CSE106 Discrete Mathematics
Credit Hours: 3 + 0 = 3
Prerequisite: CSE103 Structured Programming

Instructor:
Md. Mozammel Huq Azad Khan, PhD
Professor, CSE Dept., East West University

Text book:

Kenneth H. Rosen, *Discrete Mathematics and Its Applications with Combinatorics and Graph Theory*, 7th Edition, Tata McGraw-Hill Publishing Company Limited, New Delhi, 2015.

Final

Link:
https://drive.google.com/file/d/1YUVkvgOnHGJRwhDyWyYmWNQB0r1wFfl0/view?usp=sharing

Chapter 3

The Fundamentals: Algorithms, the Integers, and Matrices

3.1 Algorithms

Definition 1: An *algorithm* is a finite set of precise instructions for performing a computation or for solving a problem.

Example 1: Describe an algorithm using **pseudocode** for finding the maximum (largest) value in a finite sequence of integers.

ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.
procedure *max* (a_1, a_2, …, a_n: integers)
max := a_1
for *i* := 2 **to** *n*
    if *max* < a_i **then** *max* := a_i
{*max* is the largest element}

Searching Algorithms

Searching problem: Locate an element x in a list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list. The solution to this search problem is the location of the term in the list that equals x (that is, i is the solution if $x = a_i$) and is 0 if x is not in the list.

THE LINEAR SEARCH

ALGORITHM 2 The Linear Search Algorithm.
procedure *linear search* (x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** *location* := i
else *location* := 0
{*location* is the subscript of the term that equals x, or is 0 if x is not found}

THE BINARY SEARCH

The **binary search algorithm** can be used when the list has terms occurring in order of increasing value.

Example 3: Search for 19 in the list

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Index
1	2	3	5	6	7	8	10	12	13	15	16	18	19	20	22	Data

$m = \lfloor (1 + 16)/2 \rfloor = 8$ (middle point)

1	2	3	4	5	6	7	8
1	2	3	5	6	7	8	10

9	10	11	12	13	14	15	16
12	13	15	16	18	19	20	22

Since 19 > 10, solution is in the second portion

9	10	11	12	13	14	15	16
12	13	15	16	18	19	20	22

$m = \lfloor (9 + 16)/2 \rfloor = 12$

9	10	11	12
12	13	15	16

13	14	15	16
18	19	20	22

Since 19 > 16, solution is in the second portion

Example 3 (contd.): Search for 19 in the list

13	14	15	16
18	19	20	22

$m = \lfloor (13 + 16)/2 \rfloor = 14$

13	14
18	19

15	16
20	22

Since 19 not > 19, solution is in the first portion

13	14
18	19

$m = \lfloor (13 + 14)/2 \rfloor = 13$

13
18

14
19

Since 19 > 18, solution is in the second portion

14
19

19 is the14th term of the list. Return value is 14.

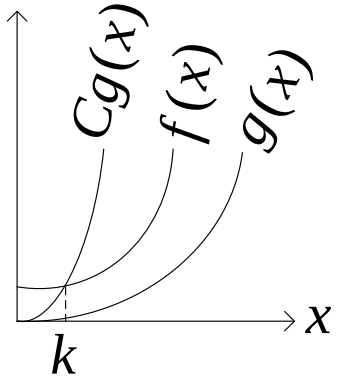
ALGORITHM 3 The Binary Search Algorithm.

procedure *binary search* (x: integer, a_1, a_2, …, a_n: increasing integers)
$i := 1$ {i is left endpoint of search interval}
$j := n$ {j is right endpoint of search internal}
while $i < j$
begin
$m := \lfloor (i + j)/2 \rfloor$ { m is middle point of search interval}
if $x > a_m$ **then** $i := m + 1$
else $j := m$
end
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
{*location* is the subscript of the term that equals x, or is 0 if x is not found}

3.2 The Growth of Functions

Big-*O* Notation

Definition 1: Let f and g be functions from the set of non-negative integers or the set of non-negative real numbers to the set of non-negative real numbers. We say that $f(x) = O(g(x))$ if there are constants C and k such that $f(x) \leq C\, g(x)$, whenever $x \geq k$.



The constants C and k in the definition of big-O notation are called **witness** to the relationship $f(x) = O(g(x))$.

To establish that $f(x) = O(g(x))$ we need only one pair of witnesses to this relationship.

Example 1: Show that $f(x) = x^2 + 2x + 1 = O(x^2)$.

Solution:

$f(x) = x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2$ $\quad [x \leq x^2, 1 \leq x^2 \text{ for } x \geq 1]$

$= 4x^2$

When $k = 1$, $C = 4$, and $g(x) = x^2$, $f(x) = x^2 + 2x + 1 \leq 4x^2$.

$\therefore f(x) = x^2 + 2x + 1 = O(x^2)$.

When big-*O* notation is used, the function g in the relationship $f(x) = O(g(x))$ is chosen to be **as small as possible** (sometimes from a set of reference functions)

Some Important Big-*O* Results

Theorem 1: Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ \ldots + a_1 x + a_0$, where $a_0, a_1, \ldots, a_{n-1}, a_n$ are real numbers with $a_n \neq 0$. Then $f(x) = O(x^n)$.

Proof: Omitted

Example 5: How can big-O notation be used to estimate the sum of the first n positive integers?

Solution:
$1 + 2 + \ldots + n \leq n + n + \ldots + n = n^2$
For $k = 1$ and $C = 1$
$1 + 2 + \ldots + n = O(n^2)$

Example 6: Give big-O estimate for the factorial function and the logarithm of the factorial function, where the factorial function $f(n) = n!$ is defined by $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$, whenever n is a positive integer, and $1! = 1$.

Solution:
$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leq n \cdot n \cdot n \cdot \ldots \cdot n = n^n$
For $k = 1$ and $C = 1$
$n! = O(n^n)$

$n! \leq n^n$
or, $\log n! \leq \log n^n = n \log n$
For $k = 1$ and $C = 1$
$\log n! = O(n \log n)$

Big-*O* notation is used to estimate the number of operations needed to solve a problem using a specified procedure or algorithm. The functions used in these estimates often include the following:

1, log$n$, n, $n$log$n$, n^2, 2^n, n!

Each function in the list is smaller than the succeeding function.

Example:
Let $f(x)$ = c, where c is a positive constant
$f(x)$ = c ≤ c·1, where k = 0, C = c, and $g(x)$ = 1
∴ $f(x)$ = c = $O(1)$

Big-*O* of any constant is $O(1)$.

The Growth of Combination of Functions

Theorem 2: Suppose that $f_1(x) = O(g_1(x))$ and $f_2(x) = O(g_2(x))$. Then $(f_1 + f_2)(x) = O(\max(g_1(x), g_2(x))$

Proof: Omitted

Corollary 1: Suppose that $f_1(x) = O(g(x))$ and $f_2(x) = O(g(x))$. Then $(f_1 + f_2)(x) = O(g(x))$.

Theorem 3: Suppose that $f_1(x) = O(g_1(x))$ and $f_2(x) = O(g_2(x))$. Then $(f_1 f_2)(x) = O(g_1(x) g_2(x))$.

Proof: Omitted

Example 8: Give a big-O estimate for $f(n) = 3n\log(n!) + (n^2 + 3)\log n$, where n is a positive integer.

Solution:

$3n = O(n)$

$\log(n!) = O(n\log n)$

$\therefore\ 3n\log(n!) = O(n\ (n\log n)) = O(n^2\log n)$

$(n^2 + 3) = O(n^2)$

$\log n = O(\log n)$

$\therefore\ (n^2 + 3)\log n = O(n^2\ (\log n)) = O(n^2\log n)$

$\therefore\ f(n) = 3n\log(n!) + (n^2 + 3)\log n$

$= O(\max(n^2\log n, n^2\log n)) = O(n^2\log n)$

Example 9: Give big-O estimate for $f(x) = (x + 1)\log(x^2 + 1) + 3x^2$

Solution:

$(x + 1) = O(x)$

$$\begin{aligned}\log(x^2 + 1) &\le \log(x^2 + x^2) \text{ [when } x \ge 1] \\ &= \log(2x^2) \\ &= \log 2 + \log x^2 \\ &= \log 2 + 2\log x \\ &\le \log x + 2\log x \text{ [when } x \ge 2] \\ &= 3\log x\end{aligned}$$

$\therefore \log(x^2 + 1) = O(\log x)$ [$k = 2$ and $C = 3$]

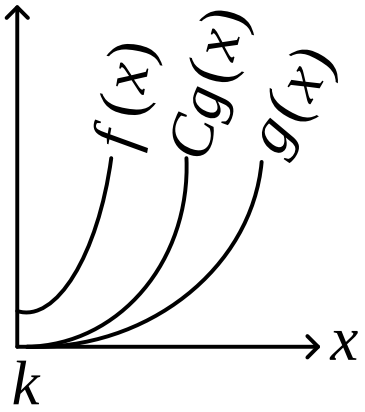
$\therefore (x + 1)\log(x^2 + 1) = O(x\log x)$

$3x^2 = O(x^2)$

$\therefore f(x) = (x + 1)\log(x^2 + 1) + 3x^2 = O(\max(x\log x, x^2) = O(x^2)$

Big-Omega and Big-Theta Notation

Definition 2: Let f and g be functions from the set of non-negative integers or the set of non-negative real numbers to the set of non-negative real numbers. We say that $f(x) = \Omega(g(x))$ if there are constants C and k such that $f(x) \geq C\, g(x)$, where $x \geq k$.



Example 10:

$f(x) = 8x^3 + 5x^2 + 7 \geq 8x^3$ [when $x \geq 0$]

$\therefore f(x) = 8x^3 + 5x^2 + 7 = \Omega(x^3)$ [$k = 0$ and $C = 8$]

Definition 3: Let f and g be functions from the set of non-negative integers or the set of non-negative real numbers to the set of non-negative real numbers. We say that $f(x) = \theta(g(x))$ if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.

Example 12: Show that $3x^2 + 8x\log x = \theta(x^2)$.

Solution:
$3x^2 + 8x\log x \leq 3x^2 + 8xx = 3x^2 + 8x^2 = 11x^2$ [when $x \geq 1$]
$\therefore 3x^2 + 8x\log x = O(x^2)$ [$k = 1$ and $C = 11$]

$3x^2 + 8x\log x \geq 3x^2$ [when $x \geq 1$]
$\therefore\ 3x^2 + 8x\log x = \Omega(x^2)$ [$k = 1$ and $C = 3$]

$\therefore\ 3x^2 + 8x\log x = \theta(x^2)$

Theorem 4: Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ \ldots + a_1 x + a_0$, where $a_0, a_1, \ldots, a_{n-1}, a_n$ are real numbers with $a_n \neq 0$. Then $f(x) = \theta(x^n)$.

Example 13:
$3x^8 + 10x^7 + 221x^2 + 1444 = \theta(x^8)$
$x^{19} - 18x^4 - 10112 = \theta(x^{19})$

3.3 Complexity of Algorithms

Time Complexity

The **time complexity** of an algorithm is an estimate of the time required by the algorithm to solve a problem of a particular size.

The time complexity of an algorithm can be expressed in terms of the number of operations used by the algorithm when the input has a particular size.

WORST-CASE COMPLEXITY

By the **worst-case** complexity of an algorithm, we mean the largest number of operations needed to solve the given problem using this algorithm on input of specified size.

AVERAGE-CASE COMPLEXITY

In **average-case** complexity, the average number of operations used to solve the problem over all inputs of a given size is found.

Example 1: Describe the time complexity of Algorithm 1 for finding the maximum element in a set.

Solution:
ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

procedure *max* (a_1, a_2, …, a_n: integers)
max := a_1
For *i* := 2 **to** *n*
if *max* < a_i **then** *max* := a_i
{*max* is the largest element}

The **for** loop will be executed (n – 1) times.
For each **for** loop, two comparisons will be done.
One more comparison will be needed to exit the **for** loop.
Therefore, exactly $2(n - 1) + 1 = 2n - 1$ comparisons are used.
Hence, the average-case time complexity of the algorithm is $2n - 1 = \theta(n)$.

Example 2: Describe the worst-case time complexity of the linear search algorithm.

Solution:
ALGORITHM 2 The Linear Search Algorithm.

procedure *linear search* (x: integer, a_1, a_2, …, a_n: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
$i := i + 1$
if $i \leq n$ **then** *location* := i
else *location* := 0
{*location* is the subscript of the term that equals x, or is 0 if x is not found}

At each step of the **while** loop, two comparisons are performed.
One more comparison is made outside the loop.

If $x = a_i$, then $2i + 1$ comparisons are used.

When the element is not in the list, $2n + 1 + 1 = 2n + 2$ comparisons are required.

Hence, the worst-case time complexity of linear search algorithm is $2n + 2 = O(n)$.

Example 3: Describe the time complexity of the binary search algorithm.

Solution:

ALGORITHM 3 The Binary Search Algorithm.

procedure *binary search* (x: integer, a_1, a_2, …, a_n: increasing integers)
i:= 1 {i is left endpoint of search interval}
j := n {j is right endpoint of search internal}
while $i < j$
begin
$m := \lfloor (i + j)/2 \rfloor$ { m is middle point of search interval}
if $x > a_m$ **then** $i := m + 1$
else $j := m$
end
if $x = a_i$ **then** *location* := i
else *location* := 0
{*location* is the subscript of the term that equals x, or is 0 if x is not found}

Example 3: Describe the time complexity of the binary search algorithm. (Contd.)

Assume $n = 2^k$, where k is a nonnegative integer
$\therefore k = \log n$

Search Interval Length	Comparisons
2^k	2
2^{k-1}	2
2^{k-2}	2
…	…
2^2	2
2^1	2
$2^0 = 1$	1

One more comparison is used to determine if this item is x.

Thus, binary search algorithm requires
$2k + 1 + 1 = 2k + 2 = 2\log n + 2$ comparisons.

Hence, the average-case time complexity of binary search is $2\log n + 2 = \theta(\log n)$.

Example 4: Describe the average-case complexity of the linear search algorithm, assuming that the element x is in the list.

Solution:
ALGORITHM 2 The Linear Search Algorithm.

procedure *linear search* (x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
If $i \leq n$ **then** *location* $:= i$
else *location* $:= 0$
{*location* is the subscript of the term that equals x, or is 0 if x is not found}

Location of x	Comparison needed
1	$2\times1 + 1 = 3$
2	$2\times2 + 1 = 5$
…	…
n	$2n + 1$

Example 4: Describe the average-case complexity of the linear search algorithm, assuming that the element x is in the list. (contd.)

Thus, the average number of comparisons used by the linear search algorithm (when x is known to be in the list) is

$$\frac{(2\times 1+1)+(2\times 2+1)+\cdots+(2n+1)}{n} = \frac{2(1+2+\cdots+n)+(1+1+\cdots+1)}{n} = \frac{2\left[\frac{n(n+1)}{2}\right]+n}{n} = (n+1)+1 = n+2$$

Hence, the average-case time complexity of linear-search algorithm is $n + 2 = \theta(n)$.

3.4 The Integer and Division

Theorem 2: THE DIVISION ALGORITHM Let a be an integer and d a positive integer. Then there are unique integers q and r, with $0 \leq r < d$, such that $a = dq + r$

Definition 2: In the equality given in the division algorithm, d is called the *divisor*, a is called the *dividend*, q is called the *quotient*, and r is called the *remainder*. These notations are used to express the quotient and remainder:
$q = a$ **div** d, $r = a$ **mod** d.

Example 3: What are the quotient and remainder when 101 is divided by 11?
Solution:
$101 = 11 \cdot 9 + 2$
The quotient is 101 **div** 11 = 9
The remainder is 101 **mod** 11 = 2

Example 4: What are the quotient and remainder when –11 is divided by 3?
Solution:
$-11 = 3(-4) + 1$
The quotient is –11 **div** 3 = –4
The remainder is –11 **mod** 3 = 1

Modular Arithmetic

Definition 3: If a and b are integers and m is a positive integer, then a is *congruent to b modulo m* if m divides $a - b$. We use the notation $a \equiv b \pmod{m}$ to indicate that a is congruent to b modulo m.
If a and b are not congruent modulo m, we write $a \not\equiv b \pmod{m}$.

Theorem 3: Let a and b be integers, and let m be a positive integer. Then $a \equiv b \pmod{m}$ if and only if $a \bmod m = b \bmod m$.

Example 5: Determine whether 17 is congruent to 5 modulo 6 and whether 24 and 14 are congruent modulo 6.

Solution:
6 divides $(17 - 5) = 12$. $\therefore 17 \equiv 5 \pmod{6}$
6 does not divide $(24 - 14) = 10$. $\therefore 24 \not\equiv 14 \pmod{6}$

Theorem 4: Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that $a = b + km$.

Proof: Omitted.

The set of all integers congruent to an integer a modulo m is called the **congruence class** of a modulo m.

Theorem 5: Let m be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then $a + c \equiv b + d \pmod{m}$ and $ac \equiv bd \pmod{m}$.

Proof: Omitted.

Example 6: Because $7 \equiv 2 \pmod{5}$ and $11 \equiv 1 \pmod{5}$, it follows from Theorem 5 that
$7 + 11 = 18 \equiv 2 + 1 = 3 \pmod{5}$
$7 \cdot 11 = 77 \equiv 2 \cdot 1 = 2 \pmod{5}$

Corollary 2: Let m be a positive integer and let a and b be integers. Then
$(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$
and
$ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m$.

Proof: Omitted.

Example:
$(8 + 14) \bmod 5 = 22 \bmod 5 = 2$
$((8 \bmod 5) + (14 \bmod 5)) \bmod 5 = (3 + 4) \bmod 5 = 7 \bmod 5 = 2$

$(8 \cdot 14) \bmod 5 = 112 \bmod 5 = 2$
$((8 \bmod 5)(14 \bmod 5)) \bmod 5 = (3 \cdot 4) \bmod 5 = 12 \bmod 5 = 2$

Application of Congruences

Example 8: Pseudorandom Numbers
The most commonly used procedure for generating pseudorandom numbers is the **linear congruential method**.

We choose four integers: the **modulus** m, **multiplier** a, **increment** c, and **seed** x_0, with $2 \leq a < m$, $0 \leq c < m$, and $0 \leq x_0 < m$.

We generate a sequence of pseudorandom numbers $\{x_n\}$, with $0 \leq x_n < m$ for all n, by successively using the congruence
$x_{n+1} = (ax_n + c)$ **mod** m.

Example 8: Pseudorandom Numbers (contd.)

Let $m = 9$, $a = 7$, $c = 4$, and $x_0 = 3$

$x_{n+1} = (ax_n + c)$ **mod** m

$x_0 = 3$
$x_1 = (7x_0 + 4) \bmod 9 = (7 \cdot 3 + 4) \bmod 9 = 25 \bmod 9 = 7$
$x_2 = (7x_1 + 4) \bmod 9 = (7 \cdot 7 + 4) \bmod 9 = 53 \bmod 9 = 8$
$x_3 = (7x_2 + 4) \bmod 9 = (7 \cdot 8 + 4) \bmod 9 = 60 \bmod 9 = 6$
$x_4 = (7x_3 + 4) \bmod 9 = (7 \cdot 6 + 4) \bmod 9 = 46 \bmod 9 = 1$
$x_5 = (7x_4 + 4) \bmod 9 = (7 \cdot 1 + 4) \bmod 9 = 11 \bmod 9 = 2$
$x_6 = (7x_5 + 4) \bmod 9 = (7 \cdot 2 + 4) \bmod 9 = 18 \bmod 9 = 0$
$x_7 = (7x_6 + 4) \bmod 9 = (7 \cdot 0 + 4) \bmod 9 = 4 \bmod 9 = 4$
$x_8 = (7x_7 + 4) \bmod 9 = (7 \cdot 4 + 4) \bmod 9 = 32 \bmod 9 = 5$

$x_9 = (7x_8 + 4) \bmod 9 = (7 \cdot 5 + 4) \bmod 9 = 39 \bmod 9 = 3$
$x_{10} = (7x_9 + 4) \bmod 9 = (7 \cdot 3 + 4) \bmod 9 = 25 \bmod 9 = 7$

3.5 Primes and Greatest Common Divisor

Definition 1: A positive integer *p* greater than 1 is called a *prime* if the only positive factors of *p* are 1 and *p*. A positive integer that is greater than 1 and is not prime is called *composite*.

Example 1: The integer 7 is prime since its only positive factors are 1 and 7, whereas the integer 9 is composite since it is divisible by 3.

Theorem 1:THE FUNDAMENTAL THEOREM OF ARITHMETIC Every positive integer greater than 1 can be written uniquely as a prime or as the product of two or more primes where the prime factors are written in order of non-decreasing size.

Proof: Omitted.

Example 2: The prime factorizations of 100, 641, 999, and 1024 are given by

$100 = 2 \cdot 2 \cdot 5 \cdot 5 = 2^2 \cdot 5^2$

$641 = 641$

$999 = 3 \cdot 3 \cdot 3 \cdot 37 = 3^3 \cdot 37^1$

$1024 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^{10}$

Theorem 2: If n is a composite integer, then n has at least one prime divisor less than or equal to $\sqrt{n}$.

Proof. Omitted.

Example 3: Show that 101 is prime.

Solution:
The only primes not exceeding $\sqrt{101} = 10$ are 2, 3, 5, and 7. Since 101 is not divisible by 2, 3, 5, or 7, it follows that 101 is prime.

Example 4: Find the prime factorization of 7007.

Solution:
First, perform divisions of 7007 by successive primes, beginning with 2 but not exceeding $\sqrt{7007} = 83$:
$7007 / 7 = 1001$

Next, divide 1001 by successive primes, beginning with 7 but not exceeding $\sqrt{1001} = 31$:
$1001 / 7 = 143$

Next, divide 143 by successive primes, beginning with 7 but not exceeding $\sqrt{143} = 11$:
$143 / 11 = 13$

Next, divide 13 by successive primes, beginning with 11 but not exceeding $\sqrt{13} = 3$, which is not possible. The procedure is completed.

Therefore, $7007 = 7 \cdot 7 \cdot 11 \cdot 13 = 7^2 \cdot 11^1 \cdot 13^1$

Greatest Common Divisors and Least Common Multiples

Definition 2: Let a and b be integers, not both zero. The largest integer d such that $d \mid a$ and $d \mid b$ is called the *greatest common divisor* of a and b. The greatest common divisor of a and b is denoted by $\gcd(a, b)$.

Example 10: What is the greatest common divisor of 24 and 36?

Solution:
The positive common divisors of 24 and 36 are 1, 2, 3, 4, 6, and 12. Hence, $\gcd(24, 36) = 12$.

Example 11: What is the greatest common divisor of 17 and 22?

Solution:
The only common divisor of 17 and 22 is 1, hence $\gcd(17, 22) = 1$.

Definition 3: The integers a and b are *relatively prime* if their greatest common divisor is 1.

Example 12: The integers 17 and 22 are relatively prime, since $\gcd(17, 22) = 1$.

Definition 4: The integers $a_1, a_2, \ldots, a_n$ are *pair wise relatively prime* if $\gcd(a_i, a_j) = 1$ whenever $1 \leq i < j \leq n$.

Example 13: Determine whether the integers 10, 17, and 21 are pairwise relatively prime and whether the integers 10, 19, and 24 are pairwise relatively prime.

Solution:
gcd(10, 17) = 1, gcd(10, 21) = 1, and gcd(17, 21) = 1. Therefore, 10, 17, and 21 are pairwise relatively prime.

gcd(10, 24) = 2 > 1. Therefore, 10, 19, and 24 are not pairwise relatively prime.

One way to find the greatest common divisor of two integers is to use the prime factorizations of these integers.

Suppose that the prime factorizations of the integers a and b, neither equal to zero, are

$$a = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}$$

$$b = p_1^{b_1} p_2^{b_2} \cdots p_n^{b_n}$$

where exponent is a nonnegative integer, and where all primes occurring in the prime factorization of either a or b are included in both factorization, with zero exponents if necessary. Then

$$\gcd(a,b) = p_1^{\min(a_1,b_1)} p_2^{\min(a_2,b_2)} \cdots p_n^{\min(a_n,b_n)}$$

Example 14: The prime factorization of 120 and 500 are

$120 = 2^3 \cdot 3^1 \cdot 5^1$
$500 = 2^2 \cdot 3^0 \cdot 5^3$

$\gcd(120, 500) = 2^{\min(3, 2)}\ 3^{\min(1, 0)}\ 5^{\min(1, 3)} = 2^2\ 3^0\ 5^1 = 20$

Definition 5: The least common multiple of the positive integers *a* and *b* is the smallest positive integer that is divisible by both *a* and *b*. The least common multiple of *a* and *b* is denoted by lcm(*a*, *b*).

Suppose that the prime factorizations of the integers *a* and *b*, neither equal to zero, are

$$a = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}$$

$$b = p_1^{b_1} p_2^{b_2} \cdots p_n^{b_n}$$

where exponent is a nonnegative integer, and where all primes occurring in the prime factorization of either *a* or *b* are included in both factorization, with zero exponents if necessary. Then

$$lcm(a,b) = p_1^{\max(a_1,b_1)} p_2^{\max(a_2,b_2)} \cdots p_n^{\max(a_n,b_n)}$$

Example 15: What is the least common multiple of $95256 = 2^3 \cdot 3^5 \cdot 7^2$ and $432 = 2^4 \cdot 3^3$?

Solution:
$\text{lcm}(2^3 \cdot 3^5 \cdot 7^2, 2^4 \cdot 3^3) = \text{lcm}(2^3 \cdot 3^5 \cdot 7^2, 2^4 \cdot 3^3 \cdot 7^0) = 2^{\max(3,4)} 3^{\max(5,3)} 7^{\max(2,0)} = 2^4 \cdot 3^5 \cdot 7^2 = 190512$

Chapter 8
Graphs

8.1 Graphs and Graph Models

Definition 1: A *graph* $G = (V, E)$ consists of V, a nonempty set of *vertices* (or *nodes*) and E, a set of *edges*. Each edge has either one or two vertices associated with it, called its *endpoints*. An edge is said to *connect* its end points.

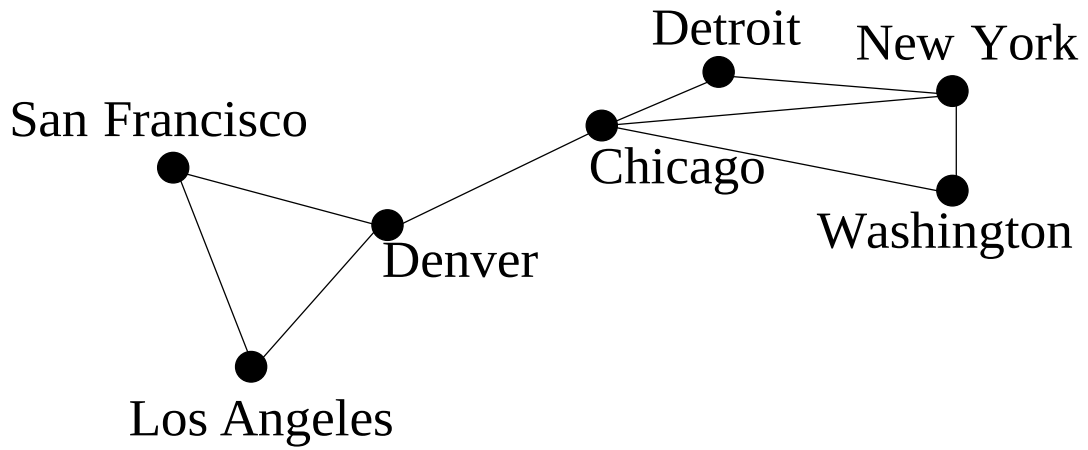


Figure 1. A computer network represented by a graph.

When there is an edge of a graph associated to $\{u, v\}$, we can say that $\{u, v\}$ is an edge of the graph.

If an edge connects a vertex to itself, it is called a **loop**.

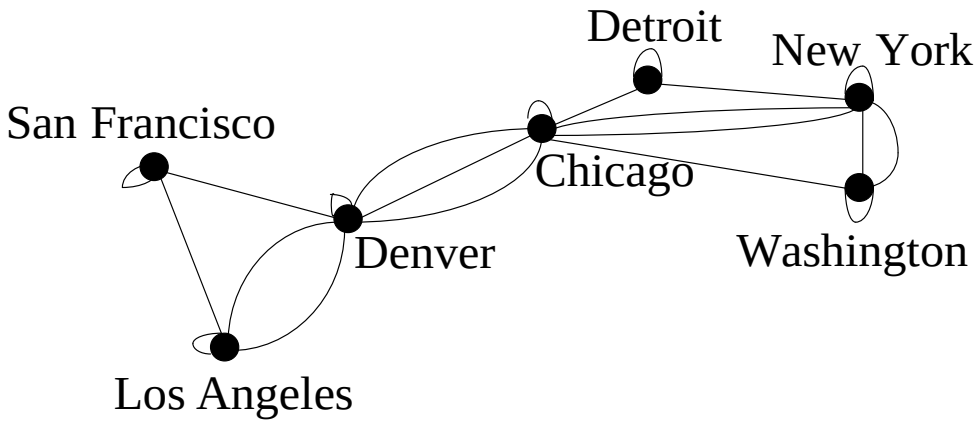


Figure 3. A graph representing a computer network with multiple edges and loops.

So far the graphs we have introduced are **undirected graphs**. Their edges are also said to be **undirected**.

Definition 2: A *directed graph* (or *digraph*) $G = (V, E)$ consists of a nonempty set of vertices V and a set of *directed edges* (or *arcs*) E. Each directed edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (u, v) is said to *start* at u and *end* at v.

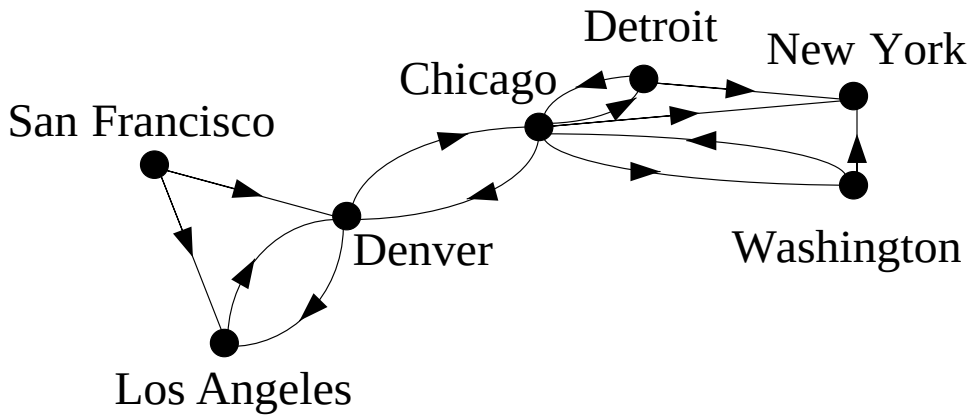


Figure 4. A graph representing a communication network with one-way communications links.

We use an arrow pointing from u to v to indicate the direction of the edge that starts at u and ends at v.

We call (u, v) an edge if there is an edge associated to it in the graph.

A graph with both directed and undirected edges is called a **mixed graph**.

8.2 Graph Terminology and Special Types of Graphs

Definition 1: Two vertices *u* and *v* in an undirected graph *G* are called *adjacent* (or *neighbors*) in *G* if *u* and *v* are endpoints of an edge of *G*. If *e* is associated with {*u*, *v*}, the edge *e* is called *incident with* the vertices *u* and *v*. The edge *e* is also said to *connect u* and *v*. The vertices *u* and *v* are called *endpoints* of an edge associated with {*u*, *v*}.

Definition 2: The *degree of a vertex in an undirected graph* is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex *v* is denoted by deg(*v*).

Example 1: What are the degrees of the vertices in the graph *G* and *H* displayed in Figure 1?

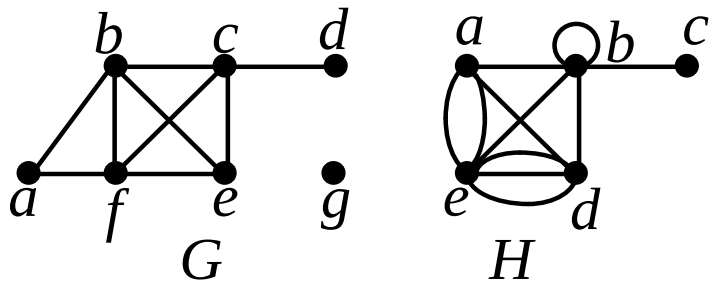


Figure 1. The Undirected Graphs *G* and *H*.

Solution:
In *G*, deg(*a*) = 2, deg(*b*) = 4, deg(*c*) = 4, deg(*d*) = 1, deg(*e*) = 3, deg(*f*) = 4, deg (*g*) = 0

In *H*, deg(*a*) = 4, deg(*b*) = 6, deg(*c*) =1, deg(*d*) = 5, deg(*e*) = 6

A vertex of degree zero is called **isolated**. A vertex of degree one is called **pendant**.

Theorem 1: THE HANDSHAKING THEOREM
Let $G = (V, E)$ be an undirected graph with $|E|$ edges. Then

$$2|E| = \sum_{v \in V} \deg(v)$$

Example 3: How many edges are there in a graph with 10 vertices each of degree six?

Solution:
Sum of degrees of the vertices = $6 \cdot 10 = 60$

$$2|E| = 60$$

$$\therefore |E| = \frac{60}{2} = 30$$

Definition 3: When (u, v) is an edge of the graph G with directed edges, u is said to be *adjacent to* v and v is said to be *adjacent from* u. The vertex u is called the *initial* or *start vertex* of (u, v), and v is called the *terminal* or *end vertex* of (u, v). The initial vertex and terminal vertex of a loop are the same.

Definition 4: In a graph with directed edges the *in-degree of a vertex v*, denoted by $\deg^-(v)$, is the number of edges with v as their terminal vertex. The *out-degree of v*, denoted by $\deg^+(v)$, is the number of edges with v as their initial vertex. (Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of this vertex.)

Example 4: Find the in-degree and out-degree of each vertex in the graph G with directed edges shown in Figure 2.

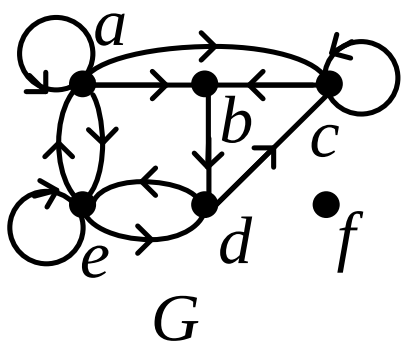


Figure 2. The Directed Graph G.

Solution:
The in-degree in G are: $\deg^-(a) = 2$, $\deg^-(b) = 2$, $\deg^-(c) = 3$, $\deg^-(d) = 2$, $\deg^-(e) = 3$, $\deg^-(f) = 0$
The out-degree in G are: $\deg^+(a) = 4$, $\deg^+(b) = 1$, $\deg^+(c) = 2$, $\deg^+(d) = 2$, $\deg^+(e) = 3$, $\deg^+(f) = 0$

Theorem 3: Let $G = (V, E)$ be a graph with directed edges. Then

$$\sum_{v \in V} \deg^{-}(v) = \sum_{v \in V} \deg^{+}(v) = |E|$$

The undirected graph that results from ignoring directions of edges is called the **underlying undirected graph**.

Some Special Graphs

Example 5: Complete Graphs

The **complete graph on *n* vertices**, $n \geq 1$, denoted by K_n, is the graph that contains exactly one edge between each pair of distinct vertices.

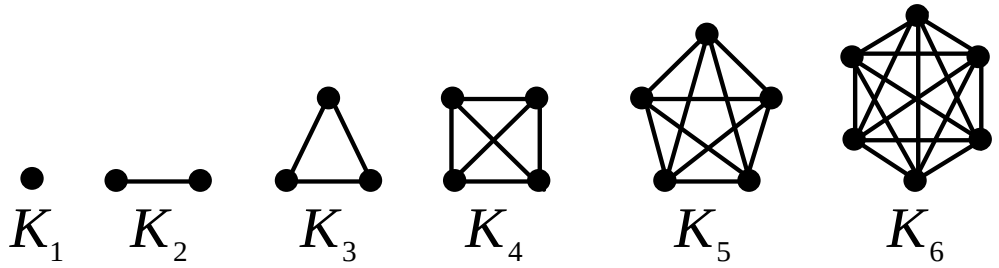


Figure 3. The Graphs K_n for $1 \leq n \leq 6$.

Example 6: Cycles

The **cycle** C_n, $n \geq 3$, consists of n vertices $v_1, v_2, \ldots, v_n$ and edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}$, and $\{v_n, v_1\}$.

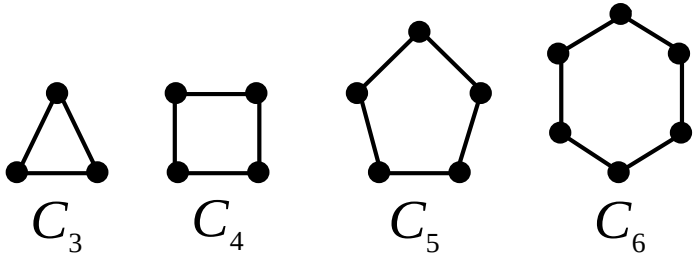


Figure 4. The Cycles C_3, C_4, C_5, and C_6.

Example 7: Wheels

We obtain the **wheel** W_n when we add an additional vertex to the cycle C_n, for $n \geq 3$, and connect this new vertex to each of the n vertices in C_n, by new edges.

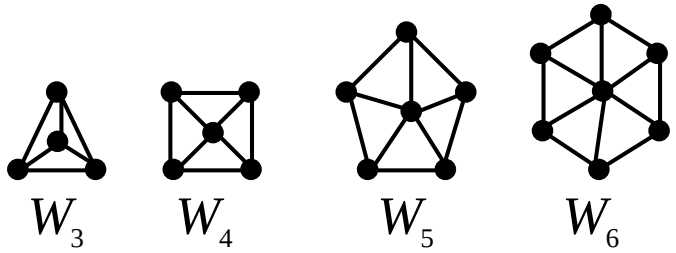


Figure 5. The Wheels W_3, W_4, W_5, and W_6.

Bipartite Graphs

Definition 5: A simple graph G is called *bipartite* if its vertex set V can be portioned into two disjoint sets V_1 and V_2 such that every edge in the graph connects a vertex in V_1 and a vertex in V_2 (so that no edge in G connects either two vertices in V_1 or two vertices in V_2). When this condition holds, we call the pair (V_1, V_2) a *bipartition* of the vertex set V of G.

Example 9: C_6 is bipartite as shown in Figure 7.

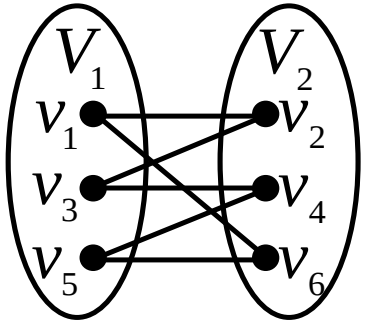
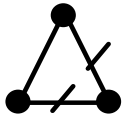


Figure 7. Showing that C_6 is Bipartite.

Example 10: K_3 is not bipartite.



K_3

Example 11. Are the graphs G and H displayed in Figure 8 bipartite?

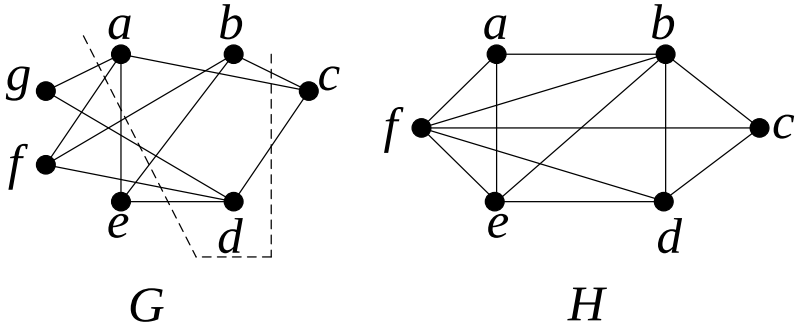


Figure 8. The Undirected Graphs G and H.

Solution:
Graph G is bipartite, since its vertex set is the union of two disjoint subsets, $\{a, b, d\}$ and $\{c, e, f, g\}$, and each edge connects a vertex in one of these subsets to a vertex in the other subset.

Graph H is not bipartite, since its vertex set cannot be partitioned into two subsets so that edges do not connect two vertices from the same subset.

8.3 Representing Graphs and Graph Isomorphism

Representing Graphs

Adjacency Lists

One way to represent a graph is to use **adjacency lists**, which specify the vertices that are adjacent to each vertex of the graph.

Example 1: Use adjacency lists to describe the graph given in Figure 1.

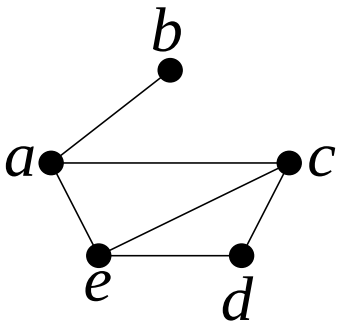


Figure 1

Solution:

Table 1. An Edge List for a Graph.

Vertex	Adjacent vertices
a	*b, c, e*
b	*a*
c	*a, d, e*
d	*c, e*
e	*a, c, d*

Example 2: Represent the directed graph shown in Figure 2 by listing all the vertices that are the terminal vertices of edges starting at each vertex of the graph.

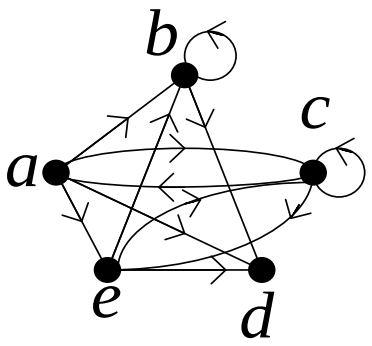


Figure 2. A Directed Graph.

Solution:
Table 2. An Edge List for a Directed Graph.

Initial vertex	Terminal Veritces
a	*b, c, d, e*
b	*b, d*
c	*a, c, e*
d	
e	*b, c, d*

Adjacency Matrices

Suppose that $G = (V, E)$ is a graph where $|V| = n$. Suppose that the vertices of G are listed arbitrarily as $v_1, v_2, \ldots, v_n$. The **adjacency matrix A** of G, with respect to this listing of the vertices, is an $n \times n$ zero-one matrix with 1 as its (i, j)th entry when v_i and v_j are adjacent, and 0 as its (i, j)th entry when they are not adjacent. In other words, if its adjacency matrix is $\mathbf{A} = [a_{ij}]$, then

$$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \text{ is an edge of } G \\ 0 & \text{otherwise} \end{cases}$$

Example 3: Use an adjacency matrix to represent the graphs shown in Figure 3.

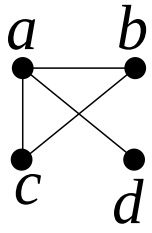


Figure 3. A Simple Graph.

Solution:
We order the vertices as a, b, c, d. The matrix representing this graph is

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Example 4: Draw a graph with the adjacency matrix

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

with respect to the ordering of vertices *a*, *b*, *c*, *d*.

Solution:

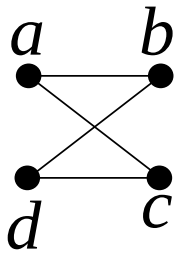


Figure 4. A Graph with the Given Adjacency Matrix.

The matrix for a directed graph $G = (V, E)$ is $\mathbf{A} = [a_{ij}]$, where

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \text{ is an edge of } G \\ 0 & \text{otherwise} \end{cases}$$

Example 2: Represent the directed graph shown in Figure 2 by adjacency matrix.

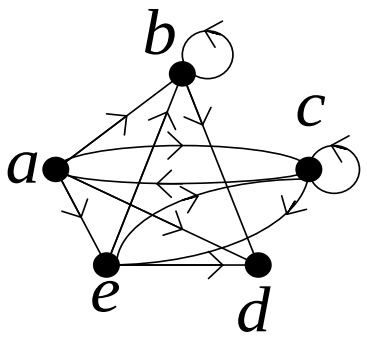


Figure 2. A Directed Graph.

Solution: We order the vertices as a, b, c, d, e. The matrix representing this graph is

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Chapter 9
Trees

9.1 Introduction to Trees

Definition 1: A *tree* is a connected undirected graph with no simple circuits.

Example 1: Which of the graphs shown in Figure 2 are trees?

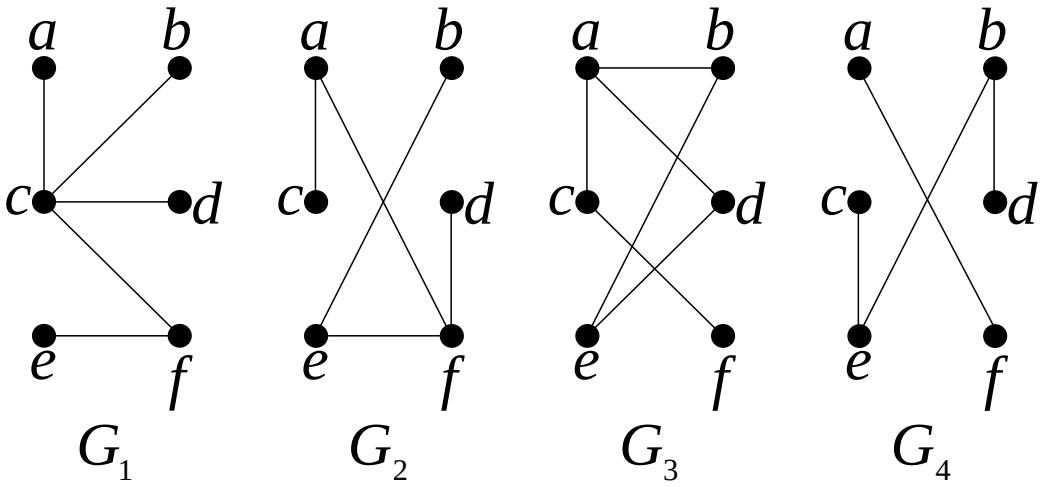


Figure 2. Examples of Trees and Graphs That Are Not Trees.

Solution:
G_1 and G_2 are trees, since both are connected graphs with no simple circuits.
G_3 is not a tree because e, b, a, d, e is a simple circuit in this graph.
G_4 is not a tree since it is not connected.

Theorem 1: An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Definition 2: A *rooted tree* is a tree in which one vertex has been designated as the root and every edge is directed away from the root.

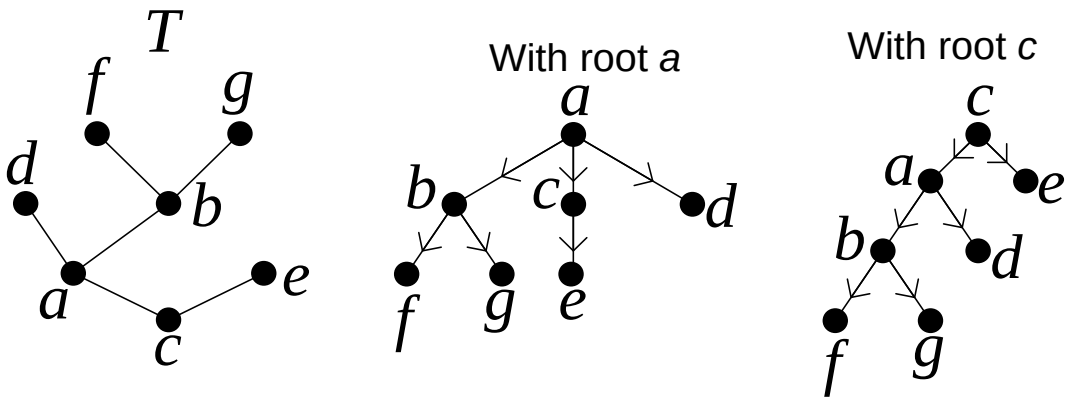


Figure 4. A tree and Rooted Trees Formed by Designating Two Roots.

We usually draw a rooted tree with its root at the top of the graph. The arrows indicating the directions of the edges in a rooted tree can be omitted, since the choice of root determines the direction of the edges.

Suppose that T is a rooted tree.

If v is a vertex in T other than the root, the **parent** of v is the unique vertex u such that there is a directed edge from u to v.

When u is the parent of v, v is called a **child** of u.

Vertices with the same parent are called **siblings**.

The **ancestors** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root.

The **descendants** of a vertex v are those vertices that have v as an ancestor.

A vertex of a tree is called a **leaf** if it has no children.

Vertices that have children are called **internal vertices**.

If a is a vertex in a tree, the **subtree** with a as its root is the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.

Example 2: In the rooted tree T (with root a) shown in Figure 5, find the parent of c, the children of g, the siblings of h, all ancestors of e, all descendants of b, all internal vertices, and all leaves. What is the subtree rooted at g?

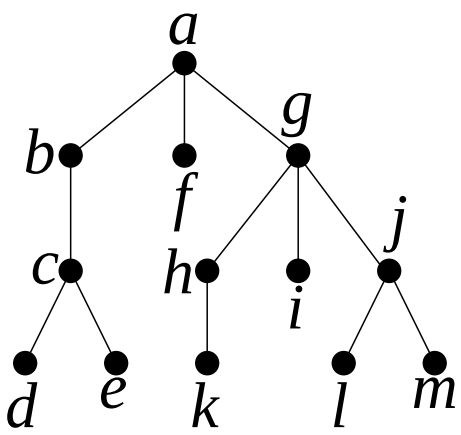


Figure 5. A Rooted Tree H.

Solution:
The parent of c is b.
The children of g are h, i, and j.
The siblings of h are i and j.
The ancestors of e are c, b, and a.
The descendants of b are c, d, and e.
The internal vertices are a, b, c, g, h, and j.
The leaves are d, e, f, i, k, l, and m.

The subtree rooted at g is shown in Figure 6.

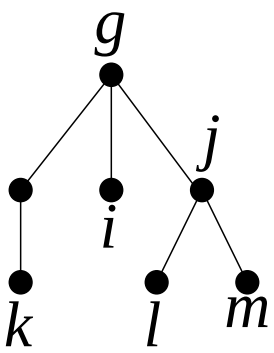


Figure 6. The Subtree Rooted at g.

Definition 3: A rooted tree is called an *m-ary tree* if every internal vertex has no more than *m* children. The tree is called a *full m-ary tree* if every internal vertex has exactly *m* children. An *m*-ary tree with $m = 2$ is called a *binary tree*.

Example 3:
Are the rooted trees in Figure 7 full *m*-ary trees for some positive integer *m*?

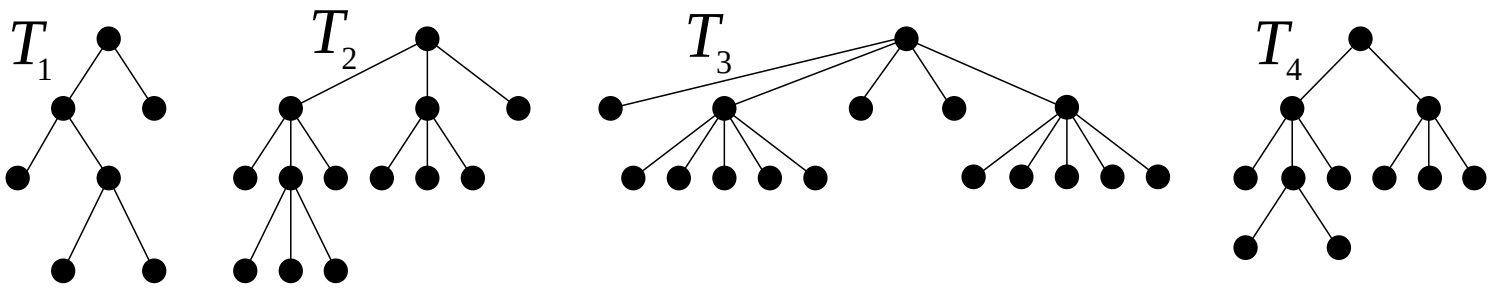


Figure 7. Four Rooted Trees.

Solution:
T_1 is a full binary tree.
T_2 is a full 3-ary tree.
T_3 is full 5-ary tree.
T_4 is not a full *m*-ary tree for any *m* since some of its internal vertices have two children and others have three children.

An **ordered rooted tree** is a rooted tree where the children of each internal vertex are ordered. Ordered rooted trees are drawn so that the children of each internal vertex are shown in ordered from left to right.

In an ordered binary tree (usually called just a **binary tree**), if an internal vertex has two children, the first child is called the **left child** and the second child is called the **right child**.

The tree rooted at the left child of a vertex is called the **left subtree** of this vertex, and the tree rooted at the right child of a vertex is called the **right subtree** of the vertex.

Example 4: What are the left and right children of *d* in the binary tree *T* shown in Figure 8(a)? What are the left and right subtrees of *c*?

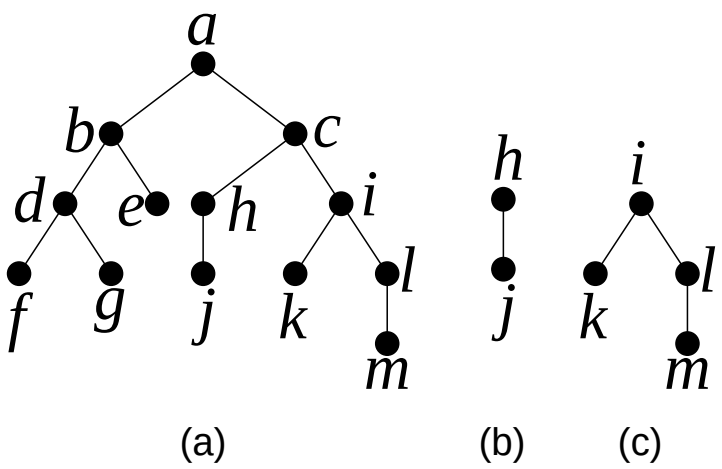


Figure 8. A Binary Tree *T* and Left and Right Subtrees of Vertex *c*.

Solution:
The left child of *d* is *f* and the right child is *g*.

We show the left and right subtrees of *c* in Figures 8(b) and 8(c), respectively.

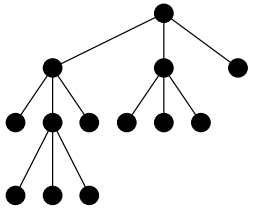
Properties of Trees

Theorem 2: A tree with n vertices has $n - 1$ edges.

Theorem 3: A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.

Theorem 4: A full m-ary three with
(i) n vertices has $i = (n - 1)/m$ internal vertices and $l = [(m - 1)n + 1]/m$ leaves,
(ii) i internal vertices has $n = mi + 1$ vertices and $l = (m - 1)i + 1$ leaves,
(iii) l leaves has $n = (ml - 1)/(m - 1)$ vertices and $i = (l - 1)/(m - 1)$ internal vertices.

Example:



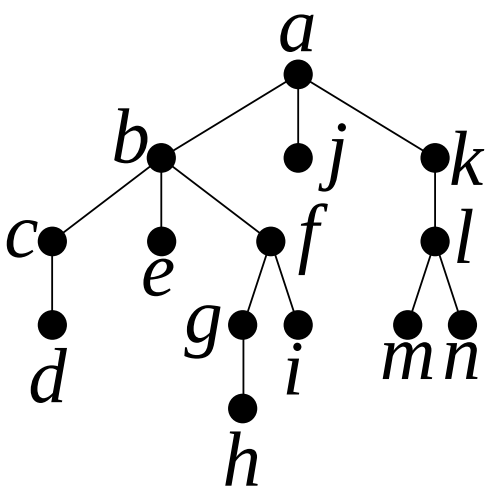
The above tree is a full 3-ary tree, that is, $m = 3$.
The number of vertices is $n = 13$.
The number of internal vertices is $i = 4$.
The number of leaves is $l = 9$.

From Theorem 2,
The number of edges is $e = (n - 1) = 13 - 1 = 12$.
From Theorem 4,
(i) $i = (n - 1)/m = (13 - 1)/3 = 4$
$l = [(m - 1)n + 1]/m = [(3 - 1)13 + 1]/3 = 9$
(ii) $n = mi + 1 = 3 \times 4 + 1 = 13$
$l = (m - 1)i + 1 = (3 - 1)4 + 1 = 9$
(iii) $n = (ml - 1)/(m - 1) = (3 \times 9 - 1)/(3 - 1) = 13$
$i = (l - 1)/(m - 1) = (9 - 1)/(3 - 1) = 4$

The **level** of a vertex *v* in a rooted tree is the length of the unique path from the root to this vertex. The level of the root is defined to be zero.

The **height** of a rooted tree is the maximum of the levels of vertices. In other words, the height of a rooted tree is the length of the longest path from the root to any vertex.

Example 10: Find the level of each vertex in the rooted tree shown in Figure 13. What is the height of this tree?



Solution:
The root *a* is at level 0.
Vertices *b*, *j*, and *k* are at level 1.
Vertices *c*, *e*, *f*, and *l* are at level 2.
Vertices *d*, *g*, *i*, *m*, and *n* are at level 3.
Finally, vertex *h* is at level 4.
Since the largest level of any vertex is 4, this tree has height 4.

Figure 13. A Rooted Tree.

A rooted *m*-ary tree of height *h* is **balanced** if all leaves are at levels *h* or *h* – 1.

Example 11: Which of the rooted trees shown in Figure 14 are balanced?

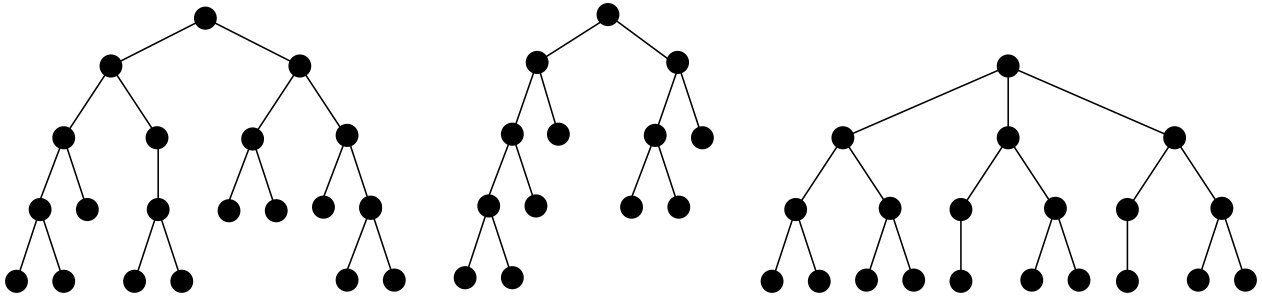


Figure 14. Some Rooted Trees.

Solution:
T_1 is balanced, since all its leaves are at levels 3 and 4.
T_2 is not balanced, since it has leaves at levels 2, 3, and 4.
T_3 is balanced, since all its leaves are at level 3.