DIPLOMA, DCOM, NOTES

Discrete Structure

EG2104CT

(for Diploma 2st Yrs. 1 Part)

Notes

DCOM

By

Arjun Chaudhary

Published in : **www.arjun00.com.np**

FB page:- **www.facebook.com/arjun00.com.np**

Unit. 1

Discrete Structure: Sets, Relations, and Functions

SETS

A *set* is a collection of well-defined objects, called elements or members. These *elements* or *members* are said to belong to the set. Sidebar 4.1 defines the mathematical symbols used in these and other definitions.

SIDEBAR 4.1 GLOSSARY OF MATHEMATICAL SYMBOLS

$\in$	is an element of
$\notin$	is not an element of
$\subseteq$	is a subset of
$\subset$	is a proper subset of
$\not\subset$	is not a subset of
$\supseteq$	is a superset of
$\supset$	is a proper superset of
$\cap$	intersection
$\cup$	union
$\rightarrow, \Rightarrow$	implies
$\Leftrightarrow$	if and only if

$\neq$	is not equal to
$\boldsymbol{\Phi}$	the null set
$\boldsymbol{U}$	the universal set
$\bar{A}$	the complement of A
$\forall$	for all
$\exists$	there exists
$\ni$	such that
$\mid$	given that
$\sim, \neg$	not (negation)
$\wedge$	and
$\vee$	or

Examples of sets are:

- An interval of numbers [7, 21]
- The students in SYST 520 at George Mason University during the spring semester of 1996
- The categories of inputs to elevator
- The possible states or outcomes that a particular input to the elevator can take
- The functions of an ATM (automated teller machine)

4.2.1 Writing Set Membership

A set is denoted by *capital* letter A, B, X, Y, with the exception of sets that are functions, which will be denoted by a lowercase italic, letter. Members are also denoted by *lowercase* letters: a, b, x, y. The mathematical expression of set membership is

$$x \in A : x \text{ is an element of } A$$
$$x \notin A \text{ or } \neg (x \in A) : x \text{ is not an element of } A$$

4.2.2 Describing Members of a Set

There are at least five ways to describe the members of a set.

1. A is the set of elements, x, that satisfies the property (or predicate), $p(x)$. $A = \{x | p(x) \text{ is true}\}$ (braces are the common delimiter of a set's definition). The property $p(x)$ must be *well-defined*, that is, able to be determined by means of rules. One test of such a property is called the

clairvoyant's test — a clairvoyant is able to predict the future or describe the past/present perfectly. Is the property or rule defined sufficiently well that the clairvoyant can answer the question? For example, the property "Is tall" does not meet the clairvoyant's test, but the property "is taller than 6 feet 3 inches" does.

2. Complete enumeration is the listing of all of the members of the set.

$$A_1 = \{0, 1, 2, 3, 4\}$$

$$A_2 = \{\text{student}_1, \text{student}_2, \ldots \text{student}_{31}\}$$

3. Use the characteristic function of the

$$\mu_A(x) = \begin{cases} 1 & \text{for } x = 0, 1, 2, 3, 4 \\ 0 & \text{otherwise} \end{cases}$$

where $\mu_A(x)$ is the characteristic function of set A for elements, x, in the set, U, of all elements. For conventional (crisp, nonfuzzy) sets, $\mu_A(x)$ may only take the values 0 for nonmembers or 1 for members.

4. Use recursive definition: $A = \{x_{i+1} = x_i + 1,\ i = 0,\ 1,\ 2,\ 3;\ \text{where } x_0 = 0\}$. Here A is defined by a recursive formula.
5. Use one or more set operators such as union, intersection, and complement. These operations should be familiar to most readers and will be defined shortly.

4.2.3 Special Sets

***U*:** the universal set or set of all possible members.

***Φ*:** the null set, a set with no elements. Φ and $\{\Phi\}$ are not the same. Φ has no elements, while $\{\Phi\}$ has one.) We can write $\Phi = \{x \in U \mid x \neq x\}$.

Singleton set: a set with only one element.

Finite set: a set with a finite number of distinct elements.

Infinite set: a set with an infinite number of distinct elements.

For example: $A_1 = \{1,\ 2,\ 3,\ 4,\ \ldots,\ 101\}$ is finite, $A_2 = \{1,\ 2,\ 3,\ 4, \ldots,\}$ is infinite, and $A_3 = \{x, \{1, 2\}, y, \{z\}\}$ may be finite or infinite. The finiteness of A_3 depends on whether x and y are finite or infinite. (Note $\{1,2\}$ and $\{z\}$ are sets, but each is only one element of A_3. Also note that z is *not* an element of A_3, but $\{z\}$ is.)

Subsets or set inclusion: if A and B are two sets, and if every element of A is an element of B, then A is a *subset* of B, $A \subseteq B$. If A is a subset of B, and if B has

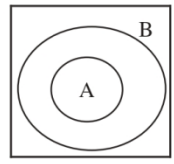


FIGURE 4.1 Set inclusion.

at least one element that is *not* in A, then A is a *proper* subset of B, $A \subset B$. See Figure 4.1.

Equality of sets: if A and B are sets, and A and B have precisely the same elements, then A and B are *equal*, $A = B$.

The following properties follow from the above definitions:

$A \subseteq A$; a set is a subset of itself.

$\Phi \subseteq A$, $A \subseteq U$. The null set is a subset of every set; every set is a subset of the universal set.

If $\Phi \neq A$, then $\Phi \subset A$. If a set is not the null set, then the null set is a proper subset of the set.

If $A \subseteq B$ and $B \subseteq A$, then $A = B$. If two sets are subsets of each other, then they are equal.

If $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$. Set inclusion is transitive, a property that we will formally define later.

4.2.4 Operations on Sets

The following operations are performed on sets:

Absolute complement, $\bar{A}$: Let $A \subseteq U$. $\bar{A} = \{x \mid x \notin A\}$ (Note $\bar{\Phi} = U$, $\bar{U} = \Phi$, $\bar{\bar{A}} = A$) See Figure 4.2.

Relative complement of A with respect to B, $B - A$: Let A and B be sets, $B - A = \{x \mid x \in B$ and $x \notin A\}$. The relative complement is also called *set difference*. See Figure 4.3.

Union of A and B, $A \cup B$: $A \cup B = \{x \mid x \in A$ or $x \in B$ or both$\}$.

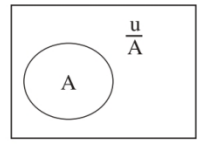


FIGURE 4.2 Absolute complement.

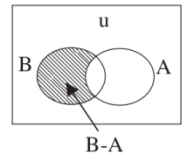


FIGURE 4.3 Relative complement.

Intersection of A and B, $A \cap B$: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$. (Note A and B are called *disjoint* if $A \cap B = \Phi$. See Figure 4.4.

Boolean sum (symmetrical difference), $A + B$ or $A \Delta B$:

$$A + B = \{x | x \in A \text{ or } x \in B, \text{but not both}\} = (A - B) \cup (B - A)$$

The following properties of the above set operations can be easily derived:

1. $A \cup \Phi = A$, and $A \cap \Phi = \Phi$.
2. $A \cup U = U$, and $A \cap U = A$.
3. Idempotent: $A \cup A = A$, and $A \cap A = A$
4. Associative:

$$(A \cup B) \cup C = A \cup (B \cup C)$$
$$(A \cap B) \cap C = A \cap (B \cap C)$$

5. Commutative: $A \cup B = B \cup A$, and $A \cap B = B \cap A$
6. Distributive:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

7. DeMorgan's Laws: $(\overline{A \cup B}) = \bar{A} \cap \bar{B}$, and $(\overline{A \cap B}) = \bar{A} \cup \bar{B}$

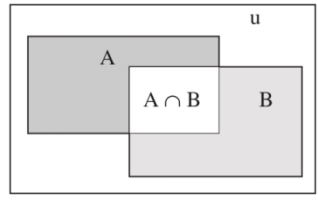


FIGURE 4.4 Set intersection.

Example Use DeMorgan's laws to prove that the complement of $(\bar{A} \cap B) \cap (A \cup \bar{B}) \cap (A \cup C)$ is $(A \cup \bar{B}) \cup (\bar{A} \cap (B \cup \bar{C}))$.

Solution: Starting with $(\bar{A} \cap B) \cap (A \cup \bar{B}) \cap (A \cup C)$, note that $(A \cup \bar{B}) \cap (A \cup C)$ is the same as $A \cup (\bar{B} \cap C)$.

Step 1: Making this substitution, we want to find the complement $\overline{(\bar{A} \cap B) \cap (A \cup (\bar{B} \cap C))}$.

Step 2: By DeMorgan's law, the complement of an intersection is the union of set complements. So this can be written as $\overline{(\bar{A} \cap B)} \cup \overline{(A \cup (\bar{B} \cap C))}$.

Step 3: Again, the complement of an intersection is the union of the set complements. So this can be written as $(A \cup \bar{B}) \cup \overline{(A \cup (\bar{B} \cap C))}$.

Step 4: Also by DeMorgan's law, the complement of a union is the intersection of the set complements. So this can be written as $(A \cup \bar{B}) \cup (\bar{A} \cap \overline{(\bar{B} \cap C)})$.

Step 5: Again, the complement of an intersection is the union of the set complements. This yields $(A \cup \bar{B}) \cup (\bar{A} \cap (B \cup \bar{C}))$. QED

4.2.5 Partitions

A *partition* on a set A is a collection P of disjoint subsets of A whose union is A. For a collection B_i ($i = 1, 2, \ldots, n$) to be a partition P of A:

1. $B_i \subseteq A$ for $i = 1, 2, \ldots, n$.
2. $B_i \cap B_j = \Phi$ for $i \neq j$.
3. for any $x \in A$, $x \in B_i$ for some i; (alternatively $B_1 \cup B_2 \cup \ldots \cup B_n$)

The concept of a partition (Fig. 4.5) is the most basic and *far-reaching mathematical concept* to our development of systems engineering. We will talk

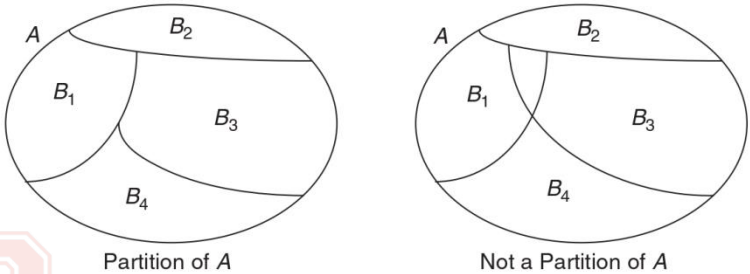


FIGURE 4.5 Set partition.

about the importance of creating a partition of the system's requirements, and a partition of the system's function, and a partition of the system's physical resources. This is just the beginning.

4.2.6 Power Set

The power set of a set A is denoted, $\mathbf{P}(A)$. The *power set* is the set of all sets that are subsets of A. Mathematically, the power set is the family (or set) of sets such that $X \subseteq A \Leftrightarrow X \in \mathbf{P}(A)$, or $\mathbf{P}(A) = \{X \mid X \subseteq A\}$.

1. Let $A_0 = \Phi$, $\mathbf{P}(\Phi) = \{\Phi\}$, [where A_0 is a set with ***zero*** elements and $\mathbf{P}(A_0)$ has one element].
2. Let $A_1 = \{a\}$; $\mathbf{P}(A_1) = \{\Phi, A_1\} = \{\Phi, \{a\}\}$ [where A_1 is a set with ***one*** element and $\mathbf{P}(A_1)$ has two elements].
3. Let $A_2 = \{a, b\}$; $\mathbf{P}(A_2) = \{\Phi, \{a\}, \{b\}, \{a, b\}\}$ [where A_2 is a set with ***two*** elements and $\mathbf{P}(A_2)$ has four elements].

How many elements does the power set of a set of A_n have?

Theorem If A_n is a set with n elements, then $\mathbf{P}(A_n)$ has 2^n elements.

Proof We will use mathematical induction. For $n = 0, 1, 2, 3, \ldots$, let $S(n)$ be the statement: If A_n is a set with n elements, then $\mathbf{P}(A_n)$ has 2^n elements.

i. First show that if A_0 has 0 elements, then $\mathbf{P}(A_0)$ has $2^0 = 1$ element.

$$A = \Phi, \mathbf{P}(A) = \{\Phi\}$$

ii. Assume $S(k)$ is true and then show that $S(k+1)$ is true. Let A_{k+1} be a set with $k+1$ elements. Define B to be a proper subset of A_{k+1} with k of A_{k+1}'s elements:

$$A_{k+1} = \{a_1, a_2, \ldots, a_k, a_{k+1}\}$$

$$B = A_k = \{a_1, a_2, \ldots, a_k\}$$

$$\text{So } A_{k+1} = \{a_{k+1}\} \cup B.$$

Therefore, every subset of A_{k+1} either contains a_{k+1}, or it does not.

1. If a subset does not contain a_{k+1}, then it is a subset of B, and we know there are 2^k subsets of B, by induction.
2. If a subset does contain a_{k+1}, then it is the union of a subset of B and a_{k+1}. There must be 2^k of these since there are 2^k subsets of B. So there are $2^k + 2^k = 2^k(1+1) = 2^k\ 2 = 2^{k+1}$ subsets of A_{k+1} or 2^{k+1} elements of $\mathbf{P}(A_{k+1})$.

The concept of a power set has many potential uses in systems engineering. For example, the power set of system inputs is an upper bound on the test sequences required to test the system exhaustively.

4.3 RELATIONS

This section defines relations using the concepts of ordered pairs and Cartesian products. Important properties of relations are defined, followed by definitions of partial orderings and equivalence relations.

4.3.1 Ordered Pairs and Cartesian Products

An *ordered pair* is (x, y) if $x \in A$, $y \in B$. A *Cartesian product*, $A \times B$, is defined over two sets, A and B, such that $A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$. That is, the Cartesian product of two sets is the set of all possible ordered pairs of those two sets. The following are examples of Cartesian products:

1. $A = \{1\}$, $B = \{2\}$: $A \times B = \{(1, 2)\}$ and $B \times A = \{(2, 1)\} \neq A \times B$.
2. $X = \{\text{students of SYST 520 during the spring semester of 1996}\} = \{S_1, S_2, \ldots, S_{31}\}$, $Y = \{A, B, C\}$: $X \times Y = \{(S_1, A), (S_1, B), (S_1, C), \ldots, (S_{31}, A), (S_{31}, B), (S_{31}, C)\}$

An ordered n-tuple is defined to be $A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) \mid a_i \in A_i, i = 1, 2, \ldots, n\}$, where $(a_1, a_2, \ldots, a_n)$.

4.3.2 Unary and Binary Relations

A *unary relation* on a set A relates elements of A to itself and is a subset, R, of $A \times A$. R is usually described by a predicate that defines the relation. Examples are $\leq, =, >$, "taller than," and "older than." If a_1 and $a_2 \in A$, we write $(a_1, a_2) \in R$, which means that $a_1 \; R \; a_2$ or a_1 "is related to" a_2.

A binary *relation* is a relation R that relates elements of A to elements of B and is a subset of $A \times B$. The *domain* of R, written as "dom R," is defined as: dom $R = \{x \mid x \in A \text{ and } (x, y) \in R \text{ for some } y \in B\}$. The *range* of R, written as "ran R," is defined as: ran $R = \{y \mid y \in B \text{ and } (x, y) \in R \text{ for some } x \in A\}$. Again $(a_1, b_1) \in R \Leftrightarrow a_1 \; R \; b_1$.

Example Let R be the relation from $A = \{1, 3, 5, 7\}$ to $B = \{1, 3, 5\}$, which is defined by "x is less than y." Write R as a set of ordered pairs.

Solution:

$R = \{(x, y) \mid x \in A, y \in B, x < y\}$

$R = \{(1, 3), (1, 5), (3, 5)\}$

Recall the relations within and between systems engineering classes that were discussed in Chapter 2. The hierarchy of requirements was defined by the relation "incorporates" in moving from the top of the requirements hierarchy to the bottom; "incorporated in" was the relation that moved from bottom to top. The relation "is decomposed by" moved from the top of the functional decomposition to the bottom; "decomposes" moves in the opposite direction. The physical hierarchy of a system and its components used the relation "is built from" in moving from top to bottom and "is built in" for moving from bottom to top.

Binary relations included the tracing from requirements to functions or the system, the performance of functions by the system and its components, and inputs and outputs of items for functions. The relation "is traced to" was used for the binary relations of input/output stakeholders' requirements being mapped to functions and for system-wide/technology requirements being mapped to the system. The binary relation for the system and components being related to functions used the relation "pertains." The relations "inputs" and "outputs" addressed functions being related to items.

To discuss the properties of unary relations, some additional information is needed concerning the possible ways to prove an implication. An implication is an "If …, then …" statement, which is commonly written as "If p is true, then q is true" or "$p \rightarrow q$." There are eight common methods for proving implications of this form.

1. *Trivial proof*: Show that q is true independently of the truth of p.
2. *Vacuous proof*: By mathematical convention, whenever p is false, $p \rightarrow q$ is true. The vacuous proof involves showing that p is false. *This method is key to understanding the full implications of the properties of unary* relations *that are discussed below*.
3. *Direct proof*: Assume that p is true and use arguments based upon other known facts and logic to show that q must be true.
4. *Indirect proof*: Use direct proof of the contrapositive of $p \rightarrow q$. The contrapositive of a true implication is known to be true; the contrapositive of $p \rightarrow q$ is $\sim q \rightarrow \sim p$ (or q is false implies p is false). Here we assume q is false and prove via logic and known facts that p must be false.
5. *Contradiction-based proof*: DeMorgan's laws can be used to show that $p \rightarrow q$ is equivalent to $\sim(p \wedge (\sim q))$, that is, the statement "p is true and q is false" is false. Proof by contradiction starts by assuming that $(p \wedge (\sim q))$ is true and then proving, based on this assumption, that some known truth must be false. If the only weak link in the argument is the assumption of $(p \wedge (\sim q))$, then this assumption must be wrong.
6. *Proof by cases*: If p can be written in the form of p_1 or p_2 or … or p_n $(p_1 \vee p_2 \vee \ldots \vee p_n)$, then $p \rightarrow q$ can be proven by proving $p_1 \rightarrow q$, $p_2 \rightarrow q$, …, $p_n \rightarrow q$ as separate arguments.
7. *Proof by elimination of cases is an extension of the method above*: Recall from the second method that $p \rightarrow q$ is equivalent to $[(p \vee q) \wedge (\sim p)]$, that is

(p and q are true) or (p is false). Now p can be partitioned into a set of cases as done in 6 and attacked one at a time.

8. *Conditional proof*: If we are to prove $p \rightarrow (q \rightarrow r)$, we can prove the equivalent $(p \wedge q) \rightarrow r$.

4.3.3 Properties of Unary Relations on *A*

The seven properties discussed here are reflexive, irreflexive, symmetric, antisymmetric, asymmetric, transitive, and intransitive.

1. *Reflexive*: $x \; R \; x$ for all $x \in A$, e.g., equality, $\leq$, $\geq$.
2. *Irreflexive*: $x \not R \; x$ for all $x \in A$, for example, greater than, is the father of.
3. *Symmetric*: If $x \; R \; y$, then $y \; R \; x \; \forall x, y \in A$, for example, equality, is spouse of. Note if $x \not R \; y$ for all x and y in A, then the relation is symmetric by a vacuous proof.
4. *Antisymmetric*: If $x \; R \; y$ and $y \; R \; x$, then $x = y \; \forall x, \; y \in A$, for example, equality, $\leq$, $\geq$. Note if there is no situation in which "$x \; R \; y$ and $y \; R \; x$" is true, then the relation is antisymmetric by vacuous proof.
5. *Asymmetric*: If $x \; R \; y$, then $y \not R \; x \; \forall x, \; y \in A$, e.g., $<$, $>$.
6. *Transitive*: If $x \; R \; y$ and $y \; R \; z$, then $x \; R \; z \; \forall x, \; y, \; z \in A$, for example, $\leq$, $\geq, =, >$. This property is the most difficult to grasp. If there is no situation in which "$x \; R \; y$ and $y \; R \; z$," then the relation is transitive by vacuous proof.
7. *Intransitive*: If for some $x, y, z \in A$, it is true that $x \; R \; y$, $y \; R \; z$, but $x \not R \; z$, the relation is considered intransitive.

Example Let L be the set of lines in the Euclidean plane and let R be the relation on L defined by "x is parallel to y." Is R a reflexive relation? Why? Is R a symmetric relation? Why? Is R a transitive relation?

Solution:

1. This question reduces to whether a line is parallel to itself. If the definition of parallel is having no points in common (everywhere equidistant), then a line cannot be parallel to itself because the two lines have every point in common. So R is not a reflexive relation.
2. R is a symmetric relation. Consider each $x \in L$. x will have an infinite number of $y \in L$ which satisfy the parallel relationship. Each such y is in turn parallel to x. Thus, $(x, y) \in R$ for all x and y that are parallel, and $(y, x) \in R$, so the relation is symmetric.
3. R is a transitive relation. Again, consider $(x, y) \in R$ and $(y, z) \in R$; x will be parallel to z, so $x \; R \; z$ and R is transitive for all $x, y, z \in L$.

Example Let F be the set of functions in the functional decomposition for a system. Let R be the relation on F defined by "is decomposed by." Is R a

reflexive relation? Why? Is R a symmetric relation? Why? Is R a transitive relation?

Solution:

1. R is not a reflexive relation because a function does not decompose itself.
2. R is not a symmetric relation because if f_1 decomposes f_0, then f_0 cannot decompose f_1.
3. R is not a transitive relation. The function f_0 is decomposed by f_1, f_2 and f_3, and f_1 is decomposed by f_{11}, f_{12} and f_{13}. However f_0 is not decomposed by f_{11}, f_{12} or f_{13}.

4.3.4 Partial Ordering

A relation R on A is a *partial ordering* if R is reflexive, antisymmetric, and transitive. Examples of partial orderings are $\geq$ or $\leq$ on the real number line, or $\supseteq$ or $\subseteq$ on $\mathbf{P}(A)$. Examples of nonpartial orderings are $<$ or $>$ on the real number line, $\subset$ or $\supset$ on $\mathbf{P}(A)$. (Both of these are asymmetric and antisymmetric.)

4.3.5 Equivalence Relations

A relation R on a set A is an *equivalence relation* if R is reflexive, symmetric, and transitive. An example of an equivalence relation is equality.

4.4 FUNCTIONS

This section defines functions and discusses the composition of functions.

4.4.1 Definitions

Let A and B be two nonempty sets. We write a function f as $f: A \to B$ and say that f maps every element of A (the domain) to one and only one element of B (the range). If $(a, b) \in f$, then element b is the *image* of element a under f. Note that a function can map elements of A onto itself, $f: A \to A$. A *function* f from A to B is a *relation* such that

(a) dom $f = A$
  (i) f is defined for each element of A, $a \in A$.
  (ii) $\exists(a, b)$ where $b \in B$ for each element of A, $a \in A$.

(b) if $(a, b) \in f$ and $(a, c) \in f$, then $b = c$; that is, f is single-valued, or no element of A is related to two elements of B.

A function is called *one-to-one* or *injective* if $(a, b) \in f$ and $(c, b) \in f$ implies $a = c$. That is, no two elements of A can be mapped into the same element of B by f.

A function $f: A \rightarrow B$ is *onto* or *surjective* if and only if the range of $f = B$, that is, f is defined for every $b \in B$.

If a function is both one-to-one and onto (or *bijective*), then the relation f^{-1} is single-valued and maps every element of B onto some element of A; f^{-1} is therefore a function, called the *inverse* function.

Example If $A=\{1, 2, 3, 4\}$ and $B=\{a, b, c, d\}$, determine if the following functions are one-to-one or onto.

(a) $f = \{(1,a),(2,a),(3,b),(4,d)\}$
(b) $g = \{(1,d),(2,b),(3,a),(4,a)\}$
(c) $h = \{(1,d),(2,b),(3,a),(4,c)\}$

Solution:

(a) f is NOT one-to-one since $f^{-1}(a) = \{1,2\}$. f is NOT onto since $f^{-1}(c) = \phi$.
(b) g is NOT one-to-one since $g^{-1}(a) = \{3,4\}$. g is NOT onto since $g^{-1}(c) = \phi$.
(c) h is *one-to-one* since all elements of B correspond to unique elements in A. h is *onto* since every element of B has some pre-image in A.

So we have progressed mathematically from sets to relations to functions.

Functions $\subseteq$ Relations $\supseteq$ Sets, or a function is a relation is a set.

As systems engineers we will focus on functional architectures. We will represent the functions of the system as relations or functions in graph-like structures. The underlying theory is set theory.

4.1.1 Composition

Let R be a relation from A to B, and S be a relation from B to C. (a, c) is an element of the *composition* of R and S, (denoted $R \bullet S$ or $R\ S$) if and only if there is an element $b \in AB$ such that $a\ R\ b$ and $b\ S\ c$. That is, a and c must be linked together by b; a is mapped to b and b is mapped to c. (Note that some authors write the composition of R and S as $S \bullet R$ so be careful.)

The composition of functions is defined in the same way as the composition of relations.

Example Assume R and S are relations from A to A. If $A=\{1, 2, 3, 4\}$, $R=\{(1, 2), (2, 3), (3, 4), (4, 2)\}$, and $S=\{(1, 3), (2, 4), (4, 2), (4, 3)\}$, then compute $R \bullet S$, $S \bullet R$ and $R \bullet R$.

Solution:

$R \bullet S=\{(1, 4), (3, 2), (3, 3), (4, 4)\}$.

(1, 2) from R is composed with (2, 4) from S (this is written (1, 2) ● (2, 4)) and yields (1, 4).

(1, 2) from R cannot be composed with any of the other elements of S because they do not begin with a 2.

(3, 4) ● (4, 2)=(3, 2).

(3, 4) ● (4, 3)=(3, 3).

(4, 2) ● (2, 4)=(4, 4).

S ● R={(1, 4), (2, 2), (4, 3), (4, 4)}, which is not equal to R ● S.

R ● R={(1, 3), (2, 4), (3, 2), (4, 3)}.

As systems engineers we will employ functional decomposition to develop the functional architecture. Composition is the mathematical property from which decomposition derives its name. However, as discussed in Chapter 7, composition is only applicable to functional decomposition in limited situations.

4.5 SUMMARY

This chapter began with the introduction of a set, the foundation of a branch of mathematics called discrete mathematics. A great deal of terminology was introduced to define special sets such as the universal and null sets and operations on sets.

During the discussion of sets, the concept of partition was defined. The partition is perhaps the most important mathematical concept introduced in this chapter for application in this book. A partition is a subdivision of a set into subsets, which contain no common members, and yet the union of the subsets contains every element of the original set. In future chapters requirements will be partitioned, functional decompositions will be defined to be partitions, and the physical decomposition will be defined to be a partition.

The power set of a set is the set of all subsets of that set. This notion of a power set is not exploited fully in this book but will become key to the future development and application of mathematics to the engineering of systems.

The next major section of this chapter dealt with relations and the key properties associated with relations. A relation is a set of ordered pairs; the elements of the ordered pairs come from one or two sets. If the functions of a system are not fully defined in terms of inputs, then these system functions are, in fact, mathematical relations. Functions are relations that satisfy certain properties; a function maps every element of the domain of the function to some element of the range, but does not map any element of its domain to more than one element of the range. One-to-one and onto properties of functions were also discussed. Finally the composition of functions was defined.

PROBLEMS

4.1 Define the students enrolled in this class during this semester as a set, S.

a. Specify a partition of S into 2 subsets.

b. Specify a partition of S into 3 subsets.

c. Specify a partition of S into 5 subsets.

4.2 Let $A_1=\{1, 3, 5, 7, 9, 11\}$, $A_2=\{2, 6, 9, 11\}$, $A_3=\{2, 4, 6, 9, 11\}$. Show that:

a. $A_1+A_2=(A_1-A_2) \cup (A_2-A_1)$

b. $A_1 \cup (A_2 \cap A_3)=(A_1 \cup A_2) \cap (A_1 \cup A_3)$

4.3 Prove that the following relations are true in general:

a. $A_1+A_2=(A_1-A_2) \cup (A_2-A_1)$

b. $A_1 \cup (A_2-A_3)=(A_1 \cup A_2)-(A_1 \cup A_3)$

4.4 Let R be a relation from A to B and defined ''x is at least twice as big as y.'' Write R as a set of ordered pairs for

a. $A=\{1, 3, 5, 7\}$ and $B=\{2, 3, 4, 6\}$

b. $A=\{0, 1\}$ and $B=\{0, 1\}$

c. $A=\{1, 2, 3, 4, 5, 6, 7\}$ and $B=\{3, 6\}$

4.5 Let R be relation from A to B where ''x is greater than or equal to y squared.''

Then define R as a set of ordered pairs for the following:

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

b. $A=\{25\}$, $B=\{5, 6, 7\}$

4.6 There are three families defined by the sets A, B, and C; each family has a dad, mom and three kids:

$A=\{$Dad, Mom, Doris, Bill, Tom$\}$

$B=\{$Dad, Mom, Doris, Daisy, Debbie$\}$

$C=\{$Dad, Mom, Bill, Bob, Biff$\}$

Consider the relations ''is the spouse of,'' ''is the brother of,'' and ''is the blood relative of.'' (Hints: I am not the brother of myself. Two people are blood relatives if they share the blood of a common ancestor, who may or may not be part of sets A, B, or C. I am the blood relative of myself. Biff is a male.)

Identify which of these relations satisfy which of the seven properties of unary relations for each of the three sets by placing a yes or no in the empty cells of the following table.

	Reflexive	Irreflexive	Symmetric	Anti-symmetric	Asymmetric	Transitive	Intransitive
''is the spouse of'' on A							
''is the brother of'' on A							
''is the blood relative of'' on A							
''is the spouse of'' on B							
''is the brother of'' on B							
''is the blood relative of'' on B							
''is the spouse of'' on C							
''is the brother of'' on C							
''is the blood relative of'' on C							

4.7 Let R be a relation from A to B and S be a relation from B to C.

a. Find $R \bullet S$ for $A=\{1, 3, 5, 7\}$, $B=\{1, 2, 4, 5, 7\}$, $C=\{1, 2, 3, 4, 5, 6\}$, $R=\{(1, 2), (3, 4), (5, 2), (7, 4)\}$ and $S=\{(1, 2), (2, 4), (4, 3), (7, 5)\}$.

b. Are any of these relations R, S, $R \bullet S$ functions? One-to-one functions? One-to-one and onto functions?

4.8 If $A_1=\{1, 2, 3, 4\}$ and $A_2=\{1, 4, 9, 25\}$, determine if the following functions that map A_1 onto A_2 are one-to-one, onto, or both one-to-one and onto.

a. $f_1=\{(1, 1), (2, 4), (3, 4), (4, 25)\}$

b. $f_2=\{(1, 1), (2, 4), (3, 25), (4, 25)\}$

c. $f_3=\{(1, 1), (2, 4), (3, 9), (4, 25)\}$

4.9 Develop two relations R (from A to B) and S (from B to C) that have to do with people. Show the result of $R \bullet S$.

4.10 Let R and S be relations from A—A, where $A=\{1, 2, 3, 4\}$ and:

$R=\{(1, 1), (2, 2), (3, 3), (1, 2), (2, 3), (1, 3), (2, 1), (3, 1), (3, 2)\}$

$S=\{(2, 3), (1, 2), (2, 1), (3, 1), (1, 3)\}$

a. Find if these relations are symmetric, reflexive, and transitive.

b. Find $R \bullet S$, $S \bullet R$ and $R \bullet R$.

4.11 Let A be a set of three colors: {red, blue, green}. What are the elements of the power set of A?

4.12 Let *SIBLINGS*={Andrea, Bobby, Catherine, David, Eric}. Find the elements of the power set of *SIBLINGS*, P(*SIBLINGS*).

4.13 Show that the P{Andrea, Bobby} is a subset of the P(*SIBLINGS*) from Problem 4.12.

4.14 Prove that for any two sets *A* and *B*, (P(*A*)∩P(*B*))=P(*A*∩*B*).

4.15 Find two sets *A* and *B* that show (P(*A*)∪P(*B*)) 6¼ P(*A*∪*B*).

4.16 Prove that for any two sets *A* and *B*, (P(*A*) ∪P(*B*))DP(*A*∪*B*).

4.17 Prove that the seven properties of set operations in Section 4.2.4 are true.

Unit Two: Logic, Induction and Reasoning

1.1 Logic

- **Logics** consist of the following
  - ✓ A formal system for describing states of affairs, consisting of
    1. the syntax of the language ,and
    2. the semantics of the language
  - ✓ The Proof –Theory: a set of rules for deducing the entailments of a set of sentences.
- Logic is a formal language, with precisely defined syntax and semantics, which supports reasoning.
- Different logics exist, which allow you to represent different kinds of things, and which allow more or less efficient inference.
- Syntax: of a language describes the possible configurations that can constitute a sentence.
- Semantics: determines the facts in the world to which the sentence refer.

1.2 Proposition

- A proposition is a declarative sentence (that is, a sentence that declares a fact) that is either true or false, but not both.
- Example of declarative sentences which are propositions.
  1. Kathmandu is the capital of the Nepal. **True proposition**
  2. Sun rises in west. **False proposition**
  3. 5+ 1 = 6.**True proposition**
  4. 7+ 2 = 8**False proposition**
- Example 2: Consider the following sentences.
  1. What time is it?
  2. Read this carefully.
  3. z + 1 = 5.
  4. a + b = c.
  - ✓ Sentences 1 and 2 are not propositions because they are not declarative sentences. Sentences 3and 4 are not propositions because they are neither true nor false. Note that each of sentences 3and 4 can be turned into a proposition if we assign values to the variables.
- Propositional Logic: letters are used to represent **propositional variables** (or **statement variables**), that is, variables that represent propositions,
- The letters used for propositional variables: *p, q, r, s . . .*
- The **truth value** of a proposition is true, denoted by T, if it is a true proposition, and the truth value of a proposition is false, denoted by F, if it is a false proposition.
- The area of logic that deals with propositions is called the **propositional calculus** or **propositional logic**.
- **Compound propositions** are formed from existing propositions using logical operators.

Propositions using logical operators

Negation

- Let p be a proposition. The negation of p, denoted by "¬p", is the statement "It is not the case that p."
- The proposition ¬p is read "not p."
- The truth values of p, and¬p, are opposite to each other.
- Example1
  - ✓ Find the negation of the proposition "Ram's laptop runs Windows"
  - ✓ Solution: The negation is "It is not the case that Ram's laptop runs Windows."
  - ✓ In other way: "Ram's laptop does not run Windows."
  - ✓ The notable fact is that Ram's laptop does not run Windows mean that Ram's laptop runs Linux or UNIX.
- Example 2
  - ✓ Find the negation of the proposition "Sony's Smartphone has at least 4GB of RAM"
  - ✓ Solution: The negation is: "It is not the case that Sony's Smartphone has at least 4GB of RAM."
  - ✓ In other way "Sony's Smartphone does not have at least 4GB of RAM"
  - ✓ More simply as "Sony's Smartphone has less than 4GB of RAM."
- Truth Table of Negation

P	¬p
T	F
F	T

Conjunction

- Let p and q be propositions. The conjunction of p and q, denoted by p ∧ q, is the proposition "p and q." The conjunction p ∧ q is true when both p and q are true and is false otherwise.
- Example: Find the conjunction of
  - ✓ Kathmandu is the capital city of Nepal and Ancient name of Kathmandu is Patan.
  - ✓ Suppose p means "Kathmandu is the capital city of Nepal" and q mans "Ancient name of Kathmandu is Patan".
  - ✓ Then p ∧ q: Kathmandu is the capital city of Nepal and Ancient name of Kathmandu is Patan.

Disjunction

- Let p and q be propositions. The disjunction of p and q, denoted by p ∨ q, is the proposition "p or q." The disjunction p ∨ q is false when both p and q are false and is true otherwise.
- Example: Find the disjunction of the propositions p and q where p is the proposition "Ram's PC has more than 16 GB free hard disk space" and q is the proposition "The processor in Ram's PC runs faster than 1 GHz."
- Then p ∨ q: "Ram's PC has more than 16 GB free hard disk space, or the processor in Ram's PC runs faster than 1 GHz."

- Truth Table of Conjunction

p	q	p ∧ q
F	F	F
F	T	F
T	F	F
T	T	T

- Truth Table of Disjunction

p	q	p ∨ q
F	F	F
F	T	T
T	F	T
T	T	T

Exclusive OR

- Let p and q be propositions. The exclusive or of p and q, denoted by p ⊕ q, is the proposition that is true when exactly one of p and q is true and is false otherwise.
- Truth Table of exclusive or

p	q	p ∧ q
F	F	F
F	T	T
T	F	T
T	T	F

Conditional Statement

- Let p and q be propositions. The conditional statement p → q is the proposition "if p, then q." The conditional statement p → q is false when p is true and q is false, and true otherwise.
- In the conditional statement p → q, p is called the hypothesis (or antecedent or premise) and q is called the conclusion (or consequence).
- The statement p → q is called a conditional statement because p → q asserts that q is true on the condition that p holds.
- A conditional statement is also called an implication. Thus p → q can be read as "p implies q".
- We can express all of the following ways to express this conditional statement:
  - ✓ "if p, then q" "p implies q"
  - ✓ "if p, q" "p only if q"
  - ✓ "p is sufficient for q" "a sufficient condition for q is p"
  - ✓ "q if p" "q whenever p"
  - ✓ "q when p" "q is necessary for p"
  - ✓ "a necessary condition for p is q" "q follows from p
  - ✓ "q unless ¬p" means that if ¬*p* is false, then *q* must be true.

- Truth Table of Conditional Statement

p	q	p → q
F	F	T
F	T	T
T	F	F
T	T	T

- Consider an Example: "If you get an A grade, then I'll give you a Bike. “The statement will be *true* if I keep my promise and *false* if I don’t. Suppose it's *true* that you get an A grade and it's *true* that I give you a Bike. Since I kept my promise, the implication is true. This corresponds to the first line in the table. Suppose it's *true* that you get an A but it's *false* that I give you a Bike. Since I *didn't* keep my promise, the implication is *false*. This corresponds to the second line in the table. What if it's false that you get an A? Whether or not I give you a Bike, I haven't broken my promise. Thus, the implication can't be false, so (since this is a two-valued logic) it must be true. This explains the last two lines of the table.

Biconditional Statement

- Let p and q be propositions. The biconditional statement p ↔ q is the proposition “p if and only if q.” The biconditional statement p ↔ q is true when p and q have the same truth values, and is false otherwise. Biconditional statements are also called bi-implications.
- The statement p ↔q is true when both the conditional statements p → q and q → p are true and is false otherwise.
- Note that $p \leftrightarrow q$ has exactly the same truth value as $(p \rightarrow q) \wedge (q \rightarrow p)$.
- p ↔ q can be expressed in a some other common ways
  - ✓ “p is necessary and sufficient for q”
  - ✓ “if p then q, and conversely”
  - ✓ “p iff q.”
- Example: Let *p* be the statement “You can take the flight,” and let *q* be the statement “You buy a ticket.”
- Then $p \leftrightarrow q$ is the statement: “You can take the flight if and only if you buy a ticket.”
- The truth table for biconditional statement

p	q	p ↔ q
F	F	T
F	T	F
T	F	F
T	T	T

The precedence levels of the logical operators

- Precedence defines the order of executing operators if the more than one operator is present in single expression**.**
- The table shows the precedence of operators

Operator	Precedence
¬	1
∧	2
∨	3
→	4
↔.	5

1.3 Converse, Contrapositive, and Inverse

- The proposition $q \rightarrow p$ is called the **converse** of $p \rightarrow q$.
- The **contrapositive** of $p \rightarrow q$ is the proposition $\neg q \rightarrow \neg p$. The contrapositive always has the same truth value as $p \rightarrow q$.
- The proposition $\neg p \rightarrow \neg q$ is called the **inverse** of $p \rightarrow q$.
- When two compound propositions always have the same truth value we call them **equivalent**, so that a conditional statement and its contrapositive are equivalent.
- Example: What are the contrapositive, the converse, and the inverse of the conditional statement "The Nepali team wins whenever it is raining?"
  - ✓ Solution: The "q whenever p" is one of the ways to express the conditional statement p → q, the original statement can be rewritten as "If it is raining, then the Nepali team wins."
  - ✓ Contrapositive of this conditional statement is "If the Nepali team does not win, then it is not raining."
  - ✓ The converse is "If the Nepali team wins, then it is raining."
  - ✓ The inverse is "If it is not raining, then the Nepali team does not win."

1.4 Compound Propositions

- **Compound propositions**: involving any number of propositional variables and logical connectives.
- Show that (P→ Q)∨ (Q→ P) is a tautology.

P	Q	$P \rightarrow Q$	$Q \rightarrow P$	$(P \rightarrow Q) \vee (Q \rightarrow P)$
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

- Construct the truth table for (P→ Q)∧ (Q→ R).

P	Q	R	$P \rightarrow Q$	$Q \rightarrow R$	$(P \rightarrow Q) \wedge (Q \rightarrow R)$
T	T	T	T	T	T
T	T	F	T	F	F
T	F	T	F	T	F
T	F	F	F	T	F
F	T	T	T	T	T
F	T	F	T	F	F
F	F	T	T	T	T
F	F	F	T	T	T

- Construct the truth table of the compound proposition(p ∨¬q) → (p ∧ q).

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

1.5 Logic and Bit Operations

- A **bit,** abbreviated for *b*inary dig*it*, is a symbol with two possible values 0 (zero) and 1 (one). Zeros and ones are the digits used in binary representations of numbers.
- Information is often represented using bit strings. A bit string is a sequence of zero or more bits. The length of this string is the number of bits in the string.
- We define the bitwise OR, bitwise AND, and bitwise XOR of two strings of the same length to be the strings that have as their bits the OR, AND, and XOR of the corresponding bits in the two strings, respectively.
- We use the symbols $\vee$,, and $\oplus$ to represent the bitwise OR, bitwise AND, and bitwise XOR operations, respectively.
- The truth table of bitwise operator

x	y	$x \vee y$	$x \wedge y$	$x \oplus y$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

- Find the bitwise OR, bitwise AND, and bitwise XOR of the bit strings 01 1011 0110 and 11 0001 1101.
  - ✓ Solution:

First String	01 1011 0110
Second String	11 0001 1101
Bitwise *OR*	11 1011 1111
Bitwise *AND*	01 0001 0100
Bitwise *XOR*	10 1010 1011

1.6 Propositional logic

- **Atomic proposition**: It is single sentence where each symbol stands for proposition that can be true or false.
  - ✓ **Example**: P means "It is hot." Q means "It is humid." R means "It is raining."
- **Compound proposition:** It is constructed from simple sentences using logical operators.
  - ✓ $(P \wedge Q) \rightarrow R$ "If it is hot and humid, then it is raining"

- A compound proposition that is always true, no matter what the truth values of the propositional variables that occur in it, is called a **tautology**.
  - ✓ Example: **p∨¬p** is always true regardless of the truth value of the proposition p.
- A compound proposition that is always false is called a **contradiction**.
  - ✓ Example: **p ∧¬p** is always false regardless of the truth value of the proposition p.
- A compound proposition that is neither a tautology nor a contradiction is called a **contingency.**
- **Logical Equivalences**: Compound propositions that have the same truth values in all possible cases are called logically equivalent.
- The compound propositions p and q are called logically equivalent if p ↔ q is a tautology. The notation p ≡ q denotes that p and q are logically equivalent.
- **Note:** The symbol ≡ is not a logical connective, and p ≡ q is not a compound proposition but rather is the statement that p ↔ q is a tautology. The symbol ⇔is sometimes used instead of ≡ to denote logical equivalence.
- Show that ¬ *(p ∨q)* and ¬*p* A¬*q* are logically equivalent.
  - ✓ Solution: the truth values of the compound propositions ¬(p ∨pq) and q agree for all possible combinations of the truth values of p and q, it follows that ¬(p ∨pq) ↔(q) is a tautology and that these compound propositions are logically equivalent.

p	*q*	*p* ∨ *q*	¬(*p* ∨ *q*)	¬ *p*	¬*q*	¬*p* ∧ ¬*q*
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

- Show that p ∨ (q ∧ r) and (p ∨ q) ∧ (p ∨ r) are logically equivalent.
  - ✓ Solution: these compound propositions are logically equivalent because the truth values of *p ∨(q* A*r)* and *(p ∨q)* A*(p ∨r)* agree.

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

Laws of Logical Equivalences

- In these equivalences, **T** denotes the compound proposition that is always true and **F** denotes the compound proposition that is always false.

Equivalence	*Name*
$p \wedge \mathbf{T} \equiv p$ $p \vee \mathbf{F} \equiv p$	Identity laws
$p \vee \mathbf{T} \equiv \mathbf{T}$ $p \wedge \mathbf{F} \equiv \mathbf{F}$	Domination laws
$p \vee p \equiv p$ $p \wedge p \equiv p$	Idempotent laws
$\neg(\neg p) \equiv p$	Double negation law
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Commutative laws
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative laws
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive laws
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan's laws
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$	Absorption laws
$p \vee \neg p \equiv \mathbf{T}$ $p \wedge \neg p \equiv \mathbf{F}$	Negation laws

- Logical equivalences involving Conditional Statements:

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

- Logical equivalences involving Biconditional Statements

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

De Morgan's Laws

- De Morgan's Laws state that the negation of a conjunction is formed by taking the disjunction of the negations of the component propositions. i.e. $\neg(p \wedge q) \equiv \neg p \vee \neg q$.
- Similarly, the negation of a disjunction is formed by taking the conjunction of the negations of the component propositions. i.e. $\neg(p \vee q) \equiv \neg p \wedge \neg q$

Propositional Satisfiability

- A compound proposition is **satisfiable** if there is an assignment of truth values to its variables that makes it true. When no such assignments exist, that is, when the compound proposition is false for all assignments of truth values to its variables, the compound proposition is **unsatisfiable**.
- Note that a compound proposition is unsatisfiable if and only if its negation is true for all assignments of truth values to the variables, that is, if and only if its negation is a tautology.
- Determine whether each of the compound propositions $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$, (p ∨ q ∨ r) A (¬p ∨¬q ∨¬r), and (p ∨¬q) A (q ∨¬r) A (r ∨¬p) (p ∨ q ∨ r) A (¬p ∨¬q ∨¬r) is satisfiable.
  - ✓ Solution: Note that (p ∨¬q) A (q ∨¬r) A (r ∨¬p) is true when the three variable p, q, and r have the same truth value. Hence, it is satisfiable as there is at least one assignment of truth values for p, q, and r that makes it true.

- ✓ Similarly, note that (p ∨ q ∨ r) ∧ (¬p ∨¬q ∨¬r) is true when at least one of p, q, and r is true and at least one is false. Hence, (p ∨ q ∨ r) ∧ (¬p ∨¬q ∨¬r) is satisfiable, as there is at least one assignment of truth values for p, q, and r that makes it true.
- ✓ Finally, note that for (p ∨¬q) ∧ (q ∨¬r) ∧ (r ∨¬p) ∧ (p ∨ q ∨ r) ∧ (¬p ∨¬q ∨¬r)to be true, (p∨¬q) ∧ (q ∨¬r) ∧ (r ∨¬p) and (p ∨ q ∨ r) ∧ (¬p ∨¬q ∨¬r) must both be true. For the first to be true, the three variables must have the same truth values, and for the second to be true, at least one of three variables must be true and at least one must be false. However, these conditions are contradictory. From these observations we conclude that no assignment of truth values to p, q, and r makes (p ∨¬q) ∧ (q ∨¬r) ∧ (r ∨¬p) ∧ (p ∨ q ∨ r) ∧ (¬p ∨¬q ∨¬r) true. Hence, it is unsatisfiable.

- **Applications of Satisfiability:** Many problems, in diverse areas such as robotics, software testing, computer-aided design, machine vision, integrated circuit design, computer networking, and genetics, can be modeled in terms of propositional satisfiability.

1.7 Rules of Inference

- An **argument** in propositional logic is a sequence of propositions. All but the final proposition in the argument are called **premises** and the final proposition is called the **conclusion**.
- An argument is **valid** if the truth of all its premises implies that the conclusion is true.
- An argument form in propositional logic is a sequence of compound propositions involving propositional variables. An argument form is valid no matter which particular propositions are substituted for the propositional variables in its premises; the conclusion is true if the premises are all true.
- **Proofs**: Proofs are valid arguments that establish the truth of mathematical statements.
- **Fallacies:** It is the common forms of incorrect reasoning which lead to invalid arguments.
- **Rules of inference** are our basic tools for establishing the truth of statements.
  - ✓ **Modus Ponens**: Whenever any sentences of the form $P \rightarrow Q$ and P are given, then the sentence Q can be inferred.

$$\frac{P \rightarrow Q,\ P}{Q}$$

It means P implies Q; P is true; therefore Q holds.

- Example 1: Ram is a OOP major. Therefore, Ram is either a OOP major or a DSA major.
  - ✓ Solution: P: Ram is a OOP major & Q: Ram is a DSA major

P

------- *addition rule*

P V Q

- Example 2:Hari is a computer science major and a mathematics major. Therefore, Hari is a mathematics major.
  - ✓ Solution: P: Hari is a mathematics major & Q: Ram is a computer science major

P ∧ Q

------- *simplification rule*

P

- **Rule of Inference**

Rule of Inference	*Tautology*	*Name*
p $p \rightarrow q$ $\therefore q$	$(p \wedge (p \rightarrow q)) \rightarrow q$	Modus ponens
$\neg q$ $p \rightarrow q$ $\therefore \neg p$	$(\neg q \wedge (p \rightarrow q)) \rightarrow \neg p$	Modus tollens
$p \rightarrow q$ $q \rightarrow r$ $\therefore p \rightarrow r$	$((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$	Hypothetical syllogism
$p \vee q$ $\neg p$ $\therefore q$	$((p \vee q) \wedge \neg p) \rightarrow q$	Disjunctive syllogism
p $\therefore p \vee q$	$p \rightarrow (p \vee q)$	Addition
$p \wedge q$ $\therefore p$	$(p \wedge q) \rightarrow p$	Simplification
p q $\therefore p \wedge q$	$((p) \wedge (q)) \rightarrow (p \wedge q)$	Conjunction
$p \vee q$ $\neg p \vee r$ $\therefore q \vee r$	$((p \vee q) \wedge (\neg p \vee r)) \rightarrow (q \vee r)$	Resolution

- Example 3: If it is snows today, we will go snow tubing. We didn't go snow tubing. Therefore, it didn't snow today.
  - ✓ Solution: P: It is snows today & Q: We will go snow tubing

    $P \rightarrow Q$
    $\neg Q$
    ------- Modus Tollens
    $\neg P$
- Example 4: Show that the hypotheses "Ram works hard," "If Ram works hard, and then he is a dull boy," and "If Ram is a dull boy, and then he will not get the job" imply the conclusion "Ram will not get the job."
  - ✓ Solution: P: Ram works hard, Q: Ram is a dull boy & R: Ram will not get the job.

    1. P
    2. $P \rightarrow Q$
    3. $Q \rightarrow R$
    4. Q from 1 and 2 by MP
    5. R from 3 and 4 by MP

- Example 5: Show that the hypotheses "Ram is skiing or it is not snowing" and "It is snowing or Hari is playing hockey" imply that "Ram is skiing or Hari is playing hockey".
  - ✓ Solution: Q: Ram is skiing P: it is snowing R: Hari is playing hockey

    ¬P V Q
    P V R
    ---------- Resolution
    Q V R
- Example 6: If it rains today, then we will not have a barbecue today. If we do not have a barbecue today, then we will have a barbecue tomorrow. Therefore, if it rains today, then we will have a barbecue tomorrow.
  - ✓ Solution: p:It is raining today, q:We will not have a barbecue today and r:We will have a barbecue tomorrow

    $p \rightarrow q$
    $q \rightarrow r$
    ---------- Hypothetical syllogism
    $\therefore p \rightarrow r$
- Example 7: Show that the premises "It is not sunny this afternoon and it is colder than yesterday," "We will go swimming only if it is sunny," "If we do not go swimming, then we will take a canoe trip," and "If we take a canoe trip, then we will be home by sunset" lead to the conclusion "We will be home by sunset."
  - ✓ Solution: p: "It is sunny this afternoon," q: "It is colder than yesterday," r: "We will go swimming," s: "We will take a canoe trip," and t: "We will be home by sunset."
  - ✓ Then the premises become $\neg p \wedge q$, $r \rightarrow p$, $\neg r \rightarrow s$, and $s \rightarrow t$. The conclusion is simply t.

Step	Reason
1. $\neg p \wedge q$	Premise
2. $\neg p$	Simplification using (1)
3. $r \rightarrow p$	Premise
4. $\neg r$	Modus tollens using (2) and (3)
5. $\neg r \rightarrow s$	Premise
6. s	Modus ponens using (4) and (5)
7. $s \rightarrow t$	Premise
8. t	Modus ponens using (6) and (7)

- Example 8: Show that the premises "If you send me an e-mail message, then I will finish writing the program," "If you do not send me an e-mail message, then I will go to sleep early," and "If I go to sleep early, then I will wake up feeling refreshed" lead to the conclusion "If I do not finish writing the program, then I will wake up feeling refreshed."
  - ✓ Solution: p: "You send me an e-mail message," q: "I will finish writing the program," r: "I will go to sleep early," and s: "I will wake up feeling refreshed."
  - ✓ Then the premises are $p \rightarrow q$, $\neg p \rightarrow r$, and $r \rightarrow s$. The desired conclusion is $\neg q \rightarrow s$.

Step	Reason
1. $p \rightarrow q$	Premise

2. ¬q →¬p	Contrapositive of (1)
3. ¬p → r	Premise
4. ¬q → r	Hypothetical syllogism using (2) and (3)
5. r → s	Premise
6. ¬q → s	Hypothetical syllogism using (4) and (5)

Assignment: Use rules of inference to show that the hypotheses "If it does not rain or if it is not foggy, then the sailing race will be held and the lifesaving demonstration will go on," If the sailing race is held, then the trophy will be awarded," and "The trophy was not awarded" imply the conclusion "It rained."

1.8 Predicates and Quantifiers

- Propositional logic cannot adequately express the meaning of all statements in mathematics and in natural language. For example "Everyone likes some kind of food."
- No rules of propositional logic allow us to conclude the truth of the statement "All triangles have 3 sides."
- Propositional logic can't directly talk about properties of individuals or relations between individuals e.g., "Bill is tall".
- The predicate logic can be used to express the meaning of a wide range of statements in mathematics and computer science in ways that permit us to reason and explore relationships between objects.

Predicates

- Statements involving variables, such as "x > 3" is often found in mathematical assertions, in computer programs, and in system specifications. These statements are neither true nor false when the values of the variables are not specified. This is the way how propositions can be produced from this statement.
  - ✓ The statement "x is greater than 3" has two parts. **The first part**, the variable x, is the subject of the statement. **The second part**—the predicate, "is greater than 3"—refers to a property that the subject of the statement can have.
  - ✓ We can denote the statement "x is greater than 3" by P(x), where P denotes the predicate "is greater than 3" and x is the variable.
  - ✓ The statement P(x) is also said to be the value of the propositional function P at x.
  - ✓ Once a value has been assigned to the variable x, the statement P(x) becomes a proposition and has a truth value.
- Example1: Let P(x) denote the statement "x > 3." What are the truth values of P (4) and P (2)?
  - ✓ Solution: We obtain the statement P (4) by setting x = 4 in the statement "x > 3." Hence, P (4), which is the statement "4 > 3," is true. However, P (2), which is the statement "2 > 3,"is false.
- Example 2: Let A (x) denote the statement "Computer x is under attack by an intruder." Suppose that of the computers on campus, only CS2 and MATH1 are currently under attack by intruders. What are truth values of A (CS1), A (CS2), and A (MATH1)?
  - ✓ Solution: We obtain the statement A (CS1) by setting x = CS1 in the statement "Computer x is under attack by an intruder." Because CS1 is not on the list of computers currently under attack, we conclude that A (CS1) is false. Similarly,

because CS2 and MATH1 are on the list of computers under attack, we know that A (CS2) and A (MATH1) are true.

- Example3: Let Q (x, y) denote the statement "x = y + 3." What are the truths values of the propositions Q (1, 2) and Q (3, 0)?
  - ✓ Solution: To obtain Q (1, 2), set x = 1 and y = 2 in the statement Q(x, y). Hence, Q (1, 2) is the statement "1 = 2 + 3," which is false. The statement Q (3, 0) is the proposition "3 = 0 + 3," which is true.

Quantifiers

- Quantification expresses the extent to which a predicate is true over a range of elements. In English, the words all, some, many, none, and few are used in quantifications.
- There are two types of quantification **universal quantification, and existential quantification.**
- The area of logic that deals with predicates and quantifiers is called the predicate calculus.

The Universal Quantifier

- The universal quantification of P(x) is the statement
  "P(x) for all values of x in the domain."
- The notation $\forall$ xP(x) denotes the universal quantification of P(x).
- Here $\forall$ is called the universal quantifier.
- We read $\forall$ xP(x) as "for all xP(x)" or "for every xP(x)."
- An element for which P(x) is false is called a **counterexample** of $\forall$ xP(x).
- Typically" $\rightarrow$" is the main connective with$\forall$.
- Besides "for all" and "for every," universal quantification can be expressed in many other ways, including "all of," "for each," "given any," "for arbitrary," "for each," and "for any."
- It is best to avoid using "for any x" because it is often ambiguous as to whether "any" means "every" or "some." In some cases, "any" is unambiguous, such as when it is used in negatives, for example, "there is not any reason to avoid studying."
- Example1: What is the truth value of $\forall x(x^2 \geq x)$ if the domain consists of all real numbers? What is the truth value of this statement if the domain consists of all integers?
  - ✓ Solution: The universal quantification $\forall x(x^2 \geq x)$, where the domain consists of all real numbers, is false. For example, $(1/2)^2 \geq 1/2$. Note that $x^2 \geq x$ if and only if $x^2 - x = x(x - 1) \geq 0$.
  - ✓ Consequently, $x^2 \geq x$ if and only if $x \leq 0$ or $x \geq 1$. It follows that $\forall x(x^2 \geq x)$ is false if the domain consists of all real numbers (because the inequality is false for all real numbers x with $0 < x < 1$).
  - ✓ However, if the domain consists of the integers, $\forall x(x^2 \geq x)$ is true, because there are no integers x with $0 < x < 1$.

The Existential Quantifier

- The existential quantification of P(x) is the proposition
  "There exists an element x in the domain such that P(x)."
- The notation $\exists$xP(x) use for the existential quantification of P(x).
- Here $\exists$ is called the existential quantifier.
- Typically, "$\wedge$" is the main connective with $\exists$.

- Besides the phrase "there exists, "we can also express existential quantification in many other ways, such as by using the words "for some," "for at least one," or "there is." The existential quantification ∃xP(x) is read as
  "There is an x such that P(x),"
  "There is at least one x such that P(x)," or
  "For some xP(x)."

  The meanings of the quantifiers are summarized in below table.

Statement	*When True?*	*When False?*
$\forall x\, P(x)$	$P(x)$ is true for every x.	There is an x for which $P(x)$ is false.
$\exists x\, P(x)$	There is an x for which $P(x)$ is true.	$P(x)$ is false for every x.

Precedence of Quantifiers: The quantifiers ∀ and ∃ have higher precedence than all logical operators from propositional calculus. For example, ∀xP(x) ∨ Q(x) is the disjunction of ∀xP(x) and Q(x). In other words, it means (∀xP(x)) ∨ Q(x) rather than ∀x (P(x) ∨ Q(x)).

Logical Equivalences Involving Quantifiers

- Statements involving predicates and quantifiers are logically equivalent if and only if they have the same truth value no matter which predicates are substituted into these statements and which domain of discourse is used for the variables in these propositional functions.
- We use the notation S ≡ T to indicate that two statements S and T involving predicates and quantifiers are logically equivalent.
- **De Morgan's laws for quantifiers**

Negation	*Equivalent Statement*	*When Is Negation True?*	*When False?*
$\neg\exists x\, P(x)$	$\forall x \neg P(x)$	For every x, $P(x)$ is false.	There is an x for which $P(x)$ is true.
$\neg\forall x\, P(x)$	$\exists x \neg P(x)$	There is an x for which $P(x)$ is false.	$P(x)$ is true for every x.

- Example 1: What are the negations of the statements $\forall x(x^2 > x)$ and $\exists x(x^2 = 2)$?
  - ✓ Solution: The negation of $\forall x(x^2 > x)$ is the statement $\neg\forall x(x^2 > x)$, which is equivalent to $\exists x \neg(x^2 > x)$. This can be rewritten as $\exists x(x^2 \leq x)$.
  - ✓ The negation of $\exists x(x^2 = 2)$ is the statement $\neg\exists x(x^2 = 2)$, which is equivalent to $\forall x \neg(x^2 = 2)$. This can be rewritten as $\forall x(x^2 \neq 2)$. The truth values of these statements depend on the domain.
- Example 2: Show that $\neg\forall x\,(P(x) \rightarrow Q(x))$ and $\exists x(P(x) \wedge \neg Q(x))$ are logically equivalent.
  - ✓ Solution: By De Morgan's law for universal quantifiers, we know that $\neg\forall x\,(P(x) \rightarrow Q(x))$ and $\exists x\,(\neg(P(x) \rightarrow Q(x)))$ are logically equivalent. We know that $\neg(P(x) \rightarrow Q(x))$ and $P(x) \wedge \neg Q(x)$ are logically equivalent for every x. Because we

can substitute one logically equivalent expression for another in a logical equivalence, it follows that ¬∀x (P(x) → Q(x)) and ∃x (P(x) ∧¬Q(x)) are logically equivalent.

Translating from English into Logical Expressions

1. Example1: Every student in this class has studied Discrete Structure.
   Solution: ∀x(S(x) → D(x))
   D(x): "x has studied Discrete Structure"
   S(x): student x is in this class
2. Example2: Everyone at HCOE is smart: ∀x At(x , HCOE) → Smart(x)
3. Example3: Everyone likes McDonalds: ∀x, likes(x, McDonalds)
4. Example4: All children like McDonalds:∀x, child(x) → likes(x, McDonalds)
5. Common mistake: using "∧" as the main connective with∀. Example: ∀x At(x , HCOE) ∧ Smart(x) means "Everyone is at HCOE and everyone is smart"
6. Example6: "Some student in this class has visited Mexico" : ∃x(S(x) ∧ M(x))
   M(x): "x has visited Mexico.
7. Every student in this class has visited either Canada or Mexico. : ∀x(S(x)→ (C(x) ∨M(x))). C(x): "x has visited Canada.
8. Example7: Someone at HCOE is smart:∃x At(x , HCOE) ∧ Smart(x).
9. Example8: Someone likes McDonalds: ∃x likes(x, McDonalds).
10. Common mistake: using "→" as the main connective with∃. Example: ∃x At(x, HCOE) → Smart(x) is true "if there is anyone who is not at HCOE".

Nested Quantifiers

- It is important to note that the order of the quantifiers is important, unless all the quantifiers are universal quantifiers or all are existential quantifiers.
- Quantifications of Two Variables

Statement	*When True?*	*When False?*
$\forall x \forall y P(x, y)$ $\forall y \forall x P(x, y)$	$P(x, y)$ is true for every pair x, y.	There is a pair x, y for which $P(x, y)$ is false.
$\forall x \exists y P(x, y)$	For every x there is a y for which $P(x, y)$ is true.	There is an x such that $P(x, y)$ is false for every y.
$\exists x \forall y P(x, y)$	There is an x for which $P(x, y)$ is true for every y.	For every x there is a y for which $P(x, y)$ is false.
$\exists x \exists y P(x, y)$ $\exists y \exists x P(x, y)$	There is a pair x, y for which $P(x, y)$ is true.	$P(x, y)$ is false for every pair x, y.

11. Example9: The sum of two positive integers is always positive.
    ∀x ∀y ((x > 0) ∧ (y > 0) → (x +y > 0))
12. Example10: Every real number except zero has a multiplicative inverse." (A multiplicative inverse of a real number x is a real number y such that xy = 1.)
    ∀x ((x≠0) → ∃y(xy = 1)).

13. Example11: Every student at in your college has a computer or has a friend who has a computer. $\forall x(hasComp(x) \vee \exists y(hasComp(y) \wedge areFriend(x, y)))$
14. Example12: "If a person is female and is a parent, then this person is someone's mother".
    $\forall x ((Female(x) \wedge Parent(x)) \rightarrow \exists y Mother(x, y))$ or
    $\forall x \exists y ((Female(x) \wedge Parent(x)) \rightarrow Mother(x, y))$
    [Because *y* does not appear in $Female(x) \wedge Parent(x)$]
15. Example13: Everyone has exactly one best friend.
    $\forall x \exists y (Bestfren(x, y) \wedge \forall z ((z \neq y) \rightarrow \neg Bestfren(x, z)))$.
    [To say that x has exactly one best friend means that there is a person y who is the best friend of x, and furthermore, that for every person z, if person z is not person y, then z is not the best friend of x.]

1.9 Introduction to Proofs

- A **proof** is a valid argument that establishes the truth of a mathematical statement. A proof can use the hypotheses of the theorem, if any, axioms assumed to be true and previously proven theorems.
- **Formal proofs:** formal proofs are statements involving propositions and quantified statements which are true, where all steps were supplied, and the rules for each step in the argument were given.
- **Informal proofs** may be used more than one rule of inference in each step, where steps may be skipped, where the axioms being assumed and the rules of inference used are not explicitly stated.
- A less important theorem that is helpful in the proof of other results is called a **lemma.**
- A **corollary** is a theorem that can be established directly from a theorem that has been proved.
- A **conjecture** is a statement that is being proposed to be a true statement, usually on the basis of some partial evidence, a heuristic argument, or the intuition of an expert.

Direct Proofs

- A direct proof of a conditional statement $p \rightarrow q$ is constructed when the first step is the assumption that p is true; subsequent steps are constructed using rules of inference, with the final step showing that q must also be true.
- A direct proof shows that a conditional statement $p \rightarrow q$ is true by showing that if p is true, then q must also be true, so that the combination p true and q false never occurs.
- In a direct proof, we assume that p is true and use axioms, definitions, and previously proven theorems, together with rules of inference, to show that q must also be true.
- Exaple1: Give a direct proof of the theorem "If n is an odd integer, then n^2 is odd."
  - ✓ Solution: we assume that the hypothesis of this conditional statement is true, and assume that n is odd. By the definition of an odd integer $n = 2k + 1$, where k is some integer. We want to show that n^2 is also odd. We can square both sides of the equation $n = 2k + 1$ to obtain a new equation that expresses n^2.

    $$n^2 = (2k + 1)^2$$
    $$= 4k^2 + 4k + 1$$
    $$= 2(2k^2 + 2k) + 1$$

  - ✓ By the definition of an odd integer, we can conclude that n2 is an odd integer. Consequently, we have proved that if n is an odd integer, then n2 is an odd integer.

- Example 2: Give a direct proof that if m and n are both perfect squares, then nm is also a perfect square. (An integer is a perfect square if there is an integer b such that a = b2.)
  - ✓ Solution: we assume that the hypothesis of this conditional statement is true and assume that m and n are both perfect squares. By the definition of a perfect square, it follows that there are integers s and t such that $m = s^2$ and $n = t^2$. We have to show that mn must also be a perfect square when m and n are. Now substituting s^2 for m and t^2 for n into mn.

$$\begin{aligned} mn &= s^2t^2 \\ &= (ss)(t\,t) \\ &= (st)(st) \\ &= (st)^2 \end{aligned}$$

  - ✓ Using commutativity and associativity of multiplication. By the definition of perfect square, it follows that mn is also a perfect square, because it is the square of st, which is an integer.

Indirect Proofs

- Indirect Proofs: It does not start with the premises and end with the conclusion. An extremely useful type of indirect proof is known as proof by contraposition.
- **Proofs by contraposition** make use of the fact that the conditional statement $p \rightarrow q$ is equivalent to its contrapositive, $\neg q \rightarrow \neg p$. This means that the conditional statement $p \rightarrow q$ can be proved by showing that its contrapositive, $\neg q \rightarrow \neg p$, is true.
- In a proof by contraposition of $p \rightarrow q$, we take $\neg q$ as a premise, and using axioms, definitions, and previously proven theorems, together with rules of inference, we show that $\neg p$ must follow.
- Example 3: Prove that if n is an integer and 3n + 2 is odd, then n is odd.
  - ✓ Solution: Assume that the conclusion of the conditional statement "If 3n + 2 is odd, then n is odd" is false; and assume that n is even. Then, by the definition of an even integer, n = 2k for some integer k. Substituting 2k for n,

$$\begin{aligned} 3n + 2 &= 3(2k) + 2 \\ &= 6k + 2 \\ &= 2(3k + 1) \end{aligned}$$

  - ✓ This tells us that 3n + 2 is even and therefore not odd. This is the negation of the premise of the theorem. Because the negation of the conclusion of the conditional statement implies that the hypothesis is false, the original conditional statement is true. Our proof by contraposition succeeded; we have proved the theorem "If 3n + 2 is odd, then n is odd."

- **Proofs by Contradiction:** Suppose we want to prove that a statement p is true. Furthermore, suppose that we can find a contradiction q such that $\neg p \rightarrow q$ is true. Because q is false, but $\neg p \rightarrow q$ is true, we can conclude that $\neg p$ is false, which means that p is true.
- Prove that: $\sqrt{2}$ is irrational by giving a proof by contradiction. A rational number is any number that can be expressed as fraction p/q of two integers, p and q, with the denominator q not equal to zero. All numbers that are not rational are considered

irrational. An irrational number has endless non-repeating digits to the right of the decimal point.

- ✓ Solution: If$\sqrt{2}$ is rational, there exist integers a and b with $\sqrt{2} = a/b$, where $b \neq 0$ and a and b have no common factors so that the fraction a/b is in lowest terms. $\sqrt{2} = a/b$, when both sides of this equation are squared, it follows that

$$2 = \frac{a2}{b2}$$

Hence, $2b^2 = a^2$

- ✓ By the definition of an even integer it follows that a^2 is even. We next use the fact that if a^2 is even, a must also be even. We can write a = 2c for some integer c. Thus, $2b^2 = 4c^2$. Dividing both sides of this equation by 2 gives $b^2 = 2c^2$. Here b^2 is even. Again using the fact that if the square of an integer is even, then the integer itself must be even, we conclude that b must be even as well. We have now shown that the assumption leads to the equation $\sqrt{2} = a/b$, where a and b have no common factors, but both a and b are even, that is, 2 divides both a and b. Note that the statement that $\sqrt{2} = a/b$, where a and b have no common factors, means, in particular, that 2 does not divide both a and b. Because our assumption leads to the contradiction that 2 divides both a and b and 2 does not divide both a and b, the statement must be false. That is, the statement p, "$\sqrt{2}$ is irrational," is true.

- **Vacuous and Trivial Proofs**: We can quickly prove that a conditional statement $p \rightarrow q$ is true when we know that p is false, because $p \rightarrow q$ must be true when p is false. Consequently, if we can show that p is false, then we have a proof, called a vacuous proof, of the conditional statement $p \rightarrow q$. Vacuous proofs are often used to establish special cases of theorems that state that a conditional statement is true for all positive integers

1.10 Proof by Mathematical Induction

- **Principle Of Mathematical Induction**: To prove that P (n) is true for all positive integers n, where P (n) is a propositional function, we complete two steps:
  1. **Basis Step**: We verify that P (1) is true.
  2. **Inductive Step:** We show that the conditional statement $P(k) \rightarrow P(k + 1)$ is true for all positive integers k.
- To complete the inductive step of a proof using the principle of mathematical induction, we assume that P (k) is true for an arbitrary positive integer k and show that under this assumption, P (k + 1) must also be true. The assumption that P (k) is true is called the **inductive hypothesis.**
- **Steps for Proofs by Mathematical Induction**
  1. Express the statement that is to be proved in the form "for all $n \geq b$, P (n)" for a fixed integer b.
  2. Write out the words "Basis Step." Then show that P (b) is true; taking care that the correct value of b is used. This completes the first part of the proof.
  3. Write out the words "Inductive Step."

4. State, and clearly identify, the inductive hypothesis, in the form "assume that P(k) is true for an arbitrary fixed integer $k \geq b$."
5. State what needs to be proved under the assumption that the inductive hypothesis is true. That is, write out what P (k + 1) says.
6. Prove the statement P (k + 1) making use the assumption P (k). Be sure that your proof is valid for all integers k with $k \geq b$, taking care that the proof works for small values of k, including k = b.
7. Clearly identify the conclusion of the inductive step, such as by saying "this completes the inductive step."
8. After completing the basis step and the inductive step, state the conclusion, namely that by mathematical induction, P(n) is true for all integers n with $n \geq b$.

- Example 1: Prove by mathematical induction :

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

✓ Solution: Let P (n) be the sum of the first n positive integers.
Basis Step: P (1) is true, because $1 = \frac{1(1+1)}{2}$

Inductive Step: For the inductive hypothesis we assume that P (k) holds for an arbitrary positive integer k. That is, we assume that

$$1 + 2 + \cdots + k = \frac{k(k+1)}{2}$$

Under this assumption, it must be shown that P (k + 1) is true

$$1 + 2 + \cdots + k + (k+1) = \frac{(k+1)[(k+1)+1]}{2}$$

$$= \frac{(k+1)(k+2)}{2}$$

Now we obtain,

$$1 + 2 + \cdots + k + (k+1) = k(k+1)/2 + (k+1)$$

$$= \frac{k(k+1) + 2(k+1)}{2}$$

$$= \frac{(k+1)(k+2)}{2}$$

This last equation shows that P (k + 1) is true under the assumption that P (k) is true.

- Example 2: Use mathematical induction to show that $1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$ for all nonnegative integers n.

✓ Solution: Let P(n): $1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$ for the integer n.
Basis Step: P(0) is true because $2^0 = 1 = 2^1 - 1$.
Inductive Step: For the inductive hypothesis, assume that P(k) is true for an arbitrary nonnegative integer k. That is, we assume that

$$1 + 2 + 2^2 + \cdots + 2^k = 2^{k+1} - 1$$

Under this assumption, it must be shown that P (k + 1) is true

$$1 + 2 + 2^2 + \cdots + 2^k + 2^{k+1} = 2^{(k+1)+1} - 1 = 2^{k+2} - 1$$

Now we obtain

$$1 + 2 + 2^2 + \cdots + 2^k + 2^{k+1} = (1 + 2 + 2^2 + \cdots + 2^k) + 2^{k+1}$$
$$= (2^{k+1} - 1) + 2^{k+1}$$
$$= 2 \cdot 2^{k+1} - 1$$
$$= 2^{k+2} - 1$$

The induction step is complete. Because we have completed the basis step and the inductive step, by mathematical induction we know that P(n) is true for all nonnegative integers n. That is, $1 + 2 + 22 + \cdots + 2n = 2^{n+1} - 1$ for all nonnegative integers n.

- Example 3: Use mathematical induction to prove the inequality $n < 2^n$ for all positive integers n.
  - ✓ Solution: Let P(n) : $n < 2n$

    Basis Step: P(1) is true, because $1 < 2^1 = 2$

    Inductive Step: For the inductive hypothesis, assume that P(k) is true for an arbitrary all positive integers k. That is, we assume that

    $$k < 2^k$$

    Under this assumption, it must be shown that P (k + 1) is true

    $$k + 1 < 2^{k+1}$$

    To show this, first add 1 to both sides of $k < 2^k$, and then note that $1 \leq 2^k$. Then

    $$k + 1 < 2^k + 1$$
    $$\leq 2^k + 2^k$$
    $$= 2 \cdot 2^k$$
    $$= 2^{k+1}$$

    The induction step is complete. Therefore, because we have completed both the basis step and the inductive step, by the principle of mathematical induction we have shown that $n < 2^n$ is true for all positive integers n.
- Example 4: Use mathematical induction to prove that $7^{n+2} + 8^{2n+1}$ is divisible by 57 for every nonnegative integer n.
  - ✓ Solution: let P(n): 7n+2 + 82n+1 is divisible by 57.

    Basis Step: P(0) is true because $7^{0+2} + 8^{2 \cdot 0+1} = 7^2 + 8^1 = 57$ is divisible by 57.

    Inductive Step: For the inductive hypothesis, assume that P(k) is true for an arbitrary all positive integers k. That is, we assume that

    $7^{k+2} + 8^{2k+1} = 57(t)$, where t is an integer (also divisible by 57)

    Under this assumption, it must be shown that P (k + 1) is true

    $7^{(k+1)+2} + 8^{2(k+1)+1}$ is divisible by 57 is also true.

    Now can obtain

    $$7^{(k+1)+2} + 8^{2(k+1)+1} = 7^{k+3} + 8^{2k+3}$$
    $$= 7 \cdot 7^{k+2} + 8^2 \cdot 8^{2k+1}$$
    $$= 7 \cdot 7^{k+2} + 64 \cdot 8^{2k+1}$$
    $$= 7 \cdot 7^{k+2} + 7 \cdot 8^{2k+1} + 57 \cdot 8^{2k+1}$$
    $$= 7(7^{k+2} + 8^{2k+1}) + 57 \cdot 8^{2k+1}$$
    $$= 7(57(t)) + 57 \cdot 8^{2k+1}$$; from induction step
    $$= 57\,(7.t + 8^{2k+1})$$

    $57\,(7.t + 8^{2k+1})$ this is the multiple of 57.This completes the inductive step. Because we have completed both the basis step and the inductive step, by the

principle of mathematical induction we know that 7 n+2 + 8 2n+1 is divisible by 57 for every nonnegative integer n.

Unit 3: Automata Theory

Finite State Automata:-

Deterministic finite automaton (D.F.A)

- In the automata theory, a branch of theoretical computer science, a deterministic finite automaton (DFA)—also known as deterministic finite state machine—is a finite state machine that accepts/rejects finite strings of symbols and only produces a unique computation (or run) of the automaton for each input string. 'Deterministic' refers to the uniqueness of the computation.
- A DFA has a start state (denoted graphically by an arrow coming in from nowhere) where computations begin, and a set of accept states (denoted graphically by a double circle) which help define when a computation is successful.
- A DFA is defined as an abstract mathematical concept, but due to the deterministic nature of a DFA, it is implementable in hardware and software for solving various specific problems. For example, a DFA can model a software that decides whether or not online user-input such as email addresses are valid.
- DFAs recognize exactly the set of regular languageswhich are, among other things, useful for doing lexical analysis and pattern matching. DFAs can be built from nondeterministic finite automata through the powerset construction.

Formal definition

A deterministic finite automaton M is a 5-tuple, **$(Q, \Sigma, \delta, q_0, F)$** consisting of

- a finite set of states (Q)
- a finite set of input symbols called the alphabet (Σ)
- a transition function ($\delta : Q \times \Sigma \rightarrow Q$)
- a start state ($q_0 \in Q$)
- a set of accept states ($F \in Q$)

  Let w = a1a2 ... an be a string over the alphabet Σ. The automaton M accepts the string w if a sequence of states, $r_0, r_1, ..., r_n$, exists in Q with the following conditions:

- $r_0 = q_0$
- $r_{i+1} = \delta(r_i, a_{i+1})$, for i = 0, ..., n−1
- $r_n \in F$.

  In words, the first condition says that the machine starts in the start state q_0. The second condition says that given each character of string w, the machine will transition from state to state according to the transition function δ. The last condition says that the machine accepts w if the last input of w causes the machine to halt in one of the accepting states. Otherwise, it is said that the automaton rejects the string. The set of strings M accepts is the language recognized by M and this language is denoted by L(M).

Transition Function Of DFA

A deterministic finite automaton without accept states and without a starting state is known as a transition system or semi automaton.

Given an input symbol $a \in \Sigma$, one may write the transition function as $\delta_a : Q \rightarrow Q$, using the simple trick of currying, that is, writing $\delta(q, a) = \delta_a(q)$ for all $q \in Q$. This way, the transition function can be seen in simpler terms: it's just something that "acts" on a state in Q, yielding another state. One may then consider the result of function composition repeatedly applied to the various functions δ_a, δ_b, and so on. Using this notion we define

$\widehat{\delta} : Q \times \Sigma^{\star} \to Q$. Given a pair of letters $a, b \in \Sigma$, one may define a new function $\widehat{\delta}$, by insisting that $\widehat{\delta}_{ab} = \delta_a \circ \delta_b$, where $\circ$ denotes function composition. Clearly, this process can be recursively continued. So, we have following recursive definition

$\widehat{\delta}(q, \epsilon) = q$ where ϵ is empty string and

$\widehat{\delta}(q, wa) = \delta_a(\widehat{\delta}(q, w))$ where $w \in \Sigma^*, a \in \Sigma$ and $q \in Q$.

- $\hat{\delta}$ is defined for all words $w \in \Sigma^*$

Advantages and disadvantages

- DFAs were invented to model real world finite state machines in contrast to the concept of a Turing machine, which was too general to study properties of real world machines.
- DFAs are one of the most practical models of computation, since there is a trivial linear time, constant-space, online algorithm to simulate a DFA on a stream of input. Also, there are efficient algorithms to find a DFA recognizing:

1. the complement of the language recognized by a given DFA.
2. the union/intersection of the languages recognized by two given DFAs.

- Because DFAs can be reduced to a canonical form (minimal DFAs), there are also efficient algorithms to determine:

1. whether a DFA accepts any strings
2. whether a DFA accepts all strings
3. whether two DFAs recognize the same language
4. the DFA with a minimum number of states for a particular regular language

- DFAs are equivalent in computing power to nondeterministic finite automata (NFAs). This is because, firstly any DFA is also an NFA, so an NFA can do what a DFA can do. Also, given an NFA, using the powerset construction one can build a DFA that

recognizes the same language as the NFA, although the DFA could have exponentially larger number of states than the NFA.

- On the other hand, finite state automata are of strictly limited power in the languages they can recognize; many simple languages, including any problem that requires more than constant space to solve, cannot be recognized by a DFA.
- The classical example of a simply described language that no DFA can recognize is bracket language, i.e., language that consists of properly paired brackets such as word "(( )( ))".
- No DFA can recognize the bracket language because there is no limit to recursion, i.e., one can always embed another pair of brackets inside.
- It would require an infinite amount of states to recognize. Another simpler example is the language consisting of strings of the form $a_n b_n$—some finite number of a's, followed by an equal number of b's.

Nondeterministic finite automaton (N.F.A)

- In the automata theory, a nondeterministic finite automaton (NFA) or nondeterministic finite state machine is a finite state machine where from each state and a given input symbol the automaton may jump into several possible next states.
- This distinguishes it from the deterministic finite automaton (DFA), where the next possible state is uniquely determined.
- Although the DFA and NFA have distinct definitions, a NFA can be translated to equivalent DFA using powerset construction, i.e., the constructed DFA and the NFA recognize the same formal language. Both types of automata recognize only regular languages
- A NFA is represented formally by a 5-tuple, (Q, Σ, Δ, q0, F), consisting of

1. a finite set of states Q
2. a finite set of input symbols Σ
3. a transition relation $\Delta : Q \times \Sigma \rightarrow P(Q)$.
4. an initial (or start) state q0

5. a set of states F distinguished as accepting (or final) states F .

- Here, P(Q) denotes the power set of Q. Let w = a1a2 ... an be a word over the alphabet Σ. The automaton M accepts the word w if a sequence of states, r0,r1, ..., rn, exists in Q with the following conditions:

1. r0 = q0
2. ri+1 ∈ Δ(ri, ai+1), for i = 0, ..., n−1
3. $r_n \in F$.

Implementation

There are many ways to implement a NFA:

- Convert to the equivalent DFA. In some cases this may cause exponential blowup in the size of the automaton and thus auxiliary space proportional to the number of states in the NFA (as storage of the state value requires at most one bit for every state in the NFA)[4]
- Keep a set data structure of all states which the machine might currently be in. On the consumption of the last input symbol, if one of these states is a final state, the machine accepts the string.
- In the worst case, this may require auxiliary space proportional to the number of states in the NFA; if the set structure uses one bit per NFA state, then this solution is exactly equivalent to the above.

- Create multiple copies. For each n way decision, the NFA creates up to $n - 1$copies of the machine. Each will enter a separate state.
- If, upon consuming the last input symbol, at least one copy of the NFA is in the accepting state, the NFA will accept. (This, too, requires linear storage with respect to the number of NFA states, as there can be one machine for every NFA state.)
- Explicitly propagate tokens through the transition structure of the NFA and match whenever a token reaches the final state. This is sometimes useful when the NFA should encode additional context about the events that triggered the transition.

Decision property of Regular Language

A decision property for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.

- Example: Is language L empty?
- You might imagine that the language isdescribed informally, so if the description is “the empty language”then yes, otherwise no.But the representation is a DFA (or aRE that you will convert to a DFA).

Closure Properties

- A closure property of a language classsays that given languages in the class,an operator (e.g., union) producesanother language in the same class.
- Example: the regular languages areobviously closed under union,concatenation, and (Kleene) closure.Use the RE representation of languages.
- The principal closure properties of regular languages are:

  1.The union of two regular languages is regular.

  If L and M are regular languages, then so is L □ M.

  2. The intersection of two regular languages is regular.

  If L and M are regular languages, then so is L ∩ M.

  3. The compliment of two regular languages is regular.

  If L is a regular language over alphabet Σ , then Σ *-L is also regular language.

  4. The difference of two regular languages is regular.

  If L and M are regular languages, then so is L - M.

  5. The reversal of a regular language is regular.

  The reversal of a string means that the string is written backward, i.e. reversal of abcde is edcba.

  The reversal of a language is the language consisting of reversal of all its strings, i.e. if L={001,110} then

  L□ = {100,011}.

  6.The closure of a regular language is regular.

  If L is a regular language, then so is L*.

7. The concatenation of regular languages is regular.

If L and M are regular languages, then so is L M.

8.The homomorphism of a regular language is regular.

A homomorphism is a substitution of strings for symbol. Let the function h be defined by h(0) = a and h(1) = b then h applied to 0011 is simply aabb.

If h is a homomorphism on alphabet Σ and a string of symbols w = abcd…z then

$$h(w) = h(a)\, h(b)\, h(c)\, h(d) \ldots h(z)$$

The mathematical definition for homomorphism is

$$h: \Sigma^* \rightarrow \Gamma^* \text{ such that } \square\, x, y \square \Sigma^*$$

A homomorphism can also be applied to a language by applying it to each of strings in the language. Let L be a language over alphabet Σ , and h is a homomorphism on Σ , then

$$h(L) = \{ h(w) \mid w \text{ is in } L \}$$

The theorem can be stated as " If L is a regular language over alphabet Σ , and h is a homomorphism on Σ , then h(L) is also regular " .

9. The inverse homomorphism of two regular languages is regular.

Suppose h be a homomorphism from some alphabet Σ to strings in another alphabet Γ and L be a language over Γ then h inverse of L, $h^{-1}(L)$ is set of strings w in Σ^* such that h(w) is in L.

The theorem states that " If h is a homomorphism from alphabet Σ to alphabet T , and L is aregular language on T , then $h^{-1}(L)$ is also a regular language.

Pumping lemma for regular languages

- The pumping lemma for regular languages describes an essential property of all regular languages.
- Informally, it says that all sufficiently long words in a regular language may be pumped — that is, have a middle section of the word repeated an arbitrary number of times — to produce a new word which also lies within the same language.
- Specifically, the pumping lemma says that for any regular language L there exists a constant p such that any word w in L with length at least p can be split into three substrings, w = xyz, where the middle portion y must not be empty, such that the words

xz, xyz, xyyz, xyyyz, … constructed by repeating y an arbitrary number of times (including zero times) are still in L. This process of repetition is known as "pumping".

- Moreover, the pumping lemma guarantees that the length of xy will be at most p, imposing a limit on the ways in which w may be split.
- Finite languages trivially satisfy the pumping lemma by having p equal to the maximum string length in L plus one.

  Let L be a regular language. Then there exists an integer $p \geq 1$ depending only on L such that every string w in L of length at least p (p is called the "pumping length") can be written as w = xyz (i.e., w can be divided into three substrings), satisfying the following conditions:

  1. $|y| \geq 1$
  2. $|xy| \leq p$
  3. for all $i \geq 0$, $xy^i z \in L$

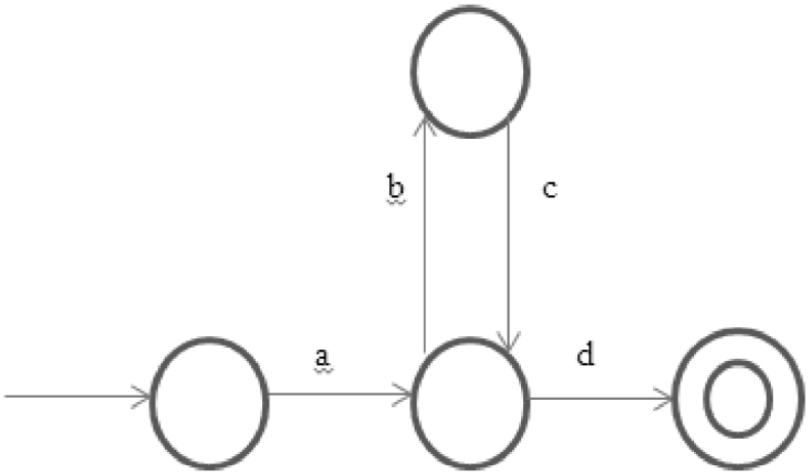
  y is the substring that can be pumped (removed or repeated any number of times, and the resulting string is always in L). (1) means the loop y to be pumped must be of length at least one; (2) means the loop must occur within the first p characters. There is no restriction on x and z.

  In simple words, For any regular language L, any sufficiently long word w(in L) can be split into 3 parts. i.e w = xyz , such that all the strings $xy^k z$ for $k \geq 0$ are also in L.

Proof of the pumping lemma

For every regular language there is a finite state automaton (FSA) that accepts the language. The number of states in such an FSA are counted and that count is used as the pumping length p. For a string of length at least p, let s0 be the start state and let s1, ...,spbe the sequence of the next p states visited as the string is emitted. Because the FSA has only p states, within this sequence of p + 1 visited states there must be at least one state that is repeated. Write S for such a state. The transitions that take the machine from the first encounter of state S to the second encounter of state S match some string. This string is called y in the lemma, and since the machine will match a string without the y portion, or the string y can be repeated any number of times, the conditions of the lemma are satisfied.

For example, the following image shows an FSA.



The FSA accepts the string: abcd. Since this string has a length which is at least as large as the number of states, which is four, the pigeonhole principle indicates that there must be at least one repeated state among the start state and the next four visited states. In this example, only q1 is a repeated state. Since the substring bc takes the machine through transitions that start at state q1 and end at state q1, that portion could be repeated and the FSA would still accept, giving the string abcbcd. Alternatively, the bc portion could be removed and the FSA would still accept giving the string ad. In terms of the pumping lemma, the string abcd is broken into an x portion a, a y portion bc and a z portion d.

DFA minimization

- DFA minimization is the task of transforming a given deterministic finite automaton (DFA) into an equivalent DFA that has minimum number of states. Here, two DFAs are called equivalent if they describe the same regular language.
- For each regular language that can be accepted by a DFA, there exists a DFA with a minimum number of states (and thus a minimum programming effort to create and use) and this DFA is unique (except that states can be given different names.)

- There are three classes of states that can be removed/merged from the original DFA without affecting the language it accepts.
- Unreachable states are those states that are not reachable from the initial state of the DFA, for any input string.
- Dead states are those nonaccepting states whose transitions for every input character terminate on themselves. These are also called Trap states because once entered there is no escape.
- Nondistinguishable states are those that cannot be distinguished from one another for any input string.
- DFA minimization is usually done in three steps, corresponding to the removal/merger of the relevant states. Since the elimination of nondistinguishable states is computationally the most expensive one, it's usually done as the last step.

Grammar

It is often convenient to specify languages in terms of grammars. The advantage in doing so arises mainly from the usage of a small number of rules for describing a language with a large number of sentences. For instance, the possibility that an English sentence consists of a subject phrase followed by a predicate phrase can be expressed by a grammatical rule of the form <sentence>→<subject><predicate>. (The names in angular brackets are assumed to belong to the grammar metalanguage.) Similarly, the possibility that the subject phrase consists of a noun phrase can be expressed by a grammatical rule of the form <subject>→<noun>.

G is defined as a mathematical system consisting of a quadruple <N, Σ, P, S>, where

N : is an alphabet, whose elements are called nonterminalsymbols.
Σ : is an alphabet disjoint from N, whose elements are called terminalsymbols.
P :is a relation of finite cardinality on (N ∪Σ)*, whose elements are called productionrules. Moreover, each production rule (α, β) in P, denoted α→β, must have at least one nonterminal

symbol in α. In each such production rule, $\alpha$is said to be the left-handside of the production rule, and $\beta$is said to be the right-handside of the production rule.

S is a symbol in N called the start, or sentence, symbol.

Types of grammars

Prescriptive: prescribes authoritative norms for a language

Descriptive: attempts to describe actual usage rather thanenforce arbitrary rules

Formal: a precisely defined grammar, such as context-free

Generative: a formal grammar that can generate naturallanguage expressions

Chomsky hierarchy of languages.

The Chomsky hierarchy consists of the following levels:

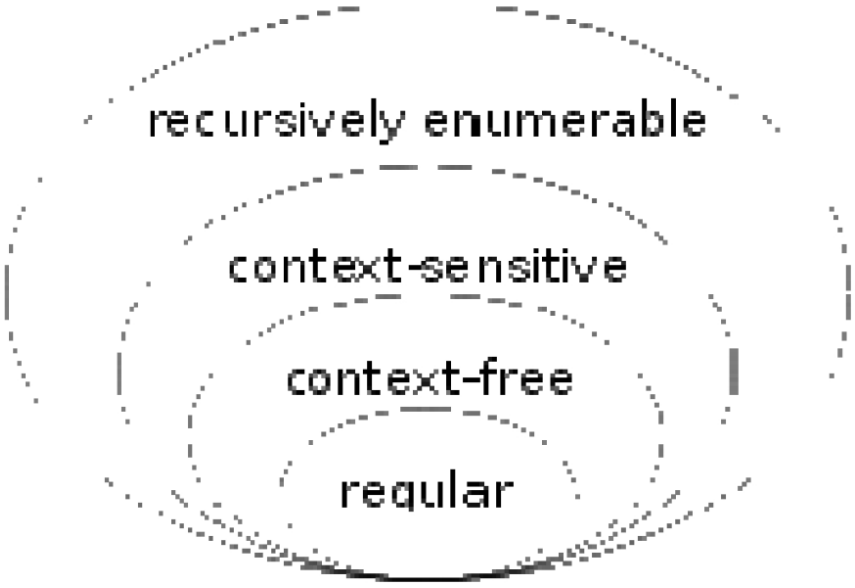
Type-0 grammars (unrestricted grammars) include all formal grammars. They generate exactly all languages that can be recognized by a Turing machine. These languages are also known as the recursively enumerable languages. Note that this is different from the recursive languages which can be decided by an always-halting Turing machine.

Type-1 grammars (context-sensitive grammars) generate the context-sensitive languages. These grammars have rules of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$with Aa nonterminal and α, $\beta$and γstrings of terminals and nonterminals. The strings $\alpha$and βmay be empty, but γmust be nonempty. The rule $S \rightarrow \epsilon$is allowed if Sdoes not appear on the right side of any rule. The languages described by these grammars are exactly all languages that can be recognized by a linear bounded automaton (a nondeterministic Turing machine whose tape is bounded by a constant times the length of the input.)

Type-2 grammars (context-free grammars) generate the context-free languages. These are defined by rules of the form $A \rightarrow \gamma$with Aa nonterminal and γa string of terminals and nonterminals. These languages are exactly all languages that can be recognized by a non-

deterministic pushdown automaton. Context-free languages are the theoretical basis for the syntax of most programming languages.

Type-3 grammars (regular grammars)generate the regular languages. Such a grammar restricts its rules to a single nonterminal on the left-hand side and a right-hand side consisting of a single terminal, possibly followed (or preceded, but not both in the same grammar) by a single nonterminal. The rule $S \rightarrow \epsilon$is also allowed here if Sdoes not appear on the right side of any rule. These languages are exactly all languages that can be decided by a finite state automaton. Additionally, this family of formal languages can be obtained by regular expressions. Regular languages are commonly used to define search patterns and the lexical structure of programming languages.



Examples:

1. The language consists of all strings begin with 0.

{0}{0, 1}*

2. The language consists of all strings begin with 0, and end with 1.

{0}{0, 1}*{1}

3. The language consists of all strings with odd lengths.

{0, 1}2n−1, n = 1, 2, . . .

4. The language consists of all strings with substring of three consecutive 0.

{0, 1}*000{0, 1}*

5. The language consists of all strings without substring of three consecutive 0.

{001, 01, 1}*

Regular grammar

A regular grammar is any right-linear or left-linear grammar.

A right regular grammar (also called right linear grammar) is a formal grammar (N, Σ, P, S) such that all the production rules in P are of one of the following forms:

B → a - where B is a non-terminal in N and a is a terminal in Σ

B → aC - where B and C are in N and a is in Σ

B → ε - where B is in N and ε denotes the empty string, i.e. the string of length 0.

In a left regular grammar (also called left linear grammar), all rules obey the forms

A → a - where A is a non-terminal in N and a is a terminal in Σ

A → Ba - where A and B are in N and a is in Σ

A → ε - where A is in N and ε is the empty string.

An example of a right regular grammar G with N = {S, A}, Σ = {a, b, c}, P consists of the following rules

S → aS

S → bA

A → ε

A → cA

andS is the start symbol. This grammar describes the same language as the regular expression a*bc*.

Extended regular grammars

An extended right regular grammar is one in which all rules obey one of

1. B → a - where B is a non-terminal in N and a is a terminal in Σ
2. A → wB - where A and B are in N and w is in Σ*
3. A → ε - where A is in N and ε is the empty string.

Some authors call this type of grammar a right regular grammar (or right linear grammar) and the type above a strictly right regular grammar (or strictly right linear grammar).

An extended left regular grammar is one in which all rules obey one of

1. A → a - where A is a non-terminal in N and a is a terminal in Σ
2. A → Bw - where A and B are in N and w is in Σ*
3. A → ε - where A is in N and ε is the empty string.

Regular expression

A regular expression (or regexp, or pattern, or RE) is a text string that describes some (mathematical) set of strings. A RE r matches a string s if s is in the set of strings described by r. Regular Expressions have their own notation. Characters are single letters for example 'a', ' '(single blank space), '1' and '-' (hyphen). Operators are entries in a RE that match one or more characters.

Regular expressions consist of constants and operator symbols that denote sets of strings and operations over these sets, respectively. The following definition is standard, and found as such in most textbooks on formal language theory. Given a finite alphabet Σ, the following constants are defined as regular expressions:

- (**empty set)**∅ denoting the set ∅.
- (**empty string)** ε denoting the set containing only the "empty" string, which has no characters at all.
- (**literal character)** a in Σ denoting the set containing only the character a.

Given regular expressions R and S, the following operations over them are defined to produce regular expressions:

- (**concatenation**) RS denoting the set { αβ | α in set described by expression R and β in set described by S }. For example {"ab", "c"}{"d", "ef"} = {"abd", "abef", "cd", "cef"}.
- (**alternation**) R | S denoting the set union of sets described by R and S. For example, if R describes {"ab", "c"} and S describes {"ab", "d", "ef"}, expression R | S describes {"ab", "c", "d", "ef"}.
- (**Kleene star**) R* denoting the smallest superset of set described by R that contains ε and is closed under string concatenation. This is the set of all strings that can be made by concatenating any finite number (including zero) of strings from set described by R. For

example, {"0","1"}* is the set of all finite binary strings (including the empty string), and {"ab", "c"}* = {ε, "ab", "c", "abab", "abc", "cab", "cc", "abababab", "abcab", ... }.

To avoid parentheses it is assumed that the Kleene star has the highest priority, then concatenation and then alternation. If there is no ambiguity then parentheses may be omitted. For example, (ab)c can be written as abc, and a|(b(c*)) can be written as a|bc*.

Examples:

- a|b* denotes {ε, "a", "b", "bb", "bbb", ...}
- (a|b)* denotes the set of all strings with no symbols other than "a" and "b", including the empty string: {ε, "a", "b", "aa", "ab", "ba", "bb", "aaa", ...}
- ab*(c|ε) denotes the set of strings starting with "a", then zero or more "b"s and finally optionally a "c": {"a", "ac", "ab", "abc", "abb", "abbc", ...}

Left and right linear grammars.

A linear language is a language generated by some linear grammar.

Example

A simple linear grammar is G with N = {S}, Σ = {a, b}, P with start symbol S and rules

$S \rightarrow aSb$

$S \rightarrow \varepsilon$

Two special types of linear grammars are the following:

- the left-linear or left regular grammars, in which all nonterminals in right hand sides are at the left ends;
- the right-linear or right regular grammars, in which all nonterminals in right hand sides are at the right ends.

Collectively, these two special types of linear grammars are known as the regular grammars; both can describe exactly the regular languages.

Another special type of linear grammar is the following:

- linear grammars in which all nonterminals in right hand sides are at the left or right ends, but not necessarily all at the same end.

By inserting new nonterminals, every linear grammar can be brought into this form without affecting the language generated. For instance, the rules of G above can be replaced with

$S \rightarrow aA$

$A \rightarrow Sb$

$S \rightarrow \varepsilon$

Hence, linear grammars of this special form can generate all linear languages.

Left linear Grammar

A -> B a or A -> a,where A and B are in N and a is in S

Right linearGrammar

A -> a B or A -> a,

where A and B are in N and a is in S

Example:

S -> a T
S -> b T
S -> a
S -> b
T -> a T
T -> b T

T -> 1
T
T -> 2
T
T -> a
T -> b
T -> 1
T -> 2

S => a
S => a T => a 1
S => a T => a 1 T

Constructing a Nondeterministic Finite State Automaton from a Right Linear Grammar

Let G = (N, S, P, S).

Construct a nondeterministic finite state automaton

M = (Q, S, d, S, F), where

Q = N U {X}, X not in N or S

F = {X}

d is constructed by

If A -> a B is in P, then B is in d(A,a)

If A -> a is in P, then X is in d (A,a)

S -> a T
S -> b T
S -> a
S -> b
T -> a T
T -> b T

T -> 1 T
T -> 2 T
T -> a
T -> b
T -> 1
T -> 2

d(S,a) = {T, X}
d(S,b) = {T,X}
d(S,1) = F
d(S,2) = F
d(T,a) = {T,X}
d(T,b) = {T,X}
d(T,1) = {T,X}
d(T,2) = {T,X}

A Left Linear Grammar for Identifiers

Example:

S -> S a
S -> S b
S -> S 1
S -> S 2
S -> a
S -> b

S => a
S => S 1 => a 1
S => S 2 => S b 2
=> S 1 b 2 => a 1 b 2

Constructing a Nondeterministic Finite State Automaton from a Left Linear Grammar

Let G = (N, S, P, S).
Construct a nondeterministic finite state automaton M = (Q, S, d, X, F), where
Q = N U {X}, X not in N or S
F = {S}
d is constructed by
If A -> B a is in P, then A is in d(B,a)
If A -> a is in P, then A is in d (X,a)

S -> S a
S -> S b
S -> S 1
S -> S 2
S -> a
S -> b

d(X,a) = {S}
d(X,b) = {S}
d(X,1) = F
d(X,2) = F
d(S,a) = {S}
d(S,b) = {S}
d(S,1) = {S}
d(S,2) = {S}

Context free grammars

A context-free grammar G is defined by the 4-tuple:

G=(V,T,P,S) where

1. V is a finite set; each element v □V is called a non-terminal character or a variable. Each variable represents a different type of phrase or clause in the sentence. Variables are also sometimes called syntactic categories. Each variable defines a sub-language of the language defined by G.
2. T is a finite set of terminals, disjoint from V, which make up the actual content of the sentence. The set of terminals is the alphabet of the language defined by the grammar G.
3. P is a set of production rule.
4. S is the start variable (or start symbol), used to represent the whole sentence (or program). It must be an element of V.

Example:

1. S → x
2. S → y
3. S → z
4. S → S + S
5. S → S - S
6. S → S * S
7. S → S / S
8. S → (S)

This grammar can, for example, generate the string

(x + y) * x - z * y / (x + x)

as follows:

S (the start symbol)

→ S - S (by rule 5)

→ S * S - S (by rule 6, applied to the leftmost S)

→ S * S - S / S (by rule 7, applied to the rightmost S)

→ (S) * S - S / S (by rule 8, applied to the leftmost S)

→ (S) * S - S / (S) (by rule 8, applied to the rightmost S)

→ (S + S) * S - S / (S) (etc.)

→ (S + S) * S - S * S / (S)

→ (S + S) * S - S * S / (S + S)

→ (x + S) * S - S * S / (S + S)

→ (x + y) * S - S * S / (S + S)

→ (x + y) * x - S * y / (S + S)

→ (x + y) * x - S * y / (x + S)

→ (x + y) * x - z * y / (x + S)

→ (x + y) * x - z * y / (x + x)

Problem 1. Give a context-free grammar for the language

$L = \{a^n b^m : n \neq 2m\}$.

Is your grammar ambiguous?

Ans:

A grammar for the language is

S →aaSb | A | B | ab

A →aA | a

B →bB | b

This grammar is unambiguous; it is not hard to prove that every string in the langauge has one

and only one parse tree.

Problem 2. Give a context-free grammar for the language

L = {x □{0, 1}_ : x has the same number of 0's and 1's}

Is your grammar ambiguous?

Ans:

A grammar for the language is

S →S0S1S | S1S0S | ε

This grammar is ambiguous; for example, 0101 has two different parse trees

Problem 3. Prove that L = {aibjck : j = max{i, k}} is not context free.

Ans:

Suppose for contradiction that L were context free. Let N be the "N" of the pumping lemma

for context-free languages. Consider the string $w = a^N b^N c^N$. Suppose w = uvxyz, where $|vxy| \leq N$ and $|vy| \geq 1$. If vy contains only a's or vy contains only c's, then pump up: the string uv^2xy^2z □ L. Suppose vy contains only b's. Then we can pump either way to get a string not in L. Suppose v contains two different letters or y contains two different letters. Then uv^2xy^2z is not even of the form a*b*c*, so certainly it is not in L. Finally, suppose (v □a+ and) y □b+, or v □ b+ (and y □ c+). Then we can pump down and there will be too few b's. By $|vwy| \leq N$, these are all the possible cases. So in all cases there is some i for which uv^ixy^iz□ L, a contradiction.

Theorem : L □A* is CF iff□NPDA M that accepts L.

Proof ->: Given CFL L, consider any grammar G(L) for L. Construct NPDA M that simulates all possible derivations of G. M is essentially a single-state FSM, with a state q that applies one of G's rules at a time. The start state q0 initializes the stack with the content S ¢, where S is the start symbol of G, and ¢ is the bottom of stack symbol. This initial stack content means that M aims to read an input that is an instance of S. In general, the current stack content is a sequence of symbols that represent tasks to be accomplished in the characteristic LIFO order (last-in first-out). The task on top of the stack, say a non-terminal X, calls for the next characters of the input string to be an instance of X. When these characters have been read and verified to be an instance of X, X is popped from the stack, and the new task on top of the stack is started. When ¢ is on top of the stack, i.e. thestack is empty, all tasks generated by the first instance of S have been successfully met, i.e. the input string read so far is an instance of S. M moves to the accept state and stops.

The following transitions lead from q to q:

1) □, X-> w for each rule X -> w. When X is on top of the stack, replace X by a right-hand side for X.

2) a, a -> □ for each a □A. When terminal a is read as input and a is also on top of the stack, pop the stack.

Rule 1 reflects the following fact: one way to meet the task of finding an instance of X as a prefix of the inputstring not yet read, is to solve all the tasks, in the correct order, present in the right-hand side w of the productionX -> w. M can be considered to be a non-deterministic parser for G. A formal proof that M accepts precisely Lcan be done by induction on the length of the derivation of any w □L.

Ambiguous Grammar;

A grammar is said to be ambiguous if more than two parse trees can be constructed from it.

Example 1:

The context free grammar

$$A \to A + A \mid A - A \mid a$$

is ambiguous since there are two leftmost derivations for the string a + a + a:

$A \to A + A$ $A \to A + A$

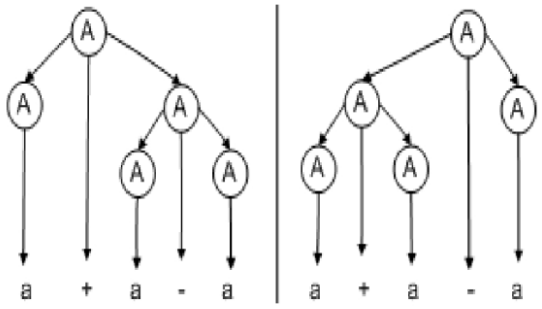
$\to a + A$ $\to A + A + A$ (First A is replaced by A+A. Replacement of the second A would yield a similar derivation)

$\to a + A + A$ $\to a + A + A$

$\to a + a + A$ $\to a + a + A$

$\to a + a + a$ $\to a + a + a$

As another example, the grammar is ambiguous since there are two parse trees for the string a + a − a:



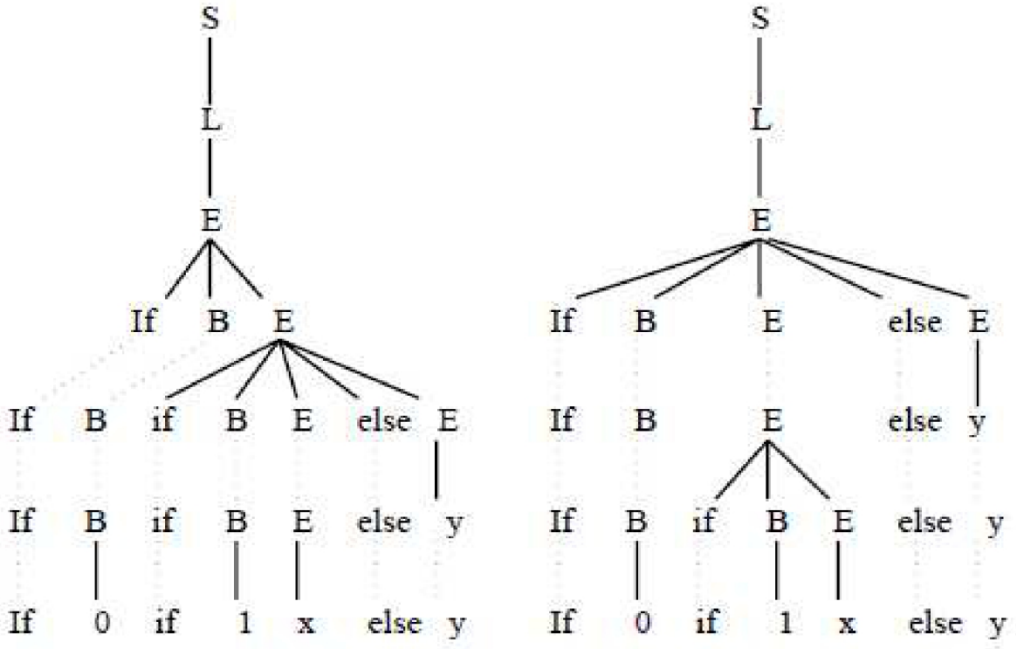
The language that it generates, however, is not inherently ambiguous; the following is a non-ambiguous grammar generating the same language:

$$A \to A + a \mid A - a \mid a$$

Example 2 : Show that the following grammar is ambigious

$$
\begin{aligned}
S &\longrightarrow L \\
L &\longrightarrow E \\
L &\longrightarrow LE \\
E &\longrightarrow \text{if B E else E} \\
E &\longrightarrow \text{if B E} \\
E &\longrightarrow x \\
E &\longrightarrow y \\
B &\longrightarrow 0 \\
B &\longrightarrow 1
\end{aligned}
$$

Answer :



Unit 4: Recurrence Relation

3.1 Introduction

Recurrence Relation : A recurrence relation for the sequence $\{an\}$ is an equation that expresses an in terms of one or more of the previous terms of the sequence, namely, $a0, a1, \ldots, an\text{-}1$, for all integers n with $n \geq n0$, where $n0$ is a nonnegative integer.

A sequence is call solution of a recurrence relation if its term satisfy the recurrence relation.

Recursive definition

$$a_n = 2a_{n-1} \longrightarrow \text{Recurrence relation}$$

$$a_0 = 5 \longrightarrow \text{Initial condition}$$

- A recursive algorithm provides the solution of a problem of size n in term of the solutions of one or more instances of the same problem in smaller size.
- The initial condition specify the terms that precede the first term where the recurrence elation takes effect.

A recurrence relation for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more of the previous terms of the sequence, namely, $a_0, a_1, \ldots, a_{n-1}$, for all integers n with $n \geq n_0$, where n_0 is a nonnegative integer.

A sequence is call solution of a recurrence relation if its term satisfy the recurrence relation.

Recursive definition

$$a_n = 2a_{n-1} \longrightarrow \text{Recurrence relation}$$

$$a_0 = 5 \longrightarrow \text{Initial condition}$$

- A recursive algorithm provides the solution of a problem of size n in term of the solutions of one or more instances of the same problem in smaller size.
- The initial condition specify the terms that precede the first term where the recurrence elation takes effect.

Example 1: Suppose that f is defined recursively by f (0) = 3, f (n + 1) = 2f (n) + 3. Find f (1), f (2), f (3), and f (4).

Solution: From the recursive definition it follows that

$f(1) = 2f(0) + 3 = 2 \cdot 3 + 3 = 9,$

$f(2) = 2f(1) + 3 = 2 \cdot 9 + 3 = 21,$

$f(3) = 2f(2) + 3 = 2 \cdot 21 + 3 = 45,$

$f(4) = 2f(3) + 3 = 2 \cdot 45 + 3 = 93.$

Example 2: The Fibonacci number can be defined by recursive definition.

Solution: $\mathbf{F_n = F_{n-1} + F_{n-2}}$, Where F(0)=0, F(1)=1

Example 3: Give a recursive definition of an, where a is a nonzero real number and n is a nonnegative integer.

Solution: The recursive definition contains two parts. First a^0 is specified, $a^0 = 1$. Then we can define a^{n+1} from a^n

$\mathbf{a^{n+1} = a \cdot a^n}$, for n = 0, 1, 2, 3, . . .

3.2 Tower of Hanoi

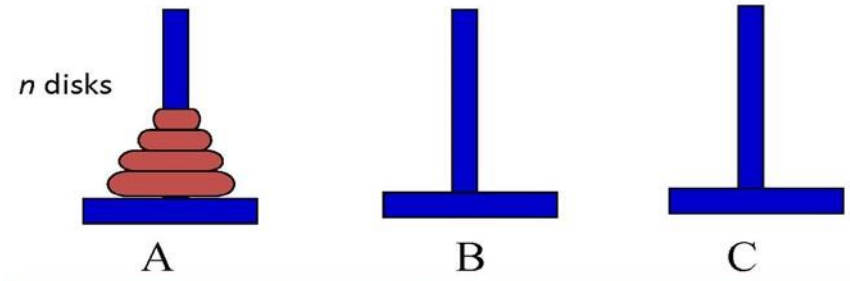
Tower of Hanoi

- Popular Puzzle invented by Edouard Lucas (French Mathematician, late 19th century)

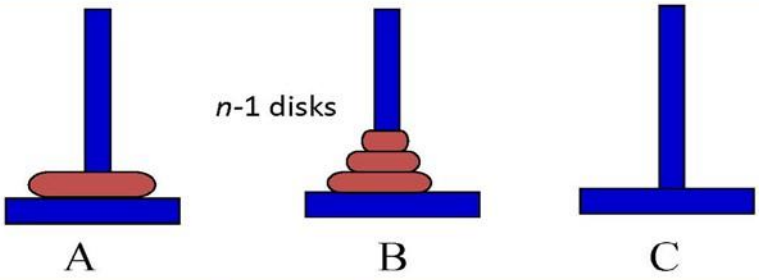
RULES OF PUZZLE:

- Suppose we have 3 pegs labeled A, B, C and a set of disks of different sizes.
- These disks are arranged from the largest disk at the bottom to the smallest disk at the top (1 to *n* disks) on peg A.
- The goal: to have all disks on peg C in order of size, with the largest on the bottom.
- Only one disk is allow to move at a time from a peg to another.
- Each peg must always be arranged from the largest at the bottom to the smallest at the top

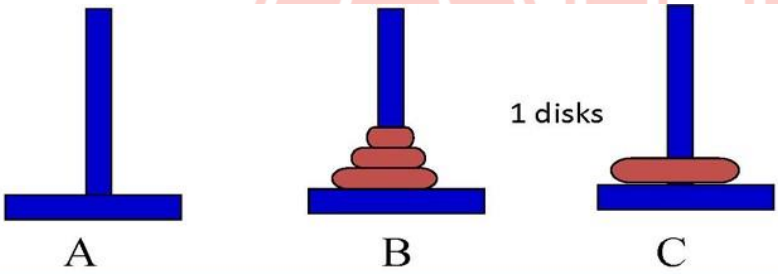
ILLUSTRATION:



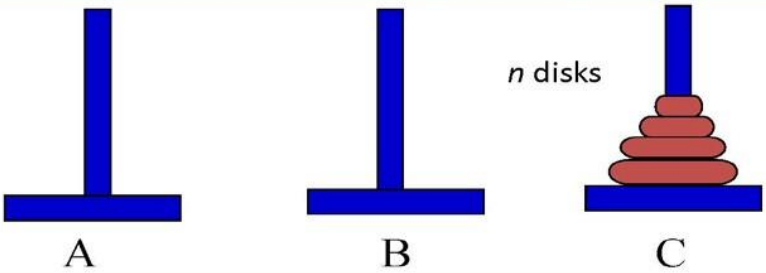
Initial Position in the Tower of Hanoi



Intermediate Position in the Tower of Hanoi



Intermediate Position in the Tower of Hanoi



Last Position in the Tower of Hanoi

SOLUTION

Let H_n :
the numbers of
moves with n disks

transfer the top of
n-1 disks to peg B

H_{n-1} moves

Moves the largest
disk to peg C

1 move

transfer the top of
n-1 disks to peg C

H_{n-1} moves

The number of moves is given by: $H_n = 2H_{n-1} + 1 \quad ; \quad H_1 = 1$

Where,

transfer the top of n-1 disks to peg B	Moves the largest disk to peg C	transfer the top of n-1 disks to peg C
H_{n-1} moves	+ 1 move	+ H_{n-1} moves

Now solving it, we get

$$\begin{aligned} H_n &= 2H_{n-1} + 1 \\ &= 2(2H_{n-2} + 1) + 1 = 2^2 H_{n-2} + 2 + 1 \\ &= 2^2(2H_{n-3} + 1) + 2 + 1 = 2^3 H_{n-3} + 2^2 + 2 + 1 \\ &\vdots \\ &= 2^{n-1} H_1 + 2^{n-2} + 2^{n-3} + \ldots + 2 + 1 \\ &= 2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2 + 1 \\ &= 2^n - 1 \end{aligned}$$

3.3 Solving Recurrence Relations

Definition: A linear homogeneous recurrence relation of degree k with constant coefficients is a recurrence relation of the form:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k},$$

Where $c_1, c_2, \ldots, c_k$ are real numbers, and $c_k \neq 0$.

- A sequence satisfying such a recurrence relation is uniquely determined by the recurrence relation and the k initial conditions $a_0 = C_0$, $a_1 = C_1$, $a_2 = C_2, \ldots, a_{k-1} = C_{k-1}$.
- **Linear :** The RHS is the sum of previous terms of the sequence each multiplied by a function of n.
- **Homogeneous :** No terms occur that are not multiplies of the previous term.
- **Degree *k* :** *an* is expressed in terms of the previous *k* terms of the sequence
- **Constant coefficients :** $c1, c2, \ldots, ck$
- **Recurrence relation :** with *k* initial condition

$$a0 = C0, \quad a1 = C1, \quad \ldots \quad ak\text{-}1 = Ck\text{-}1$$

- **Examples:**
  - The recurrence relation $\mathbf{P_n = (1.05)P_{n-1}}$ is a linear homogeneous recurrence relation of **degree one**.
  - The recurrence relation $\mathbf{f_n = f_{n-1} + f_{n-2}}$ is a linear homogeneous recurrence relation of **degree two**.
  - The recurrence relation $\mathbf{a_n = a_{n-5}}$ is a linear homogeneous recurrence relation of **degree five**.
- The solutions of the form $\mathbf{a_n = r^n}$, where r is a constant. $a_n = r^n$ is a solution of the recurrence relation

  $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$ if and only if

  $$\mathbf{r^n = c_1r^{n-1} + c_2r^{n-2} + \ldots + c_kr^{n-k}.}$$

  Divide this equation by r^{n-k} and subtract the right-hand side from the left: $r^k - c_1r^{k-1} - c_2r^{k-2} - \ldots - c_{k-1}r - c_k = 0$. This is called the **characteristic equation** of the recurrence relation.
- The solutions of this equation are called the **characteristic roots** of the recurrence relation.

3.4 Solving Linear Homogenous Recurrence Relations

- **Theorem 1:** Let c_1 and c_2 are real numbers. Suppose $r^2-c_1r-c_2=0$ has two distinct roots r_1 and r_2. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n=c_1a_{n-1}+c_2a_{n-2}$ iff $a_n=\alpha_1r_1^n+\alpha_2r_2^n$ for n=0, 1, 2, …, where α_1 and α_2 are constants.
- **Theorem 2:** Let c_1 and c_2 are real numbers with $c_2 \neq 0$. Suppose $r^2-c_1r-c_2=0$ has only one root r_0. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n=c_1a_{n-1}+c_2a_{n-2}$ iff $a_n=\alpha_1r_0^n+\alpha_2nr_0^n$ for n=0, 1, 2, …, where α_1 and α_2 are constants.

Example 1: What is the solution of the recurrence relation $a_n = a_{n-1} + 2a_{n-2}$ with $a_0 = 2$ and $a_1 = 7$?

Solution:

The characteristic equation of the recurrence relation is $r^2 - r - 2 = 0$. Its roots are $r = 2$ and $r = -1$.

Hence, the sequence $\{a_n\}$ is a solution to the recurrence relation if and only if: $a_n = \alpha_1 2^n + \alpha_2 (-1)^n$ for some constants α_1 and α_2.

Given the equation $a_n = \alpha_1 2^n + \alpha_2 (-1)^n$ and the initial conditions $a_0 = 2$ and $a_1 = 7$. Now we get,

$$a_0 = 2 = \alpha_1 + \alpha_2$$

$$a_1 = 7 = \alpha_1 \cdot 2 + \alpha_2 \cdot (-1)$$

Solving these two equations , we get $\alpha_1 = 3$ and $\alpha_2 = -1$.

Therefore, the solution to the recurrence relation and initial conditions is the sequence $\{a_n\}$ with

$$a_n = 3 \cdot 2^n - (-1)^n.$$

Example 2: Give an explicit formula for the Fibonacci numbers.

Solution: The Fibonacci numbers satisfy the recurrence relation $f_n = f_{n-1} + f_{n-2}$ with initial conditions $f_0 = 0$ and $f_1 = 1$. The characteristic equation is $r^2 - r - 1 = 0$. Its roots are $r_1 = \frac{1+\sqrt{5}}{2}$, $r_2 = \frac{1-\sqrt{5}}{2}$

Therefore, the Fibonacci numbers are given by $\boxed{f_n = \alpha_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + \alpha_2 \left(\frac{1-\sqrt{5}}{2}\right)^n}$

for some constants α_1 and α_2. We can determine values for these constants so that the sequence meets the conditions $f_0 = 0$ and $f_1 = 1$:

$$f_0 = \alpha_1 + \alpha_2 = 0 \qquad f_1 = \alpha_1 \left(\frac{1+\sqrt{5}}{2}\right) + \alpha_2 \left(\frac{1-\sqrt{5}}{2}\right) = 1$$

The unique solution to this system of two equations and two variables is

$$\alpha_1 = \frac{1}{\sqrt{5}}, \quad \alpha_2 = -\frac{1}{\sqrt{5}}$$

So finally we obtained an explicit formula for the Fibonacci numbers:

$$f_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^n$$

Example 3: What is the solution of the recurrence relation $a_n = 6a_{n-1} - 9a_{n-2}$ with $a_0 = 1$ and $a_1 = 6$?

Solution: The only root of $r^2 - 6r + 9 = 0$ is $r_0 = 3$. Hence, the solution to the recurrence relation is $a_n = \alpha_1 3^n + \alpha_2 n3^n$ for some constants α_1 and α_2.

To match the initial condition, we need

$$a_0 = 1 = \alpha_1, \quad a_1 = 6 = \alpha_1 \cdot 3 + \alpha_2 \cdot 3$$

Solving these equations yields $\alpha_1 = 1$ and $\alpha_2 = 1$.

Consequently, the overall solution is given by $a_n = 3^n + n3^n$.

- Example 4: Find a solution to $a_n = 5a_{n-1} - 6a_{n-2}$ with initial conditions $a_0=1, a_1=4$.

Solution: The characteristic equation is $r^2 - 5r + 6 = 0$

The roots are $r^2 - 5r + 6 = (r-2)(r-3)$ ⟹ $r_1=2, r_2=3$

Using the 2nd order theorem we have a solution $a_n = \alpha_1 2^n + \alpha_2 3^n$

Now put roots in the two initial conditions to get a system of linear equations $a_0 = \alpha_1 2^0 + \alpha_2 3^0$ $a_1 = \alpha_1 2^1 + \alpha_2 3^1$

Thus: $1 = \alpha_1 + \alpha_2$ $4 = 2\alpha_1 + 3\alpha_2$

Solving for $\alpha_1 = (1 - \alpha_2)$, we get $\alpha_1 = -1$ and $\alpha_2 = 2$

The solution becomes: $a_n = -1 \cdot 2^n + 2 \cdot 3^n$

Example 5: Solve the recurrence relation $an = 5an\text{-}1 - 6an\text{-}2$ where $a0 = 4$ and $a1 = 7$.

Solution: The characteristic equation is given by $r^2 - 5r + 6 = 0$

The characteristic roots are 2 and 3. Thus, the general solution is

$$a_n = A \cdot 2^n + B \cdot 3^n$$

To find the constant values, A and B using the initial conditions.

Solving the linear system $a_0 = A + B = 4$

$a_1 = 2A + 3B = 7$ **A = 5, B = -1**

Thus the solution to the recurrence relation and initial conditions is

$$a_n = 5 \cdot 2^n - 3^n, n \geq 0$$

- **Theorem 3:** Let $c_1, c_2, \ldots, c_k$ be real numbers. Suppose $r^k - c_1 r^{k-1} - \ldots - c_k = 0$ has k distinct roots $r_1, r_2, \ldots, r_k$. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$ iff $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n + \ldots + \alpha_k r_k^n$ for n=0, 1, 2, …, where $\alpha_1, \alpha_2 \ldots, \alpha_k$ are constants.
- **Theorem 4:** Let $c_1, c_2, \ldots, c_k$ be real numbers. Suppose $r^k - c_1 r^{k-1} - \ldots - c_k = 0$ has t distinct roots $r_1, r_2, \ldots, r_t$ with multiplicities $m_1, m_2, \ldots, m_t$ such that $m_i >= 1$ and $m_1 + m_2 + \ldots + m_t = k$. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$ iff $a_n = (\alpha_{1,0} + \alpha_{1,1} n + \ldots + \alpha_{1,m1-1}) r_1^n + (\alpha_{2,0} + \alpha_{2,1} n + \ldots + \alpha_{2,m2-1}) r_2^n + \ldots + (\alpha_{t,0} + \alpha_{t,1} n + \ldots + \alpha_{t,mt-1}) r_t^n$ for n=0, 1, 2, …, where $\alpha_{i,i}$ are constants.

Example 6: Solve the recurrence relation $an = 6an\text{-}1 - 11an\text{-}2 + 6an\text{-}3$ where $a0 = 2$, $a1 = 5$, and $a2 = 15$.

Solution: The characteristic equation is given by $r^3 - 6r^2 + 11r - 6 = 0$

The characteristic roots are 1, 2 and 3. Thus, the general solution is

$$a_n = A \cdot 1^n + B \cdot 2^n + C \cdot 3^n$$

To find the constant values, *A, B and C* using the initial conditions.

Solving the linear system:

$$a_0 = A + B + C = 2, \quad a_1 = A + 2B + 3C = 5, \quad a_2 = A + 4B + 9C = 15$$

$$(\boldsymbol{A = 1, B = -1, C = 2})$$

Thus the unique solution to the recurrence relation and initial conditions is $a_n = 1 - 2^n + 2 \cdot 3^n, n \geq 0$

Example 7: Suppose that the roots of the characteristic equation of a linear homogenous recurrence relation are 2, 2, 2, 5, 5, and 9. What is the form of general solution?

Solution: Thus the general solution to the recurrence relation is

$$a_n = A2^n + Bn2^n + Cn^2 2^n + D5^n + En5^n + F \cdot 9^n$$
$$= (A + Bn + Cn^2)2^n + (D + En)5^n + F \cdot 9^n$$

3.5 Solving Linear Nonhomogeneous Recurrence Relations

- **Theorem 5:** If $\{a_n^{(p)}\}$ is a particular solution of the nonhomogeneous recurrence relation with constant coefficients $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k} + F(n)$, then every solution is of the form $\{a_n^{(p)} + a_n^{(h)}\}$, where $a_n^{(h)}$ is a solution of the associated homogeneous recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$.
- **Theorem 6:** Suppose $\{a_n\}$ satisfies the nonhomogeneous recurrence relation with constant coefficients

  $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k} + F(n)$,

  and $F(n) = (b_t n^t + b_{t-1} n^{t-1} + \ldots + b_1 n + b_0)s^n$.

  When s is not a root of the associate linear homogeneous recurrence recurrence relation, there is a particular solution of the form $(p_t n^t + p_{t-1} n^{t-1} + \ldots + p_1 n + p_0)s^n$. When s is a root of this characteristic equation and its multiplicity is m, there is a particular solution of the form $n^m (p_t n^t + p_{t-1} n^{t-1} + \ldots + p_1 n + p_0)s^n$.

- **Example 8:** Find the solution of the recurrence relation $a_n = 3a_{n-1} + 2^n$ with the initial condition $a_0 = 2$.

Solution: The solution of linear homogeneous recurrence relation which is, $a_n = 3a_{n-1}$. The characteristic equation gives us $r = 3$, and therefore $\mathbf{a_n = c_1(3^n)}$. Now to solve the non-homogeneous part. Let $\mathbf{a_n = c_2 2^n}$

From here, we then deduce that $a_{n-1} = c_2 2^{n-1}$. Putting these 2 equations back to the initial recurrence relation $a_n = 3a_{n-1} + 2^n$, we have

$c_2 2^n = 3c_2 2^{n-1} + 2^n \Rightarrow (c_2 - 1)2^n = 3c_2 2^{n-1} \Rightarrow 2(c_2 - 1) = 3c_2$
$\Rightarrow 2(c_2 - 1) = 3c_2 \Rightarrow c_2 = -2$ which then gives us $a_n = -2(2^n) = -2^{n+1}$.

Combining both part, The general solution : $\mathbf{a_n = c_1(3^n) - 2^{n+1}}$

The initial condition $\mathbf{a_0 = 2}$, we can find the value of c_1 from the general solution : $c_1 = 4$

Then our particular solution will be $\mathbf{a_n = 4(3^n) - 2^{n+1}}$

- **Example 9:** Find the solution of the recurrence relation $\mathbf{a_n = 2a_{n-1} + 2^n}$ with the initial condition $\mathbf{a_0 = 2}$.

Solution: The solution of linear homogeneous recurrence relation which is, $a_n = 2a_{n-1}$. The characteristic equation gives us $r = 2$, and therefore $\mathbf{a_n = c_2 2^n}$, which has the same form with the non-homogeneous part! In this case, we need to multiply our non-homogeneous part with **n**. So, let $\mathbf{a_n = nc_1 2^n}$ and $\mathbf{a_{n-1} = (n-1)c_1 2^{n-1}}$

Now putting these 2 equations back to the initial recurrence relation,

$$nc_1 2^n = 2(n-1)c_1 2^{n-1} + 2^n$$

By solving this relation, we get $\mathbf{c_1 = 1}$

Combining both part, The general solution : $\mathbf{a_n = c_2 2^n + n\, c_1\, 2^n}$

$$\mathbf{a_n = c_2 2^n + n2^n}$$

The initial condition $\mathbf{a_0 = 2}$, we can find the value of c_2 from the general solution : $c_2 = 2$

Then our particular solution will be $\mathbf{a_n = 2^{n+1} + n\, 2^n}$

Example 10: Find the solution of the recurrence Relation

$$\begin{cases} x_n &= 3x_{n-1} + 10x_{n-2} + 7 \cdot 5^n \\ x_0 &= 4 \\ x_1 &= 3 \end{cases}$$

Solution. The characteristic equation is

$$r^2 - 3r - 10 = 0 \Longleftrightarrow (r-5)(r+2) = 0.$$

We have roots $r_1 = 5$, $r_2 = -2$. Since $r = 5$, then $r = r$
A special solution can be of the type $x_n = An5^n$. Put the so
non-homogeneous relation. We have

$$An5^n = 3A(n-1)5^{n-1} + 10A(n-2)5^{n-2} + 7 \cdot 5^n$$

Dividing both sides by 5^{n-2},

$$An5^2 = 3A(n-1)5 + 10A(n-2) + 7 \cdot 5^2.$$

Thus

$$-35A + 7 \cdot 25 = 0 \Longrightarrow A = 5.$$

So

$$x_n = n5^{n+1}.$$

The general solution is

$$x_n = n5^{n+1} + c_1 5^n + c_2(-2)^n.$$

The initial condition implies $c_1 = -2$ and $c_2 = 6$. Therefore

$$x_n = n5^{n+1} - 2 \cdot 5^n + 6(-2)^n.$$

Example 11: Find the solution of the recurrence Relation.

$$\begin{cases} x_n &= 10x_{n-1} - 25x_{n-2} + 8 \cdot 5^n \\ x_0 &= 6 \\ x_1 &= 10 \end{cases}$$

Solution. The characteristic equation is

$$r^2 - 10r + 25 = 0 \Longleftrightarrow (r-5)^2 = 0.$$

We have roots $r_1 = r_2 = 5$, then $r = r_1 = r_2 = 5$. A special solution can be of the type $x_n = An^2 5^n$. Put the solution into the non-homogeneous relation. We have

$$An^2 5^n = 10A(n-1)^2 5^{n-1} - 25A(n-2)^2 5^{n-2} + 8 \cdot 5^n$$

Dividing both sides by 5^{n-2},

$$An^2 5^2 = 10A(n-1)^2 5 - 25A(n-2)^2 + 8 \cdot 5^2.$$

Since $An^2 5^2 = 10An^2 5 - 25n^2$, we have

$$10A(-2n+1)5 - 25A(-4n+4) + 8 \cdot 5^2 = 0 \Longrightarrow A = 4.$$

So a nonhomogeneous solution is

$$x_n = 4n^2 5^n.$$

The general solution is

$$x_n = 4n5^n + c_1 5^n + c_2 n5^n.$$

The initial condition implies $c_1 = 6$ and $c_2 = -8$. Therefore

$$x_n = (4n^2 - 8n + 6)5^n.$$

Example 12: What form does a particular solution of the linear nonhomogeneous recurrence relation $a_n = 6a_{n-1} - 9a_{n-2} + F(n)$ have when $F(n) = 3^n$, $F(n) = n3^n$, $F(n) = n^2 2^n$, and $F(n) = (n^2 + 1)3^n$?

Solution: The associated linear homogeneous recurrence relation is $a_n = 6a_{n-1} - 9a_{n-2}$. Its characteristic equation, $r2 - 6r + 9 = (r - 3)^2 = 0$, has a single root, 3, of multiplicity two.

With $F(n)$ of the form $P(n)s^n$, where $P(n)$ is a polynomial and s is a constant. Because $s = 3$ is a root with multiplicity $m = 2$ but $s = 2$ is not a root.

- ✓ A particular solution has the form $p_0 n^2 3^n$ if $F(n) = 3^n$
- ✓ the form $n^2(p_1 n + p_0)3^n$ if $F(n) = n3^n$,
- ✓ the form $(p_2 n^2 + p_1 n + p_0)2^n$ if $F(n) = n^2 2^n$,
- ✓ the form $n^2(p_2 n^2 + p_1 n + p_0)3^n$ if $F(n) = (n^2 + 1)3^n$.

Example 13: Consider the recurrence relation

$$a_n = 5a_{n-1} - 6a_{n-2} + F(n).$$

The characteristic equation of its associated homogeneous equation is

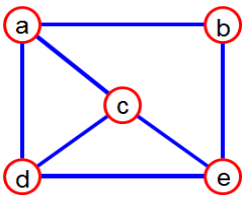
$$s^2 - 5s + 6 = (s - 2)(s - 3) = 0.$$

1. If $F(n) = 2n^2$, then a particular solution has the form $a_n^{(p)} = An^2 + Bn + C$.
2. If $F(n) = 5^n(3n^2 + 2n + 1)$, then $a_n^{(p)} = 5^n(An^2 + Bn + C)$.
3. If $F(n) = 5^n$, then $a_n^{(p)} = 5^n A$.
4. If $F(n) = 3^n$, then $a_n^{(p)} = 3^n An$.
5. If $F(n) = 2^n(3n + 1)$, then $a_n^{(p)} = 2^n n(An + B)$.

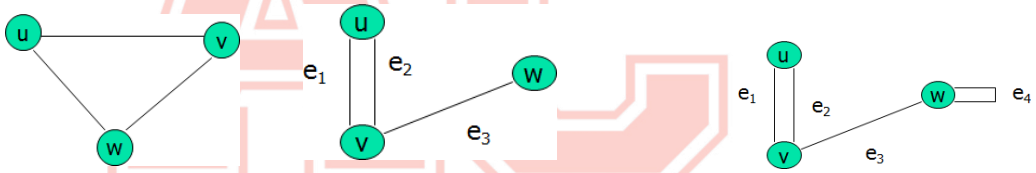
Unit Five – Graph Theory

4.1 Introduction

- **Graph:** 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 endpoints.
  - ✓ V= {a, b, c, d, e}
  - ✓ E= {(a, b),(a ,c),(a, d),(b, e),(c, d),(c, e),(d, e)}



- **Infinite Graph:** A graph with an infinite vertex set or an infinite number of edges is called an infinite graph.
- **Finite Graph:** A graph with a finite vertex set and a finite edge set is called a finite graph.
- **Simple Graph**: A graph in which each edge connects two different vertices and where no two edges connect the same pair of vertices is called a simple graph.
  - ✓ Representation Example: G(V, E), V = {u, v, w}, E = {{u, v}, {v, w}, {u, w}}



Simple Graph Multigraph Pseudograph

Figure 4.1 (a) Simple Graph, (b) Multigraph and (c) Pseudograph

- **Multigraph**: Graphs that may have multiple edges connecting the same vertices are called multigraphs. For example the edges e1 and e2 have same set of vertices.
- **Loops**: An edge in a graph that connects a vertex to itself is called a loop.
- **Pseudographs:** A graph that may include loops, and possibly multiple edges connecting the same pair of vertices or a vertex to itself, is sometimes called Pseudographs.
- **Directed graph (Digraph):** A directed graph (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. Representation Example: G(V, E), V = {u, v, w}, E = {(u, v), (v, w), (w, u)}.

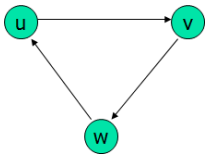


Figure 4.2.a :(Simple) Directed graph

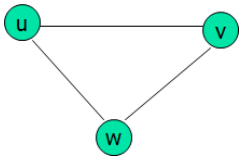


Figure 4.2.b: Undirected graph

- **Undirected Graph:** A graph in which the direction of connection or communication is not specified of defined is called undirected graph.
- **Simple Directed Graph: A** directed graph which has no loops and no two ordered pair of vertices are connected by more than one edge is called a simple directed graph.
- **Directed Multigraphs**: A graphs that may have multiple directed edges from a vertex to a second vertex are called directed multigraphs. When there are m directed edges, each associated to an ordered pair of vertices (u, v), then (u, v) is an edge of multiplicity m.
- **Mixed graph:** A graph with both directed and undirected edges is called a mixed graph.

4.2 Graph Models

Social Networks Graphs

- **Acquaintanceship and Friendship Graphs**: It is used to represent whether two people know each other, that is, whether they are acquainted, or whether they are friends. Each person in a particular group of people is represented by a vertex. An undirected edge is used to connect two people when these people know each other, when we are concerned only with acquaintanceship, or whether they are friends. No multiple edges and usually no loops are used.
- **Influence Graphs**: A directed graph called an influence graph can be used to model that certain people can influence the thinking of others. Each person of the group is represented by a vertex. There is a directed edge from vertex a to vertex b when the person represented by vertex a can influence the person represented by vertex b. This graph does not contain loops and it does not contain multiple directed edges.
- **Collaboration Graphs**: It is used to model social networks where two people are related by working together in a particular way. These graphs are undirected and there are no multiple edges or loops. Vertices in these graphs represent people; two people are connected by an undirected edge when the people have collaborated. The **Hollywood graph** is a collaborator graph that represents actors by vertices and connects two actors with an edge if they have worked together on a movie or television show.

Communication Networks

- **Call Graphs:** These graphs can be used to model telephone calls made in a network. A directed multigraph can be used to model calls where each telephone number is represented by a vertex and each telephone call is represented by a directed edge. The edge representing a call starts at the telephone number from which the call was made and ends at the telephone number to which the call was made. Multiple directed graphs are possible, but it contains no loops.

Information Networks Graphs

- **Web Graph:** The World Wide Web can be modeled as a directed graph where each Web page is represented by a vertex and where an edge starts at the Web page a and ends at the Web page b if there is a link on a pointing to b. Multigraph and loops are possible.
- **Citation Graph**: It can be used to represent citations in different types of documents, including academic papers, patents, and legal opinions. Each document is represented by a vertex, and there is an edge from one document to a second document if the first

document cites the second in its citation list. A citation graph is a directed graph without loops or multiple edges.

Software Design Applications Graphs

- **Module Dependency Graphs:** It provides a useful tool for understanding how different modules of a program interact. In a program dependency graph, each module is represented by a vertex. There is a directed edge from a module to a second module if the second module depends on the first.
- **Precedence Graphs and Concurrent Processing:** The dependence of statements (to be executed) on previous statements can be represented by a directed graph. Each statement is represented by a vertex, and there is an edge from one statement to a second statement if the second statement cannot be executed before the first statement.

Transportation Networks Graphs

- **Airline Routes**: It can model by representing each airport by a vertex. All the flights by a particular airline each day can be model using a directed edge to represent each flight, going from the vertex representing the departure airport to the vertex representing the destination airport. It can be a directed multigraph, as there may be multiple flights from one airport to some other airport during the same day.
- **Road Networks Graphs**: Vertices represent intersections and edges represent roads. A simple undirected graph used to model two-way roads and directed graph represent one-way road. Multiple undirected edges represent multiple two-way roads connecting the same two intersections. Multiple directed edges represent multiple one-way roads that start at one intersection and end at a second intersection. Loops represent loop roads. Mixed graphs are needed to model road networks that include both one-way and two-way roads.

Biological Networks Graphs

- **Niche Overlap Graphs**: It is used to models the interaction of different species of animals in the ecosystem. Each species is represented by a vertex. An undirected edge connects two vertices if the two species represented by these vertices compete with each other for same food. This model has no loops or multiple edges.
- **Protein Interaction Graphs:** It is an undirected graph in which each protein is represented by a vertex, with an edge connecting the vertices representing each pair of proteins that interact. It can be used to deduce important biological information, such as by identifying the most important protein for various functions and the functionality of newly discovered proteins.

Tournaments Graphs

- **Round-Robin Tournaments:** A tournament where each team play every other team exactly once and no ties are allowed is called a round-robin tournament. Such tournaments can be modeled using directed graphs where each team is represented by a vertex. Note that (a, b) is an edge if team a beats team b. This graph is a simple directed graph, containing no loops or multiple directed edges.
- **Single-Elimination Tournaments:** A tournament where each contestant is eliminated after one loss is called a single-elimination tournament. We can model such a tournament

using a vertex to represent each game and a directed edge to connect a game to the next game the winner of this game played in.

4.3 Graph Terminology

- **Adjacent Vertices:** Two vertices u and v in an undirected graph G are called adjacent in G if u and v are endpoints of an edge e of G. Such an edge e is called incident with the vertices u and v and e is said to connect u and v.
- **Degree of Vertex:** The degree of a vertex in an undirected graph is the number of edges incident with it.
  - ✓ A loop at a vertex contributes twice to the degree of that vertex.
  - ✓ The degree of the vertex v is denoted by deg(v).
  - ✓ A vertex of degree zero is called **isolated**.
  - ✓ A vertex with degree one is called **pendant.**
  - ✓ Example: deg(a) = 4, deg(b) = deg(e) = 6, deg(c) = 1, and deg(d) = 5.

Directed Graph Undirected Graph

Figure 4.3: (a) Directed Graph and (b) Undirected Graph

- **Degree of Vertex For directed graph**
  - ✓ 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 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.
  - ✓ **In-degree**: the in-degree of a vertex v is the number of edges that are incoming - towards v (head of edge).It is denoted by deg– (u).
  - ✓ **Out-degree**: the out-degree of a vertex v is the number of edges that are outgoing from v (tail of edge).It is denoted by deg+ (u).
  - ✓ Example: $\deg^-(u) = 0$, $\deg^+(u) = 2$, and $\deg^-(w) = 2$, $\deg^+(w) = 0$.

The Handshaking Theorem

- Let G = (V, E) be an undirected graph with m edges. Then

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

- This means the sum of degrees of all the vertices of an undirected graph is equal to twice the number of edges.
- **Theorem:** An undirected graph has even number of vertices having odd degree.
  - ✓ Proof: Let G = (V, E) be an undirected graph with m number of edges. Le t the vertex V is partitioned into V1 and V2; the set of vertices having even degree

vertices and odd degree vertices respectively and V1 ∩ V2 =∅, and V1 ∪V2 = V. Then according to handshaking theorem

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

This can be rewritten as:

$$2e = \sum_{v \in V1} deg(v) + \sum_{v \in V2} deg(v) \ldots\ldots.(1)$$

- ✓ The first term in the right-hand side of the equation one is even since each vertex in set V1 has even degree and sum of any even n umbers is always even. Furthermore, the sum of the two terms on the right-hand side of the equation one is even, because this sum is 2m. Hence, the second term in the sum is also even. Hence the number of vertices with odd degrees must be even.

- **Theorem:** 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 sum of the in-degrees and the sum of the out-degrees of all vertices in a graph with directed edges are the same. Both of these sums are the number of edges in the graph.

4.4 Special Simple Graphs

- **Complete Graphs:** A complete graph on *n* vertices, denoted by K_n, is a simple graph that contains exactly one edge between each pair of distinct vertices.
- **Non-complete**: A simple graph for which there is at least one pair of distinct vertex not connected by an edge is called non-complete.

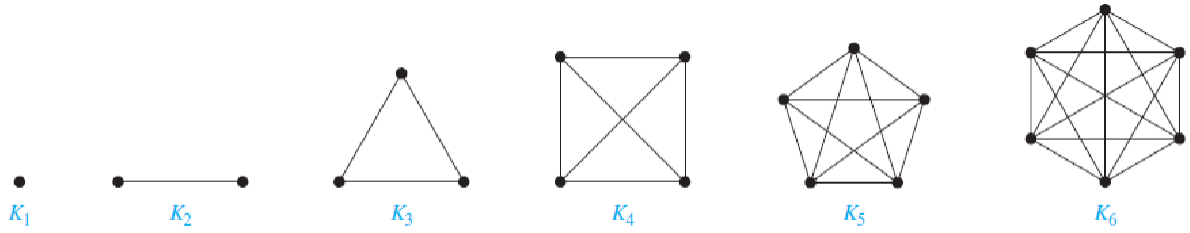


Figure 4.4: The Graphs K_n for $1 \leq n \leq 6$

- **Cycles:** A cycle Cn, n ≥ 3, consists of n vertices v1, v2, . . .,vn and edges {v1, v2}, {v2, v3}, . . . , {vn−1, vn}, and {vn, v1}.

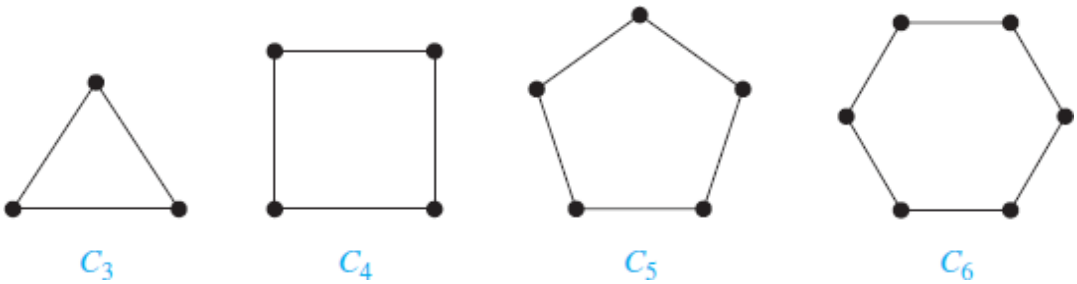


Figure 4.5 : The Cycles C_3, C_4, C_5, and C_6

- **Wheel:** We obtain the wheel W_n when we add an additional vertex to the cycle C_n, for n ≥ 3, and connect this new vertex to each of the n vertices in C_n by adding new edges.

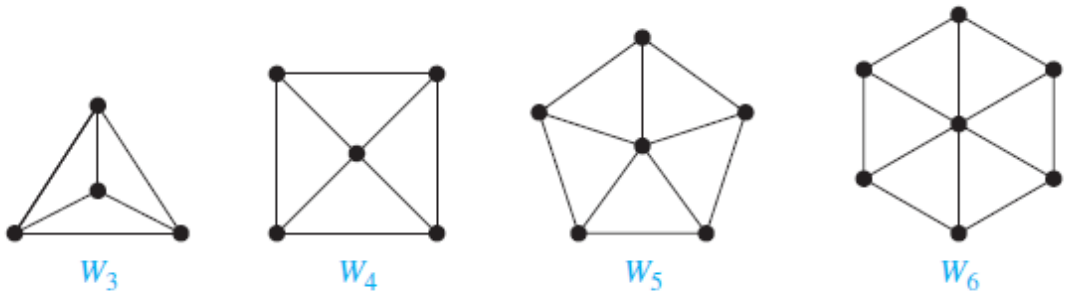


Figure 4.6: The Wheels W3, W4, W5, and W6

- **n-Cubes**: An n-dimensional hypercube, or n-cube, denoted by Q_n, is a graph that has vertices representing the 2^n bit strings of length n. Two vertices are adjacent if and only if the bit strings n that they represent differ in exactly one bit position. For example: Q_3 - n=3 there will be 2^3= 8 bit strings as vertices namely 000, 001,010,011,100,101,110,111. Note that adjacent vertices differ in only one bit position.

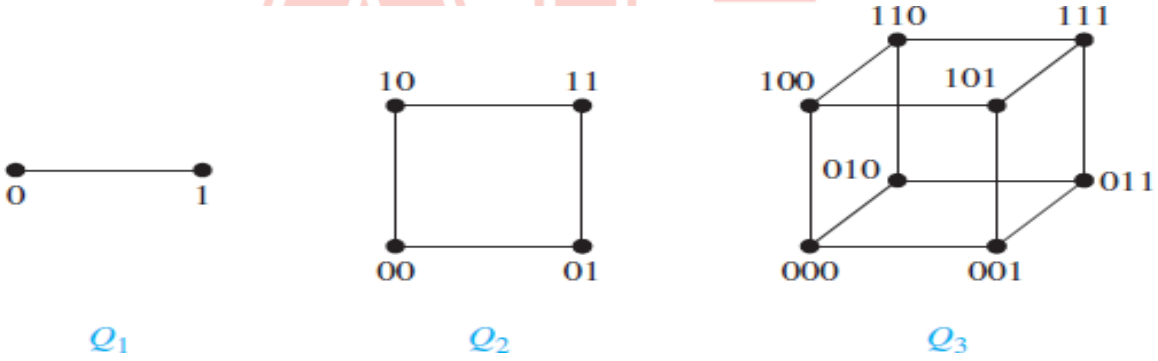


Figure 4.7: The n-cube Q_n, n = 1, 2, 3

- **Bipartite Graphs:** A simple graph G is called bipartite if its vertex set V can be partitioned into two disjoint sets V1 and V2 such that every edge in the graph connects a vertex in V1 and a vertex in V2. When this condition holds, we call the pair (V1, V2) a bipartition of the vertex set V of G.
- **Example I:** Is C_3 bipartite?
  - ✓ No, because there is no way to partition the vertices into two sets so that there are no edges with both endpoints in the same set.
- **Example2:** Is C_6 bipartite?
  - ✓ C_6 is bipartite because its vertex set can be partitioned into the two setsV1 = {v1, v3, v5} and V2 = {v2, v4, v6}, and every edge of C_6 connects a vertex in V1 and a vertex in V2.

Figure 4.8: C_6 change into Bipartite

- **Complete Bipartite**: Graphs A complete bipartite graph $K_{m,n}$ is a graph that has its vertex set partitioned into two subsets of m and n vertices, respectively with an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset.

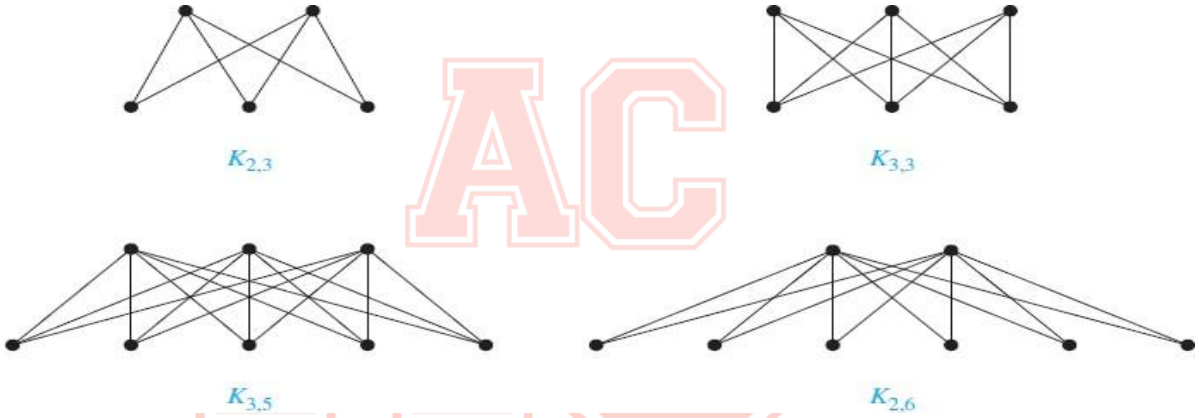


Figure 4.9: Complete Bipartite

- **Regular Graph:** A graph is regular if every vertex has the same degree.
- Example: The complete graph K_n is regular of degree n-1.
- Example: A cycle graph is regular of degree 2.
- Example: A cube graph Q_n is regular of degree n.

4.5 Operations on Graphs

- **Subgraph**: A subgraph of a graph G = (V, E) is a graph H = (W, F) where W⊆V and F⊆E.
- Representation example: V = {u, v, w}, E = ({u, v}, {v, w}, {w, u}}, H_1 , H_2

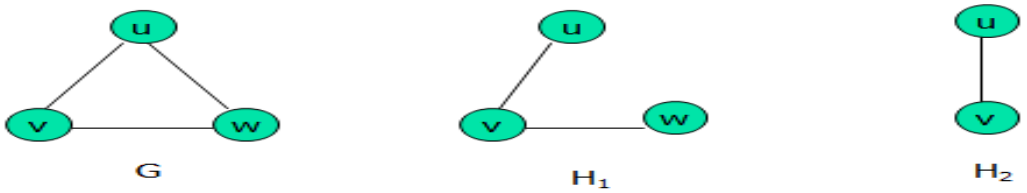


Figure 4.10: Subgraph

- **Union:** The union of two simple graphs G1= (V1, E1) and G2=(V2, E2) is a simple graph G=(v, E) such that V=V1∪ V2 and E= E1 ∪ E2. We call G= G1 ∪ G2.

- Representation example: V1 = {u, w}, E1 = {{u, w}}, V2 = {w, v}, E1 = {{w, v}}, V = {u, v ,w}, E = {{{u, w}, {{w, v}}

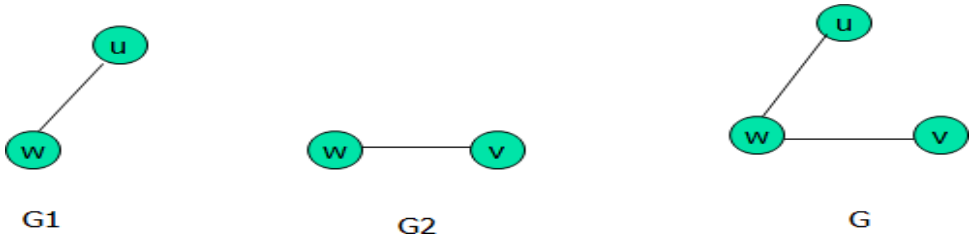


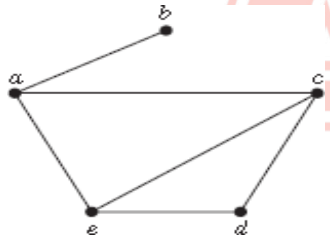
Figure 4.11: Graph Union

4.6 Graph Representations

- It is the techniques to represent the graph

Adjacency lists

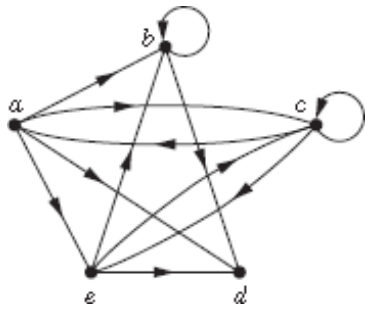
- Adjacency list specifies vertices that are adjacent to a vertex.
- This list contains adjacency entries for all vertices for an undirected graph. So, it is possible and very easy to drawn the graph if adjacency list is known.
- An Adjacency list for Simple Undirected Graph



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

Figure 4.12: Simple Undirected Graph

- For directed graphs, the entries of each vertex (u) in the adjacency list contains list of vertices that are incident to u or vertices at which directed edges from u terminates.
- An Adjacency list for simple Directed Graph



Initial Vertex	*Terminal Vertices*
a	*b, c, d, e*
b	*b, d*
c	*a, c, e*
d	
e	*b, c, d*

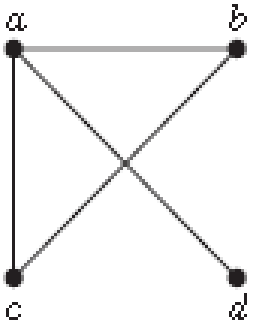
Figure4. 13: Simple Directed Graph

Adjacency Matrix

- Let G = (V, E) be a simple graph with |V| = n. Suppose that the vertices of G are listed in arbitrary order as v1, v2, …,vn. The adjacency matrix A (or A_G) of G, with respect to this

listing of the vertices, is the n×n zero-one matrix with 1 as its (i, j)th entry when v_i and v_j are adjacent, and 0 otherwise.

- In other words, for an adjacency matrix A = [a_{ij}],
  aij = 1 if {vi, vj} is an edge of G,
  aij = 0 otherwise.
- The adjacency matrix for undirected graph



(a)

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

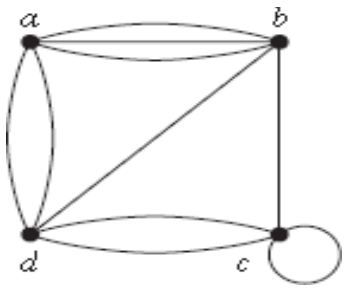
(b)

Figure 4.14: (a) Undirected Graph and (b) Adjacency Matrix

- Adjacency matrices of undirected graphs are always symmetric, that is, $a_{ij} = a_{ji}$, because both of these entries are 1 when v_i and v_j are adjacent, and both are 0 otherwise.
- The matrix for a directed graph G = (V, E) has a 1 in its (i, j) th position if there is an edge from v_i to v_j, where $v_1, v_2, \ldots, v_n$ is an arbitrary listing of the vertices of the directed graph.
- In other words, if A = [a_{ij}] is the adjacency matrix for the directed graph with respect to this listing of the vertices, then

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

- The adjacency matrix for a directed graph does not have to be symmetric, because there may not be an edge from v_j to v_i when there is an edge from v_i to v_j.
- For multigraph, When multiple edges connecting the same pair of vertices v_i and v_j, or multiple loops at the same vertex the adjacency matrix is the (i, j)th entry of this matrix equals the number of edges that are associated to {v_i , v_j }.
- All undirected graphs, including multigraphs and pseudographs, have symmetric adjacency matrices.



$$\begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}$$

(a) (b)

Figure4.15: (a) Multigraph and (b) Adjacency Matrix of Multigraph

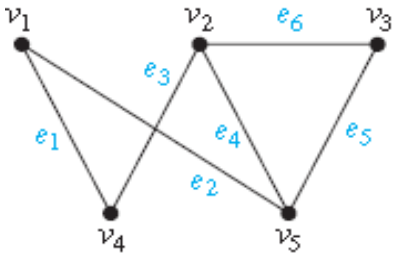
- The adjacency matrix for a directed multigraph, a_{ij} equals the number of edges that are associated to (v_i, v_j).

Incidence Matrices

- Let G = (V, E) be an undirected graph. Suppose that $v_1, v_2, \ldots, v_n$ are the vertices and $e_1, e_2, \ldots, e_m$ are the edges of G. Then the incidence matrix with respect to this ordering of V and E is the n × m matrix M = $[m_{ij}]$, where

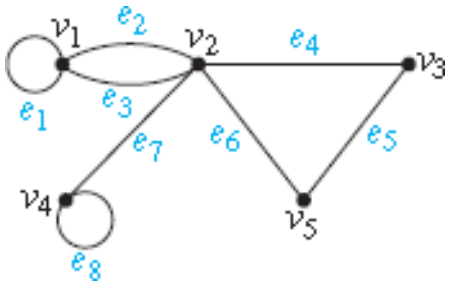
$$m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_i, \\ 0 & \text{otherwise.} \end{cases}$$

- The incidence matrix for undirected graph



$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}.$$

- Multiple edges are represented in the incidence matrix using columns with identical entries, because these edges are incident with the same pair of vertices. Loops are represented using a column with exactly one entry equal to 1, corresponding to the vertex that is incident with this loop.
- The incidence matrix for pseudograph



$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array}.$$

4.7 Connectivity

- **Path in Undirected Graph:** Let n be a nonnegative integer and G an undirected graph. A path of length n from u to v in G is a sequence of n edges $e_1, \ldots, e_n$ of G for which there exists a sequence $x_0 = u, x_1, \ldots, x_{n-1}, x_n = v$ of vertices such that e_i has, for $i = 1, \ldots, n$, the endpoints x_{i-1} and x_i.
- The **path is a circuit** if it begins and ends at the same vertex, that is, if u = v, and has length greater than zero. The path or circuit is said to pass through the vertices $x_1, x_2, \ldots, x_{n-1}$ or traverse the edges $e_1, e_2, \ldots, e_n$.
- A path or circuit is simple if it does not contain the same edge more than once.
- The term **walk** is used instead of **path**.
- **Closed walk** is used instead of circuit to indicate a walk that begins and ends at the same vertex, and **trail** is used to denote a walk that has no repeated edge (replacing the term simple path).
- **Path in Directed Graph**: Let n be a nonnegative integer and G a directed graph. A path of length n from u to v in G is a sequence of edges $e_1, e_2, \ldots, e_n$ of G such that e1 is associated with (x_0, x_1), e_2 is associated with (x_1, x_2), and so on, with e_n associated with (x_{n-1}, x_n), where $x_0 = u$ and $x_n = v$.
- When there are no multiple edges in the directed graph, this path is denoted by its vertex sequence $x_0, x_1, x_2, \ldots, x_n$. A path of length greater than zero that begins and ends at the same vertex is called a **circuit or cycle**. A path or circuit is called **simple** if it does not contain the same edge more than once.
- **Connected Graph**: An undirected graph is called connected if there is a path between every pair of distinct vertices of the graph.
- **Not-Connected Graph**: An undirected graph that is not connected is called disconnected.
- Example: G1 is the connected graph because for every pair of distinct vertices there is a path between them and G2 is the not-connected graph because there is no path between vertices a and d.

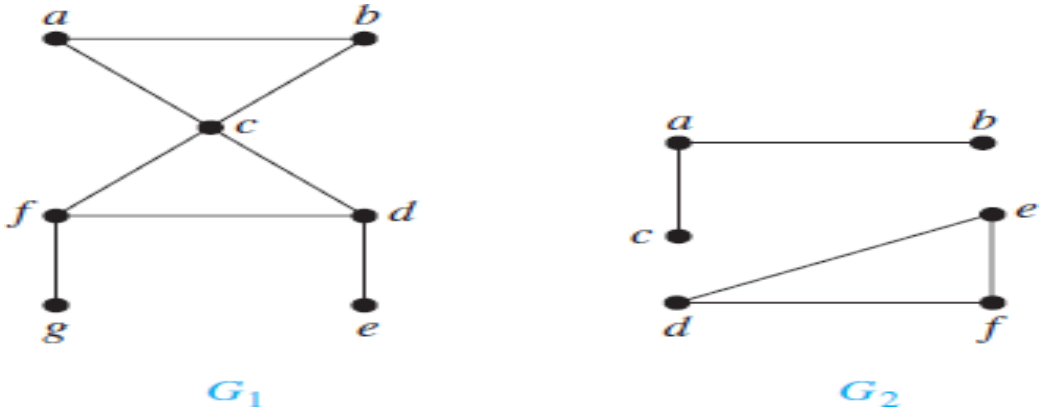
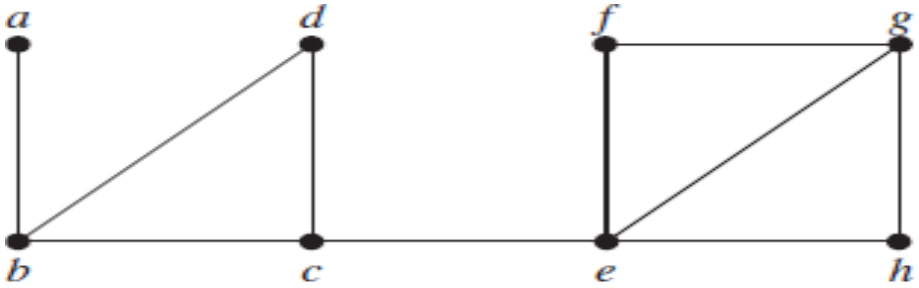


Figure 4.16: G1: Connected Graph and G2: Not-connected Graph

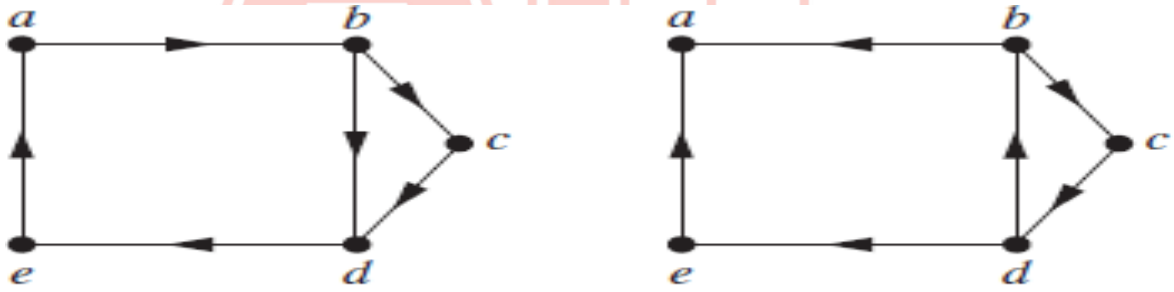
- **Cut vertices:** the removal from a graph of a vertex and all incident edges produces a subgraph with more connected components. Such vertices are called cut vertices (or articulation points).
- **Cut edge**: The removal of a cut vertex from a connected graph produces a subgraph that is not connected. Analogously, an edge whose removal produces a graph with more connected components than in the original graph is called a cut edge or bridge.

- **Example**: The cut vertices of G1 are b, c, and e. The removal of one of these vertices (and its adjacent edges) disconnects the graph. The cut edges are {a, b} and {c, e}. Removing either one of these edges disconnects G1.



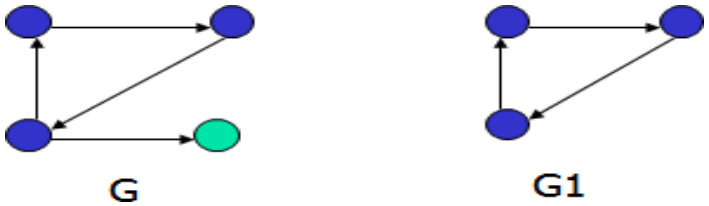
G_1

- **Strongly Connected:** A directed graph is strongly connected if there is a path from a to b and from b to a whenever a and b are vertices in the graph.
- **Weakly Connected**: A directed graph is weakly connected if there is a path between every two vertices in the underlying undirected graph. That is, a directed graph is weakly connected if and only if there is always a path between two vertices when the directions of the edges are disregarded.
- **Example**: G is strongly connected because there is a path between any two vertices in this directed graph. H is weakly connected, because there is a path between any two vertices in the underlying undirected graph of H.



G H

- **Strongly Connected Components or Strong Components:** The subgraphs of a directed graph G that are strongly connected but not contained in larger strongly connected subgraphs are called the strongly connected components or strong components of G.
- Representation example: G1 is the strongly connected component in G



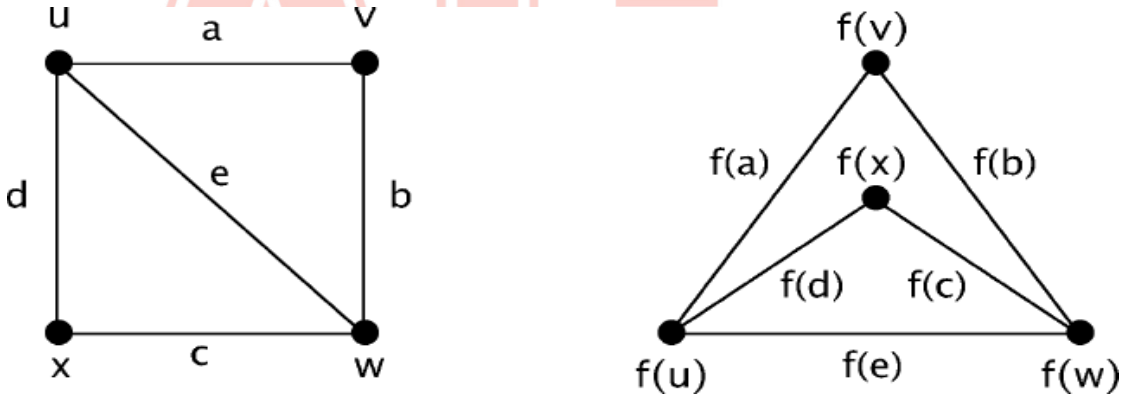
G G1

- **Isomorphism:** Two simple graph $G_1=(V_1, E_1)$ and $G_2=(V_2, E_2)$ are called isomorphic if there is one –to-one and onto function f from V_1 to V_2 with the property that u and v are

adjacent in G1 if and only if f(u) and f(v) are adjacent in G2 for all u,vεV1. Such a function is called isomorphism.

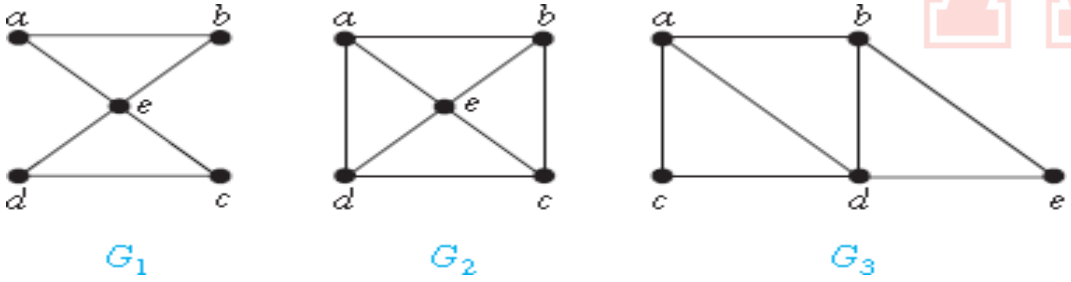
- In other words, G_1 and G_2 are isomorphic if their vertices can be ordered in such a way that the adjacency matrices M_{G1} and M_{G2} are identical.
- **Example:** the paths u1, u4,u3, u2, u5 in G and v3, v2, v1, v5, v4 in H both go through every vertex in the graph; start at a vertex of degree three; go through vertices of degrees two, three, and two, respectively; and end at a vertex of degree two. By following these paths through the graphs, we define the mapping f with f (u1) = v3, f (u4) = v2, f (u3) = v1, f (u2) = v5, and f (u5) = v4.Hence G and H are isomorphic.

- **Example 2:** The given two graphs are isomorphic.



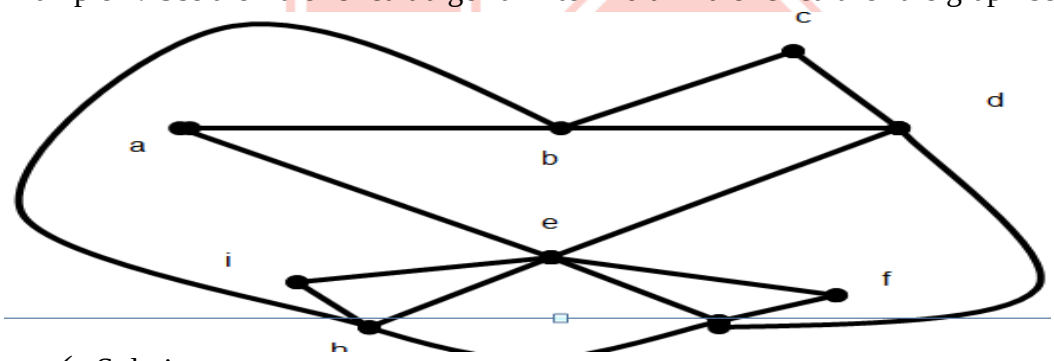
4.8 Euler Paths and Circuits

- An **Euler path** is a path using every edge of the graph G exactly once.
- An **Euler circuit** is an Euler path that returns to its start.
- A connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has even degree.
- A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.
- **Example:** The graph G1 has an Euler circuit, for example, a, e, c, d, e, b, a. Neither of the graphs G2 or G3 has an Euler circuit. G3 has an Euler path, namely, a, c, d, e, b, d, a, b. G2 does not have an Euler path.

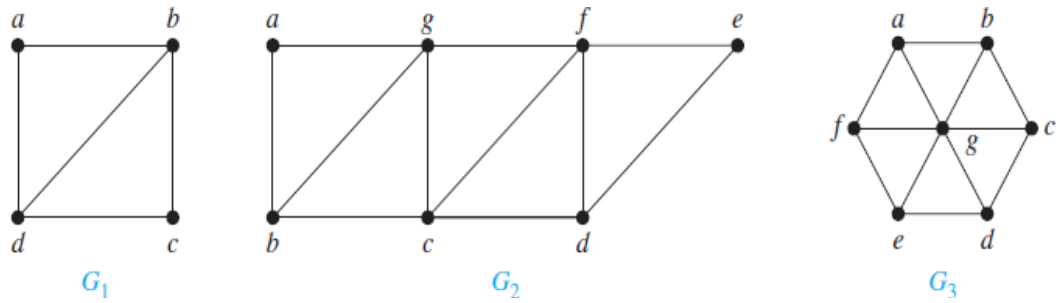


G_1 G_2 G_3

- **An algorithm for constructing Euler circuits**
  1. Make an initial circuit C1 by starting at an arbitrarily chosen vertex, making a path that returns to this vertex.
  2. If C1 is a circuit that includes all edges of the graph, go to step 6.
  3. If a vertex V that is in C1 and has an edge that is not in C1, then construct a circuit C2 that starts and ends at V using edges not in C1.
  4. Combine C1 and C2 to form a new circuit. Call this new circuit C1.
  5. Go to Step 3.
  6. C1 is an Euler circuit for the graph. Stop.
- Example 1: Use the Euler circuit algorithm to find an Euler circuit for the graph below.



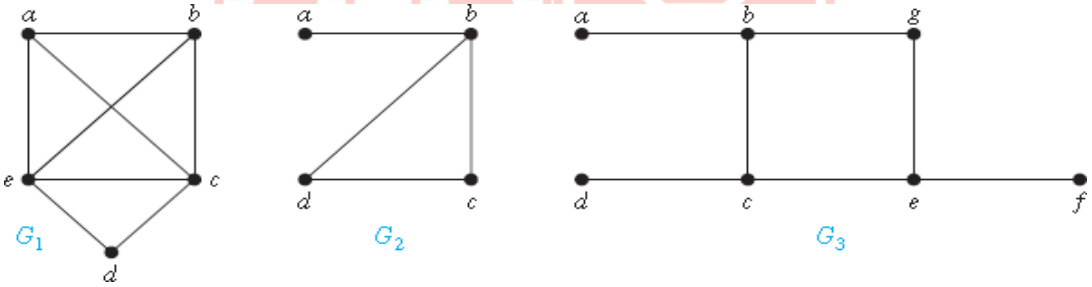
  - ✓ Solution:
    - Apply Step 1 of the algorithm. Choose vertex b, Let C1 be the circuit b, c, d, e, a, b.
    - C1 does not contain all edges of the graph, so go to Step 3 of the algorithm.
    - Choose vertex d. Let C2 be the circuit d, g, f, e, g, h, e, I, h, b, d.
    - Combine C1 and C2 by replacing vertex d in the circuit C2 with the circuit C1. Let C1 now be the circuit b, c, d, g, f, e, g, h, e, I, h, s, d, e, a, b.
    - Go to Step 5 of the algorithm.
    - C1 now contains all edges of the graph, so go to Step 6 of the algorithm and stop. C1 is an Euler circuit for the graph.
- Example 2: G1 contains exactly two vertices of odd degree, namely, b and d. Hence, it has an Euler path that must have b and d as its endpoints. One such Euler path is d, a, b, c, d, b. Similarly, G2has exactly two vertices of odd degree, namely, b and d. So it has an Euler path that must have b and d as endpoints. One such Euler path is b, a, g, f, e, d, c, g, b, c, f, d. G3 has no Euler path because it has six vertices of odd degree.



- **Applications of Euler Paths and Circuits**: Euler paths and circuits can be used to solve practical problem such as applications ask for a path or circuit that traverses each street in a neighborhood, each road in a transportation network, each connection in a utility grid, or each link in a communications network exactly once.

4.9 Hamilton Paths and Circuits

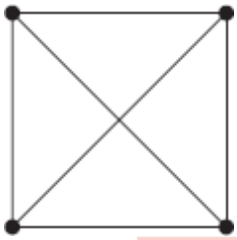
- **Hamilton path:** A simple path in a graph G that passes through every vertex exactly once is called a Hamilton path. That is, the simple path x0, x1, . . . , xn−1, xn in the graph G = (V ,E) is a Hamilton path if V = {x0, x1, . . . , xn−1, xn} and xi≠xj for $0 \leq i < j \leq n$,
- **Hamilton circuit:** A simple circuit in a graph G that passes through every vertex exactly once is called a Hamilton circuit. That is, the simple circuit x0, x1, . . . , xn−1, xn, x0 (with n > 0) is a Hamilton circuit if x0, x1, . . . , xn−1, xn is a Hamilton path.
- Example: G1 has a Hamilton circuit: a, b, c, d, e, a. There is no Hamilton circuit in G2, but G2 does have a Hamilton path a, b, c, d. G3 has neither a Hamilton circuit nor a Hamilton path, because any path containing all vertices must contain one of the edges {a, b},{e, f}, and {c, d} more than once.



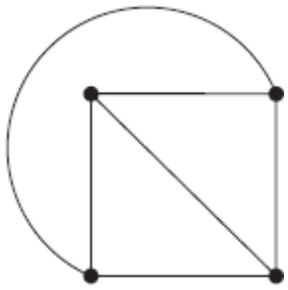
- **Sufficient Condition Hamilton circuit**: If G is a connected simple graph with n vertices where $n \geq 3$, then G has a Hamilton circuit if the degree of each vertex is at least n/2.
- **Dirac's Theorem:** If G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least n/2, then G has a Hamilton circuit.
- Example: Show that K_n has a Hamilton circuit whenever $n \geq 3$.
  - ✓ For K_n, degree of each vertex is (n-1) which is greater than n/2 for $n \geq 3$. Thus K_n for $n \geq 3$ has a Hamilton circuit.
- **Applications of Hamilton Circuits**: Hamilton paths and circuits can be used to solve practical applications such as for a path or circuit that visits each road intersection in a city, each place pipelines intersect in a utility grid, or each node in a communications network exactly once.

4.10 Planar Graphs

- **Planar Graph**: A graph is called planar if it can be drawn in the plane without any edges crossing, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. In other words, it can be drawn in such a way that no edges cross each other.
- Example 1: Is K_4 planar?
  - ✓ Solution: K_4 is planar because it can be drawn without crossings, as shown in Figure.

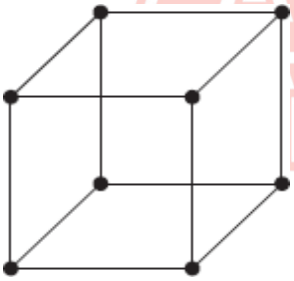


Graph K_4

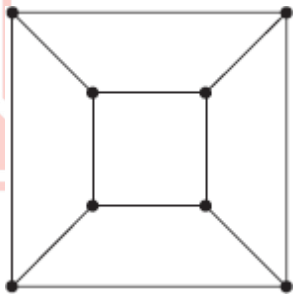


K_4 drawn with no crossing

- Example 2: Is Q_3 planar?
  - ✓ Solution: Q_3 is planar, because it can be drawn without any edges crossing, as shown in Figure.



\ Graph Q_3



A Planar representation of Q_3

- **Applications of Planar Graphs**: Planarity of graphs plays an important role in the design of electronic circuits. We can model a circuit with a graph by representing components of the circuit by vertices and connections between them by edges. We can print a circuit on a single board with no connections crossing if the graph representing the circuit is planar.
- **Euler's Formula:** Let G be a connected planar simple graph with e edges and v vertices. Let r be the number of regions in a planar representation of G. Then $r = e - v + 2$.
  - ✓ **Proof:** Suppose G is planar graph with a sequence of subgraphs G_1, G2, . . . , G_e = G, successively adding an edge at each stage.
  - ✓ Arbitrarily pick one edge of G to obtain G1. Obtain G_n from G_{n-1} by arbitrarily adding an edge that is incident with a vertex already in G_{n-1}, adding the other vertex incident with this edge if it is not already in G_{n-1}.
  - ✓ Let r_n, e_n, and v_n represent the number of regions, edges, and vertices of the planar representation of G_n respectively.

- ✓ The relationship $r_1 = e_1 - v_1 + 2$ is true for G_1, because $e_1 = 1$, $v_1 = 2$, and $r_1 = 1$. This is shown in figure.

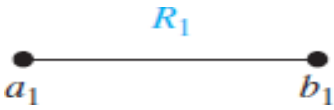


Figure 4.16: The Basis Case of the Proof of Euler's Formula.

- ✓ Now assume that $r_k = e_k - v_k + 2$. Let $\{a_{k+1}, b_{k+1}\}$ be the edge that is added to G_k to obtain G_{k+1}. There are two possibilities to consider.
- ✓ In the first case, both a_{k+1} and b_{k+1} are already in G_k. These two vertices must be on the boundary of a common region R. The addition of this new edge splits R into two regions. Consequently, in this case, $r_{k+1} = r_k + 1$, $e_{k+1} = e_k + 1$, and $v_{k+1} = v_k$. Thus, each side of the formula relating the number of regions, edges, and vertices increases by exactly one, so this formula is still true. In other words, $r_{k+1} = e_{k+1} - v_{k+1} + 2$. This case is illustrated in Figure 2.a.

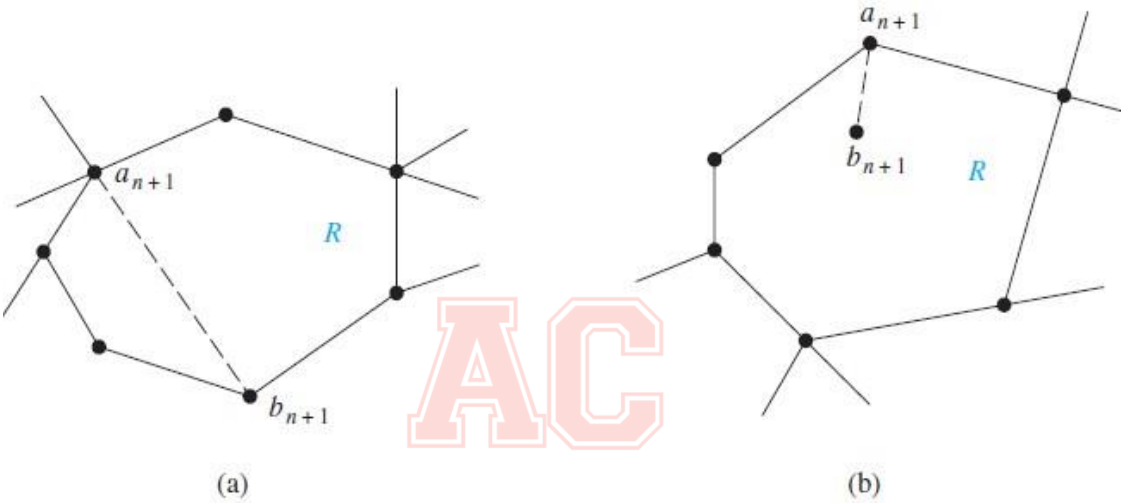


Figure 4.17: Adding an Edge to G_n to Produce G_{n+1}.

- ✓ In the second case, one of the two vertices of the new edge is not already in G_k. Suppose that a_{k+1} is in G_k but that b_{k+1} is not. Adding this new edge does not produce any new regions, because b_{k+1} must be in a region that has a_{k+1} on its boundary.
- ✓ Consequently, $r_{k+1} = r_k$. Moreover, $e_{k+1} = e_k + 1$and $v_{k+1} = vk + 1$. Each side of the formula relating the number of regions, edges, and vertices remains the same, so the formula is still true. In other words, $r_{k+1} = e_{k+1} - v_{k+1} + 2$. This case is illustrated in Figure 2.b.

- **Corollary 1:** If G is a connected planar simple graph with e edges and v vertices, where $v \geq 3$, then $e \leq 3v - 6$.
  - ✓ Proof: A connected planar simple graph drawn in the plane divides the plane into regions, say r of them. The degree of each region is at least three. The sum of the degrees of the regions is exactly twice the number of edges in the graph, because each edge occurs on the boundary of a region exactly twice. Because each region has degree greater than or equal to three, it follows that

$$2e = \sum_{\text{all regions } R} \deg(R) \geq 3r.$$

Hence,

(2/3)e ≥ r

Using r = e − v + 2 (Euler's formula), we obtain

e − v + 2 ≤ (2/3)e.

It follows that e/3 ≤ v − 2.

This shows that e ≤ 3v − 6.

- **Corollary 2:** If G is a connected planar simple graph, then G has a vertex of degree not exceeding five.
  - ✓ Proof: If G has one or two vertices, the result is true. If G has at least three vertices, by Corollary 1 we know that e ≤ 3v − 6, so 2e ≤ 6v − 12. If the degree of every vertex were at least six, then because $2e = \sum_{v \in V} \deg(v)$ (by the handshaking theorem), we would have 2e ≥ 6v. But this contradicts the inequality 2e ≤ 6v − 12. It follows that there must be a vertex with degree no greater than five.
- Example 3: Show that K_5 is nonplanar using Corollary 1.
  - ✓ Solution: The graph K_5 has five vertices and 10 edges. The inequality e ≤ 3v − 6 is not satisfied for this graph because e = 10 and 3v − 6 = 9. Therefore, K5 is not planar.
- **Corollary 3:** If a connected planar simple graph has e edges and v vertices with v ≥ 3 and no circuits of length three, then e ≤ 2v − 4.
- Example 4: Use Corollary 3 to show that $K_{3,3}$ is nonplanar.
  - ✓ Solution: Because $K_{3,3}$ has no circuits of length three (this is easy to see because it is bipartite),Corollary 3 can be used. $K_{3,3}$ has six vertices and nine edges. Because e = 9 and 2v − 4 = 8, Corollary 3 shows that $K_{3,3}$ is nonplanar.
- Example 3: Is $K_{3,\ 3Planar}$?
  - ✓ Solution: $K_{3,3}$ has v=6, e=9 and it is with v ≥ 3 and has v no circuit of length 3. If $K_{3,3\ is}$ planar, then e ≤ 2v – 4 should be true.
    Now, 2v – 4= 2*6 - 4= 8 and e=9. Here 9 ≤ 8, which is false.
    Hence $K_{3,3}$ is not planar.

4.11 Shortest Path Problem

- **Weighted graphs:** Graphs that have a number assigned to each edge are called weighted graphs. In a weighted graph, each edge has an associated numerical value, called the weight of the edge. Edge weights may represent distances, costs, etc.Example: In a flight route graph, the weight of an edge represents the distance in miles between the endpoint airports.
- The **length** of a path in a weighted graph is the sum of the weights of the edges of this path.
- **Dijkstra's algorithm** solves the single-source shortest path problem for a non-negative weights graph. It finds the shortest path from an initial vertex, says, to all the other vertices. It works on both directed and undirected graphs.
  - ✓ **Input:** Weighted graph G= {E, V} and source vertex vεV, such that all edge weights are nonnegative.

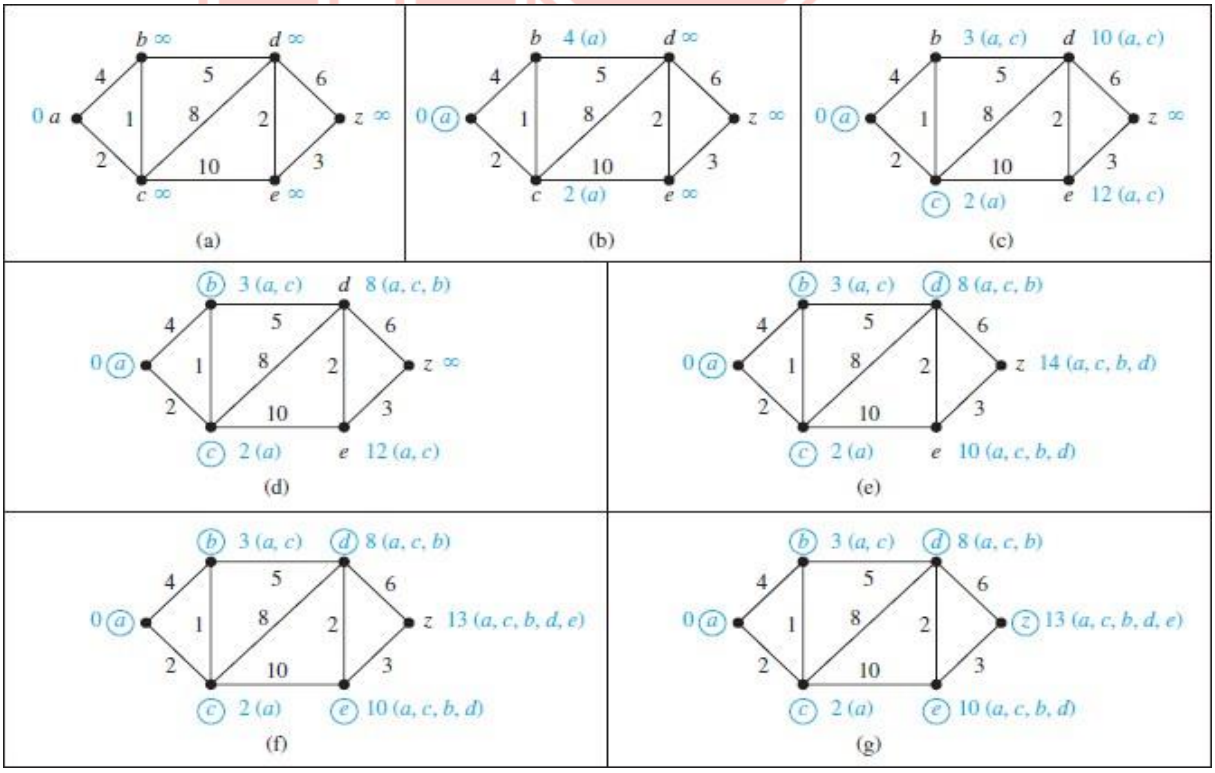
- ✓ **Output:** Lengths of shortest paths (or the shortest paths themselves) from a given source vertex vεV to all other vertices.

- **Dijkstra's Algorithm.**

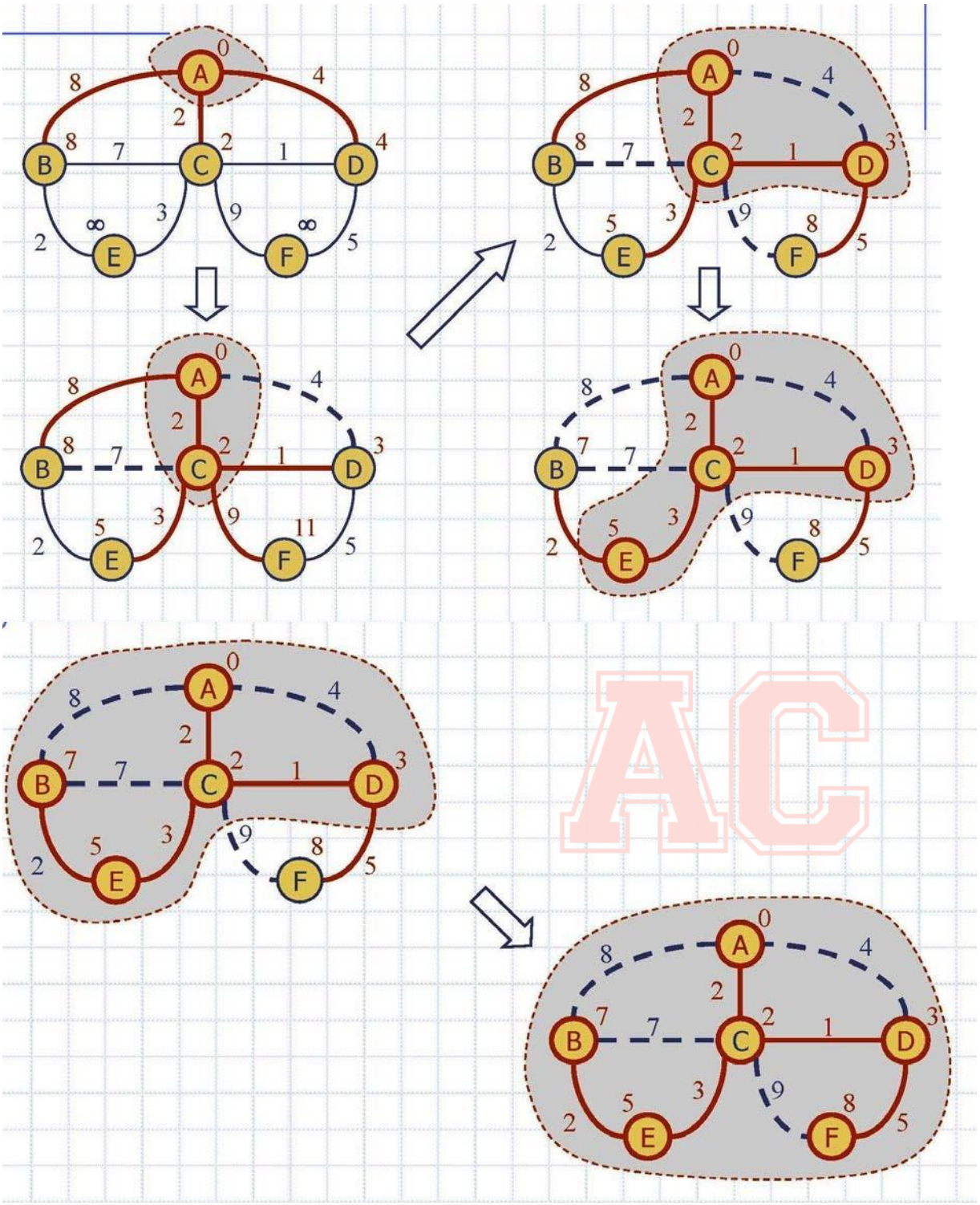
```
Dijkstra(G: weighted connected simple graph, with all weights positive)
    {G has vertices a = v0, v1, . . . , vn = z and lengths w(vi , vj ) where w(vi ,
    vj )=∞ if {vi , vj } is not an edge in G}
 For i := 1 to n
        L(vi ) :=∞
 L(a) := 0
 S: =∅
    {The labels are now initialized so that the label of a is 0 and all other
    labels are∞, and S is the empty set}
 While z ∈ S
    u := a vertex not in S with L(u) minimal
    S: = S U {u}
 For all vertices v not in S
    if L(u) + w(u, v) < L(v) then L(v) := L(u) + w(u, v)
      {This adds a vertex to S with minimal label and updates the labels of
    vertices not in S}
 Return L(z)
    {L(z) = length of a shortest path from a to z}
```

- Example 1: Use Dijkstra's algorithm to find the length of a shortest path between the vertices a and z in the weighted graph shown below.

\

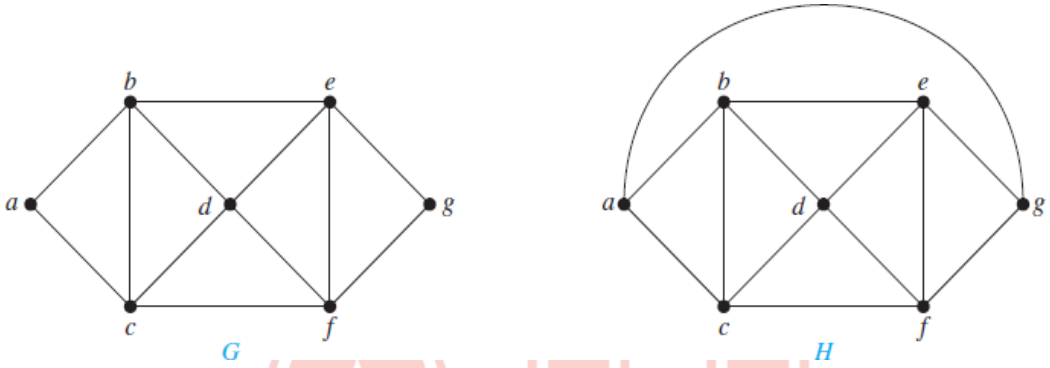


- Example 2: Use Dijkstra's algorithm to find the length of a shortest path from the vertex A to others in the weighted graph shown below.



4.12 Graph Coloring

- **Graph Coloring:** A coloring of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.
- **Chromatic number**: The chromatic number of a graph is the least number of colors needed for a coloring of this graph. The chromatic number of a graph G is denoted by $\chi(G)$. (Here χ is the Greek letter chi.)
- **Theorem 1: The Four Color Theorem:** The chromatic number of a planar graph is no greater than four.
- Example1: What are the chromatic numbers of the graphs G and H shown in Figure?



- ✓ Solution: The chromatic number of G is at least three, because the vertices a, b, and c must be assigned different colors. To see if G can be colored with three colors, assign red to a, blue to b, and green to c. Then, d can (and must) be colored red because it is adjacent to b and c. Furthermore, e can (and must) be colored green because it is adjacent only to vertices colored red and blue, and f can (and must) be colored blue because it is adjacent only to vertices colored red and green. Finally, g can (and must) be colored red because it is adjacent only to vertices colored blue and green. This produces a coloring of G using exactly three colors.
- ✓ The graph H is made up of the graph G with an edge connecting a and g. Any attempt to color H using three colors must follow the same reasoning as that used to color G, except at the last stage, when all vertices other than g have been colored. Then, because g is adjacent (in H) to vertices colored red, blue, and green, a fourth color, say brown, needs to be used. Hence, H has a chromatic number equal to 4.

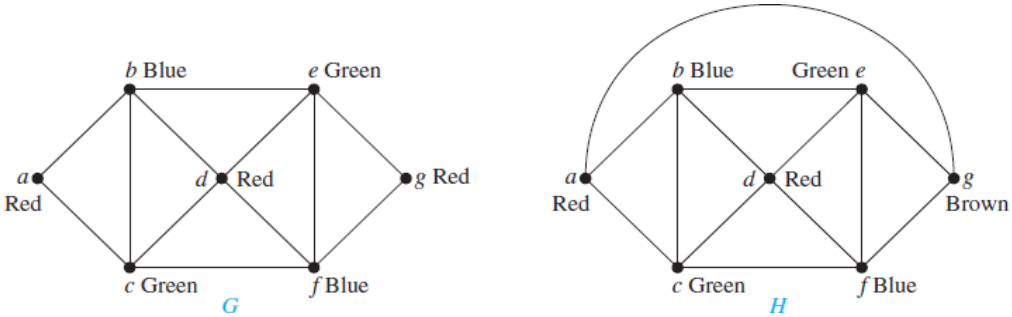


Figure 4.17: After Coloring Graph G and H

- Example 2: What is the chromatic number of K_n?
  - ✓ Solution: A coloring of K_n can be constructed using n colors by assigning a different color to each vertex. No two vertices can be assigned the same color, because every two vertices of this graph are adjacent. Hence, the chromatic number of K_n is n. That is, $\chi(Kn) = n$.
- Example 3: What is the chromatic number of the complete bipartite graph $K_{m,n}$, where m and n are positive integers?
  - ✓ Solution: The number of colors needed may seem to depend on m and n. Only two colors are needed, because $K_{m,n}$ is a bipartite graph. Hence, $\chi(K_{m,n}) = 2$. This means that we can color the set of m vertices with one color and the set of n vertices with a second color. Because edges connect only a vertex from the set of m vertices and a vertex from the set of n vertices, no two adjacent vertices have the same color.
- Example 4: What is the chromatic number of the graph C_n, where $n \geq 3$?
  - ✓ Solution: First consider some individual cases. Let n = 6. Pick a vertex and color it red. Proceed clockwise in the planar depiction of C_6. It is necessary to assign a second color, say blue, to the next vertex reached. Continue in the clockwise direction; the third vertex can be colored red, the fourth vertex blue, and the fifth vertex red. Finally, the sixth vertex, which is adjacent to the first, can be colored blue. Hence, the chromatic number of C_6 is 2.

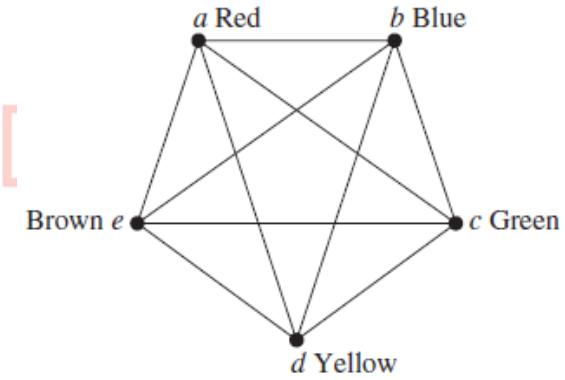


Figure 4.18: A Coloring of K_5

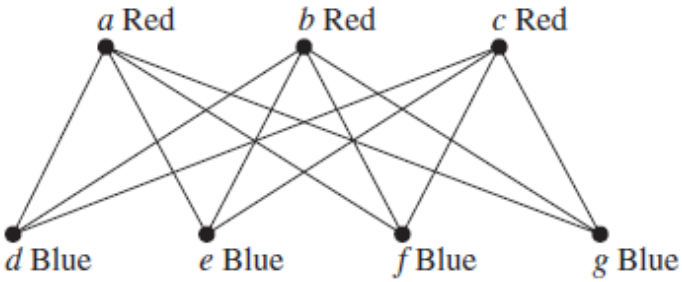


Figure 4.19: A Coloring of $K_{3,4}$

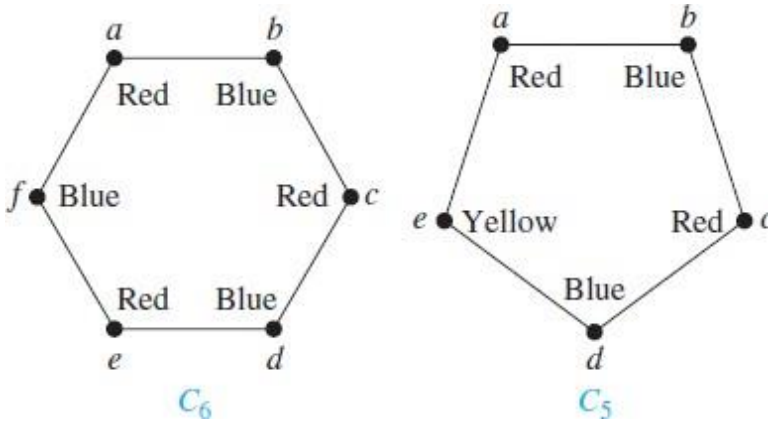


Figure 4.20: Colorings of C_5 and C_6

4.13 Trees

Tree terminologies

- **Tree:** A tree is a connected undirected graph with no simple circuits. A tree cannot contain multiple edges or loops. Therefore any tree must be a simple graph.
- An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.
- Example: G1 and G2 are trees, because both are connected graphs with no simple circuits. G3 is not a tree because e, b, a, d, e is a simple circuit in this graph. Finally, G4 is not a tree because it is not connected.

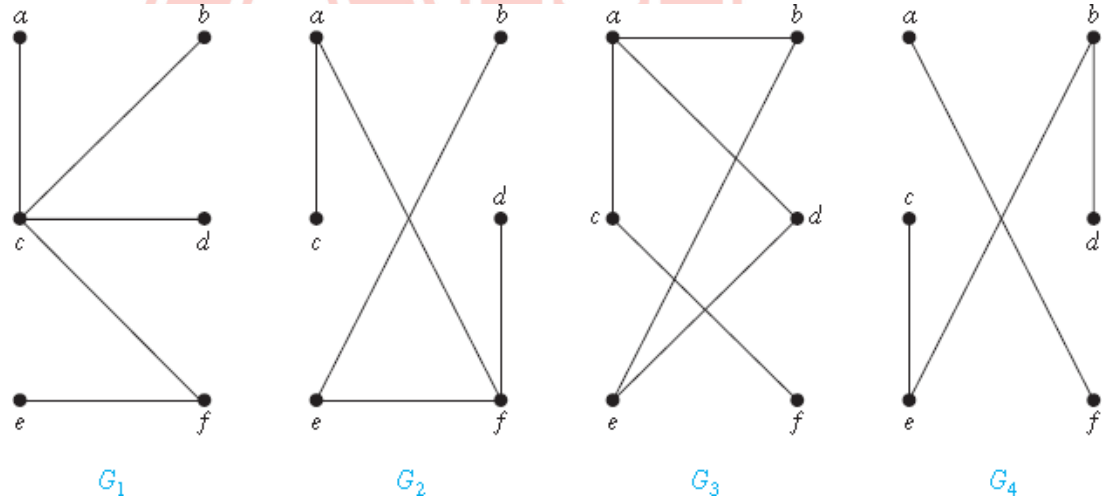
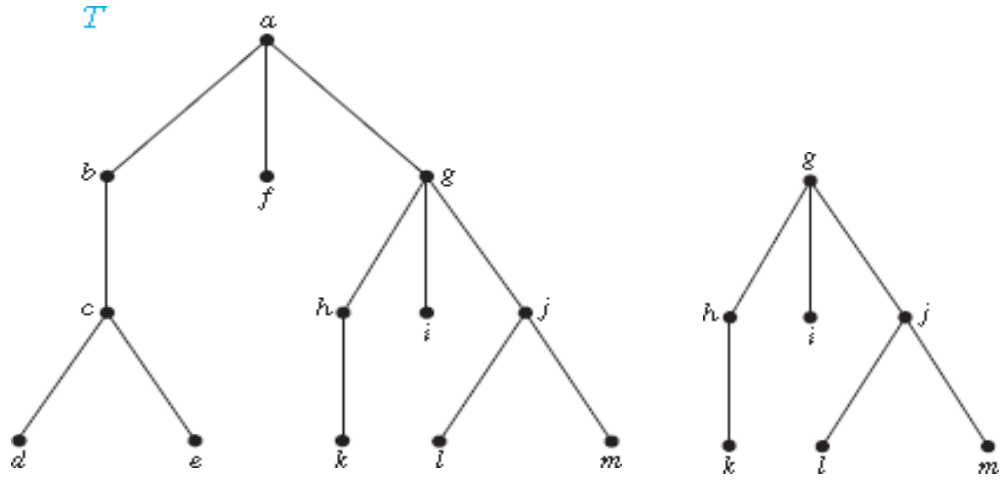


Figure 2.21: Connected Graph

- **Forest:** an undirected graph with no simple circuits.
- **Rooted Tree:** 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.
- **Parent:** If v is a vertex in Tree other than the root, a vertex u is the parent of v if there is a directed edge from u to v.
- **Child/Son:** If u is the parent of v, then v is called child/son of vertex u.
- **Siblings:** Vertices with the same parent are called siblings of each other.

- **Ancestors:** The ancestors of a vertex other than the root are the vertices in the path from the root to vertex v (excluding the vertex itself).
- **Descendants:** The descendants of a vertex v are those vertices that have v as an ancestor.
- **Leaf:** A vertex of a tree is called a leaf if it has no children.
- **Internal vertex (Non-leaf):** A vertex that has at least one child is called internal vertex.
- **Subtree:** If V is a vertex in a tree, the subtree with V as its root is the subgraph of the tree consisting of V and its descendants and all edges incident to these descendants.
- **Degree of Vertex (Node):** Degree of a vertex is the number of subtree of that vertex in a given tree.
- **Path**: A sequence of vertices and edges connecting a vertex with a descendant.
- **Level:** The number of edges in the path from the root to that vertex. The level of the root is zero and level of descendant is one more than its parent vertex.
- **Height of Tree:** The height of a tree is the number of edges on the longest path from the root to a leaf.
- **Height of node (Vertex):** The height of a node is the number of edges on the longest path from that node to a leaf.
- **Depth of a Vertex:** The length of the unique path from the root to that vertex. The depth of a tree is equal to the depth of its deepest leaf.
- **Example:** In the given Tree below,
  - ✓ 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 2.
  - ✓ The level of a is 0, level of d is 3 and level of f is 1.
  - ✓ The height of tree is 3, height of h is 2 in figure 1 & 2 and height of f is 0.
  - ✓ The depth of e is 3, depth of k is 3 in figure 1 and depth of k is 2 in figure 2.



(a) **(b)**

Figure 4.22: (a) Tree and (b) Subtree

- **m-ary Tree**: A rooted tree is called an m-ary tree if every internal vertex has no more than m children.
- **Full m-ary Tree:** The tree is called a full m-ary tree if every internal vertex has exactly m children.
- **Balanced m-ary Tree:** A rooted m-ary tree of height h is balanced if all leaves are at levels h or h –1, where h is the height of tree.
- **Binary Tree**: An m-ary tree with m = 2 is called a binary tree. Or a rooted tree is called a binary tree if every internal vertex has no more than two children. The tree is called a full binary tree if every internal vertex has exactly two children.
- **Ordered Rooted Tree:** 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 order from left to right. 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 1:T1 is a full binary tree because each of its internal vertices has two children. T2 is a full 3-ary tree because each of its internal vertices has three children. In T3 each internal vertex has five children, so T3 is a full 5-ary tree. T4 is not a full m-ary tree for any m because some of its internal vertices have two children and others have three children.

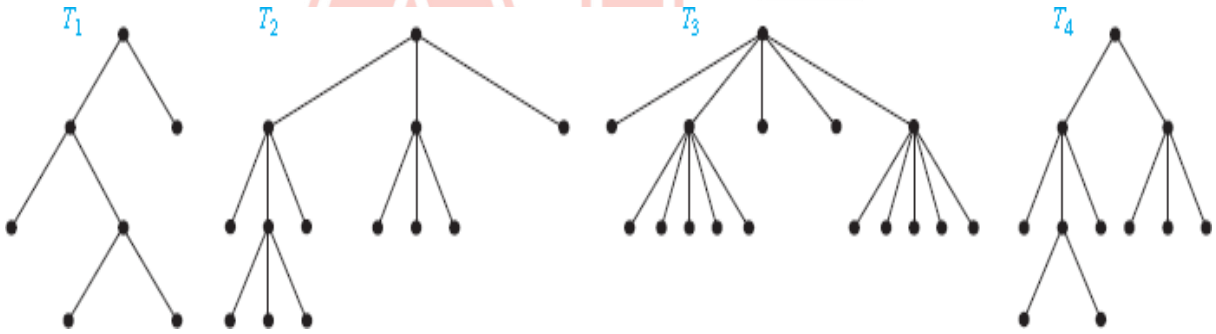


Figure4.23: m-ary Tree

- Example 2:T1 is balanced, because all its leaves are at levels 3 and 4. T2 is not balanced, because it has leaves at levels 2, 3, and 4. T3 is balanced, because all its leaves are at level 3.

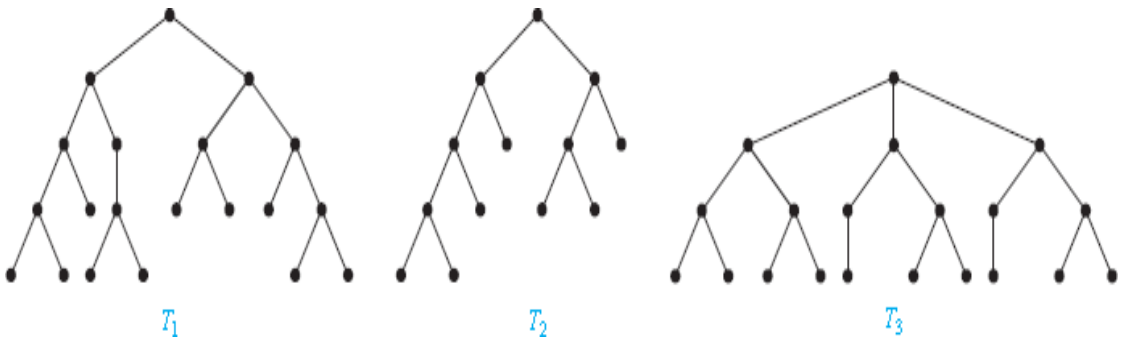


Figure 4.24: m-ary Trees

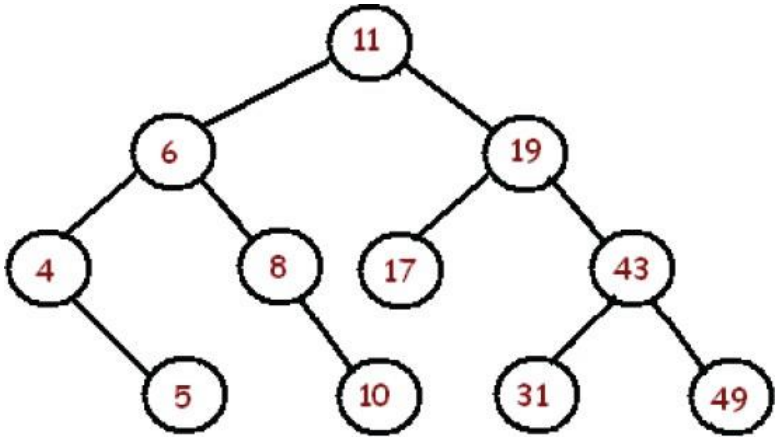
Properties of Trees

1. A tree with n vertices has n−1 edges.
2. A full m-ary tree with I internal vertices contains n = m*I+ 1 vertices.
3. A full m-ary tree with
    a) n vertices has I = (n − 1)/m internal vertices and L = [(m − 1)n + 1]/m leaves,
    b) I internal vertices has n = m*I + 1 vertices and L = (m − 1)*I + 1 leaves,
    c) L leaves has n = (m*L− 1)/ (m − 1) vertices and I = (L − 1)/(m − 1) internal vertices.
4. There are at most m^h leaves in an m-ary tree of height h.
5. If an m-ary tree of height h has L leaves, then $h \geq [\log_m L]$. If the m-ary tree is full and balanced, then $h= [\log_m L]$.

4.14 Applications of Trees

Binary search tree

- **Binary search tree** (BST): Binary search tree is a binary tree that is either empty or in which each node contains a data value that satisfies the following:
    a) All the keys in the left subtree are smaller than the key in its parent node.
    b) All the keys in the right subtree are greater than the key in its parent node.
    c) The left and right subtrees are also binary search tees.
    d) Duplicate keys are not allowed.
- Recursive procedure to insert New Node into the BST
    - ✓ If it is first key, assign it as root node.
    - ✓ If new node's key is less than current's key
        - If current node has a left child, search left
        - Else add new node as current's left child.
    - ✓ If new node's key is greater than current's key.
        - If current node has a right child, search right.
        - Else add new node as current's right child
- **Example:** Construct BST for given keys: 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31



- How to search a key in binary search tree?
    1. Start at the root node as current node
    2. If the search key's value matches the current node's key then found a match
    3. If search key's value is greater than current node's
        a) If the current node has a right child, search right

b) Else, no matching node in the tree

4. If search key is less than the current node's
   a) If the current node has a left child, search left
   b) Else, no matching node in the tree

- **Time Complexity for Searching:** Balanced BST with n nodes has a maximum order of $\log_2(n)$ levels, and thus it takes at most $\log_2(n)$ comparisons to find a particular node. But in worst case BST needs n comparisons to find a particular node.

Decision Trees

- **Decision Tree:** A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a decision tree. The possible solutions of the problem correspond to the paths to the leaves of this rooted tree.
- **Example:** Comparison of a, b, c for which one is greatest, which one is second greatest, and which one is smallest.

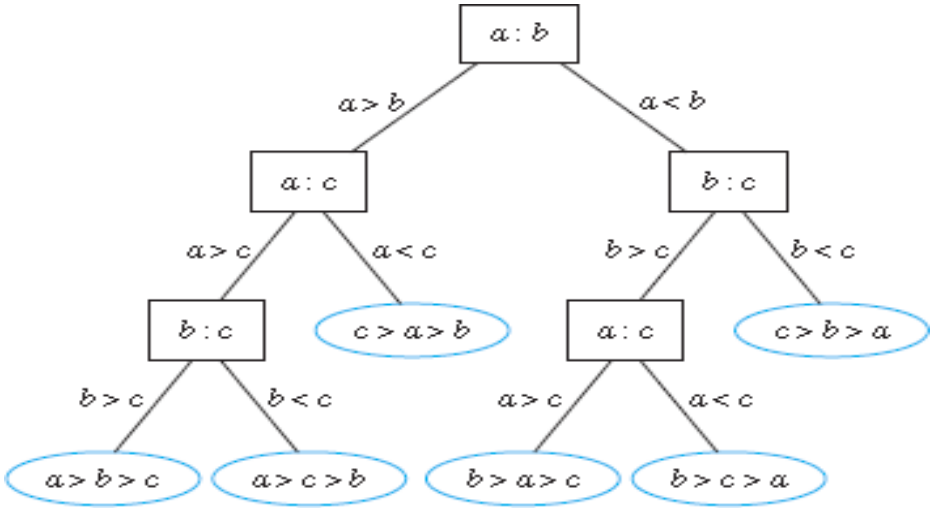


Figure4.25: A Decision Tree for Sorting Three Distinct Elements.

Prefix Codes

- A prefix code can be constructed from any binary tree where the left edge at each internal vertex is labeled by 0 and the right edge by a 1 and where the leaves are labeled by characters. Characters are encoded with the bit string constructed using the labels of the edges in the unique path from the root to the leaves.
- Example: the tree in Figure 1 represents the encoding of e by 0, a by 10, t by 110, n by 1110, and s by 1111.The tree representing a code can be used to decode a bit string. For instance, consider the word encoded by 11111011100 using the code in Figure 1. This bit string can be decoded by starting at the root, using the sequence of bits to form a path that stops when a leaf is reached. Each 0 bit takes the path down the edge leading to the left child of the last vertex in the path, and each 1 bit corresponds to the right child of this vertex. Consequently, the initial 1111 corresponds to the path starting at the root, going right four times, leading to a leaf in the graph that has s as its label, because the string 1111 is the code for s. Continuing with the fifth bit, we reach a leaf next after going right then left, when the vertex labeled with a, which is encoded by 10, is visited. Starting with

the seventh bit, we reach a leaf next after going right three times and then left, when the vertex labeled with n, which is encoded by 1110, is visited. Finally, the last bit, 0, leads to the leaf that is labeled with e. Therefore, the original word is sane.

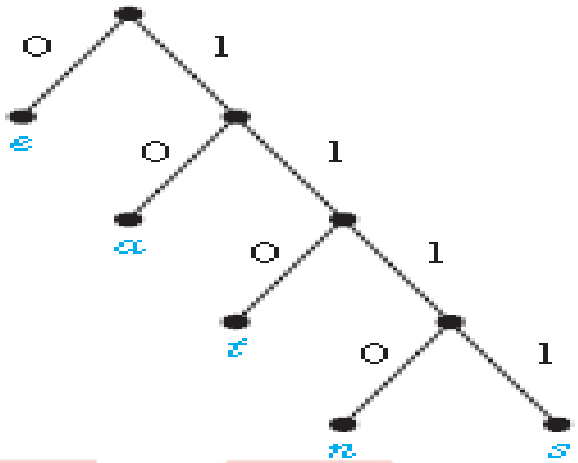


Figure4.26 : A Binary Tree with a Prefix Code.

Huffman Coding

- **Huffman Coding:** It is an algorithm that takes as input the frequencies of symbols in a string and produces as output a prefix code that encodes the string using the fewest possible bits, among all possible binary prefix codes for these symbols.
- Huffman coding is extensively used to compress bit strings representing text and it also plays an important role in compressing audio and image files.
- Huffman coding steps:
  - ✓ Sort the frequencies of alphabet in ascending order
  - ✓ Add two lowest frequencies and form a tree and assume the root of this tree as new item
  - ✓ Repeat step 1 and 2 until a single tree is formed containing all items.
- Example: Use Huffman coding to encode the following symbols with the frequencies listed: A: 0.08, B: 0.10, C: 0.12, D: 0.15, E: 0.20, F: 0.35. What is the average number of bits used to encode a character?
  - ✓ Solution: The encoding produced encodes A by 111, B by 110, C by 011, D by 010, E by 10, and F by 00. The average number of bits used to encode a symbol using this encoding is 2.45 bits per symbol.
    3 * 8 + 3 *10 + 3*12 + 3*15 + 2*20 + 2*35 = 245.

Game Trees

- Game Trees: In the game trees the vertices represent the positions that a game can be in as it progresses; the edges represent legal moves between these positions.
- We simplify game trees by representing all symmetric positions of a game by the same vertex. The same position of a game may be represented by different vertices if different sequences of moves lead to this position.
- The root represents the starting position. The usual convention is to represent vertices at even levels by boxes and vertices at odd levels by circles.
- When the game is in a position represented by a vertex at an even level, it is the first player's move; when the game is in a position represented by a vertex at an odd level, it is the second player's move.

- Game trees may be infinite when the games they represent never end, such as games that can enter infinite loops, but for most games there are rules that lead to finite game trees.
- The leaves of a game tree represent the final positions of a game.
- We assign a value to each leaf indicating the payoff to the first player if the game terminates in the position represented by this leaf.
- For games that are win–lose, we label a terminal vertex represented by a circle with a 1 to indicate a win by the first player and we label a terminal vertex represented by a box with a −1 to indicate a win by the second player.
- For games where draws are allowed, we label a terminal vertex corresponding to a draw position with a 0.
- Note that for win–lose games, we have assigned values to terminal vertices so that the larger the value, the better the outcome for the first player.

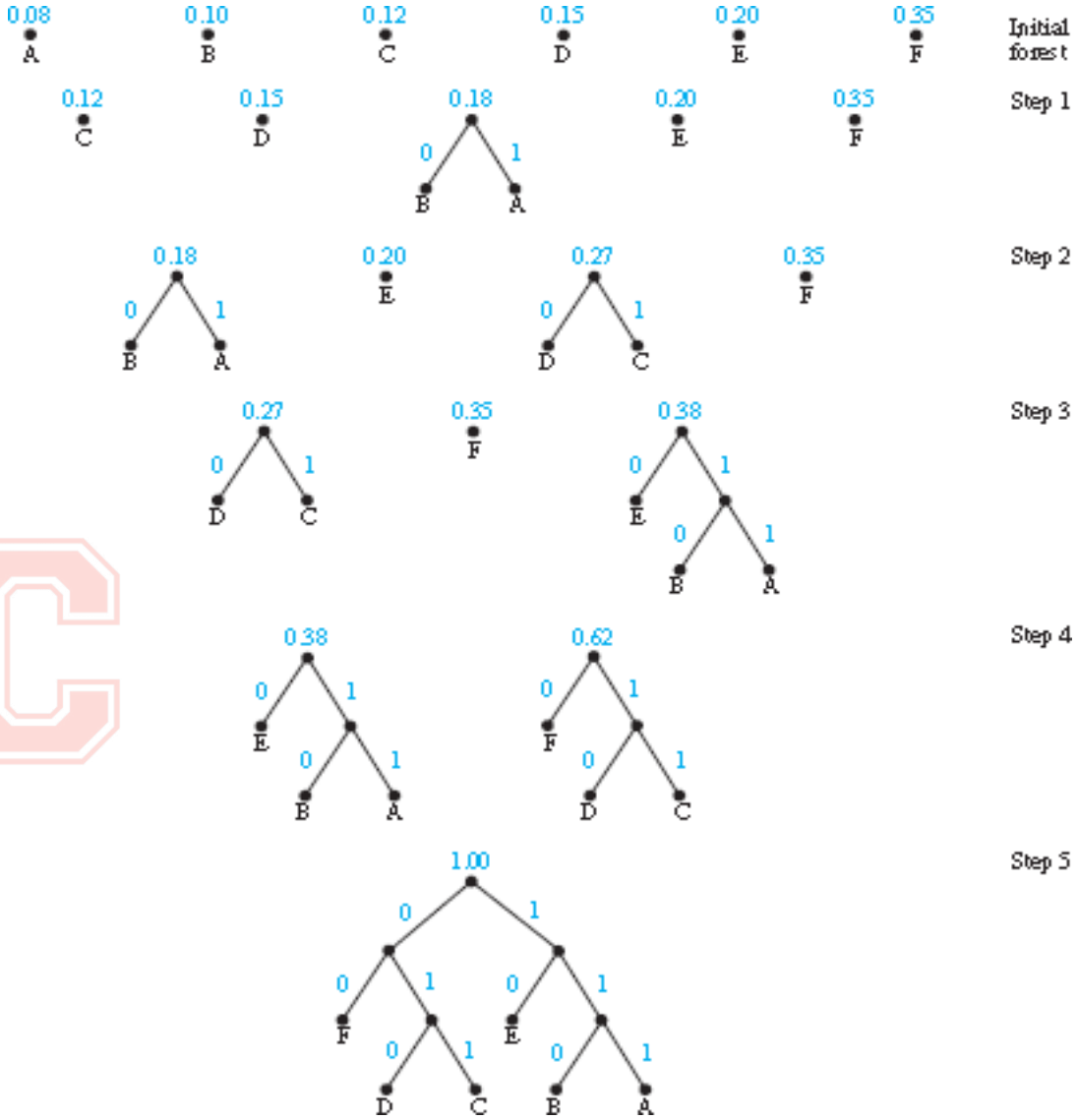


Figure4.27: Huffman coding of given Symbols

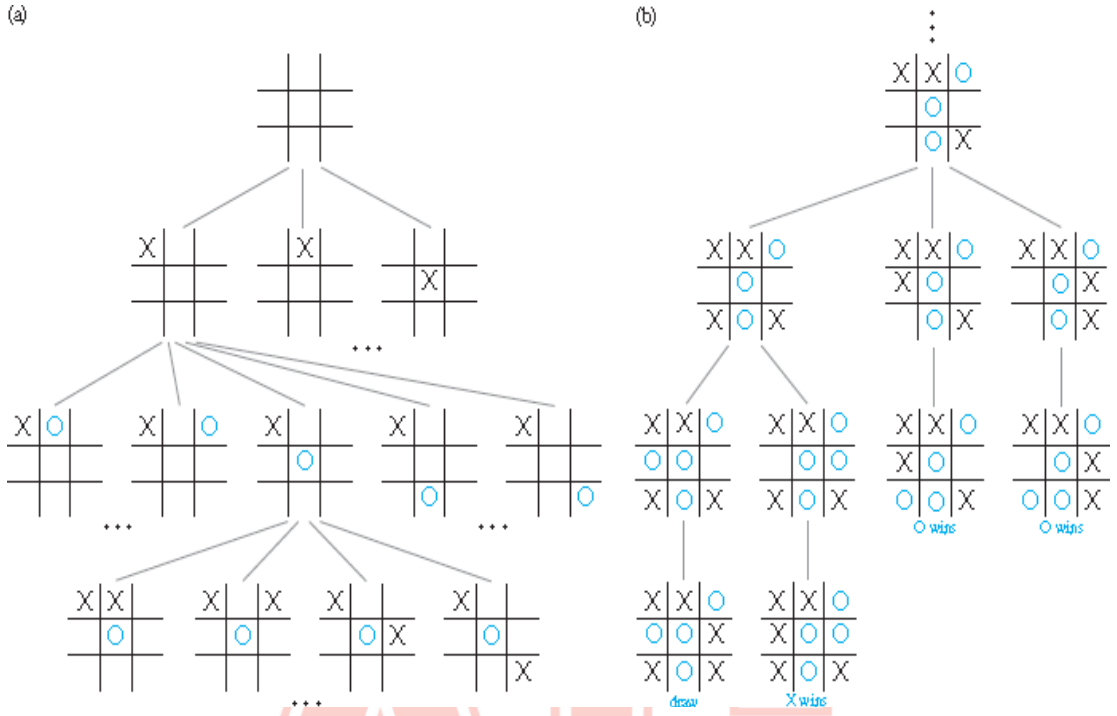
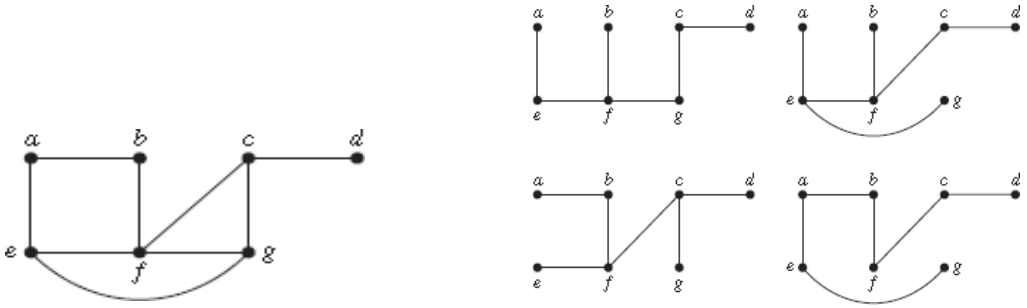


Figure 4.28: Some of the Game Tree for Tic-Tac-Toe

- The value of a vertex in a game tree is defined recursively as:
  - ✓ The value of a leaf is the payoff to the first player when the game terminates in the position represented by this leaf.
  - ✓ The value of an internal vertex at an even level is the maximum of the values of its children, and the value of an internal vertex at an odd level is the minimum of the values of its children.

4.15 Spanning Trees

- **Spanning Tree**: Let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.
- A simple graph is connected if and only if it has a spanning tree.
- Example 1: Find a spanning tree of the simple graph G shown in Figure 4.29.a.
  - ✓ Solution: The graph G is connected, but it is not a tree because it contains simple circuits. Remove the edge {a, e}. This eliminates one simple circuit, and the resulting subgraph is still connected and still contains every vertex of G. Next remove the edge {e, f} to eliminate a second simple circuit. Finally, remove edge {c, g} to produce a simple graph with no simple circuits. This subgraph is a spanning tree, because it is a tree that contains every vertex of G. The sequence of edge removals used to produce the spanning tree is illustrated in Figure 4.30.



(a) **(b)**

Figure 4.29: (a) The Simple Graph G and (b) Spanning Trees of G

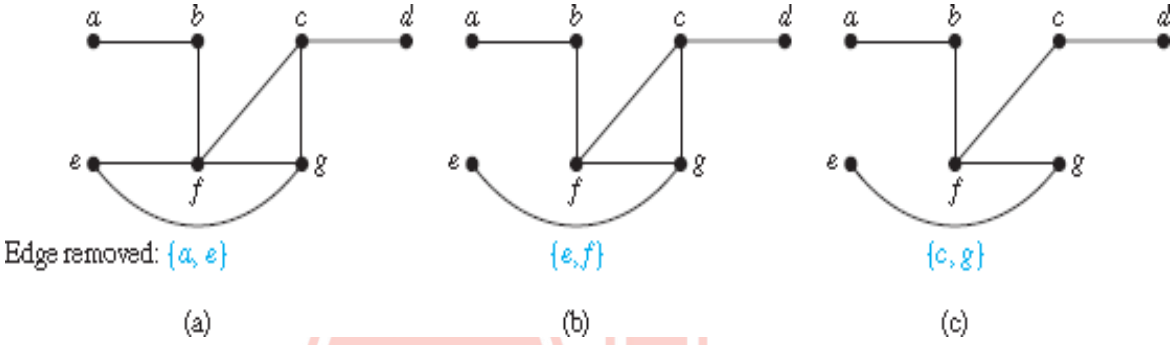


Figure 4.30: Producing a Spanning Tree for G by Removing Edges That Form Simple Circuits

Depth-First Search (DFS)

- **DFS:** First, arbitrarily choose a vertex of the graph as the root. Form a path starting at this vertex by successively adding vertices and edges, where each new edge is incident with the last vertex in the path and a vertex not already in the path.
- Continue adding vertices and edges to this path as long as possible.
- If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree.
- If the path does not go through all vertices, more vertices and edges must be added. Move back to the next to last vertex in the path, and, if possible, form a new path starting at this vertex passing through vertices that were not already visited.
- If this cannot be done, move back another vertex in the path, that is, two vertices back in the path, and try again.
- Repeat this procedure, beginning at the last vertex visited, moving back up the path one vertex at a time, forming new paths that are as long as possible until no more edges can be added.
- Each vertex that ends a path at a stage of the algorithm will be a leaf in the rooted tree, and each vertex where a path is constructed starting at this vertex will be an internal vertex.
- Example 1: Use depth-first search to find a spanning tree for the graph *G* shown in Figure 1.

- ✓ Solution: choose arbitrarily first vertex f. A path is built by successively adding edges incident with vertices not already in the path, as long as this is possible. This produces a path f , g, h, k, j (note that other paths could have been built).
- ✓ Next, backtrack to k. There is no path beginning at k containing vertices not already visited. So we backtrack to h. Form the path h, i. Then backtrack to h, and then to f.
- ✓ From f build the path f, d, e, c, a. Then backtrack to c and form the path c, b. This produces the spanning tree.

- The edges selected by depth-first search of a graph are called **tree edges**. All other edges of the graph must connect a vertex to an ancestor or descendant of this vertex in the tree are called **back edges**. The back edges (e, f) and (f, h) present in figure 1. All edges in figure 2 .e are tree edges.

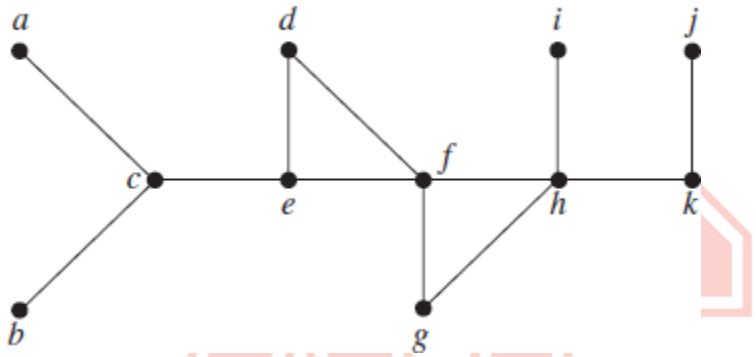


Figure 4.31: The Graph G.

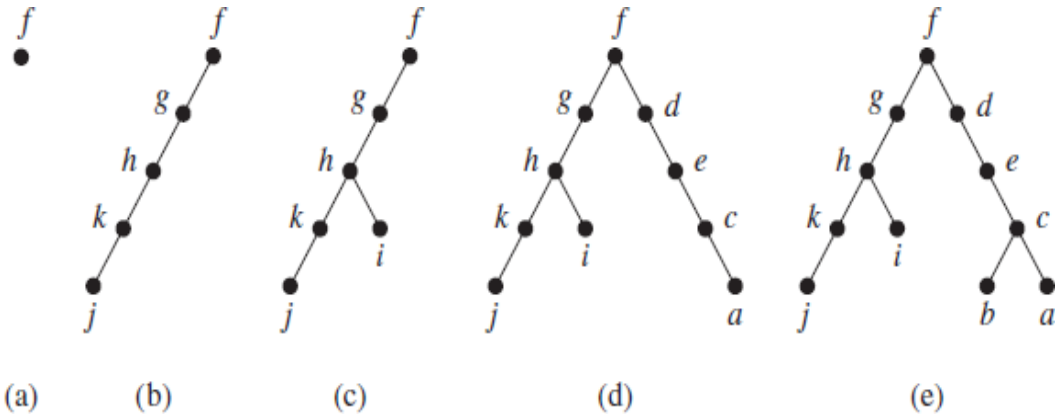


Figure 4.32: Depth-First Search of *G*.

Breadth-First Search (DFS)

- To create the spanning tree using BFS, first arbitrarily choose a root from the vertices of the graph. Then add all edges incident to this vertex. The new vertices added at this stage become the vertices at level 1 in the spanning tree.

- Arbitrarily order them. Next, for each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce a simple circuit.
- Arbitrarily order the children of each vertex at level 1. This produces the vertices at level 2 in the tree.
- Follow the same procedure until all the vertices in the tree have been added. A spanning tree is produced containing every vertex of the graph.

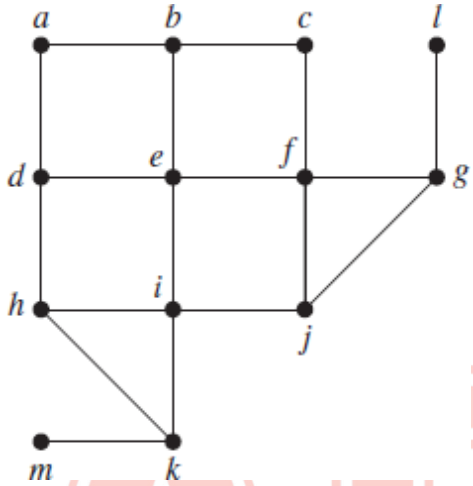


Figure 4.33: A Graph G

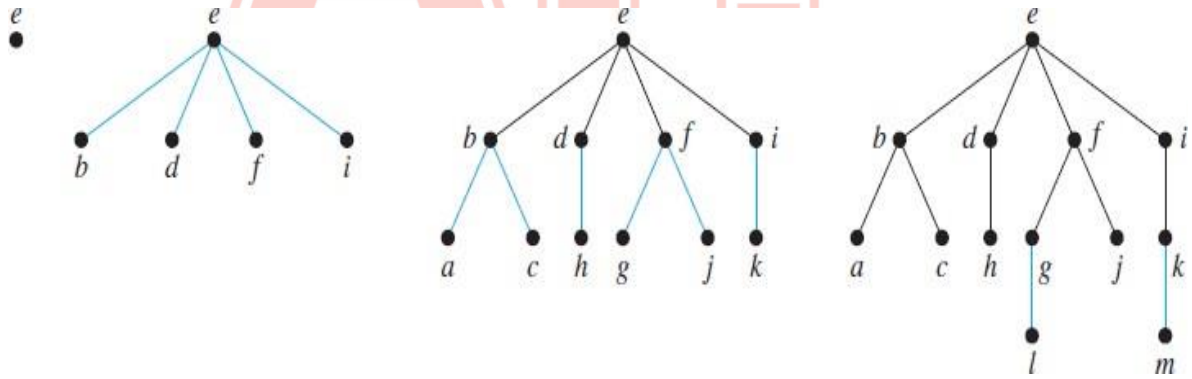


Figure 4.34: Breadth-First Search of G.

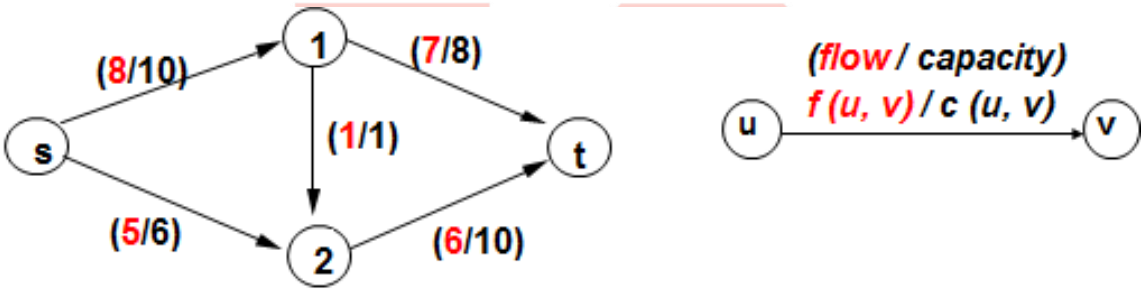
4.16 Network Flow

- Let $G = (V, E)$ be a finite directed graph in which every edge $(u, v) \in E$ has a non-negative, real-valued capacity (u, v). If $(u, v) \notin$ E, we assume that $C\ (u, v) = 0$. We distinguish two vertices: a source (s) and a sink (t).
- A flow in a flow network is real function F**:** V x V → R with the following three properties for all nodes u and v:
  - ✓ Capacity Constraints: F (u, v) ≤ C (u, v), for all u, v. The flow along an edge cannot exceed its capacity.

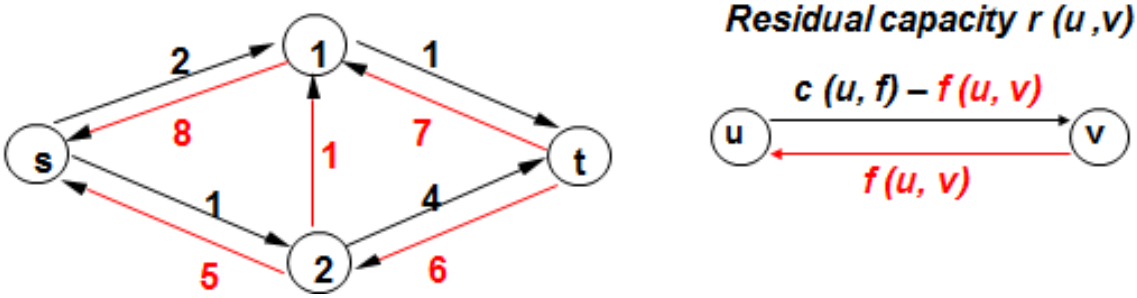
- ✓ Skew Symmetry: F (u, v) = - F (v, u). The net flow from u to v must be the opposite of the net flow from v to u.
- ✓ Flow Conservation: The net flow at a vertex is zero, except for the source, which "produces" flow, and the sink, which "consumes" flow.

$$\sum_{w \in V} f(u,w) = 0, unlsee\ u = s\ or\ u = t$$

- Residual Capacity: It is simply an edge's unused capacity. In other words: The amount of additional flow we can push directly from u to v. Initially none of the capacities will have been used, so all of the residual capacities will be just the original capacity.
- Residual Capacity: r(u, v) = c(u, v) – f (u, v) ≥ 0 (since f (u, v) $\leq$ c(u, v))
- Augmenting Path: Given a flow network G=(V,E) and a flow f, an augmenting path is a simple path from s to t in the residual network G_f.
- The Residual Network:

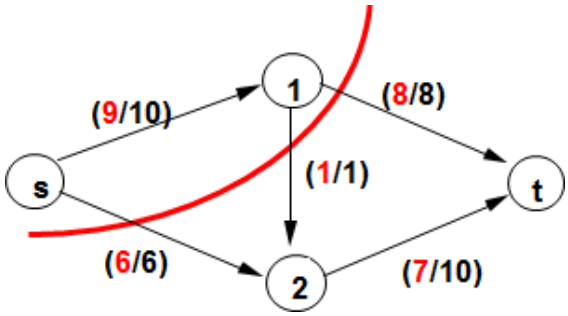


Residual Network



Maxflow and Mincut Theorem

- The cut-set of a cut is the set of edges that begin in S and end in T.
- The capacity of a cut is sum of the weights of the edges beginning in S and ending in T.
- An (S,T)-cut in a flow network G = (V,E) is a partition of vertices V into two disjoint subsets S and T such that s $\in$ S, t $\in$ T e.g., S = { s, 1 } and T = { 2, t }.The capacity of a cut (S,T) is CAP(S,T) = $\sum u \in S \sum v \in T$ c(u,v)



- **Max-flow Min-cut theorem:** For any network, the value of the maximum flow is equal to the capacity of the minimum cut.
- In optimization theory, the max-flow min-cut theorem states that in a flow network, the maximum amount of flow passing from the source to the sink is equal to the minimum capacity that, when removed in a specific way from the network, causes the situation that no flow can pass from the source to the sink.

7.0 Proof by Contradiction

Chapter 4 of the text explains this topic very clearly.

This topic has a huge history of philosophic conflict. Beginning around 1920, a prominent Dutch mathematician by the name of L. E. J. Brouwer claimed that proof by contradiction was sometimes invalid. This claim was part of a larger philsophical project called *intuitionism*, which attempted to describe mathematics as an independent realm about which humans have intuitive knowledge. In opposition to Bertrand Russell and David Hilbert, Brouwer did not believe that logic was a sufficient foundation for mathematics. His writing is not always easy to follow, as the following fragment shows:

> 'In the edifice of mathematical thought ... language plays no other part than that of an efficient, but never infallible or exact, technique for memorizing mathematical constructions and for suggesting them to others; so that the wording of a mathematical theorem has no sense unless it indicates the construction either of an actual mathematical entity or of an incompatibility (e.g. the identity of the empty two-ity with an empty unity) out of some constructional condition imposed on a hypothetical mathematical system. So that language, in particular logic, can never by itself create new mathematical entities nor deduce a mathematical state of things.' (Brouwer, 1954, quoted in an Internet discussion at http://www.cs.nyu.edu/pipermail/fom/1998-September/002129.html)

You can look up informtion about Brouwer's reservations yourself. Most mathematicians ignore them, and that's what we will do.

To prove a statement by contradiction, you show that the negation of the statement is impossible, or leads to a contradiction. Here is an example:

Theorem: there do not exist integers m and n such that $14m + 21n = 100$.

Proof: suppose $14m + 21n = 100$. Since $14m + 21n = 7(2m + 3n)$, we have that 7 divides 100. This is not true, so the hypothesis $14m + 21n = 100$ is not true.

Many theorems have the form $P \Rightarrow Q$. To prove such a theorem by contradiction, you must show that $not(P \Rightarrow Q)$ is impossible. But $not(P \Rightarrow Q)$ is the same as P and not Q. So to use contradiction to prove an implication, you show that the hypothesis joined to the negation of the conclusion leads to an impossibility. Here is an example.

Theorem: If n is an integer and n^2 is odd, then n is odd.

Proof: Suppose n is an integer, n^2 is odd, and n is even. But then n^2 is even, since the product of even numbers is even. Thus n^2 is a number which is both odd and even, which is impossible.

Remark: in certain contexts allowing for infinite numbers, a valid way of showing that a number is infinite is to show that it is both odd and even. For example, the quantity of integers is odd, because we have 0 and all other integers can be divided into pairs of the form $\pm n$. On the other

hand the quantity of integers is even, since then can be divided into pairs $\{2n, 2n + 1\}$. Therefore there is an infinite quantity of integers.

A slight variation of proof by contradition is proof by contrapositive. We know that the implication $P \Rightarrow Q$ is equivalent to $(not(P)) \Rightarrow (not(Q))$. For example:

Theorem: If *n* is an integer and n^2 is odd, then *n* is odd.

Proof: We will show that if *n* is an integer and *n* is even then n^2 is even. This follows because the product of two even numbers is even.

How do you know when to use proof by contradiction or contrapositive? When you cannot find a direct proof:

1. if the statement is an implication, try contrapositive.
2. then try contradiction.
3. when that fails, go back and try harder to find a direct proof.
4. when that fails, try contrapositive and contradiction again.
5. and so forth until you get the answer.

Let's do another example, one of truly historic importance going back to the Pythagorians and the earliest days of Greek mathematics.

Theorem: There is no rational number (fraction) *a* such that $a^2 = 2$. That is: $\sqrt{2}$ is not a fraction.

Proof: Since *a* is a fraction, we can find integers *n* and *m* such that $m \neq 0$, $a = \frac{n}{m}$, and *n* and *m* are not both even. We have $\left(\frac{n}{m}\right)^2 = 2$. Then $n^2 = 2m^2$, so n^2 is even. Thus *n* is even, or $n = 2p$ for some integer *p*. Thus $(2p)^2 = 2m^2$ or $2p^2 = m^2$. Therefore m^2 is even, so *m* is even. Thus *m* and *n* are both even, which contradicts the assumption that *n* and *m* are not both even.

Class prove: $\sqrt{3}$ is not a fraction.

Theorem: suppose *n* is a positive integer. If *n* is odd then *n* + 1 is even.

Proof: Suppose *n* and *n* + 1 are both odd. Since 0 is even and $n > 0$, there is an even integer less than *n*. Thus there is a largest even integer 2*p* such that $2p < n$. Also, since $2(n + 1) > 2n + 1$, there is an even integer larger than *n* + 1. Thus there is a smallest even integer 2*q* such that $n + 1 < 2q$. Thus

$$2p < n < n + 1 < 2q$$

$$2q - 2p > 2$$

$$q - p > 1$$

$$p < p + 1 < q$$

$$2p < 2(p + 1) < 2q$$

Since $2p$ is the largest even integer smaller than n, $n < 2(p + 1)$. Since $2q$ is the smallest even integer greater than $n + 1$, $2(p + 1) < n + 1$. Thus $n < 2(p + 1) < n + 1$, which is impossible.

EXAMPLE 3: Prove that

$$n! \leq n^n \tag{1.3}$$

for any integer $n \geq 1$.

Proof:

STEP 1: For n=1 (1.3) is true, since $1! = 1^1$.

STEP 2: Suppose (1.3) is true for some $n = k \geq 1$, that is $k! \leq k^k$.

STEP 3: Prove that (1.3) is true for $n = k+1$, that is $(k+1)! \overset{?}{\leq} (k+1)^{k+1}$. We have

$$(k+1)! = k! \cdot (k+1) \overset{\text{ST. 2}}{\leq} k^k \cdot (k+1) < (k+1)^k \cdot (k+1) = (k+1)^{k+1}. \blacksquare$$

EXAMPLE 4: Prove that

$$8 \mid 3^{2n} - 1 \tag{1.4}$$

for any integer $n \geq 0$.

Proof:

STEP 1: For n=0 (1.4) is true, since $8 \mid 3^0 - 1$.

STEP 2: Suppose (1.4) is true for some $n = k \geq 0$, that is $8 \mid 3^{2k} - 1$.

STEP 3: Prove that (1.4) is true for $n = k+1$, that is $8 \mid 3^{2(k+1)} - 1$. We have

$$3^{2(k+1)} - 1 = 3^{2k+2} - 1 = 3^{2k} \cdot 9 - 1 = 3^{2k}(8+1) - 1 = \underbrace{3^{2k} \cdot 8}_{\text{div. by } 8} + \underbrace{3^{2k} - 1}_{\substack{\text{St. 2} \\ \text{div. by } 8}}. \blacksquare$$

EXAMPLE 5: Prove that

$$7 \mid n^7 - n \tag{1.5}$$

for any integer $n \geq 1$.

Proof:

STEP 1: For n=1 (1.5) is true, since $7 \mid 1^7 - 1$.

STEP 2: Suppose (1.5) is true for some $n = k \geq 1$, that is

$$7 \mid k^7 - k.$$

STEP 3: Prove that (1.5) is true for $n = k+1$, that is $7 \mid (k+1)^7 - (k+1)$. We have

$$(k+1)^7 - (k+1) = k^7 + 7k^6 + 21k^5 + 35k^4 + 35k^3 + 21k^2 + 7k + 1 - k - 1$$

$$= \underbrace{k^7 - k}_{\substack{\text{St. 2} \\ \text{div. by } 7}} + \underbrace{7k^6 + 21k^5 + 35k^4 + 35k^3 + 21k^2 + 7k}_{\text{div. by } 7}. \blacksquare$$

EXAMPLE 1: Prove that

$$1+2+3+\ldots+n=\frac{n(n+1)}{2} \tag{1.1}$$

for any integer $n \geq 1$.

Proof:

STEP 1: For n=1 (1.1) is true, since

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

STEP 2: Suppose (1.1) is true for some $n=k \geq 1$, that is

$$1+2+3+\ldots+k=\frac{k(k+1)}{2}.$$

STEP 3: Prove that (1.1) is true for $n=k+1$, that is

$$1+2+3+\ldots+k+(k+1) \stackrel{?}{=} \frac{(k+1)(k+2)}{2}.$$

We have

$$1+2+3+\ldots+k+(k+1) \stackrel{\text{ST.2}}{=} \frac{k(k+1)}{2}+(k+1)=(k+1)\left(\frac{k}{2}+1\right)=\frac{(k+1)(k+2)}{2}. \blacksquare$$

EXAMPLE 2: Prove that

$$1+3+5+\ldots+(2n-1)=n^2 \tag{1.2}$$

for any integer $n \geq 1$.

Proof:

STEP 1: For n=1 (1.2) is true, since $1=1^2$.

STEP 2: Suppose (1.2) is true for some $n=k \geq 1$, that is

$$1+3+5+\ldots+(2k-1)=k^2.$$

STEP 3: Prove that (1.2) is true for $n=k+1$, that is

$$1+3+5+\ldots+(2k-1)+(2k+1) \stackrel{?}{=} (k+1)^2.$$

We have: $1+3+5+\ldots+(2k-1)+(2k+1) \stackrel{\text{ST.2}}{=} k^2+(2k+1)=(k+1)^2. \blacksquare$

Question 5. *Show that* $n! > 3^n$ *for* $n \geq 7$.

Solution.

For any $n \geq 7$, let P_n be the statement that $n! > 3^n$.

Base Case. The statement P_7 says that $7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040 > 3^7 = 2187$, which is true.

Inductive Step. Fix $k \geq 7$, and suppose that P_k holds, that is, $k! > 3^k$.

It remains to show that P_{k+1} holds, that is, that $(k+1)! > 3^{k+1}$.

$$\begin{aligned}(k+1)! &= (k+1)k! \\ &> (k+1)3^k \\ &\geq (7+1)3^k \\ &= 8 \times 3^k \\ &> 3 \times 3^k \\ &= 3^{k+1}.\end{aligned}$$

Therefore P_{k+1} holds.

$$\begin{aligned}&= \frac{(k+1)(2k+3)(4k+5)}{3} \\ &= \frac{(2k^2+5k+3)(4k+5)}{3} \\ &= \frac{8k^3+30k^2+37k+15}{3}.\end{aligned}$$

Therefore P_{k+1} holds.

Question 3. *Verify that for all $n \geq 1$, the sum of the squares of the first $2n$ positive integers is given by the formula*

$$1^2 + 2^2 + 3^2 + \cdots + (2n)^2 = \frac{n(2n+1)(4n+1)}{3}$$

Solution.

For any integer $n \geq 1$, let P_n be the statement that

$$1^2 + 2^2 + 3^2 + \cdots + (2n)^2 = \frac{n(2n+1)(4n+1)}{3}.$$

Base Case. The statement P_1 says that

$$1^2 + 2^2 = \frac{(1)(2(1)+1)(4(1)+1)}{3} = \frac{3(5)}{3} = 5,$$

which is true.

Inductive Step. Fix $k \geq 1$, and suppose that P_k holds, that is,

$$1^2 + 2^2 + 3^2 + \cdots + (2k)^2 = \frac{k(2k+1)(4k+1)}{3}.$$

It remains to show that P_{k+1} holds, that is,

$$1^2 + 2^2 + 3^2 + \cdots + (2(k+1))^2 = \frac{(k+1)(2(k+1)+1)(4(k+1)+1)}{3}.$$

$$\begin{aligned}
1^2 + 2^2 + 3^2 + \cdots + (2(k+1))^2 &= 1^2 + 2^2 + 3^2 + \cdots + (2k+2)^2 \\
&= 1^2 + 2^2 + 3^2 + \cdots + (2k)^2 + (2k+1)^2 + (2k+2)^2 \\
&= \frac{k(2k+1)(4k+1)}{3} + (2k+1)^2 + (2k+2)^2 \qquad \text{(by } P_k\text{)} \\
&= \frac{k(2k+1)(4k+1)}{3} + \frac{3(2k+1)^2 + 3(2k+2)^2}{3} \\
&= \frac{k(2k+1)(4k+1) + 3(2k+1)^2 + 3(2k+2)^2}{3} \\
&= \frac{k(8k^2+6k+1) + 3(4k^2+4k+1) + 3(4k^2+8k+4)}{3} \\
&= \frac{(8k^3+6k^2+k) + (12k^2+12k+3) + (12k^2+24k+12)}{3} \\
&= \frac{8k^3+30k^2+37k+15}{3}
\end{aligned}$$

On the other side of P_{k+1},

$$\frac{(k+1)(2(k+1)+1)(4(k+1)+1)}{3} = \frac{(k+1)(2k+2+1)(4k+4+1)}{3}$$

Question 2. *Use the Principle of Mathematical Induction to verify that, for n any positive integer, $6^n -$ is divisible by 5.*

Solution.

For any $n \geq 1$, let P_n be the statement that $6^n - 1$ is divisible by 5.

Base Case. The statement P_1 says that

$$6^1 - 1 = 6 - 1 = 5$$

is divisible by 5, which is true.

Inductive Step. Fix $k \geq 1$, and suppose that P_k holds, that is, $6^k - 1$ is divisible by 5.

It remains to show that P_{k+1} holds, that is, that $6^{k+1} - 1$ is divisible by 5.

$$\begin{aligned} 6^{k+1} - 1 &= 6(6^k) - 1 \\ &= 6(6^k - 1) - 1 + 6 \\ &= 6(6^k - 1) + 5. \end{aligned}$$

By P_k, the first term $6(6^k - 1)$ is divisible by 5, the second term is clearly divisible by 5. Therefore th left hand side is also divisible by 5. Therefore P_{k+1} holds.

Question 1. *Prove using mathematical induction that for all* $n \geq 1$,

$$1 + 4 + 7 + \cdots + (3n - 2) = \frac{n(3n-1)}{2}.$$

Solution.

For any integer $n \geq 1$, let P_n be the statement that

$$1 + 4 + 7 + \cdots + (3n - 2) = \frac{n(3n-1)}{2}.$$

Base Case. The statement P_1 says that

$$1 = \frac{1(3-1)}{2},$$

which is true.

Inductive Step. Fix $k \geq 1$, and suppose that P_k holds, that is,

$$1 + 4 + 7 + \cdots + (3k - 2) = \frac{k(3k-1)}{2}.$$

It remains to show that P_{k+1} holds, that is,

$$1 + 4 + 7 + \cdots + (3(k+1) - 2) = \frac{(k+1)(3(k+1)-1)}{2}.$$

$$\begin{aligned}
1 + 4 + 7 + \cdots + (3(k+1) - 2) &= 1 + 4 + 7 + \cdots + (3(k+1) - 2) \\
&= 1 + 4 + 7 + \cdots + (3k + 1) \\
&= 1 + 4 + 7 + \cdots + (3k - 2) + (3k + 1) \\
&= \frac{k(3k-1)}{2} + (3k + 1) \\
&= \frac{k(3k-1) + 2(3k+1)}{2} \\
&= \frac{3k^2 - k + 6k + 2)}{2} \\
&= \frac{3k^2 + 5k + 2)}{2} \\
&= \frac{(k+1)(3k+2)}{2} \\
&= \frac{(k+1)(3(k+1)-1)}{2}.
\end{aligned}$$

Therefore P_{k+1} holds.

Prove by mathematical induction that $n^3 - n$ is divisible by 3 for all natural numbers n.

Proof

For n = 1,

$$n^3 - n = 1 - 1 = 0$$

which is divisible by 3.

Assume the statement is true for *some* number n, that is, $n^3 - n$ is divisible by 3.
Now,

$$(n+1)^3 - (n+1) = n^3 + 3n^2 + 3n + 1 - n - 1 = (n^3 - n) + 3(n^2 + n)$$

which is $n^3 - n$ plus a multiple of 3.

Since we assumed that $n^3 - n$ was a multiple of 3, it follows that $(n+1)^3 - (n+1)$ is also a multiple of 3.

So, since the statement "$n^3 - n$ is divisible by 3" is true for n = 1, and its truth for n implies its truth for n + 1, the statement is true for all whole numbers n.

1. For every integer $n \in \mathbb{N}$, it follows that $1+2+3+4+\cdots+n = \dfrac{n^2+n}{2}$.
2. For every integer $n \in \mathbb{N}$, it follows that $1^2+2^2+3^2+4^2+\cdots+n^2 = \dfrac{n(n+1)(2n+1)}{6}$.
3. For every integer $n \in \mathbb{N}$, it follows that $1^3+2^3+3^3+4^3+\cdots+n^3 = \dfrac{n^2(n+1)^2}{4}$.
4. If $n \in \mathbb{N}$, then $1\cdot 2+2\cdot 3+3\cdot 4+4\cdot 5+\cdots+n(n+1) = \dfrac{n(n+1)(n+2)}{3}$.
5. If $n \in \mathbb{N}$, then $2^1+2^2+2^3+\cdots+2^n = 2^{n+1}-2$.
6. For every natural number n, it follows that $\sum_{i=1}^{n}(8i-5) = 4n^2 - n$.
7. If $n \in \mathbb{N}$, then $1\cdot 3+2\cdot 4+3\cdot 5+4\cdot 6+\cdots+n(n+2) = \dfrac{n(n+1)(2n+7)}{6}$.
8. If $n \in \mathbb{N}$, then $\dfrac{1}{2!}+\dfrac{2}{3!}+\dfrac{3}{4!}+\cdots+\dfrac{n}{(n+1)!} = 1-\dfrac{1}{(n+1)!}$
9. For any integer $n \geq 0$, it follows that $24 \mid (5^{2n}-1)$.
10. For any integer $n \geq 0$, it follows that $3 \mid (5^{2n}-1)$.
11. For any integer $n \geq 0$, it follows that $3 \mid (n^3+5n+6)$.
12. For any integer $n \geq 0$, it follows that $9 \mid (4^{3n}+8)$.

Proposition If $n \in \mathbb{Z}$ and $n \geq 0$, then $\sum_{i=0}^{n} i \cdot i! = (n+1)! - 1$.

Proof. We will prove this with mathematical induction.

(1) If $n = 0$, this statement is

$$\sum_{i=0}^{0} i \cdot i! = (0+1)! - 1.$$

Since the left-hand side is $0 \cdot 0! = 0$, and the right-hand side is $1! - 1 = 0$, the equation $\sum_{i=0}^{0} i \cdot i! = (0+1)! - 1$ holds, as both sides are zero.

(2) Consider any integer $k \geq 0$. We must show that S_k implies S_{k+1}. That is, we must show that

$$\sum_{i=0}^{k} i \cdot i! = (k+1)! - 1$$

implies

$$\sum_{i=0}^{k+1} i \cdot i! = ((k+1)+1)! - 1.$$

We use direct proof. Suppose $\sum_{i=0}^{k} i \cdot i! = (k+1)! - 1$. Observe that

$$\begin{aligned}
\sum_{i=0}^{k+1} i \cdot i! &= \left(\sum_{i=0}^{k} i \cdot i!\right) + (k+1)(k+1)! \\
&= \big((k+1)! - 1\big) + (k+1)(k+1)! \\
&= (k+1)! + (k+1)(k+1)! - 1 \\
&= \big(1 + (k+1)\big)(k+1)! - 1 \\
&= (k+2)(k+1)! - 1 \\
&= (k+2)! - 1 \\
&= ((k+1)+1)! - 1.
\end{aligned}$$

Therefore $\sum_{i=0}^{k+1} i \cdot i! = ((k+1)+1)! - 1$.

It follows by induction that $\sum_{i=0}^{n} i \cdot i! = (n+1)! - 1$ for every integer $n \geq 0$. ■