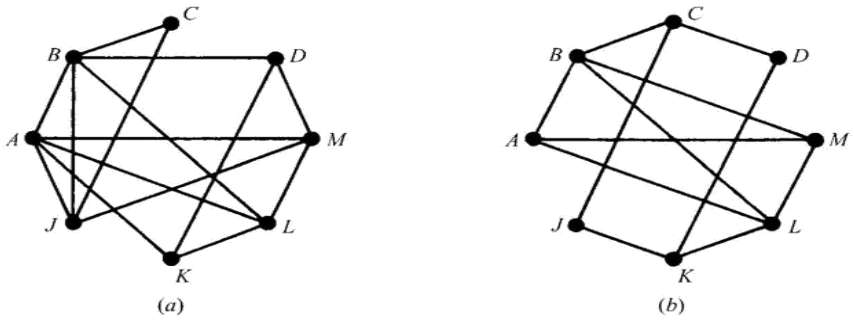


Fig. 4-65

SEQUENTIAL REPRESENTATION OF GRAPHS

4.67. Find the adjacency matrix A of each multigraph in Fig. 4-66.

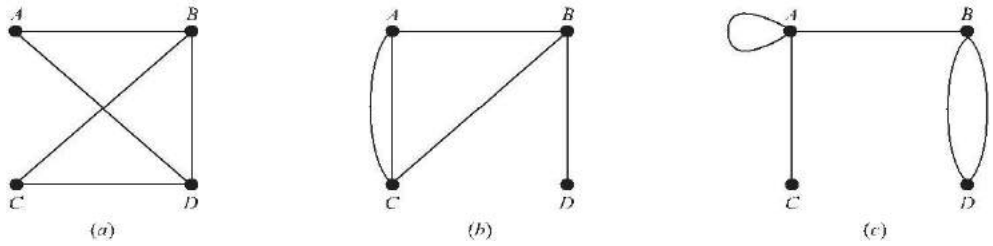


Fig. 4-66

4.68. Draw the multi-graph G corresponding to each of the following adjacency matrices:

(a) $\mathbf{A}=\begin{bmatrix} 0 & 2 & 0 & 1 \\ 2 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ $\quad A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 \\ 2 & 0 & 2 & 2 \end{bmatrix}$

4.69. Suppose a graph G is bipartite. Show that one can order the vertices of G so that its adjacency matrix A has the form:

$$' A = \begin{bmatrix} 0 & B \\ C & 0 \end{bmatrix}$$

LINKED REPRESENTATION OF GRAPHS

4.70. Suppose a graph G is stored in memory as in Fig. 4-67.

Vertex file

		1	2	3	4	5	6	7	8
	VERTEX	*C*		*F*	*E*	*A*		*B*	*D*
START 7	NEXT-V	0		5	1	8		3	4
	PTR	2		11	6	12		4	1

Edge file

	1	2	3	4	5	6	7	8	9	10	11	12
ADJ	7	7	4	5		7	1		8	3	1	7
NEXT	0	10	0	7		0	9		3	0	0	0

Fig. 4-67

(a) List the vertices in the order in which they appear in memory.

(b) Find the adjacency structure of G, that is, find the adjacency list adj(v) of each vertex v of G.

4.71 Exhibit the adjacency structure (AS) for each graph G in Fig. 4-59.

4.72 Figure 4-68(a) shows a graph G representing six cities A, B, . . ., F connected by seven highways numbered 22, 33, . . ., 88. Show how G may be maintained in memory using a linked representation with sorted arrays for the cities and for the numbered highways. (Note that VERTEX is a sorted array and so the field NEXT-V is not needed.)

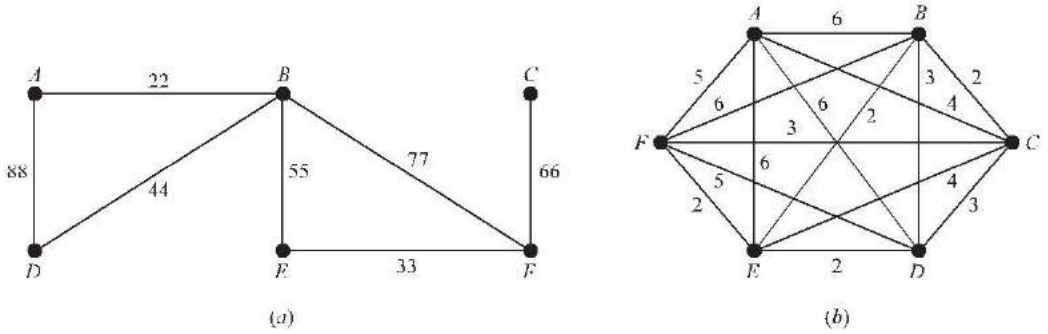


Fig. 4-68

TRAVELING–SALESMAN PROBLEM

4.73 Apply the nearest-neighbor algorithm to the complete weighted graph G in Fig. 4-68(b) beginning at: (a) vertex A; (b) vertex B.

4.74 Consider the complete weighted graph G in Fig. 4-57 with 5 vertices.

(a) Beginning at vertex A, list the H = (n – 1)!/2 = 12 Hamiltonian circuits of G, and find the weight of each of them.
(b) Find a Hamiltonian circuit of minimal weight.

GRAPH ALGORITHMS

4.75 Consider the graph G in Fig. 4-57 (where the vertices are ordered alphabetically).

(a) Find the adjacency structure (AS) of G.
(b) Using the DFS (depth-first search) Algorithm 4.5 on G and beginning at vertex C, find the STACK sequence and the order in which the vertices

are processed.

(c) Repeat (b) beginning at vertex K.

4.76 Using the BFS (breadth-first search) Algorithm 4.6 on the graph G in Fig. 4-57, find the QUEUE sequence and the order the vertices are processed beginning at: (a) vertex C; (b) vertex K.

4.77 Repeat Problem 4.75 for the graph G in Fig. 4-65(a).

4.78 Repeat Problem 4.76 for the graph G in Fig. 4-65(a).

4.79 Repeat Problem 4.75 for the graph G in Fig. 4-65(b).

4.80 Repeat Problem 4.76 for the graph G in Fig. 4-65(b).

Answers to Supplementary Problems

4.34. (a) 2, 4, 3, 2, 2, 2, 3, 2; (b) ABL, ABKL, AJ BL, AJ BKL; (c) BLC, BKLC, BAJ BLC, BAJ BKLC; d) 3; (e) 4.

4.35 (a) AJ BA, BKLB, CDMC; (b) B, C, L; (c) only {C, L}.

4.36. (a) E = {BJ, BK, CD }; (b) E = {AJ, CM, LC}; (c) E = {BJ, DM}; (d) E = {KL, LC, CM}, Also, (a) and (b) are isomorphic, and (a), (b), and (c) are homeomorphic.

4.38. Hint: Consider a maximal simple path α, and show that its endpoints have degree 1.

4.40. There are five of them, as shown in Fig. 4-69.

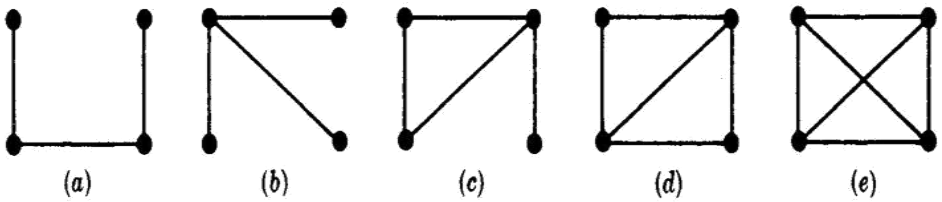


Fig. 4-69

4.42. Hint: Use Theorem 4.1.

4.43. First delete all edges in G not in H , then delete all vertrices in G not in H .

4.44 (a) Eulerian since all vertices are even: ABCDEACEBDA. (b) None, since four vertices are odd. (c) Euler path beginning at B and ending at D (or vice versa): BADCBED.

4.45 (a) ABCDEA; (b) ABCDEFA; (c) none, since B or D must be visited twice in any closed path including all vertices.

4.46 (5 – 1)!/2 = 12.

4.47 Hint: Adding a vertex by dividing an edge does not change the degree of the original vertices and simply adds a vertex of even degree.

4.48 (a) The two 3-regular graphs in Fig. 4-70 are not isomorphic: (b) has a 5-cycle, but (a) does not. (b) There are none. The sum of the degrees of an r-regular graph with s vertices equals rs, and rs must be even.

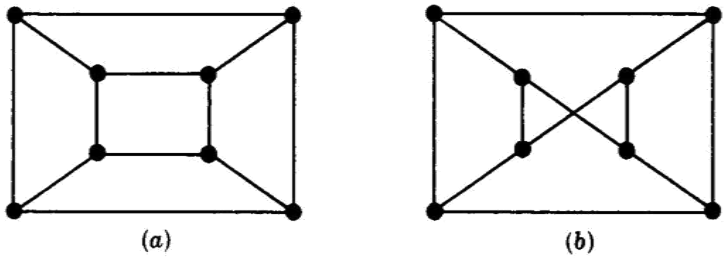


Fig. 4-70

4.49. (a) diam(K_1) = 0; all others have diameter 1; (b) m = C(n, 2) = n(n − 1)/2; (c) n − 1; (d)

(i) n = 2 and n odd;
(ii) all n.

4.50. (a) diam($K_{1,1}$) = 1; all others have diameter 2; (b) E = mn; (c) $K_{1,1}$, $K_{1,2}$, and all $K_{m,n}$ where m and n are even;

(d) none are isomorphic; only $K_{1,1}$ and $K_{1,2}$ are homeomorphic.

4.51. (a) n; (b) $n2^{n-1}$; (c) n; (d) n = 1, even; (e) consider the 4 × 16 matrix:

$$\mathbf{M} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

which shows how $\mathbf{Q}_4$ (the columns of M) is obtained from $\mathbf{Q}_3$. That is, the upper left 3 × 8 submatrix of M is $\mathbf{Q}_3$, the upper right 3 × 8 submatrix of M is $\mathbf{Q}_3$ written in reverse, and the last row consists of eight 0s followed by eight 1s.

4.52. (a) n and n; (b) n/2 when n is even, (n + 1)/2 when n is odd.

4.53. $K_{m,m}$ is bipartite and m-regular. Also, beginning with $K_{m,m}$, delete m disjoint edges to obtain a bipartite graph which is (m – 1)-regular, delete another m disjoint edges to obtain a bipartite graph which is (m – 2)-regular, and so on. These graphs may be disconnected, but their connected components have the desired properties.

4.54 There are eight such trees, as shown in Fig. 4-71. The graph with one vertex and no edges is called the trivial tree.

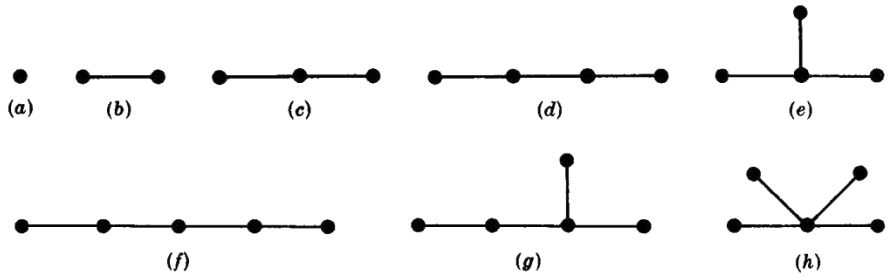


Fig. 4-71

4.55. 10

4.56. 15

4.57. 1 + 1 + 1 + 1 + 1 + 2 + 2 + 3 = 12.

4.58. m = 1.

4.59. Only (a) is non-planar, and $K_{3,3}$ is a sub-graph.

4.60. Figure 4-70(a) is a planar representation of Q3.

4.61. The outside region has degree 8, and the other two regions have degree 5.

4.62. (a) 5, 8, 5; (b) 12, 17, 7; (c) 3, 6, 5; (d) 7, 12, 7.

4.63. (a) 3; (b) 3; (c) 2; (d)3.

4.64. See Fig. 4-72.

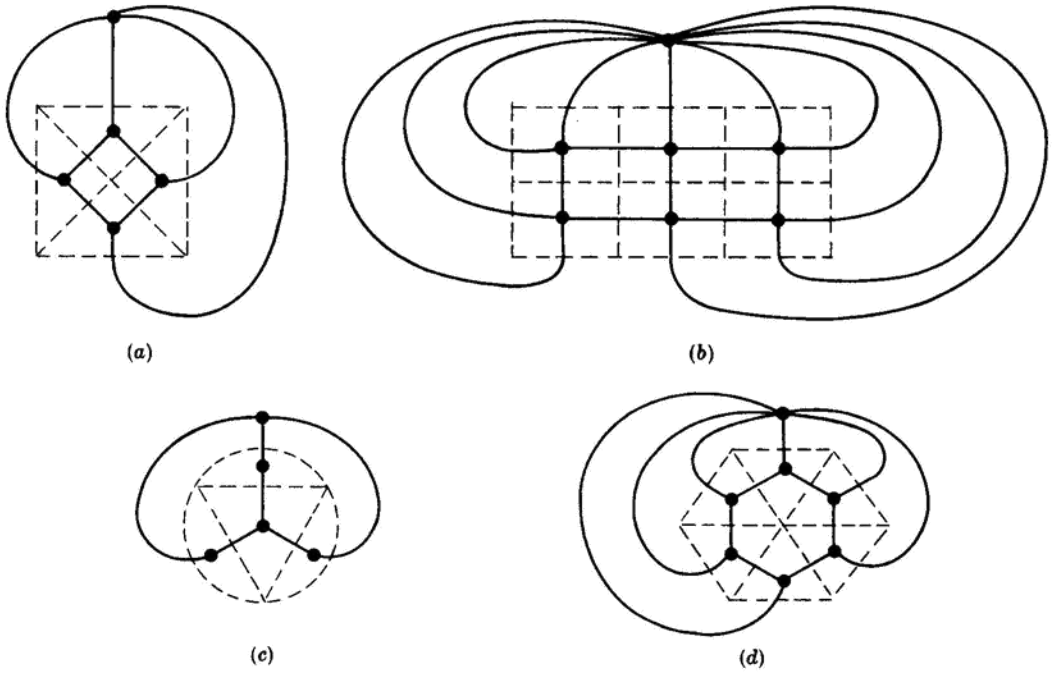


Fig. 4-72

4.66. (a) n = 3; (b) n = 4.

4.67. (a)= $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ **(b)=** $\begin{bmatrix} 0 & 1 & 2 & 0 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$ **(c)=** $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$

8.68. See Fig. 4-73.

8.69. Let M and N be the two disjoint sets of vertices determining the bipartite graph G. Order the vertioes in M first and then those in N .

8.70. (a) B, F , A, D, E, C.

(b) G = [A:B; B:A, C, D, E; C:F ; D:B; E:B; F :C].

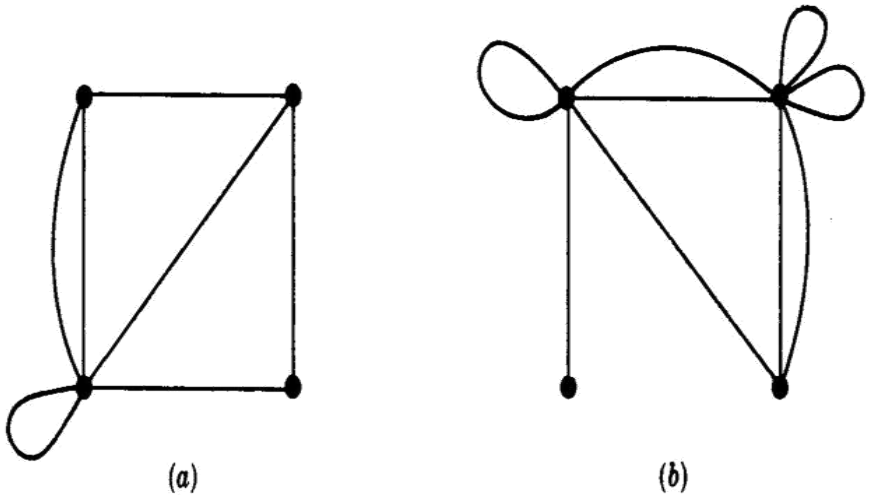


Fig. 4-73

8.71. (a) Each vertex is adjacent to the other four vertices.

(b) G = [A:B, D, F ; B:A, C, E; C:B, D, F ; D:A, C, E; E:B, D, F ; F :A, C, E].

(c) G = [A:B, D; B:A, C, E; C:B, D; D:A, C, E; E:B, D].

8.72. See Fig. 4-74.

Vertex file

	1	2	3	4	5	6	7	8
VERTEX	*A*	*B*	*C*	*D*	*E*	*F*		
PTR	1	2	9	14	8	12		

Edge file

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NUMBER	22	22	33	33	44	44	55	55	66	66	77	77	88	88	
ADJ	2	1	6	5	4	2	5	2	6	3	6	2	4	1	
NEXT	13	5	0	0	7	0	11	3	0	4	0	10	0	6	

Fig. 4-74

8.73. (a) |ACBEDFA| = 20 or |ACBEF DA| = 21; (b) |BCF EDAB| = 21 or |BCDEFAB| = 20

8.74. (a) |ABCDEA| = 775, |ABCEDA| = 725, |ABDCEA| = 1100, |ABDECA| = 900, |ABECDA| = 1050, |ABEDCA| = 900, |ACBDEA| = 825, |ACBEDA| = 775, |ACDBEA| = 1150, |ACEBDA| = 1100, |ADBCEA| = 975;

(b) |ABCEDA| = 725

8.75. (a) G = [A:BJ ; B:AJ KL; C:DLM; D:CM; J :AB; K:BL; L:BCK; M:CD]

(b) [STACK : C, MLD, DL, L, KB, B, J, A], CMDLKBJ A

(c) [STACK : K, LB, CB, MDB, DB, B, J A, A], KLCMDBJ A

8.76. (a) [QUEUE : C, MLD, ML, L, KB, J AK, J A, J], CDMLBKAJ

(b) [QUEUE : K, LB, J AL, CJ A, CJ, C, MD, M], KBLAJ CDM

8.77. (a) G = [A:BMJ KL; B:ACDJ L; C:BJ ; D:BKM; J :ABCM; K:ADL; L:ABKM; M:ADJ L]

(b) [STACK : C, J B, MBA, LDAB, KBAD, DAB, AB, B], CJ MLKDAB

(c) [STACK : K, LDA, MBAD, J DAB, CBAD, BAD, AD, D], KLMJ CBAD

8.78. (a) [QUEUE : C, J B, LDAJ, MLDA, KMLD, KML, KM, K], CBJ ADLMK

(b) [QUEUE : K, LDA, J MBLD, J MBL, CJ MB, CJ M, CJ, C], KADLBMJ C

8.79. (a) G = [A:BLM; B:ACLM; C:BDJ ; D:CK; J :CK; K:DJ L; L:ABKM; M:ABL]

(b) [STACK : C, J DB, KDB, LDB, MBAD, BAD, AD, D], CJ KLMBAD

(c) [STACK : K, LJ D, MBAJ D, BAJ D, CAJ D, J DA, DA, A], KLMBCJ DA

8.80. (a) [QUEUE : C, J DB, MLAJ D, KMLAJ, KMLA, KML, KM, K],
CBDJ ALMK

(b) [QUEUE : K, LJ D, CLJ, CL, MBAC, MBA, MB, M], KDJ LCABM

CHAPTER 5

LANGUAGES, AUTOMATA, GRAMMARS

5.1 INTRODUCTION

This chapter discusses three topics, languages, automata, and grammars. These three topics are closely related to each other. Our languages will use the letters a, b, . . . to code data rather than the digits 0 and 1 used by some other texts.

5.2 ALPHABET, WORDS, FREE SEMIGROUP

Consider a nonempty set A of symbols. A word or string w on the set A is a finite sequence of its elements. For example, suppose $A = \{a, b, c\}$. Then the following sequences are words on A:

$$u = ababb \quad \text{and} \quad v = accbaaa$$

When discussing words on A, we abbreviate our notation and write $v = ac^2ba^3$. frequently call A the alphabet, and its elements are called letters. We will also a^2 for aa, a^3 for aaa, and so on. Thus, for the above words, $u = abab^2$ and The empty sequence of letters, denoted by λ (Greek letter lambda) or ∈ (Greek letter epsilon), or 1, is also considered to be a word on A, called the empty word. The set of all words on A is denoted by

$$A^* \text{ (read: “}A\text{ star”).}$$

The length of a word u, written |u| or l(u), is the number of elements in its sequence of letters. For the above words u and v, we have $l(u) = 5$ and l(v) = 7. Also, $l(\lambda) = 0$, where λ is the empty word.

In abstract algebra, the free monoid on a set is the monoid whose elements

are all the finite sequences (or strings) of zero or more elements from that set, with string concatenation as the monoid operation and with the unique sequence of zero elements, often called the empty string and denoted by ε or λ, as the identity element. The free monoid on a set A is usually denoted A^*. The free semigroup on A is the bsemigroup of A^* containing all elements except the empty string. It is usually denoted A^+.

Remark: Unless otherwise stated, the alphabet A will be finite, the symbols *u, v, w* will be reserved for words on A, and the elements of A will come from the letters *a, b, c*.

5.3 CONCATENATION

Consider two words u and v on the alphabet A. The *concatenation* of u and v, written uv, is the word obtained by writing down the letters of u followed by the letters of v. For example, for the above words u and v, we have

$$uv = ababbaccbaaa = abab^2 ac^2 ba^3$$

As with letters, for any word u, we define $u^2 = uu$, $u^3 = uuu$, and, in general, $u^{n+1} = uu^n$.

Clearly, for any words u, v, w, the words $(uv)w$ and $u(vw)$ are identical, they simply consist of the letters of u, v, w written down one after the other. Also, adjoining the empty word before or after a word u does not change the word u. That is: A monoid is an algebraic structure intermediate between groups and semigroups, and is a semigroup having an identity element, thus obeying all but one of the axioms of a group; existence of inverses is not required of a monoid. A natural example is strings with concatenation as the binary operation, and the empty string as the identity element. Restricting to non-empty strings gives an example of a semigroup that is not a monoid. Positive integers with addition form a commutative semigroup that is not a monoid. Whereas the non-negative integers do form a monoid. A semigroup without an identity element can be easily turned into a monoid by just adding an identity element. Consequently, monoids are studied in the theory of semigroups rather than in group theory. Semigroups should not be confused withquasigroups, which are a generalization of groups in a different direction; the operation in a quasigroup

need not be associative but quasigroups preserve from groups a notion of division. Division in semigroups (or in monoids) is not possible in general. In theoretical computer science and formal language theory, a regular language (also called a rational language[1][2]) is a formal language that can be expressed using a regular expression, in the strict sense of the latter notion used in theoretical computer science (as opposed to many regular expressions engines provided by modern programming languages, which are augmented with features that allow recognition of languages that cannot be expressed by a classic regular expression). Alternatively, a regular language can be defined as a language recognized by a finite automaton. The equivalence of regular expressions and finite automata is known as Kleene's theorem. In the Chomsky hierarchy, regular languages are defined to be the languages that are generated by Type-3 grammars (regular grammars). l finite languages are regular; in particular the empty string language $\{\varepsilon\} = \emptyset^*$ is regular. Other typical examples include the language consisting of all strings over the alphabet $\{a, b\}$ which contain an even number of as, or the language consisting of all strings of the form: several as followed by several bs.

A simple example of a language that is not regular is the set of strings $\{ a^n b^n \mid n \geq 0 \}$.[4] Intuitively, it cannot be recognized with a finite automaton, since a finite automaton has finite memory and it cannot remember the exact number of a's. The set of regular languages over an alphabet Σ is defined recursively as below. Any language belonging to this set is a regular language over Σ. Regular languages are very useful in input parsing and programming language design.

THEOREM 5.1: The concatenation operation for words on an alphabet A is associative. The empty word λ is an identity element for the operation.

(Generally speaking, the operation is not commutative, e.g., $uv \neq vu$ for the above words u and v.)

Subwords, Initial Segments

Consider any word $u = a_1 a_2 \ldots a_n$ on an alphabet A. Any sequence $w = a_j a_{j+1} \ldots a_k$ is called a *subword* of u. In particular, the subword $w = a_1 a_2 \ldots a_k$ beginning with the first letter of u, is called an *initial segment* of u. In other words, w is a

subword of u if $u = v_1wv_2$ and w is an initial segment of u if $u = wv$. Observe that λ and u are both subwords of uv since $u = \lambda u$.

Consider the word $u = abca$. The subwords and initial segments of u follow:

(1) Subwords: $\lambda, a, b, c, ab, bc, ca, abc, bca, abca = u$

(2) Initial segments: $\lambda, a, ab, abc, abca = u$.

Observe that the subword $w = a$ appears in two places in u. The word ac is not a subword of u even though all its letters belong to u.

Free Semigroup, Free Monoid

Let F denote the set of all nonempty words from an alphabet A with the operation of concatenation. As noted above, the operation is associative. Thus F is a semigroup; it is called the *free semigroup over A* or the *free semigroup generated by A*. One can easily show that F satisfies the right and left cancellation laws. However, F is not commutative when A has more than one element. We will write F_A for the free semigroup over A when we want to specify the set A.

Now let $M = A^*$ be the set of all words from A including the empty word λ. Since λ is an identity element for the operation of concatenation, M is a monoid, called the *free* monoid over A.

5.4 LANGUAGES

A language L over an alphabet A is a collection of words on A. Recall that A^* denotes the set of all words on A. Thus a language L is simply a subset of A^*.

EXAMPLE 5.1 Let $A = \{a, b\}$. The following are languages over A.

(a) $L_1 = \{a, ab, ab^2, \ldots\}$
(b) $L_2 = \{a^m b^n \mid m > 0, n > 0\}$
(c) $L_3 = \{a^m b^m \mid m > 0\}$
(d) $L_4 = \{b^m a b^n \mid m \geq 0, n \geq 0\}$

One may verbally describe these languages as follows.

(a) L_1 consists of all words beginning with an a and followed by zero or more b's.

(b) L_2 consists of all words beginning with one or more as followed by one or more b's.

(c) L_3 consists of all words beginning with one or more as and followed by the same number of b's.

(d) L_4 consists of all words with exactly one a.

Operations on Languages

Suppose L and M are languages over an alphabet A. Then the "concatenation" of L and M, denoted by LM, is the language defined as follows:

$$LM = \{uv \mid u \in L, v \in V\}$$

That is, LM denotes the set of all words which come from the concatenation of a word from L with a word from M. For example, suppose

$$L_1 = \{a, b^2\}, \quad L_2 = \{a^2, ab, b^3\}, \quad L_3 = \{a^2, a^4, a^6, \ldots\}$$

Then:

$$L_1L_1 = \{a^2, ab^2, b^2a, b^4\}, L_1L_2 = \{a^3, a^2b, ab^3, b^2a^2, b^2ab, b^5\}$$
$$L_1L_3 = \{a^3, a^5, a^7, \ldots, b^2a^2, b^2a^4, b^2a^6, \ldots\}$$

Clearly, the concatenation of languages is associative since the concatenation of words is associative. *Powers* of a language L are defined as follows:

$$L^0 = \{\lambda\}, \quad L^1 = L, \quad L^2 = LL, \quad L^{m+1} = L^mL \quad \text{for} \quad m > 1.$$

The unary operation L^* (read "L star") of a language L, called the Kleene closure of L because Kleene proved Theorem 5.2, is defined as the infinite union:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \cdots =$$

The definition of L^* agrees with the notation A^* which consists of all words over A. Some texts define L^+ to be the union of $L^1, L^2, \ldots$ that is, L^+ is the same as L^* but without the empty word λ.

5.5 REGULAR EXPRESSIONS, REGULAR LANGUAGES

Let A be a (nonempty) alphabet. This section defines a regular expression r over A and a language $L(r)$ over A associated with the regular expression r. The expression r and its corresponding language $L(r)$ are defined inductively as follows.

DEFINITION 5.1: Each of the following is a regular expression over an alphabet A.

(1) The symbol "λ" (empty word) and the pair "( )" (empty expression) are regular expressions.

(2) Each letter a in A is a regular expression.

(3) If r is a regular expression, then (r^*) is a regular expression.

(4) If r_1 and r_2 are regular expressions, then $(r_1 \vee r_2)$ is a regular expression.

(5) If r_1 and r_2 are regular expressions, then $(r_1 r_2)$ is a regular expression. All regular expressions are formed in this way.

Observe that a regular expression r is a special kind of a word (string) which uses the letters of A and the five symbols:

$$( ) \quad * \quad \vee \quad \lambda$$

We emphasize that no other symbols are used for regular expressions.

DEFINITION 5.2: The language $L(r)$ over A defined by a regular expression r over A is as follows:

(1) $L(\lambda) = \{\lambda\}$ and $L(( )) = \emptyset$, the empty set.

(2) $L(a) = \{a\}$, where a is a letter in A.
(c) $L(r^*) = (L(r))^*$ (the Kleene closure of $L(r)$).
(d) $L(r_1 \vee r_2) = L(r_1) \cup L(r_2)$ (the union of the languages).
(e) $L(r_1 r_2) = L(r_1)L(r_2)$ (the concatenation of the languages).

Remark: Parentheses will be omitted from regular expressions when possible. Since the concatenation of languages and the union of languages are associative, many of the parentheses may be omitted. Also, by adopting the convention that "*" takes precedence over concatenation, and concatenation takes precedence over "∨," other parentheses may be omitted.

DEFINITION 5.3: Let L be a language over A. Then L is called a regular language over A if there exists a regular expression r over A such that $L = L(r)$.

EXAMPLE 5.2 Let $A = \{a, b\}$. Each of the following is an expression r and its corresponding language $L(r)$:

(a) Let $r = a^*$. Then $L(r)$ consists of all powers of a including the empty word λ.

(b) Let $r = aa^*$. Then $L(r)$ consists of all positive powers of a excluding the empty word.

(c) Let $r = a \vee b^*$. Then $L(r)$ consists of a or any word in b, that is, $L(r) = \{a, \lambda, b, b^2, \cdots\}$.

(d) Let $r = (a \vee b)^*$. Note $L(a \vee b) = \{a\} \cup \{b\} = A$; hence $L(r) = A^*$, all words over A.

(e) Let $r = (a \vee b)^*bb$. Then $L(r)$ consists of the concatenation of any word in A with bb, that is, all words ending in b^2.

(f) Let $r = a \wedge b^*$. $L(r)$ does not exist since r is not a regular expression. (Specifically, ∧ is not one of the symbols used for regular expressions.)

EXAMPLE 5.3 Consider the following languages over $A = \{a, b\}$:

(a) $L_1 = \{a^m b^n \mid m > 0, n > 0\}$;
(b) $L_2 = \{b^m a b^n \mid m > 0, n > 0\}$;
(c) $L_3 = \{a^m b^m \mid m > 0\}$.

Find a regular expression r over $A = \{a, b\}$ such that $L_i = L(r)$ for $i = 1, 2, 3$.

(a) L_1 consists of those words beginning with one or more a's followed by one or more b's. Thus we can set $r = aa^*bb^*$. Note that r is not unique; for example, $r = a^*abb^*$ is another solution.

(b) L_2 consists of all words which begin with one or more b's followed by a single a which is then followed by one or more b's, that is, all words with exactly one a which is neither the first nor the last letter. Hence
$r = bb^*abb^*$ is a solution.

(c) L_3 consists of all words beginning with one or more a's followed by the same number of b's. There exists no regular expression r such that $L_3 = L(r)$; that is, L_3 is not a regular language. The proof of this fact appears in Example 5.8.

5.6 FINITE STATE AUTOMATA

A *finite state automaton* (FSA) or, simply, an *automaton M*, consists of five parts:

(1) A finite set (alphabet) A of inputs.

(2) A finite set S of (internal) states.

(3) A subset Y of S (called accepting or "yes" states).

(4) An initial state s_0 in S.

(5) A next-state function F from $S \times A$ into S.

Such an automaton M is denoted by $M = (A, S, Y, s_0, F)$ when we want to indicate its five parts. (The plural of automaton is *automata*.)

Some texts define the next-state function $F : S \times A \to S$ in (5) by means of a collection of functions

$f_a : S \to S$, one for each $a \in A$. Setting $F(s, a) = f_a(s)$ shows that both definitions are equivalent.

EXAMPLE 5.4 The following defines an automaton M with two input symbols and three states:

(1) $A = \{a, b\}$, input symbols.

(2) $S = \{s_0, s_1, s_2\}$, internal states.

(3) $Y = \{s_0, s_1\}$, "yes" states.

(4) s_0, initial state.

(5) Next-state function $F : S \times A \to S$ defined explicitly in Fig. 5-1(a) or by the table in Fig. 5-1(b).

$$F(s_0, a) = s_0, \quad F(s_1, a) = s_0, \quad F(s_2, a) = s_2$$
$$F(s_0, b) = s_1, \quad F(s_1, b) = s_2, \quad F(s_2, b) = s_2$$

(a)

F	a	b
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_2	s_2

(b)

Fig. 5-1

State Diagram of an Automaton M

An automaton M is usually defined by means of its state diagram $D = D(M)$ rather than by listing its five parts. The state diagram $D = D(M)$ is a labeled directed graph as follows.

(1) The vertices of $D(M)$ are the states in S and an accepting state is denoted by means of a double circle.

(2) There is an arrow (directed edge) in $D(M)$ from state s_j to state s_k labeled by an input a if $F(s_j, a) = s_k$ or, equivalently, if $f_a(s_j) = s_k$.

(3) The initial state s_0 is indicated by means of a special arrow which terminates at s_0 but has no initial vertex.

For each vertex s_j and each letter a in the alphabet A, there will be an arrow leaving s_j which is labeled by a; hence the outdegree of each vertex is equal to number of elements in A. For notational convenience, we label a single arrow by all the inputs which cause the same change of state rather than having an arrow for each such input.

The state diagram $D = D(M)$ of the automaton and b label the arrow from s_2 to s_2 since $F(s_2, a)$ = vertex is 2, the number of elements in A. M in Example 5.4 appears in Fig. 5-2. Note that both a s_2 and $F(s_2, b) = s_2$. Note also that the out degree of each.

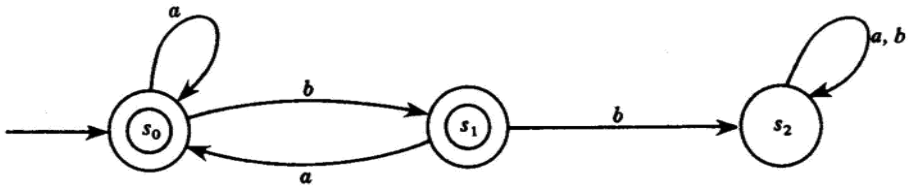


Fig. 5-2

Language $L(M)$ Determined by an Automaton M

Each automaton M with input alphabet A defines a language over A, denoted by $L(M)$, as follows.

Let $w = a_1a_2 \cdots a_m$ be a word on A. Then w determines the following path in the state diagram graph $D(M)$ where s_0 is the initial state and $F(s_{i-1}, a_i) = s_i$ for $i \geq 1$:

$$P = (s_0, a_1, s_1, a_2, s_2, \cdots, a_m, s_m)$$

We say that M *recognizes* the word w if the final state s_m is an accepting state in Y. The language $L(M)$ of M is the collection of all words from A which are accepted by M.

EXAMPLE 5.5 Determine whether or not the automaton M in Fig.5-2 accepts the words:

$$w_1 = ababba;\ w_2 = baab;\ w_3 = \lambda \text{ the empty word.}$$

Use Fig. 5-2 and the words w_1 and w_2 to obtain the following respective paths:

$$P_1 = s_0 \xrightarrow{a} s_0 \xrightarrow{b} s_1 \xrightarrow{a} s_0 \xrightarrow{b} s_1 \xrightarrow{b} s_2 \xrightarrow{a} s_2 \qquad P_2 = s_0 \xrightarrow{b} s_1 \xrightarrow{a} s_0 \xrightarrow{a} s_0 \xrightarrow{b} s_1$$

The final state in P_1 is s_2 which is not in Y ; hence w_1 is not accepted by M. On the other hand, the final state in P_2 is s_1 which is in Y ; hence w_2 is accepted by M. The final state determined by w_3 is the initial state s_0 since $w_3 = \lambda$ is the empty word. Thus w_3 is accepted by M since $s_0 \in Y$.

EXAMPLE 5.6 Describe the language $L(M)$ of the automaton M in Fig. 5-2.

$L(M)$ will consist of all words w on A which do not have two successive b's. This comes from the following facts:

(1) We can enter the state s_2 if and only if there are two successive b's.
(2) We can never leave s_2.
(3) The state s_2 is the only rejecting (non-accepting) state.

The fundamental relationship between regular languages and automata is contained in the following theorem (whose proof lies beyond the scope of this text).

THEOREM 5.2 (Kleene): A language L over an alphabet A is regular if and only if there is a finite state automaton M such that $L = L(M)$.

The star operation L^* on a language L is sometimes called the Kleene closure of L since Kleene first proved the above basic result.

EXAMPLE 5.7 Let $A = \{a, b\}$. Construct an automaton M which will accept precisely those words from A which end in two b´s.

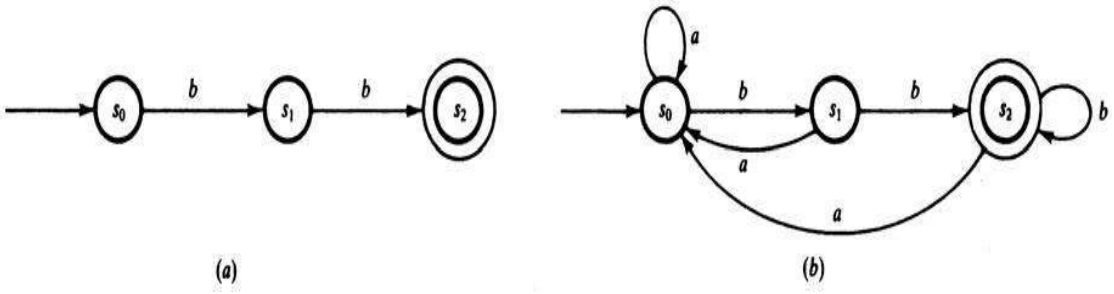


Fig. 5-3

Since b^2 is accepted, but not λ or b, we need three states, s_0, the initial state, and s_1 and s_2 with an arrow labeled b going from s_0 to s_1 and one from s_1 to s_2. Also, s_2 is an accepting state, but not s_0 nor s_1. This gives the graph in Fig. 5-3(a). On the other hand, if there is an a, then we want to go back to s_0, and if we are in s_2 and there is a b, then we want to stay in s_2. These additional conditions give the required automaton M which is shown in Fig. 5-3(b).

5.7 PUMPING LEMMA

Let M be an automaton over A with k states. Suppose $w = a_1a_2 \cdots a_n$ is a word over A accepted by M and suppose $|w| = n > k$, the number of states. Let

$$P = (s_0, s_1, \ldots, s_n)$$

be the corresponding sequence of states determined by the word w. Since $n > k$, two of the states in P must be equal, say $s_i = s_j$ where $i < j$. Let w be divided into subwords x, y, z as follows:

$$x = a_1a_2 \cdots a_i\,, \quad y = a_{i+1} \cdots a_j\,, \quad z = a_{j+1} \cdots a_n$$

As shown in Fig. 5-4, xy ends in $s_i = s_j$; hence xy^m also ends in s_i . Thus, for every m, $w_m = xy^mz$ ends in s_n, which is an accepting state.

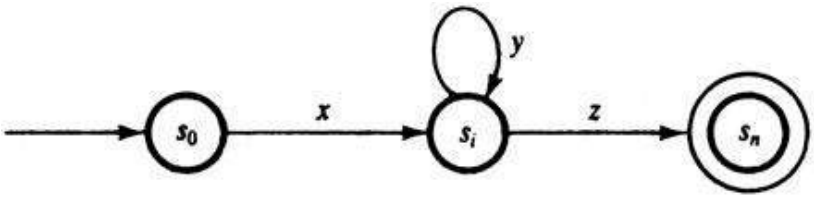


Fig. 5-4

The above discussion proves the following important result.

THEOREM 5.3 (Pumping Lemma): Suppose M is an automaton over A such that:

(i) M has k states. (ii) M accepts a word w from A where $|w| > k$.

Then $w = xyz$ where, for every positive m, $w_m = xy^m z$ is accepted by M. The next example gives an application of the Pumping Lemma.

EXAMPLE 5.8 Show that the language $L = \{a^m b^m \mid m \text{ is positive}\}$ is not regular.

Suppose L is regular. Then, by Theorem 5.2, there exists a finite state automaton M which accepts L. Suppose M has k states. Let $w = a^k b^k$. Then $|w| > k$. By the Pumping Lemma (Theorem 5.3), $w = xyz$ where y is not empty and $w_2 = xy^2z$ is also accepted by M. If y consists of only a's or only b's, then w^2 will not have the same number of a's as b's. If y contains both a's and b's, then w_2 will have a's following b's. In either case w_2 does not belong to L, which is a contradiction. Thus L is not regular.

5.8 GRAMMARS

Figure 5-5 shows the grammatical construction of a specific sentence. Observe that there are:

(1) various variables, e.g., (sentence), (noun phrase), · · ·;

(2) various terminal words, e.g., "The", "boy," · · ·;

(3) a beginning variable (sentence);

(4) various substitutions or productions, e.g.

$$\langle sentence \rangle \rightarrow \langle noun\ phrase \rangle\ \langle verb\ phrase \rangle$$
$$\langle object\ phrase \rangle \rightarrow \langle article \rangle\ \langle noun \rangle$$
$$\langle noun \rangle \rightarrow \text{apple}$$

The final sentence only contains terminals, although both variables and terminals appear in its construction by the productions. This intuitive description is given in order to motivate the following definition of a grammar and the language it generates.

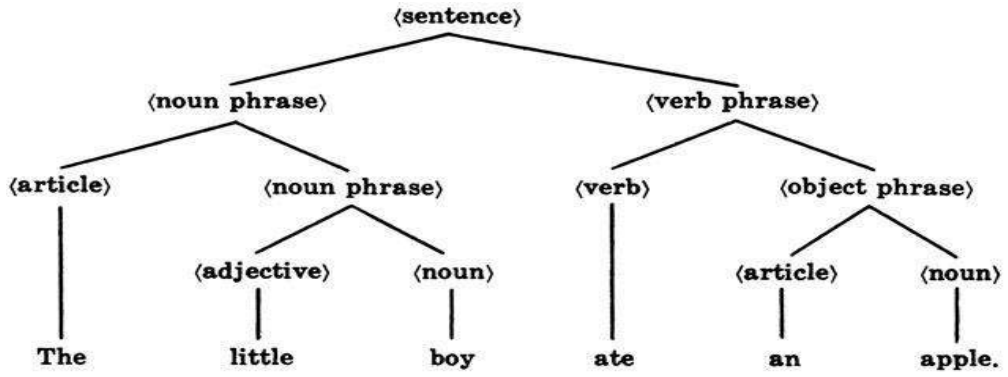


Fig. 5-5

DEFINITION 5.4: A *phrase structure grammar* or, simply, a *grammar G* consists of four parts:

(1) A finite set (vocabulary) V .

(2) A subset T of V whose elements are called *terminals*; the elements of $N = V \setminus T$ are called *non-terminals* or *variables*.

(3) A non-terminal symbol S called the *start* symbol.

(4) A finite set P of productions. (A production is an ordered pair (a, β), usually written $a \rightarrow \beta$, where a and

β are words in V , and the production must contain at least one non-terminal on its left side a.)

Such a grammar G is denoted by $G = G(V, T, S, P)$ when we want to indicate its four parts.

The following notation, unless otherwise stated or implied, will be used for our grammars. Terminals will be denoted by italic lower case Latin letters, a, b, c, $\cdots$, and nonterminals will be denoted by italic capital Latin letters, A, B, C, $\cdots$, with S as the start symbol. Also, Greek letters, α, β, $\cdots$, will denote words in V, that is, words in terminals and nonterminals. Furthermore, we will write

$$a \to (\beta_1, \beta_2, \cdots, \beta_k) \quad \text{instead of} \quad a \to \beta_1, a \to \beta_2, \cdots, a \to \beta_k$$

Remark: Frequently, we will define a grammar G by only giving its productions, assuming implicitly that S is the start symbol and that the terminals and nonterminals of G are only those appearing in the productions.

EXAMPLE 5.9 The following defines a grammar G with S as the start symbol:

$$V = \{A, B, S, a, b\}, \quad T = \{a, b\}, \quad P = \{S \overset{1}{\to} AB, \text{A} \overset{2}{\to} \text{Aa}, \text{B} \overset{3}{\to} \text{Bb}, \text{A} \overset{4}{\to} \text{a}, \text{B} \overset{5}{\to} \text{b}\}$$

The productions may be abbreviated as follows: $S \to AB$, $A \to (Aa, a)$, $B \to (Bb, b)$

Language $L(G)$ of a Grammar G

Suppose w and w' are words over the vocabulary set V of a grammar G. We write $w \Rightarrow w'$

if w' can be obtained from w by using one of the productions; that is, if there exists words u and v such that $w = uav$ and $w = u\beta v$ and there is a production $a \to \beta$. Furthermore, we write

$$\text{w} \overset{*}{\Rightarrow} w'$$

If w' can be obtained from w using a finite number of productions.

Now let G be a grammar with terminal set T. The language of G, | denoted by $L(G)$, consists of all words in T that can be obtained from the start symbol S by the above process; that is,

$$L(G) = \{w \in T^* \mid S \Rightarrow\Rightarrow w\}$$

EXAMPLE 5.10 Consider the grammar G in Example 5.9. Observe that $w = a^2b^4$ can be obtained from the start symbol S as follows:

$$S \Rightarrow AB \Rightarrow AaB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aaBbb \Rightarrow aaBbbb \Rightarrow aabbbb = a^2b^4$$

Here we used the productions 1, 2, 4, 3, 3, 3, 5, respectively. Thus we can write $S \Rightarrow\Rightarrow a^2b^4$. Hence $w = a^2b^4$ belongs to $L(G)$. More generally, the production sequence:

1, 2 (r times), 4, 3 (s times), 5

will produce the word $w = a^r ab^s b$ where r and s are nonnegative integers. On the other hand, no sequence of productions can produce an a after a b. Accordingly,

$$L(G) = \{a^m b^n \mid m \text{ and } n \text{ are positive integers}\}$$

That is, the language $L(G)$ of the grammar G consists of all words which begin with one or more a's followed by one or more b's.

EXAMPLE 5.11 Find the language $L(G)$ over $\{a, b, c\}$ generated by the grammar G:

$$S \to aSb, \quad aS \to Aa, \quad Aab \to c$$

First we must apply the first production one or more times to obtain the word $w = a^nSb^n$ where $n > 0$. To eliminate S, we must apply the second production to obtain the word $w = a^mAabb^m$ where $m = n - 1 \geq 0$. Now we can only apply the third production to finally obtain the word $w = a^mcb^m$ where $m \geq 0$. Accordingly,

$$L(G) = \{a^m cb^m \mid m \text{ nonnegative}\}$$

That is, $L(G)$ consists of all words with the same nonnegative number of a's and b's separated by a c.

Types of Grammars

Grammars are classified according to the kinds of production which are allowed. The following grammar classification is due to Noam Chomsky.

A Type 0 grammar has no restrictions on its productions. Types 1, 2, and 3 are defined as follows:

(1) A grammar G is said to be of Type 1 if every production is of the form $a \to \beta$ where $|a| \leq |\beta|$ or of the form $a \to \lambda$.

(2) A grammar G is said to be of Type 2 if every production is of the form $A \to \beta$ where the left side A is a nonterminal.

(3) A grammar G is said to be of Type 3 if every production is of the form $A \to a$ or $A \to aB$, that is, where the left side A is a single nonterminal and the right side is a single terminal or a terminal followed by a nonterminal, or of the form $S \to \lambda$.

Observe that the grammars form a hierarchy; that is, every Type 3 grammar is a Type 2 grammar, every Type 2 grammar is a Type 1 grammar, and every Type 1 grammar is a Type 0 grammar.

Grammars are also classified in terms of context-sensitive, context-free, and regular as follows.

(a) A grammar G is said to be *context-sensitive* if the productions are of the form

$$aA\alpha' \to a\beta\alpha'$$

The name "context-sensitive" comes from the fact that we can replace the variable A by β in a word only when A lies between a and α'.

(b) A grammar G is said to be *context-free* if the productions are of the form

$$A \to \beta$$

The name "context-free" comes from the fact that we can now replace the variable A by β regardless of where A appears.

(c) A grammar G is said to be *regular* if the productions are of the form

$$A \to a, \quad A \to aB, \quad S \to \lambda$$

Observe that a context-free grammar is the same as a Type 2 grammar, and a regular grammar is the same as a Type 3 grammar.

A fundamental relationship between regular grammars and finite automata follows.

THEOREM 5.4: A language L can be generated by a Type 3 (regular) grammar G, if and only if there exists a finite automaton M which accepts L .

Thus a language L is regular iff $L = L(r)$ where r is a regular expression iff $L = L(M)$ where M is a finite automaton iff $L = L(G)$ where G is a regular grammar. (Recall that "iff" is an abbreviation for "if and only if.")

EXAMPLE 5.12 Consider the language $L = \{a^n b^n \mid n > 0\}$.

(a) Find a context-free grammar G which generates L. Clearly the grammar G with the following productions will generate L:

$$S \to ab, \quad S \to aSb$$

Note that G is context-free

(b) Find a regular grammar G which generates L. By Example 5.8, L is not a regular language. Thus L cannot be generated by a regular grammar.

Derivation Trees of Context-Free Grammars

Consider a context-free (Type 2) grammar G. Any derivation of a word w in $L(G)$ can be represented graphically by means of an ordered, rooted tree T , called a *derivation tree or parse tree*. We illustrate such a derivation tree below.

Let G be the context-free grammar with the following productions:

$$S \to aAB, \quad A \to Bba, \quad B \to bB, \quad B \to c$$

The word $w = acbabc$ can be derived from S as follows:

$$S \Rightarrow aAB \Rightarrow a(Bba)B \Rightarrow acbaB \Rightarrow acba(bB) \Rightarrow acbabc$$

One can draw a derivation tree T of the word w as indicated by Fig. 5-6. Specifically, we begin with S as the root and then add branches to the tree according to the production used in the derivation of w. This yields the completed tree T which is shown in Fig. 5-6(e). The sequence of leaves from left to right in T is the derived word w. Also, any non-leaf in T is a variable, say A, and the immediate successors (children) of A form a word α where $A \to a$ is the production of G used in the derivation of w.

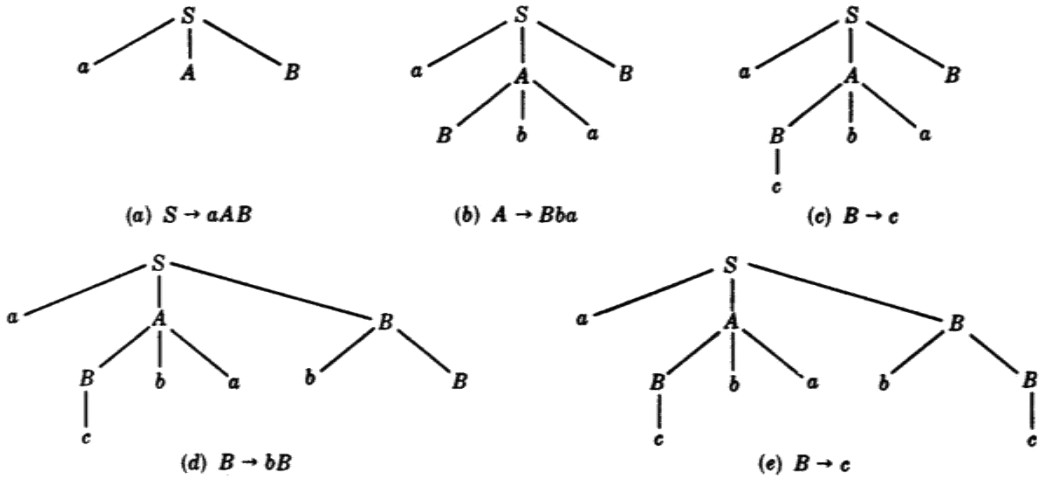


Fig. 5-6

Backus-Naur Form

There is another notation, called the Backus-Naur form, which is sometimes used for describing the produc-tions of a context-free (Type 2) grammar. Specifically:

(i) "::=" is used instead of "→."

(ii) Every nonterminal is enclosed in bracket ⟨ ⟩.

(iii) All productions with the same nonterminal left-hand side are combined into one statement with all the right-hand sides listed on the right of :: = separated by vertical bars.

For instance, the productions $A \to aB$, $A \to b$, $A \to BC$ are combined into the one statement:

$$\langle A \rangle \ :: = \text{a} \ \langle B \rangle \ | \text{b} | \ \langle B \rangle \langle C \rangle$$

5.9 MACHINES AND GRAMMARS

Theorem 5.4 tells us that the regular languages correspond to the finite state automata (FSA). There are also machines, more powerful than the FSA, which correspond to the other grammars.

(a) **Pushdown Automata:** A pushdown automaton P is similar to a FSA except that P has an auxiliary stack which provides an unlimited amount of memory for P . A language L is recognized by a pushdown automaton P if and only if L is context-free.

(b) **Linear Bounded Automata:** A linear bounded automaton B is more powerful than a pushdown automaton. Such an automaton B uses a tape which is linearly bounded by the length of the input word w. A language L is recognized by a linear bounded automaton B if and only if L is context-sensitive.

(c) **Turing Machine:** A Turing machine M, named after the British mathematician Alan Turing, uses an infinite tape; it is able to recognize every language L that can be generated by any phase-structure grammar G. In fact, a Turing machine M is one of a number of equivalent ways to define the notion of a "computable" function.

The discussion of the pushdown automata and the linear bounded automata lies beyond the scope of this text.

SOLVED PROBLEMS

WORDS

5.1. Consider the words $u = a^2ba^3b^2$ and $v = bab^2$. Find: (a) uv; $|uv|$; (b) vu, $|vu|$; (c) v^2, $|v^2|$.

Write the letters of the first word followed by the letters of the second word, and then count the number of letters in the resulting word.

(a) $uv = (a^2\, ba^3\, b^2\,)(bab^2\,) = a^2\, ba^3\, b^3\, ab^2\,$; $|uv| = 12$
(b) $vu = (bab^2\,)(a^2\, ba^3\, b^2\,) = bab^2\, a^2\, ba^3\, b^2\,$; $|vu| = 12$
(c) $v^2 = vv = (bab^2\,)(bab^2\,) = bab^3\, ab^2\,$; $|v^2\,| = 8$

5.2. Suppose $u = a^2b$ and $v = b^3ab$. Find: (a) uvu; (b) $\boldsymbol{\lambda}u$, $u\lambda$, $u\lambda v$.

(a) Write down the letters in u, then v, and finally u to obtain $uvu = a^2\, b^4\, aba^2\, b$. (b) Since λ is the empty word, $\lambda u = u\lambda = u = a^2\, b$ and $u\lambda v = uv = a^2\, b^4\, ab$.

5.3. Let $w = abcd$. (a) Find all subwords of w. (b) Which of them are initial segments?

(a) The subwords are: λ, a, b, c, d, ab, bc, cd, abc, bcd, $w = abcd$. (We emphasize that $v = acd$ is not a subword of w even though all its letters belong to w.)

(b) The initial segments are λ, a, ab, abc, $w = abcd$.

5.4. For any words u and v, show that: (a) $|uv| = |u| + |v|$; (b) $|uv| = |vu|$.

(a) Suppose $|u| = r$ and $|v| = s$. Then uv will consist of the r letters of u followed by the s letters of v; hence $|uv| = r + s = |u| + |v|$.

(b) Using (a) yields $|uv| = |u| + |v| = |v| + |u| = |vu|$.

5.5. State the difference between the free semigroup on an alphabet A and the free monoid on A.

The free semigroup on A is the set of all nonempty words in A under the operation of concatenation; it does not include the empty word λ. On the other hand, the free monoid on A does include the empty word λ.

LANGUAGES

5.6. Let $A = \{a, b\}$. Describe verbally the following languages over A (which are subsets of A^*):

(a) $L_1 = \{(ab)^m \mid m > 0\}$; (b) $L_2 = \{a^r ba^s ba^t \mid r, s, t \geq 0\}$; (c) $L_3 = \{a^2 b^m a^3 \mid m > 0\}$.

(a) L_1 consisits of words $w = ababab \ldots ab$, that is, beginning with a, alternating with b.
(b) L_2 consists of all words with exactly two b's.
(c) L_3 consists of all words beginning with a^2 and ending with a^3 with one or more b's between them.

5.7. Let $K = \{a, ab, a^2\}$ and $L = \{b^2, aba\}$ be languages over $A = \{a, b\}$. Find: (a) KL; (b) LL.

(a) Concatenate words in K with words in L to obtain $KL = \{ab^2 , a^2ba, ab^3 , ababa, a^2 b^2 , a^3ba\}$.
(b) (*b*) Concatenate words in L with words in L to obtain $LL = \{b^4 , b^2 aba, abab^2 , aba^2 ba\}$.

5.8. Consider the language $L = \{ab, c\}$ over $A = \{a, b, c\}$. Find: (a) L^0; (b) L^3; (c) L^{-2}.

(a) $L^0 = \{\lambda\}$, by definition.
(b) Form all three-word sequences from L to obtain:

$$L^3 = \{ababab, ababc, abcab, abc^2, cabab, cabc, c^2 ab, c^3\}$$

(c) The negative power of a language is not defined.

5.9. Let $A = \{a, b, c\}$. Find L^* where: (a) $L = \{b^2\}$; (b) $L = \{a, b\}$; (c) $L = \{a, b, c^3\}$.

(a) L^* consists of all words b^n where n is even (including the empty word l).
(b) L^* consists of words in a and b.

(c) L^* consists of all words from A with the property that the length of each maximal subword composed entirely of c's is divisible by 3.

5.10 Consider a countable alphabet $A = \{a_1, a_2, \ldots\}$. Let L_k be the language over A consisting of those words w such that the sum of the subscripts of the letters in w is equal to k. (For example, $w = a_2a_3a_3a_6a_4$ belongs to L_{18}.) (a) Find L_4. (b) Show that L_k is finite. (c) Show that A^* is countable. (d) Show that any language over A is countable.

(a) No word in L_4 can have more than four letters, and no letter a_n with $n > 4$ can be used. Thus we obtain the following list:

$$a_1a_1a_1a_1, \quad a_1a_1a_2, \quad a_1a_2a_1, \quad a_2a_1a_1, \quad a_1a_3, \quad a_3a_1, \quad a_2a_2, \quad a_4$$

(b) Only a finite number of the a's, that is, $a_1, a_2, \ldots, a_k$, can be used in L_k and no word in L_k can have more than k letters. Thus L_k is finite.

(c) A^* is the countable union of the finite sets L_k ; hence A^* is countable. (d) L is a subset of the countable set A^*; hence L is also countable.

REGULAR EXPRESSIONS, REGULAR LANGUAGES

5.11. Let $A = \{a, b\}$. Describe the language $L(r)$ where:

(a) $r = abb^*a$; (b) $r = b^*ab^*ab^*$; (c) $r = a^* \vee b^*$; (d) $r = ab^* \wedge a^*$.

(a) $L(r)$ consists of all words beginning and ending in a and enclosing one or more b's. (b) $L(r)$ consists of all words with exactly two a's.

(c) $L(r)$ consists of all words only in a or only in b, that is, $L(r) = \{\lambda, a, a^2, \cdots, b, b^2, \cdots\}$

(c) Here r is not a regular expression since $\wedge$ is not one of the symbols used in forming regular expressions.

5.12. Let $A = \{a, b, c\}$ and let $w = abc$. State whether or not w belongs to $L(r)$ where:

(a) $r = a^* \vee (b \vee c)^*$; (b) $r = a^*(b \vee c)^*$.

(a) No. Here $L(r)$ consists of word in a or words in b and c.
(b) Yes, since a

5.13. Let A = {a, b}. Find a regular expression r such that L(r) consists of all words w where:

(a) w begins with a^2 and ends with b^2; (b) w contains an even number of a's.

(a) Let $r = a^2 (a \vee b)^* b^2$. (Note $(a \vee b)^*$ consists of all words on A.)
(b) Note $s = b^* ab^* ab^*$ consists of all words with exactly two a's. Then let $r = s^* = (b^* ab^* ab^*)^*$.

FINITE AUTOMATA

5.14 Let M be the automaton with the following input set A, state set S with initial state s_0, and accepting ("yes") state set Y :

$$A = \{a, b\}, \quad S = \{s_0, s_1, s_2\}, \quad Y = \{s_2\}$$

Suppose next state function F of M is given by the table in Fig. 5-7(a).

(a) Draw the state diagram D = D(M) of M.

(b) Describe the language L = L(M) accepted by M.

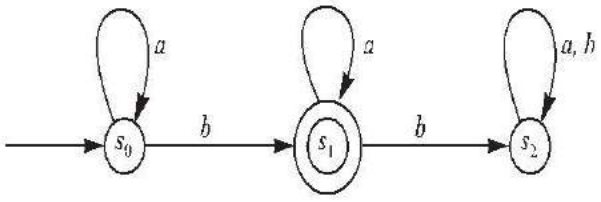
(a) The state diagram D appears in Fig. 5.7(b). The vertices of D are the states, and a double circle indicates an accepting state. If $F(s_j, x) = s_k$, then there is a directed edge from s_j to s_k labeled by the input symbol x. Also, there is a special arrow which terminates at the initial state s_0 .

(b) L(M) consists of all words w with exactly one b. Specifically, if an input

word w has no b's, then it terminates in s_0 and if w has two or more b's then it terminates in s_2 . Otherwise w terminates in s_1 , which is the only accepting state.

F	a	b
s_0	s_0	s_1
s_1	s_1	s_2
s_2	s_2	s_2

(a)



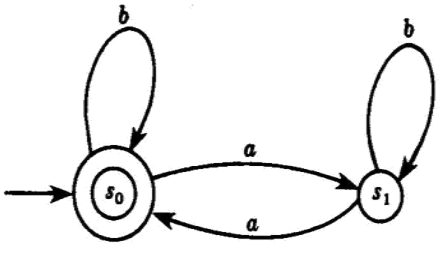
(b)

Fig.5-7

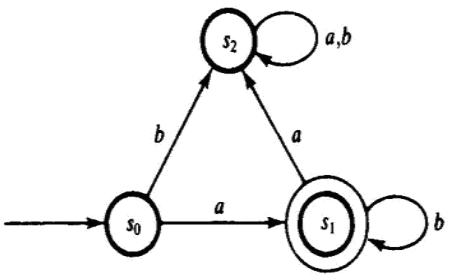
5.15 Let A = {a, b}. Construct an automaton M which will accept precisely those words from A which have an even number of a's. For example, aababbab, aa, bbb, ababaa will be accepted by M, but ababa, aaa, bbabb will be rejected by M.

We need only two states, s_0 and s_1 . We assume that M is in state s_0 or s_1 according as the number of a's up to the given step is even or odd. (Thus s_0 is an accepting state, but s_1 is a rejecting state.) Then only a will change the state. Also, s_0 is the initial state. The state diagram of M is shown in Fig. 5-8(a).

5.16 Let A = {a, b}. Construct an automaton M which will accept those words from A which begin with an a followed by (zero or more) b's. The automaton M appears in Fig. 5-8(b).



(a)



(b)

Fig. 5-8

5.17. Describe the words w in the language L accepted by the automaton M in Fig. 5-9(a).

The system can reach the accepting state s_2 only when there exists an a in w which follows a b.

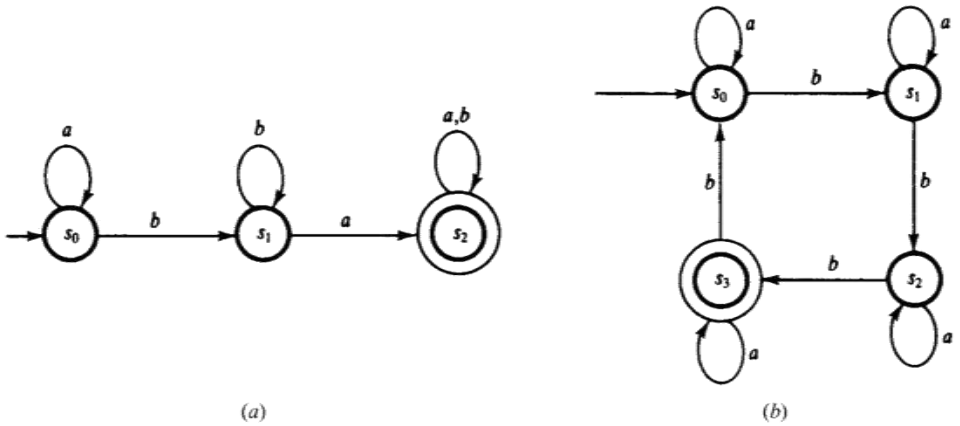


Fig. 5-9

5.18. Describe the words w in the language L accepted by the automaton M in Fig. 5-9(b).

Each a in w does not change the state of the system, whereas each b in w changes the state from R_i , to s_{i+1} (modulo 4). Thus w is accepted by M if the number n of b's in w is congruent to 3 modulo 4, that is, where

$$n = 3, 7, 11, \cdots.$$

5.19 Suppose L is a language over A which is accepted by the automaton M = (A, S, Y, s_0, F). Find an automaton N which accepts L^C , that is, those words from A which do not belong to L.

Simply interchange the accepting and rejecting states in M to obtain N . Then w will be accepted in the new machine

N if and only if w is rejected in M, that is, if and only if w belongs to L^C . Formally, N = (A, S, S\Y, s_0 , F).

5.20 Let $M = (A, S, Y, s_0, F)$ and $M = (A, S', Y', S_0, F')$ be automata over the same alphabet A which accept the languages L(M) and L(M) over A, respectively. Construct an automaton N over A which accepts precisely $L(M) \cap L(M')$.

Let $S \times S'$ be the set of states of N . Let (s, s') be an accepting state of N if both s and s are accepting states in M and M , respectively. Let (s_0, s_0') be the initial state of N . Let the next-state function of N , $G : (S \times S') \times A \rightarrow (S \times S')$, be defined by:

$$G((s, s'), a) = (F(s, a), F(s', a))$$

Then N will accept precisely those words in $L(M) \cap L(M')$.

5.21. Repeat Problem 5.20 except now let N accept precisely $L(M) \cup L(M')$.

Again, let $S \times S'$ be the set of states of N and let (s_0, s_0') be the initial state of N . Now let $(S \times Y') \cup (Y \times S')$ be the accepting states in N . The next-state function G is again defined by

$$G((s, s'), a) = (F(s, a), F(s', a))$$

Then N will accept precisely those words in $L(M) \cup L(M')$.

GRAMMARS

5.22 Define: (a) context-free grammar; (b) regular grammar.

(a) A context-free grammar is the same as a Type 2 grammar, that is, every production is of the form $A \rightarrow \beta$, that is, the left side is a single variable and the right side is a word in one or more symbols.

(b) A regular grammar is the same as a Type 3 grammar, that is, every production is of the form $A \rightarrow a$ or of the form $A \rightarrow aB$, that is, the left side is a single variable and the right side is either a single terminal or a terminal followed by a variable.

5.23 Find the language L(G) generated by the grammar G with variables S, A, B, terminals a, b, and produc-tions S → aB, B → b, B → bA, A → aB.

Observe that we can only use the first production once since the start symbol S does not appear anywhere else. Also, we can only obtain a terminal word by finally using the second production. Otherwise we alternately add a's and b's using the third and fourth productions. Accordingly,

$$L(G) = \{(ab)^n = abababab \cdots ab \mid n \in \mathbf{N}\}$$

5.24. Let L be the language on A = {a, b} which consists of all words w with exactly one b, that is,

$$L = \{b, a^r b, ba^s, a^r ba^s \mid r > 0, s > 0\}$$

(a) Find a regular expression r such that L = L(r).
(b) Find a regular grammar G which generates the language L.
(a) Let $r = a^*ba^*$. Then L(r) = L.
(b) The regular grammar G with the following productions generates L:

$$S \to (b, aA), \quad A \to (b, aA, bB), \quad B \to (a, aB)$$

That is, the letter b can only appear once in any word derived from S. G is regular since it has the required form.

5.25. Let G be the regular grammar with productions: S → aA, A → aB, B → bB, B → A.

(a) Find the derivation tree of the word w = aaba.
(b) Describe all words w in the language L generated by G.

(a) Note first that w can be derived from S as follows:

$$S \Rightarrow aA \Rightarrow a(aB) \Rightarrow aa(bB) \Rightarrow aaba$$

Figure 5-10(a) shows the corresponding derivation tree.

(b) Using the production 1, then 2, then 3, r times, and finally 4 will derive the word $w = aab^r a$ where $r \geq 0$. No other word can be derived from S.

5.26 Figure 5-10(b) is the derivation tree of a word w in the language L of a context-free grammar G. (a) Find w. (b) Which terminals, variables, and productions must lie in G?

(a) The sequence of leaves from left to right yields the word w = ababbbba.

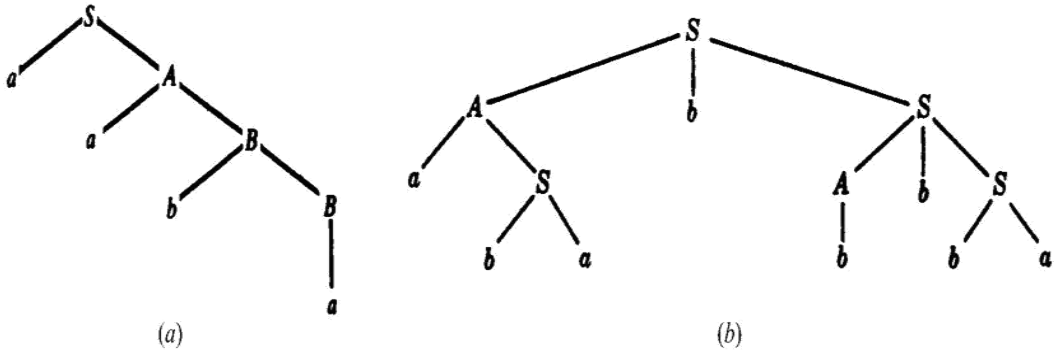


Fig. 5-10

(b) The leaves show that a and b must be terminals, and the internal vertices show that S and A must be variables with S the starting variable. The children of each variable show that $S \rightarrow AbS$, $A \rightarrow aS$, $S \rightarrow ba$, and $A \rightarrow b$ must be productions.

5.27. Does a derivation tree exist for any word w derived from the start symbol S in a grammar G?

No. Derivation trees only exist for Type 2 and 3 grammars, that is, for context-free and regular grammars.

5.28 Determine the type of grammar G which consists of the following productions:

(a) $S \rightarrow aA$, $A \rightarrow aAB$, $B \rightarrow b$, $A \rightarrow a$
(b) $S \rightarrow aAB$, $AB \rightarrow bB$, $B \rightarrow b$, $A \rightarrow aB$
(c) $S \rightarrow aAB$, $AB \rightarrow a$, $A \rightarrow b$, $B \rightarrow AB$
(d) $S \rightarrow aB$, $B \rightarrow bA$, $B \rightarrow b$, $B \rightarrow a$, $A \rightarrow aB$, $A \rightarrow a$

(a) Each production is of the form $A \to \alpha$; hence G is a context-free or Type 2 grammar.
(b) The length of the left side of each production does not exceed the length of the right side; hence G is a Type 1 grammar.
(c) The production $AB \to a$ means G is a Type 0 grammar.
(d) G is a regular or Type 3 grammar since each production has the form $A \to a$ or $A \to aB$.

5.29 Rewrite each grammar G in Problem 5..28 in Backus-Naur form.

The Backus-Naur form only applies to context-free grammars (which includes regular grammars). Thus only (a) and (d) can be written in Backus-Naur form. The form is obtained as follows:

(i) Replace $\to$ by :: =.
(ii) Enclose nonterminals in brackets) *.
(iii) All productions with the same left-hand side are combined in one statement with all the right-hand sides listed on the right of :: = separated by vertical bars.

Accordingly:

(a))S* ::= a)A*,)A* ::= a)A*)B* | a,)B* ::= b
(b))S* ::= a)B*,)B* ::= b)A* | b | a,)A* ::= a)B* | a

PROBLEMS

WORDS

5.30. Consider the words $u = ab^2 a^3$ and $v = aba^2 b^2$. Find: (a) uv; (b) vu; (c) u^2 ; (d) lu; (e) vlv.
5.31 For the words $u = ab^2 a^3$ and $v = aba^2 b^2$, find: |u|, |v|, |uv|, |vu|, and $|v^2|$.
5.32 Let w = abcde. (a) Find all subwords of w. (b) Which of them are initial segments?
5.34 Suppose $u = a_1 a_2 \cdots a_r$ and the a_k are distinct. Find the number n of subwords of u.

LANGUAGES

5.35. Let L = {a^2 , ab} and K = {a, ab, b^2 }. Find: (a) LK; (b) KL; (c) L ∨ K; (d) K ∧ L.

5.36. Let L = {a^2 , ab}. Find: (a) L^0 ; (b) L^2 ; (c) L^3 .

5.37. Let A = {a, b, c}. Describe L* if: (a) L = {a^2 }; (b) L = {a, b^2 }; (c) L = {a, b^2 , c^3 }.

5.38 Does $(L^2)^* = (L^*)^2$? If not, how are they related?

5.39 Consider a countable alphabet A = {a_1 , a_2 , · · ·}. Let L_k the language over A consisting of those words w such that the sum of the subscripts of the letters in w is equal to k. (See Problem 12.10.) Find: (a) L_3 ; (b) L_5 .

REGULAR EXPRESSIONS, REGULAR LANGUAGES

5.39. Let A = {a, b, c}. Describe the language L(r) for each of the following regular expressions:

(a) $r = ab^* c$; (b) $r = (ab \vee c)^*$; (c) $r = ab \vee c^*$.

5.40 Let A = {a, b}. Find a regular expression r such that L® consists of all words w where:

(a) w contains exactly three a's.

(b) The number of a's is divisible by 3.

5.41 Let A = {a, b, c} and let w = ac. State whether or not w belongs to L(r) where:

(a) $r = a^*bc^*$; (b) $r = a^* b^* c$; (c) $r = (ab \vee c)^*$

5.42. Let A = {a, b, c} and let w = abc. State whether or not w belongs to L(r) where:

(a) $r = ab^* (bc)^*$; (b) $r = a^* \vee (b \vee c)^*$; (c) $r = a^* b(bc \vee c^2)^*$.

FINITE AUTOMATA

5.43 Let A = {a, b}. Construct an automaton M such that L(M) will consist of those words w where:

(a) the number of b's is divisible by 3. (b) w begins with a and ends in b.

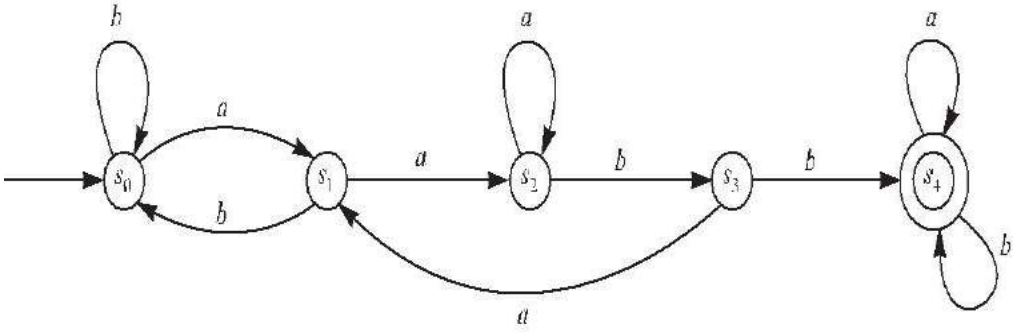
5.44 Let A = {a, b}. Construct an automaton M which will accept the language:

(a) $L(M) = \{b^r ab^s \mid r > 0, s > 0\}$; (b) $L(M) = \{a^r b^s \mid r > 0, s > 0\}$.

5.45 Let A = {a, b}. Construct an automaton M such that L(M) will consist of those words where the number of a's is divisible by 2 and the number of b's is divisible by 3.

(Hint: Use Problems 5.15, 5.43(a), and 5.20.)

5.46 Find the language L(M) accepted by the automaton M in Fig. 5-11



. **Fig. 5-11**

GRAMMARS

5.47. Determine the type of grammar G which consists of the productions:

(a) $S \to aAB$; $S \to AB$; $A \to a$; $B \to b$
(b) $S \to aB$; $B \to AB$; $aA \to b$; $A \to a$; $B \to b$
(c) $S \to aB$; $B \to bB$; $B \to bA$; $A \to a$; $B \to b$

5.48 Find a regular grammar G which generates the language L which consists of all words in a and b such that no two a's appear next to each other.

5.49 Find a context-free grammar G which generates the language L which consists of all words in a and b with twice as many a's as b's.

5.50 Find a grammar G which generates the language L which consists of all words in a and b with an even number of a's.

5.51 Find a grammar G which generates the language L which consists of all words of the form $a^n b a^n$ with $n \geq 0$.

5.52 Show that the language G in Problem 5.51 is not regular. (Hint: Use the Pumping Lemma.)

5.53 Describe the language L = L(G) where G has the productions $S \to aA$, $A \to bbA$, $A \to c$.

5.54 Describe the language L = L(G) where G has the productions $S \to aSb$, $Sb \to bA$, $abA \to c$.

5.55 Write each grammar G in Problem 5.47 in Backus-Naur form.

5.56 Let G be the context-free grammar with productions $S \to (a, aAS)$ and $A \to bS$.

(a) Write G in Backus-Naur form. (b) Find the derivation tree of the word w = abaa.

5.57 Figure 5-12 is the derivation tree of a word w in a language L of a context-free grammar G.

(b) Find w. (b) which terminals, variables, and productions must belong to G?

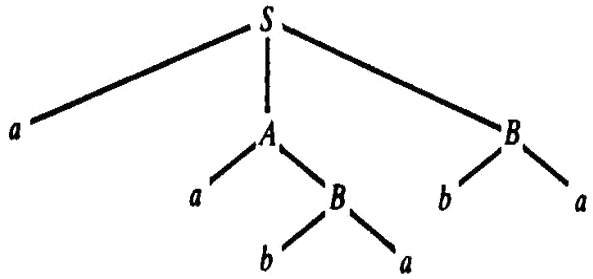


Fig. 5-12

ANSWERS TO THE PROBLEMS

5.30. (a) $uv = ab^2 a^4 ba^2 b^2$; (b) $vu = aba^2 b^2 ab^2 a^3$;
(c) $u^2 = ab^2 a^4 b^2 a^3$; (d) lu = u;
(e) $vlv = v^2 = aba^2 b^2 aba^2 b^2$.

5.31 6, 6, 12, 12, 12.

5.32 (a) l, a, b, c, d, e, ab, bc, cd, de, abc, bcd, cde, abcd, bcde, w = abcde.

(b) l, a, ab, abc, abcd, w = abcde.

5.33. If **u** = l then n = 1; otherwise,
$n = 1 + [r + (r - 1) + \cdots + 2 + 1] = 1 + r(r + 1)/2.$

5.34 (a) $LK = \{a^3 , a^3 b, a^2 b^2 , aba, abab, ab^3 \}$;
(b) $KL = \{a^3 , a^2 b, aba^2 , abab, b^2 a^2 , b^2 ab\}$;
(c) $L \vee K = \{a^2 , ab, a, b^2 \}$; (d) $K \wedge L$ not defined

5.35. (a) 0 ; $L^3 = \{l\}_6$
(b) L^2 a^4 , a^3 b, aba^2 , abab $L^3 = \{l\}_6 53^{=} 2\{343\}$

(c) L2= {a, a b, a ba , a bab, aba, aba b, ababa, ababab}.

5.36 (a) $L* = \{a^n \mid n \text{ is even}\}$. (b) All words w in a and b with only even powers of b.

(c) All words in a, b, c with each power of b even and each power of c a multiple of 3.

5.37 No. $(L^2)^* \subseteq (L^*)^2$.

5.38 (a) $a_1\ a_1\ a_1$, $a_1\ a_2$, $a_2\ a_1\ a_3$ (b) $a_1\ a_1\ a_1\ a_1\ a_1$, $a_1\ a_1\ a_1\ a_2$, $a_1\ a_1\ a_2\ a_1$, $a_1\ a_2\ a_1\ a_1$, $a_2\ a_1\ a_1\ a_1$, $a_1\ a_1\ a_3$, $a_1\ a_3\ a_1$, $a_3\ a_1\ a_1$, $a_2\ a_3$, $a_3\ a_2$, $a_1\ a_4$, $a_4\ a_1$, a_5

5.39 (a) L® = $\{ab^nc \mid n \geq 0\}$. (b) All words in x and c where x = ab. (c) L® = ab ∪ $\{c^n \mid n \geq 0\}$.

5.40 (a) $r = b^* ab^* ab^* ab^*$; (b) $r = (b^*ab^* ab^* ab^*)^*$.

5.41 (a) No; (b) yes; (c) no.

5.42 (a) Yes; (b) no; (c) no.

5.43 See: (a) Fig. 5-13(a); (b) Fig. 5-13(b).

5.44 See: (a) Fig. 5-14; (b) Fig. 5-15(a).

5.55 See: Fig. 5-15(b).

5.56 L(M) consists of all words w which contain aabb as a subword.

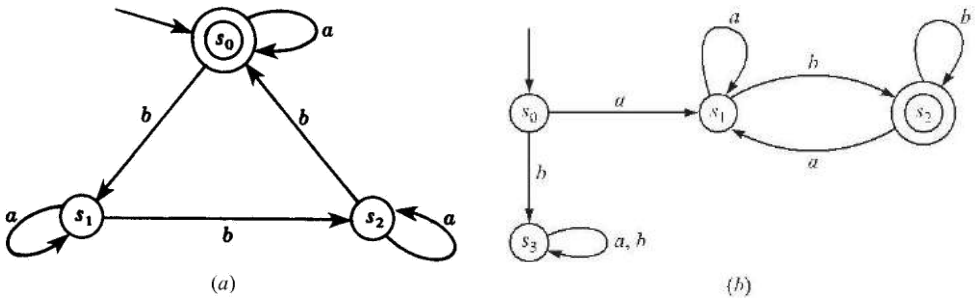


Fig. 5-13

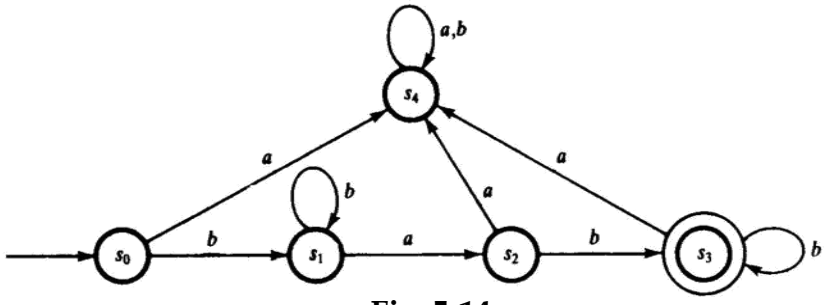


Fig. 5-14

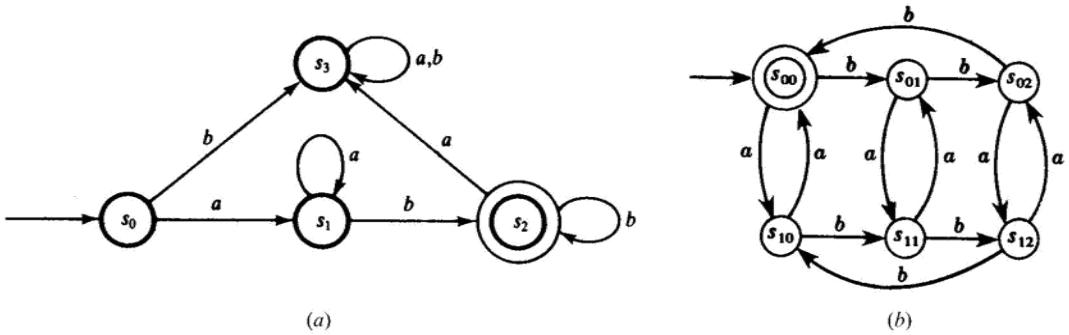


Fig. 5-15

5.47 (a) Type 2; (b) Type 0; (c) Type 3.

5.48 S → (a, b, aB, bA), A → (bA, ab, a, b), B → (b, bA).

5.49 S → (AAB, ABA, BAA), A → (a, BAAA, ABAA, AABA, AAAB), B → (b, BBAA, BABA, aBAAB, ABAB, AABB).

5.50 S → (aA, bB), A → (aB, bA, a), B → (bB, aA, b)

5.51 S → (aSa, b).

5.52 L = $\{ab^{2n}c \mid n \geq 0\}$

L = $\{a^n cb^n \mid n > 0\}$

5.53. (a))S* ::= a)A*)B*|)A*)B*,)A* ::= a,)B* ::= b.

(b) Not defined for Type 0 language.

(c))S* ::= a)B*,)B* ::= b)B* | b)A*,)A* ::= a | b.

5.56. (a))S* ::= a | a)A*)S*,)A* ::= b)S*; (b) See Fig. 12-16.

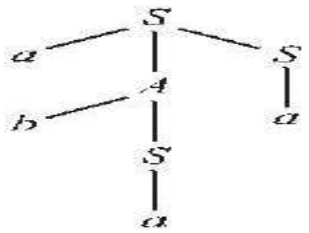


Fig. 5-16

5.57. (a) w = aababa; (b) S → aAB, A → aB, B → ba.

NOTICE

NAIRJC BOOKS INDIA

PUBLISHERS AND DISTRIBUTORS
NAIRJC PUBLISHING HOUSE: 221, GANGOO,
PULWAMA, JAMMU AND KASHMIR, INDIA.
TEL: 0193321815, 9906662570, 9419043159

ISBN 978-81-932604-1-8