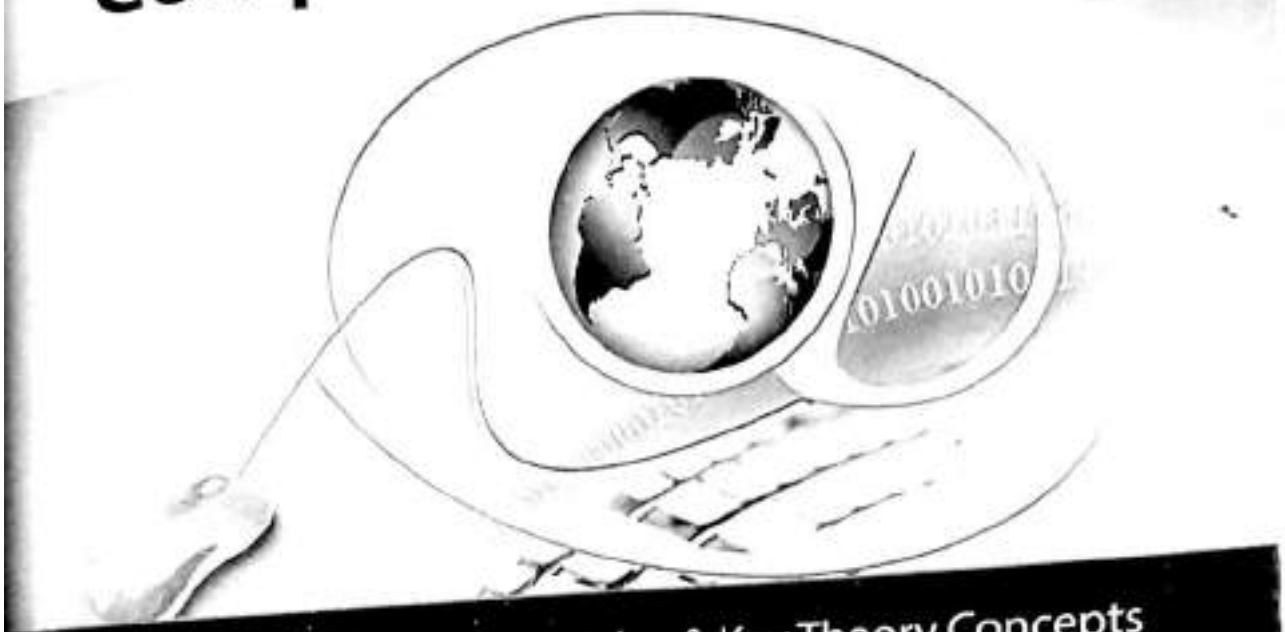
A Handbook on Computer Science & IT



Contains Well Illustrated Formulae & Key Theory Concepts

For

GATE

Easy to learn & Easy to remember

& OTHER COMPETITIVE EXAMS

MADE EASY Publications

Scanned by CamScanner

A Handbook on Computer Science & IT

CONTENTS

Unit-1: Discrete and Engineering Mathematics 07-70

Unit-2: Digital Logic 71-114

Unit-3: Computer Organization and Architecture 115-149

Unit-4: Programming and Data Structures.....150-172

Unit-5: Algorithms.....173-195

Unit-6: Theory of Computation 196-221

Unit-7: Compiler Design.....222-239

Unit-8: Operating System 240-272

Unit-9: Databases.....273-303

Unit-10: Information Systems and Software Engineering 304-320

Unit-11: Computer Networks 321-352

Unit-12: Web Technologies 353-364

oooo

Scanned by CamScanner

A Handbook on
Computer Science

1

Discrete Mathematics & Engineering Mathematics

CONTENTS

1. Mathematical Logic 8
2. Combinatorics 13
3. Set Theory & Algebra 19
4. Graph Theory 38
5. Probability 48
6. Linear Algebra 54
7. Numerical Methods 59
8. Calculus 63

Scanned by CamScanner

Mathematical Logic

1

Introduction

- **Proposition:** It is a declarative statement either TRUE or FALSE.
- **Compound Proposition:** It is a proposition formed using the logical operators (Negation ($\neg$), Conjunction ($\wedge$), Disjunction ($\vee$), etc.) with the existing propositions.
- **Logical Operators:**
  - (*i*) Negation of p : $\neg p$ or $\bar{p}$ or $\sim p$
  - (*ii*) Conjunction of p and q : $p \wedge q$
  - (*iii*) Disjunction of p and q : $p \vee q$
  - (*iv*) Implication/Conditional : $p \rightarrow q$ (if p, then q)
  - (*v*) Bi-conditional : $p \leftrightarrow q$
- Precedence order of logical operators from high to low: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- $P \oplus R = PR' + P'R$, $P \leftrightarrow R = P'R' + PR$
- Number of distinct boolean expression with n variable $= 2^{2^n}$.
- **Normal form:** PCNF ($\vee$) = (POS = 0), PDFL ($\wedge$) = (SOP = 1)
  Total size $= 2^n$ with n variable.

Note:

- Converse of $p \rightarrow q$ is : $q \rightarrow p$
- Inverse of $p \rightarrow q$ is : $\neg p \rightarrow \neg q$
- Contrapositive of $p \rightarrow q$ is : $\neg q \rightarrow \neg p$

Tautology

If compound proposition is always true then it is tautology.

Example: $p \vee \neg p$

Contradiction

If Compound proposition is always false then it is contradiction.

Example: $p \wedge \neg p$

Contingency

Neither tautology nor contradiction.

Example: p

Logical Equivalence

$P \Leftrightarrow Q$ is tautology iff P and Q are logically equivalent.

Scanned by CamScanner

Functionally Complete

If any formula can be written as an equivalent formula containing only the connectives in a set of operators, then such a set of operators is called as functionally complete.

Example:

$\{\uparrow\}, \{\downarrow\}, \{\neg, \vee\}, \{\neg, \wedge\}, \{\neg, \vee, \wedge\}$ are functionally complete (NAND).

Consistent

If $H_1 \wedge H_2 \wedge H_3 \wedge \ldots \wedge H_n$ is satisfiable then $H_1, H_2, \ldots$ and H_n are consistent (Tautology, contingency but not contradiction).

Inconsistent

If $H_1 \wedge H_2 \wedge H_3 \wedge \ldots \wedge H_n$ is unsatisfiable then $H_1, H_2, \ldots$ and H_n are inconsistent (only contradiction).

- **Valid**: tautology, **Satisfiable:** tautology + contingency, Invalid: contradiction + contingency, **Unsatisfiable:** contradiction
- Sufficient ($\rightarrow$), necessary ($\leftarrow$), but = and, if = when = whenever, is = will = would = are = p unless q.
- $p \rightarrow q \equiv q$ unless $\neg p$ = "q is true unless p is false" either p is not true or q is true.
- p is necessary but not sufficient for $q = (q \rightarrow p) \wedge (p \rightarrow q)' = p \wedge q'$.

Equivalences

$$P \vee (P \wedge Q) \equiv P$$

$$P \rightarrow Q \equiv \neg P \vee Q \equiv \neg Q \rightarrow \neg P$$

$$P \wedge (P \vee Q) \equiv P$$

$$P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$$

$$P \leftrightarrow Q \equiv (P \wedge Q) \vee (\neg P \wedge \neg Q)$$

$$P \leftrightarrow Q \equiv \neg P \leftrightarrow \neg Q$$

$$P \rightarrow (Q \rightarrow R) \equiv (P \wedge Q) \rightarrow R$$

$$\neg(P \leftrightarrow Q) \equiv P \leftrightarrow (\sim Q) \equiv (\sim P) \leftrightarrow Q \equiv P \oplus Q$$

$$(P \rightarrow Q) \wedge (P \rightarrow R) \equiv P \rightarrow (Q \wedge R)$$

$$(P \rightarrow R) \wedge (Q \rightarrow R) \equiv (P \vee Q) \rightarrow R$$

$$(P \rightarrow Q) \vee (P \rightarrow R) \equiv P \rightarrow (Q \vee R)$$

$$(P \rightarrow R) \vee (Q \rightarrow R) \equiv (P \wedge Q) \rightarrow R$$

$$P \vee Q \equiv \neg P \rightarrow Q$$

Scanned by CamScanner

$$P \wedge Q \equiv \neg(P \rightarrow \neg Q)$$

$$\neg(P \rightarrow Q) \equiv (P \wedge \neg Q)$$

Identity Laws : (*i*) $P \wedge T = P$, (*ii*) $P \vee F = P$

Domination Laws : (*i*) $P \vee T = T$, (*ii*) $P \wedge F = F$

Idempotent Laws : (*i*) $P \wedge P = P$, (*ii*) $P \vee P = P$

Commutative Laws :

(*i*) $P \vee Q = Q \vee P$

(*ii*) $P \wedge Q = Q \wedge P$

Associative Laws :

(*i*) $(P \vee Q) \vee R = P \vee (Q \vee R)$

(*ii*) $(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$

Distributive Laws :

(*i*) $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$

(*ii*) $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$

Demorgan's Laws :

(*i*) $\neg(P \wedge Q) = \neg P \vee \neg Q$

(*ii*) $\neg(P \vee Q) = \neg P \wedge \neg Q$

Absorption Laws :

(*i*) $P \vee (P \wedge Q) = P$

(*ii*) $P \wedge (P \vee Q) = P$

Negation Laws :

(*i*) $P \vee \neg P = T$

(*ii*) $P \wedge \neg P = F$

Double Negation Laws : $\neg(\neg P) = P$

Rules of Inference (Tautological Implications)

Simplification :

$$(P \wedge Q) \Rightarrow P$$

$$(P \wedge Q) \Rightarrow Q$$

Addition :

$$P \Rightarrow (P \vee Q)$$

$$Q \Rightarrow (P \vee Q)$$

Disjunctive Syllogism : $(\sim P, P \vee Q) \Rightarrow Q$

Modus Ponens : $(P, P \rightarrow Q) \Rightarrow Q$

Modus Tollens : $(\sim Q, P \rightarrow Q) \Rightarrow \sim P$

Hypothetical Syllogism : $(P \rightarrow Q, Q \rightarrow R) \Rightarrow (P \rightarrow R)$

Conjunctive Syllogism : $((P \vee Q), P) \Rightarrow \sim Q$

Dilemma : $(P \vee Q, P \rightarrow R, Q \rightarrow R) \Rightarrow R$

Scanned by CamScanner

Constructive Dilemma : $(P \vee Q, P \to R, Q \to S) \Rightarrow R \vee S$

Destructive Dilemma : $(\sim R \vee \sim S, P \to R, Q \to S) \Rightarrow \sim P \vee \sim Q$

Other rules :

$$\sim P \Rightarrow (P \to Q)$$
$$Q \Rightarrow (P \to Q)$$
$$\sim(P \to Q) \Rightarrow P$$
$$\sim(P \to Q) \Rightarrow \sim Q$$

Exactly one = $\exists!$ or $\exists x\,[P(x) \wedge P(y) \Rightarrow y = x]$, $\forall x[\exists y\,(B(x, y) \wedge (B(x, z) \to y = z)$

$p \Rightarrow q \Rightarrow r \equiv (p \wedge q) \to r = q \Rightarrow (p \Rightarrow r)$

Principle Conjunctive Normal Form (PCNF)

Product of sums (max term)

PCNF: $[P(x_1) \vee P(x_2)] \wedge [P(x_3) \vee P(x_4)]$

Principle Disjunctive Normal Form (PDNF)

Sums of products (min term)

PDNF: $[P(x_1) \wedge P(x_2)] \vee [P(x_3) \wedge P(x_4)]$

Number of non equivalent propositional functions with n-propositional variables are = 2^{2^n}.

- $\forall x\,(\alpha \to \beta) \Rightarrow (\forall x\,\alpha \Rightarrow \forall x\,\beta)$ true only with properties always use and but not $\to$.

Predicate Logic

Quantifiers

- Universal ($\forall$) : "for all" or "for every"
- Existential ($\exists$) : "there exist"

Predicates

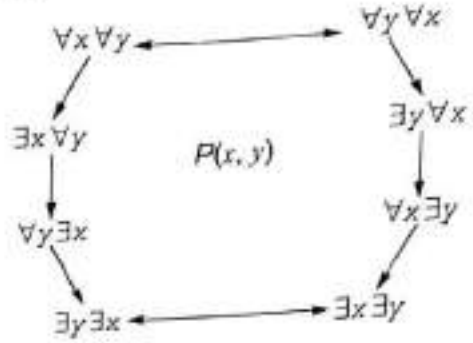
- $P(x)$: Propositional statement with one variable.
- $Q(x, y)$: Propositional statement with two variables.

Note:

- $\neg\exists x\,P(x) = \forall x\,\neg P(x)$
- $\neg\forall x\,P(x) = \exists x\,\neg P(x)$

Scanned by CamScanner

Logical Equivalences



1. $\forall x\,[P(x) \wedge Q(x)] \equiv \forall x\, P(x) \wedge \forall x\, Q(x)$
2. $\exists x\,[P(x) \vee Q(x)] \equiv \exists x\, P(x) \vee \exists x\, Q(x)$
3. $\forall x\,(P(x) \vee Q) \equiv \forall x\, P(x) \vee Q$
4. $\forall x\,(P(x) \wedge Q) \equiv \forall x\, P(x) \wedge Q$
5. $\exists x\,(P(x) \vee Q) \equiv \exists x\, P(x) \vee Q$
6. $\exists x\,(P(x) \wedge Q) \equiv \exists x\, P(x) \wedge Q$
7. $\forall x\, P(x) \wedge \exists y Q(y) \equiv \forall x \exists y [P(x) \wedge Q(y)]$
8. $\forall x\, P(x) \vee \exists y Q(y) \equiv \forall x \exists y [P(x) \vee Q(y)]$

■■■

Scanned by CamScanner

Combinatorics

2

Permutations (Ordered Selection/Arrangement)

- The number of permutations of n-objects :

1. taken 'r' at a time = ${}^nP_r = P(n, r) = (n)_r$ (arrangement).
2. taken 'r' at a time = n^r (with repetition) (arrangement).
3. taken all at a time = $n!$
4. taken not more than 'r' = $\dfrac{(n^{r+1}-1)}{n-1}$ (with repetition).
5. taken 'r' (atleast one repeated) = All – none = $n^r - {}^nP_r$
6. taken all at a time, in which 'r' of them are alike (identical) = $\dfrac{n!}{r!}$.
7. taken all at a time, in which n_1 are alike, n_2 are alike, ..., n_r are alike

   $= \dfrac{n!}{n_1!\,n_2!\ldots n_r!}$
8. taken 'r' at a time, in which m-particular objects are:

   (a) Never included = ${}^{(n-m)}P_r$.

   (b) Always included ($n \geq m$ and $r \geq m$) = ${}^{(n-m)}P_{(r-m)} \cdot {}^rP_m$.
9. ${}^nP_r = P(n-1, r) + r \cdot P(n-1, r-1)$
10. ${}^nP_r = r! \cdot {}^nC_r = n \times {}^{(n-1)}P_{(r-1)}$
11. n types of objects (each of infinity in number)

    $\underbrace{n_1 + n_2 + n_3 + n_4 + \ldots\ldots + n_n}_{n} = r = {}^{n-1+r}C_r$
12. m boys and n girls are arrange in straight line = $(m + n - 1)!$

- **The Number of Circular Permutations of n object :**

1. Taken all at a time = $(n-1)!$; (If not directed)

   $= \dfrac{(n-1)!}{2}$; (If direction (clockwise/anticlockwise) is given)

Scanned by CamScanner

2. Taken 'r' at a time = $\frac{{}^nP_r}{r}$; (If not directed), and $\frac{1}{2}\cdot\frac{{}^nP_r}{r}$; (if directed)
3. Necklace with m identical beads, n identical beads in circular number of way = $\frac{(m+n-1)}{m!\times n!\times 2!}$.

Distinct → Distinct → Ordered partition	Distinct → Indistinct → Unordered	Indistinct → Distinct → Ball in box
$\frac{n!}{n_1!\,n_2!\ldots n_n!}$	$\frac{n!}{(G!)^P}$	$\frac{n!}{(G!)^P\times P!}$

Ordered partition
- Fully specified: $\frac{n!}{n_1!\,n_2!\ldots n_m!}$
- Fully Unspecified: m^n
- Partially specified: Manually

Derangements

- Number of derangements for n-objects = $D_n = n!\sum_{i=0}^{n}\frac{(-1)^i}{i!}$

$$\left(D_n \cong \frac{n!}{e} \text{ for large value of } n\right) = \left[\frac{n!}{2!}-\frac{n!}{3!}+\frac{n!}{4!}\ldots\frac{n!}{n!}\right]$$

$$\left(\text{The prob. of a given permutation being a derangement} = \frac{D_n}{n!} = \sum_{i=0}^{n}\frac{(-1)^i}{i!} \cong \frac{1}{e} \text{ for large } n\right)$$

- **Binomial theorem:**

$$(a+b)^n = {}^nC_0\,a^n\,b^0 + {}^nC_1\,a^{n-1}b + \ldots + {}^nC_n\,a^0b^n = \sum_{r=0}^{n}{}^nC_r\,a^{n-r}b^r$$

- **Multi-nomial coefficients:**

$$\binom{n}{n_1,\ n_2,\ldots,n_r} = \frac{n!}{n_1!\,n_2!\ldots\ldots n_r!}$$

Note: .....

- Number of ways to distribute 'r' apples to n-persons = ${}^{(n+r-1)}C_r$
- The number of permutations of (n_1+n_2) objects, taken all at a time, in which n_1 objects are alike and n_2 objects are alike = $\frac{(n_1+n_2)!}{n_1!\,n_2!}$.

.....

Scanned by CamScanner

Combinations (Unordered Selection)

$$\boxed{{}^nC_r = C(n, r) = \frac{n!}{(n-r)!\,r!}}$$

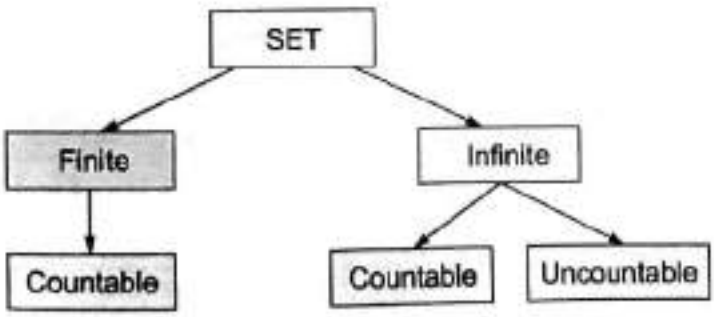
$$\boxed{C(n, r) = C(n-1, r) + C(n-1, r-1)}$$

- The number of combinations of n-objects :
  - (*i*) Taken (selecting) 'r' at a time = ${}^nC_r$
  - (*ii*) Taken (selecting zero or more at a time) = 2^n
  - (*iii*) Taken one or more at a time = $2^n - 1$
  - (*iv*) Where the objects are divided into p-objects of one type, q-objects of second type and so on; A non-empty selection from this can be made in $[(p+1)(q+1).....] - 1$ ways.
  - (*v*) Taken 'r' at a time, in which m-particular objects:
    - (a) Always included = $C(n-m, r-m)$
    - (b) Never included = $C(n-m, r)$
- Number of subsets of a set X containing n-elements

  $= 2^n = \left\{{}^nC_0 + {}^nC_1 + ... + {}^nC_n\right\}$
- Number of diagonals for a regular polygon with n-sides

  $= {}^nC_2 - n = \dfrac{n^2 - 3n}{2} \left\{\begin{array}{l} {}^nC_2 = \text{Number of diagonals and sides both,} \\ n = \text{Number of sides only = number of points} \end{array}\right\}$
- Number of triangles formed by vertices of a polygon with n-sides = ${}^nC_3$.

Counting



- Union of two countable sets in countable.
- Union of countable number of countable sets is countable.

Scanned by CamScanner

- The set of real numbers that are solutions to quadratic equations $ax^2 + bx + c = 0$, where a, b and c are integers is '**countable**'.
- The set of lines through the origin is '**uncountable**'.

Counting Principle [Inclusion-Exclusion]

- $n(A \cup B) = n(A) + n(B) - n(A \cap B)$
- $n(A \cup B \cup C) = n(A) + n(B) + n(C) - n(A \cap B) - n(A \cap C) - n(B \cap C) + n(A \cap B \cap C)$
- Let $A_1, A_2, \ldots, A_n$ are finite sets. $n(A_1 \cup A_2 \cup \ldots \cup A_n) = \sum_{1 \le i \le n} n(A_i) - \sum_{1 \le i < j \le n} n(A_i \cap A_j) + \sum_{1 \le i < j < k \le n} n(A_i \cap A_j \cap A_k) - \ldots + (-1)^{n-1} n(A_1 \cap A_2 \ldots \cap A_n)$
- If A and B are disjoint, $n(A \cup B) = n(A) + n(B)$.
- The set of functions "from" the positive integers to $\{0,1,2,3,4,5,6,7,8,9\}$ is '**uncountable**' since the power set of positive integers is uncountable.

Pigeonhole Principle

If n pigeons occupy m pigeon holes, then at least $\left\lfloor \frac{n-1}{m} \right\rfloor + 1 = \left\lceil \frac{n}{m} \right\rceil$ pigeons share the same pigeon hole.

Fundamental Principle of Counting

If some event can occur in 'n_1' different ways, and if following this event a second event occur in 'n_2' different ways,... then the number of ways the event can occur in the order $= n_1 \cdot n_2 \cdot n_3 \ldots$

Summation

- $\Sigma n = \frac{n(n+1)}{2}$
- $\Sigma n^2 = \frac{n(n+1)(2n+1)}{6}$
- $\Sigma n^3 = \frac{n^2(n+1)^2}{4} = [\Sigma n]^2$
- $\sum_{k=0}^{n} a \cdot r^k = \frac{a[r^{n+1} - 1]}{r - 1}; \quad r \ne 1$
- $S_n = \sum_{k=1}^{n} f(k) = S_{n-1} + f(n)$
- $\sum_{k=n_1}^{n_2} f(k) = f(n_1) + f(n_1+1) + f(n_1+2) + \ldots + f(n_2)$; for $n_1 \le n_2$

Scanned by CamScanner

- $\sum_{k=1}^{n}[f(k)+g(k)] = \sum_{k=1}^{n} f(k) + \sum_{k=1}^{n} g(k)$

Generating Functions

- $G(x) = \sum_{r=0}^{\infty} a_r x^r = a_0 + a_1 x + a_2 x^2 + \ldots + a_r x^r + \ldots$
- ${}^nC_r = {}^{n-1}C_r + {}^{n-1}C_{r-1}$
- ${}^nC_0{}^2 + {}^nC_1{}^2 + {}^nC_2{}^2 \ldots\ldots + {}^nC_n{}^2 = {}^{2n}C_n$
- $(1+ax)^n = \sum_{r=0}^{n} {}^nC_r \cdot a^r \cdot x^r = \sum_{r=0}^{n} {}^nC_r (ax)^r$
- $(1+x^k)^n = \sum_{r=0}^{n} {}^nC_r \cdot x^{kr}$
- $\frac{1-x^{n+1}}{1-x} = \sum_{r=0}^{n} x^r$
- $e^x = \sum_{r=0}^{\infty} \frac{x^r}{r!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots$
- $\log(1+x) = \sum_{r=0}^{\infty} \frac{(-1)^{r+1} \cdot x^r}{r!} = -1 + \frac{x}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!} \ldots$
- $\frac{1}{1-ax} = \sum_{r=0}^{\infty} (ax)^r = \sum_{r=0}^{\infty} a^r \cdot x^r$
- $\frac{1}{1-x^n} = \sum_{r=0}^{\infty} (x^n)^r = \sum_{r=0}^{\infty} x^{nr}$
- $\frac{1}{(1-ax)^n} = \sum_{r=0}^{\infty} {}^{n+r-1}C_r \cdot (ax)^r$
- $\frac{1}{(1+ax)^n} = \sum_{r=0}^{\infty} {}^{n+r-1}C_r \cdot (-1)^r \cdot (ax)^r$
- $\sin x = \sum_{r=0}^{\infty} (-1)^r \frac{x^{2r+1}}{(2r+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots^{\infty}$
- $\sinh x = \sum_{r=0}^{\infty} \frac{x^{2r+1}}{(2r+1)!} = x + \frac{x^3}{3!} + \frac{x^5}{4!} + \ldots^{\infty}$

Scanned by CamScanner

- $\cos x = \sum_{r=0}^{\infty} (-1)^r \cdot \frac{x^{2r}}{(2r)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots\infty$
- $\cosh x = \sum_{r=0}^{\infty} \frac{x^{2r}}{(2r)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots\infty$

Recurrence Relation

$$\boxed{a_n = a_n^H + a_n^P}$$

where a_n^H is complementary solution and a_n^P is a particular solution.

- Complementary function $[a_n^H]$ (Solution to homogenous part)

Root	a_n^H
1. $t_1, t_2, t_3, \ldots t_k$ are distinct real numbers	$C_1 \cdot t_1^n + C_2 t_2^n + \ldots + C_k t_k^n$
2. $t_1 = t_2 = t_3$	$(C_1 + C_2 n + C_3 n^2)$
3. Pair of roots are complex $(\alpha \pm i\beta)$	$r^n[C_1 \cos n\theta + C_2 \sin n\theta]$ where $r = \sqrt{\alpha^2 + \beta^2}$, $\theta = \tan^{-1}\frac{\beta}{\alpha}$

- Particular function $[a_n^P]$:

R.H.S	a_n^P
1. Constant	d_0
2. $C_0 + C_2 n$	$d_0 + d_1 n$
3. $C_0 + C_1 n + C_2 n^2$	$d_0 + d_1 n + d_2 n^2$
4. a^n	$d_0 a^n$
5. $n \cdot a^n$	$(d_0 + d_1 n) a^n$

d_1, d_2 and d_3 values are obtained by substituting a_n^P in given reccurrence relation.

■■■

Scanned by CamScanner

Set Theory & Algebra

3

Set

A set is an unordered collection of objects.

The objects in a set are called the elements, or members of the set.

- $\mathbb{N}$ be set of natural numbers : {1, 2, 3, ...}
- $\mathbb{Z}$ be set of integers : {..., –2, –1, 0, 1, 2,...}
- $\mathbb{Q}$ be set of rational numbers
- $\mathbb{R}$ be set of real numbers
- $\mathbb{C}$ be set of complex numbers

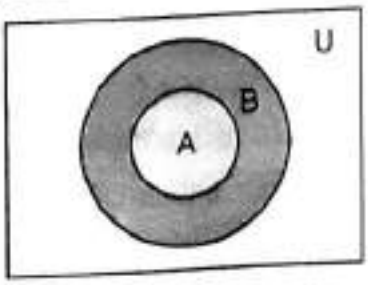
$$\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R} \subseteq \mathbb{C}$$

Types of Set

1. **Universal set U:** A set which contains all objects under consideration (the universal set varies depending on which objects are of interest)
2. **Equal sets:** Two sets are equal iff they have the same elements i.e., if A and B are sets, then A and B are equal iff $\forall x\ (x \in A \leftrightarrow x \in B)$; denoted by $A = B$.
3. **Equivalent sets:** Two sets A and B are equivalent if they have same number of elements. i.e. $|A| = |B|$.
4. **Empty set or Null set:** A special set that has no elements. Null set can be denoted by ϕ or { }.

   Example: The set of all positive integers that are greater than their squares is the null set.
5. **Singleton set:** A set with one element is called a singleton set.
6. **Subset:** The set A is said to be a subset of B iff every element of A is also an element of B. $A \subseteq B$ indicates that A is a subset of the set B.

   i.e. $A \subseteq B$ iff $x \in A \Rightarrow x \in B$



Venn Diagram Showing that A is a subset of B

Scanned by CamScanner

Note:

- For every set S : $\phi \subseteq S$ and $S \subseteq S$

Comparable: If $A \subseteq B$ or $B \subseteq A$ then A and B are comparable.

7. **Proper subset:** A set A is a subset of the set B but also $A \neq B$, we write $A \subset B$ and say that A is a proper subset of B i.e. A is a proper subset of B iff $\forall x\,(x \in A \rightarrow x \in B) \wedge \exists x\,(x \in B \wedge x \notin A)$
8. **Finite set:** A set in which number of elements are finite and hence countable i.e., cardinality of set can be obtained as a number.
9. **Infinite set:** A set is said to be infinite if it is not finite.
   Example: The set of positive integer is infinite.
10. **Power Set:** The power set of a set S is the set of all subsets of the set S. The power set of S is denoted by $\mathcal{P}(S)$.

Note:

- If a set has n-elements, then its power set has 2^n elements.
- *Example:* Power set of the set $\{0, 1, 2\}$ is
  $\mathcal{P}(\{0, 1, 2\}) = \{\phi, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\}$

Cartesian Product of Sets

Let A and B be sets. The cartesian product of A and B, denoted by $A \times B$, is the set of all ordered pairs (a, b), where $a \in A$ and $b \in B$.

Hence, $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$

Note:

- If $|A| = m$ and $|B| = n$ then $|A \times B| = mn$
- In general $A \times B \neq B \times A$ but $A \times B = B \times A$ iff $A = B$ or $A = \phi$ or $B = \phi$.
  However, $|A \times B| = |B \times A|$ always.

Set Operations

1. **Union:** The union of the sets A and B, denoted by $A \cup B$, is the set that contains those elements that are either in A or in B, or in both.

$$A \cup B = \{x \mid x \in A \vee x \in B\}$$

Example: The union of the sets $\{1, 3, 5\}$ and $\{1, 2, 3\}$ is : $\{1, 2, 3, 5\}$

Scanned by CamScanner

Remember:

- $\text{Max}(|A|, |B|) \le |A \cup B| \le (|A| + |B|)$
- $|A \cup B| = |A| + |B| - |A \cap B|$

2. **Intersection:** The intersection of the sets A and B, denoted by $A \cap B$, is the set containing those elements in both A and B.

$$A \cap B = \{x \mid x \in A \wedge x \in B\}$$

3. **Disjoint:** Two sets are called disjoint if their intersection is the empty set.
4. **Difference:** The difference of A and B, denoted by $A - B$, is the set containing those elements that are in A but not in B.

   The difference of A and B is also called the complement of B with respect to A or the relative complement of B in A.

$$A - B = \{x \mid x \in A \wedge x \notin B\}$$

Inclusion Exclusion Principle

$$n(A_1 \cup A_2 \cup \ldots \cup A_n) = \sum_{1 \le i \le n} n(A_i) - \sum_{1 \le i < j \le n} n(A_i \cap A_j) + \sum_{1 \le i < j < k \le n} n(A_i \cap A_j \cap A_k) - \ldots + (-1)^{n-1} n(A_1 \cap A_2 \cap \ldots \cap A_n)$$

Properties of Sets

Let A, B and C are sets, U is universal set and ϕ is an empty set.

Identity	Name
$A \cup B = B \cup A$ $A \cap B = B \cap A$	Commutative Laws
$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$	Associative Laws
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	Distributive Laws
$A \cup \phi = A$ $A \cap U = A$	Identity Laws
$A \cup \bar{A} = U$ $A \cap \bar{A} = \phi$	Complement Laws
$A \cup A = A$ $A \cap A = A$	Idempotent Laws
$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$	Absorption Laws
$(\bar{\bar{A}}) = A$	Double Complement Law
$A \cup U = U$ $A \cap \phi = \phi$	Domination Laws
$\overline{A \cup B} = \bar{A} \cap \bar{B}$ $\overline{A \cap B} = \bar{A} \cup \bar{B}$	De Morgans Laws

Scanned by CamScanner

Multiset

A collection of objects in which an element can appear more than once is called a multiset.

Example: $\{a, a, b, b, b, c, c, c, c, c, d\} = \{2 \cdot a, 3 \cdot b, 5 \cdot c, 1 \cdot d\}$

Let $A = \{m_1 \cdot a_1, m_2 \cdot a_2, \ldots\ldots, m_k \cdot a_k\}$ where m_i = multiplicity of a_i

$B = \{n_1 \cdot a_1, n_2 \cdot a_2, \ldots\ldots, n_k \cdot a_k\}$ where n_i = multiplicity of a_i

Then $A \cup B$ = a multiset, in which multiplicity of a_i is max $\{m_i, n_i\}$

$A \cap B$ = a multiset, in which multiplicity of a_i is min $\{m_i, n_i\}$

$A + B$ = a multiset, in which multiplicity of a_i is $(m_i + n_i)$

$A - B$ = a multiset, in which multiplicity of

$$a_i = \begin{cases} m_i - n_i & \text{if} \quad m_i > n_i \\ 0 & , \quad \text{otherwise} \end{cases}$$

Functions

Definition

Let A and B be nonempty sets. A function f from A to B is an assignment of exactly one element of B to each element of A.

We write $f(a) = b$ if b is the unique element of B assigned by the function f to the element a of A. If f is a function from A to B, we write $f : A \to B$.

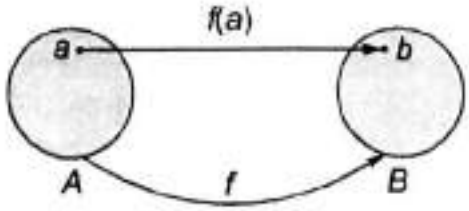
Note:

- Functions are sometimes also called mappings or transformations.

Domain and Codomain

If f is a function from A to B, we say that A is the domain of f and B is the codomain of f.

If $f(a) = b$, we say that "b is the image of a" and "a is the preimage of b".



The function f maps A to B.

- If number of elements $|A| = m$ and $|B| = n$, then number of functions possible from A to $B = n^m$.

Scanned by CamScanner

- A function $f: A \to A$ is called a function on the set A.
  If $|A| = n$ then number of functions possible on $A = n^n$.

Types of Functions

1. **One-to-one function (Injection):** A function f is said to be one-to-one, or injective, iff $f(a) = f(b)$ implies that $a = b$ for all a and b in the domain of f.
   - If A and B are finite sets then a one-to-one from A to B is possible iff $|A| \le |B|$.
   - If $|A| = m$ and $|B| = n$ then $(m \le n)$ then number of one-to-one function from A to B is $P(n, m) = n(n-1)(n-2)\ldots\ldots(n-(m-1))$.
   - If $|A| = |B| = n$ then number of one-to-one functions from A to B is $P(n, n) = n(n-1)(n-2)\ldots. 1 = n!$

2. **Onto Function (Surjection):** A function f from A to B is called onto, or surjective, iff for every element $b \in B$ there is an element $a \in A$ with $f(a) = b$
   i.e. range(f) = co-domain(f) = B.
   - If A and B are finite sets then an onto function from A to B is possible only when $|B| \le |A|$.
   - If $|A| = |B|$ then every one-to-one function from $A \to B$ is onto and vice-versa.
   - If $|A| = |B| = n$ then number of onto functions possible from A to B $= n!$
   - If $|A| = m$ and $|B| = n\,(n < m)$ then number of onto functions from $A \to B = n^m - {}^nC_1(n-1)^m + {}^nC_2(n-2)^m \ldots + (-1)^{n-1}\,{}^nC_{n-1}(1^m)$.

3. **Bijection:** A function which is one-to-one and onto is called a bijection. If A, B are finite sets, then a bijection from A to B is possible only when $|A| = |B|$.
   - If $|A| = |B|$ then number of bijections = number of one-to-one = number of onto possible from A to $B = n!$

4. **Inverse Function:** Let f be a one-to-one correspondence from the set A to the set B. The inverse function of f is the function that assigns to an element b belonging to B the unique element a in A such that $f(a) = b$. The inverse function of f is denoted by f^{-1}. Hence, $f^{-1}(b) = a$ when $f(a) = b$. Inverse of function f exists iff f is a bijection.

5. **Identity function:** Identity function on A is denoted by I_A. Inverse of identity function is the function itself. Every identity function is bijection, if $f(a) = a$; $\forall a \in A$.

Scanned by CamScanner

6. **Constant function:** A function $f : A \to B$ is said to be constant function if $f(x) = c; \forall x \in A$ i.e., all the elements of domain are mapped to only one element of codomain. Therefore the range of constant function contains only one element.

Function Composition

Let f and g are two functions defined on set A:

$(f \circ g) : A \to A$ defined by $(f \circ g)x = f(g(x))$

$(g \circ f) : A \to A$ defined by $(g \circ f)x = g(f(x))$

Note:

- In general $(f \circ g)x \neq (g \circ f)x$
- Let $f : A \to B$ and $g : B \to C$ then $(g \circ f) : A \to C$
  but $(f \circ g)$ may not be defined
  $(f \circ g)$ is defined if range of $g(x)$ is a subset of A.
- If $f : A \to A$ is a bijection then
  $f \circ f^{-1} = f^{-1} \circ f = I$ where I is identity function on A.
- If $f : A \to B$ is a bijection then $f \circ f^{-1} = I_B$, $f^{-1} \circ f = I_A$, $f^{-1} : B \to A$
- If $f : A \to B$ and $g : B \to C$ are injective (one-one) then $g \circ f : A \to C$ is also injective.
- If $f : A \to B$ and $g : B \to C$ are surjective (onto) then $g \circ f : A \to C$ is also surjective.
- If $f : A \to B$ and $g : B \to C$ are functions, and $g \circ f : A \to C$ is injective then f is also injective.
- If $f : A \to B$ and $g : B \to C$ are functions, and $g \circ f : A \to C$ is surjective then g is also surjective (onto)

Relation

Definition

Let A and B be two sets. Then a binary relation from A to B is a subset of $A \times B$.

Relations on a Set

A relation on the set A is a relation from $A \times A$ i.e., a relation on a set A is a subset of $A \times A$.

- If $|A| = m$ and $|B| = n$ then number of relations possible on $A = 2^{mn}$.
- If $|A| = n$ and $|B| = n$ then number of relations possible on $A = 2^{(n^2)}$.

Scanned by CamScanner

Types of Relation

1. **Inverse Relation:** Let R be a relation from a set A to B. The inverse of R, denoted by R^{-1} is the relation from B to A which consists of those ordered pairs, which when reversed belongs to R i.e.,
$R^{-1} = \{(b, a) \mid (a, b) \in R\}$

2. **Complementary Relation:** If R is a relation from A to B then
$R^c = \bar{R} = \{(a, b) \mid (a, b) \notin R\} = (A \times B) - R$.

3. **Diagonal Relation:** A relation R on a set A is called diagonal relation or identity relation if $R = \{(a, a) \mid a \in A\} = \Delta_A$.

4. **Reflexive Relation:** A relation R on a set A is said to be reflexive if aRa $\forall a \in A$ i.e. $(a, a) \in R$, $\forall a \in A$.
   - If $|A| = n$ then number of reflexive relations possible on $A = 2^{n(n-1)}$.
   - A diagonal relation on a set A is reflexive and any superset of diagonal relation is also reflexive.
   - Smallest reflexive relation on $A = \Delta_A$ (diagonal relation)
   - Largest reflexive relation on $A = A \times A$.

5. **Irreflexive Relation:** A relation R on a set A is said to be irreflexive, if $a \not R a$ i.e., $(a, a) \notin R$, $\forall a \in A$.
   - If $|A| = n$ then number of irreflexive relations possible on $A = 2^{n(n-1)}$.
   - Smallest irreflexive relation on $A = \phi$
   - Largest irreflexive relation on $A = (A \times A) - \Delta_A$.

6. **Symmetric Relation:** A relation R on a set A, is said to be symmetric if aRb then bRa, $\forall a, b \in A$.
   - If $|A| = n$ then number of symmetric relations possible on
   $$A = 2^n \times 2^{\frac{n^2-n}{2}} = 2^{n(n+1)/2}.$$
   - Number of symmetric relations possible with only diagonal pairs $= 2^n$.
   - Number of symmetric relations possible with only non-diagonal pairs $= 2^{(n^2-n)/2}$.
   - Smallest symmetric relation on $A = \phi$
   - Largest symmetric relation on $A = A \times A$.

7. **Antisymmetric Relation:** A relation R on a set A is said to be antisymmetric, if aRb and bRa then $a = b$, $\forall a, b \in A$.

Scanned by CamScanner

- Smallest antisymmetric relation is ϕ.
- Largest antisymmetric relation on A is not unique. Number of elements in largest antisymmetric relation includes all diagonal pairs and half of non-diagonal pairs.
  i.e., $n + (n^2 - n)/2$ elements $= (n^2 + n)/2$ elements.
- Any subset of antisymmetric relation is also antisymmetric relation.
- If $A = \{1, 2, \ldots, n\}$ then number of antisymmetric relations possible on $A = 2^n \times 3^{n(n-1)/2}$.
  With n diagonal pairs, 2^n choices.
  With $\frac{n(n-1)}{2}$ non-diagonal pairs. $3^{n(n-1)/2}$ choices.
- A relation R is antisymmetric iff $R \cap R^{-1} \subseteq \Delta_A$.

8. **Asymmetric Relation:** A relation R on a set A is called asymmetric, if $(b, a) \notin R$, whenever $(a, b) \in R$, $\forall a, b \in A$.
   - Relation R is asymmetric iff it is both antisymmetric and irreflexive.
   - If $A = \{1, 2, \ldots, n\}$ then number of asymmetric relations $= 3^{n(n-1)/2}$.

Note:
- Number of reflexive and symmetric relations with n-elements $= 2^{n(n-1)/2}$.
- Number of neither reflexive nor irreflexive relations $= 2^{n^2} - 2.2^{n(n-1)}$.
- ϕ is not reflexive [empty relation]

9. **Partial Ordering Relation:** A relation R on a set A is partial ordered if R is reflexive, antisymmetric and transitive.
   Poset: A set A with a partial ordered relation R defined on A is called a poset. Poset is partially ordered set.
   Totally ordered set: A poset $[A; R]$ is totally ordered set, if every pair of elements in A are comparable i.e., either aRb or bRa $\forall a, b \in A$.

Note:
- **A relation R on a set A is:**
  (*i*) Symmetric $\Leftrightarrow R = R^{-1}$
  (*ii*) Antisymmetric $\Leftrightarrow (R \cap R^{-1}) \subseteq \Delta_A$
  (*iii*) Reflexive $\Leftrightarrow R^{-1}$ is also reflexive
  (*iv*) Reflexive $\Leftrightarrow R^C$ or $\bar{R}$ is irreflexive

Scanned by CamScanner

- If R and S are antisymmetric, then $(R \cap S)$ is also antisymmetric for any relation S on A.
- If R is antisymmetric then every subset of R is also antisymmetric.
- If R is relation on a set A then $R \cup R^{-1}$ is always symmetric and $R \cup \Delta$ is always reflexive.
- **If R and S on set A are any two:**
  - (*i*) Reflexive relations then $(R \cup S)$ and $(R \cap S)$ are also reflexive.
  - (*ii*) Symmetric relations then $(R \cup S)$ and $(R \cap S)$ are also symmetric
  - (*iii*) Antisymmetric relations the $(R \cap S)$ is always antisymmetric
  - (*iv*) Transitive relations then $(R \cap S)$ is always transitive.
  - (*v*) Equivalence relations then $(R \cap S)$ is always equivalence relation.

Closures of Relations

1. **Transitive Closure** : Transitive closure of $R = R^*$ = smallest transitive relation on set A which contains R.

   Example: Let $A = \{1, 2, 3\}$ and $R = \{(1, 2), (2, 3)\}$.

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

2. **Reflexive Closure** : Reflexive closure of $R = R^+$ = smallest reflexive relation on set A which contains $R = (R \cup \Delta_A)$.

   Example: Let $A = \{1, 2, 3\}$ and $R = \{(1, 2), (2, 3)\}$.

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

3. **Symmetric Closure** : Symmetric closure of $R = R^\#$ = smallest symmetric relation on set A which contains $R = (R \cup R^{-1})$

   Example: Let $A = \{1, 2, 3\}$ and $R = \{(1, 2), (2, 3)\}$.

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

Partition of a Set

Let A be a set with 'n' elements dividing the set A into subsets $\{A_1, A_2, \ldots, A_n\}$ is called partition of A, if

(*i*) every subset is a non-empty set and

(*ii*) $\forall_{i,j}\ A_i \cap A_j = \phi;\ (i \neq j)$ and

(*iii*) $(A_1 \cup A_2 \cup A_3 \cup \ldots \cup A_n) = A$.

Example: Let $A = \{1, 2, 3\}$. Then there are 5 partitions possible on A.

$P_1 = \{\{1, 2, 3\}\}$, $P_2 = \{\{1\}, \{2, 3\}\}$, $P_3 = \{\{2\}, \{1, 3\}\}$, $P_4 = \{\{3\}, \{1, 2\}\}$, and $P_5 = \{\{1\}, \{2\}, \{3\}\}$

Scanned by CamScanner

Groups

Closure

Binary operator $*$ is said to be closed on a non-empty set A, if $a * b \in A$ for all $a, b \in A$

$$\boxed{\text{Number of binary operations on set } 'G' = |G|^{|G| \times |G|}}$$

Associativity

$(a*b)*c = a*(b*c); \quad \forall a,b,c \in G$

Identity

$\exists e \in G \forall a \in G \quad (a*e = e*a = a)$ where 'e' is identity

Note:

- If there exist an identity element in G then it must be unique.

Inverse

$\forall a \in G \exists b \in G \quad (a*b = b*a = e)$.

This also means $a^{-1} = b$ and $b^{-1} = a$

Commutative

$\forall a, b \in G \quad (a*b = b*a)$

Groupoid

An algebraic system $(G, *)$ is groupoid if it is closed operation on G.

Semigroup

An algebraic system which is groupoid and associative.

Monoid

An algebraic system $(G, *)$ which is semigroup and there is an identity in G.

Group

An algebraic system $(G, *)$ which is monoid and every element in G has inverse.

Abelian Group

An algebraic system $(G, *)$ which is a group and it is also commutative i.e., $a*b = b*a; \forall a, b \in G$.

Scanned by CamScanner

Order of an Element of a Group

1. Let G be a group and let $g \in G$ be an element of G. Then the order of g is the smallest positive number k, such that $ak = e$.
2. Let G be a finite group and let $g \in G$. Then the order of g divides the order of G.
3. Any group of even order contains an element of order two.

Cyclic Group

- Let $G = \langle a \rangle$ be a cyclic group $G = \{a^i \mid i \in \mathbb{Z}\}$.
- Let G be a group. We say that G is cyclic if it is generated by one element.
- Let G be a cyclic group, generated by a. Then
  1. G is abelian
  2. If G is infinite, the elements of G are precisely
     $\ldots a^{-3}, a^{-2}, a^{-1}, e, a, a^2, a^3, \ldots$
  3. If G is finite, of order n, then the elements of G are precisely
     $e, a, a^2, \ldots, an^{-2}, an^{-1}$ and $an = e$.
  4. Let G be a group of prime order. Then G is cyclic.
  5. A finite group is cyclic iff there exists an element $g \in G$ whose order is same as the order of the group. Also such an element g will be the generator of that cyclic group.
  6. Let G be a finite cyclic group of order n, say $G = \langle g \rangle$. For every positive integer $d \mid n$ there is exactly one subgroup of G of order d. These are all the subgroups of G.
  7. Let G be a cyclic group. Every subgroup of G is cyclic.

Subgroup

- H is a subgroup of G iff
  (*i*) H is subset of G ($H \subseteq G$)
  (*ii*) Closure: $ab \in H$ for $a, b \in H$.
  (*iii*) Identity: The identity element of G is contained in H.
  (*iv*) Inverse: For all $a \in H$ we have $a^{-1} \in H$.
- Let G be a group and let H_i, $i \in I$ be a collection of subgroups of G. Then the intersection

$$H = \bigcap_{i \in I} H_i \text{, is a subgroup of } G.$$

Scanned by CamScanner

- **Lagrange's theorem:** If H is a subgroup of a finite group G, then $|H|$ divides $|G|$.

Coset

- **Left Coset:** Let G be a group H is subgroup of G. A right H-coset in G is a set of the form $aH := \{ah \mid h \in H\}$.
- **Right Coset:** Let G be a group H is subgroup of G. A right H-coset in G is a set of the form $Ha := \{ha \mid h \in H\}$.
- The number of distinct right cosets (equivalently left cosets) of G is called the index of H in G and is denoted $[G : H]$.
- A left coset of a subgroup $H < G$ is is a subset of G of the form $gH = (gh : h \in H)$.
- Two left cosets are either equal or disjoint; $gH = g'H \Leftrightarrow g^{-1}g' \in H$
- A right coset of H in H is a subset of the form $Hg = (hg : h \in H)$. Two right cosets are either equal or disjoint; we have $Hg = Hg' \Leftrightarrow g^{-1}g' \in H$
- A coset is a left or right coset. Any element of a coset is called a representative of that coset.
- If H is finite, all cosets have cardinality $|H|$.
- There are equal number of left and right cosets in group G.

Group Theory Classification

Groupoid	Semigroup	Monoid	Group	Abelian
Closure	closure + Associative	closure + Associative + Identity	closure + Associative + Identity + Inverse	Group + Commutative
Example:	*Example:*	*Example:*	*Example:*	*Example:*
$(N, +, *)$ $(Z, +, -, *)$ $(R, +, -)$ $(R - \{0\}, *, /)$	$(N, +, *)$ $(Z, +, *)$ $(R, +)$ $(R - \{0\}, *)$ [– and + are always not associative]	$(N, *)$ $(Z, +, \times)$ $(\{0, 1\}, \times)$ $(\{a, b\}^*, +)$	Non-singular matrices closed under '*' (multiplication)	$(\{0, 1, 2, 3\}, +_4)$ $(Z, +)$ $(R, +)$ $(R - \{0\}, *)$ $(Q, +)$ $(Q - \{0\}, *)$ $(\{1, -1, i, -i\}, *)$ $(\{1, \omega, \omega^2\}, *)$ $(\{1, -1\}, *)$

Note: .....

- $O(G) \le 5$ is always "Abelian group".
- Order of a group is equal to the number of elements in the group
- Every group of prime order is a cyclic group and every cyclic group is an Abelian group.

Scanned by CamScanner

- $(\{0, 1, 2, \ldots, m-1\}, +_m)$ Addition modulo is Abelian group.
- If G is a finite group, and $g \in G$, then $g^{|G|} = e$, and $|g|$ always divides $|G|$ (where $|g|$ means order of element g).
- $(\{1, 2, 3, \ldots, q-1\}, \times_q)$ Multiplication modulo is Abelian group.
- If $O(G) = 2n$, then there exist atleast one element other than identity element which is "Self Invertible".
- Set of all non-singular matrices is a group under matrix multiplication, but not abelian.
- The set {0, 1, 2 . . . m – 1} with $\oplus_m$ is always a group, where $\oplus_m$ is also called addition modulo m define as follows:

  $$a \oplus_m b = r\left(\frac{a+b}{m}\right);$$

  Identity of this group is $e = 0$
- The set {1, 2 . . . p – 1} with $\otimes_p$ is always a group, where $\otimes_p$ is also called multiplication modulo p defined as follows:

  $$a \otimes_p b = r\left(\frac{a \times b}{p}\right);$$

  Identity of this group is $e = 1$
- Order of an element $O(a) = n$ and $O(a) = O(a^{-1})$ where $a \in G$ and $a^n = e$.

Lattice Theory

- **First (Least) Element:** Let A be an ordered set, the element 'a' in 'A' is first element of A if for every element 'x' in A, $a \le x$.
- **Last (Greatest) Element:** Let A be an ordered set. The element 'b' in 'A' is last element of A if for every element 'x' in A, $x \le b$.

  Example:

  1. Let N be the set of natural numbers, then first element of $N = 1$ and there is no last element.
  2. Let 'A' be any set and let $\mathcal{P}(A)$ be the power set of A. Then first element of $\mathcal{P}(A) = \phi$ and last element of $\mathcal{P}(A) = A$
- **Minimal Element:** Elements which do not have predecessors.
- **Maximal Element:** Elements which do not have successors.

Scanned by CamScanner

Note:

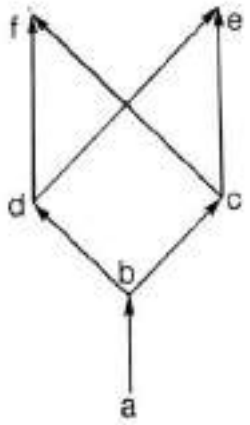
- Many minimals and maximals may exist.

- **Least Element:** 'a' is a least element of poset P; if $a \le x$; $\forall x \in P$.
- **Greatest Element:** 'b' is a greatest element of P; if $x \le b$; $\forall x \in P$.
- **Lower Bound:** Let $A \subseteq P$. Element a is a lower bound of A, if $a \in P$ and $a \le x, \forall x \in A$.
- **Upper Bound:** Let $A \subseteq P$. Element b is an upper bound of A, if $b \in P$ and $x \le b, \forall x \in A$.
- **Greatest Lower Bound [Infimum]:** 'y' is infimum of A, if y is a lower bound of 'A' and if 'z' is any other lower bound of A then $z \le y, \forall z \in P$.
- **Least Upper Bound [Supremum]:** 'x' is supremum of A, if x is an upper bound of 'A' and if 'z' is any other upper bound, then $x \le z, \forall z \in P$.

Note:

- If only one minimal exist then it is always "least"
- If only one maximal exist then it is always "greatest"
- Immediate successors of lower bound are called "atoms"

Example: Consider the following hasse diagram for a poset P. Given P = {a, b, c, d, e, f}. Let S = {b, c, d} and $S \subseteq P$.



Then

Lower bound of S = b, a

Upper bound of S = e, f

Infimum = b

No supremum element exist.

Minimal = a

Maximal = e, f

Least = a

No greatest element exist.

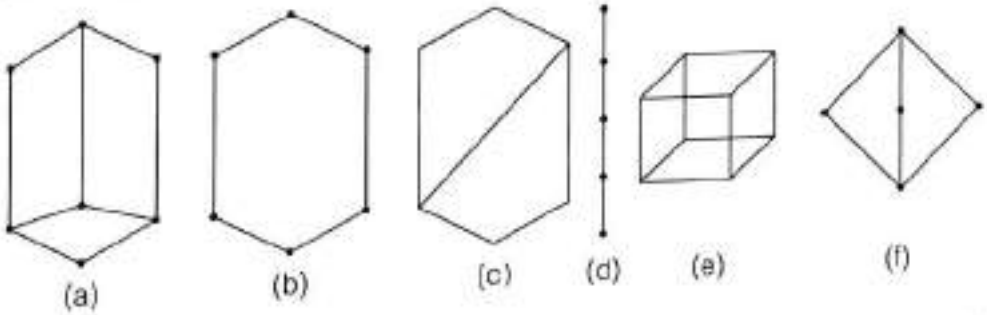
Lattice

- Let (P, ≤) is a poset, in which for every two elements there exist infimum or greatest lower bound or meet (∧) and supremum or least upper bound or Join (∨) then such poset is called a "lattice".

Scanned by CamScanner

OR

- Let 'L' be a non-empty set closed under two binary operations called meet ($\wedge$) and join ($\vee$), then 'L' is a "lattice" if for any element a, b and c of 'L' the following axioms hold.
  1. **Commutative Laws:**
     (*i*) $a \wedge b = b \wedge a$
     (*ii*) $a \vee b = b \vee a$
  2. **Associative Laws:**
     (*i*) $(a \wedge b) \wedge c = a \wedge (b \wedge c)$
     (*ii*) $(a \vee b) \vee c = a \vee (b \vee c)$
  3. **Absorption Laws:**
     (*i*) $a \wedge (a \vee b) = a$
     (*ii*) $a \vee (a \wedge b) = a$
- Following hasse diagrams are "lattices".



Note:
- Every chain is a lattice (i.e., linearly ordered set is a lattice).
- Let 'L' be a lattice, then $a \wedge b = a$ iff $a \vee b = b$.
- $x \wedge y$ = infimum (x, y) and $x \vee y$ = supremum (x, y).
- In lattice every 2-element subset has infimum and supremum.

Types of Lattices

Bounded Lattice

If there exist least element (0) and greatest element (1) for a lattice, such lattice is called "Bounded Lattice" i.e., if $0 \in L$ and $1 \in L$ then $0 \le x \le 1$, $\forall x \in L$.

- $(L, \le, \wedge, \vee)$ and $(P(S), \subseteq, \cap, \cup)$ are bounded lattices.
- $(N, \le, \text{Min}, \text{Max})$ and $(N, /, \text{gcd}, lcm)$ are not bounded.
- Every finite lattice is "Bounded Lattice".

Scanned by CamScanner

Complemented Lattice

In a bounded lattice, if there exist atleast one complement for every element then such a bounded lattice is "complemented lattice".

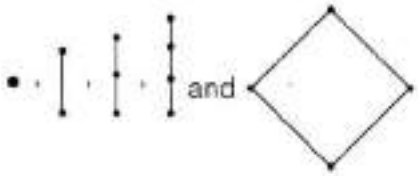
- If $x \vee y = 1$ and $x \wedge y = 0$, then x and y are complements to each other.
- Every element of complemented lattice can contain one or more complements.

Distributive Lattice

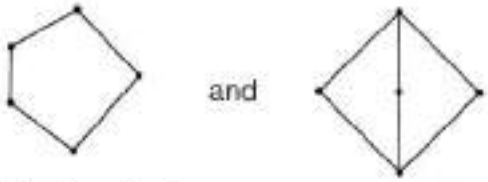
A distributive lattice 'L' satisfies:

$$\left.\begin{array}{ll} (i) & a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \\ (ii) & a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \end{array}\right\} \forall\, a, b, c \in L$$

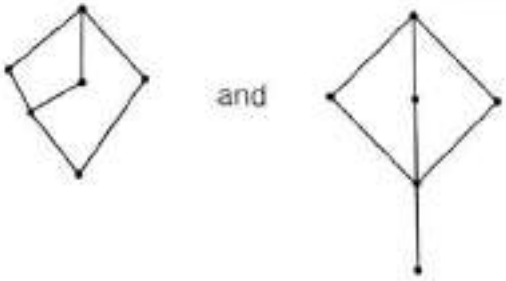
- If a distributive lattice is complemented, then every element has a unique complement.
- A lattice with less than 5-elements is always 'distributive'.


and are distributive lattices

- A lattice 'L' is non-distributive iff it contains a sublattice isomorphic to the following lattices:


and [Non-distributive Lattices]

- The following lattices are non-distributive lattices since they contain a sublattice isomorphic to one of the above lattice


and

Modular Lattice

A modular lattice 'L' satisfies: $a \vee (b \wedge c) = (a \vee b) \wedge c$; $\forall a, b, c \in L$ and $a \leq c$.

Note:

- Every distributive lattice is modular

Scanned by CamScanner

Sublattice

A lattice 'L' is called "sublattice", if has the same meet ($\wedge$) and same join ($\vee$) as the parent lattice.

Example: (D_{12}, $/$, gcd, lcm) is sublattice of (N, $/$, gcd, lcm), where '$/$' represents the 'divides' relation.

Dual Poset

If (P, $\leq$) is poset then (P, $\geq$) is also a poset, such posets are called "dual posets".

Dual Lattice

If (L, $\leq$, $\wedge$, $\vee$) is a lattice, (L, $\geq$, $\vee$, $\wedge$) is also a lattice, such lattices are called "Dual Lattices".

Complete Lattice

A lattice 'L' is said to be complete if every subset of 'L' has infimum and supremum in L.

Lexicographical Order (Dictionary Order)

Let A_1 and A_2 be partial ordered sets, the lexicographical ordering ($\leq$) on $A_1 \times A_2$ is defined as:

$(a_1, a_2) < (b_1, b_2)$; either "if $a_1 < b_1$" or "both $a_1 = b_1$ and $a_2 \leq b_2$".

Well-ordered Set

An ordered set 'A' is well-ordered if it is a chain (linearly ordered) and if it is a discrete set and every subset of 'A' contains "first element" (least element).

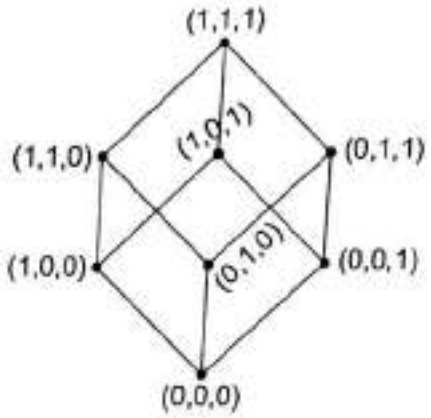
- Every "Finite Linearly Ordered Set" is well-ordered.
- Every well-ordered set must be linearly ordered (chain).

Boolean Algebra (B, $\leq$, $\wedge$, $\vee$)

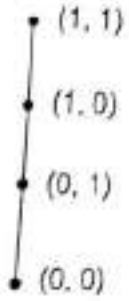
- If a lattice is complemented and distributive, it is boolean algebra.

  Example: (P(S), $\subseteq$, $\cap$, $\cup$)

Scanned by CamScanner



Boolean algebra
(complemented & distributive)



Not boolean algebra
(not complemented)

- Boolean algebra satisfies: "Lattice [Poset, meet, join], Bounded [lower, upper], distributed and complemented lattices".
- Let B be a finite boolean algebra having n-atoms. Then B has 2^n elements and "every non-zero element of B is the sum of unique set of atoms".

  Example: B is boolean algebra with less than 100 elements, then B can have 2^1, 2^2, 2^3, 2^4, 2^5 or 2^6 elements.

- Let a, b, c be any elements in a boolean algebra 'B' $(B, +, *, ', 0, 1)$

1. **Commutative Laws:**

$$a + b = b + a$$
$$a * b = b * a$$

2. **Associative Laws:**

$$(a + b) + c = a + (b + c)$$
$$(a * b) * c = a * (b * c)$$

3. **Distributive Laws:**

$$a + (b * c) = (a + b) * (a + c)$$
$$a * (b + c) = (a * b) + (a * c)$$

4. **Identity Laws:**

$$a + 0 = a$$
$$a * 1 = a$$

5. **Complement Laws:**

$$a + a' = 1$$
$$a * a' = 0$$

6. **Idempotent Laws:**

$$a + a = a$$
$$a * a = a$$

Scanned by CamScanner

7. Absorption Laws:

$$a + (a * b) = a$$
$$a * (a + b) = a$$

8. Involution Law or Double Complement Law:

$$[(a')' = a]$$

$$\begin{matrix} 0' = 1 \\ 1' = 0 \end{matrix} \Rightarrow (0')' = 0$$

9. DeMorgan's Law

$$(a + b)' = a' * b'$$
$$(a * b)' = a' + b'$$

10. Domination Law:

$$a + 1 = 1$$
$$a * 0 = 0$$

■■■

Scanned by CamScanner

Graph Theory

4

Introduction

Graph

A graph G is defined by $G = (V, E)$ where V is set of all vertices in G and E is set of all edges in G.

- **Null Graph:** A graph with no edges is called null graph.
- **Directed Graph:** In a digraph an edge (u, v) is said to be from u to v.
- **Undirected Graph:** In a undirected graph an edge $\{u, v\}$ is said to join u and v or to be between u and v.
- **Isolated Vertex:** A vertex with degree zero is called as Isolated vertex or lone vertex.
- **Pendent Vertex:** A vertex with degree one is called as Pendent vertex.
- **Pendent Edge:** It is an edge which incident with pendent vertex.
- **Path:** It is the sequence of edges, without vertex repetition.
- **Network:** It is a graph with only one source and one sink.
- **Trial (Tour or Walk):** It is the sequence of edges without edge repetition (vertex may repeat).
- **Independence Number:** Number of vertices in largest maximal independent set.
- **Diameter of a Graph:** Maximum distance between any two vertices in a graph.
- **Loop:** An edge drawn from a vertex to itself.
- **Trivial Graph:** A graph with one vertex and no edges.
- **Discrete Graph or Null Graph:** A graph with only isolated vertices and no edges.
- **Pseudo Graph:** A graph in which self loops are allowed as well as parallel or multiple edges are allowed.
- **Simple Graph:** A graph with no loops and no parallel edges is called a simple graph.
- **Peterson graph** number of edges = 15, number of vertex = 10

Scanned by CamScanner

- Girth = size of shortest cycle
- **Hand Shaking Theorem:** Indegree = Outdegree
- $\underset{\text{min degree}}{\delta} \le \left\lfloor \frac{2e}{n} \right\rfloor \le \underset{\text{max degree}}{\Delta}$
- **Complete Bipartite Graph:** (m, n)
  Diameter = 2, Chromatic = 2,
  Number of vertex = $m + n$, Number of edges = $m \times n$

Note:

- Maximum number of edges possible in a simple graph with n-vertices = $n(n-1)/2 = {}^nC_2$.
- Complete graph has $n!$ Hamiltonian cycle.
- Cycle graph has $(n-1)!$ Hamiltonian cycle.
- Number of edges disjoint Hamiltonian cycle = $\frac{n-1}{2}$ i.e. for even no edge disjoint Hamiltonian cycle.
- Number of simple graphs possible with n labelled vertices $= 2^{n(n-1)/2}$.
- **Hand Shaking Theorem:** Let $G = (V, E)$ be a non-directed graph with $V = \{V_1, V_2, \ldots V_n\}$. Then $\sum_{i=1}^{n} \deg(V_i) = 2|E|$.
- In any graph the number of vertices with odd degree is always even.
- If degree of each vertex is k then such a graph is called k-regular graph and in such a graph $|E| = \frac{k|V|}{2} = \frac{nk}{2}$ (where $|V| = n$).
- If degree of each vertex is atleast k i.e. the minimum degree = k, then $|E| \ge \frac{k|V|}{2}$.

- **Regular Graph:** A graph in which all vertices have same degree. If degree of each vertex is 'k' then it is "**k-regular Graph**".
- **Complete Graph:** A simple graph with "**n-mutually adjacent vertices**" is complete graph, represented by K_n.

Scanned by CamScanner

Note:

- Each vertex in K_n has degree $n-1$
- Number of edges in $K_n = |E(K_n)| = n(n-1)/2$
- Every complete graph is regular but converse need not be true.

- **Cycle Graph:** A simple graph with n-vertices ($n \geq 3$) and n-edges is called a cycle graph if all the edges form a cycle of length n. 2-chromatic = even cycle, 3-chromatic = odd cycle.
- **Wheel Graph:** A wheel graph (w_n) with n-vertices ($n \geq 4$) is a simple graph obtained from c_{n-1}, by adding a new vertex which is adjacent to all vertices of c_{n-1}. Diameter = 2, Hamiltonian graph, 3 color = n = even, 4 color = n = odd, $n + 1$ vertex.

  Number of edges = $|E(w_n)| = 2n$
- **Cut Vertex (Articulation Point):** A vertex whose removal makes the graph disconnected.
- **Cut Edge (Bridge):** An edge whose removal makes the graph disconnected.
- **Cut Set:** It is a set of edges, whose removal from graph makes the graph disconnected, provided no proper subset of these edges disconnects the graph.
- ***n* cube:** Number of vertex = 2^n, number of edges = $n \times 2^{n-1}$. 2 colors needed.

Note:

- One or more cut sets may exist in the connected graph.
- Whenever a cut edge exists, cut vertex also exist in graph because atleast one vertex of the cut edge is a cut vertex but the vice versa is not true.
- In a connected graph G an edge of graph G is a cut edge iff the edge is not part of any cycle in G.

Edge Connectivity

Number of edges in a smallest cutset of G is called edge connectivity of G. It is also the minimum number of edges whose removal disconnects the graph.

Biconnected Graph

A graph with no cut vertices and no bridges.

Scanned by CamScanner

Weakly Connected

A digraph is weakly connected if the underlying undirected graph (obtained by removing all the arrows in directed graph) is connected.

Vertex Connectivity

Minimum number of vertices whose removal results in a disconnected graph or reduces it to a trivial graph.

k-connected Graph

On removal of k-vertices, the connected graph becomes disconnected.

k-line-connected Graph

On removal of k-edges, the connected graph becomes disconnected.

Non-separable Graph

Graph with no cut vertices and hence no cut edges (bridges).

Strongly Connected

In digraph, if a path exist between any vertex to any other vertex i.e. for two given vertices u and v $\exists$ a path from u to v as well as from v to u.

Unilaterally Connected

In digraph, if for every two vertices u and v there is a path from u to v or from v to u (not necessarily both)

Weak Graph

Some vertex has indegree but not out degree so vertex not reach to each other.

Note: .....

- $\underset{\text{Vertex connect}}{K(G)} \le \underset{\text{Edge connect}}{\lambda(G)} \le \underset{\text{Min degree}}{\delta} \le \left\lfloor \frac{2e}{n} \right\rfloor \le \underset{\text{Max degree}}{\Delta}$
- A simple graph G with n-vertices is necessarily connected if $|E| > \frac{(n-1)(n-2)}{2}$
- A simple graph with n-vertices and k component has atleast $(n-k)$ edges i.e., $|E| \ge (n-k)$.

Scanned by CamScanner

- A simple graph G with n vertices, k components has atmost $[(n-k)(n-k+1)]/2$ edges.

Tree

A connected graph with no cycle is called a tree.

Spanning Tree

It is a tree and subgraph to a graph 'G' which includes all vertices of 'G'.

- A tree with 'n' vertices has $n-1$ edges.
- Number of binary trees with n-vertices $= \frac{{}^{2n}C_n}{n+1} = \frac{2n!}{n!(n+1)!}$
- k-trees (forest) with total n-vertices have $(n-k)$ edges.

  Number of spanning trees for $k_n = n^{n-2}$.
- "Number of edges that must be removed" from connected graph with n-vertices and e-edges to produce a spanning tree is called 'circuit rank of graph'; Circuit Rank or nullity or cyclomatic complexity $= e-(n-1)$ edges.
- Number of edges that must be removed to produce a spanning forest of graph with 'n' vertices, 'e' edges and 'k' components $= e-(n-k)$ edges.
- Number of bridges possible, for spanning tree with n-edges $= n-1$.
- A finite tree (with atleast one edge) has atleast two vertices of degree '1'.
- The number of different trees with vertex set $\{v_1, v_2, \ldots, v_n\}$ in which the vertices have degree $d_1, d_2, \ldots, d_n$ respectively

$$= \frac{(n-2)!}{(d_1-1)!(d_2-1)!\ldots(d_n-1)!}.$$

Bipartite Graph

In bipartite graph, Vertex set V of a graph is divided into two vertex sets V_1 and V_2, such that $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \phi$.

- Number of edges with 'n' vertices cannot exceed $\left\lfloor \frac{n^2}{4} \right\rfloor$.
- It is either acyclic or contains only even length cycles.

Complete Bipartite Graph

A bipartite graph $G = (V, E)$ with vertex partition $V = \{V_1, V_2\}$ is complete bipartite graph, if every vertex in V_1 is adjacent to every vertex in V_2.

Scanned by CamScanner

- A complete bipartite graph ($K_{m,n}$) has $(m+n)$ vertices and mn edges.
- A complete bipartite graph $K_{m,m}$ is a regular graph of degree m.

Planar Graph

Planar graph is a graph or a multigraph that can be drawn in a plane or sphere such that it's edges do not cross.

- $\boxed{\sum_{i=1}^{n} \deg(V_i) = 2|E|}$
- $\boxed{\sum_{i=1}^{n} \deg(r_i) = 2|E|}$ where r_i are the regions.
- If degree of each region is K then $K \cdot |R| = 2|E|$
- If degree of each region is atleast 3 then $3|R| \le 2|E|$
- For simple planar graph :

  (*i*) **Euler's formula:**

  $\boxed{|R| = |E| - |V| + 2}$ if graph is connected

  $\boxed{|R| = |E| - |V| + (k+1)}$ with 'k' components

  (*ii*) **For connected planar simple graph:** $\boxed{|E| \le \{3|V| - 6\}}$

  (*iii*) **For connected planar simple graph with no triangles:** $\boxed{|R| \le \{2|V| - 4\}}$

- If $K_{3,3}$ and K_5 homomorphic fusion (degree = 1 vertex) subgraph then not planner = Kuratowski's.
- **For disconnected graph:** $n - k \le e \le \frac{(n-k)(n-k+1)}{2}$, $k \ge n - e$
- **For connected graph:** $n - 1 \le e \le \frac{n(n-1)}{2}$
- There exists atleast one vertex $v \in G$ such that degree $(v) \le 5$.
- $K_{m,n}$ is planner $\Leftrightarrow$ ($m \le 2$ or $n \le 2$)
- K_n is planner $\Leftrightarrow n \le 4$

Scanned by CamScanner

- A non planar graph with minimum number of vertices is K_5.
- A non planar graph with minimum number of edges is $K_{3,3}$.

Polyhedral Graph

A simple connected planar graph in which every interior region is a polygon of same degree and degree of every vertex $\deg(V) \geq 3 \quad \forall V \in G$

- $\boxed{3|V| \leq 2|E|}$
- $\boxed{3|R| \leq 2|E|}$

Complementary Graph

Complement of a graph G denoted by $\bar{G}$ is also a simple graph with same vertices as of G, and two vertices are adjacent in $\bar{G}$ iff the two vertices are not adjacent in G.

- $\boxed{G \cup \bar{G} = K_n}$
- $\boxed{|E(G)| + |E(\bar{G})| = |E(K_n)| = \frac{n(n-1)}{2}}$

Isomorphic Graphs

Two graphs G and G^* are isomorphic, if there is a function $f: V(G) \to V(G^*)$ such that f is bijection and "for each pair of vertices u and v of G: $\{u, v\} \in E(G)$ iff $\{f(u), f(v)\} \in E(G^*)$".

- Two graphs are isomorphic, iff their complements are isomorphic.
- If G and $\bar{G}$ are isomorphic then
  (*i*) The number of vertices in G and G' are same.
  (*ii*) The number of edges in G and G' are same.
  (*iii*) The degree sequence of G and G' are same.
  (*iv*) The number of cycles of every length in G and G' are same.
- If G is a simple graph such that $G \cong \bar{G}$ then G is said to be "self complementary".
- **In a self-complementary graph:**

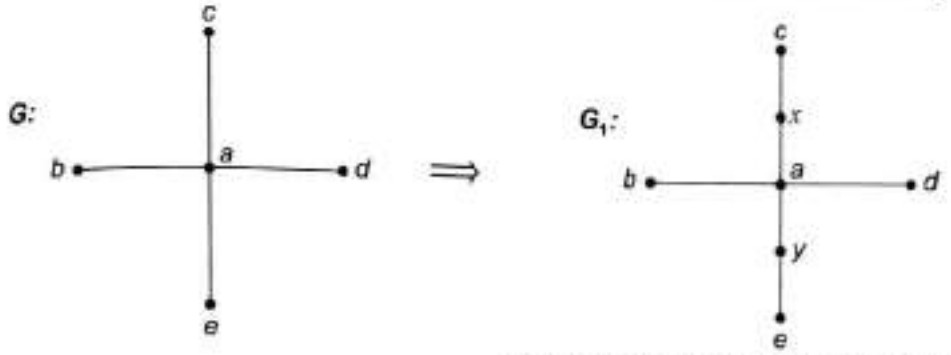
$$\boxed{|E(G)| = \frac{n(n-1)}{4}; \text{ where } n \text{ is number of vertices in } G}$$

Scanned by CamScanner

Homomorphism

A graph G_1 is said to be homomorphic to G if G_1 can be obtained by dividing some edge(s) of G.

Example: G_1 is homomorphic to G is shown in the following:



$\{a, c\}$ of G is divided into $\{c, x\}$ and $\{x, a\}$ edges.
$\{e, a\}$ of G is divided into $\{e, y\}$ and $\{y, a\}$ edges.

Coloring or Proper Coloring

Vertices of a graph G are colored such that no two adjacent vertices have same color.

Chromatic Number $\chi(G)$

Minimum number of colors needed for vertex coloring of graph G is called *chromatic number*.

- Chromatic number of $K_n = \chi(K_n) = n$
- Bipartite graph is 2-colorable. i.e. a non-empty graph is bichromatic iff it is Bipartite.
- Any planar graph is 4-colorable. i.e. χ(planar graph) ≤ 4.
- Tree is 2-colorable
- Equivalence relation between vertices of the same color in a connected graph gives the chromatic partition.
- k-cycle graph is 2-colorable; k is even length cycle
- k-cycle graph is 3-colorable; k is odd length cycle

Matching

- $\beta \geq$ independent number $\geq \dfrac{n}{\text{Chromatic}}$.
- It is a set of edges with atmost one edge incident on every vertex.

$$\boxed{\text{degree}(v) \leq 1, \quad \forall v \in G}$$

Scanned by CamScanner

Maximal Matching

Maximal matching is a maximal matching in which no edge of the graph can be added to it.

Maximum Matching

Maximum matching is a matching with maximum number of edges.

Matching Number

The number of edges in maximum matching of the graph.

Perfect Matching

Every vertex of the matching contain exactly one degree. i.e., every vertex is incident with exactly one edge. i.e. A matching which is also a covering is called perfect matching.

degree $(v) = 1, \ \forall v \in G$

Number of perfect matchings for $K_{2n} = \dfrac{(2n)!}{2^n \times n!}$

K_n has a perfect matching iff n is even.

Complete Matching

In a bipartite graph having a vertex partition V_1 and V_2. A complete matching of vertices in a set V_1 into those of V_2 is a matching such that every vertex in V_1 is matched against some certain vertex in V_2, such that no two vertices of V_1 is matched against a single vertex in V_2.

Covering

It is set of edges where every vertex of graph incident with atleast one edge in 'G' $[\deg(v_i) \geq 1]; \forall v_i \in G$.

Note:

- A line covering of a graph with n-vertices has atleast $n/2$ edges.
- No minimal line covering can contain a cycle and the components of a minimal cover are always stargraphs and from a minimal cover no edge can be deleted.

Minimal Covering

It is covering in which no deletion of an edge is possible while still covering the vertices.

Scanned by CamScanner

Minimum Covering

Smallest (less number of edges) minimal covering is minimum covering.

Covering Number

The number of edges in minimum covering is covering number.

Traversable Multigraph

If there is a path in graph which includes all the vertices and uses each edge exactly once (i.e. the graph has either Euler cycle or Euler trail) then such graph is traversable.

Eulerian Graph

If a graph contains "closed traversable trial or Euler circuit" (it may repeat vertices), then it is Eulerian Graph. When all vertex of even degree.

Note:

- A graph G is traversable, if number of vertices with odd degree in the graph is exactly zero or two.
- Euler path exists but Euler Circuit doesn't exist iff the number of vertices with odd degree is exactly two.
- Euler Circuit exists but Euler path does not exist iff number of vertices with odd degree is 0.

Hamiltonian Path

If there exists a path which contains each vertex of the graph exactly once, then such a path is called as Hamiltonian path.

Hamiltonian Cycle

It is Hamiltonian path where first and last vertices are same. If pendant edge then not Hamiltonian.

DIRAC

Sufficient condition degree of all vertex $\geq \frac{n}{2}$ then Hamiltonian.

ORES

$d(u) + d(v) \geq n$: u and v are non adjacent vertex.

■■■

Scanned by CamScanner

Probability

5

Mean, Median and Mode

- $$\text{Mean } (\bar{X}) = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{\sum_{i=1}^{n} f_i x_i}{\sum f_i}$$

- $$\text{Median} = \frac{x_{n/2} + (x_{n/2}+1)}{2}; \; n \text{ is even}$$
$$= x_{n+1/2}; \; n \text{ is odd}$$
$$= L_{med} + \left(\frac{N/2 - F}{f}\right) \times w$$

where L_{med} = lower limit of median class
$N = \Sigma f_i$
F = Cumulative frequency upto the median class (cumulative frequency of the preceding class)
f = Frequency of the median class

Mode : Value of 'x' corresponding to maximum frequency.

$$L_{mode} + \frac{f_m - f_1}{(2f_m - f_1 - f_2)} \times h$$

where L_{mode} = lower limit of modal class
f_m = frequency of modal class
f_1 = preceding frequency of modal class
f_2 = Following frequency of modal class

Note:

- Mode = 3 Median – 2 Mean [for Asymmetric distribution]
- Mean = Mode = Median [for Symmetric distribution]

Scanned by CamScanner

Axioms of Probability

Let A and B be two events. Then

1. $P(\bar{A}) = 1 - P(A)$
2. $P(\phi) = 0$;
3. $P(A - B) = P(A) - P(A \cap B)$
4. $P(A - B) = P(A \cap \bar{B})$
5. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
6. $P(A \cup B) = P(A) + P(B)$; mutually exclusive events.
7. $P(A \cap B) = P(A) \cdot P(B/A) = P(B) \cdot P(A/B)$
8. $P(A \cap B) = P(A) \cdot P(B)$; independent events.
9. $P(A \cap B) = \phi$; mutually exclusive events.
10. $P(S) = 1$; S is sample space.
11. $\max(0, P(A) + P(B) - 1) \le P(A \cap B) \le \min(P(A), P(B))$
12. $\max(P(A), P(B)) \le P(A \cup B) \le \min(1, P(A) + P(B))$
13. $\max(0, P(A_1) + P(A_2) \ldots + P(A_n) - (n-1)) \le P(A_1 \cap A_2 \ldots \cap A_n)$
    $\le \min(P(A_1), P(A_2) \ldots, P(A_n))$
14. $\max(P(A_1), P(A_2) \ldots, P(A_n)) \le P(A_1 \cup A_2 \ldots \cup A_n)$
    $\le \min(1, P(A_1) + P(A_2) \ldots + P(A_n))$
15. $P(A|B) = \dfrac{P(A \cap B)}{P(B)}$
16. $P(A|B) = P(A)$; independent events.
17. $P(E_1 \cap E_2 \cap \ldots \cap E_n) = P(E_1) \cdot P(E_2) \ldots P(E_n)$; independent events.
18. Rule of total probability: $P(X) = \sum_{i=1}^{n} P(E_i)\, P(X|E_i)$
19. Baye's Theorem:

$$P(E_1|X) = \frac{P(E_1) \cdot P(X|E_1)}{P(E_1) \cdot P(X|E_1) + P(E_2) \cdot P(X|E_2) + P(E_3) \cdot P(X|E_3)}$$

In general,

$$P(E_i|X) = \frac{P(E_i) \cdot P(X|E_i)}{\sum_{j=1}^{n} P(E_j) \cdot P(X|E_j)}$$

Scanned by CamScanner

Random Variable (Stochastic Variable)

Random variable assigns a real number to each possible outcome.

Let X be a discrete random variable, then

1. $F(x) = P(X \le x)$ is called distribution function $\sum_{i=0}^{n} P(i)$ of X.
2. Mean or Expectation of $X = \mu = E(X) = \sum_{i=1}^{n} x_i \, P(x_i)$
3. Variance of $X = \sigma^2 = E(X^2) - [E(X)]^2 = \sum_{i=1}^{n} (x_i - \mu)^2 P(x_i)$
4. Standard deviation of $X = \sigma = \sqrt{\text{Variance}}$
5. $\sum_{i=1}^{n} P(x_i) = 1$

Types of Random Variables

1. **Discrete Random Variable:** "Finite set of values" or "Countably infinite".
2. **Continous Random Variable (Non-discrete):** "Infinite number of uncountable values".

Discrete Distributions

1. **Binomial Distribution:** The probability that the event will happen exactly r times in n trials i.e. r successes and $n - r$ failures will occur.

$$P(X = r) = P(r) = \Sigma\, {}^nC_r p^r q^{n-r}$$

$$\text{Mean} = E(x) = np$$

$$\text{Variance } (\sigma^2) = V(x) = npq = np(1-p)$$

$$S.D\,(\sigma) = \sqrt{npq} = \sqrt{np(1-p)}$$

Where $r = 0, 1, \ldots, n$

$q = 1 - p$

n = fixed number of trials

p = probability of success

2. **Poisson Distribution:**

$$\boxed{P(X = x) = \sum \frac{e^{-\lambda} \cdot \lambda^x}{x!};\ x = 0, 1, 2, \ldots, \infty}$$

Scanned by CamScanner

Where X = Discrete random variable

λ = Parameter of distribution (positive constant)

- Mean (μ) = Variance (σ^2) = λ
- $S.D = \sqrt{\lambda}$

Poisson distribution is a limiting case of binomial distribution as $n \to \infty$ and $p \to 0$.

X	X Counts	p(x)	Value of X	E(x)	V(x)
Discrete uniform	Outcomes that are equally likes (finite)	$\frac{1}{b-a+1}$	$a \le x \le b$	$\frac{b+a}{2}$	$\frac{(b-a+2)(b-a)}{12}$
Binomial	Number of successes in n fixed trials	$\binom{n}{x} p^x (1-p)^{n-x}$	$x = 0, 1, \ldots, n$	np	np(1 − p)
Poisson	Number of arrivals in a fixed time period	$\frac{e^{-\lambda}\lambda^x}{x!}$	$x = 0, 1, 2, \ldots$	λ	λ
Geometric	Number of trials up through 1st success	$(1-p)^{x-1} p$	$x = 1, 2, 3, \ldots$	$\frac{1}{p}$	$\frac{1-p}{p^2}$
Negative Binomial	Number of trials up through kth success	$\binom{x-1}{k-1}(1-p)^{x-k} p^k$	$x = k, k+1, \ldots$	$\frac{k}{p}$	$\frac{k(1-p)}{p^2}$
Hyper-geometric	Number of marked individuals in sample taken without replacement	$\frac{\binom{M}{x}\binom{N-M}{n-x}}{\binom{N}{n}}$	max (0, M + n − N) $\le x \le$ min (M, n)	$n \cdot \frac{M}{N}$	$\frac{nM(N-M)(N-n)}{N^2(N-1)}$

Continuous Distribution

Let X be continous random variable. Then

(*i*) **Density functions:**

$$\textbf{(a)}\ P(X \le a) = \int_{-\infty}^{a} f(x) \cdot dx,\ \textbf{(b)}\ P(a \le X \le b) = \int_{a}^{b} f(x) \cdot dx$$

(*ii*) **Mean** $= E(x) = \int_{-\infty}^{\infty} x \cdot f(x) \cdot dx$

(*iii*) **Variance of** $X = V(X) = \int_{-\infty}^{\infty} [x - E(x)]^2 \cdot f(x)\, dx = E(x^2) - (E(x))^2$

$$= \int_{-\infty}^{\infty} x^2 f(x) dx - \left[\int_{-\infty}^{\infty} x f(x) dx\right]^2$$

(*iv*) $\int_{-\infty}^{\infty} f(x) dx = 1$

Scanned by CamScanner

1. Uniform Distribution (Rectangular Distribution)

(*i*) Density function:

$$f(x) = \frac{1}{b-a}; \quad a \le x \le b$$

$$= 0 \text{ ; otherwise}$$

(*ii*) Cumulative function:

$$P(X \le x) = \int_{-\infty}^{x} f(x) \cdot dx = \begin{cases} 0 & ; \text{ if } x \le a \\ \dfrac{x-a}{b-a} & ; \text{ if } a \le x \le b \\ 1 & ; \text{ if } x > b \end{cases}$$

(*iii*) Mean (μ) = $(a + b)/2 = E(X)$

(*iv*) Variance (σ^2) = $(b - a)^2/12$

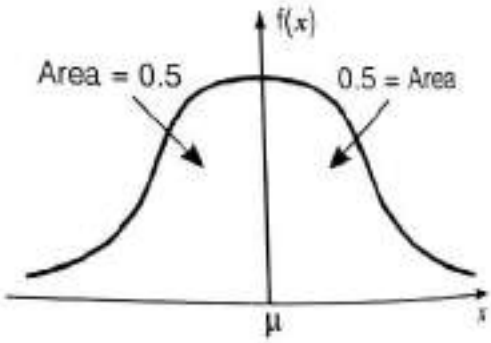
2. Normal Distribution

(*i*) Density function $f(x) = \dfrac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}; -\infty \le x \le \infty, \sigma > 0, -\infty < \mu < \infty$

(*ii*) Normal distribution is symmetrical

(*iii*) Mean = μ; Variance = σ^2

(*iv*) $f(x) \ge 0$ for all x



(*v*) $\int_{-\infty}^{\infty} f(x) \cdot dx = 1$

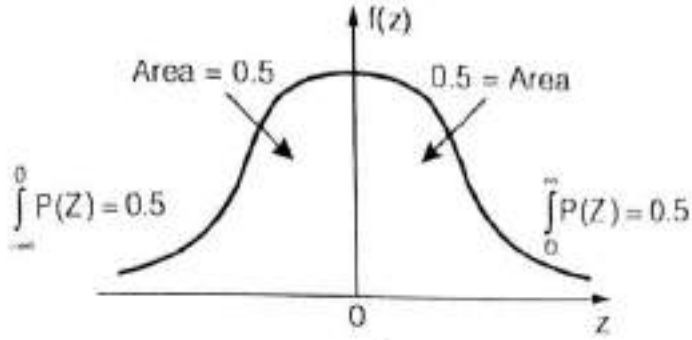
(*vi*) $P(Z) = \dfrac{1}{\sqrt{2\pi}} e^{-\frac{Z^2}{2}}; -\infty \le Z \le \infty$

and $Z = \dfrac{x-\mu}{\sigma}$

$Z = \dfrac{x-np}{\sqrt{npq}}$ (when approximating binomial by normal)

Z = Standard normal variate

Scanned by CamScanner



3. Exponential Distribution

(*i*) Density function:

$$f(x) = \lambda \cdot e^{-\lambda x} \quad ; x > 0$$
$$= 0 \quad ; \text{ Otherwise}$$

(*ii*) Mean (μ) $= \dfrac{1}{\lambda} = S.D(\sigma)$

(*iii*) Variance (σ^2) $= \dfrac{1}{\lambda^2}$

X	X Measures	f(x)	Value of X	E(x)	V(x)
Continuous uniform	Outcomes with equal density (continuous)	$\frac{1}{b-a}$	$a \le x \le b$	$\frac{b+a}{2}$	$\frac{(b-a)^2}{12}$
Exponential	Time between events time until an event	$\lambda e^{-\lambda x}$	$x \ge 0$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Normal	Values with a bell-shaped distribution(continuous)	$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$	$-\infty < x < \infty$	μ	σ
Standard normal (Z)	Standard scores	$\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}$	$Z = \frac{x-\mu}{\sigma}$	0	1
Binomial approximation	Number of successes in a fixed time period (large average)	Approx, normal if $np \ge 5$ and $n(1-p) \ge 5$ by CLT	$Z = \frac{x-np}{\sqrt{np(1-p)}}$	np	np(1 − p)
Poisson approximation	Number of occurrences in a fixed time period (large average)	Approx normal if $\lambda > 30$	$Z = \frac{x-\lambda}{\sqrt{\lambda}}$	λ	λ

■■■

Scanned by CamScanner

Linear Algebra

6

Matrix

Principal Diagonal: In a square matrix all elements a_{ij} for which $i = j$ are elements of principal diagonal.

Matrices

1. **Upper Triangular matrix:** A square matrix in which all the elements below the principle diagonal are zero.
2. **Lower Triangular Matrix:** A square matrix in which all the elements above the principle diagonal are zero.
3. **Diagonal Matrix:** A square matrix in which all the elements other than the elements of principle diagonal are zero.
4. **Scalar Matrix:** A diagonal matrix with all elements of principle diagonal being same.
5. **Idempotent Matrix:** 'A' is square matrix $\ni A^2 = A$.
6. **Involutary Matrix:** 'A' is square matrix $\ni A^2 = I$.
7. **Nilpotent Matrix:** 'A' is square matrix $\ni A^m = 0$ where m is the least positive integer and m is also called as Index of class of Nilpotent matrix A.
8. **Transpose Matrix:** A^T is transpose matrix of matrix A. A^T can be obtained by switching the rows as columns and columns as rows of A.
9. **Symmetric Matrix:** 'A' is a square matrix $\ni A^T = A$.
10. **Skew-Symmetric Matrix:** 'A' is a square matrix $\ni A^T = -A$.
11. **Orthogonal Matrix:** 'A' is a orthogonal matrix $\ni A^T = A^{-1}$ or $AA^T = I = A^TA$.
12. **Conjugate Matrix of A ($\bar{A}$) or ($-A$) :** 'A' is any matrix, by replacing the elements by corresponding conjugate complex numbers the matrix obtained is conjugate of 'A'.

    Example:

$$A = \begin{bmatrix} 2+3i & 4+7i & 5 \\ 2i & 3 & 9-i \end{bmatrix} \Rightarrow \bar{A} = \begin{bmatrix} 2-3i & 4-7i & 5 \\ -2i & 3 & 9+i \end{bmatrix}$$

13. **Transpose Conjugate Matrix (A^θ) or (A^*):** $(\bar{A})^T$.

Scanned by CamScanner

14. **Hermitian Matrix:** 'A' is a square matrix $\ni$ $A^{\theta} = A$
All diagonal elements of hermitian matrix are real number and all off-diagonal elements above and below the principle diagonal must be conjugate of each other. i.e. $a_{ij} = \overline{a_{ji}}$.

Example: $\begin{bmatrix} 2 & 3-4i \\ 3+4i & 5 \end{bmatrix}$

15. **Skew-Hermitian Matrix:** 'A' is a square matrix $\ni$ $A^{\theta} = -A$
All diagonal elements of Skew-Hermitian matrix are purely imaginary or zero and all off-diagonal elements above and below the principle diagonal must be conjugate of each other with opposite sign. i.e. $a_{ij} = -\overline{a_{ji}}$.

Example: $\begin{bmatrix} 2i & 3-4i \\ -3-4i & 5i \end{bmatrix}$

16. **Unitary Matrix:** 'A' is a square matrix $\ni$ $A^{\theta} = A^{-1}$ or $A\,A^{\theta} = I = A^{\theta}\,A$
17. **Boolean Matrix:** Any matrix with only elements '0' or '1'
18. **Sparse Matrix:** A matrix 'A' in which more number of elements are zeros.
19. **Dense Matrix:** A matrix which is not sparse.
20. **Singular and Non-singular Matrix:** A square matrix 'A' is singular if $|A| = 0$, and non singular if $|A| \neq 0$. Only non-singular matrices have inverse.
21. **Adjoint Matrix:** Transpose of cofactors matrix. i.e. Adj (A) = (Cof (A))^T

Properties of Matrices

- $A + B = B + A$ (Commutative)
- $(A + B) + C = A + (B + C)$ (Associative)
- $AB \neq BA$ (Not commutative)
- $(AB)\,C = A(BC)$ (Associative)
- $A(B + C) = AB + AC$ (Distributive)
- $A = \frac{1}{2}(A + A^T) + \frac{1}{2}(A - A^T)$
- $A(Adj\,A) = (Adj\,A)\,A = |A|I_n$
- $Adj\,(AB) = (Adj\,B).\,(Adj\,A)$

Scanned by CamScanner

- $A^{-1} = \frac{Adj\,A}{|A|}, |A| \neq 0$
- $(A^{-1})^{-1} = A$ and $(A^{-1})^T = (A^T)^{-1}$
- $(AB)^{-1} = B^{-1}A^{-1}$.
- If A is a square matrix of order n then $|Adj\,A| = (\det A)^{n-1} = |A|^{n-1}$ and $|Adj(Adj\,A)| = |A|^{(n-1)^2}$
- If $|A| \neq 0$ then $|A^{-1}| = \frac{1}{|A|}$
- If $A_{n \times n}$ matrix then $|KA| = K^n|A|$
- If A is square matrix then
  (*i*) $A + A^T$ is always symmetric
  (*ii*) $A - A^T$ always skew-symmetric
- If A and B are symmetric then
  (*i*) $A + B$ is also symmetric.
  (*ii*) $A - B$ is also symmetric.
  (*iii*) $AB + BA$ is symmetric
  (*iv*) $AB - BA$ is skew-symmetric
  (*v*) A^n and B^n are symmetric
- If A and B are skew-symmetric then,
  (*i*) $A + B$ is also skew-symmetric.
  (*ii*) $A - B$ is also skew-symmetric.
  (*i*) A^n and B^n are symmetric, if 'n' is even
  (*ii*) A^n and B^n are skew-symmetric, if 'n' is odd
- The determinant of orthogonal matrix and unitary matrix A has absolute value '1'.
- If $A_{m \times n}$ and $B_{n \times p}$ then product of AB requires
  (*i*) mnp multiplications
  (*ii*) $m(n-1)p$ additions
  (*iii*) for each entry, n multiplications and $(n-1)$ additions.
- $(A^T)^T = A$, $(kA)^T = k(A^T)$
- $(A+B)^T = A^T + B^T$, $(AB)^T = B^T A^T$

Scanned by CamScanner

- **Rank of Matrix ($r(A)$):** It is the order of its largest non-vanishing (non-zero) minor of the matrix.
- Rank is equal to the number of linearly independent rows or columns in the matrix.
- The system of linear equation $AX = B$ has a solution (consistent) iff rank of A = Rank of $(A|B)$
- The system $AX = B$ has
  - (*i*) A unique solution iff Rank (A) = Rank $(A|B)$ = Number of variables
  - (*ii*) Infinitely many solutions $\Leftrightarrow$ Rank (A) = Rank $(A|B)$ < number of variables
  - (*iii*) No solution if Rank $(A) \neq$ Rank $(A|B)$ i.e. Rank (A) < Rank $(A|B)$
- The system $AX = 0$ has
  - (*i*) Unique solution (zero solution or trivial solution) if Rank (A) = number of variables
  - (*ii*) Infinitely many number of solutions (non-trivial solutions) if Rank (A) < number of variables
- If Rank $(A) = r$, and number of variables $= n$ then, the number of linearly independent infinite solutions of $AX = 0$ is $(n - r)$
- In the system of homogenous linear equation $AX = 0$
  - (*i*) If A is singular then the system possesses non-trivial solution (i.e. infinite solution)
  - (*ii*) If A is non-singular then the system possesses trivial (zero) solution (i.e. unique solution)
- Rank of a diagonal matrix = Number of non-zero elements in diagonal.
- If A and B are two matrices
  - (*i*) $r(A + B) \leq r(A) + r(B)$
  - (*ii*) $r(A - B) \geq r(A) - r(B)$
  - (*iii*) $r(AB) \leq \min\{r(A), r(B)\}$
- If a matrix A has rank 'R', then A contains 'R' linearly independent vectors (row/column)
- The system of homogeneous linear equations such that number of unknowns (or variables) exceeds the number of equations necessarily possesses a non-zero solution.

Eigen Value

Let 'A' be a square matrix of order n and λ be a scalar then $|A - \lambda I| = 0$ is the characteristic equation of A. The roots of characteristic equation are called eigen values/lantent roots/Characteristic roots.

Scanned by CamScanner

- The set of eigen values of matrix is called "spectrum of matrix"
- A matrix of order n will have n latent roots, not necessarily distinct.

Eigen Vector

Corresponding to each eigen value λ, there exists a non-zero solution X such that $(A - \lambda I)X = 0$ then X is eigen vector/latent/vector/characteristic vector of A.

Properties of Eigen Values

- Sum of eigen values of a matrix = sum of elements of principal diagonal (trace).

$$\Sigma\lambda_i = \lambda_1 + \lambda_2 + \lambda_3 + \ldots + \lambda_n = \text{Trace of } A$$

- Product of eigen values = Determinant of matrix.

$$\Pi\lambda_i = \lambda_1 \cdot \lambda_2 \cdot \ldots \lambda_n = |A|$$

- If λ is eigen value of A then $\frac{1}{\lambda}$ is eigen value of A^{-1}. (provided $\lambda \neq 0$ i.e. A is non-singular).
- Eigen values of A and A^T are same
- If λ is eigen value of orthogonal matrix then $\frac{1}{\lambda}$ is also its eigen value $[\because A^T = A^{-1}]$
- If $\lambda_1, \lambda_2, \ldots, \lambda_n$ are eigen values of matrix 'A', then
  - (*i*) $\lambda_1^m, \lambda_2^m \ldots, \lambda_n^m$ are eigen values of matrix A^m.
  - (*ii*) $\lambda_1 + K, \lambda_2 + K, \ldots, \lambda_n + K$ are eigen values of $A + KI$
  - (*iii*) $(\lambda_1 - K)^2, (\lambda_2 - K)^2, \ldots, (\lambda_n - K)^2$ are eigen values of $(A - KI)^2$
  - (*iv*) $K\lambda_1, K\lambda_2, \ldots, K\lambda_n$ are eigen values of KA.
- The eigen values of symmetric matrix are real.
- The eigen values of skew-symmetric matrix are either purely imaginary or zero.
- The modulus of the eigen values of orthogonal and unitary matrices = 1.
- If a matrix is either lower or upper triangular or diagonal then the principal diagonal elements themselves are the eigen values.
- Zero is eigen value of a matrix iff the matrix is singular.

■■■

Scanned by CamScanner

Numerical Methods

7

Linear Algebraic Equations

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}_{3\times3}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{3\times1}, \quad B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}_{3\times1}$$

LU Decomposition or LU Factorization or Triangularization (Doolittle's or Crout's triangularisation Method)

- Principle minors of A are non-singular.

$$AX = B \Rightarrow LUX = B \quad [\text{since } A = LU]$$

- $$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \text{ (Doolittle's method)}$$

- $$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} \text{ (Crout's method)}$$

Order of Computation in Doolittle's Method:

In Doolittle's method the equations are set-up row-wise for A's elements.

(a) First row of $A \Rightarrow u_{11} = a_{11}$; $u_{12} = a_{12}$; $u_{13} = a_{13}$

(b) Second row of $A \Rightarrow l_{21} = \dfrac{a_{21}}{a_{11}}$; $u_{22} = a_{22} - \dfrac{a_{21}}{a_{11}} \cdot a_{12}$; $u_{23} = a_{23} - l_{21}\, u_{13}$

(c) Third row of $A \Rightarrow$

$$l_{31} = \frac{a_{31}}{a_{11}};\ l_{32} = \frac{1}{u_{22}}\left[a_{32} - \frac{a_{31}}{a_{11}} \cdot a_{12}\right];\ u_{33} = a_{33} - l_{31} \cdot u_{13} - l_{33}\, u_{23}$$

Similarly, in Crout's method the equations are set-up column-wise for A's elements.

Scanned by CamScanner

(v) Error is order of h^2 and error term is h^3

(vi) If for the same integration limits, tabulation (step size) is halved then error is reduced by factor '4'.

2. **Simpson's Rule:**

(i) It gives exact result when polynomial of degree ≤ 3.

(ii) If we are evaluating $I = \int_a^b f(x)\,dx$ by Simpson's rule then

$$\boxed{\text{Error} = -\frac{h^5}{90} \cdot f^{iv}(\xi) \times n_i} \quad (a \le \xi \le b)$$

(iii) Simpson's rule is more accurate as compare to trapezoidal rule since it is a fourth order method as compared to trapezoidal rule which is second order.

(iv) Simpson's 1/3rd Rule:

$$\boxed{\int_{x_0}^{x_n} y\,dx = \frac{h}{3}[(y_0 + y_n) + 2(y_2 + y_4 + \ldots + y_{n-2}) + 4(y_1 + y_3 + \ldots + y_{n-1})]}$$

(v) Simpson's 3/8 Rule:

$$\boxed{\int_{x_0}^{x_n} y\,dx = \frac{3h}{8}[(y_0 + y_n) + 2(y_3 + y_6 + \ldots + y_{n-3}) + 3(y_1 + y_2 + \ldots + y_{n-1})]}$$

(vi) If tabulation (step size) is halved, the error in Simpson's rule evaluation of the integral is reduced by a factor of '16'.

■■■

Scanned by CamScanner

Calculus

8

Limit, Continuity and Differentiability

Limit Existence at a Point

- Left Limit (LHL): $\lim_{x \to a^-} f(x) = \lim_{h \to 0} f(a-h)$

  Right Limit (RHL): $\lim_{x \to a^+} f(x) = \lim_{h \to 0} f(a+h)$

  $\lim_{x \to a} f(x)$ exists, iff both LHL and RHL exists.
- Indeterminate forms: $0/0$, ∞/∞, $0 \times \infty$, $\infty - \infty$, 0^0, 1^∞, 0^∞
- L Hospital's Rule: If $\lim_{x \to a} [f(x)/g(x)]$ is of the form either $0/0$ or ∞/∞ then

  $$\lim_{x \to a} [f(x)/g(x)] = \lim_{x \to a} [f'(x)/g'(x)]$$

Continuity

A function $f(x)$ is said to be continuous at $x = a$ if:

$$\lim_{x \to a^+} f(x) = \lim_{x \to a^-} f(x) = f(a)$$

- If $f(x)$ and $g(x)$ is continuous function then following are also continuous. (*i*) $f(x) + g(x)$, (*ii*) $f(x) \cdot g(x)$, (*iii*) $f(x) - g(x)$, (*iv*) $f(x)/g(x)$ $[g(x) \neq 0]$
- Polynomial, Exponential, Sin and Cos functions are continuous everywhere.

Differentiability

A function $f(x)$ is said to be differentiable at $x = a$ if it is continuous at $x = a$ and if left derivative $= f'(a) =$ right derivative

$$\text{i.e. } \lim_{h \to 0^-} \frac{f(a) - f(a-h)}{h} = f'(a) = \lim_{h \to 0^+} \frac{f(a+h) - f(a)}{h}$$

- If $f(x)$ is differentiable in (a, b), then it is always continuous at (a, b), but converse need not be true.

Mean Value Theorems

Rolle's Mean Value Theorem

If (*i*) '$f(x)$' is continuous in $[a, b]$, (*ii*) '$f(x)$' is differentiable in (a, b), (*iii*) $f(a) = f(b)$ then $\exists\, c \in (a, b)$ such that $f'(c) = 0$

Scanned by CamScanner

Lagrange's Mean Value Theorem

If (*i*) '$f(x)$' is continuous in $[a, b]$, (*ii*) '$f(x)$' is differentiable in (a, b)

then $\exists\, c \in (a, b)$ such that $f'(c) = \dfrac{f(b) - f(a)}{b - a}$

Cauchy's Mean Value Theorem

Let $f(x)$ and $g(x)$ are two functions if

(*i*) $f(x)$ and $g(x)$ are continuous in $[a, b]$

(*ii*) $f(x)$ and $g(x)$ are differentiable in (a, b)

(*iii*) $g'(x) \neq 0 \;\; \forall x \in (a, b)$

then $\exists\, c \in (a, b)$ such that $\dfrac{f'(c)}{g'(c)} = \dfrac{f(b) - f(a)}{g(b) - g(a)}$

Derivatives

- $\dfrac{d}{dx}(\sin x) = \cos x$
- $\dfrac{d}{dx}(\cos x) = -\sin x$
- $\dfrac{d}{dx}(\tan x) = \sec^2 x$
- $\dfrac{d}{dx}(\cot x) = -\operatorname{cosec}^2 x$
- $\dfrac{d}{dx}(\sec x) = \sec x \cdot \tan x$
- $\dfrac{d}{dx}(\operatorname{cosec} x) = \operatorname{cosec} x \cot x$
- $\dfrac{d}{dx}(\sin^{-1} x) = \dfrac{1}{\sqrt{1-x^2}}$
- $\dfrac{d}{dx}(\cos^{-1} x) = \dfrac{-1}{\sqrt{1-x^2}}$
- $\dfrac{d}{dx}(\tan^{-1} x) = \dfrac{1}{1+x^2}$
- $\dfrac{d}{dx}(\cot^{-1} x) = \dfrac{-1}{1+x^2}$
- $\dfrac{d}{dx}(\sec^{-1} x) = \dfrac{1}{x\sqrt{x^2-1}}$
- $\dfrac{d}{dx}(\operatorname{cosec}^{-1} x) = \dfrac{-1}{x\sqrt{x^2-1}}$
- $\dfrac{d}{dx}(\sinh x) = \cosh x$
- $\dfrac{d}{dx}(\cosh x) = \sinh x$
- $\dfrac{d}{dx}(e^x) = e^x$
- $\dfrac{d}{dx}(a^x) = a^x \cdot \log a$
- $\dfrac{d}{dx}[f(x)]^n = n \cdot [f(x)]^{n-1}$
- $\dfrac{d}{dx}(ax+b)^n = n \cdot (ax+b)^{n-1} \cdot a$
- $\dfrac{du}{dx} = \dfrac{du}{dy}\dfrac{dy}{dx}$ (chain rule)
- $\dfrac{d}{dx}(uv) = u \cdot \dfrac{dv}{dx} + v \cdot \dfrac{du}{dx}$

Scanned by CamScanner

- $\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{v.\frac{du}{dx} - u.\frac{dv}{dx}}{v^2}$
- $\sinh x = \frac{e^x - e^{-x}}{2}$
- $\cosh x = \frac{e^x + e^{-x}}{2}$
- $\frac{d}{dx}(\log_e x) = \frac{1}{x}$

Integrals

- $\int x^n \cdot dx = \frac{x^{n+1}}{n+1}; \ (n \neq -1)$
- $\int \frac{1}{x} \cdot dx = \log_e x$
- $\int e^x \cdot dx = e^x$
- $\int a^x = \frac{a^x}{\log a}$
- $\int \frac{f'(x)}{f(x)} \cdot dx = \log [f(x)]$
- $\int [f(x)]^n \cdot f'(x) \cdot dx = \frac{[f(x)]^{n+1}}{n+1}$
- $\int \frac{1}{a^2 + x^2} dx = \frac{1}{a} \cdot \tan^{-1} \frac{x}{a}$
- $\int \frac{1}{a^2 - x^2} dx = \frac{1}{2a} \cdot \log\left[\frac{a+x}{a-x}\right]$
- $\int \frac{1}{x^2 - a^2} dx = \frac{1}{2a} \cdot \log\left[\frac{x-a}{x+a}\right]$
- $\int \frac{dx}{\sqrt{a^2 - x^2}} = \sin^{-1} \frac{x}{a}$
- $\int \frac{dx}{\sqrt{a^2 + x^2}} = \sinh^{-1} \frac{x}{a}$
- $\int \frac{dx}{\sqrt{x^2 - a^2}} = \cosh^{-1} \frac{x}{a}$
- $\int \sinh x = \cosh x$
- $\int \cosh x = \sinh x$
- $\int \sin x \cdot dx = -\cos x$
- $\int \cos x \cdot dx = \sin x$
- $\int \tan x \cdot dx = \log \cos x = \log \sec x$
- $\int \cot x = \log \sin x$
- $\int \sec x \cdot dx = \log (\sec x + \tan x)$
- $\int \operatorname{cosec} x \cdot dx = \log (\operatorname{cosec} x - \cot x)$
- $\int \sec^2 x \cdot dx = \tan x$
- $\int \operatorname{cosec}^2 x \cdot dx = -\cot x$
- $\int e^{ax} \sin bx \, dx = \frac{e^{ax}}{a^2 + b^2} (a \sin bx - b \cos bx)$
- $\int e^{ax} \cos bx \, dx = \frac{e^{ax}}{a^2 + b^2} (a \cos bx + b \sin bx)$
- $\int \sqrt{a^2 - x^2} \cdot dx = \frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2} \sin^{-1} \frac{x}{a}$

Scanned by CamScanner

- $\int \sqrt{a^2 - x^2} \cdot dx = \frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2}\sinh^{-1}\frac{x}{a}$
- $\int \sqrt{x^2 - a^2} \cdot dx = \frac{x}{2}\sqrt{x^2 - a^2} - \frac{a^2}{2}\cosh^{-1}\frac{x}{a}$

Definite Integrals

Theorem 1:

If $f(x)$ is continous in $[a, b]$ and $F(x)$ is any antiderivative of $f(x)$ in $[a, b]$

then $\int_a^b f(x)dx = F(b) - F(a)$.

Theorem 2:

If $f(x)$ is continous on $[a, b]$ then $F(x) = \int_a^b f(t) \cdot dt$ is differentiable at every point of x in $[a, b]$ and $\frac{dF}{dx} = \frac{d}{dx}\int_a^b f(t) \cdot dt = f(x)$.

i.e., If $f(x)$ is continous on $[a, b]$ then $\exists$ a function $F(x)$ whose derivative on $[a, b]$ is $f(x)$.

Theorem 3:

If $f(x)$ is continous on $[a, b]$ and $u(x)$ and $v(x)$ are differentiable functions of x whose values lie in $[a, b]$ then $\frac{d}{dx}\int_{u(x)}^{v(x)} f(t) \cdot dt = f(v(x))\frac{dv}{dx} - f(u(x)) \cdot \frac{du}{dx}$

Properties of Definite Integrals

1. $\int_a^a f(x) \cdot dx = 0$
2. $\int_a^b f(x) \cdot dx = \int_b^a f(z) \cdot dz$
3. $\int_a^b f(x) \cdot dx = -\int_b^a f(x) \cdot dx$
4. $\int_a^b k \cdot dx = k(b - a)$
5. $\int_a^b f(x) \cdot dx = \int_a^c f(x) \cdot dx + \int_c^b f(x) \cdot dx$
6. $\int_0^a f(x) \cdot dx = \int_0^a f(a - x) \cdot dx$

Scanned by CamScanner

7. $\int_{-a}^{a} f(x)\cdot dx = 2\int_{0}^{a} f(x)\cdot dx$; if $f(x)$ is even function $f(-x)=f(x)$

$= 0$; if $f(x)$ is odd function $f(-x) = -f(x)$

8. $\int_{a}^{b} f(x)\cdot dx = \int_{a}^{b} f(a+b-x)\cdot dx$

9. $\int_{0}^{2a} f(x)\cdot dx = \int_{0}^{a} f(x)\cdot dx + \int_{0}^{a} f(2a-x)\cdot dx$

10. $\int_{0}^{2a} f(x)\cdot dx = 2\int_{0}^{a} f(x)\cdot dx$; if $f(2a-x)=f(x)$

$= 0$; If $f(2a-x) = -f(x)$

11. $\int_{0}^{na} f(x)\cdot dx = n\cdot\int_{0}^{a} f(x)\cdot dx$; if $f(x+a)=f(x)$

[$f(x)$ is periodic function with period 'a']

12. $\int_{0}^{a} x\cdot f(x)\cdot dx = \frac{a}{2}\int_{0}^{a} f(x)\cdot dx$; if $f(a-x)=f(x)$

13. **Mean value theorem for definite integrals**: Let f be continuous in (a, b) then $\exists c \in (a, b)$ such that $f(c) = \dfrac{\int_{a}^{b} f(x)dx}{(b-a)}$

$f(c)$ is the average value of the function in the interval (a, b).

14. $\int_{a}^{b} f(x)dx \le \int_{a}^{b} g(x)dx$ If $f(x) \le g(x)$ on (a, b)

15. $\int_{0}^{\pi/2} \sin^n x\cdot dx = \int_{0}^{\pi/2} \cos^n x\cdot dx = \dfrac{(n-1)\cdot(n-3)\cdot(n-5)....}{n\cdot(n-2)(n-4)....}\times K$

Where, $K = 1$; if n is odd and $K = \pi/2$; if n is even

Scanned by CamScanner

Improper Integrals

1. $\int_a^b f(x)\cdot dx$ is "improper integral";

   (*i*) If $a = -\infty$ or $b = \infty$ or both

   (*ii*) If $f(x) = \infty$ for one or more values of x in $[a, b]$.

2. $\int_a^b f(x)\cdot dx$ is "convergent", if the value of integral is "finite".

3. $\int_a^b f(x)\cdot dx$ is "divergent", if the value of integral is "infinite".

4. If $0 \le f(x) \le g(x)$; $\forall x$ and $\int_a^\infty g(x)\cdot dx$ converges then $\int_a^\infty f(x)$ also converges.

5. If $0 \le g(x) \le f(x)$; $\forall x$ and $\int_a^\infty g(x)\cdot dx$ diverges then $\int_a^\infty f(x)$ also diverges.

6. If $f(x)$ and $g(x)$ are two functions, such that $\lim_{n\to\infty} \frac{f(x)}{g(x)} = k$

   where k = finite and $k \ne 0$ then

   $\int_a^\infty f(x)\cdot dx$ and $\int_a^\infty g(x)\cdot dx$ "converge" or "diverge" together.

7. $\int_a^\infty e^{-Px}\cdot dx$ and $\int_{-\infty}^b e^{Px}\cdot dx$ converges for any constant $P > 0$ and diverges for $P \le 0$.

8. $\int_a^b \frac{dx}{(b-x)^P}$ is convergent iff $P < 1$.

9. $\int_a^b \frac{dx}{(x-a)^P}$ is convergent iff $P < 1$.

10. $\int_1^\infty \frac{dx}{x^P}$ converges if $P > 1$ and diverges if $P \le 1$.

Scanned by CamScanner

Total Derivatives

If $u = f(x, y)$ where x and y are functions of t, then the "total derivative" of u with respect to t is:

$$\frac{du}{dt} = \frac{\partial u}{\partial x}\left(\frac{dx}{dt}\right) + \frac{\partial u}{dy}\left(\frac{dy}{dt}\right)$$

By putting $t = x$ in above equation, we get

$$\frac{du}{dx} = \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} \cdot \frac{dy}{dx}$$; where x and y are connected by some relation.

Partial Derivatives

1. If $u = f(x, y)$, then $\frac{\partial u}{\partial x} = \lim_{h \to 0} \frac{[f(x+h, y) - f(x, y)]}{h}$ and

   $$\frac{\partial u}{\partial y} = \lim_{k \to 0} \frac{[f(x, y+k) - f(x, y)]}{k}.$$

2. If $u = f(x, y)$, where $x = g(r, s)$ and $y = h(r, s)$ then

   $$\frac{\partial u}{\partial r} = \frac{\partial u}{\partial x} \cdot \frac{\partial x}{\partial r} + \frac{\partial u}{\partial y} \cdot \frac{\partial y}{\partial r}$$

   $$\frac{\partial u}{\partial s} = \frac{\partial u}{\partial x} \cdot \frac{\partial x}{\partial s} + \frac{\partial u}{\partial y} \cdot \frac{\partial y}{\partial s}$$

3. If $u = f(x, y)$ is a homogenous function of degree 'n' then

   $x \cdot u_x + y \cdot u_y = n \cdot u$

   $x^2 u_{xx} + 2xy \cdot u_{xy} + y^2 u_{yy} = n(n-1)u$

4. If $u = f(x, y, z)$ is a homogenous function of degree 'n', then

   $x \cdot u_x + y \cdot u_y + z \cdot u_z = n \cdot u$

5. If $u = f(x, y)$ is not homogenous function, but $F(u)$ is a homogenous function of degree 'n', then

   $$x \cdot u_x + y \cdot u_y = n\left[\frac{F(u)}{F'(u)}\right] = G(u)$$

   $$x^2 \cdot u_{xx} + 2xy \cdot u_{xy} + y^2 \cdot u_{yy} = G(u)\,[G'(u) - 1]$$

Scanned by CamScanner

Maximum and Minima [Extremum]

1. A necessary condition for $f(a)$ to be an extreme value of $f(x)$ is $f'(a) = 0$.
2. The vanishing of $f'(a) = 0$ is only a necessary but not a sufficient condition for $f(a)$ to be an extreme value of $f(x)$.
3. **Stationary Values:** A function $f(x)$ is said to be stationary for $x = C$ and $f(C)$ is a stationary value of $f(x)$ if $f'(C) = 0$.
4. To find points of maximum, points of minimum and saddle points the following steps are required:
   (*i*) Put $f'(x) = 0$ and find stationary points (x_i)
   (*ii*) If $f''(x_i) > 0$ then $f(x)$ is minimum at x_i $\forall x_i$.
   (*iii*) If $f''(x_i) < 0$ then $f(x)$ is maximum at x_i $\forall x_i$.
   (*iv*) $f'(x) = 0$ and $f''(x) = 0$ and so on
      (a) Continue finding derivatives at x_i untill the first non-zero derivative is found.
      (b) If the first non-zero derivative is obtained at odd derivative then x_i is a point of inflection or saddle point. If the first non-zero derivative is obtained at even derivative then x_i is either a point of maximum or a point of minimum depending on whether the derivative is negative or positive respectively.
5. **Sufficient condition for extrema:**
   Theorem: $f(C)$ is an extremum of $f(x)$ iff $f'(x)$ changes sign as x passes through C.
   Case (*i*): If $f'(x) = 0$ and $f'(x)$ changes sign from positive to –ve as 'x' passes through C then $f(C)$ is maximum.
   Case (*ii*): If $f'(x) = 0$ and $f(x)$ changes sign from –ve to +ve as x passes through C then $f(C)$ is minimum.
   Case (*iii*): If $f'(x) = 0$ and $f(x)$ does not changes sign as x passes through C then $f(C)$ is not extremum but rather a point of inflection or saddle point.
6. **For 2-variable functions:**
   (*i*) Put $f_x = 0$ and $f_y = 0$; find stationary points by solving simultaneously.
   (*ii*) If $f_{xx} f_{yy} > f_{xy}^2$ and (a) If $f_{xx} > 0$ then $f(x)$ is "minimum" at a, (b) If $f_{xx} < 0$ then $f(x)$ is "maximum" at a
   (*iii*) If $f_{xx} f_{yy} - f_{xy}^2 < 0$ then $x = a$ is a point of inflection or saddle point.

■■■

Scanned by CamScanner

A Handbook on
Computer Science

2

Digital Logic

CONTENTS

1. Logic Functions 72
2. Minimization 78
3. Combinational Circuits 82
4. Sequential Circuits 94
5. Number Representation and Computer Arithmetic 107

Scanned by CamScanner

Logic Functions

1

Logic Gates

- OR, AND, NOT are Basic Gates
- NAND, NOR are Universal Gates
- EX-OR, EX-NOR are Arithmetic Gates

NOT Gate

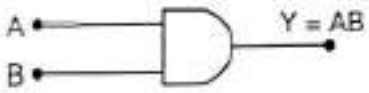
- Also referred to as "Inversion" or "Complementation".

  Symbol and Truth table:

Input A	Output $Y = \overline{A}$
0	1
1	0

AND Gate

Symbol and Truth table:



Inputs		Output
A	B	Y = AB
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

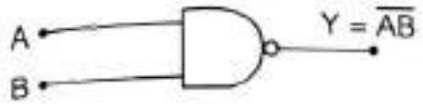
Symbol and Truth table:

Inputs		Output
A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Scanned by CamScanner

NAND Gate

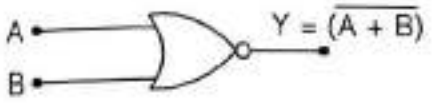
Symbol and Truth table:



Inputs		Output
A	B	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

Symbol and Truth table:



Inputs		Output
A	B	$Y = \overline{(A + B)}$
0	0	1
0	1	0
1	0	0
1	1	0

EXOR Gate

- It is also called "stair case switch".
- Symbol and Truth table :

Inputs		Output
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

- Boolean function of 2-input EXOR operation :

$$\boxed{Y = A \oplus B = \bar{A}B + A\bar{B}}$$

(*i*) It acts like as an "odd number of 1's detector in the input".

(*ii*) It is mostly used in "parity generation and detection".

(*iii*) When both the inputs are same, then output becomes LOW or Logic '0'.

(*iv*) When both the inputs are different, then output becomes HIGH or Logic '1'.

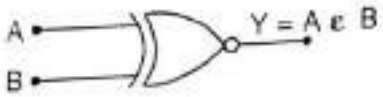
Scanned by CamScanner

(v) $\boxed{\begin{array}{ll} A \oplus A = 0, & A \oplus 0 = A \\ A \oplus \overline{A} = 1, & A \oplus 1 = \overline{A} \end{array}}$

- $A \oplus A \oplus A \oplus ...$ upto n terms = 0, when n = even
- $A \oplus A \oplus A \oplus ...$ upto n terms = A, when n = odd

EXNOR Gate

- It acts like as an "even number of 1's detector".
- Symbol and Truth table :



Inputs		Output
A	B	Y = A ε B
0	0	1
0	1	0
1	0	0
1	1	1

- Boolean function of 2-input EXNOR operation :

$$\boxed{Y = A \odot B = \overline{A \oplus B} = (\overline{\overline{A}B + A\overline{B}}) = AB + \overline{A}\overline{B}}$$

(*i*) When both the inputs are same, then output becomes HIGH or Logic '1'.

(*ii*) When both the inputs are different, then output becomes LOW or Logic '0'.

(*iii*) $\boxed{\begin{array}{ll} A \odot A = 1, & A \odot 1 = A \\ A \odot \overline{A} = 0, & A \odot 0 = \overline{A} \end{array}}$

- $A \odot A \odot A \odot ...$ upto n terms = 1, when n is even
- $A \odot A \odot A \odot ...$ upto n terms = A, when n is odd

Note: .....

- $\overline{A} \oplus B = A \oplus \overline{B} = A \odot B$
- $\overline{A} \odot B = A \odot \overline{B} = A \oplus B$
- For odd no. of inputs EXOR and EXNOR are same and for even number of variables they are complements to each other.

  i.e. $A \oplus B \oplus C = A \odot B \odot C$

  as $A \oplus B \oplus C = \overline{A \oplus B}C + (A \oplus B)\overline{C}$

Scanned by CamScanner

$$= (A \odot B)\,C + (A \oplus B)\,\overline{C}$$

$$= (A \odot B)C + \overline{A \odot B}\,\overline{C}$$

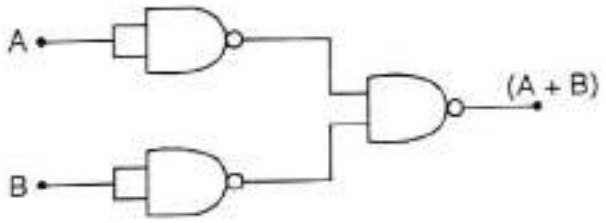
$$= A \odot B \odot C$$

- $\overline{A \oplus B \oplus C} = A \odot B \oplus C = A \oplus B \odot C$
- $\overline{A \odot B \odot C} = A \oplus B \odot C = A \odot B \oplus C$
- $\boxed{A \oplus B \oplus AB = A + B}$
- EXOR and EXNOR are also called arithmetic gates as they are used in addition, subtraction, comparator circuits.

Implementation of Gates using NAND

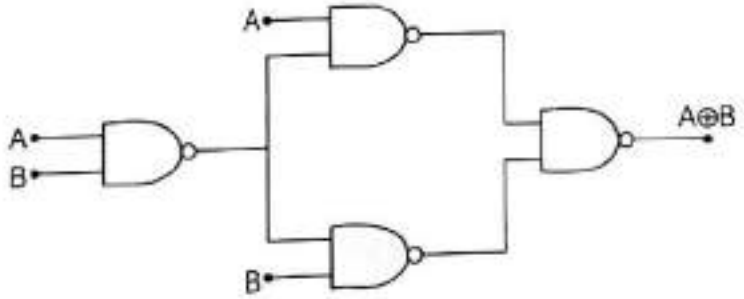
- AND gate :

- OR gate :

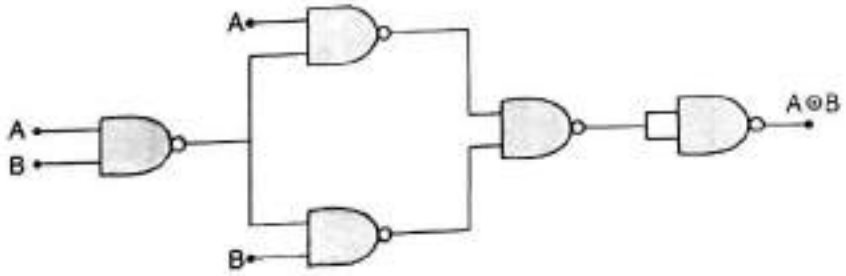


- NOT gate :

- EXOR gate :



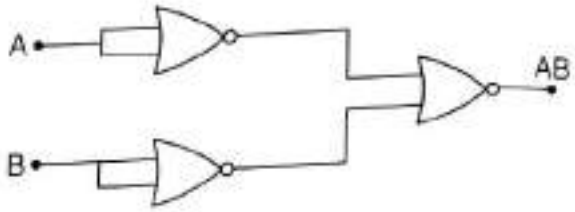
- EXNOR Gate :



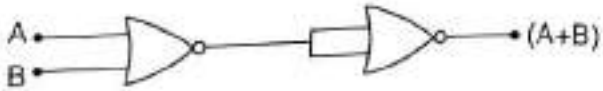
Scanned by CamScanner

Implementation of Gates using NOR

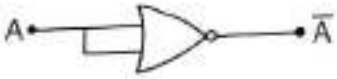
- AND gate :



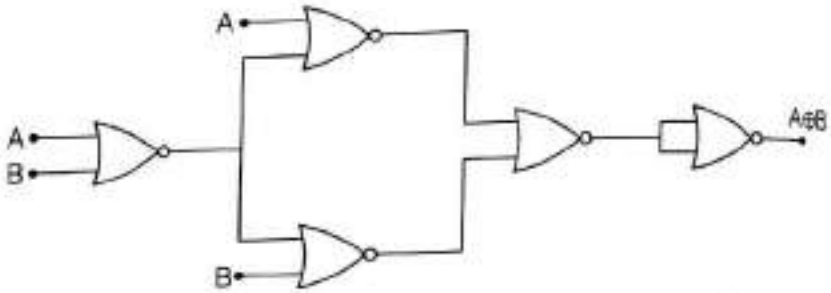
- OR gate :



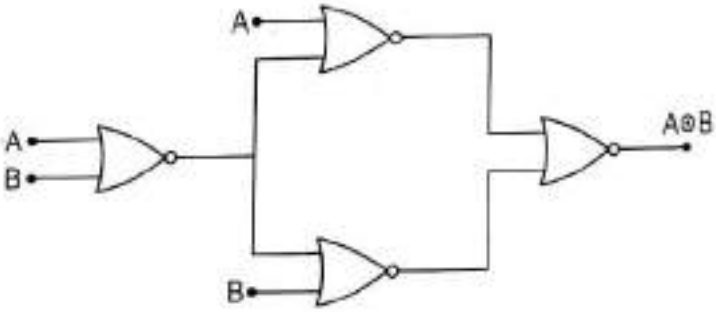
- NOT gate :



- EXOR gate :



- EXNOR gate :



Note: .....

- Number of NAND and NOR gates needed to implement other logic gates is shown in the following table.

Logic gate	No. of NAND gates	No. of NOR gates
NOT	1	1
AND	2	3
OR	3	2
EX OR	4	5
EX NOR	5	4

Scanned by CamScanner

- **Alternative Symbols of Gates**

1. Bubbled – OR gate ≡ NAND gate
2. Bubbled – NAND gate ≡ OR gate
3. Bubble – NOR gate ≡ AND gate
4. Bubbled – AND gate ≡ NOR gate

■■■

Scanned by CamScanner

Minimization

2

Boolean Algebra Laws

Identity Law

(*i*) $A + A = A$

(*ii*) $A \cdot A = A$

Commutative Law

(*i*) $A + B = B + A$

(*ii*) $A \cdot B = B \cdot A$

Associative Law

(*i*) $A + (B + C) = (A + B) + C = A + B + C$

(*ii*) $A \cdot (B \cdot C) = (A \cdot B)\, C = A\, B\, C$

Distributive Law

(*i*) $A(B + C) = AB + AC$

(*ii*) $A + (BC) = (A + B)(A + C)$

De Morgan's Law

(*i*) $(A + B)' = A'B'$

(*ii*) $(AB)' = A' + B'$

Note:

- NAND and NOR gates do not follow associative law but follow commutative law.
- OR, AND, EXOR, EXNOR follow commutative and associative both laws.
- $A + AB = A$
- $A\,(A + B) = A$
- $A + A'B = A + B$
- $A\,(A' + B) = AB$

Scanned by CamScanner

Boolean Algebraic Theorems

AND-operation Theorem

(i) $A \cdot A = A$

(ii) $A \cdot 0 = 0$

(iii) $A \cdot 1 = A$

(iv) $A \cdot \overline{A} = 0$

Involution Theorem

$$(A')' = \overline{\overline{A}} = A$$

OR-operation Theorem

(i) $A + A = A$

(ii) $A + 0 = A$

(iii) $A + 1 = 1$

(iv) $A + \overline{A} = 1$

De Morgan's Theorem

(i) $\overline{(A_1 \cdot A_2 \cdot A_3 \cdot \ldots A_n)} = \overline{A}_1 + \overline{A}_2 + \overline{A}_3 + \ldots + \overline{A}_n$

(ii) $\overline{(A_1 + A_2 + A_3 + \cdots + A_n)} = \overline{A}_1 \cdot \overline{A}_2 \cdot \overline{A}_3 \cdots \overline{A}_n$

Transposition Theorem

$$(A + B)(A + C) = A + BC$$

Distribution Theorem

$$A + BC = (A + B)(A + C)$$

Boolean Algebraic Theorems

Theorem No.	Theorem
1.	$(A+B)\cdot(A+\overline{B}) = A$
2.	$AB + \overline{A}C = (A+C)(\overline{A}+B)$
3.	$(A+B)(\overline{A}+C) = AC + \overline{A}B$
4.	$AB + \overline{A}C + BC = AB + \overline{A}C$
5.	$(A+B)(\overline{A}+C)(B+C) = (A+B)(\overline{A}+C)$
6.	$\overline{A \cdot B \cdot C \cdot \ldots} = \overline{A} + \overline{B} + \overline{C} + \ldots$
7.	$\overline{A + B + C + \ldots} = \overline{A} \cdot \overline{B} \cdot \overline{C} \ldots$

Scanned by CamScanner

Duality Theorem

- "Dual expression" is equivalent to write a negative logic of the given boolean relation. For this we :
  - (*i*) Change each OR sign by an AND sign and vice-versa.
  - (*ii*) Complement any '0' or '1' appearing in expression.
  - (*iii*) Keep literals as it is.
- For 1-time Dual, it is called "Self Dual Expression".
- For n-variables, maximum possible Self-Dual Function = $(2)^{2^{n-1}} = 2^{2^n/2}$
- Number of Boolean functions formed over
  - (*i*) *n*-boolean variables = 2^{2^n}
  - (*ii*) *n*-*n* valued variables = 2^{n^n}
- Number of *n*-valued functions formed over
  - (*i*) *n*-boolean variables = n^{2^n}
  - (*ii*) *n*-*n* valued varibles = n^{n^n}

Complementary Theorem

For obtaining complement expression we :

1. Change each OR sign by AND sign and vice-versa.
2. Complement any '0' or '1' appearing in expression.
3. Complement the individual literals.

Boolean Function Representation

Canonical Form

All the terms contain each literal.

Example: $F(A,B,C) = \overline{A}BC + ABC + \overline{A}B\overline{C}$

Standard Form

All the terms do not have each and every literal.

Example: $F(A,B,C) = \overline{A} + BC + A\overline{B}C$

Minterms and Maxterms

- *n*-binary variables have 2^n possible combinations and each of these possible combination is called "Minterm or Standard Product".

Scanned by CamScanner

- "Maxterm" is the complement of corresponding "Minterm" i.e. $\boxed{M = \bar{m}}$.
- In an n-variable Karnaugh-map there are 2^n cells.

Complete Simplification Rules

- Construct the K-map and place 1's in those cells corresponding to the 1's in the truth table. Place 0's in the other cells.
- Examine the map for adjacent 1's and loop those 1's which are not adjacent to any other 1's. These are called isolated 1's.
- Next, look for those 1's which are adjacent to only one other 1. Loop any pair containing such a 1.
- Loop any octet even it contains some 1's that have already been looped.
- Loop any quad that contains one or more 1's which have not already been looped, making sure to use the minimum number of loops.
- Loop any pairs necessarily to include any 1's that have not yet been looped, making sure to use the minimum number of loops.
- Form the OR sum of all the terms generated by each loop.
- K-map will provide minimized expression but not necessarily unique.
- Maximum number of product terms for irreducible expression with n-variables are: 2^{n-1} and maximum number of literals = $n \times 2^{n-1}$.
- Let $f(x_1, x_2, \ldots, x_n)$ be equal to 1 iff k or more variables are equal to 1, then number of prime implicants in f are ${}^nC_k$.

■■■

Scanned by CamScanner

Combinational Circuits

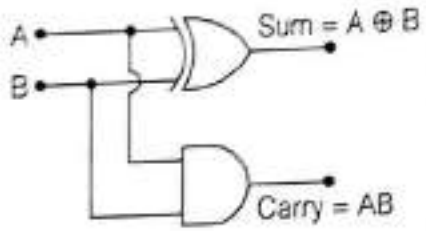
3

Combinational Circuits

- Output does not depend on previous value of input.
- No feedback is required.
- It consists of input variables, logic gates and output variables.
- No memory is required.

Half Adder

- A logic circuit for the addition of two one-bit numbers is known as half adder.
- Symbol and Truth Table:



Inputs		Outputs	
A	B	Sum(S)	Carry(C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Logical expression :

  Sum = $S = \bar{A}B + A\bar{B} = A \oplus B$

  Carry = $C = AB$
- Total number of NAND/NOR gates required to implement half adder = 5.
- Minimum number of logic gates needed to implement half adder circuit (if we have all gates except EX-OR and EX-NOR) is "3".
- To implement half adder, three 2 : 1 MUXs are required.

Full Adder

- It performs the arithmetic sum of the three input bits i.e. addend bit, augend bit and carry bit.
- **Logical expression :**

  Sum = $S = A \oplus B \oplus C$

  Carry = $C = AB + BC + CA = AB + C(A \oplus B)$
- A full adder can be implemented by two half adders and one OR gate or 1 full subtractor and 1 NOT gate.

Scanned by CamScanner

- Total number of NAND gates/NOR gates required to implement a full adder is "9".

Half Subtractor

- **Logical expression :**

  Difference = $D = \bar{A}B + A\bar{B} = A \oplus B$

  Borrow = $B = \bar{A}B$

- Total number of NAND/NOR gates required to implement the half subtractor is "5".

Full Subtractor

- It is a circuit which performs a subtraction between two bits taking into account that a '1' may have been borrowed by a lower significant stage.
- **Logical expression :**

  Difference = $D = A \oplus B \oplus C$

  Borrow = $B = \bar{A}B + \bar{A}C + BC = \bar{A}B + \left(\overline{A \oplus B}\right) \cdot C$

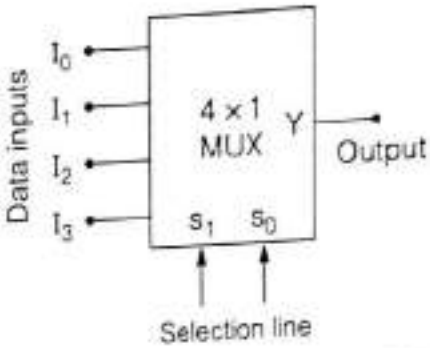
- A full subtractor can be implemented with two half subtractor and one OR gate or 1 full adder and 1 NOT gate.
- Number of NAND/NOR gates required to implement the full subtractor is "9".
- In parallel adder n-full adders or [$(n-1)$ full adders and 1 half adder] or [$(2n-1)$ half adders and $(n-1)$ OR-gates] are required to add two n-bit numbers.

Multiplexers (MUX)

- It selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- There are 2^n input lines where 'n' is the select line.
- MUX is also called data selector or many to one circuit or universal logic circuit or parallel to serial converter.
- If there are M number of data inputs and n is number of select lines then $n = \log_2 M$.

Scanned by CamScanner

4 : 1 MUX

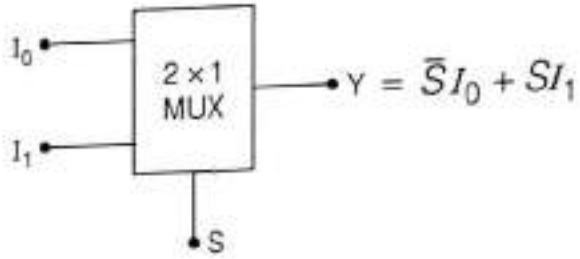


S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$\boxed{Y = \text{output} = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_0 I_3}$$

- The size of the MUX is specified by the 2^n input lines and the single output line for n selection lines.

2 : 1 MUX



Implementation of Higher order MUX using Lower order MUX

Given MUX	To be Implemented MUX	Required Number of MUX
2 : 1	4 : 1	3
4 : 1	16 : 1	4 + 1 = 5
4 : 1	64 : 1	16 + 4 + 1 = 21
8 : 1	64 : 1	8 + 1 = 9
8 : 1	256 : 1	32 + 4 + 1 = 37

Note: .....

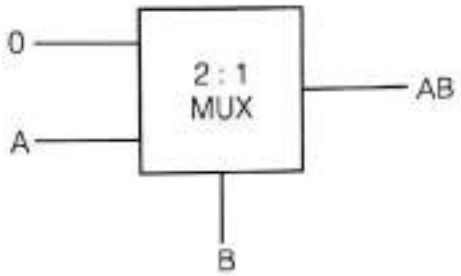
- To implement 2^n : 1 MUX by using 2 : 1 MUX, the total number of 2 : 1 MUXs required is $(2^n - 1)$.
- MUX is an Universal Logic circuit.
- Number of $(n \times 1)$ MUXs required to implement $(m \times 1)$ MUXs = $\left\lceil \frac{m-1}{n-1} \right\rceil$ and number of levels required = $\log_n m$.

Scanned by CamScanner

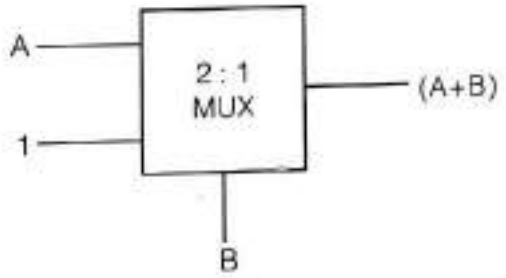
- Any two variable function can be implemented using one 4 : 1 MUX. (some of three variable functions can be implemented)
- With one 4 : 1 MUX and 1 NOT Gate
  - (*i*) All 2 variables logic functions can be implemented.
  - (*ii*) All 3 variables logic functions can be implemented.
- With one 8 : 1 MUX
  - (*i*) All three variable logic functions can be implemented.
  - (*ii*) Some of four variable logic functions can be implemented .
- With one 8 : 1 MUX and 1 NOT gate
  - (*i*) All three variable functions can be implemented.
  - (*ii*) All four variable functions can be implemented.

MUX as Universal Logic Circuit

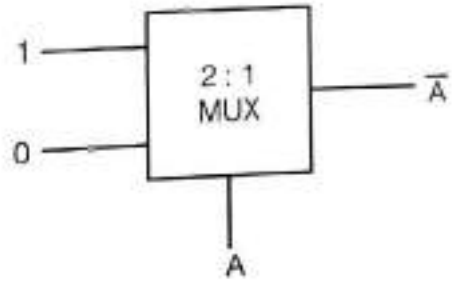
- AND gate using MUX :



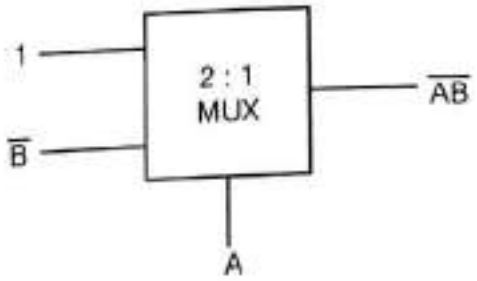
- OR gate using MUX :



- NOT gate using MUX :

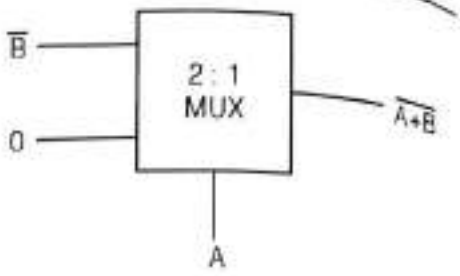


- NAND gate using two 2 : 1 MUXs :

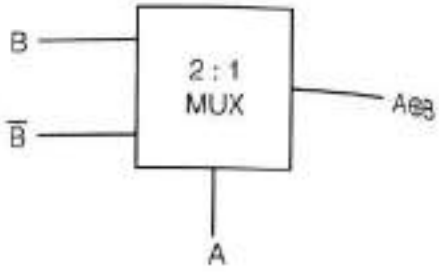


Scanned by CamScanner

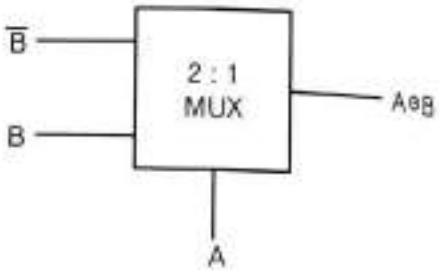
- NOR gate using two 2 : 1 MUXs :



- EXOR gate using two 2 : 1 MUXs :

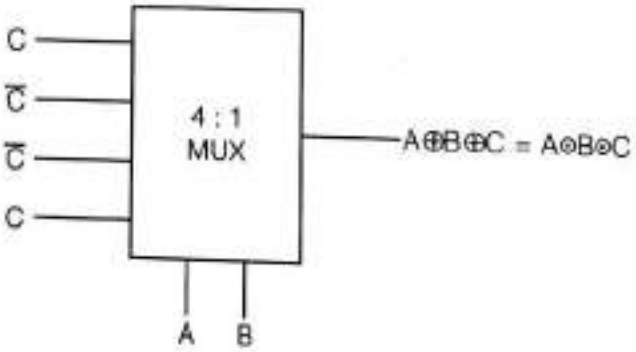


- EXNOR gate using two 2 : 1 MUXs :

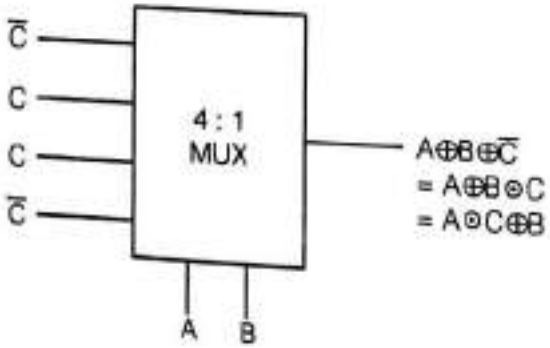


Three Input EXOR and EXNOR Gate Using MUX

- $A \oplus B \oplus C$

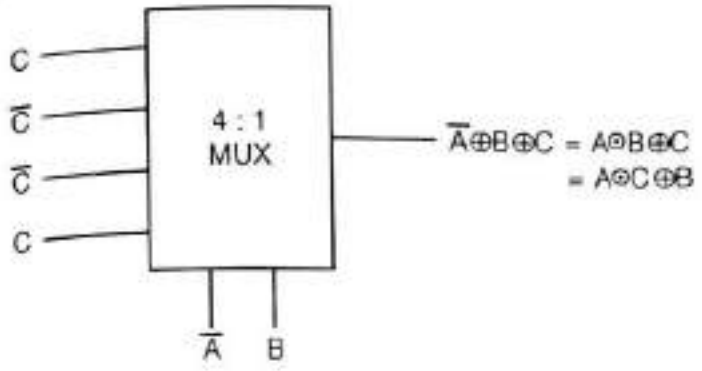


- $A \oplus B \oplus \overline{C}$

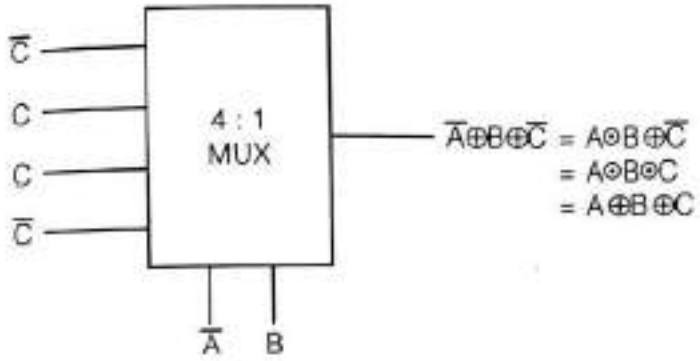


Scanned by CamScanner

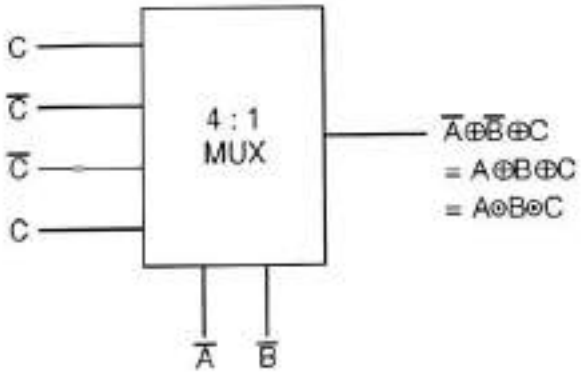
- $\bar{A} \oplus B \oplus C$



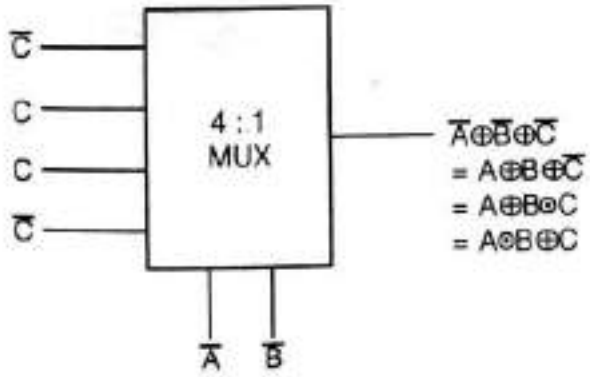
- $\bar{A} \oplus B \oplus \bar{C}$



- $\bar{A} \oplus \bar{B} \oplus C$

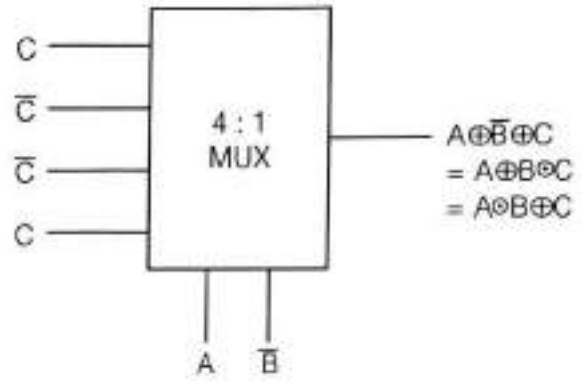


- $\bar{A} \oplus \bar{B} \oplus \bar{C}$

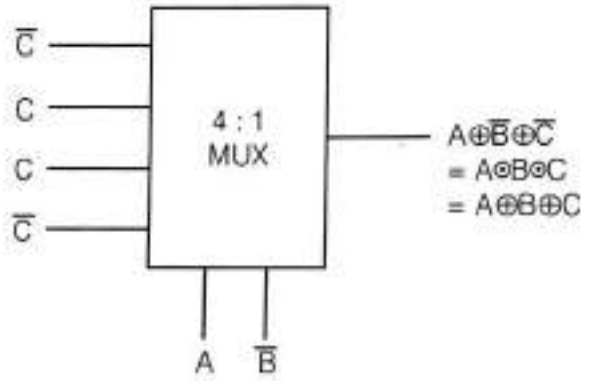


Scanned by CamScanner

- $A \oplus \bar{B} \oplus C$



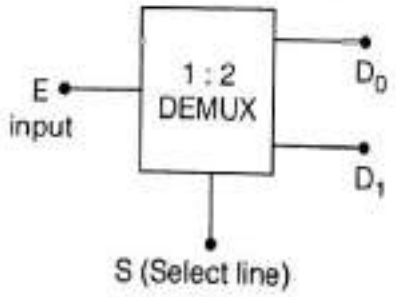
- $A \oplus \bar{B} \oplus \bar{C}$



Demultiplexer (DEMUX)

- It receives information on a single line and transmits this on one of 2^n possible output lines.
- A "Decoder" with an enable input can function as a "Demultiplexer".
- Demux is also called data distributor or one to many circuit.
- The number of select lines required in a single input and 'n' output DEMUX is $\log_2 n$.

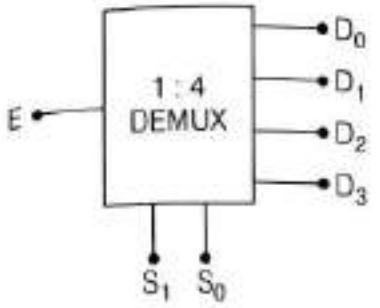
1 : 2 DEMUX



$$D_0 = \bar{S}\,E \text{ and } D_1 = SE$$

Scanned by CamScanner

1 : 4 DEMUX



Select lines		Outputs			
S_1	S_0	D_3	D_2	D_1	D_0
0	0	0	0	0	E
0	1	0	0	E	0
1	0	0	E	0	0
1	1	E	0	0	0

$$D_0 = \bar{S}_1 \bar{S}_0 E,\ D_1 = \bar{S}_1 S_0 E,\ D_2 = S_1 \bar{S}_0 E,\ D_3 = S_1 S_0 E$$

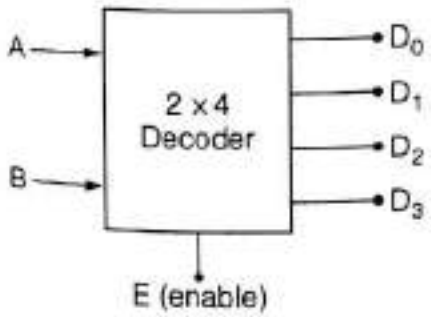
Implementation of Higher order DEMUX using Lower order DEMUX

Given DEMUX	To be Implemented DEMUX	Required Number of DEMUX
1 : 2	1 : 4	1 + 2 = 3
1 : 2	1 : 8	1 + 2 + 4 = 7
1 : 2	1 : 16	1 + 2 + 4 + 8 = 15
1 : 2	1 : 64	1 + 2 + 4 + 8 + 16 + 32 = 63
1 : 4	1 : 16	1 + 4 = 5

Decoders

- A "Decoder" have many inputs and many output lines.
- It is a combinational circuit that converts binary information from n input lines to a maximum 2^n unique output lines.
- If the n-bit decoded information has unused or don't care combinations, the decoder output will have less than 2^n outputs.
- Decoder is used to convert binary data into other codes like binary to octal (3 : 8 decoder) binary to hexadecimal (4 : 16 decoder).
- Total number of output lines : $m \le 2^n$, where n is total number of input lines.

2 × 4 Decoder



E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Scanned by CamScanner

- Number of $n \times 2^n$ decoders required to realize $m \times 2^m$ decoders:

$$x + y + z + \ldots + (I < 2^n) \text{ where } x = \left\lceil \frac{2^m}{2^n} \right\rceil,\ y = \left\lceil \frac{x}{2^n} \right\rceil,\ z = \left\lceil \frac{y}{2^n} \right\rceil \ldots \text{ and}$$

  Number of levels = 1+Number of times dividing with 2^n.
- 2×4 Decoder may act like a 1 : 4 DEMUX and Vice-versa.
- Decoder and Demux circuits are almost same.
- Decoder contains AND gates or NAND gates.
- To implement n-functions, we require:
  (a) n-muxs or
  (b) One $n \times 2n$ decoder and n OR gates.

Implementation of Decoders using Decoder

Given Decoder	To be Implemented Decoder	Required Number of Decoder
2 : 4	4 : 16	1 + 4 = 5
2 : 4	3 : 8	2 + 1 NOT Gate
4 : 16	8 : 256	1 + 16 = 17

Encoder

- Encoder is a combinational circuit which has many inputs and many outputs.
- It is used to convert other codes to binary such as octal to binary, hexadecimal to binary etc.
- In normal encoder only one of input line is high at a time and corresponding binary is available at outputs.
- In priority encoder more than one input line can be high but only for highest priority input line, binary output is available at output.

Code Converters

BCD to Excess-3 Code

- Let input variables of **BCD code** is A, B, C, D and output variables of excess-3 is W, X, Y, Z.
- Required truth table: (BCD + 0011 = excess-3-code)

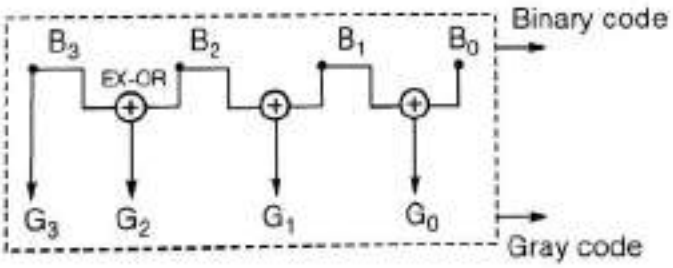
Scanned by CamScanner

Inputs (BCD)				Output (Excess-3 code)			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

- Minimised boolean function :

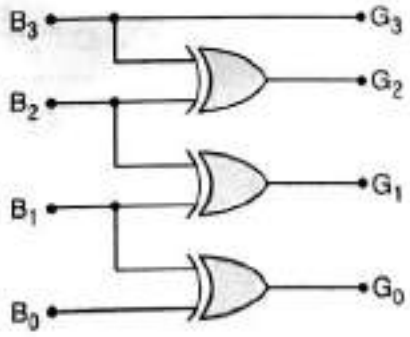
$$Z = \bar{D}$$
$$Y = CD + \bar{C}\bar{D}$$
$$X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$$
$$W = A + BC + BD$$

Binary-to Gray Code Converter



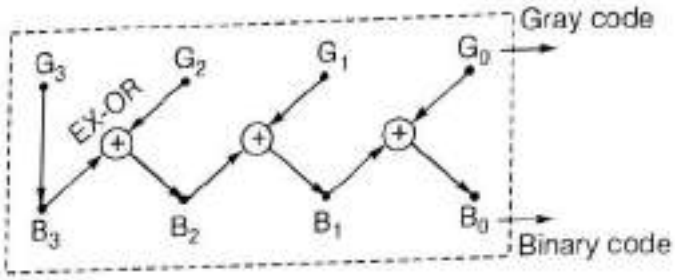
- Equivalent logical gate diagram :

$$G_3 = B_3$$
$$G_2 = B_3 \oplus B_2$$
$$G_1 = B_2 \oplus B_1$$
$$G_0 = B_1 \oplus B_0$$



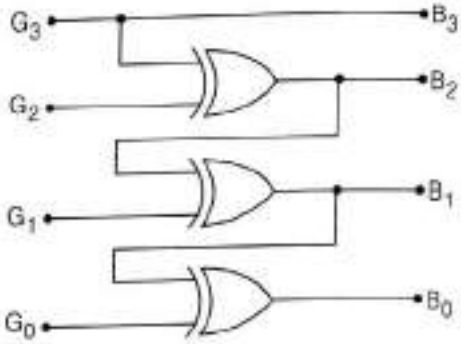
Scanned by CamScanner

Gray to Binary Converter



- Equivalent logical gate diagram :

$$B_3 = G_3$$
$$B_2 = B_3 \oplus G_2$$
$$B_1 = B_2 \oplus G_1$$
$$B_0 = B_1 \oplus G_0$$



Look Ahead Carry Adder

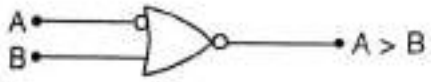
- In parallel adder for n bits there is $2n\ t_{pd}$ delay for providing result where t_{pd} is delay provided by each logic gate.
- Look ahead carry adder is used to make the addition faster.
- In look ahead carry adder to provide carry output it requires 3 logic gate delay.
- To provide sum it requires 4 logic gate array.
- Carry circuit is implemented using two level AND-OR gate circuit.
- Number of AND gates required in carry circuit for n bit look ahead carry adder is :

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

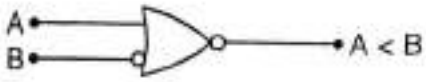
- Number of OR gates require in carry circuit for n-bit look ahead carry adder is n.

Comparator

- $(A > B) = A\bar{B}$

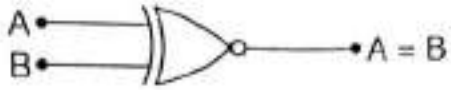


- $(A < B) = \bar{A}B$



Scanned by CamScanner

- $(A = B) = AB + \bar{A}\bar{B}$



- **Example:**
  Suppose we have

$$P = A_3 A_2 A_1 A_0$$
$$Q = B_3 B_2 B_1 B_0$$

Logic expression for $P = Q$, $P > Q$, $P < Q$ is

$$Y_1 [\text{for } P = Q] = (A_2 \odot B_3) \cdot (A_2 \odot B_2) \cdot (A_1 \odot B_1) \cdot (A_0 \odot B_0)$$

$$Y_2 [\text{for } P > Q] = A_3 \bar{B}_3 + (A_3 \odot B_3)(A_2 \bar{B}_2)$$
$$+ (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \bar{B}_1)$$
$$+ (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1).(A_0 \bar{B}_0)$$

$$Y_3 [\text{for } P < Q] = \bar{A}_3 B_3 + (A_3 \odot B_3)(\bar{A}_2 B_2)$$
$$+ (A_3 \odot B_3)(A_2 \odot B_2)(\bar{A}_1 B_1)$$
$$+ (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(\bar{A}_0 B_0)$$

■■■

Scanned by CamScanner

Sequential Circuits

4

Sequential Circuit

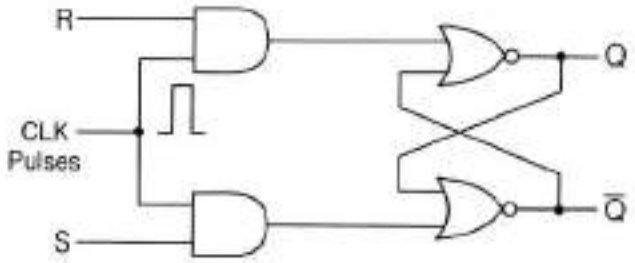
- Output depends on the present as well previous value of inputs.
- It consists input variables, sequential circuit and output.
- Memory is required.

Flip-Flops

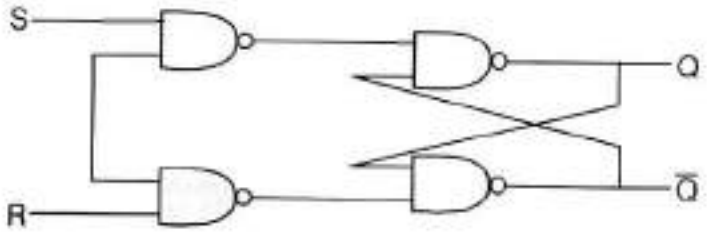
- These can store "one-bit of information".
- Flip-flop circuit is also known as "Bistable Multivibrator" or Latch".
- A flip-flop circuit can be constructed from two NAND-gates or two NOR-gates.

Clocked S-R Flip-flop

- Logic diagram 1 :



- Logic diagram 2 :



- Truth table of S-R FF :

Clock	S_n	R_n	Q_{n+1}	
0	X	X	Q_n	
1	0	0	Q_n	→ HOLD state
1	0	1	0	→ RESET state
1	1	0	1	→ SET state
1	1	1	Invalid	→ FORBIDDEN state

Scanned by CamScanner

- S_n and R_n denotes the inputs and Q_n the output during the bit time 'n'.
- 'Q_{n+1}' denotes the output Q after CLK passes, i.e. in bit time $(n + 1)$.
- Characteristics table of SR-FF :

S	R	Q_n	Q_{n+1}	
0	0	0	0	Q_n
0	0	1	1	
0	1	0	0	0
0	1	1	0	
1	0	0	1	1
1	0	1	1	
1	1	0	X	Invalid
1	1	1	X	

- Excitation table of S-R flip flop :

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

- Characteristic equation of SR-FF :

$$\boxed{Q_{n+1} = S + \bar{R}Q_n}$$

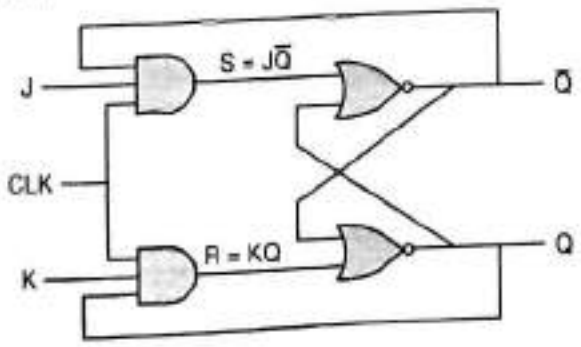
- Invalid states are present when both the inputs (S and R) are HIGH.
- Characteristic equation is valid only for $S.R = 0$.

J-K Flip-Flop

- JK-flip-flop is universal flip-flop because the flip-flops like D-flip-flop, SR-flip-flop and T-flip-flop can be derived from it.

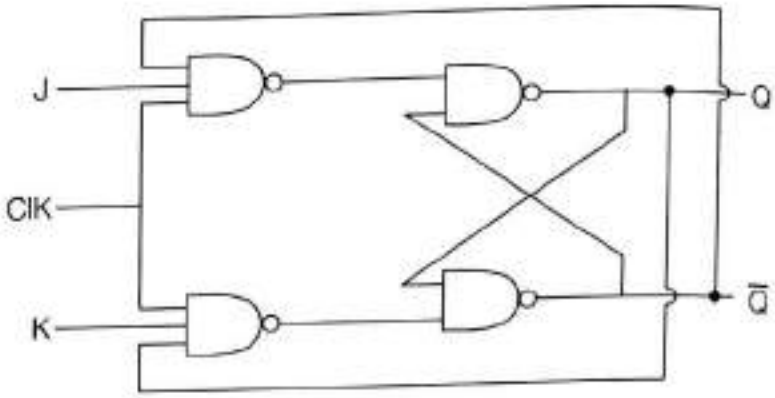
$$\boxed{S = J\bar{Q} \text{ and } R = KQ}$$

- Logic diagram 1 :



Scanned by CamScanner

- Logic diagram 2 :



- Truth Table of JK-FF :

Clock	J	K	Q_{n+1}	
0	X	X	Q_n	
1	0	0	Q_n	→ HOLD state
1	0	1	0	→ RESET state
1	1	0	1	→ SET state
1	1	1	$\overline{Q}_n$	→ TOGGLE state

- Characteristic table :

J	K	Q_n	Q_{n+1}	
0	0	0	0	→ Q_n
0	0	1	1	
0	1	0	0	→ 0
0	1	1	0	
1	0	0	1	→ 1
1	0	1	1	
1	1	0	→ 1	→ $\overline{Q}_n$
1	1	1	→ 0	

- **Excitation table of J-k flip flop :**

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Scanned by CamScanner

- Characteristic equation of JK Flip-flop :

$$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$$

- The "Race-around condition" will occur when :

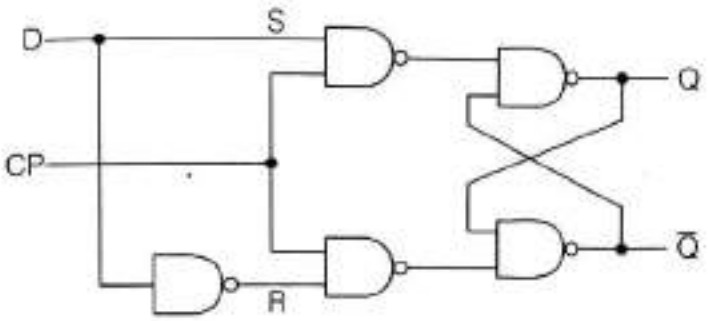
$$J = K = 1 \text{ and } t_{pd(FF)} < t_{pw}$$

- To avoid the "Race-around condition", we should maintain :

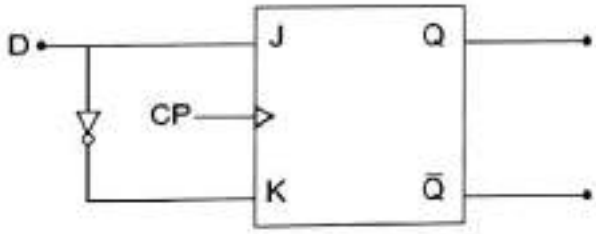
$$t_{pw} < t_{pd(FF)} < T$$

D-Flip-flop

- It is a FF with a delay equal to exactly one cycle of CLK :



- Graphical diagram :



$$J = D \text{ and } K = \bar{D}$$

- Truth table and Characteristic table :

CLK	D	Q_{n+1}
0	X	Q_n
1	0	0
1	1	1

(Truth table)

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

(Characteristic table)

Scanned by CamScanner

- Characteristic equation of D-Flip-flop :

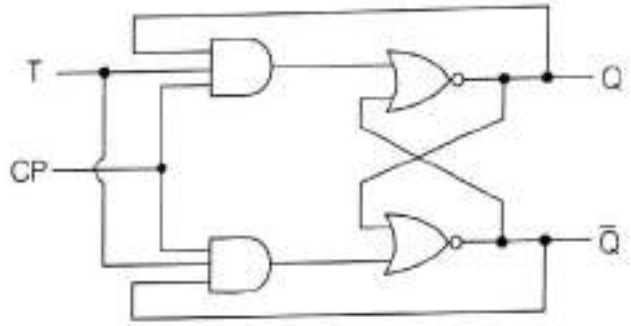
$$\boxed{Q_{n+1} = D}$$

- Excitation table of D-flip flop :

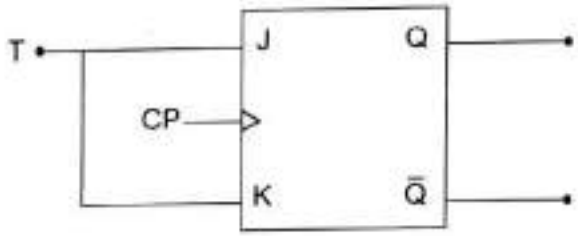
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Toggle Flip-Flop (T-FF)

- Logic diagram of T-FF :



- Graphical symbol of T-FF :



$$\boxed{J = K = T}$$

- Truth table and Characteristic table :

CLK	T	Q_{n+1}	
0	X	Q_n	
1	0	Q_n	→ HOLD state
1	1	$\bar{Q}_n$	→ RESET state

(Truth table)

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

(Characteristic table)

- Characteristic equation of T-Flip-flop :

$$\boxed{Q_{n+1} = \bar{T}\, Q_n + T\, \bar{Q}_n = T \oplus Q_n}$$

Scanned by CamScanner

- Excitation table of T-flip flop :

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	0
1	1	0

Conversion from one Flip-flop to another Flip-flop

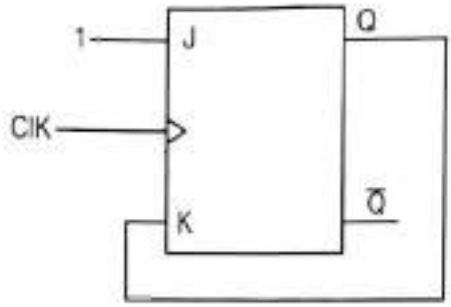
- SR-FF to JK-FF : $S = J\bar{Q}_n$ and $R = KQ_n$
- SR-FF to D-FF : $S = D$ and $R = \bar{D}$
- SR-FF to T-FF : $S = T\bar{Q}$ and $R = TQ$
- JK-FF to SR-FF : $J = S$ and $K = R$
- JK-FF to D-FF : $J = D$ and $K = \bar{D}$
- JK-FF to T-FF : $J = K = T$
- D-FF to SR-FF : $D = S + \bar{R}Q_n$
- D-FF to JK-FF : $D = J\bar{Q} + \bar{K}Q$
- D-FF to T-FF : $D = T \oplus Q$
- T-FF to JK-FF : $T = J\bar{Q} + KQ$
- T-FF to SR-FF : $T = S\bar{Q} + RQ$
- T-FF to D-FF : $T = D \oplus Q$

Note:

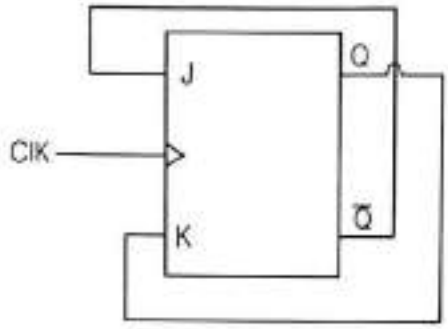
- Race-around condition occurs in JK-FF and T Flip-Flop to store 2-bits of information.
- Race-around condition always arises in "Asynchronous circuits".
- A Master-slave FF consists of an SR-FF followed by a T-FF.
- The frequency is always halved at the output of any FFs whose behaviour is same as TOGGLED-FF.

Scanned by CamScanner

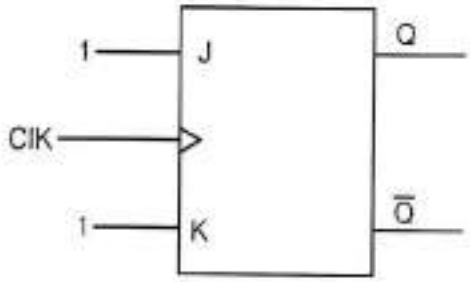
Toggle Mode of Operation

1. 

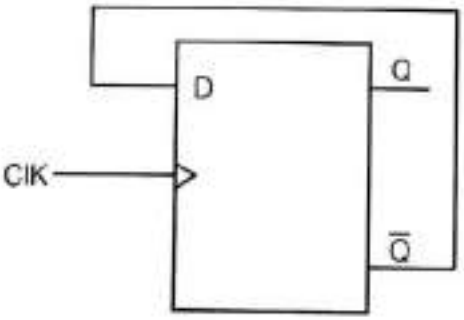
$Q(n+1) = \overline{Q}(n)$

2. 

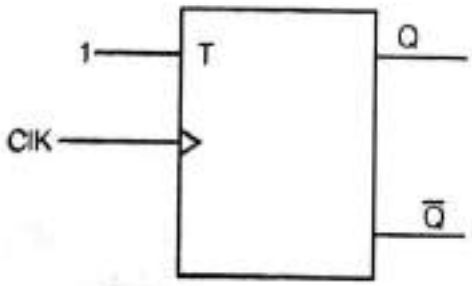
$Q(n+1) = \overline{Q}(n)$

3. 

$Q(n+1) = \overline{Q}(n)$

4. 

$Q(n+1) = \overline{Q}(n)$

5. 

$Q(n+1) = \overline{Q}(n)$

Scanned by CamScanner

Shift Registers

- In shift register each CLK PULSE shifts the content of register by one-bit to the RIGHT or LEFT.
- The "serial input" determines what goes into the left most flip-flop during the shift.

Serial-In Serial-Out (SISO)

4-bit Right-shift SISO Register

- In right shift SISO register, LSB data is applied at the MSB FF (D-FF).
- In 'n' bit register, to enter 'n' bit data, it requires 'n' clock pulses in serial form.
- If 'n' bit data is to be stored in SISO register then output to take serially for $(n-1)$ clock pulses are required.
- SISO register is used to provide 'n' clock pulse delay to the input data.
- If 'T' is the time period of clock pulse, then delay provided by SISO is nT.

4-bit Left-shift SISO Register

- In this above SISO register MSB data is applied to the LSB FF(D-FF).
- To enter the 'n' bit data in serial form we require 'n' clock pulses.
- To exit or getting output of 'n' bit data as serially we require $(n-1)$ clock pulses.

Serial-In Parallel-Out (SIPO)

- For n-bit serial input data to be stored the number of CLK pulses required = n.
- For n-bit-parallel output data to be stored the number of CLK pulses required = 0 (there is no need of CLK pulse).

Parallel-In Serial-Out (PISO)

- It stores parallel data. To store n bit number of CLK pulses required = 1 CLK pulse.
- To give serial out data number of CLK pulse required = $(n-1)$.

Parallel-In Parallel-Out (PIPO)

- For parallel in data the number of CLK pulses required = 1 CLK pulse.
- For parallel out data the number of CLK pulses required = 0 CLK pulse.
- **Time delay :** A SISO SRs may be used to introduce time delay "Δt" in digital signals

Scanned by CamScanner

$$\Delta t = N \times T = N \times \frac{1}{f_c}$$

where, N = Number of FFs

T = Time period of CLK pulse

f_c = CLK frequency

- The amount of delay can be controlled by the "f_c" or number of FFs in the SR.

Note:

- All shift registers are JK-FFs.
- "PIPO" register is a storage register made up with D-FFs.
- "PIPO" register is not a shift register.
- Each Shift Left operation multiply the data by 2 and each Shift Right operation divides the data by 2.
- To convert temporal code into spacial code, we use SIPO register. While to convert spacial code into temporal code we use PISO register.

Counters

- Depending on clock pulse applied counters are of two types :

Asynchronous Counters	Synchronous Counters
• Different FF's are applied with different clocks. • Slower • Fixed count sequence either up or down. • Decoding error.	• All FF's are applied with same clock. • Faster • Any count sequence is possible. • No decoding error.

MOD Counter

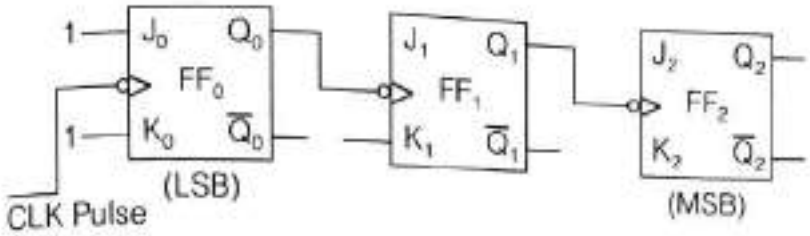
- The "MOD-number" indicates the number of states in counting sequence.
- For n-FFs, counter will have 2^n different states and then this Counter is said to be "MOD-2^n Counter".
- MOD number indicates the frequency division obtained from the Last FF.
- It would be capable of counting upto $(2^n - 1)$ before returning to zero state.

Scanned by CamScanner

Up/Down counter

- An "Up/Down Counter" can count in any direction depending upon the control input.

Ripple Counter



- In ripple counter with n-FFs there are 2^n possible states.
- With n-FFs the maximum count that can be counted by this counter is $2^n - 1$.

Disadvantage of Ripple Counter

- Decoding error is present due to propagation delay of FFs i.e. $t_{pd(FF)}$.
- For proper operation of the ripple counter :

$$T_{CLK} \geq nt_{pd(FF)} \quad ; \quad f_{CLK} \leq \frac{1}{nt_{pd(FF)}}$$

- Maximum CLK frequency : $f_{CLK,m} = \dfrac{1}{nt_{pd(FF)}}$

- For determination of Up/Down Counter :

Triggering with	CLK connection in	Access as
(–ve) edge	Q	UP Counter
(–ve) edge	$\overline{Q}$	Down Counter
(+ve) edge	Q	Down Counter
(+ve) edge	$\overline{Q}$	UP Counter

Note:

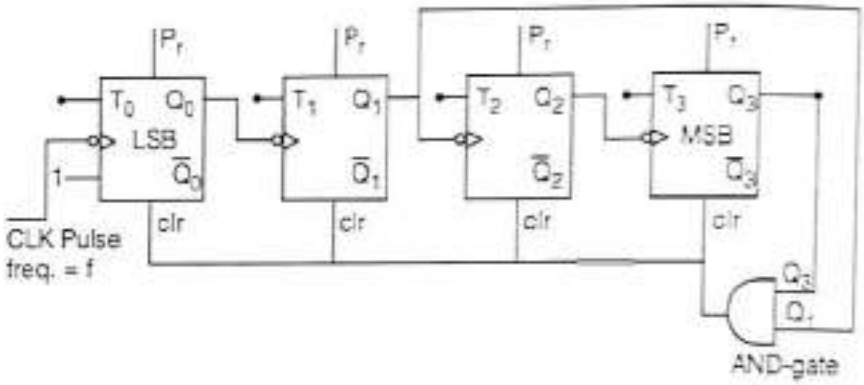
- In ripple counter flip flop applied with external clock will act as LSB bit.
- Clear and preset are known as asynchronous input to flip flop.
- If N = total number of states and n = number of FFs then $N \leq 2^n$ and $n \geq 3.32 \log_{10} N$.
- If $N = 2^n$, then we get BINARY COUNTER.
- If $N < 2^n$, then we get NON-BINARY COUNTER.

Scanned by CamScanner

- In "MOD-N Counter", if applied input frequency is 'f', then output frequency is f/N.
- If two counters are cascaded with MOD-M followed by MOD-N, the number of overall states of cascaded counter is $(M \times N)$ and counter is called "MOD-MN" counter.

Non-Binary Ripple Counter

Decade Counter or Mod-10 Counter

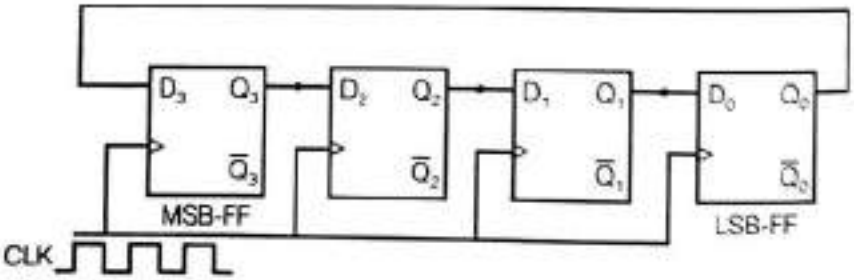


- Used states = 10 and unused states = 6
- For down counter, Mod Number = $2^n - N$.
- Output frequency of MOD-10 counter = $f/10$.

Synchronous (Parallel) Counters

NOT-Self Starting Ring Counter

- It is SISO shift register.



In 4-bit ring counter :

Used states = 4

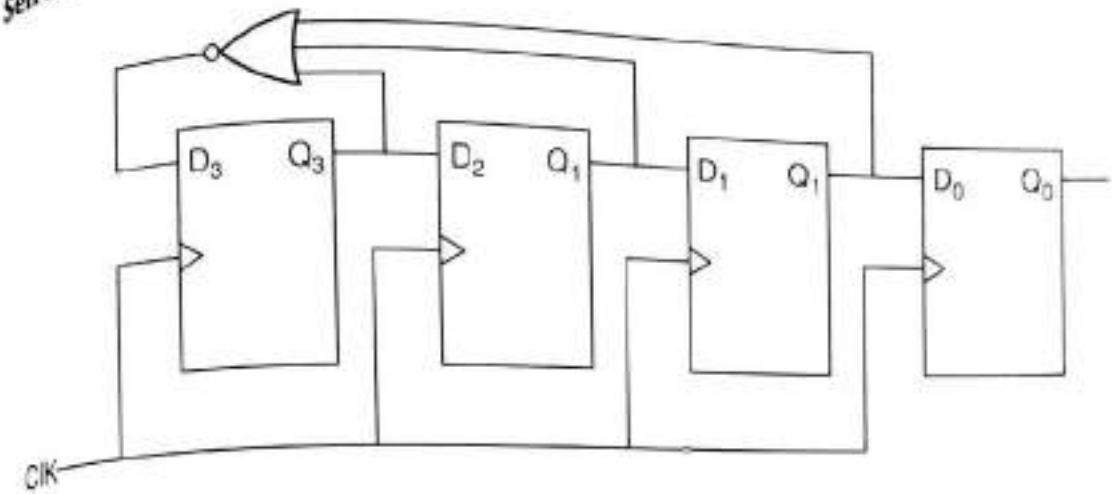
Unused states = $2^4 - 4 = 12$

- In any counter if CLK frequency is 'f' the FFs output frequency is 'f/N' (where N = Number of states).
- With n-FFs, there are n-states present in ring counter.

Scanned by CamScanner

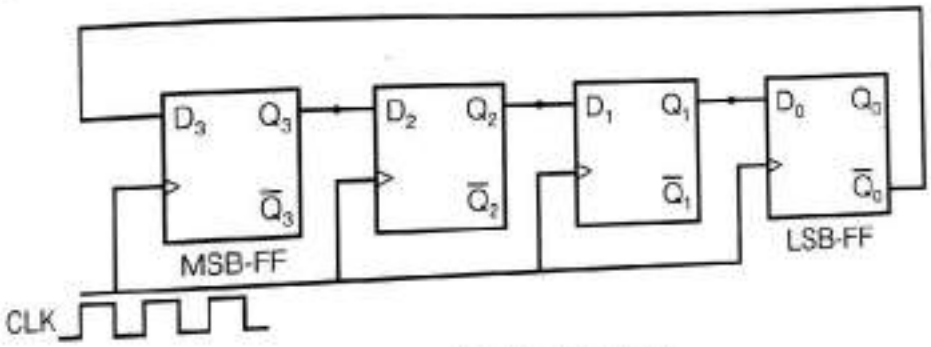
- With n-FFs, maximum count possible in ring counter is (2^{n-1}).
- Decoding is very easy in Ring counter, because there is no aid of extra circuit.

Self Starting Ring Counter



Twisted-Ring Counter

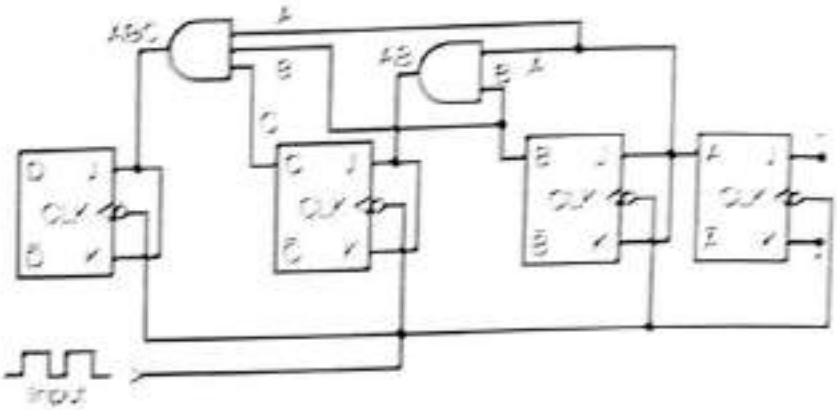
- Also known as Johnson Counter or Switch Tail Ring Counter



- With n-FFs there are $2n$ states in this counter.
- With n FFs the maximum count by this counter is $(2^n - 1)$.
- In normal "Johnson Counter" with n FFs and the input frequency is 'f' then output frequency of FFs is "$f/2n$"
- In a "Counter" if a feedback connection is used the number of possible states will decrease.
- In normal Johnson counter frequency at output of each flip is $f/2n$ and duty cycle is 50%.
- In Johnson counter to decode each state one two input AND or NOR gate is used.
- Lockout may occur when counter enter into unused state.

Scanned by CamScanner

Synchronous-Series Carry Counter



- Clock frequency : $f_{CLK} \le \dfrac{1}{t_{pd(FF)} + (n-2)\, t_{pd(AND\text{-}gate)}}$
- Total delay of this counter is much lower than an asynchronous counter with same number of FFs.

$$\text{Total delay} = FF\, t_{pd(FF)} + t_{pd(AND\ gate)}$$

Synchronous-Parallel Carry Counter

- It is the "Fastest Counter".
- Clock frequency : $f_{CLK} = \dfrac{1}{t_{pd(FF)} + t_{pd(AND\ gate)}}$

■■■

Scanned by CamScanner

Number Representation and Computer Arithmetic

5

Digital Number System

- A number system with base 'b' will have 'b' different digits from 0 to $(b-1)$.
- Number representation :

$$\boxed{(N)_b = d_{n-1}\, d_{n-2} \cdots d_i \cdots d_1 d_0 . d_{-1}\, d_{-2} \cdots d_{-f} \cdots d_{-m}}$$

where N = number,

b = base or radix,

$d_{n-1}\, d_{n-2} \ldots d_i \ldots d_1\, d_0$ represents integer portion of number $(N)_b$, $d_{-1}\, d_{-2} \ldots d_{-f} \ldots d_{-m}$ represents fraction portion of number and between these two there is a radical sign.

Weighted Number System

- Binary, octal, hexa decimal, BCD, 2421 etc.

Unweighted Number System

- Gray code, Excess 3-code

Number system	Base (b)	Digits
Binary	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexa decimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9 A, B, C, D, E, F

- In binary number system, a group of Four bits is known as "Nibble" and group of Eight bits is known as "Byte". (4 bits = 1 Nibble, 8 bits = 1 byte).

Codes

Binary Coded Decimal Code (BCD)

- Each digit of a decimal number is represented by binary equivalent.
- **In 4-bit binary formats :** Total number of possible representation = 2^4 = 16, Valid BCD codes = 10 and Invalid BCD codes = 6.

Scanned by CamScanner

- **In 8-bit binary formats** : Valid BCD codes = 100, Invalid BCD codes = 256 – 100 = 156
- BCD is also called 8421 code.
- During arithmetic operation if invalid BCD is present then add $(0110)_{BCD}$ to get correct result.
- It is a not a self complementing code.

Excess-3 Code

- It is a 4-bit code.
- It can be derived from BCD code by adding "3" to each coded number.
- It is a "self-complementing code".
- It is the only code which is unweighted and self complementing.
- 2421, 3321, 4311 and 5211 all are self complementing codes where sum of weight is 9.

Gray Code

- Also called "minimum change codes" in which only one bit in the code group changes when going from one step to the next.
- Gray code is also called cyclic code or reflective code. Since error is minimum so also called **minimum error code**.

Binary-to-Gray Conversion

- 'MSB' in the gray code is same as corresponding digit in binary number.
- Starting from "Left to Right", EXOR each adjacent pair of binary digits to get next gray code digit.

Gray-to-Binary Conversion

- "MSB" of Binary is same as that of gray code.
- If a number system has base b then we can find its b's complement and $(b-1)$'s complement.
- To determine $(b-1)$'s complement subtract given number from maximum number possible to the given base i.e. $(r^n - 1)$.
- To determine b's complement, first determine $(b-1)$'s complement then add 1 to get b's complement of number.
- To convert decimal number into any other base b divide integer part with b and multiply fractional part with b.
- If $(X_3 X_2 X_1 X_0)_b = (A)_{10}$ then $A = X_0 + X_1 b^1 + X_2 b^2 + X_3 b^3$

Scanned by CamScanner

Digital Number Representation

- In unsigned magnitude representation with 'n' bits, the possible integer values are [0 to $(2^n - 1)$].
- Extra bit is sign bit (MSB)
  if MSB = 0 → positive number
  if MSB = 1 → negative number
- In a sign magnitude representation the range of number.

$$\boxed{-(2^{n-1} - 1) \text{ to } +(2^{n-1} - 1)}$$

- In 1's complement representation, the positive number are represented similar to positive number in sign magnitude, but for representing negative number, first consider positive number and then take 1's complement of that.
- Range of 1's complement number

$$\boxed{-(2^{n-1} - 1) \text{ to } +(2^{n-1} - 1)}$$

- In 2's complement representation of a number positive numbers are represented similar to positive number in sign magnitude, but representing negative number first to write positive number then take 2's complement of it.
- Range of 2's complement representation number : $\boxed{-2^{n-1} \text{ to } +(2^{n-1} - 1)}$

Binary Arithmetic

- When both numbers have same sign then we add only magnitudes and use the sign as MSB.

1's Complement Addition

- When the numbers have different sign, keep one number as it is and 1's complement the other then we add magnitude only.
- If carry is present :
  (*i*) Add carry to LSB
  (*ii*) Sign of the result is sign of the uncomplemented number.
- If carry is not present :
  (*i*) Take 1's complement as the result.
  (*ii*) Sign of the result is sign of the complemented number.

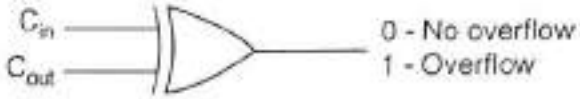
Scanned by CamScanner

2's Complement Addition

- In 2's complement method of representation if the number are of same sign then add the numbers and if carry is generated discard it and if carry is not generated take the 2's complement of results.

Over Flow

- Overflow occurs when two same signed numbers are added in signed representation. To detect overflow we use XOR gate whose inputs are C_{in} and C_{out} where C_{in} is input carry to MSB and C_{out} is output carry from MSB. If output of XOR gate is 1 then overflow, if it is zero then no overflow.



- In 2's complement representation there is only one representation for zero while in 1's complement representation there are two zeros + 0 and – 0.

Fixed Point and Floating Point Representation

- **Fixed Point :** Used for small range of values [Integers].
- **Floating Point :** Used for large range of values [Float/Real].

Fixed Point Representation

- **Fixed point unsigned integers:**

  Let the following n-bit number represent fixed point unsigned integer $(a_{n-1}\, a_{n-2}\, a_{n-3} \ldots a_1\, a_0)_2$

$$V = \sum_{i=0}^{n-1} a_i * 2^i = a_0 * 2^0 + a_1 * 2^1 + \ldots + a_{n-1} * 2^{n-1}; \quad a_i \in (0,\ 1)$$

  (*i*) $V \to V_{min}$, Iff $\forall i, a_i = 0 \Rightarrow V_{min} = 0$

  (*ii*) $V \to V_{max}$, Iff $\forall i, a_i = 1 \Rightarrow V_{max} = 2^0 + 2^1 + \ldots + 2^{n-1}$

  (*iii*) Range: $(0 \text{ to } 2^n - 1)$

- **Fixed Point Unsigned Fraction :**

  Let the following n-bit number represent unsigned fixed point fraction $(\cdot a_1\, a_2\, a_3 \ldots a_n)_2$

$$V = \sum_{i=1}^{n} a_i * 2^{-i}; \quad a_i \in (0,\ 1)$$

Scanned by CamScanner

(i) $V \to V_{min}$, Iff $\forall i, a_i = 0 \Rightarrow V_{min} = 0$

(ii) $V \to V_{max}$, Iff $\forall i, a_i = 1$

(iii) $\text{Error} = \frac{1}{2^{n+1}} + \frac{1}{2^{n+2}} + \ldots\infty = \frac{1}{2^n}\underbrace{\left[\frac{1}{2} + \frac{1}{2^2} + \ldots\infty\right]}$

(iv) Error $= 2^{-n} \equiv$ Weight of least significant bit of a fraction.

(v) Range: $(0 \text{ to } 1 - 2^{-n})$

(vi) The more the bits in the fraction, less the error, and more the accuracy (precision).

- **Fixed Point Signed Numbers:** The fixed point signed numbers are denoted using sign magnitude notation, 1's and 2's complement notation.

(i) **Signed Magnitude Form :**

- Sign bit is considered explicitly, hence additional hardware is required for resultant sign of arithmetic.
- Addition and substraction are performed on seperate hardware.
- 0 has 2 representations i.e., +0 : 0000 0000
  −0 : 1000 0000

(ii) **1's Complement Form :**

- Sign bit is not considered explicitly hence no additional hardware required.
- 0 has 2 representations i.e., +0 : 0000 0000
  −0 : 1111 1111
- Non weighted system.

(iii) **2's Complement Form :**

- Sign bit is not considered explicitly.
- Addition and subtractions are performed by using adder only.

$$-B = \bar{B} + 1$$
$$A - B = A + \bar{B} + 1$$

- 0 has only 1 representation.
- Only code that assigns negative weight to the sign bit.

Floating Point Representation

- The floating numbers are stored in mantissa exponent form. Any floating point number is represented in the form $\boxed{\pm m \times r^{\pm e}}$;

Scanned by CamScanner

where, m = mantissa/significant
e = exponent
r = radix/base

- Any floating point number is represented in "normalised" form.
- Zero cannot be normalized.
- In a normalised floating point number the MSB of mantissa must be non-zero.
- The value of expression is given by :
  (*i*) Explicit Normalization : $V = (-1)^{sign}\,[0.M]_2 \times 2^{E-\text{Bias}}$
  (*ii*) Implicit Normalization : $V = (-1)^{sign}\,[1.M]_2 \times 2^{E-\text{Bias}}$
- The biased exponent is an unsigned number which can represent signed exponent of original number.
- True Exponent = Biased Exponent – Bias

Sign	$E(K)$ Biased Exponent	Mantissa

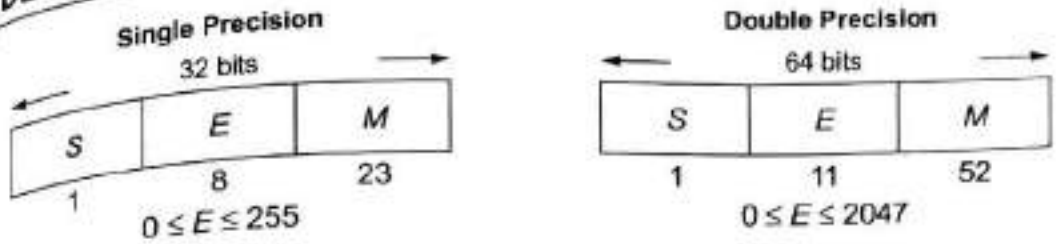
Bias = 2^{K-1}

$0 \le E \le 2^K - 1$, Bias = 2^{K-1}

- The IEEE 754 is concerned with floating point standard. Some of its features are:
  (*i*) The base of the system is 2.
  (*ii*) There is a provision for the values ±0 and ±∞.
  (*iii*) The floating point number is stored either with single precision or with double precision.
  (*iv*) Certain mantissa exponent combinations does not represent any number representing NAN.
  (*v*) The value represented in single precision
  (*vi*)

$S(1)$	$E(8)$	$M(23)$	Value
0/1	000....0 $E = 0$	000....0 $M = 0$	± 0
0/1	11....1 $E = 255$	00....0 $M = 0$	± ∞
0/1	11....1 $E = 255$	Not all zeros	NAN (Not A Number)
0/1	000....0 $E = 0$	$M \neq 0$	$(-1)^S (0.M)_2 \times 2^{-126}$
0/1	$1 \le E \le 254$	xxx.....x	$(-1)^S (1.M)_2 \times 2^{E-127}$

Scanned by CamScanner



Memory

- **Primary memories** : RAM and ROM.
- **Secondary memories** : Semi random and Serial access.

Semi Random Memory

Example : All disc memories (CD's DVD's).

Serial Access

Example : Magnetic tape, Magnetic bubble ferrite core, CCD (Charge coupled device).

Difference between RAM and ROM

RAM	ROM
• It is read/write memory.	• It read only memory. with decoders and encoders
• Random access (time required to access any memory is same)	• Random access.
• Volatile memory (temporary data storage)	• Permanent data storage.

Random Access Memory (RAM)

There are two types of RAM : **Static RAM** and **Dynamic RAM**.

Static RAM	Dynamic RAM
• Data is stored like flip-flop.	• Data is stored in MOS capacitor.
• BJT, MOSFET is used.	• MOSFET is used.
• Faster	• Slower
• Power dissipation is more.	• Power dissipation is low.
• Density is less.	• Density is high.
• Used as cache memory.	• Used as main memory.
• No refreshing is required.	• Refreshing is required.

Difference between Static RAM and Dynamic RAM

Scanned by CamScanner

Note:

- Size of ROM, m-function with n-variables implementation $= m \times 2^n$.
- Size of PLA, m-functions with n-variables each and p-product terms $= m \times n \times p$.
- Number of links required to implement PLA with x-inputs, p-product terms and y-outputs $= 2xp + yp + y$.

■■■

Scanned by CamScanner

A Handbook on
Computer Science

3

Computer Organization and Architecture

CONTENTS

1. Machine Instructions and Addressing Modes 116
2. ALU and Data-path, CPU Control Design 120
3. Memory and I/O Interfaces 127
4. Instruction Pipelining 131
5. Cache and Main Memory, Secondary Storage 138

Scanned by CamScanner

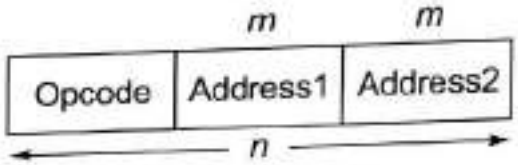
Machine Instructions and Addressing Modes

1

Machine Instructions

Types of instructions : Zero address instructions, one address instructions and two address instructions.

- Zero address instructions : NOP
- One address instructions : PUSH, POP
- Two address instructions : ADD, MULT, SUB



$$P \times 2^{2m} + Q \times 2^{2m} + R = 2^n \text{ (Total Instructions)}$$

2-address instructions, 1-address instructions, Zero Address Instructions

Addressing Modes

- Addressing modes shows the way where the required object is present. The object may be an instruction or data.
- Addressing Modes are basically classified in two types:
  - (a) **Sequential Control Flow Addressing Modes:** When the program is stored in the sequential memory locations, the program counter itself points the next instruction address, therefore such Addressing Modes are focused on data.
  - (b) **Transfer of Control Flow Addressing Modes:** When the program is stored in the random memory locations (using structured programming) there is a need of special instructions and addressing modes in order to calculate the next instruction address.

Sequential Control Flow Addressing Modes

1. **Register Based Addressing Modes:**
   - (*i*) These Addressing Modes are used to access the data when it is available in the registers.
   - (*ii*) **Register/Register Direct Addressing Mode:** This Addressing Mode is used to access the local variables. In this mode the data is present

Scanned by CamScanner

in the register, that register address is available in the address field of the instruction. Therefore, the effective address is equal to address field value. Data = [EA] = [Register Name]

Example: MOV R_1, R_2

2. **Memory Based Addressing Modes:** When the data is available in the memory, different memory based addressing modes are used to access the data. Under this, the EA is always the memory address.

(*i*) **Implied/Implicit Addressing Modes:** In this mode the data is available in the opcode itself. Therefore, there is no effective address.

Example: Compliment Accumulator (CMA) and all zero address instructions.

(*ii*) **Immediate Addressing Mode:**

- This mode is used to access the constants or to initialize registers to a constant value.
- Here the data is present in the address field of the instruction.
- The range of values initialized is limited by the size of the address field.
- If the address field size is *n*-bit, the possible range of immediate constants or data is: 0 to $2^n - 1$.

  Example: MOV A, 10

(*iii*) **Direct/Absolute Addressing Mode:**

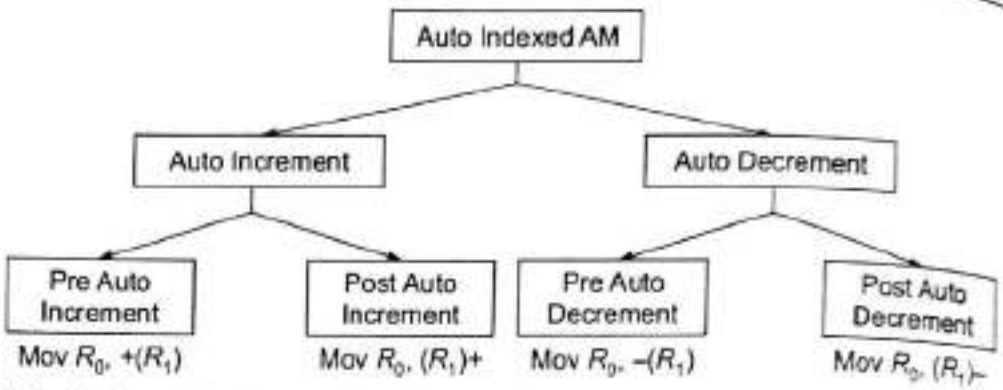
- Used to access the static variables.
- The data is present in the memory, that memory cell address is present in the address field of the instruction. Data = [EA] = [Memory Address]
- One memory reference is required to read or write the data by using the direct addressing mode.

  Example: MOV R_1, [1000]

(*iv*) **Auto Indexed Addressing Mode:**

- This mode is used to access the linear array elements. Thus, "base address" is required to access the data.
- The base address is maintained in the base register. Therefore the EA is Base Register ± Step Size.
- Step size is dependency on the amount of the data to be accessed from the memory. Data = [EA] = [[Base Register] ± Step size]

Scanned by CamScanner



(*v*) **Indirect Addressing Mode: (Array as parameter)**

- Used to implement the pointers. The EA is available in either register or memory.
- This mode is divided into two types:
  - (a) **Register Indirect Addressing Mode:** Here, the EA is present in the address register, that register name is available in the address field of the instruction.
    EA = [Address Field Value] ⇒ [Register Name]
  - (b) **Memory Indirect Addressing Mode:** Here, the EA is present in the memory, that memory address is available in the address field of the instruction.
    EA = [Address field value] ⇒ [Memory address]
    Data = [EA] = [[Memory address]]
- Two memory references are required in memory indirect Addressing Mode.

(*vi*) **Indexed Addressing Mode:**

- Allows to implement array indexing.
  EA = Base Address + Index Value
- The register that holds the index value is "Register Indexed Addressing Mode", it is used to access the Random array element.
- In Indirect Index Addressing Mode the base address is present in the memory, that memory address is present in the address field of the instruction.

Transfer of Control Flow Addressing Mode

- During the execution of selection statements (if, then, else, goto, switch), iterative statements (For loop, while loop, do while loop) and subprogram concept, the control is transferred from one location to another location.
- Transfer-of-Control operations can be implemented by using three possible mnemonics: (*i*) Jump (*ii*) Branch and (*iii*) Skip.

Scanned by CamScanner

- Transfer-of-Control instruction is divided into two types:
  - (i) **Unconditional Transfer-of-Control:** While execution of these instruction the program control is transferred to the target address without checking any condition. ***Example:*** HALT
  - (ii) **Conditional Transfer-of-Control:** While execution of these instruction the associated condition undergoes evaluation. If it evaluates to true, then the target address is loaded into Program Counter (PC) else no change in PC.

    The condition can be evaluated based on the status of the previous instruction.

    Example: They are used to implement 'if', 'switch', 'for', 'do while' and 'while' statements.
- Instruction set = opcode
- Vector interrupt – Interrupting source supplies branch info at processor through an interrupted vector.
- $\overline{INTA} \rightarrow INTR$

Computation of EA for the Next Instruction

Relative/PC Relative Addressing Mode

(Relocation data, Branch address at run time; Reduce instruction size)

- This Addressing Mode is used to access the instruction within the segment, therefore only the offset address is required.
- The offset address is available in the address field of the instruction.
- EA = Base Address + Offset Address
  - Base Address ↓ PC; Offset Address ↓ Address Field Value
- PC ← PC + IR [Address Field]
- Position dependent.
- Relocation at run time.

Base Register Addressing Mode (Position independent)

- This Addressing Mode is used to access the instruction between segments. Therefore base address as well as offset are required.
- Base address is maintained in the base register and offset address is maintained in the address field of the instruction.
- EA = [Base Register] + IR [Address Field]
- PC ← [Base Register] + IR [Address Field]

■■■

Scanned by CamScanner

ALU and Data-path, CPU Control Design

2

Introduction

A CPU divided into two sections : Data section and Control section. Data section also called as "**data path**" and Control section is "**control unit**".

Data path contains registers and ALU, which performs certain operations on data items. Control unit issues the control signals to the data path.

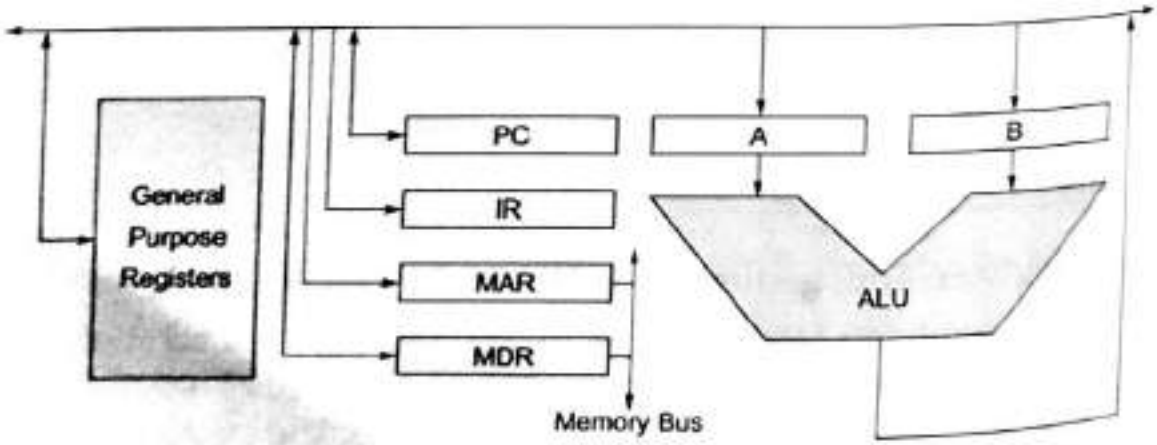
ALU and Data path

Types of data path : One bus data path, two bus data path and three bus data path.

In one bus data path, CPU registers and ALU use same bus for all incoming and outgoing data.

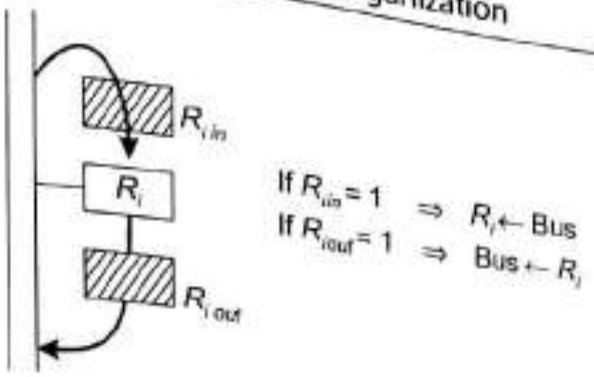
In two bus data path, general purpose registers are connected to two buses. Data can be transferred from two different registers to the input point of the ALU at the same time.

In three bus data path, two buses are used as source (moves data out of registers) while the third is used as destination (Moves data into registers). Source bus is also called as in-bus and destination bus is also called as out-bus.



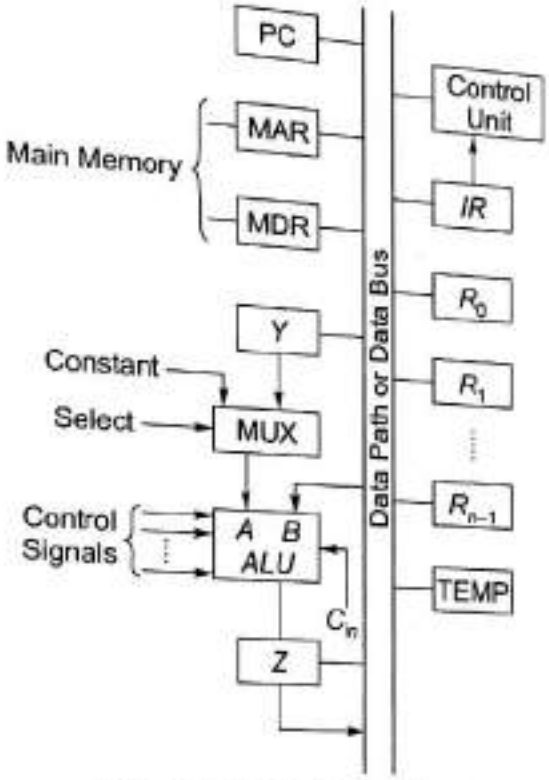
One-bus Datapath

Scanned by CamScanner



Data flow Into register and Out from register

- Data bus = Bi-directional



Processor with One Bus

- The ALU, registers and interconnecting path is called "ALU data path".
- Constant is used to increment PC.
- If select = 0; output of MUX is constant
  = 1; output of MUX is *Y*.
- **CPU Registers:**
  (*i*) **PC:** It is used to hold the starting instruction address and immediately point the next instruction's address.
  (*ii*) **IR:** It is used to hold the currently fetched instruction to decode. As instruction format it predefined in this register therefore it is not a user accessible register.

Scanned by CamScanner

(*iii*) **Accumulator:** It is used to hold the one of the ALU input and output.

(*iv*) **MAR (Memory Address Register):** It is used to carry the address. This register is directly connected to the address lines of the system bus.

(*v*) **MBR (Memory Buffer Register) or MDR (Memory Data Register):** It is used to carry the binary sequence. This register is directly connected to the data lines of the system bus.

- **Micro-operation:** It is a basic register to register transfer operation, also called as automatic operation/control mode/microinstruction.
  (*i*) All the micro-operations must complete the operation in one cycle.
  (*ii*) Control signals are required to execute the micro-operation.

Example: I_1 : $R_1 \longrightarrow R_2$

$$\underset{R_{1\,out}}{\boxed{R_1}} \longrightarrow \underset{R_{2\,in}}{\boxed{R_2}}$$

- **Microprogram:** The sequence of micro-operation is called as the microprogram.

  The fetch cycle microprogram is:

  T_1 : PC → MAR; PC_{out}, MAR_{in}
  T_2 : M[MAR] → MBR; MAR_{out}, MBR_{in}
  T_3 : MBR → IR; MBR_{out}, IR_{in}
  PC ← PC + Step size; PC_{out}, PC_{in}

- The basic operations performed are:

  (*i*) **Register transfer ($R_2 \leftarrow R_1$):**

  $R_{1\,out}$, $R_{2\,in}$ (only one clock cycle required)

  (*ii*) **ALU operation ($R_3 \leftarrow R_1 + R_2$):**

  $R_{1\,out}$, Y_{in}
  $R_{2\,out}$, Select = 1, Add
  Z_{out}, $R_{3\,in}$

  Minimum number of clocks = 3

  (*iii*) **If two data paths are used:**

  $R_{1\,out}$, Y_{in}, $R_{2\,out}$, Select = 1, Add
  Z_{out}, $R_{3\,in}$

  Minimum number of clocks = 2

  (*iv*) **Memory Read:**

  $R_2 \leftarrow M[(R_1)]$
  $R_{1\,out}$, MAR_{in}, read

Scanned by CamScanner

Wait for memory function to complete (WMFC).

MDR_{out}, $R_{2\,in}$

Here number of clocks can be 2 or 3 depending on WMFC.

(v) **Memory Write:**

$M[(R_1)] \leftarrow R_2$

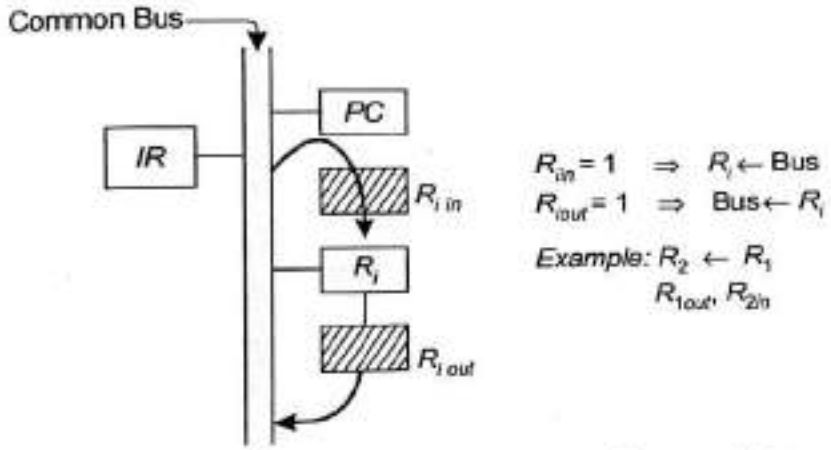
$R_{1\,out}$, MAR_{in}

$R_{2\,out}$, MDR_{in}, write

WMFC

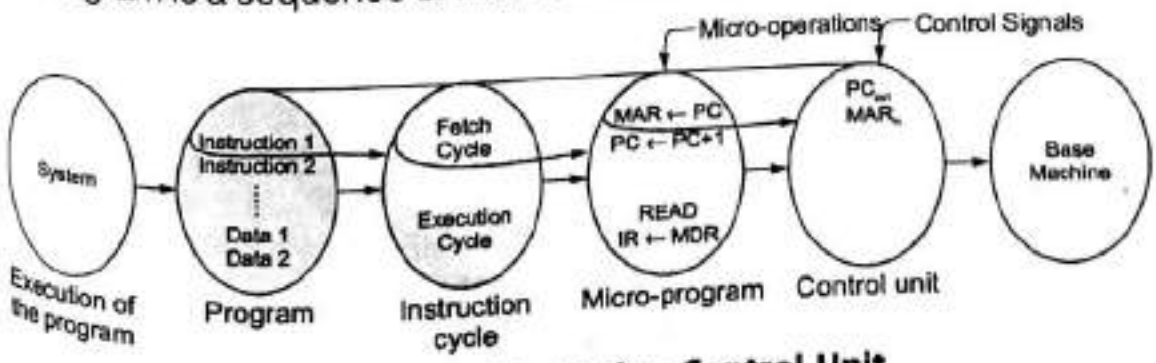
Minimum number of clocks = 3.

Control Unit



Data flow from one register to another register

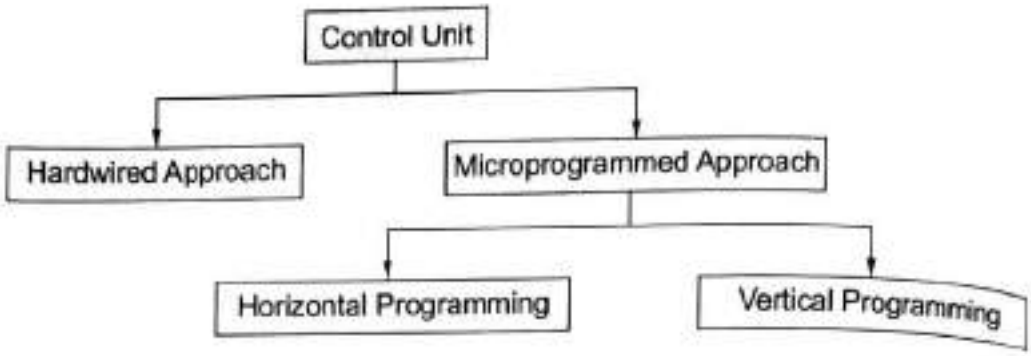
- The purpose of control unit is to provide appropriate timing and control signals which are required to execute micro-operations.
- Control signals are directly executed on the base machine.
- Micro-program contains sequence of micro-operations.
- Subcycles of instruction cycle invokes their own micro-program.
- Instruction cycle is used to execute one instruction.
- System functionality is execution of the program.
- Program is a sequence of instruction along with data.



Program Execution using Control Unit

Scanned by CamScanner

Control Unit Design



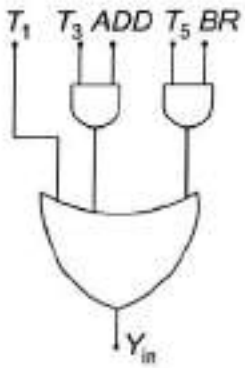
Hardwired Control Unit

1. The control signals are expressed as Sum-of-Product (SOP) expression and they are directly realized on the independent hardware.

   Example: $Y_{in} = T_1 + T_3 \cdot ADD + T_5 \cdot BR + \ldots$

   Here Y_{in} is enabled during T_1 for all instructions, during T_3 for *ADD* and so on...

   It can be realised as :



2. It is the fastest control unit design because of independent circuits.
3. For large number of instructions and control signals the complexity of hardware is more.
4. Relatively inflexible for any of the changes like modification, recorrection thus it is not used in design and testing places.
5. Used in the Real Time Applications.

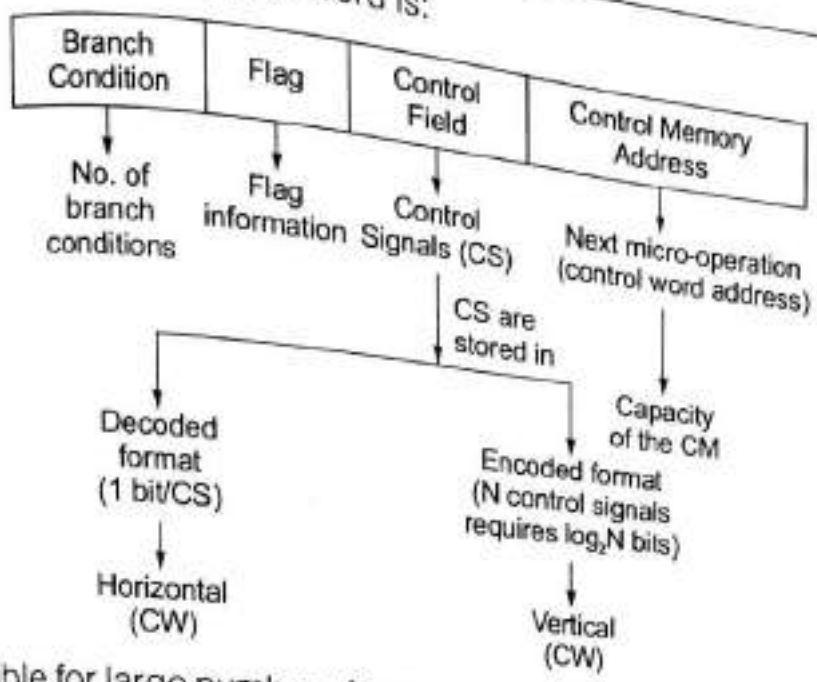
   Example: Air Craft simulation and weather forecasting etc.
6. Implemented in RISC processor.

Micro-programmed Control Unit

- Control Memory (CM) is used to store the microprogram.
- Control Memory (CM) is a permanent memory that is ROM.
- Microprogram contains sequence of microoperations (control word).

Scanned by CamScanner

- The format of the control word is:



- Flexible for large number of instructions and CS.
- Control signals are generated slow in comparison to hardwired.
- Based on the type of Control Word (CW) stored in the Control Memory (CM), the control unit is classified into two types:
  (*i*) Horizontal Microprogrammed Control Unit (H μPCU)
  (*ii*) Vertical Microprogrammed Control Unit (V μPCU)

Horizontal μPCU

1. The control signals are represented in the decoded binary format that is 1 bit/CS.
   Example: If 68 CS are present in processor then 68 bits are required. More than one CS can be enabled at a time.
2. It supports longer Control Word (CW).
3. It is used in parallel processing applications.
4. It allows high degree of parallelism. If degree is *n*, *n* CS are enabled at a time.
5. Requires no additional hardware (decoders) ⇒ Faster than vertical μPCU.

Vertical μPCU

1. The control signals are represented in the encoded binary format.
   For *N* control signals $\log_2 N$ bits required.
   Example: If $N = 68$ then $\log_2 68 = 7$ bits required.
2. It supports shorter Control Word (CW).
3. It supports easy implementation of new control signals therefore it is more flexible.

Scanned by CamScanner

4. It allows low degree of parallelism i.e. degree of parallelism is 1 or 0.
5. Requires an additional hardware (decoders) to generate control signals, it implies slower than horizontal μPCU.

Note:

- The ascending order of control unit w.r.t speed vertical < Horizontal < Hardwired (highest)
- The ascending order of control unit w.r.t flexibility (low flexibility) Hardwired < Horizontal < Vertical.
-

RISC	CISC
1. Reduced instruction set computer	1. Complex instruction set computer
2. Supports rich register set	2. Supports less number of registers
3. Supports less number of addressing modes	3. Supports more number of addressing modes
4. Supports fixed length instructions	4. Supports variable length instructions
5. One instruction/Cycle (CPI = 1)	5. (CPI ≠ 1)
6. Allows successful pipeline implementation (CPI = 1)	6. Supports unsuccessful pipeline (CPI ≠ 1)
7. *Example:* Motorola, Power PC, Advance Risk Machine	7. *Example:* Pentium

■■■

Scanned by CamScanner

Memory and I/O Interfaces

3

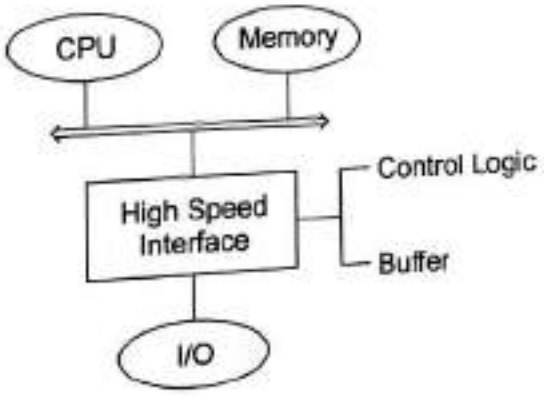
Introduction

In the isolated I/O CPU has distinct instructions for memory read/write and I/O read/write. The address what we use for accessing a device may be same as one of the address in the memory however separate lines are used to indicate I/O device address.

In memory mapped devices, their interface registers are memory mapped on to the address space of the memory.

Input Output Organisation

- Input output devices are very slow devices, therefore they are not directly connected with the system bus because Input output devices are electromagnetic devices and the CPU is the electronic device; there is difference in the operating mode, data transfer rate and word format.
- To synchronize the input output devices with the processor there is a need of high speed interface (I/O Module).



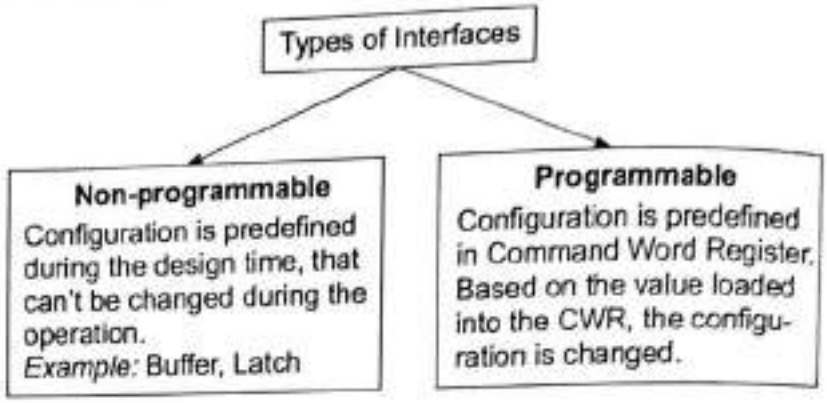
I/O Interface

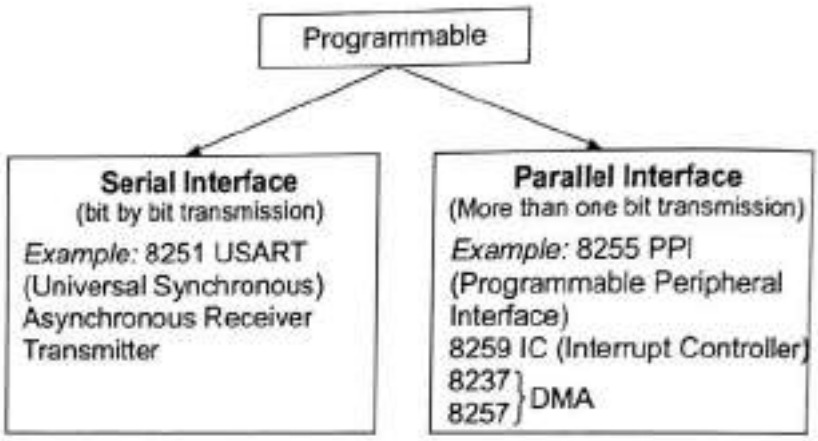
The high speed interface enables the corresponding device for the operation. After preparing the data, the I/O device transfers it to the buffer.

- When the data is present in the buffer than the high speed interface generates the signal to the CPU and waiting for the acknowledgment. After receiving the acknowledgment it transfers the data to the CPU with high rate. Thus, the speed gap is minimized.

Scanned by CamScanner

- Types of Interfaces:



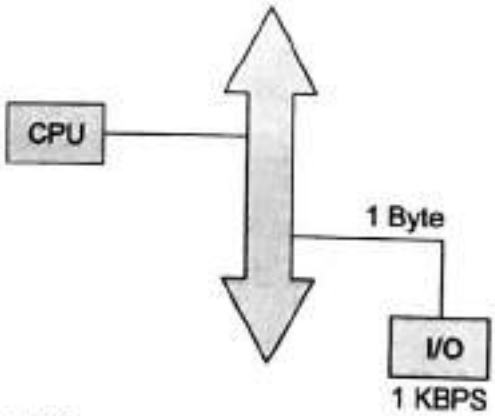


I/O Transfer Modes

Three different modes available to transfer data from I/O to CPU/memory.

1. Programmed I/O

- Here the CPU directly communicates with the I/O device. Hence, the processor utilization is inefficient.
- The processor undergoes waiting until completion of the I/O operation. This waiting depends on the speed of the I/O device.

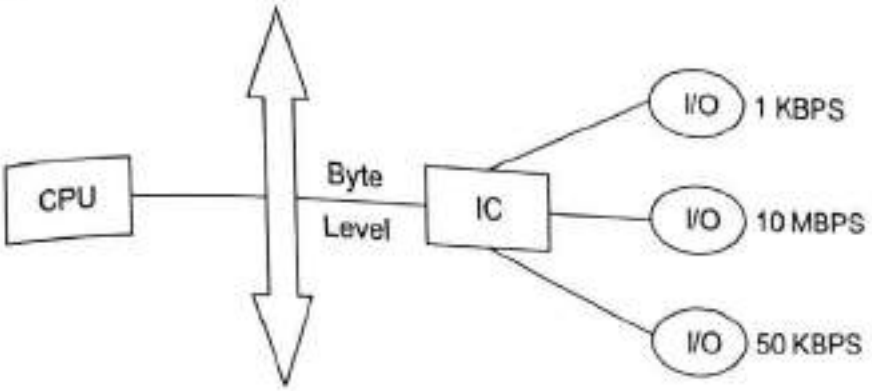


2. Interrupt Driven I/O

- Interrupt controller (IC) is used as the high speed interface between the basic I/O devices to CPU. Hence, the processor's utilization is good (efficient).

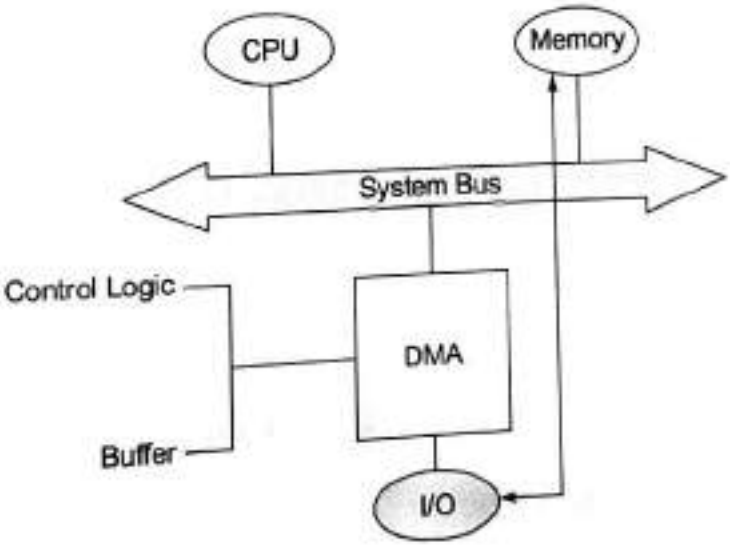
Scanned by CamScanner

- Here, the CPU is communicating with IC only. Thus, the execution/transfer time is not depending on the speed of I/O device, rather it depends on the latency of IC.



3. Direct Memory Access (DMA)

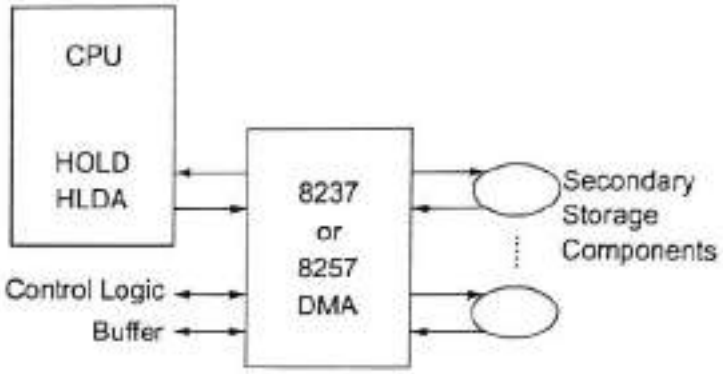
- The bulk amount of data is transferred from the I/O devices to the main memory without involvement of the CPU.
- The secondary storage devices are connected to the system bus through DMA, while execution of user program from the auxiliary memory to main memory the program is transferred page by page through the DMA.



- The CPU initializes the DMA along with source and destination address, control signals and count value. Later it is busy with some other execution.
- Based on priority - Daisy chain - non uniform priority to each device - polling.
- RFE is top privileged instruction.

Scanned by CamScanner

- The DMA control logic interprets the request and enables the corresponding device for the operation. After preparing the data device transfers it to the buffer.
- When the data is available in the buffer then DMA enables the HOLD signal, to gain the control of the bus and waiting for the acknowledgment. After receiving the HLDA signal, the DMA transfers the data from I/O device to main memory until the count becomes zero. After transfer operation, it establish connection to CPU.



- **During the DMA operation, the CPU is in two states :**
  (*i*) Busy State
  (*ii*) Block /Hold State
- Until preparing the data, the CPU is busy with other executions. While transferring the data, the CPU is in blocked state.
- Let 'X' = Preparation time (Depends on I/O speed),
  'Y' = Transfer time (Depends on main memory speed).

$$\text{Then, \% time CPU is blocked} = \left(\frac{Y}{X+Y}\right) \times 100$$

$$\text{\% time CPU is busy} = \left(\frac{X}{X+Y}\right) \times 100$$

- **DMA is operated in 3 modes:**
  (*i*) **Burst Mode:** After receiving the HLDA signal, bulk amount of data is transferred to main memory.
  (*ii*) **Cycle Stealing Mode:** Before receiving the HLDA signal, it forcefully suspend the CPU operation and transfers very important data to the main memory.
  (*iii*) **Block Mode:** After receiving the HLDA signal, the data is transferred to the main memory in block wise.

■■■

Scanned by CamScanner

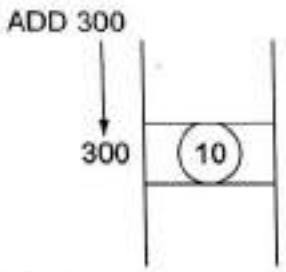
Instruction Pipelining

4

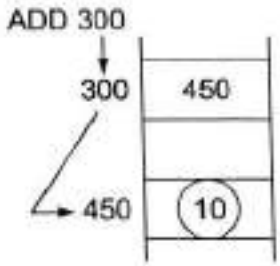
Introduction

The set of phases for an instruction execution cycle:

1. **Instruction Fetch (IF):**
   (*i*) MAR ← PC [MAR ← 2000]
   (*ii*) PC ← PC + 1 [Next instruction: 2004]
   (*iii*) READ CONTROL Signal is enabled
   (*iv*) IR ← MDR
2. **Instruction Decode (ID):** Two know the purpose of instruction.
3. **Operand Fetch (OF):**
   (*i*) Memory Read: MAR ← LOC X
   (*ii*) Write into any general purpose register: R_i ← MDR
4. **Execute and Store (EX):**
   (*i*) MAR ← LOC Z
   (*ii*) MDR ← R_i
   (*iii*) Write into memory.
5. **Indirect Phase:**



Here *EA* = 300 (Direct Address)



Here *EA* = 450 (Indirect Address)
(2 memory cycles required)

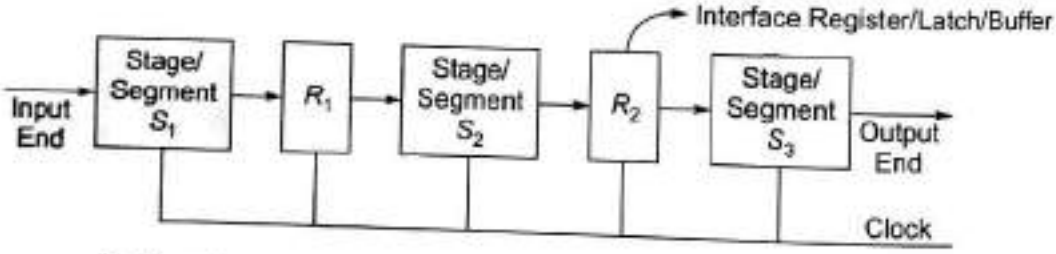
6. **Interrupt Phase:**
   (*i*) Each instruction execution involves a sequence of microoperations to be implemented by the processor.
   (*ii*) A microoperation is a register transfer operation, completes in 1 clock cycle.
   (*iii*) If *T* is the total time for instruction cycle that *T* will be divided into T_1, T_2, T_3 ..... among different phases.

Scanned by CamScanner

- The CPU is operated in 2 modes:
  (*i*) User mode or Non-privileged mode executes user program.
  (*ii*) System mode/Supervisor/Kernel/Privileged mode executes operating system to obtain system services.
- Types of interrupts:
  (*i*) **External or Hardware interrupt:** Araised due to timing and I/O devices.
  (*ii*) **Software Interrupt:** Araised due to switching from user to system mode or vice versa.
  (*iii*) **Internal Interrupt:** Araised due to incorrect use of instructions and data.
  Example: Division by zero register overflow invalid opcode.

Pipelining

- Accepting new input at one end before previously accepted input appears as an output at the other end is pipelining.
- Pipelining allows overlapping execution.
- The successful characteristic of the pipeline is for every new cycle, new input must be inserted into the pipeline.
  $\therefore$ CPI = 1



Types of Pipelines

1. **Linear Pipeline:** This pipeline is used to perform only one specific function.
2. **Non-linear pipeline:** This pipeline is used to perform the multiple functionalities. It uses feed forward and feed backward connections.
3. **Synchronous Pipeline:** On a common load/clock all the registers transfer data to the next stages simultaneously.
4. **Asynchronous Pipeline:** The data flow along the pipeline stages is controlled using handshake protocol.

Performance Evaluation of Pipeline Processor

- Consider '*K*' segment pipeline with clock cycle time t_p used to execute *n*-tasks. The first task is executed in the non-overlapping time span, so it requires *K* cycles to complete the operation.

Scanned by CamScanner

- The remaining $(n-1)$ tasks emerge from the pipe at the rate of one cycle per task, so $(n-1)$ tasks requires $(n-1)$ cycles to complete the operation. Thus, execution time of the K segment pipeline

$$ET_{pipe} = K + (n-1) \text{ cycles}$$

$$\boxed{ET_{pipe} = (K + (n-1)) \cdot t_p}$$

- The performance gain of the pipeline processor over the non-pipeline is:

$$\text{Speed up(S)} = \frac{\text{Performance}_{pipe}}{\text{Performance}_{nonpipe}} = \frac{\dfrac{1}{ET_{pipe}}}{\dfrac{1}{ET_{nonpipe}}} = \frac{ET_{nonpipe}}{ET_{pipe}}$$

$$\boxed{S = \frac{nt_n}{(K + (n-1))t_p}}$$

- For large number of tasks or instructions $K + (n-1)$ approaches to n

$$\boxed{S = \frac{nt_n}{nt_p}} \Rightarrow \boxed{S = \frac{t_n}{t_p}}$$

- When all the instructions are taking the same number of cycles then one instruction execution time is equal to the number of stages in the pipeline (K) i.e.,

$$\boxed{t_n = K_{t_p}} \Rightarrow \boxed{S = \frac{K \cdot t_p}{t_p}} \Rightarrow \boxed{S = K}$$

Instruction Pipeline

The instruction pipeline operates on a stream of instructions by overlapping the phases of instruction cycle like Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Execute (EX), Memory Access (MA), Write Back (WB) and so on.....

Stall Cycle

- A stall cycle is one during which no meaningful operation performed for an instruction.
- It arises due to:
  (i) Uneven clock cycles needed by each stage for instructions.
  (ii) Increased buffer overhead

Scanned by CamScanner

(*iii*) Memory operands

(*iv*) Pipelined dependencies

- $$S_{eff} = \frac{S_{ideal}}{(1 + \text{Stall frequency} \times \text{Stall cycles})} = \frac{K(\text{number of stage})}{(1 + \text{Stall frequency} \times \text{Stall cycles})}$$

- The maximum speedup that can be achieved by using a pipelined processor = number of stages.

$$S_{max} = S_{ideal} = K$$

- $$\text{Efficiency } E_K = \frac{S}{K} = \frac{n}{K + (n-1)}$$

- $$\text{Throughput} = \frac{\text{Number of tasks processed}}{\text{Total time required to process the tasks}}$$

$$\text{Throughput} = \frac{n}{(K + (n-1)) \cdot t_p}$$

Advance Pipeline (RISC Pipeline)

- In the RISC processor, instruction pipeline is implemented to execute the instruction.
- RISC processor supports three categories of instructions.

  (*i*) **Data Transfer:** Data transfer instruction can be implemented by using load and store

  Load r_0, $2(r_1)$; $r_0 \leftarrow M[2 + [r_1]]$

  Store $3(r_2)$, r_1; $M[3 + [r_2]] \leftarrow r_1$

  (*ii*) **Data Manipulation (ALU Operation):** ALU operation are directly performed on the registers.

  Add r_0, r_1, r_2; $r_0 \leftarrow r_1 + r_2$

  (*iii*) **Branch Operation:** The unconditional branch operation syntax is

  Jmp 1000; PC ← target address.

  Conditional Branch Operation

  JNZ r_0, 2000; $\begin{cases} \text{True ; PC} \leftarrow \text{target address} \\ \text{False ; P} \leftarrow \text{sequential address} \end{cases}$

- In RISC pipeline the branch instruction execution is completed at the end of the second stage. So branch penalty [2 – 1] = 1.

Scanned by CamScanner

- Register renaming can eliminate all WAR hazard and WAW.
- Bypass cannot handle all RAW hazard (Memory load instruction).
- Control hazard penalties can be eliminated by dnomic branch prediction.

Dependencies in the Pipeline

- Dependency causes stalls in the pipeline. Stall is a extra cycle created in the pipeline with NOP.
- There are three kinds of dependencies possible in the pipeline

(*i*) **Structural dependency:**

- This dependency is present because of resource conflict. The resource may be a memory or register or functional unit.
- Conflict is an unsuccessful operation, so to make it successful make the instruction wait until the resource become available. This waiting creates stalls in the pipeline.
- To minimize the number of stalls in the pipeline due to the structural dependency, "Renaming" concept is used.
- RAW = Register rename decrease.
- WAR = All decrease (removed).

(*ii*) **Data dependency:**

- Consider the program segment

  i : instruction

  j : instruction

  Data dependency exists between *i* and *j* when the "instruction *j* tries to read the data before instruction *i* writes it".

(*iii*) **Control dependency:**

- While execution of transfer of control operation the program control is transferred from the current location to the target location.
- If the current instruction is decoded as data transfer or data manipulation operation than the sequential instruction is a wanted instruction.
- The process of removing unwanted instruction from the pipeline is called as flush or freeze.

  Flush operation creates stalls in the pipeline.
- The number of stall cycles created in the pipeline due to branch operation is called as Branch Penalty.

Branch Penalty = 'At what stage the target address is available – 1

Scanned by CamScanner

- To reduce the number of stalls in the pipeline due to the branch operation, branch prediction buffer or loop buffer or branch target buffer is used.

  Branch target buffer: It is the high speed buffer maintained at the instruction fetch stage to store the predicted target addresses.

- The out-of-order execution creates two more dependencies in the pipeline
  1. **Anti Dependency:** It exists when instruction *j* tries to write the register before instruction *i* reads it.
  2. **Output Dependency:** It exists when instruction *j* tries to write the destination before instruction *i* writes it.

- To handle the Anti and Output dependencies "Register Renaming" concept is used.

  Register Renaming means use the temporary storage to hold the out-of-order execution output. After receiving the exception from the dependent instruction update the register file with temporary storage content.

Hazards

- Hazard is a delay. Delay creates extra cycles, these extra cycles without operation is called as stalls.
- **Hazards are classified as:**
  (*i*) Read-After-Write (RAW)
  (*ii*) Write-After-Read (WAR)
  (*iii*) Write-After-Write (WAW)
- Consider a program segment where the instruction *j* follows instruction *i* in the program order.
  (*i*) RAW: This hazard is created when instruction *j* tries to read the data before instruction *i* write it (TRUE DATA DEPENDENCY).
  (*ii*) WAR: This hazard is created when instruction *j* writes the data before instruction *i* reads it (ANTI DEPENDENCY).
  (*iii*) WAW: Instruction *j* writes the data before instruction *i* writes it (OUTPUT DEPENDENCY).

Performance Evaluation of the Pipeline with Stalls

$$S = \frac{CPI_{Nonpipe} \times \text{Cycle Time}_{Nonpipe}}{CPI_{pipe} \times \text{Cycle Time}_{pipe}} \quad \text{for ideal } CPI_{pipe} = 1$$

Scanned by CamScanner

$$S = \frac{CPI_{Nonpipe} \times \text{Cycle Time}_{Nonpipe}}{\left(\text{Ideal CPI} + \frac{\text{No. of stall cycles}}{\text{Instruction}}\right) \times \text{Cycle Time}_{pipe}}$$

- When there is no setup time overhead, then both the cycle times are equal.

$$S = \frac{CPI_{Nonpipe}}{1 + \left(\frac{\text{No. of stall cycles}}{\text{Instruction}}\right)}$$

- When all instructions are taking the same number of cycles then one instruction requires cycle equal to number of stages in pipeline to complete the execution.

$$S = \frac{\text{Pipeline Depth}}{1 + \frac{\text{No. of stall cycles}}{\text{Instruction}}}$$

- When efficiency is 100%, no stalls

$$S = \text{Pipeline depth}$$

- Instruction pipeline can create one or more stall, since register rename can eliminate it also. So always get false.

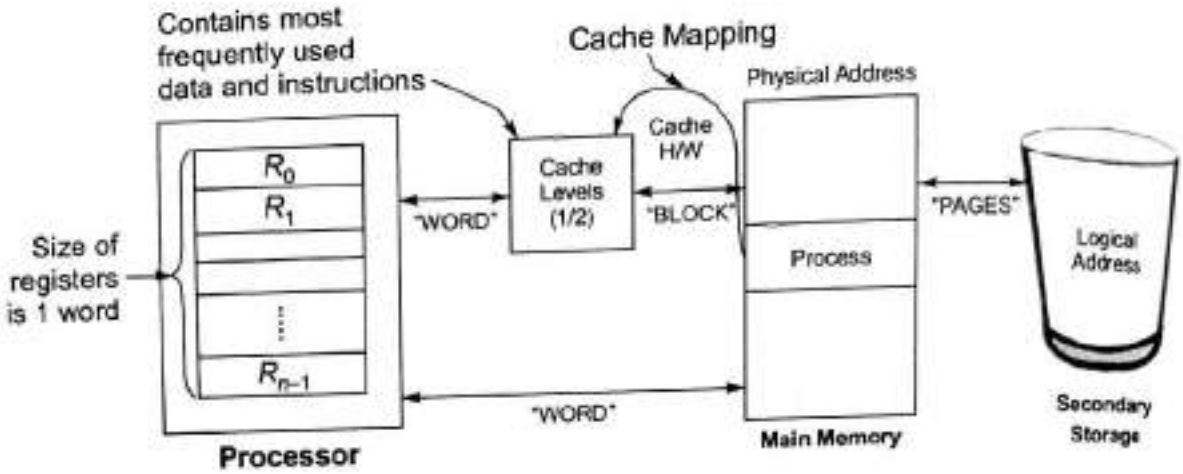
■■■

Scanned by CamScanner

Cache and Main Memory, Secondary Storage

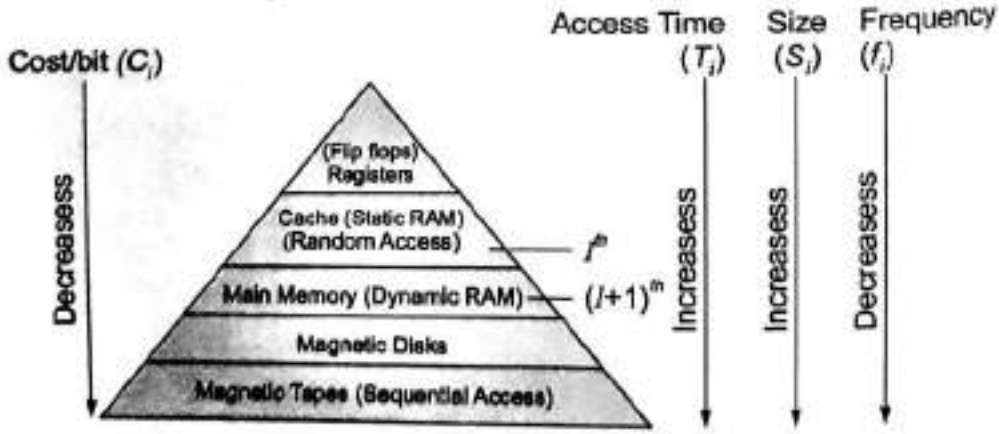
5

Memory Organisation



Memory Hierarchy

- The memory hierarchy shows the accessing order of the existing memory system.
- Average Memory Access Time = $\frac{\text{All Instruction Fetch Time + Write Time}}{\text{Total instruction}}$
- Smaller cache block → ↓ Cache miss penalties
  ↑ Larger TAG → ↑ Cache TAG overhead
  ↑ Spatial locality
- The purpose is to bridge the speed mismatch between fastest processor to the slowest memories at reasonable cost.



- When the processor refers to I^{th} level memory if it is found then "hit" else "miss".

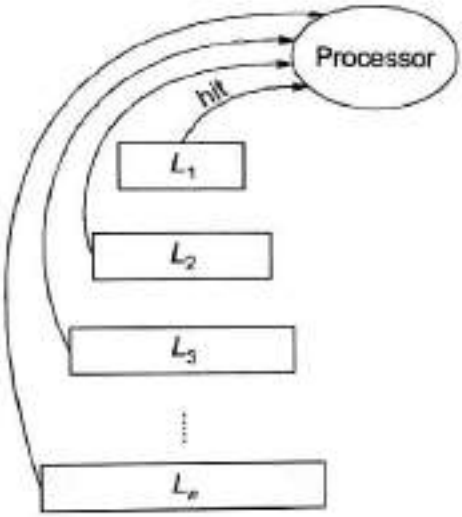
Scanned by CamScanner

Types of Memory Organisation

Based on the order of accessing the memory system, the memory organisation is divided into two types:

Simultaneous Access

- In this the CPU directly communicates with all the levels of memories.
- Consider a 'n' level memory system.



Let $H_1, H_2, \ldots, H_n$ be the hit ratios upto n level and $T_1, T_2, \ldots T_n$ be the access time upto n level memory system.

(*i*) The average access time of the memory is;

$$T_{avg} = H_1 T_1 + (1 - H_1) H_2 T_2 + \ldots + (1 - H_1)(1 - H_2)\ldots(1 - H_{n-1}) H_n T_n$$

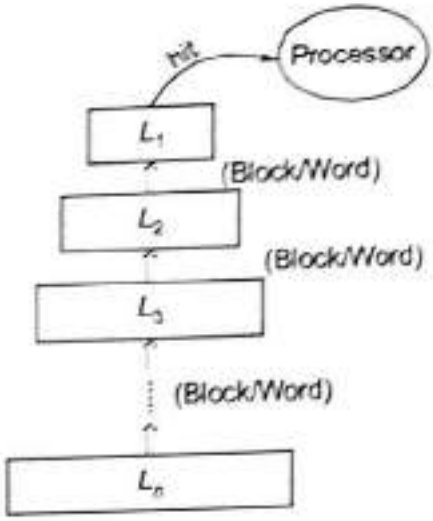
(*ii*) Throughput of the memory = $\dfrac{1}{T_{avg}}$ word/sec

(*iii*) C_{avg}/bit = $\dfrac{C_1 S_1 + C_2 S_2 + \ldots + C_n S_n}{S_1 + S_2 + \ldots + S_n}$

Hierarchial Access

- In this the data from 2nd level is transferred to 1st level and CPU accesses the data from 1st level.
- Consider a 'n' level memory system.

Scanned by CamScanner



- Let $H_1, H_2, ..., H_n$ be the hit ratios upto n level memory system and $T_1, T_2, ... T_n$ be the access time upto n level memory system.

$$T_{avg} = H_1T_1 + (1 - H_1)\,H_2\,(T_1 + T_2) + (1 - H_1)\,(1 - H_2)\,H_3\,(T_3 + T_2 + T_1) + ... + ((1 - H_1)(1 - H_2)...(1 - H_{n-1})H_n\,(T_n + T_{n-1} + ... + T_2 + T_1))$$

Cache Memory

- Cache is used as the intermediate memory between CPU and main memory.
- It is small and fast memory.
- By placing most frequently used data and instructions in cache, the average access time can be minimized.
- When miss occurs the processor directly obtains data from main memory and a copy of it is send to cache for future references.
- The cache memory works on the principle of Locality of Reference (LOR). LOR states that "the references to memory at any given interval of time tends to be confined to within localized areas of memory".
- **The performance of cache depends on:**
  - (*i*) Cache size (small)
  - (*ii*) Cache block size
  - (*iii*) Number of levels of cache
  - (*iv*) Cache mapping technique
  - (*v*) Cache replacement policy
  - (*vi*) Cache updation policy
- Cache is divided into equal parts called as cache blocks/cache lines.
- Data is transferred from main memory to cache in form of blocks, thus both are divided into blocks of equal size.

Scanned by CamScanner

- Cache line size = Main memory block size

  Number of cache lines = $\frac{\text{Cache size}}{\text{Block size}}$

Mapping Techniques

The process of transferring the data from the main memory to cache memory is called as mapping. There are three different mapping techniques:

1. Direct mapping
2. Associative mapping
3. Set associative mapping

Direct Mapping

- In this technique, mapping function is used to transfer the data from main memory to cache memory.

  The mapping function is $K \text{ Mod } N = i$

  where K = Main memory block number

  N = Number of cache lines

  i = Cache memory line number

- The address representation is:

← Physical address (in bits) →

Tag	Block	Word

- Accessing same block from different pages is always a miss. The replacement is done when "conflict miss" occurs.
- Hit ratio is less.
- The higher order bits of the address is compared simultaneously with each cache tag. If a match is not found it is miss. The delay of tag comparator is called hit latency or hit delay. So, the reference is forward to the main memory.
- The main memory control logic interprets the request into its known format.

Tag	Word

- The tag field is directly connected with the address logic of the main memory. Therefore, the corresponding block is enabled.
- By using the mapping function the main memory block is transferred to corresponding cache line along with the complete tag. Later the CPU accesses the data from the cache.

Scanned by CamScanner

- The number of bits in the tag is tag size or tag length.
  The maximum number of tag comparisons = 1.

Associative Mapping

- Any main memory block can be mapped to any location in cache. Thus, the cache address is not in use.
- The data is transferred from main memory to cache along with the complete tag. The address representation is

← Physical address (in bits) →

Tag	Word

- The replacement is done only when the cache is full or when "capacity miss" occurs.
- The block number is updated in the tag field. The higher order bits of the address compared sequentially with each cache tag, if a match is not found then it is "miss".
- The maximum number of tag comparisons = N (Number of cache blocks)
- The complexity of tag comparator is more.
- Cache size = Tag memory size + Data memory size
  Data memory size = Number of cache lines × Number of bits in the line

Set Associative Mapping

- In this mapping the cache memory is organised into sets, each set is capable to hold multiple main memory blocks.

$$\text{Number of sets} = \frac{N}{\text{P-way}}$$

where N = Number of cache lines
P = Number of MM blocks possible on each cache line

- The address interpretation is:

← Physical address (in bits) →

Tag	Set offset	Word offset

- The mapping function used is

$$K \bmod S = i$$

where, K = Main memory block number
S = Number of cache sets
i = Cache memory set number

- The replacement is done when the set is full.

Scanned by CamScanner

- Hit ratio and complexity of tag comparator is optimal.
- Maximum number of tag comparisons = P.
- When P = 1 set associative is direct mapping.
- There is a need of multiplexers to compare the existing tags in the set one by one with respect to CPU generated tag.
- Tag memory size = Number of sets in the cache × Number of blocks in the set × Number of tag bits in each block

Compulsory, Capacity and Conflict Misses

- Compulsory misses are caused by the first reference to a location in memory. Cache size and associatively makes no difference to the number of compulsory misses. Pre-fetching can help here, as can larger cache block sizes (which are a form of pre-fetching).
- **Capacity misses** are those misses that occur regardless of associativity or block size, solely due to the finite size of the cache.
- **Conflict misses** are those misses that could have been avoided, had the cache not evicted an entry earlier.
- **Double the associativity :**
  (*i*) Capacity and line size constant.
  (*ii*) Halves the number of sets.
  (*iii*) Decrease conflict misses, no effect on compulsory and capacity misses.
  (*iv*) Typically higher associativity reduces conflict misses because there are more places to put the same element.
- **Halving the line size:**
  (*i*) Associativity and number of sets constant.
  (*ii*) Halves the capacity.
  (*iii*) Increases compulsory and capacity misses but no effect on conflict misses.
  (*iv*) Shorter lines mean less "pre-fetching" for shorter lines. It reduces the cache's ability to exploit spatial locality. Capacity has been cut in half.
- **Doubling the number of sets:**
  (*i*) Capacity and line size constant.
  (*ii*) Halves the associativity.
  (*iii*) Increases conflict misses but no effect on compulsory and capacity misses.
  (*iv*) Associativity is less.

Scanned by CamScanner

Replacement Algorithms

When the cache is full, the existing block is replaced with new block. There are two replacement algorithms used

(a) **FIFO (First-In-First-Out):** In replaces the cache block with new block which is having longest time stamp.

(b) **LRU (Least Recently Used):** It replaces the cache block with new block which is having less number of references with longest time stamp.

Updating Techniques

Coherence: The same address contains different values at different places (cache is updated while main memory is not updated). This property is called "coherence" which creates loss of data.

There are two updating techniques used: Write through and write back.

Write Through

In this technique, the CPU performs the simultaneous updation in the cache memory and main memory. Thus, there is no coherence.

- Word updation (T_W) Time = max $\left(\begin{array}{c|c}\text{Word updation} & \text{Word updation time} \\ \text{time in cache} & \text{in main memory}\end{array}\right)$

- $$T_{avg_{read}} = \underset{\substack{\downarrow \\ \text{read} \\ \text{hit}}}{H_r} \; \underset{\substack{\downarrow \\ \text{read} \\ \text{data}}}{T_c} + \underset{\substack{\downarrow \\ \text{read} \\ \text{miss}}}{(1 - H_r)} + \underset{\substack{\downarrow \\ \text{read} \\ \text{allocate}}}{(T_m} + \underset{\substack{\downarrow \\ \text{read} \\ \text{data}}}{T_c)}$$

- $$T_{avg_{write}} = \underset{\substack{\downarrow \\ \text{write} \\ \text{hit}}}{H_w} \; \underset{\substack{\downarrow \\ \text{word} \\ \text{updation} \\ \text{time}}}{T_w} + \underset{\substack{\downarrow \\ \text{write} \\ \text{miss}}}{(1 - H_w)} + \underset{\substack{\downarrow \\ \text{write} \\ \text{allocate}}}{(T_m} + \underset{\substack{\downarrow \\ \text{word} \\ \text{updation} \\ \text{time}}}{T_w)}$$

- Total access time of memory, when both read and write cycle is considered is:

  $$T_{avg} = f_{read} \times T_{avg_{read}} + f_{write} \times T_{avg_{write}}$$

- Throughput $\eta_{WT_{cache}} = \dfrac{1}{T_{avg_{WT}}}$ word/sec

- The hit ratio for the write operation is always 1,

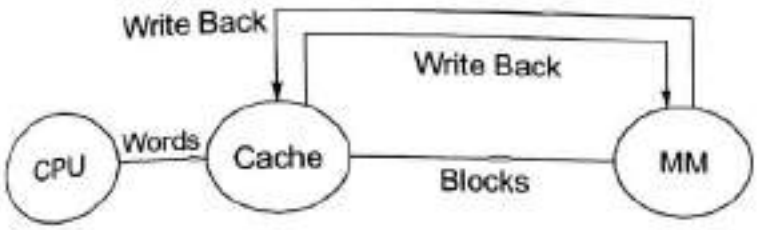
- $T_{avg_{write}} = H_w T_m$; where T_m = Main memory access time

Scanned by CamScanner

Write Back

The CPU performs the write operation only on the cache, still coherence is present but that doesn't create problem because before replacing the cache block with new block the updation are taking back into the main memory.

- Each cache line contains one extra bit, called modified bit (update bit), when the block is updated the corresponding modified bit is set.

- 

Modified bit → 1 (Dirty bit): [Hierarchical: $(T_{read} = T_{write})$] $[T_{read} = H.\ T_c + (1 - H)\ (T_{cache} + T_{mem\text{-}block} + \%(\text{dirty})\ .\ T_{mem\text{-}block})]$

Modified bit → 0 (Clean bit): [Simultaneous: $T_{read} = H.\ T_c + (1 - H)\ (T_m + \%(\text{dirty}) - T_{mem\text{-}block})]$

- $T_{avg_{read}} = \underset{\text{read hit}}{H_r}\ \underset{\text{read data}}{T_c} + \underset{\text{read miss}}{(1 - H_r)}\ [\%\ \text{dirty bits}\ (\underset{\text{write back}}{T_m} + \underset{\text{read allocate}}{T_m} + \underset{\text{read data}}{T_c}) + \%\ \text{clean bits}\ (\underset{\text{read allocate}}{T_m} + \underset{\text{read data}}{T_c})]$

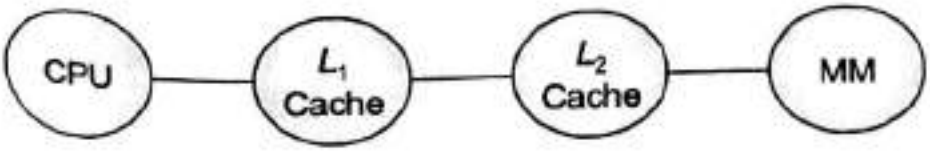
- $T_{avg_{write}} = H_w T_c + (1 - H_w) + [\%\ \text{dirty bits}\ (\underset{\text{write back}}{T_m} + \underset{\text{write allocate}}{T_m} + \underset{\text{write data}}{T_c}) + \%\ \text{clean bits}\ (\underset{\text{write allocate}}{T_m} + \underset{\text{write data}}{T_c})]$

- $T_{avg_{write\ back}} = f_{read} \times T_{avg_{read}} + f_{write} \times T_{avg_{write}}$

- Throughput $\eta_{write\ back} = \dfrac{1}{T_{avg_{write\ back}}}$ word/sec

Multilevel Caches

- To reduce the miss penalty, multilevel caches are used in the system design.



Scanned by CamScanner

- In multilevel caches, two kinds of miss rates will be calculated

  (*i*) **Local miss rate:**

$$\text{Local miss rate} = \frac{\text{No. of misses in the cache}}{\text{Total No. of references to that cache}}$$

  (*ii*) **Global miss rate:**

$$\text{Global miss rate} = \frac{\text{No. of misses in the cache}}{\text{Total No. of CPU generated references}}$$

- Average access time can be calculated as

$$T_{avg} = \text{Hit time}_{L_1} + \text{Miss rate}_{L_1}(\text{Local miss rate}) \times \text{Miss penalty}_{L_1}$$

$$\text{Miss penalty}_{L_1} = \text{Hit time}_{L_2} + \text{Miss rate}_{L_2} \times \text{Miss penalty}_{L_2}$$

$$\text{Miss penalty}_{L_2} = \text{MM Access time}$$

- Average memory stall cycles = (No. of miss$_{L1}$/Instruction × Hit time$_{L2}$)+ (No. of misses$_{L2}$/Instruction × Miss penalty$_{L2}$)

Note:

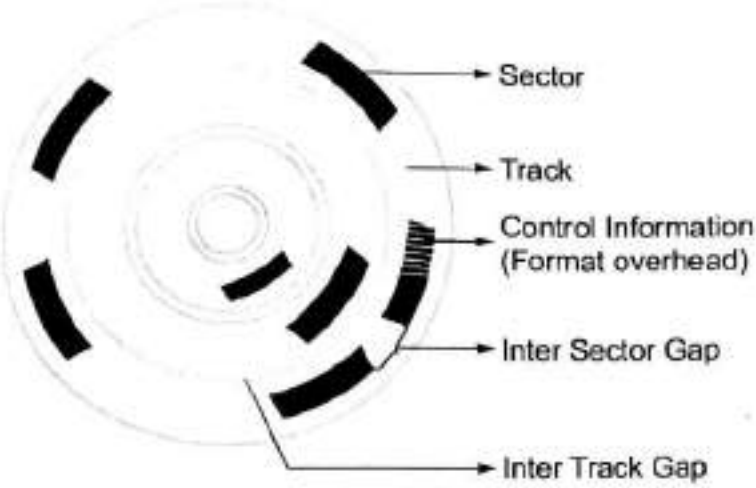
- There are three types of caches based on whether the index or tag correspond to physical or virtual addresses.
- **Physically indexed, physically tagged:** Caches use the physical address for both the index and the tag.
- **Virtually indexed, physically tagged:** Caches use the virtual address for the index and the physical address in the tag.
- **Virtually indexed, virtually tagged:** Caches use the virtual address for both the index and the tag.

Secondary Memory

- It is often required to have larger programs which exceeds the size of main memory.

  Example: Operating System.
- A magnetic disk is a thin circular metal plate, the data on the disk surface is organised as

Scanned by CamScanner



- A set of concentric circles is called "tracks".
- Each track holds same number of manageable units called sectors. The universal size of a sector is 512 Bytes.
  The basic unit of transfer from the disk is a "sector".
- The disk space without control information is called formatted disk space.
- The disk is a semi random access memory and data is distributed across circular tracks and sectors. Two types of disk construction:
  (*i*) **Constant Track Capacity Disk:** In this case variable recording density is used across all the tracks and the disk has to move with angular velocity.
  (*ii*) **Variable Track Capacity Disk:** In this case constant recording density is used.
- Each rotation of the disk covers 1 track.
- The data transfer rate depends on rpm.
- **Seek Time (t_s):** The time to move Read/Write (R/W) head to the desired track.
- **Rotational Latency (t_r):** The time to move Read/Write (R/W) head to the desired sector beginning.

By default $t_r = \frac{1}{2} \times \text{rotation time}$

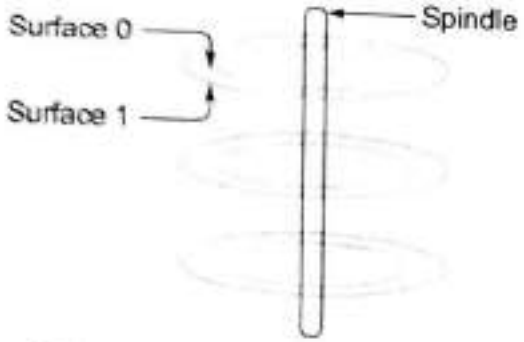
$$t = \text{access time of the disc} = t_s + t_r$$

$$t_{avg} = t_s + t_r + t_{\text{data transfer}} + t_{\text{overhead}}$$

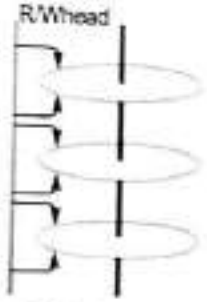
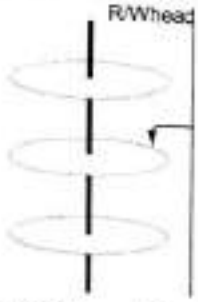
- The storage density **ρ = Number of Bytes/cm**
  ρ is maximum at innermost track.

Scanned by CamScanner

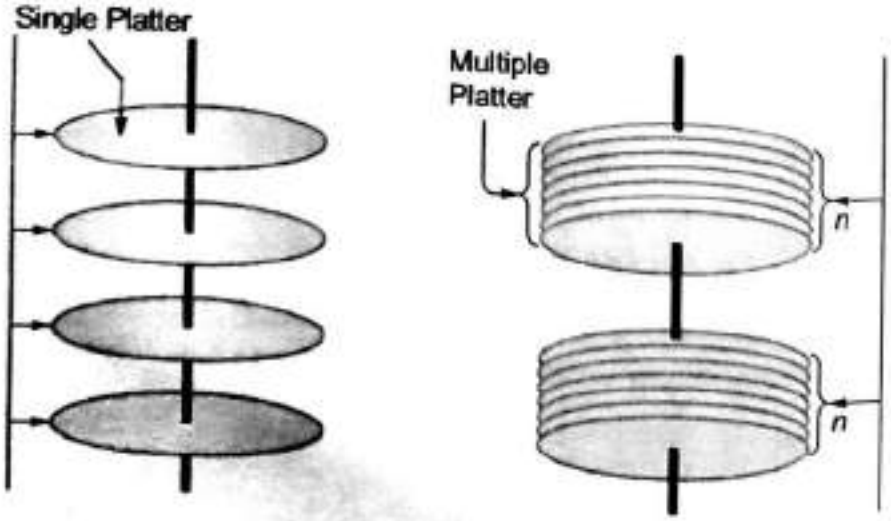
- The data transfer rate of the disc
  D = Number of Bytes/sec
- Multiple disks can be organised as one above another.



- **Characteristics of disks:**
  (*i*) **Single sided/Double sided disks:** By default double sided disks are used, in which recording is done on both the surfaces.
  (*ii*) Fixed Read/Write (R/W) head or Movable R/W head

Fixed R/W Head	Movable R/W Head
	
(a) One R/W head per surface (b) Each rotation covers 'n' tracks n = no. of surfaces (c) Hardware increases and hence cost is more	(a) One R/W head for all surface (b) Each rotation covers 1 track (c) Less hardware required but access time increases

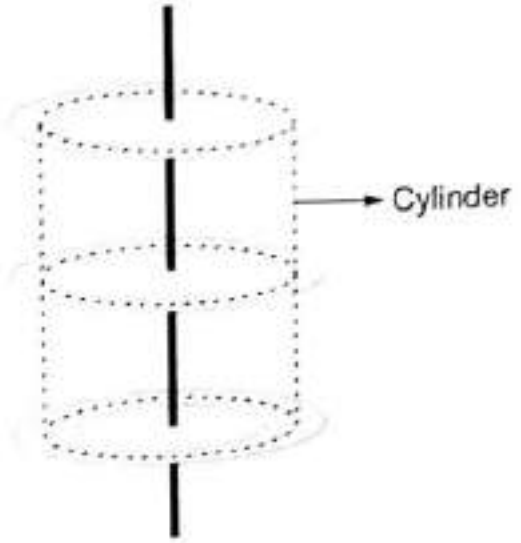
(*iii*) Single platter (or) Multiple platter disks



Scanned by CamScanner

- A vertical set of all the tracks from the same position in a disk pack forms a cylinder.

Number of cylinders = Number of tracks.



Note:

- Capacity = Track × Sectors/Track × Bytes/Sector.
- $\text{Capacity}_{\text{formatted}}$ = Tracks × Sectors/Track × (Bytes/Sector – Format overhead).

■■■

Scanned by CamScanner

A Handbook on
Computer Science

4

Programming and Data Structures

CONTENTS

1. Programming in C 151
2. Functions and Recursion 158
3. Abstract Data Types, Arrays and Linked Lists 161
4. Stacks and Queues 165
5. Trees, BSTs and Binary Heaps 168

Scanned by CamScanner

Programming in C

1

- **Variable:** A variable in simple terms is a storage place which has some memory allocated to it.
  - (*i*) **Variable declaration** refers to the part where a variable is first declared or introduced before its first use.
  - (*ii*) **Variable definition** is the part where the variable is assigned a memory location and a value.
  - (*iii*) Mostly variable declaration and definition are done together.
- **const (Constant Variable):** Constant variables are variables which when initialized, can't change their value (the value assigned to them is un-modifiable).
  - (*i*) Constant variables need to be initialized during their declaration itself.
  - (*ii*) const keyword is also used with pointers.

1. **int *ptr; (Pointer to Variable)** We can change the value of ptr and we can also change the value of object ptr is pointing to Pointer and value pointed by pointer both are stored in read-write area.
2. **const int *ptr; or int const *ptr;** : We can change pointer to point to any other integer variable, but cannot change value of object pointed using pointer ptr.
   - (*i*) Pointer is stored in read-write area.
   - (*ii*) Object pointed may be in read only or read write area.
3. **int *const ptr;** : Constant pointer to integer variable, means we can change value of object pointed by pointer, but cannot change the pointer to point another variable.
4. **const int *const ptr;** : Constant pointer to constant variable which means we cannot change value pointed by pointer as well as we cannot point the pointer to other variable.

- **Extern:** Extern indicate that the variable is already defined don't create memory again, use previous memory.
  - (*i*) By default, the declaration and definition of a C function have "extern" prepended with them.
  - (*ii*) Declaration can be done any number of times but definition only once.

Scanned by CamScanner

(*iii*) "extern" keyword is used to extend the visibility of variables/functions.

(*iv*) When extern is used with a variable, it's only declared not defined.

(*v*) When an extern variable is declared with initialization, it is taken as definition of the variable as well.

- **Static:** Static variables preserve the value of their last use in their scope.
  (*i*) These are initialized only once and exist till the termination of the program (memory is allocated only once compile time itself).
  (*ii*) Their scope is local to the function to which they were defined.
  (*iii*) Global static variables can be accessed anywhere in the program.
  (*iv*) By default, they are assigned the value 0 by the compiler.
  (*v*) By default values : if it has pointer type, it is initialized to a NULL pointer; if it has arithmetic type, it is initialized to (positive or unsigned) zero.
- **Void:** void is a special data type (means an empty data type).
  (*i*) A void pointer is a pointer that has no associated data type with it.
  (*ii*) A void pointer can hold address of any type and can be typcasted to any type.
  (*iii*) **malloc( ) and calloc( ) return void * type** and this allows these functions to be used to allocate memory of any data type.
  (*iv*) void pointers cannot be dereferenced.
- **Typedef:** typedef is used to give a new name to an already existing or even a custom data type (like a structure).

Scope rules in C

- **Global Scope:** Can be accessed anywhere in a program.
- **Local Scope:** Can be accessed inside only declared function.
- **Block Scope:** A Block is a set of statements enclosed within left and right braces **{**and**}**. Blocks may be nested in C (a block may contain other blocks inside it). A variable declared in a block is accessible in the block and all inner blocks of that block, but not accessible outside the block.

Storage Classes

- **Auto:** Default storage class for all the variables declared inside a function. They are assigned a garbage value by default when they are declared.

Scanned by CamScanner

- **Static:** Static variables have a property of preserving their value even after they are out of their scope. Static variables are allocated memory in data segment, not stack segment. In C, static variables can only be initialized using constant literals.
- **Register:** The only difference between auto and register storage is that the compiler tries to store these variables in the register of the processor if a free register is available. **This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program.** If a free register is not available, these are then stored in the memory only.
- Using '&' with a register variable may result in compiler giving an error or warning, because when we say a variable is a register means that, it may be stored in a register instead of memory and accessing address of a register is invalid.
- Register keyword can be used with pointer variables.
- Register can not be used with static.
- There can be unlimited number of register variables in a program (but the point is compiler may put some variables in register and not the others).

Memory Layout

- **Text segment** (Also known as a **code segment**, one of the sections of a program in memory [read only area], which contains executable instructions).
- **Initialized data segment** (Also known as **Data Segment** [read write area], a portion of virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer).
- **Uninitialized data segment** (contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code).
- **Stack** (Stack, where automatic variables are stored, along with information that is saved each time a function is called).
- **Heap** (Heap is the segment where dynamic memory allocation usually takes place and Heap area is shared by all shared libraries and dynamically loaded modules in a process).

Operators in C

- **Arithmetic Operators:** (*i*) Addition ('+'), (*ii*) Subtraction ('–'), (*iii*) Multiplication ('*'), (*iv*) Division ('/') and (*v*) Modulus ('%').

Scanned by CamScanner

- **Relational Operators:** (*i*) Equal checking '= =', (*ii*) Not equal to '!=', (*iii*) Grater than '>', (*iv*) Less than '<', (*v*) Grater than equal to '>=' and (*vi*) Less than equal to '<='.
- **Logical Operators:** (*i*) Logical AND ('&&'), (*ii*) Logical OR ('||') and (*iii*) Logical NOT ('!').

Short-Circuiting in Logical Operators

In case of logical AND, the second operand is not evaluated if first operand is false while in case of logical OR, the second operand is not evaluated if first operand is true.

S. No.	Operator	Associativity
1.	( ), [ ], →, .	Left to right
2.	!, ~, ++, – –, + (unary), – (unary), *, &, sizeof	Right to left
3.	*, /, %	Left to right
4.	+, –	Left to right
5.	<<, >>	Left to right
6.	<, <=, >, >=	Left to right
7.	==, !=	Left to right
8.	&	Left to right
9.	^	Left to right
10.	|	Left to right
11.	&&	Left to right
12.	||	Left to right
13.	? :	Right to left
14.	=, +=, –=, *=, /=, %=, &=, ^=, |=, <<=, >>=	Right to left
15.	,	Left to right

(Highest) ↑ ↓ (Lowest)

- **Comma has the least precedence among all operators and should be used. There is no chaining of comparison operators in C.**
- **Sizeof( ) function:**
  - (*i*) To find out number of elements in a array.
  - (*ii*) To allocate block of memory dynamically.
- **Modulus (%) on Negative Numbers:** Sign of left operand is appended to result in case of modulus operator in C.

Scanned by CamScanner

- 'char s[ ] = "madeeasy"' creates a character array. We can modify array element but not base address of array, while 'char *s = "madeeasy"'creates a string literal which stored in read only part of memory so we cannot change string value but can change their base address.

Switch Case

1. The expression used in switch must be integral type (int, char and enum). Any other type of expression is not allowed.
2. All the statements following a matching case execute until a break statement is reached.
3. The default block can be placed anywhere. The position of default doesn't matter, it is still executed if no match found.
4. The integral expressions used in labels must be a constant expressions.
5. The statements written above cases are never executed After the switch statement, the control transfers to the matching case, the statements written before case are not executed.
6. Two case labels cannot have same value.

Dangling pointer

A pointer pointing to a memory location that has been deleted (or freed) is called dangling pointer. There are three different ways where Pointer acts as dangling pointer:

(*i*) De-allocation of memory

(*ii*) Function Call

(*iii*) Void pointer

NULL Pointer

NULL Pointer is a pointer which is pointing to nothing. In case, if we don't have address to be assigned to a pointer, then we can simply use NULL.

- **Important Points:**

(*i*) **NULL vs Uninitialized pointer:** An uninitialized pointer stores an undefined value. A null pointer stores a defined value, but one that is defined by the environment to not be a valid address for any member or object.

(*ii*) **NULL vs Void Pointer:** Null pointer is a value, while void pointer is a type

Scanned by CamScanner

Wild Pointer

A pointer which has not been initialized to anything (not even NULL) is known as wild pointer. The pointer may be initialized to a non-NULL garbage value that may not be a valid address.

(*i*) Uninitialized pointers are known as wild pointers because they point to some arbitrary memory location and may cause a program to crash or behave badly.

(*ii*) If a pointer p points to a known variable then it's not a wild pointer.

(*iii*) If we want pointer to a value (or set of values) without having a variable for the value, we should explicitly allocate memory and put the value in allocated memory.

Enumeration (or enum)

- Enumeration (or enum) is a user defined data type in C.
- Used to assign names to integral constants.
- Two enum names can have same value.
- Not explicitly assign of values to enum names results in, the compiler by default assigns values starting from 0.
- We can assign values to some name in any order. All unassigned names get value as value of previous name plus one.
- The value assigned to enum names must be some integeral constant, i.e., the value must be in range from minimum possible integer value to maximum possible integer value.
- All enum constants must be unique in their scope. For example, the following program fails in compilation.

Memory Leak

Memory leak occurs when programmers create a memory in heap and forget to delete it. Memory leaks are particularly serious issues for programs like daemons and servers which by definition never terminate.

To avoid memory leaks, memory allocated on heap should always be freed when no longer needed.

Strlen ()

The function takes a single argument, i.e, the string variable whose length is to be found, and returns the length of the string passed.

Scanned by CamScanner

Parameter Passing

	Actuals used	Actuals changed	Formals changed	Similar technique
Call by value	Yes	No	Yes	–
Call by reference	Yes	Yes	Yes	–
Call by result	No	Yes	Yes	Call by reference
Call by restore (value-result)	Yes	Yes	Yes	Call by reference
Call by need	Yes	No	Yes	Call by value
Call by text	Yes	Yes	–	Call by reference
Call by name	Yes	Yes	–	Call by reference
Call by constant	Yes	No	No	–

Note:

- "Call by reference" and "call by name" parameter passing gives different result while passing address of an array element as a parameter.
- "Call by reference" and "call by value result" parameter passing gives different result in presence of loops.

Scope

Scope of a program variable is the range of statements in which the variable is visible. A variable is visible in a statement if it can be referenced in that statement.

Static Scoping

The method of binding names to non local variables called static scoping.

- Scope of variable can be statically determined prior to execution.
- Easier to read, more reliable, and executes faster.

Dynamic Scoping

It is based on the calling sequence of subprograms.

- Scope can be determined at runtime.
- More flexibility, but expense of readability, reliability and efficiency.

■■■

Scanned by CamScanner

Functions and Recursion

2

Recursion

Recursion is a function which calls itself either directly or indirectly.

- **Factorial function:**

```
int fact (int n)
{
        if (n == 0) return 1;
        else return n*fact (n – 1);
}
```

- **Fibonacci sequence:**

```
int fib (int n)
{
        if (n < 2) return n;
        else return fib (n – 1) + fib (n – 2);
}
```

(*i*) The number of function calls in fib(n) = 2 × fib(n+1)–1

(*ii*) The number of addition in fib(n) = fib(n+1)–1

(*iii*) In fib(n) after (n+1) function calls first addition will be performed.

- **Tower of Hanoi:** (A : source, B : temporary, C : destination)

```
TOH (A, B, C, n)
{
        if (n – 1) C.PUSH (POP (A));
        else
        {
                TOH (A, B, C, n – 1);
                C.PUSH (POP (A));
                TOH (B, C, A, n – 1);
        }
}
```

(*i*) Number of function calls in TOH(n) = $2^{n+1} - 1$

(*ii*) Number of moves in TOH(n) = $2^n - 1$

Scanned by CamScanner

- GCD of two numbers (iteration)

```
f(a, b)
{
        while (b ≠ 0)
        {
                t = b;
                b = a mod b;
                a = t;
        }
        return a;
}
```

- GCD of two numbers (recursive function)

```
f(a, b)
{
        if (a = b) return a;
        else if (a > b) return f(a – b, b);
        else f(a, b – a); // if (a < b)
}
```

- Finding the second maximum of three (distinct) numbers:

```
int trial (int a, int b, int c)
{
        if ((a > = b) && (c < b)) return b;
        else if (a > = b) return trial (a, c, b);
        else return trial (b, a, c);
}
```

Declarations and Notations

- **int*p;** — p can be a pointer to an integer.
- **int*p[10];** — p is a 10 element array of pointers to integer.
- **int (*p)[10];** — p is a pointer to a 10-element integer array.
- **int *p();** — p is a function that returns a pointer to an integer.
- **int p(char*a);** — p is a function that takes an argument as pointer to a character and returns an integer.
- **int*p(char *a);** — p is a function that takes an argument as pointer to a character and returns a pointer to an integer.
- **int (*p)(char *a);** — p is a pointer to a function that takes an argument as pointer to a character and returns an integer.

Scanned by CamScanner

- int (*p(char *a))[10]; p is a function that takes an argument as pointer to a character and returns a pointer to a 10 element integer array.
- int p(char (*a)[]); p is a function that takes an argument as pointer to a character array and returns an integer.
- int p(char *a[]); p is a function that takes an argument as array of pointers to characters and returns an integer.
- int *p(char a[]); p is a function that takes an argument as character array and returns a pointer to an integer.
- int *p(char (*a) []); p is a function that takes an argument as pointer to a character array returns a pointer to an integer.
- int *p(char *a[]); p is a function that takes an argument as array of pointers to characters and returns a pointer to an integer .
- int (*p)(char (*a)[]); p is a pointer to a function that takes an argument as pointer to a character array and returns an integer.
- int *(*p)(char (*a)[]); p is a pointer to a function that takes an argument as pointer to a character array and returns a pointer to an integer.
- int *(*p)(char *a[]); p is a pointer to a function that takes an argument as array of pointers to characters and returns a pointer to an integer.
- **int (*p[10])();** p is a 10 element array of pointers to functions, each function returns an integer.
- **int (*p[10])(char a);** p is a 10 element array of pointers to functions each functions takes an argument as character, and returns an integer.
- **int *(*p[10])(char a);** p is a 10 element array of pointers to functions, each function takes an argument as character, and returns a pointer to an integer.
- **int *(*p[10])(char *a);** p is a 10 element array of pointers to functions, each function takes an argument as pointer to a character, and returns a pointer to an integer.

■■■

Scanned by CamScanner

Abstract Data Types, Arrays and Linked Lists

3

Introduction

Data Structures based on the manipulation criteria:

1. **Primitive Data structures:** Integers, real numbers, characters, boolean values.
2. **Non-primitive data structures:** Arrays, stacks, queues, records, files, linked lists, tress, graphs, etc.

Data structures based on the storage criteria:

1. **Linear Data structures:** Arrays, records, stacks, queues, etc.
2. **Non-linear data structures:** Trees, Graphs, etc.

Linked Lists

- Linked list is a collection of nodes (contains data and pointer).
- **Properties:** Follow dynamic allocation and non-contiguous in nature.

Operations on Single Linked Set

Insertion

- Number of pointers modified to insert a new node at end = 2

```
node → next = newnode;
newnode → next = null;
```

- Number of pointers modified to insert a new node at beginning = 2

```
newnode → next = root;
root = newnode;
```

- Number of pointers modified to insert a new node in mid = 2

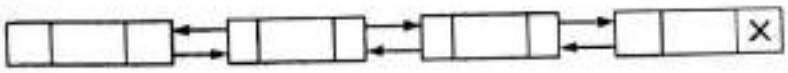
```
newnode → next = node → next;
node → next = newnode;
```

Deletion

- Number of pointers modified to insert a new node at beginning = 1

```
head → next = head → next → next;
```

Operations on Double Linked Set



```
struct node
```

Scanned by CamScanner

```
{
    int data;
    struct node * prev;
    struct node * next;
}
```

Insertion

- Number of pointers are modified to insert a node at beginning = 4

```
newnode → next = root;
newnode → prev = null;
root → prev = newnode;
root = newnode;
```

- Number of pointers are modified to insert a node at the end = 3

```
newnode → next = null;
newnode → prev = node;
node → next = newnode;
```

- Number of pointers are modified to insert a node in between = 4

```
newnode → next = node → next;
newnode → prev = node;
(node → next) → prev = newnode;
node → next = newnode;
```

Deletion

- Number of pointers are modified to delete a node in middle = 2

```
(node → prev) → next = node → next;
(node → next) → prev = node → prev; free (node)
```

Array

Array is a collection of homogeneous elements stored in contiguous memory location.

Properties: Static in nature, Compile time early binding and user friendly

1-D Array

Let '*A*' be an array of *n* elements. Address of an element *A*[*i*]:

Base(*A*) + (*i* – start index) × size of element.

2-D Array

Let *A*[*m*][*n*] be a 2-*D* array with *m* rows and *n* columns.

- Address of an element ***A*[*i*][*j*]** in Row Major order:

Scanned by CamScanner

Base (A) + (j – start index) × size of element +
(i – start index) × size of element × n

- Address of an element A[i][j] in Column Major order:

Base (A) + (i – start index) × size of elements +
(j – start index) × m × size of elements

Array Operations

Insertion

- [best case] At end takes constant time i.e. $\Omega(1)$
- [worst case] At beginning takes $O(n)$ time
- [Average case] In middle takes $\theta(n)$ time

Deletion

- [best case] At end takes constant time i.e. $\Omega(1)$
- [worst case] At beginning takes $O(n)$ time
- [Average case] In middle takes $\theta(n)$ time

Traversal

Direct access: $O(1)$

Triangular Matrices

- Lower Triangular $[\triangle]_{n\times n}$:

(*i*) Size $= n + \dfrac{n^2 - n}{2} = \dfrac{n(n+1)}{2}$

(*ii*) Row Major Order (RMO): $(j-1) + \dfrac{i(i-1)}{2}$

(*iii*) Column Major Order (CMO): $a[i][j] = (i-j) + \left[(j-1)n - \dfrac{(j-1)(j-2)}{2}\right]$

- Upper Triangular $[\triangledown]_{n\times n}$:

(*i*) Size $= \dfrac{n(n+1)}{2}$

(*ii*) RMO: $a[i][j] = (j-i) + \left[(i-1)n - \dfrac{(i-1)(i-2)}{2}\right]$

(*iii*) CMO: $a[i][j] = \left[(i-1) + \dfrac{j(j-1)}{2}\right]$

Scanned by CamScanner

- **Strictly lower triangular** $[\triangle]_{n \times n}$

  (*i*) Size = $\dfrac{n^2 - n}{2}$

  (*ii*) RMO: $a[i][j] = (j-1) + \dfrac{(i-1)(i-2)}{2}$

  (*iii*) CMO: $a[i][j] = (i-1) + \dfrac{(j-1)(j-2)}{2}$

- **Tridiagonal** $[\diagdown]_{n \times n}$:

  (*i*) Size = $3n - 2$

  (*ii*) Row: $2i + j - 3$

  (*iii*) Column: $i + 2j - 3$

■■■

Scanned by CamScanner

Stacks and Queues

4

Stack

A stack is an ordered list in which all insertion and deletions are made at one end. Top is a pointer pointing to the top most element of the stack.

Applications of Stack

1. Recursion
2. Post fix notation (Evaluation of expression)
3. Conversion of arithmetic expression from infix to polish (prefix/postfix)
4. Tower of hanoi
5. Fibonacci series
6. Permutation
7. Balancing of symbols
8. Subroutines

Stack Operations

Operations on stack are: push( ), pop( ), Is empty( ), Is full( ), peep( ) and change( ).

- **Push operation on stack:** Let N is the maximum stack size and x is the element to be inserted on to stack S.

```
push(S, Top, N, x)
{
    if (Top == N – 1)
    {
        printf("stack is overflow");
        exit(1);
    }
    Top ++;
    S [Top] = x;
}
```

- **Pop operation on stack:**

```
pop (S, Top, N)
{
    int y;
    if (Top==–1)
```

Scanned by CamScanner

```
        {
            printf ("under flow");
            exit(1);
        }
        y = S[Top];
        Top – –;
        return (y);
    }
```

Queue

It is first-in first-out (FIFO). The fist item inserted is the first to be removed. One side insertion (Rear), other side deletion (front).

- **Queue Operations:** enqueue( ), Dequeue( ), Is empty( ) and Is full( ).
- **Double ended queue:** Insertion and deletion can be on both sides.
- **Input restricted queue:** Restriction only on i/p not on deletion.
- **Output restricted queue:** Deletion on one side and no restriction of insertion.
- **Priority queue:** Ascending priority queue/Descending priority queue.
- A queue can be implemented using two stacks.

Enqueue Operation for QUEUE

- QUEUE [0: n-1] is an array, 'front' stores the subscript value of the first element of the queue, 'rear' stores the subscript value of the last element of the queue, 'front' and 'rear' are initialized to –1 when the queue is empty, and 'element' is the new element to be inserted into the queue.

```
Enqueue (QUEUE, n, front, rear, element)
{
    If (((front == 0) and (rear == n–1)) or (rear ==front – 1))
    {
        Print "Overflow"
        exit( );
    }
    if (front == –1)
        front = rear = 0;
    else if (rear == n – 1)
        rear = 0;
    else
```

Scanned by CamScanner

```
        rear = rear + 1;
    QUEUE [rear] = element;
    exit( );
}
```

queue Operation for QUEUE

```
Dequeue (QUEUE, n, front, rear)
{
    if (front == –1)
    {
        Print "Underflow"
        exit( );
    }
    element = QUEUE [front];
    if (front == rear)
        front = rear = –1;
    else if (front == n – 1)
        front = 0;
    else
        front = front + 1;
    exit( );
}
```

pplications of Queue

. Multiprogramming
. Resource sharing
. BFS
. CPU scheduling
. Radix sort
. Real time system
. Storing operands in evaluation of prefix expressions.

Deque (Double Ended Queue)

It is a linear data structure in which insertions and deletions can be made at both the ends. There are four possible operations: Insert at the left, Delete at the left, Insert at the right and Delete at the right.

■■■

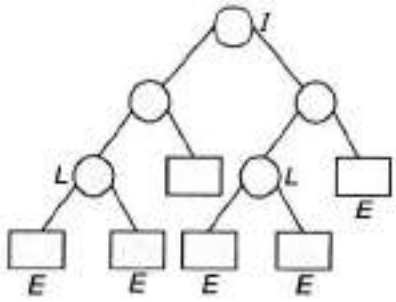
Scanned by CamScanner

Trees, BSTs and Binary Heaps

5

Tree

- Tree is defined as $T(V, E)$ where V = set of vertices and E = set of edges.
- In a tree there is exactly one edge between a pair of vertices and there are no cycles and self loops.
- The degree of a node in the tree is defined as the maximum number of children that node can hold.
- A tree can never be empty, but binary tree may be empty.
- Internal node (I): A node having minimum of 1 child is known as internal node.
- Leaf node (L): A node having no children.
- External node (E): The replacement of null leaves are external nodes.



- **Properties:**
  - (i) For 'n' nodes there are $(n-1)$ edges.
  - (ii) Let n_0 = number of nodes with degree 0.
    n_2 = number of nodes with degree 2.
    $n_0 = n_2 + 1$
  - (iii) (In heap array) If child is at location i(index starts from 0) then parent is at location $\lfloor (i-1)/2 \rfloor$.
    If parent is at location i(index starts at 0) then left child is at location $2i + 1$ and Right child is at location $2i+2$.
  - (iv) (In heap array) If parent is at location i(indexing at 1) then Left child is at location $2i$ and Right child is at location $2i+1$.
    If child is at location i(indexing at 1) then parent is at location $\lfloor i/2 \rfloor$.

Scanned by CamScanner

(v) In a linked representation of binary tree, if there are 'n' nodes then number of NULL links = n + 1.

- **Complete Binary Tree** is a tree in which nodes are inserted/filled from left to right. All the leaves may not be at the same level.
- **Full Binary Tree** is a tree in which every node has exactly 2 children and all the leaves are at the same level.
- All full binary trees are complete binary trees but vice versa is not true.
- **Strict Binary Tree** is a tree in which each node has exactly 0 or 2 children.
- A complete n-ary tree is one in which every node has '0' or 'n' sons. If 'x' is the number of internal nodes of a complete n-any tree, the number of leaves in it is: $x(n-1)+1$.
- Let $T(n)$ be the number of different binary search trees on n-distinct elements, then $T(n) = \sum_{K=1}^{n} T(k-1)\,T(n-k+1)$.
- In binary tree, Rank (or) Index of any node:

  Rank (i) = (Number of nodes in left subtree of i) + 1
- In a m-ary tree (degree = m), if n_i is number of nodes of degree 'i' ($i = 0, 1, \ldots, m$) then n_0(leaf nodes) $= 1 + \sum_{i=2}^{m} (i-1)n_i$
- Maximum size of array to store a binary tree with 'n' nodes $= 2^n - 1$.
- Minimum size of array to store a binary tree with 'n' nodes $= 2^{\lceil \log(n-1) \rceil} - 1$.
- Maximum height possible for a binary tree with 'n' nodes = n
- Minimum height possible for a binary tree with 'n' nodes = $\log_2 n$
- Maximum number of nodes in a binary tree of height 'h' $= 2^{h+1} - 1$.
- Total number of binary trees possible with n nodes $= {}^{2n}C_n / (n+1)$
- For non-empty binary tree, if n is number of nodes and e is the number of edges, then $n = e + 1$.

Binary Tree Traversals

Traversing a tree means a visiting each node in a specified order.

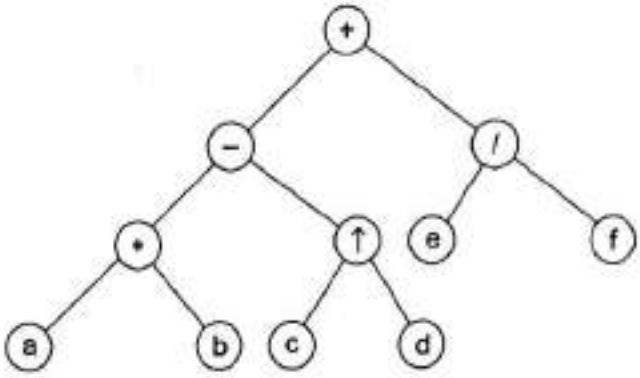
Traversal Techniques

1. Preorder (Root, Left, Right)

Scanned by CamScanner

2. Inorder (Left, Root, Right)
3. Postorder (Left, Right, Root)

Example:



Preorder Traversal: + − * ab ↑ cd / ef

Inorder Traversal: a * b− c ↑ d + e / f

Postorder Traversal: ab * cd ↑ − ef / +

Level Order Traversal: + − / * ↑ ef abcd

Binary Search Tree (BST)

- Left child < Root < Right Child
- Recursion of left child from the root gives the minimum element and Recursion of right child from the root gives the maximum element.
- Inorder traversal of BST is called "Binary Sort". Sorted order of data results in ascending order.
-

	Average Case	Worst Case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$
Build		$O(n^2)$

- **"Deletion of a node" in BST:**
  (*i*) If node has no child, simply delete that node.
  (*ii*) If node has 1 child, delete that node and replace it with child.
  (*iii*) If node has 2 children, delete that node and replace it with inorder successor or predecessor of that node.

Scanned by CamScanner

Heap Tree

- Heap tree is a complete binary tree with heap property (min heap/max heap).
- There are two types of heap trees:
  (*i*) **Min heap:** At any subtree the root value always smaller than its children.
  (*ii*) **Max heap:** At any subtree the root value always greater than its children.

- **Heap Sort:** By deleting the root node from heap tree and placing the deleted node in output, until all the nodes are deleted.
- It gives either ascending order (min heap) or descending order (max heap).

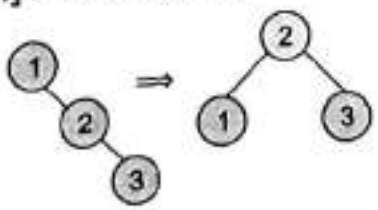
AVL Search Tree

- Devised by **Adelson Velski Landis**
- Height balanced binary search tree
- Balance factor (b_f) is :

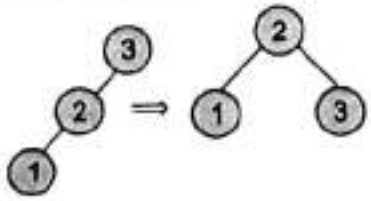
$$|b_f| = |h_L - h_R| \le 1$$

$$b_f = -1 \text{ or } 0 \text{ or } 1$$

- Rotation is a technique used in the AVL tree to balance the height.
- **Four types of rotations are:**
  (*i*) **Right-Right [RR] Rotation (Single Rotation):**

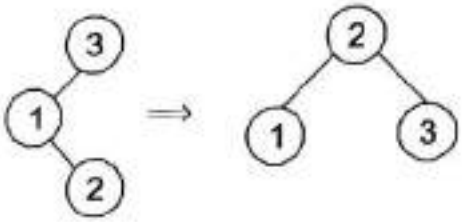


  (*ii*) **Left-Left [LL] Rotation (Single Rotation):**

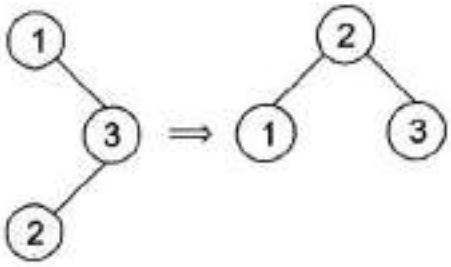


Scanned by CamScanner

(*iii*) Left-Right [LR] Rotation (Double Rotation):



(*iv*) Right-Left [RL] Rotation (Double Rotation):



- Minimum number of nodes in a AVL tree of height '*h*'

$$N_{min}(H) = \begin{cases} 1 & ; \quad L = 0 \\ 2 & ; \quad L = 1 \\ 1 + N(H-1) + N(H-2) & ; \quad L > 1 \end{cases}$$

Time and Space Complexity with each Data Structure

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log (n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Scanned by CamScanner

Time and Space Complexity with each Data Structure

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log (n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	[illegible]
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Scanned by CamScanner

A Handbook on Computer Science

5

Algorithms

CONTENTS

Analysis and Asymptotic Notations 174

Divide-and-Counquer 180

Greedy Approach 183

Dynamic Programming 186

Graph Traversals, Hashing and Sorting 190

Complexity Classes: P, NP, NPH and NPC 194

Scanned by CamScanner

Analysis and Asymptotic Notations

1

Algorithm

An algorithm is the step by step instructions to solve a given problem.

Asymptotic Notation

Big-O Notation	Comparison Notation	Limit Definition
$f \in o(g)$	$f \ominus g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$
$f \in O(g)$	$f \leqslant g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} < \infty$
$f \in \Theta(g)$	$f = g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} \in R > 0$
$f \in \Omega(g)$	$f \geqslant g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} > 0$
$f \in \omega(g)$	$f > g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \infty$

Complexity Types

Complexity	Notation	Description
Constant	$O(1)$	Constant number of operations, not depending on the input data size $n = 1000000 \rightarrow$ 1-2 operations
Logarithmic	$O(\log n)$	Number of operations proportional to $\log_2 n$ $n = 1000000000 \rightarrow 30$ operations
Linear	$O(n)$	Number of operations proportional to input data size $n = 10000 \rightarrow 5000$ operations
Quadratic	$O(n^2)$	Number of operations proportional to the square of the input data size $n = 500 \rightarrow 250000$ operations
Cubic	$O(n^3)$	Number of operations proportional to the cube of the input data size $n = 200 \rightarrow 8000000$ operations
Exponential	$O(2^n)$, $O(k^n)$, $O(n!)$	Exponential number of operations, fast growing $n = 20 \rightarrow 1048576$ operations

Scanned by CamScanner

Properties of Asymptotic Notation

- **Reflexive Property:** If $f(n) = O(f(n))$ and $f(n) = \Omega(f(n))$ then $f(n) = \Theta(f(n))$.
- **Transitive Property:**
  (i) If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ then $f(n)$ is $\Theta(h(n))$.
  (ii) If $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n)$ is $O(h(n))$.
  (iii) If $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n))$ then $f(n)$ is $\Omega(h(n))$.
- **Symmetric Property:** $f(n) = \Theta(g(n))$ **if and only if** $g(n) = \Theta(f(n))$
- **Transpose Symmetry:** $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$
- If $f(n) = O(g(n))$ and $e(n) = O(h(n))$ then
  (a) $f(n) + e(n) = \mathbf{O(g(n) + h(n))}$ **or** $\mathbf{O(\max(g(n), h(n)))}$
  (b) $f(n).e(n) = \mathbf{O(g(n).h(n))}$
  (c) If $f(n) = O(g(n))$ then $\mathbf{h(n).f(n) = O(h(n).g(n))}$

Important Formulae

1. $\log x^y = y \log x$
2. $\log^k n = (\log n)^k$
3. $\log_b x = \log_a x / \log_a b$
4. $\log_b x = \log x^a / \log b^a$
5. $\log(xy) = \log x + \log y$
6. $\log(x/y) = \log x - \log y$
7. $n! = O(n^n)$
8. $2^n = O(n^n)$
9. $\sum_{i=1}^{n} 2^i = O(2^n)$
10. $\sum_{k=1}^{n} \log k = O(n \log n)$
11. $\sum_{k=1}^{n} \frac{1}{k} \log k = O(n \log n)$
12. $\sum_{i=1}^{n} i^2 = O(n^3)$
13. $\sum_{k=1}^{n} k^p = \frac{1}{p+1} n^{p+1}$
14. $\sum_{i=1}^{n} i = 1 + 2 + 3 \ldots + n = O(n^2)$
15. $\sum_{k=0}^{n} x^k = 1 + x + x^2 + x^3 + \ldots x^n = \frac{x^{n+1}}{x-1}; (x \neq 1)$

Asymptotic Analysis

Loops

The running time of a loop, is atmost the running time of the statements inside the loop (including tests) multiplied by the number of iterations.

Scanned by CamScanner

Example:

```
main( )                          ↦        1
{
    x = y + z;                   ↦        1
    for (i =1; i <= n; i++)      ↦        2n+1
    {
        x = y + z;               ↦        n
    }
}
Total time                       ↦        = cn = O(n)
```

Nested Loop

Analyse from inner loop to outer loop. Total running time is the product of the sizes of all the loops

Example:

```
for (i = 1, i <= n, i++)         ↦        2n+1
{
    for (j = 1, j <= n, j++)     ↦        O(n²)
    k = k + 1;                   ↦        O(n²)
}
Total time                       ↦        = cn² = O(n²)
```

Consecutive for Statements

Add the time complexities of for statements.

```
x = x + 1;                       ↦        O(1)
for (i = 1; i < n; i++)          ↦        O(n)
    m = m + 2;                   ↦        O(n)
for (i = 1; i ≤ n; i++)          ↦        O(n)
{
    for (j = 1; j < n; k++)      ↦        O(n²)
    k = k + 1;                   ↦        O(n²)
}
Total time                       ↦        = O(n²)
```

If-then-else Statement

Worst case running time is either **then** part or **else** part (whichever is the larger).

Scanned by CamScanner

Example:

```
if (length() = = 0)                          ↦   O(1)
{
    return false;                            ↦   O(1)
}
else
{
    for (n = 0; n < length(); n ++)          ↦   O(n)
    {
        if (! list [n] = (otherlist [n]))    ↦   O(n)
        return false;                        ↦   O(n)
    }
}
Total Time                                   ↦   = O(n)
```

Logarithmic Complexity

An algorithm is O(log$n$) if it takes constant time to cut the problem size by a fraction (usually by 1/2)

Example:

```
for (i = 1; i < n; )                         ↦   O(log n)
{
    i = i * 2;                               ↦   O(log n)
}
Total Time                                   ↦   = O(log n)
```

Recurrence Relations

Substitution Method

Substituting the given function again and again until given function is removed.

1. $T(n) = \begin{cases} T(n-1) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$ ↦ $O(n^2)$

2. $T(n) = \begin{cases} T(n-1) * n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$ ↦ $O(n^n)$

3. $T(n) = \begin{cases} T(n/2) + c & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$ ↦ $O(\log_2 n)$

Scanned by CamScanner

4. $T(n) = \begin{cases} T(n/2)+n & \text{if } n>1 \\ 1 & \text{if } n=1 \end{cases}$ $\rightarrow$ $O(n)$

5. $T(n) = \begin{cases} T(n-1)+\log n & \text{if } n>1 \\ 1 & \text{if } n=1 \end{cases}$ $\rightarrow$ $O(n\log n)$

Recursive Tree Method

If $T(n)$ can be divided into parts then recursive tree method is used.

1. $T(n) = \begin{cases} C & \text{if } n=1 \\ 2T(n/2)+\Theta(n) & \text{otherwise} \end{cases}$ $\rightarrow$ $O(n\log_2 n)$
2. $T(n) = T(n/3) + T(2n/3) + cn$ $\rightarrow$ $O(n\log n)$
3. $T(n) = 3T(n/4) + cn^2$ $\rightarrow$ $O(n^2)$

Master Method

Let $f(n)$ be a function and $T(n)$ be defined on the non-negative integers by recurrence relation: $T(n) = aT(n/b) + f(n)$ where $a \geq 1$ and $b > 1$. Then $T(n)$ can be bounded asymptotically as follows:

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta\left(n^{\log_b a} \log^k n\right)$ k = constant $\forall k \geq 0$ then

   $T(n) = \Theta\left(n^{\log_b a} \cdot \log^{k+1} n\right)$

3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$ and if $a \cdot f(n/b) \leq c \cdot f(n)$ for some constant $c < 1$ and for all sufficiently large n, then $T(n) = \Theta(f(n))$.

Extended Master Theorem:

Recurrence: $T(n) = aT(n/b) + \Theta(n^k \log^p n)$

where $a \geq 1$, $b > 1$, $k \geq 0$ and p is a real number then,

(1) If $a > b^k$ then $T(n) = O(n^{\log_b a})$

(2) If $a = b^k$

if $p > -1$, then $T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

if $p = -1$, then $T(n) = \Theta(n^{\log_b a} \log\log n)$

if $p < -1$, then $T(n) = \Theta(n^{\log_b a})$

(3) If $a < b^k$

if $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$

if $p < 0$, then $T(n) = O(n^k)$

Scanned by CamScanner

Muster Theorem for substract and conquer recurrences:

Let $T(n)$ be a function defined on possible n:

$$T(n) = \begin{cases} aT(n-b)+f(n), & \text{if } n > 1 \\ c, & n \le 1 \end{cases}$$

for some constant c, $a > 0$, $b > 0$, $d \ge 0$ and function $f(n) = O(n^d)$.

(1) $T(n) = O(n^d)$, if $a < 1$.

(2) $T(n) = O(n^{d+1})$, if $a = 1$.

(3) $T(n) = O(n^d a^{n/b})$, if $a > 1$.

■■■

Scanned by CamScanner

Insertion is better than merge for small input

Divide-and-Counquer

2

Divide and Conquer (DAC):

A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

1. Divide and conquer algorithms are naturally adapted for execution in multi-processor machines, especially shared-memory systems where the communication of data between processors does not need to be planned in advance, because distinct sub-problems can be executed on different processors.
2. Divide-and-conquer algorithms naturally tend to make efficient use of memory caches.
3. Divide-and-conquer algorithms are naturally implemented as recursive procedures.

```
DAC (P)
{
    if (P is small) return solution (P);
    else
    {
        k = Divide (P);
        return (combine (DAC (P1), DAC (P2),...DAC (Pk)));
    }
}
```

Divide and Conquer Algorithms

Maximum Minimum

Find Maximum and minimum element of an array using minimum number of comparisons.

- **Recurrence Relation:** $T(n) = \begin{cases} 0; & \text{if } n = 1 \\ 1; & \text{if } n = 2 \\ T(n/2) + T(n/2) + 1; & \text{if } n > 2 \end{cases}$

  Time complexity with DAC = $O(n)$

  (i) Using linear search = $O(n)$

Scanned by CamScanner

(ii) **Using Tournament method:** Time complexity = $O(n)$
Number of Comparison: When n is power of 2 then
$T(n) = 2T(n/2) + 2 = (3n/2) - 2$ else more than $(3n/2) - 2$.

(iii) **Compare in Pairs:** Time complexity = $O(n)$
Number of comparison
(a) When n is even = $(3n/2) - 2$, (b) Else n is odd = $(3n-1)/2$.

erge Sort

Comparison based sorting.
Stable sorting algorithm but **outplace**.

Recurrence Relation: $T(n) = \begin{cases} c \text{ if } n = 1 \\ T(n/2) + T(n/2) + cn \text{ if } n > 1 \end{cases}$

Time complexity: **(i) Worst case** = $O(n\log n)$, **(ii) Best case** = $O(n\log n)$,
(iii) Average case = $O(n\log n)$.
Space complexity: $S(n) = 2n + \log n + c$; **Worst case** = $O(n)$

uick Sort

Comparison based sorting.
2 to 3 times faster than merge and heap sort.
Not **stable** sorting algorithm but **inplace**.
Choosing pivot is most important factor.
As Start element $T(n) = O(n^2)$, As End element $T(n) = O(n^2)$
As Middle element $T(n) = O(n^2)$, As Median element $T(n) = O(n^2)$
As Median of Median $T(n) = O(n \log n)$
Time complexity:

(i) **Best case:** each partition splits array into two halves then
$T(n) = T(n/2) + T(n/2) + n = O(n \log n)$

(ii) **Worst case:** each partition gives unbalanced splits we get
$T(n) = T(n-k) + T(k-1) + cn = O(n^2)$

(iii) **Average case:** In the average case, we donot know where the split happens for this reason we take all the possible value of split locations.

$$T(n) = \frac{1}{n}\sum_{i=1}^{n} T(i-1) + T(n-i) + n + 1 = O(n \log n)$$

Space complexity: $S(n) = 2n = n + \log n = O(n)$.
Randomized quick sort choose pivot from random places make worst case $O(n \log n)$ but for array with same element worst case $O(n^2)$.

Scanned by CamScanner

Matrix Multiplication

- Using DAC: $T(n) = \begin{cases} O(1), \text{ for } n = 1 \\ 8T(n/2) + O(n^2), \text{ for } n > 1 \end{cases}$

  Time complexity = $O(n^3)$
- Strassen's Matrix Multiplication:

  $$T(n) = \begin{cases} O(1) & , \text{ for } n = 1 \\ 7T(n/2) + an^2 & , \text{ for } n > 1 \end{cases}$$

  Time complexity = $O(n^{2.81})$ by Strassen's.
  Time complexity = $O(n^{2.37})$ by Coppersmith and Winograd.
- Karatsuba algorithm for fast multiplication of two n-digit numbers required exactly $n^{\log_2 3}$ (when n is a power of 2) using Divide and Conquer.

Power of an Element

- $T(n) = \begin{cases} 1 & , \text{ if } n = 0 \\ a & , \text{ if } n = 1 \\ T(n/2) + c, & \text{ otherwise} \end{cases}$

  Time complexity = $O(\log_2 n)$

Binary Search based on decrease and conquer approach

- Recurrence Relation: $T(n) = \begin{cases} 1 \text{ if } n = 1 \\ c + T(n/2) \text{ if } n > 2 \end{cases}$
- Best case = $O(1)$
- Average case = worst case = $O(\log_2 n)$

Closest Pair of Points

The problem is to find the closest pair of points in a set of points in x-y plane.

- **Without Divide and Conquer:** The problem can be solved in $O(n^2)$ time by calculating distances of every pair of points and comparing the distances to find the minimum.
- **With Divide and Conquer:** The Divide and Conquer algorithm solves the problem in $O(n\log_2 n)$ time.

■■■

Scanned by CamScanner

Greedy Approach

3

Greedy Algorithms

A greedy algorithm always makes the choice that seems to be the best at that moment. i.e. locally-optimal choice in the hope that it will lead to a globally-optimal solution. A Greedy algorithm makes greedy choices at each step to ensure that the objective function is optimized.

A problem must comprise two components for a greedy algorithm to work:

- It has **optimal substructures**. The optimal solution for the problem contains optimal solutions to the sub-problems.
- It has a **greedy property**, if you make a choice that seems the best at the moment and solve the remaining sub-problems later, you still reach an optimal solution. You will never have to reconsider your earlier choices.

Job Sequencing with Deadline

Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes single unit of time, so the minimum possible deadline for any job is 1. Our task is to maximize total profit if only one job can be scheduled at a time. **Steps to solve problem:**

(*i*) Sort the given list of *n* jobs in ascending order of profits

(*ii*) Find maximum deadline in the given array of *n*-deadlines and take the array of maximum deadlines size.

(*iii*) For every slot *i* apply linear search to find a job which repeat this step for every slot.

Time complexity = $O(n^2)$

Activity Selection Problem

You are given *n* activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time. **Steps to solve problem:**

(*i*) Sort the activities according to their finishing time.

(*ii*) Select the first activity from the sorted array and print it.

Scanned by CamScanner

(*iii*) For sorted array check if the start time of this activity is greater than or equal to the finish time of previously selected activity then select this activity and print it.

Time Complexity:

(a) It takes $O(n \log n)$ time if input activities may not be sorted.

(b) It takes $O(n)$ time when it is given that input activities are always sorted.

Huffman Coding

Assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code. **Steps to solve problem:**

1. Build a Huffman Tree (using Min Heap is used as a priority queue) from input characters.
   (*i*) Initially, the least frequent character is at root.
   (*ii*) Extract two nodes with the minimum frequency from the min heap.
   (*iii*) Create a new internal node with frequency equal to the sum of the two nodes frequencies.
   (*iv*) Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
2. Traverse the Huffman Tree and assign codes to characters.

- Has property that no code is prefix of any other code
  Time complexity: $O(n \log n)$
- **The number of bits per message:**
  Σ(frequency of external node 'i') × (Number of bits required for q_i)
- **Weigthed external path length** = $\Sigma q_i d_i$ (d_i = distance from root to external nodes 'i').

Fractional Knapsack Problem

Steps to solve problem:

(*i*) for ($i = 1$; $i \le n$; i++)
$a[i] = P_i/w_i$ **(Profit to weight ratio for each item)**

(*ii*) Arrange **array a** in ascending order

(*iii*) Take one by one object from a and keep in Knapsack untill Knapsack becomes full.

Time complexity = $O(n \log n) + O(n) = O(n \log n)$

Scanned by CamScanner

Kruskal's Minimum Spanning Tree Algorithm

Steps of Kruskal's Algorithm:

(*i*) Sort all the edges from low weight to high

(*ii*) Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.

(*iii*) Keep adding edges until we reach all vertices.

Time Complexity: O(E log E) or O(E log V).

- It starts with edge (min cost edge) and intermediate may results either tree or forest.

Prim's Minimum Spanning Tree

The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected.

Steps of Prim's Algorithm:

(*i*) Initialize the minimum spanning tree with a vertex chosen at random.

(*ii*) Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree.

(*iii*) Keep repeating step 2 until we get a minimum spanning tree.

- Always a connected graph.

Single Source Shortest Path (Dijkstra's Algorithm)

- **Time complexity:** O((V + E) log V) using binary min heap.
- Drawback of Dijkstra's Algorithm: It will not give shortest path for some vertices if graph contain negative weight cycle.
- Time complexity of Dijkstra's and Prim's algorithm using various data structure:

  (*i*) **Using Binary Heap:** O((V + E) log V)

  (*ii*) **Fibonacci heap:** O((V logV + E))

  (*iii*) **Binomial Heap:** O((V + E) log V)

  (*iv*) **Array:** O(V^2 + E)

Bellman Ford Algorithm

- It finds shortest path from source to every vertex, if the graph doesn't contain negative weight cycle.
- If graph contain negative weight cycle, it don't compute shortest path from source to all other vertices but it will report saying "negative weight cycle exists".

  Time complexity = O(VE) when dense graph E = V^2 and for sparse graph E = V.

■■■

Scanned by CamScanner

Dynamic Programming

4

Dynamic Programming Algorithms

Longest Common Subsequence

Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

- Recurrence Relation:

$$LCS(i, j) = \begin{cases} 0; & \text{if } i = 0 \text{ (or) } j = 0 \\ 1 + LCS(i-1, j-1); & \text{if } x[i-1] = y[j-1] \\ \max\,[LCS(i-1, j), LCS(i, j-1)]; & \text{if } x[i] \neq y[j] \end{cases}$$

Time complexity: by **Brute force** = $O(2^m)$, by **Dynamic programming** = $O(m \times n)$.

Space complexity = $O(mn)$; where m and n are length of two given sequences.

0/1 Knapsack Problem

Given weights and values of n items, select items to put items in a Knapsack of capacity W to maximise the total value in the Knapsack. You cannot break an item, either pick the complete item, or don't pick it (0 to 1 property).

- $$0/1\ KS(M, N) = \begin{cases} 0; & \text{if } M = 0 \text{ or } N = 0 \\ 0/1\ KS(M, N-1); & \text{if } w[n] > M \\ \max \begin{Bmatrix} 0/1\ KS(M - W[n], N-1) + P[n] \\ 0/1\ KS(M, N-1) \end{Bmatrix}; & \text{otherwise} \end{cases}$$

Time complexity = $O(MN)$: If M value is very large then it behaves like an exponential and **NP-complete problem**.

Travelling Salesperson Problem

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. In other words find the minimum cost Hamiltonian cycle.

Scanned by CamScanner

$$TSP(A, R) = \begin{cases} C(A,S) \text{ if } R = \phi \\ \min\{(C[A,K] + TSP(K, R-K)) \forall K \in R\} \end{cases}$$

Time:
Time complexity: without dynamic programming = $O(n^n)$, with dynamic programming = $O(2^n \cdot n^2)$
Space complexity: without dynamic programming = $O(n^2)$, with dynamic programming = $O(n^n)$.
It is one of the NP Hard problem.

Matrix Chain Multiplication

Given a sequence of matrices, find the most efficient order to multiply these matrices together in order to minimise the number of multiplications.

$$MCM = \begin{cases} 0 & \text{if } i = j \\ \min\left\{ \begin{array}{c} mcm(i,K) + mcm(K+1,j) + Pi \times Pj \times Pk \\ \text{where } i \le K < j \text{ or} \\ i \le K \le j-1 \end{array} \right\} \end{cases}$$

Number of ways to multiply matrix: $T(n) = \sum_{i=1}^{n-1} T(i).T(n-i)$

Time complexity: without dynamic programing = $O(n^n)$, with dynamic programing = $O(n^3)$.
Space complexity: without dynamic programing = $O(n)$, with dynamic programing = $O(n^2)$.

Sum of Subset Problem

Find if there is a subset of the given set, sum of whose elements is equal to given sum.

- **Recurrence Relation:**

$$SoS(M, N, S) = \begin{cases} \text{return}(S); & M = 0 \\ return(-1); & N = 0 \\ SoS(M, N-1, S); & \text{if } w[N] > M \\ \text{Min}\left\{ \begin{array}{l} SoS(M - w[N], N-1, S \cup w[N]) \\ SoS(M, N-1, S) \end{array} \right\}; & \text{otherwise} \end{cases}$$

Time complexity by Brute force = $O(MN)$

Scanned by CamScanner

Floyd-Warshall's: All Pair Shortest Path

For finding shortest paths between all pairs of vertices in a weighted graph with positive or negative edge weights (but with no negative cycles).

- $A^k(i, j)$ = the min cost required to go from i to j by considering all intermediate vertices are numbered not greater than k.

$$A^0(i, j) = C(i, j)$$

$$A^k(i, j) = \min\left\{A^{k-1}(i, j), A^{k-1}(i, k) + A^{k-1}(k, j)\right\}$$

Time complexity = $O(n^3)$ with heap = $O(n^2 \log n)$.

- Warshall's algorithm will not work for negative edge cycle.

Optimal Binary Search Tree

Given a sorted array keys[0.. $n-1$] of search keys and an array frequency[0.. $n-1$] of frequency counts, where frequency[i] is the number of searches to keys[i]. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.

- **It is lexically ordered tree:**

$$\text{cost}(i, j) = \min_{i<k\le j} \{\text{cost}(i, k-1) + \text{cost}(k, j) + w(i, j)\}$$

$$w(i, j) = p(j) + q(i) + w(i, j-1)$$
$$w(i, i) = q(i) \text{ and cost}(i, i) = 0$$

$$\text{cost} = \underbrace{\sum_{i}^{n} p_i \times \text{level } a_i}_{\text{(successful)}} + \underbrace{\sum_{i}^{n+1} q_i\,(\text{level } E_i - 1)}_{\text{(unsuccessful)}}$$

Time complexity is $O(n^3)$.

Multistage Graph

A Multistage graph is a directed graph in which the nodes can be divided into a set of stages such that all edges are from a stage to next stage only.

Task is to find shortest path from source to destination using the special property of multistage graph.

Shortcut: Travels back from destination to source with every time min cost selection.

Scanned by CamScanner

$$MSG \underset{\substack{\downarrow \\ \text{stage}}}{(S_i,} \underset{\substack{\downarrow \\ \text{vertex}}}{V_j)} = \begin{cases} 0 \text{ if } s_i = F \,\&\&\, V_j = 0 \\ \min \begin{cases} ((V_a, K) + MSG(s_i + s, K)) \\ \forall K \in s_i + 1 \,\&\&\, (V_j, K) \in E \end{cases} \end{cases}$$

Time complexity: without dynamic programing = $O(2^n)$, **with** dynamic programing = $O(V + E)$.

Space complexity: without dynamic programing = $O(v^2)$, **with** dynamic programing = $O(v^2)$.

■■■

Scanned by CamScanner

Graph Traversals, Hashing and Sorting

5

Graph traversals

Depth First Search

1. **Preorder traversal** or postorder or inorder of ordered tree [$O(n + e)$; Average case]
   [$O(n^2)$: Worst case adjacency matrix].
2. Stack and backtracking technique.
3. Computes the number of paths between two vertices.
4. Computing a cycle and to find articulation points.
5. Biconnected components and strong components.
6. Euler paths and number of connected components.
7. Whether the graph is connected or not connected.

Breadth First Search

1. Level by level traversal of ordered tree [$O(n + e)$]
2. Queue and greedy technique.
3. Used for Topological sort and dijkstra's Algorithm
4. Prim's Algorithm.
5. Whether the graph is connected or not connected.
6. Number of connected components.
7. Transitive closure of a graph.
8. Computing a cycle.

Hashing

Hashing is a searching technique in which the searching is done in constant O(1). It is based on indexing mechanism. It uses a function called HASH FUNCTION.

- **Components in hashing:** Hash table, Hash functions, Collisions and Collision Resolution Techniques.
- Load factor = $\dfrac{\text{Number of elements in hash table}}{\text{Hash table size}}$

Scanned by CamScanner

- **Direct address table:** In direct address table key is the address without any manipulation. Even though number of keys are very less but one of key may contain 64 bits then size of hash table should be $2^{64} - 1$. The range of keys determines the size of the direct address table.
- **Types of Hash Functions:**
  (i) Division modulo method:
  $H(X) = X\% \, m$, where m = Hash table size and X = Key.
  (ii) Digit extraction method (Truncation method)
  (iii) Mid Square method
  (iv) Folding Method: Fold shifting method and Fold boundary
- **Collision Resolution Techniques:**
  (i) Open addressing
  (ii) Chaining.

Collision Resolution Techniques

Collision will take place if a new element is getting mapped where an element is already present.

- **Open Addressing:**
  (i) Linear Probing: If there is a collision at location 'L' then look for empty location successively.
  Disadvantage:
  Primary clustering: The trend is for long sequence of preoccupied positions still become longer, primarily at one place.
  (ii) Quadratic Probing: Hash function = $h+i^2$ [Quadratic in nature]
  Disadvantage: Secondary clustering.
  (a) The maximum number of comparisons performed for successful search = size of largest cluster + 1
  (b) In Quadratic probing, all buckets are not compared because of quadratic nature.
  (iii) Random Probing: RNG = Random Number Generator.
  No number should be repeated within random space.

$$Y_{new} = Y_{old} + C \bmod m$$

where Y = Seed value
C = Constant
m = Hash table size

Disadvantage: Random clustering.

Scanned by CamScanner

(*iv*) *Double Hashing/Rehashing*: '*C*' is changed in random probing in order to get new sequence.

Advantage: Clustering is reduced.

Disadvantage: Clustering is not avoided.

Note:

- Load Factor (α) = n/m

  n = number of elements present in hash table

  m = Hash table size
- If α is less, collisions are less
- $1 - \alpha$ = % of free space
- In open addressing scheme $0 \le \alpha \le 1$

- **Chaining:**

  (*i*) Chaining uses linked implementation for hashing.

  (*ii*) Number of elements mapped to a location = length of the linked list.

  (*iii*) Complexity of deleting an element in the chaining = deletion of a node in single linked list.

  (*iv*) The maximum number of comparisons performed for successful search in chaining = size of largest linked list.

  (*v*) There is no overflow problem.

  (*vi*) Load factor $\alpha = n/m$, where $\alpha \ge 0$.
- Average number of probes (comparisons) in successful search

Successful Seach	Unsuccessful Seach
Linear probing: $\frac{1}{2} + \frac{1}{2(1-\alpha)}$	$\frac{1}{2} + \frac{1}{2(1-\alpha)^2}$
Chaining: $(\alpha + 1)/2$	$1 + \alpha/2$

Scanned by CamScanner

Searching and Sorting

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quick sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
Merge sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
Time sort	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
Heap sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(1)$
Bubble sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Insertion sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Selection sort	$\Omega(n^2)$	$\Omega(n^2)$	$O(n^2)$	$O(1)$
Tree sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(n)$
Shell sort	$\Omega(n \log(n))$	$\Theta(n \log(n))^2)$	$O(n \log(n))^2)$	$O(1)$
Bucket sort	$\Omega(n + k)$	$\Theta(n + k)$	$O(n^2)$	$O(n)$
Radix sort	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$	$O(n + k)$
Counting sort	$\Omega(n + k)$	$\Theta(n + k)$	$O(n + k)$	$O(k)$
Cube sort	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$

■■■

Scanned by CamScanner

Complexity Classes: P, NP, NPH and NPC

6

Complexity Theory

Complexity theory is mainly used to recognize whether the problem that is decidable is easy or hard. This can be calculated by means of time complexity and space complexity.

Complexity Classes

P-space Problem

The language in P-space represents the polynomial time and space complexity of the language.

NP-space Problem:

The language in NP space represents the non polynomial space complexity of the language.

P-problem

P is the class of problems that can be solved by deterministic algorithms in a time that is polynomially related to the size of the respective problem instance.

It consists of all languages accepted by some deterministic turing machine that runs in polynomial amount of time, as a function of input length.

Examples: 1-SAT, 2-SAT, Unary partition, Shortest path, 2-colourability, Enter cycle, Equivalence of DFA, Min cut, Shortest cycle in a graph, Sorting.

NP-problem

NP is the class of problems that can be solved by nondeterministic algorithms in a time that is polynomially related to the size of the respective problem instance.

It consists of all languages that are accepted by non deterministic turing machines with a polynomial bound on the time taken along any sequence of non deterministic choices.

Examples: Traveling sales person problem and Sub graph isomorphism.

NP-Hard Problem

If there is a language X such that every language Y in NP can be polynomially reducible to X and we cannot prove that X is in NP then X is said to be NP-Hard problem.

Scanned by CamScanner

A problem X is NPH iff $\forall Y \in NP, Y \leq_p X$.

Examples: Turing machine halting problem.

NP-Complete Problem

If there is a language X such that every language Y in NP can be polynomially reducible to X and X is in NP then X is said to be NP-Hard problem. If NP hard problem is present in NP then it is called NP-complete problem.

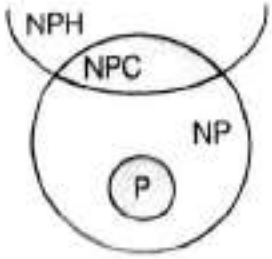
A problem X is NPC iff $\forall Y \in NP, Y \leq_p X, X \in NP$

Examples: 3-SAT problem, Traveling sales man problem, Hamilton circuit problem, Vertex cover problem, Independent set problem, Chromatic number problem, Coloring problem, Subgraph isomorphism problem, Edge cove problem, Knapsack problem, Max cut.

Reduction

$P_1 \leq P_2$: Problem P_1 is reducible to a problem P_2. It means problem P_2 is atleast as hard as problem P_1.

- If $P_1 \leq P_2$ and P_1 is undecidable then P_2 is also undecidable.
- If $P_1 \leq P_2$ and P_2 is decidable then P_1 is also decidable.
- If $P_1 \leq P_2$ and P_2 is recursive then P_1 is also recursive.
- If $P_1 \leq P_2$ and P_2 is recursive enumerable then P_1 is also recursive enumerable.
- If $P_1 \leq P_2$ and P_2 is P-problem then P_1 is also P-problem.
- If $P_1 \leq P_2$ and P_2 is NP-problem then P_1 is also NP-problem.
- If NP-complete is in P then P = NP.



■■■

Scanned by CamScanner

A Handbook on
Computer Science

6

Theory of Computation

CONTENTS

1. Regular Languages and Finite Automata 197
2. Push Down Automata and CFLs 211
3. Recursively Enumerable Sets & Turing Machines 216
4. Undecidability 220

Scanned by CamScanner

Regular Languages and Finite Automata

1

Introduction

- **Alphabet (Σ)** : An Alphabet is a set of finite number of symbols.
  Example : $\Sigma = \{a, b\}$ is alphabet with two symbols a and b.
- **String** : A string is any sequence of symbols over the given alphabet Σ.
  Example : abb is a string over $\Sigma = \{a, b\}$
- Empty string (ε or λ): Empty string is a string with no symbol over any alphabet. Empty string is also called as "Null String".
  $|\varepsilon| = 0$ (i.e, empty string length is 0)
- **Length of a string** : The number of symbols in the sequence of given string is called "length of a string".
  Example : $|abb| = 3$, for the string abb over $\Sigma = \{a, b\}$
- **Substring of a string** : Any sequence of zero or more consecutive symbols of the given string over an alphabet is called a "Substring of a string".
  Example : For string abb over $\Sigma = \{a, b\}$, the possible substrings are: ε, a, b, ab, bb and abb.
- **Prefix of a string** : Any substring with the sequence of beginning symbols of a given string is called a "Prefix".
  Example : For string "abb", the possible prefixes of abb are: ε, a, ab, abb.
- **Suffix of a string**: Any substring with the sequence of trailing (ending) symbols of a given string is called a "Suffix".
  Example : For string abb, the possible suffixed are: ε, b, bb, abb.
- **Power of an Alphabet:**
  Consider $\Sigma = \{a, b\}$. The following are powers of an alphabet over Σ.
  $\Sigma^0 = \{\varepsilon\}$: zero length string
  $\Sigma^1 = \{a, b\}$: set of 1-length strings
  $\Sigma^2 = \{aa, ab, ba, bb\}$: set of 2-length strings
  $\Sigma^* =$ the set of all strings over $\Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$
  $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots = \Sigma^* - \Sigma^0 = \Sigma^* - \{\epsilon\}$
- **Language** : A set of strings over the given alphabet Σ is called a "language" (or collection of strings).
  Example : Let $\Sigma = \{a, b\}$. Then Language $L = \{ab, aab\}$

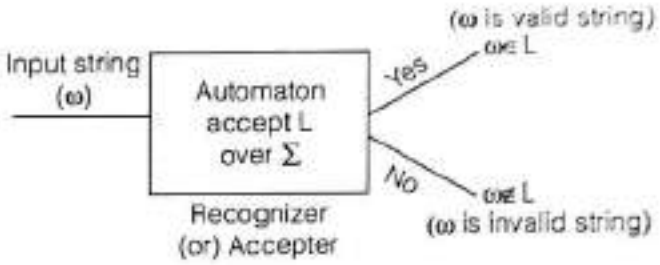
Scanned by CamScanner

- **Grammar** : Grammar Contain set of rules to generate the strings of a language. Grammar is a set of 4 tuples. Grammar $G = \{V, T, P, S\}$ where V is set of non-terminals, T is set of terminals, P is set of rules and S is start symbol ($S \in V$).

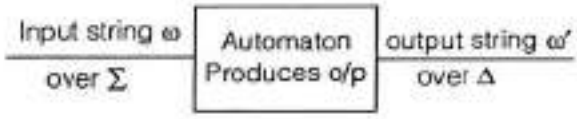
  Each rule of the grammar appears as : $X \rightarrow Y$, where X and Y are any sequence of terminals and non-terminals.

- **Automaton:**

  (*i*) **Automaton Acts as Recognizer or Accepter:** An automaton is a machine that accept or recognizes the strings of a language (L) over an input alphabet Σ.



  (*ii*) **Automaton Acts as producer or enumerator or transducer:** An automaton can also produce the output over any alphabet D for the given input alphabet Σ.



- **Formal Languages:** Formal language is a set of all strings where each string is restricted over a particular forms of a string with the given input.

- Equivalence of language, grammar and automata

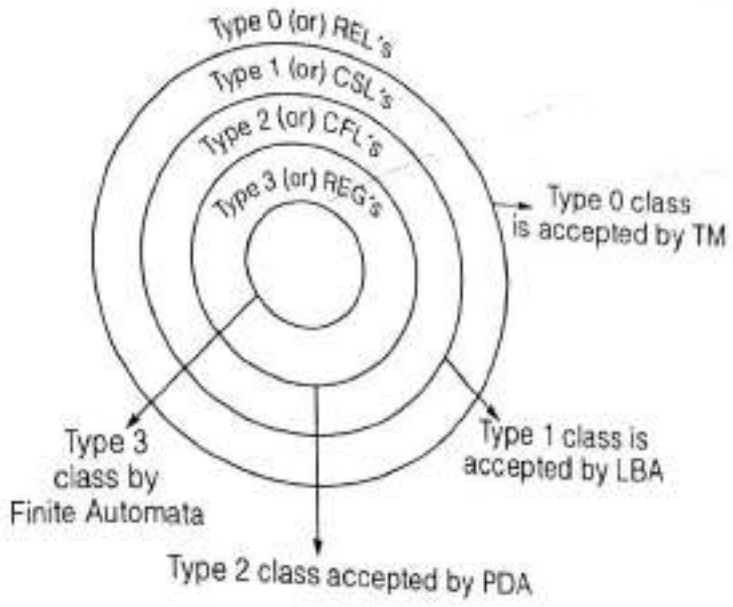
	Formal Languages	Grammars	Automata
1.	Regular Languages	Regular grammars	Finite Automata
2.	Context Free Languages	Context Free Grammars	Push down Automata
3.	Context Sensitive Languages	Context Sensitive Grammars	Linear Bound Automata
4.	Recursive Enumerable Languages	Recursive enumerable grammar (unrestricted)	Turing Machines

Every formal language can be generated by equivalent grammar and be accepted by the equivalent automaton as shown in above table.

- **Chomsky Hierarchy :**

All formal languages are divided into 4 classes by chomsky and those class hierarchy known as "Chomsky-Hierarchy".

Scanned by CamScanner



Type 3 ⊂ Type 2 ⊂ Type 1 ⊂ Type 0.

Types of Automaton :

(*i*) **Finite Automaton (FA):** An automaton that accepts a regular language is called a "FA".

Types of Finite Automaton:

(a) Deterministic FA(DFA)

(b) Non-deterministic FA(NFA or NDFA)

L(DFA) ≅ L(NFA)

i.e., both types of finite automata have same expressive power.

(*ii*) **Push Down Automaton(PDA):** An automaton that accepts a context free language is called a "PDA".

Types of PDA:

(a) Deterministic PDA(DPDA).

(b) Non-deterministic PDA(NPDA or PDA).

L(DPDA) < L(NPDA) i.e., DCFL's ⊂ CFL's

Class of languages accepted by DPDA's are proper subset of class of languages accepted by NPDA's.

The class of NPDA's have more expressive power than the class of DPDA's

(*iii*) **Linear Bound Automaton (LBA):** An automaton that accepts a context sensitive language is called a "LBA".

(*iv*) **Turing Machine (TM):** An automaton that accepted a recursive enumerable language is called a "TM". TM can accept a REL and TM can enumerate a REL.

Scanned by CamScanner

Types of TM:

(a) Deterministic TM (DTM)

(b) Non-deterministic TM (NTM)

$L(NTM) \equiv L(DTM)$

- **Types of Grammars:**

(*i*) **Regular Grammar(RG) or type-3:** A grammar is regular grammar iff it is either left linear grammar or right linear grammar. Regular grammar is a grammar where all rules are restricted as following.

Either $V \rightarrow VT^* \mid T^*$ [Left linear grammar].

or

$V \rightarrow T^*V \mid T^*$ [Right linear grammar]

where V is any variable and T any terminal.

(*ii*) **Context Free Grammar (CFG) or Type-2:** Every rule of CFG is restricted as $V \rightarrow (V \cup T)^*$

(*iii*) **Context Sensitive Grammar (CSG) or Type-1:**

Rule : $X \rightarrow Y$

Where $|X| \leq |Y|$, $X \in (V \cup T)^*$ and $Y \in (V \cup T)^*$ and X must contain atleast one variable.

If NULL productions are present in the grammar, it may or may not be CSG.

(*iv*) **Recursive Enumerable Grammar (REG) or Type-0:**

Rule : $X \rightarrow Y$

Where $X \in (V \cup T)^*$, $Y \in (V \cup T)^*$ and X must contain atleast one variable.

- **Equivalences of Language, Automata & Grammar :**

(*i*) **Regular Language (Type-3 language or finite state language):**

$\cong$

Finite Automaton [DFA $\cong$ NFA $\cong$ ε-NFA]

$\cong$

Regular Grammar (Type-3 grammar)

$\cong$

Left linear Grammar

$\cong$

Right linear Grammar

$\cong$

Regular Expression

Scanned by CamScanner

(*ii*) **Context Free language (Type-2 language):**

≅

Push Down Automata [NPDA or PDA]

≅

Context Free Grammar

(*iii*) CSG ≅ LBA ≅ CSL (Type-1 language)

(*iv*) REG ≅ TM ≅ REL (Type-0 language)

- **Expressive Power of Automata:** Expressive power of a machine is the class or set of languages accepted by the particular type of machines. FA < DPDA < PDA < LBA < TM. FA is less powerful than any other machine and TM is more powerful than any other machine.
  - (*i*) Type 3 class ⊂ Type 2 class ⊂ Type 1 class ⊂ Type 0 class
  - (*ii*) **FA** ≅ TM with read only tape ≅ TM with unidirectional tape ≅ TM with finite tape ≅ PDA with finite stack
  - (*iii*) PDA ≅ FA with stack
  - (*iv*) TM ≅ PDA with additional stack ≅ FA with two stacks
- **Applications of Automata:**
  - **(*i*) Finite automata can be used in the following cases:**
    - (a) To design a lexical analysis of a compiler
    - (b) To recognize the pattern using regular expressions
    - (c) To design the combination and sequential circuits using Moore/ mealy machines.
  - **(*ii*) PDA can be used in the following cases:**
    - (a) To design the parsing phase of a compiler (syntax analysis)
    - (b) To implement stack applications
    - (c) To evaluate arithmetic expressions
    - (d) To solve the Tower of Hanoi problem
  - **(*iii*) Linear Bounded Automata can be used in the following cases:**
    - (a) To construct syntactic parse trees for semantic analysis of the compiler.
    - (b) To implement genetic programming.
  - **(*iv*) Turing Machine can be used in the following cases:**
    - (a) To solve any recursively enumerable problem
    - (b) To understand complexity theory
    - (c) To implement neural nets
    - (d) To implement artificial Intelligence
    - (e) To implement Robotic applications

Scanned by CamScanner

Remember:

- NFA has same power as of DFA
- DPDA has less power than NPDA
- DTM has same power as of NTM
- Mealy and Moore machines have same power
- **Mealy machine:** Output depends on current state and current input
- **Moore machine:** Output depends on current state only.
- Finite automaton does not have infinite memory required for comparison.
- Every finite language is regular but converse need not be true.
- Language may be infinite but variables, terminals and production rules are finite for every grammar.
- Regular grammar is either left linear or right linear or both.
- Every NFA can be converted to DFA and then to minimal DFA
- If NFA has n-states then the equivalent DFA has atmost 2^n states.
- Let w be a string of length n. Then number of possible substring including ε is atmost $\frac{n(n+1)}{2}+1$.
- Number of prefixes for the given n-length string = $(n + 1)$
- Number of suffixes for the given n-length string = $(n + 1)$

Finite Automaton (FA)

Finite Automata

Finite Acceptor (FA without output)	**Finite Transducer (FA with output)**
Deterministic: DFA	Moore Machine
Non-deterministic: NFA and ε-NFA	Mealy Machine

- **Finite automata is categorized into two types:**
  - (*i*) **Finite automata with output:** This is of two types Mealy machines and Moore machines.
    - (a) In Mealy machine the output is associated with transition (state and i/p symbol)
    - (b) In the Moore machine output is associated with the state.

Scanned by CamScanner

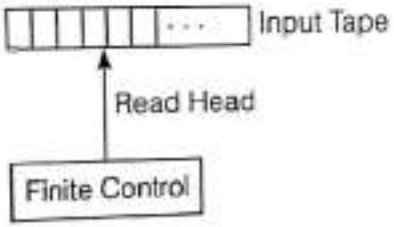
(ii) **Finite automata without output:** This is of two types DFA and NFA

(a) **DFA:** It is deterministic finite automata. For every input symbol in DFA there is an exactly one transition from each state of finite automata. It accepts all strings that are present in the language by halting in a final stage and rejects all strings that are not present in the language by halting in a non-final state.

(b) **NFA:** It is non deterministic finite automata. For every input symbol in NFA there is atleast one transition or no transitions from each state of finite automata. It accepts all strings that are present in the language by halting in a final state and rejects all strings not present in the language either by halting in a final state or by halting in a dead configuration.

(c) **ε-NFA:** It is a non deterministic finite automata which include ε transitions between states.

- **Model of Finite Automata:**



(i) Input tape contains a string over the given input alphabet.

(ii) Read head is a pointer which reads one input symbol at a time and moves to the next symbol.

(iii) Finite control contains set of states and a transition function to take an action based on current state and input symbol scanned.

Deterministic Finite Automata (DFA or FA)

For every input symbol of alphabet there is an exactly one transition from every state of finite automata is called as DFA.

DFA covers transitions for all valid and invalid strings in the given regular language.

For every valid string, DFA reaches to one of its Final state and for every invalid string it reaches to non-final state.

Scanned by CamScanner

Specifications of DFA

$$DFA = (Q, \Sigma, \delta, q_0, F) \text{ is 5-tuple notation.}$$

Where Q is set of finite states,

Σ is input alphabet contains finite number of input symbols

δ is transition function defined over transitions of FA for state and input

$\delta : Q \times \Sigma \rightarrow Q$

q_0 is initial state or start state of DFA, $q_0 \in Q$

F is set of final states of DFA

Compound FA (F1 × F2)

Let F1 and F2 are two DFA's. Then operations defined over FA are (F1 ∪ F2), (F1 ∩ F2), (F1 – F2), and (F2 – F1). Construction of compound FA for the given F1 and F2 as follows:

1. The number of states in compound FA (F1 × F2) is equal to m × n, where m is the number of states of F1 and n is the number of states of F2.
2. Initial state of compound FA is combination of initial state of F1 and initial state of F2.
3. Let Q1 = $\{q_0, q_1, q_2\}$ is set of states of F1, and Q2 = $\{q_a, q_b\}$ is set of states of F2. Let q_0 be the start state of F1 and q_a be the start of F2. Then Q1 × Q2 = $\{(q_0, q_a), (q_0, q_b), (q_1, q_a), (q_1, q_b), (q_2, q_a), (q_2, q_b)\}$, where (q_0, q_a) is an initial state of compound FA.
4. Let q_2 is final of F1 and q_b is final of F_2. Final states depends on type of compound finite automata (F1 × F2)
   (a) F1 ∪ F2 final states: Make those states of (Q1 × Q2) as final, if final state of F1 ***or*** final state of F2 contains in any of the states. Such a FA accepts L(F1) ∪ L(F2).
   (b) F1 ∩ F2 final states: Make those states of (Q1 × Q2) as final, if only final state of F1 ***and*** final state of F2 contains in any of the states. Such a FA accepts L(F1) ∩ L(F2).
   (c) F1 – F2 final states: Make those states of (Q1 × Q2) as final, if only final state of F1 ***but not*** final state of F2 contains in any of the states. Such a FA accepts L(F1) – L(F2).
   (d) F2 – F1 final states: Make those states of (Q1× Q2) as final, if only final state of F2 ***but not*** final state of F1 contains in any of the states. Such a FA accepts L(F2) – L(F1).

Scanned by CamScanner

Non-deterministic Finite Automata (NFA or NDFA)

For every valid string there exists a path from initial state that reaches to final state. NFA covers transitions for all valid strings only in the given regular language.

Specification of NFA without Null Moves

$$NFA = (Q, \Sigma, \delta, q_0, F) \text{ is a 5-tuple notation.}$$

Where Q is set of finite states,

Σ is input alphabet contains finite number of input symbols

δ is transition function defined over transitions of NFA for state and input.

$\delta: Q \times \Sigma \rightarrow 2^Q$

Epsilon NFA (ε-NFA) or NFA

For every valid string there exists a path from initial state that reaches to final state. ε-NFA is same as NFA but it can include epsilon transitions.

Specification of ε-NFA

$$\varepsilon\text{-NFA} = (Q, \Sigma, \delta, q_0, F) \text{ is a 5-tuple notation.}$$

Where $\delta: Q \times \Sigma \cup \{\varepsilon\} \rightarrow 2^Q$.

Regular Expressions

It is a declarative way of representation of a regular language. Regular expression is a compact formula that generates exactly those strings which are in a language.

- **Operators of Regular Expressions:** The operators in regular expressions are: * is kleene closure, · is concatenation operator and + is union operator.
- **Important Notations of Regular Expressions:** Some of the important notations compared to regular sets are:

Language		Regular expression
{ }	→	ϕ
{ε}	→	ε
{a}	→	a
{a, b}	→	a+b
{a, b, c}	→	a+b+c
{a, b, c,... z}	→	a+b+c... + z

Scanned by CamScanner

- **Properties of Regular Expression Operators:**
  (a) Union operator satisfies commutative property and associative property.
  (b) The concatenation operator satisfies associative property but not commutative property
  (c) Both left and right distributive property of concatenation over union are satisfied. $r(s + t) = rs + rt$ and $(s + t)r = sr + tr$
  (d) Both left and right distributive properties of union over concatenation are not satisfied. $st + r \neq (s + r)(t + r)$ and $r + st \neq (r + s)(r + t)$
  (e) Union operator satisfies idempotent property but the concatenation operator does not satisfy. i.e. $r + r = r$ but $r \cdot r \neq r$.
  (f) **Identity property:**

  $R + \phi = \phi + R = R$: ϕ acts as identity element with respect to union operation

  $\varepsilon.R = R.\varepsilon = R$: Epsilon ($\varepsilon$) acts as identity element with respect to concatenation.

- **Equivalences of Languages (or regular expressions):**
  (a) $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
  (b) $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$
  (c) $L(r^*) = (L(r))^*$
  (d) $r_1 \cdot r_2 \neq r_2 \cdot r_1$
  (e) $r_1(r_2 r_3) = (r_1 r_2) r_3$
  (f) $r_1(r_2 + r_3) = r_1 r_2 + r_1 r_3$
  (g) $r_1 + r_2 r_3 \neq (r_1 + r_2)(r_1 + r_3)$
  (h) $\phi + r = r$
  (i) $\phi.r = \phi = r.\phi$
  (j) $\lambda r = r\lambda = r$
  (k) $\lambda^* = \lambda$
  (l) $\phi^* = \lambda$
  (m) $r + r = r$
  (n) $r.r \neq r$
  (o) $r^*.r^* = r^*$
  (p) $r^*.r^+ = r^+$
  (q) $(r^*)^* = r^*$
  (r) $(r)^* = r^*$

Scanned by CamScanner

(s) $\lambda + rr^* = r^* = \lambda + r^+$

(t) $p(qp)^* = (pq)^*p$

(u) $(p + q)^* = (p^*q^*)^* = (p^*+q^*)^*$

(v) $(p + q)^*p^*q^* = (p + q)^*$

Closure Properties of Regular Languages

- The set of regular languages is closed under the union operation.
- Finite union is closed for regular languages but infinite union is not closed.
- The set of regular languages is closed under the intersection operation.
- Regular languages are closed under finite intersection but not under infinite intersection.
- The set of regular languages is closed under the difference operation.
- The set of regular languages is closed under the complement operation.
- The set of regular languages is closed under the concatenation operation.
- The set of regular languages is closed under the kleene closure operation.
- The set of regular languages is closed under the positive closure operation.
- The set of regular languages is closed under the prefix operation as well as suffix operation.
  Init(L) = Prefix(L) = {u | uv is in regular language L}
  Suffix(L) = {v | uv is in regular language L}
- The set of regular languages is closed under the reversal operation.
- Half of a Regular Language is also regular: Let L is a regular language. Then Half(L) = {u | uv ∈ L and |u| = |v|}.
- One third of a Regular Language is also regular: Let L is a regular language. Then onethird(L) = {u | uvw ∈ L and |u| = |v| = |w|}.
- Quotient of two Regular Languages is also regular: L1/L2 = {x | xy ∈ L1 for some y ∈ L2}.
- Subsequence of a Regular Language is also regular language: Subsequence(L) = {w | w is obtained by removing symbols from anywhere of a string in L}.
- Sub-word of a Regular Language is also regular: Sub-word(L) = {v | for some u and for some w, uvw is in L}.
- Homomorphism of two Regular Languages is also regular.
- Inverse Hommorphism of two Regular languages is also regular.
- Substitution of two Regular Languages is also regular.

Scanned by CamScanner

- Shuffle of two Regular Languages: Shuffle(L_1, L_2) $= \{x_1y_1x_2y_2...x_ky_k \mid x_1x_2...x_k$ is in L_1 and $y_1y_2...y_k$ is in $L_2\}$ is also regular.
- Symmetric difference of two Regular Languages is also regular.
- If L is regular then $\sqrt{L}$ is also a Regular Language: $\sqrt{L} = \{w \mid ww \text{ in } L\}$.
- If L regular then Max(L) is also a Regular Language.
- If L is regular than Min(L) is also regular. Min(L) = $\{w \mid w \in L$ and there is no proper prefix of w is in L$\}$.
- Regular language are not closed under subset, superset, infinite union as well as infinite intersection. i.e. a subset or superset of a regular language need not be regular. Infinite union and infinite intersections of regular languages need not be regular.

Decision Properties of Regular Language

- **Membership:** It is decidable.
- **Emptiness (Is L empty?):** Checking whether the given regular language is empty or not. It is decidable.
- **Finiteness (Is L finite?):** Checking whether the given language is finite or not. It is decidable.
- **Equivalence of two Regular Languages (Is L1 ≅ L2?):** Checking whether the given two regular languages are equal or not. It is decidable.
- **Subset Operation over Two Regular Languages (Is L1 ⊆ L2?):** Checking whether the regular language L1 is subset of a regular language L2 or not. $(L1 \subseteq L2) \cong (L1 \cap \overline{L2} = \phi)$. It is decidable.

Mealy and Moore Machines

Mealy and Moore machines are acts as language generators but not language recognizers. There are no final states in Moore and Mealy machine.

- **Mealy Machine:** In Finite automata, if output symbol is associated with transition (state and input) then such automata is called as Mealy machine.
- **Moore Machine:** In Finite automata if output symbol is associated with each state then such automata is called as Moore machine.

Specification of Mealy and Moore Machines:

$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a 6-tuple notation.

Scanned by CamScanner

Where,

Q is set of finite states,

Σ is input alphabet contains finite number of input symbols.

Δ is output alphabet which contains output symbols.

δ is transition function. $\delta : Q \times \Sigma \rightarrow Q$

λ is output function: $\lambda : Q \rightarrow \Delta^*$ (Moore machine)

$\lambda : Q \times \Sigma \rightarrow \Delta^*$ (Mealy machine)

q_0 is initial state or start of Moore machine, $q_0 \in Q$

Pumping Lemma

Pumping lemma can be used to prove that a given language is non-regular (using proof by contradiction). Then there exist a constant 'n' such that for every string 'w' in L of length 'n' or more the following holds. Let w has three parts x, y and z.

1. $w = xyz$, where $w \in L$ and $|w| \geq n$.
2. For all $i \geq 0$, $xy^i z \in L$
3. $|y| > 0$.
4. Optional condition $|xy| \leq n$.

Examples: (Non regular languages)

$L = \{a^m b^m | i \geq 0\}$

$L = \{a^p | p \text{ is a prime}\}$

$L = \{a^m b^n | m > n\}$

$L = \{a^m b^n | m < n\}$

$L = \{a^m b^n | m! = n\}$

$L = \{w | w \varepsilon (a + b)^*\}$ and w has #a's equal to #b's}

$L = \{ww | w \varepsilon (a + b)^*\}$

$L = \left\{a^{n^2}\right\}$

$L = \left\{a^{2^n}\right\}$

$L = \{a^n b^n c^n\}$

Scanned by CamScanner

Minimization of FA

Finite automata need to be minimized to reduce number of states of DFA.

- **Minimization methods:** Partition method (state equivalence), table filing method and Myhill-Nerode theorem.
- **Myhill – Nerode Theorem:** Let L be a regular language, then it contains finite number of equivalence classes. Union of all Myhill-Nerode equivalence classes gives universal language. All the strings of universal language partitioned into a finite number of Myhill-Nerode equivalence classes, if L is regular.

  The number of Myhill-Nerode equivalence classes in a regular language ≡ Number of states in the unique minimal DFA corresponding to that language.

■■■

U.V

Reversal → change initial → final & direction of all arrow

L^R

→ Complement → change initial → final & final → initial

$L \to L^c$

→

Scanned by CamScanner

Push Down Automata and CFLs

2

Context Free Grammar (CFG)

- Derivation of a String:
  - (*i*) **Left Most Derivation (LMD):** In each sentential form, the left most non-terminal is substituted with its productions to derive a string.
  - (*ii*) **Right Most Derivation (RMD):** In each sentential form, the right most non-terminal is substituted with its productions to derive a string.
  - (*iii*) **Parse tree:** Every intermediate node of a parse tree is a non-terminal which is represented by a Circle and all leaf nodes are terminal symbols.
- **Umambiguous grammar:** The grammar is called as unambiguous grammar if every string generated from the grammar has exactly one parse tree.

  #parse tree's = # LMD's = # RMD's = 1
- **Ambiguous grammar:** The grammar is called as ambiguous grammar if there exists a string which is derived from the grammar with more than one parse tree.

  (#parse tree's = # LMD's = # RMD's) > 1
- **Inherently Ambiguous Context Free language:** For a given language if there is no equivalent unambiguous CFG then such language is called as inherently ambiguous context free language.

  $L = \{a^k b^j c^k\} \cup \{a^m b^m c^n\}$ is an example of an inherently ambiguous CFL.
- **Reduced CFG (non-redundant CFG):** Reduced CFG is a CFG without any useless symbols.
- **Simplified CFG:** Simplified CFG is a CFG without any null-productions, unit-productions and useless symbols.

  The following order applied over a given CFG to convert into simplified CFG.
  - (*i*) Null-productions elimination.
  - (*ii*) Unit-productions elimination.
  - (*iii*) Useless symbols elimination.

Normal forms for CFG

Types of CFG Normal forms:

1. **Chomsky Normal Form:** Every CFG production of the CNF grammar contains either two non terminals or a single terminal in the RHS of the production.

Scanned by CamScanner

$A \to BC$

$A \to a$

CNF is also called binary standard form (the parse tree in CNF is always a binary tree).

(*i*) The number of steps required in derivation of an x-length string from the given CNF – context free grammar is: $(2x - 1)$

(*ii*) Let G be the given CFG without '$\in$' and unit productions and let 'K' be the maximum number of symbols on the right hand side of any production, then the equivalent CNF contain a **maximum of:** $(K - 1)|P| + |T|$ **productions**. Where, $|P|$ = the number of productions in 'G', $|T|$ = the number of terminals, K = maximum number of symbols on right hand side.

(*iii*) For the generation of 'l' length yield, the minimum height of the derivation of tree for the given CNF context free grammar is $\lceil \log_2 l \rceil + 1$ and maximum height will be l.

2. **Greibach normal form:** Every CFG production of GNF grammar contains a single terminal (T) followed by any sequence of non-terminals (V^*).

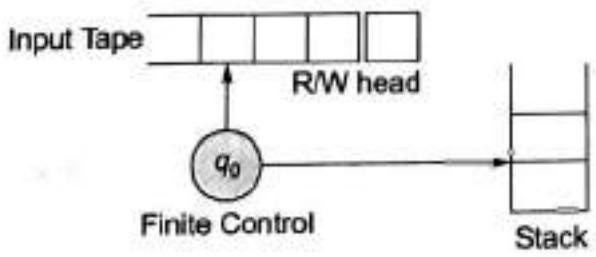
$V \to TV^*$

(*i*) In GNF context free grammar, the restrictions only on the positions, but not on length of right side of a production. $A \to a\alpha$, where $\alpha \in v$.

(*ii*) The number of steps required in derivation of an x-length string from the given GNF-context free grammar is: x

Push Down Automata

Configuration



- It contains finite control, input tape and a stack
- Finite control contains finite number of states.
- Read head or read pointer is used to read the input symbol from the input tape

Scanned by CamScanner

- The symbols can be pushed or popped from the stack, which is used to keep track of previous symbols.
- For every string finite control starts from the initial state. If the string is valid finite control reaches to the final state at end of input.

Acceptance Mechanisms

- Acceptance by final state (universal mechanism).
- Acceptance by empty stack (local mechanism).
- Acceptance by empty stack and final state.

CFL ≅ PDA by empty stack ≅ PDA by final state ≅ PDA by empty stack and final state ≅ CFG.

Operations

The operations on push down automata are as follows:

- **PUSH:** Some finite sequence of symbols is pushed into the stack.
- **POP:** One symbol at the top of the stack is popped.
- **BIPASS:** i.e. do nothing i.e. stack contents unchanged while state may be changed
- **REPLACE:** Replace one symbol at the top of the stack.

Specifications

PDA = $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a 7 tuple notation.

Where,

Q is set of finite states.

Σ is the input alphabet which contains finite number of input symbols.

Γ is the stack alphabet which contains a finite number of stack symbols.

δ is a transition function.

(*i*) (DPDA) $\delta : Q \times (\Sigma \cup \{\in\}) \times \Gamma \rightarrow Q \times \Gamma^*$

(*ii*) (NPDA) $\delta : Q \times (\Sigma \cup \{\in\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$ (only finite sub-sets)

q_0 is a single start state $q_0 \in Q$.

Z_0 is the start stack symbol $Z_0 \in \Gamma$.

F is the set of final states $F \subseteq Q$. If acceptance is by empty stack method then F is not specified.

Scanned by CamScanner

Context Free languages (Non-Regular PDA Languages)

1. $L = \{a^n b^n \mid n \geq 1\}$
2. $L = \{a^n b^n c^m \mid m, n \geq 1\}$
3. $L = \{a^m b^{m+n} c^n \mid m, n \geq 1\}$
4. $L = \{a^n b^{3n} \mid n \geq 1\}$
5. $L = \{w \in (a+b)^* \mid \text{no. of a's}(w) = \text{no. of b's}(w)\}$
6. L = {all strings of balanced parenthesis}
7. $L = \{ww^R \mid w \in (a+b)^+\}$
8. $L = \{wcw^R \mid w \in (a+b)^+\}$
9. $L = \{a^m b^m c^n d^n \mid m, n \geq 0\}$
10. $L = \{a^m b^n c^n d^m \mid m, n \geq 0\}$

Closure Properties of CFL's

- The context free languages are closed under:
  - (*i*) Union operation
  - (*ii*) Concatenation
  - (*iii*) Kleene closure
  - (*iv*) Reversal operation
  - (*v*) Homomorphism
  - (*vi*) Inverse homomorphism
  - (*vii*) Substitution
  - (*viii*) Init or prefix operation
  - (*ix*) Quotient with regular language.
  - (*x*) Cycle operation
  - (*x*) Union with regular language
  - (*xi*) Intersection with regular language
  - (*xii*) Difference with regular language
- The context free languages are not closed under:
  - (*i*) Intersection
  - (*ii*) Complement
  - (*iii*) Difference, symmetric difference (XOR), NAND, NOR and any other operation which can be reduced to intersection and complement.
  - (*iv*) Subset
  - (*v*) Superset
  - (*vi*) Infinite union

Scanned by CamScanner

Decision Properties of CFL's

1. Test for membership: Decidable.
2. Test for Emptiness: Decidable.
3. Test for finiteness: Decidable.

Non-CFLs: Examples

(a) $L = \{a^n b^n c^n \mid n \geq 0\}$

(b) $L = \{a^n b^m c^n d^m \mid m, n \geq 0\}$

(c) $L = \{a^{n^2} \mid n \geq 0\}$

(d) $L = \left\{a^{2^n} \mid n \geq 0\right\}$

(e) $L = \left\{a^n \mid n \text{ is prime}\right\}$

(f) $L = \left\{a^{n!} \mid n \geq 0\right\}$

(g) $L = \left\{a^{2^{n^2}} \mid n \geq 0\right\}$

(h) $L = \{a^i b^j c^k \mid i \leq j \leq k\}$

(i) $L = \{w w^R w^R \mid w \in (a+b)^*\}$

(j) $L = \{ww \mid w \in (a+b)^*\}$

(k) $L = \{0^p \mid p \text{ is composite}\}$

(l) $L = \{0^i 1^j \mid j = i^2\}$

(m) $L = \{a^n b^n c^i \mid i \leq n\}$

(n) $L = \{0^i 1^j 0^k \mid j = \max(i, k)\}$

(o) $L = \{a^n b^n c^i \mid i \neq n\}$

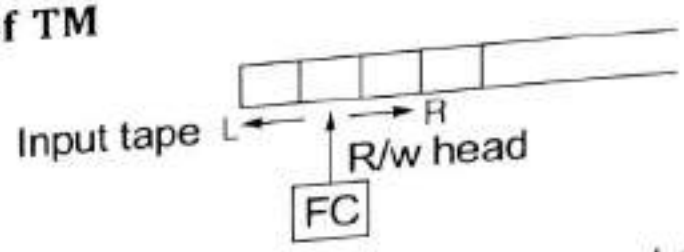
(p) $L = \{ww^R w \mid w (a+b)^*\}$

■■■

Scanned by CamScanner

Turing Machine

Configuration of TM



R/W head can move in any direction but it moves at a time only one cell, this capability is called "turn around capability"

Specification

The turing machine can be represented as a 7 tuple as follows:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

Where
Q = set of states.
Σ = input alphabet
Γ = tape alphabet
δ = transition function; for DTM $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
B = blank symbol
F = final states $F \subseteq Q$

Closure Properties of Recursive Languages

- **Recursive languages are closed under the following properties:**
  (a) Union operation
  (b) Intersection operation
  (c) Complement
  (d) Concatenation operation
  (e) Kleene closure
  (f) Positive closure
  (g) Difference, symmetric difference, XOR, NAND, NOR and any other operation which can be reduced to intersection and complement.
  (h) Reversal (transpose) operation
- **Recursive languages are not closed under following properties:**
  (a) Homomorphism of a recursive language.
  (b) Substitution of a recursive language.

Scanned by CamScanner

Closure Propterties of Recursive Enumerable Languages

- Recursive Enumerable languages are closed under the following properties:
  (a) Union operation
  (b) Intersection operation
  (c) Concatenation operation
  (d) Kleene closure
  (e) Reversal operation
  (f) Positive closure
- Recursive enumerable languages are not closed under following properties:
  (a) Complement
  (b) Difference, symmetric difference, XOR, NAND, NOR and any other operation which can be reduced to intersection and complement.

Decidability Problems for Recursive Languages

1. Membership problem: Decidable.
2. Emptiness problem: Undecidable.
3. Finiteness problem: Undecidable.
4. Equivalence problem: Undecidable.
5. Subset Problem: Undecidable.
6. Ambiguity problem: Undecidable.
7. Regularity problem: Undecidable.
8. Universality problem $L = \Sigma^*$? : Undecidable.
9. Disjointedness problem : Undecidable.

Some decidability problems related to halting turing machines (REC languages)

1. Halting problem of a (halting) turing machine: Decidable.
2. Empty word halting problem for halting turing machine : Decidable.
3. State entry problem for halting turing machine : Decidable.

Decidability Problems for Recursively Enumerable Languages

1. RE membership problem: Undecidable.
2. Emptiness problem: Undecidable.
3. Finiteness problem: Undecidable.
4. Equivalence problem: Undecidable.

Scanned by CamScanner

5. Subset Problem: Undecidable.
6. Universality problem L = Σ^*?: Undecidable.
7. Ambiguity problem: Undecidable.
8. Regularity problem: Undecidable.
9. Disjointedness problem: Undecidable.

Some decidability problems related to turing machines (RE languages)

1. Halting problem of a turing machine: Undecidable.
2. Empty word halting problem: Undecidable.
3. State entry problem: Undecidable.
4. Post correspondence problem (PCP): Undecidable.
5. Modified post correspondence problem (MPCP): Undecidable.

- **Post correspondence problem:** An instance of PCP consist of 2 lists, $A = w_1 \ldots w_k$ and $B = x_1 \ldots x_2$ of strings over some alphabet Σ. This instance of PCP has a solution if there is any sequence of integers i_1, $i_2 \ldots\ldots i_m$, with $m \geq 1$, such that $w_{i1}, w_{i2} \ldots\ldots w_{im} = x_{i1}, x_{i2} \ldots x_{im}$.
  The sequence $i_1, i_2 \ldots\ldots i_m$ is a solution to this instance of PCP.
- **Modified post correspondence problem:** In modified PCP there is an additional requirement on a solution that the first pair on A and B lists must be the first pair in the solution. More formally, an instance of modified PCP is two lists $A = w_1\, w_{i1}\, w_{i2} \ldots\ldots w_{im} = x_1\, x_{i1}\, x_{i2} \ldots\ldots x_{im}$.

Remember:

- ***Rice's theorem***: Every (non-trivial) question asked about RE languages is undecidable.
- NTM = DTM
- L is RE iff there exist a TM to accept it
- L is REC iff there exist a TM to accept it and TM halts for all $w \in \Sigma^*$
- Both RE and REC are countable.
- Only REC has membership algorithm but no membership algorithm for RE
- If L is REC than $\overline{L}$ is also REC
- If both L and $\overline{L}$ are RE then both L and $\overline{L}$ will be REC.
- Recursive languages is turing decidable.
- Recursive enumerable language is turing recognizable.

Scanned by CamScanner

Identification of Formal Languages

1. $a^{n^2} \rightarrow$ CSL
2. $a^n b^n c^n \rightarrow$ CSL
3. $\{ww \mid w \in (0, 1)^*\} \rightarrow$ CSL
4. $\{a^n b^m c^p \mid n = m = p\} \rightarrow$ CSL
5. $\{ww^R w \mid w \in (0, 1)^*\} \rightarrow$ CSL
6. $\{ww^R w^R \mid w \in (0, 1)^*\} \rightarrow$ CSL
7. $\{xww \mid x\, w \in (0, 1)^+\} \rightarrow$ CSL
8. $\{w \mid n_a(w) = n_b(w) = n_c(w)\} \rightarrow$ CSL
9. $\left\{w \left| \frac{n_a(w)}{n_b(w)} = n_c(w) \right.\right\} \rightarrow$ CSL
10. $\{a^n b^m c^n d^m\} \rightarrow$ CSL
11. $\{a^n b^m c^p \mid n = m \text{ and } m = p\} \rightarrow$ CSL
12. $a^n b^{n^2} \rightarrow$ CSL
13. $\{a^n b^{n+m} c^p\} \rightarrow$ DCFL
14. $\{w \mid n_a(w) + n_b(w) = n_c(w)\} \rightarrow$ DCFL
15. $\{a^m b^n c^m\} \rightarrow$ DCFL
16. $\{a^m b^n c^n\} \rightarrow$ DCFL
17. $\{a^n b^{n+m} c^m\} \rightarrow$ DCFL
18. $\{a^n b^n c^m d^{n/}\} \rightarrow$ DCFL
19. $\{a^n b^m c^m c^n\} \rightarrow$ DCFL
20. $\{wx\,w^R \mid w \in \{0,1\}^*, x \in \{0, 1\}\} \rightarrow$ DCFL
21. $\{ww^R \mid w \in \{0,1\}^*\} \rightarrow$ CFL
22. $\{a^m b^n c^p \mid m = n \text{ or } n = p\} \rightarrow$ CFL
23. $\{wxw^R \mid x, w \in (0,1)^*\} \rightarrow$ Regular
24. $\{wx\,w \mid x,\ w \in (0,1)^*\} \rightarrow$ Regular
25. $\{wx\,w^R \mid x,\ w \in (0,1)^+\} \rightarrow$ Regular
26. $a^* \rightarrow$ Regular
27. $a^* b^* \rightarrow$ Regular
28. $\{a^n b^n \mid n \le 10000\} \rightarrow$ Regular
29. $\{a^n b^n c^n \mid n = 10\} \rightarrow$ Regular
30. $\left\{a^{n^2} / n \ge 0\right\}^* \rightarrow$ Regular
31. $\{xww \mid x,\ w \in (0, 1)^*\} \rightarrow$ Regular

■■■

Scanned by CamScanner

Undecidability

4

Undecidablity/Decidability Table for Formal Languages

Problem	Regular Language	DCFL	CFL	Recursive Language	Recursively Enumerable Language
1.Is W in L? (member ship problem)	✓	✓	✓	✓	×
2.Is L = ϕ? (emptiness problem)	✓	✓	✓	×	×
3.Is L = Finite? (Finiteness problem)	✓	✓	✓	×	×
4.Is L1 = L2? (equivalence problem)	✓	✓	×	×	×
5.Is L1 $\subseteq$ L2? (subset problem)	✓	×	×	×	×
6.Is 'L' regular? (regularity problem)	✓	✓	×	×	×
7.Is L ambiguous (ambiguity problem)	✓[1]	×	×	×	×
8.Is L = Σ^* (Universality problem)	✓	✓	×	×	×
9.Is L1 $\cap$ L2 = ϕ? (disjointedness prob.)	✓	×	×	×	×
10.Is L = R? Where, R is given regular language	✓	✓	×	×	×
11.Is the intersection of two languages is also a language of the same type?	✓[1]	×	×	✓[1]	✓[1]
12.Is the compliment of a language is also a language of the same type.	✓[1]	✓[1]	×	✓[1]	×

Scanned by CamScanner

Closure Properties Table

Property	Regular	DCFL	CFL	CSL	REC	RE
$\cup$	✓	☒	✓	✓	✓	✓
$\cap$	✓	☒	☒	✓	✓	✓
L^c	✓	✓	☒	☒	✓	☒
$\bullet$	✓	☒	✓	✓	✓	✓
L^*, L^+	✓	☒	✓	✓	✓	✓
L^R	✓	☒	✓	No information	✓	✓
$h(L)$	✓	☒	✓		No information	No information
$h^{-1}(L)$	✓	✓	✓			
$L_1 \cap R$	✓	✓	✓	✓	✓	✓
$L_1 - R$	✓	✓	✓	✓	✓	✓
$L_1 \cup R$	✓	✓	✓	✓	✓	✓
$L_1 - L_2$	✓	☒	☒	☒	✓	☒

Some Decidable Problems Related to Theory Machines:

1. State of TM after k(finite) steps? → Decidable
2. Will a string w be accepted by TM after k(finite) steps? → Decidable
3. Will a TM hang within k(finite) steps? → Decidable
4. Whether a given TM will ever make a left move? → Decidable
5. Whether a given TM will print some non-blank character? → Decidable
6. PCP and MPCP problems on unary alphabet → Decidable.

■■■

Scanned by CamScanner

A Handbook on Computer Science

7

Compiler Design

CONTENTS

1. Lexical Analysis 223
2. Parsing 226
3. Syntax Directed Translation 233
4. Runtime Environments 235
5. Intermediate, Target Code Generation and Code Optimization 237

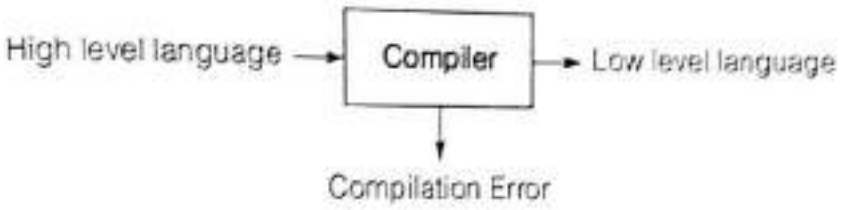
Scanned by CamScanner

Lexical Analysis

1

Introduction

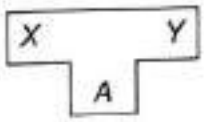
A program written in one high level language (source language) producing output in low level language (object or target language) is called compiler.



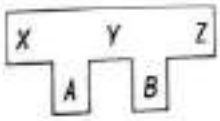
Cross Compiler

A compiler that runs on one machine '*A*' and produces a code for another machine '*B*'.

Example: (a) Compiler runs on machine *A*, which takes input language *X* and produces output language *Y*.



(b) Compiler runs on machine *A* and produces code *Y* for another machine *B*.



Compiler Parts

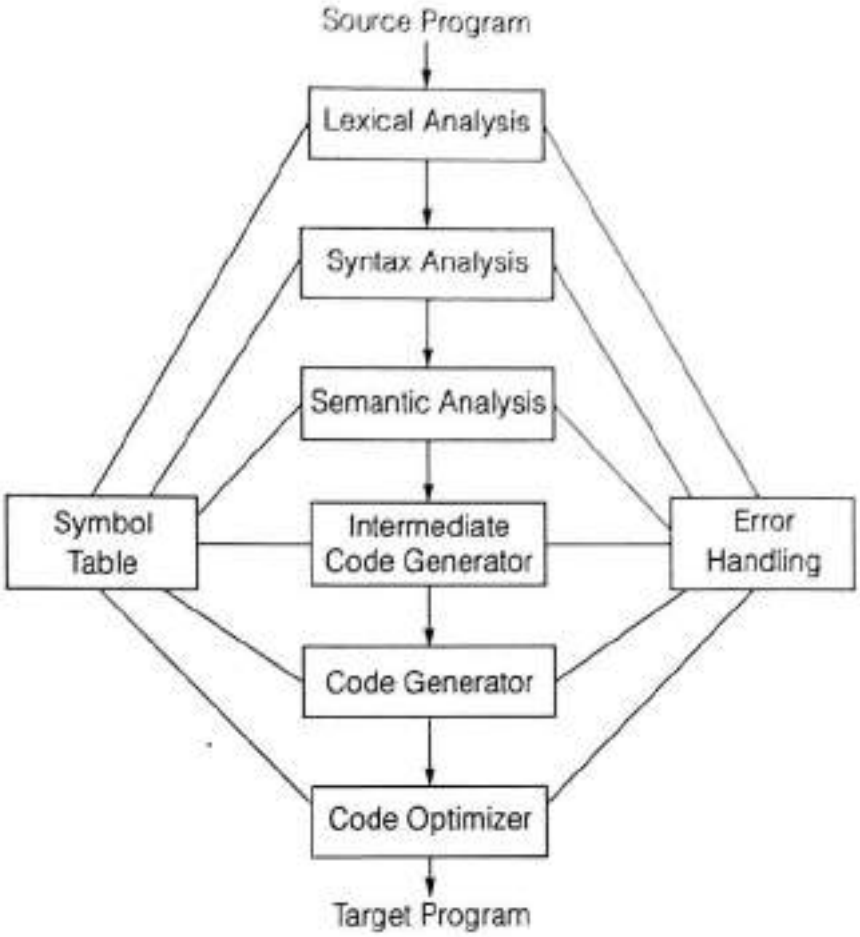
There are two major parts of a compiler:

1. **Analysis Phase:** An intermediate representation is created from the given source program
   (*i*) Lexical Analyzer
   (*ii*) Syntax Analyzer
   (*iii*) Semantic Analyzer
2. **Synthesis Phase:** Equivalent target program is created from intermediate representation
   (*i*) Intermediate Code Generator

Scanned by CamScanner

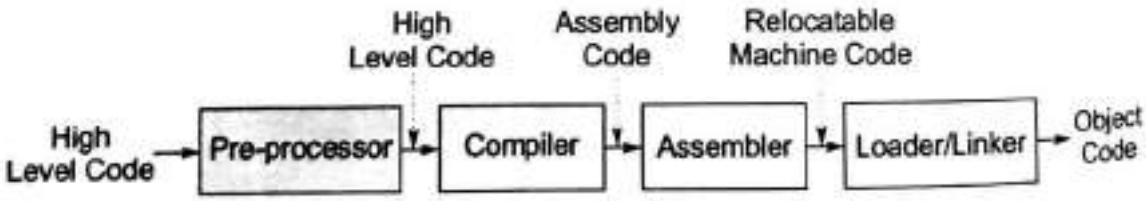
(*ii*) Code Generator

(*iii*) Code Optimizer



Compiler Phases

Language Processing System



Language Processing System Using Compiler

Lexical Analysis

- Lexical analyzer reads a source program character by character to produce tokens.

- A token can represent more than one lexeme, additional information should be held for that specific lexeme. This additional information is called as attribute of the token.

Scanned by CamScanner

- Token type and its attribute uniquely identify a lexeme
- Regular expressions are used to specify patterns and finite automata is used to recognise tokens.
- Tokens identified by lexical analyzer: Keywords, constants, identifiers, operators and symbols.

Designing of Lexical Analyzer

- Write a formal description of the tokens and use a software tool that constructs table-driven lexical analyser.
- Design a state diagram that describes the tokens and write a program that implements the state diagram.
- Design a state diagram that describes the tokens and hand-construct a table-driven implementation of the state diagram.

■■■

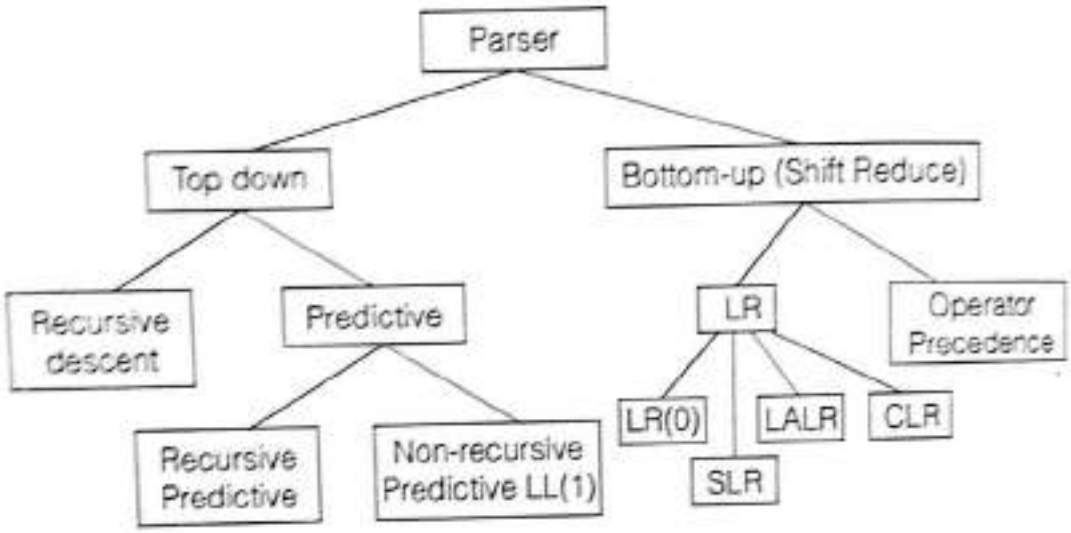
Scanned by CamScanner

Parsing

2

Introduction

- Syntax analyzer is also known as parser.
- The syntax of a programming language is described by a context free grammar.
- A parser works on stream of tokens which are generated by lexical analyser.



Ambiguous Grammar

For a given input string if there exists 'more than one parse tree' then that grammar is called as "ambiguous grammar".

Left Recursion

The grammar : $A \rightarrow A\alpha | \beta$ is left recursive. Top down parsing techniques cannot handle left recursive grammars, so we convert left recursive grammars into right recursive.

Left Recursive Elimination

Left recursive : $A \rightarrow A\alpha | \beta \Rightarrow \boxed{\begin{array}{l} A \rightarrow \beta A' \\ A' \rightarrow \alpha A' | \varepsilon \end{array}}$ (non left recursive)

Problem with left recursive grammar is that if such a grammar is used by top down parser (uses Left Most Derivation), then there is a danger of going into "infinite loop" while string is parsing.

Scanned by CamScanner

Left Factoring

- A predictive parser (a top down parser without backtracking) insists that the grammar must be left-factored.
- Left factoring avoids backtracking in parser but is not a solution for elimination of ambiguity.
- If a grammar has common prefixes in r.h.s. of non-terminal then such grammar need to be left factored by eliminating common prefixes as follows:

$$A \to \alpha\beta_1 | \alpha\beta_2 \Rightarrow \boxed{\begin{array}{l} A \to \alpha A' \\ A' \to \beta_1 | \beta_2 \end{array}} \text{ (left factored grammar)}$$

Parsers

1. **Top-down parsers:** Starting at the root (the starting symbol) and proceeding to the leaves. This parsers are easy to write, actual code capable of being derived directly from the production rules, but cannot always applied as an approach.
2. **Bottom-up parsers:** Start at the leaves and move up towards the root. Bottom-up parsers, can handle a larger set of grammar.

Top Down Parsing (TDP)

1. **Recursive-descent parsing:**
   (*i*) Follows Brute Force mechanism (Suffers from Backtracking).
   (*ii*) Not efficient, general purpose, not widely used.
2. **Predictive parsing:**
   It can uniquely choose a production rule by just looking the current symbol.
   (*i*) **Recursive predictive parsing:** each non terminal corresponds to a procedure.
   (*ii*) **Non-recursive predictive parsing:** Table driven also called as LL(1) parser.

LL(1) Parser/Predictive Parser

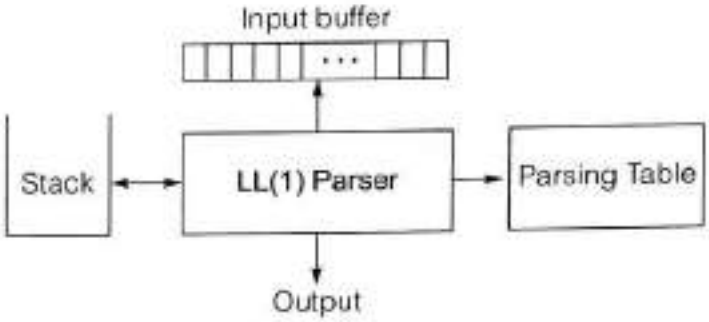
- LL(1) grammar is unambiguous, left factored and non left recursive.
- Top down parser follows left most derivation.

LL(1)
- 1 — Number of Look Ahead Symbols = 1
- L — Left Most Derivation
- L — Left to Right Scanning

Scanned by CamScanner

- FIRST(A) is a set of the terminal symbols which occur as first symbols in strings derived from A.
- Follow(A) is the set of terminals which occur immediately after the nonterminal A in the strings derived from the starting symbol.

Configuration of LL(1) Parser



Configuration of LL(1) Parser

LL(1) Parsing Table Construction

- First( ) and Follow( ) sets are used to construct LL(1) parsing table.
- For each rule $A \rightarrow \alpha$ in grammar G:
  - (*i*) For each terminal 'a' contained in FIRST(A) add $A \rightarrow \alpha$ to [A, a] entry in parsing table if α derives '*a*' as the first symbol.
  - (*ii*) If FIRST(A) contain 'ε', for each terminal 'b' in FOLLOW(A), add $A \rightarrow \varepsilon$ to [A, b] entry in parsing table.

LL(1) Grammar

- To check whether given grammar is LL(1) or not:

$$\left.\begin{array}{l} (i)\ \text{If } A \rightarrow \alpha_1 | \alpha_2 \Rightarrow \{\text{FIRST}(\alpha_1) \cap \text{FIRST}(\alpha_2) = \phi\} \\ (ii)\ \text{If } A \rightarrow \alpha | \varepsilon \Rightarrow \{\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \phi\} \end{array}\right\} \text{is LL(1)}$$

- A grammar where its LL(1) parsing table has free from multiple entries is said to be LL(1) grammar.
- A left recursive grammar cannot be a LL(1) grammar
- If a grammar is not left factored, it cannot be a LL(1)
- An ambiguous grammar cannot be a LL(1) grammar.

Bottom up Parsing

The process of constructing the parse tree starting from input string and reaching the start symbol of the grammar is known as Bottom Up Parsing.

Scanned by CamScanner

- **Handle**: The part of the input string that matches with RHS of any production is called as handle.
- **Handle pruning**: The process of finding the handle and replacing the handle by LHS of corresponding production is called "handle priority".
- Bottom up parsing is also known as shift reduce parser
- Bottom up parsing can be constructed for both ambiguous and unambiguous grammar.
- For unambiguous grammar, LR parser can be constructed and Operator Precedence Parser (OPP) can be constructed for both ambiguous and unambiguous grammar.
- Bottom up parsing is more powerful than top down parser.
- Average time complexity is $O(n^3)$ and handle pruning is major overhead for bottom up parsing.
- Bottom up parser simulates the reverse of right most derivation.
- **LR(k) parser:**

L R (k)
- k — Number of Look Ahead Symbols= k
- R — Reverse of right most derivation
- L — Left to Right Scanning

Shift Reduce Parsers

Operator Precedence Parser (OPP)

- Constructed for 'operator precedence grammar', a grammar which do not contain epsilon productions and do not contain two adjacent non-terminals on *R.H.S.* of any production. Operator precedence parser grammar is provided with precedence rules.
- **Operator Precedence Parsing Algorithm:** Let '*a*' is top of stack symbol and '*b*' is symbol pointed by input pointer.
  (*i*) If $a \doteq b \doteq \$$, successfully completion of parsing
  (*ii*) If $a \lessdot b$ or $a \doteq b$ then push '*b*' onto stack and advance input points to next symbol.
  (*iii*) If $a \gtrdot b$ then pop '*a*' and reduction takes place for '*a*'.
- Operator precedence grammar may be either ambiguous or unambiguous grammar

Scanned by CamScanner

- Example for operator precedence table:

	id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	

LR Parsers

- **LR(0) Parser:**
  - (*i*) LR(0) parser is constructed for LR(0) grammar in which the parse table is free from multiple entries (conflicts).
  - (*ii*) **Conflict in LR(0) parser:**
    - (a) **Shift reduce (SR) conflict:** When the same state in DFA contains both shift and reduced items.

      $A \to \alpha \cdot x\beta$ (shifting)

      $B \to \gamma \cdot$ (reduced)
    - (b) **Reduce Reduce (RR) conflict:**

      $\left.\begin{array}{l} A \to \alpha \cdot \text{ (reduced)} \\ B \to \gamma \cdot \text{ (reduced)} \end{array}\right\}$ Both are in same state of DFA
  - (*iii*) Closure( ) and goto( ) functions are used to create canonical collection of LR items.
  - (*iv*) LR(0) parsing table construction:
    - (a) If goto $(I_i, a) = I_j$, then set action $[i, a] = S_j$ (shift entry)
    - (b) If goto $(I_i, A) = I_j$, then set action $[i, A] = j$ (state entry)
    - (c) If I_i contains $A \to \alpha \cdot$ (reduced production) then action [*i*, all entries] = R_P, (reduced entry), where '*P*' is production number $(P : A \to \alpha)$
    - (d) If $S' \to S \cdot$ is in I_i, then set action [*i*, $] to accept.
  - (*v*) If a state contains either *SR* or *RR* conflicts then that state is called 'Inadequate State'.
- **Simple LR(1) Parser:**
  - (*i*) **Conflicts in SLR(1) parser:**
    - (a) **Shift Reduce (SR) conflict:**

      $A \to \alpha \cdot x\beta$ (shift)

      $B \to \gamma \cdot$ (reduce)

Scanned by CamScanner

If FOLLOW(B) $\cap \{x\} \neq \phi$

(b) **Reduce Reduce (RR) conflict:**

$A \rightarrow \alpha\cdot$

$B \rightarrow \gamma\cdot$ If FOLLOW(A) $\cap$ FOLLOW(B) $\neq \phi$

(*ii*) **SLR(1) parsing table construction:**

(a) SLR(1) parsing table construction is same as LR(0) except reduced entries.

(b) If I_i contains $A \rightarrow \alpha$ (reduced production), then find FOLLOW(A) and for every a in FOLLOW(A), set action $[i, a] = R_p$ {where P is production number}

(*iii*) SLR(1) is powerful then LR(0)

(*iv*) Size of SLR(1) parse table is same as size of LR(0) parse table.

(*v*) Every LR(0) grammar is SLR(1) but every SLR(1) need not be LR(0).

- **Canonical LR(1) or LR(1) Parser:**

(*i*) **CLR(1) parsing table construction:** Except reduced entries, remaining entries are same as SLR(1).

(*ii*) If I_i contains $A \rightarrow a\cdot, \$|a|b$ then reduced entry (R_p) $A \rightarrow a$ in row 'i' under the terminals given by look ahead ($, a, b) in LR(1) items [R_p entry entered into [i, $], [$i$, a] and [i, b]]

- **Look ahead LR(1) Parser:**

(*i*) It is constructed from CLR(1), if two states are having same productions but may contain different look aheads, those two states of CLR(1) are combined into a single state in LALR(1).

(*ii*) There is no chance of SR conflict but there may be chance for RR conflicts in LALR(1) after combining the states of CLR(1) parser.

(*iii*) Conflicts in CLR(1) and LALR(1)

(a) SR conflict: $A \rightarrow \alpha\cdot x\beta, a$
$B \rightarrow \gamma\cdot, x|y$

(b) RR conflict: $A \rightarrow \alpha\cdot, a|t_1$
$B \rightarrow \gamma\cdot, a|t_2$

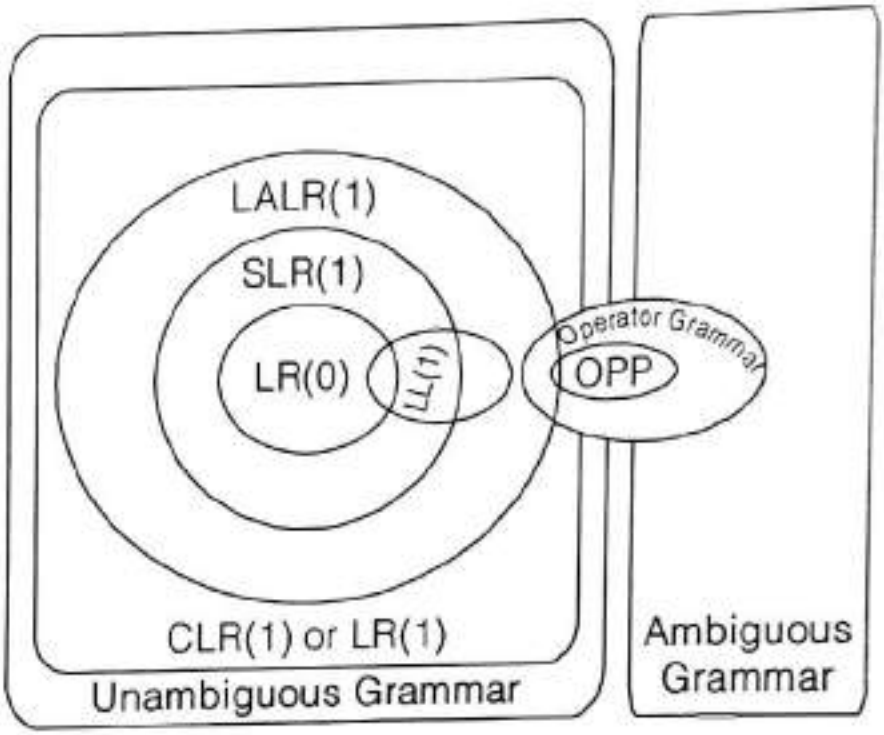
(*iv*) Every LALR(1) grammar is CLR(1) but every CLR(1) grammar need not be LALR(1).

(*v*) LALR(1) parsers are often used in practice as parsing tables are smaller then CLR(1) parsing.

Scanned by CamScanner

Parsers Comparision

- OPP < LL(1) < LR(0) < SLR(1) ≤ LALR(1) ≤ CLR(1)
- Number of states (SLR(1)) = Number of states (LALR(1)) ≤ Number of states (CLR(1))



- LR(0) ⊂ SLR(1) ⊂ LALR(1) ⊂ CLR(1)
- LL(1) ⊂ LALR(1) ⊂ CLR(1)

■■■

Scanned by CamScanner

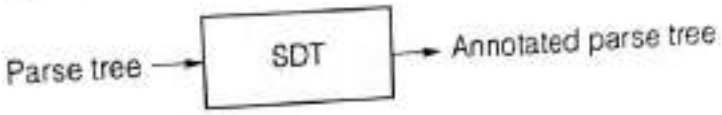
Syntax Directed Translation

3

Syntax Directed Translation (SDT)

- The CFG with semantic actions is called as SDT or the translation rules placed in RHS side of production.

Example: $S \rightarrow AB$ {print (*)}
$A \rightarrow a$ {print (1)}
$B \rightarrow S$ {print (2)}



- SDT can be constructed
  (*i*) to store the information in symbol table
  (*ii*) to verify the variable declaration
  (*iii*) to construct DAG
  (*iv*) to perform type checking, type conversion, type evaluation.
  (*v*) to evaluate algebraic expression etc.
- A parse tree which shows attributes at each and every node is called as ANNOTATED/DECORATED parse tree.

Types of Attributes

1. **Synthesized attribute:** Attribute whose value is evaluated in terms of attribute values of it's children.
2. **Inherited attribute:** Attribute whose value is evaluated in terms of attribute values of siblings or parent.

Syntax Directed Definition (SDD)

1. **S-attributed SDD:**
   (*i*) Uses "only synthesized attributes"
   (*ii*) Semantic actions can be placed at right end of the rule
   (*iii*) Follows bottom up approach
   $A \rightarrow BC \{A \cdot i = f(B \cdot i \text{ or } C \cdot i)\}$
2. **L-attributed SDD:**
   (*i*) SDT in which attributes are synthesized or restricted to inherit either from 'parent' or 'left siblings',
   (*ii*) Follows DFS and top down approach

Scanned by CamScanner

Example: $A \rightarrow BCD$ $\{C \cdot i = f(A \cdot i \text{ or } B \cdot i)\}$ or
$\{D \cdot i = f(A \cdot i \text{ or } B \cdot i \text{ or } C \cdot i)\}$ or
$\{B \cdot i = f(A \cdot i)\}$

- Every *S*-attributed SDT is L-attributed SDT.
- Every *L*-attributed SDT need not be S-attributed SDT .
- SDT follows depth first search notation.

■■■

Scanned by CamScanner

Runtime Environments

4

Run-time Storage Organisation

The runtime environment is the structure of the target computers registers and memory that serves to manage memory and maintain information needed to guide a programs execution process.

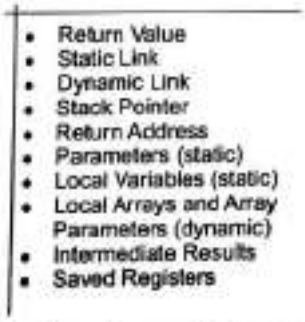
Types of Runtime Environments

1. **Fully Static Runtime Environment:** Fully static runtime environment may be useful for the languages in which pointers or dynamic allocation is not possible in addition to no support for recursive function calls
   (*i*) Every procedure will have only one activation record which is allocated before execution.
   (*ii*) Variables are accessed directly via fixed addresses.
   (*iii*) Little book keeping overhead; i.e., at most return address may have to be stored in activation record.
   (*iv*) The calling sequence involves calculation of each argument address and storing into its appropriate parameter location and saving the return address and then a jump is made.
2. **Stack based Runtime Environment:** In this, activation records are allocated (push of the activation records) whenever a function call is made. The neccessary memory is taken from stack portion of the program. When program execution returns from function the memory used by the activation record is deallocated (pop of the activation record). Thus, stack grows and shrinks with the chain of function calls.
3. **Fully Dynamic Runtime Environment:** Functional languages such as Lisp, ML, etc. uses this style of call stack management. Sailently, here activation records are de-allocated only when all references to them have disappeared, and this requires that activation records to dynamically freed at arbitrary times during execution. Memory manager (garbage collector) is needed.
   The data structure that handles such a management is heap and thus this is also called as heap management.

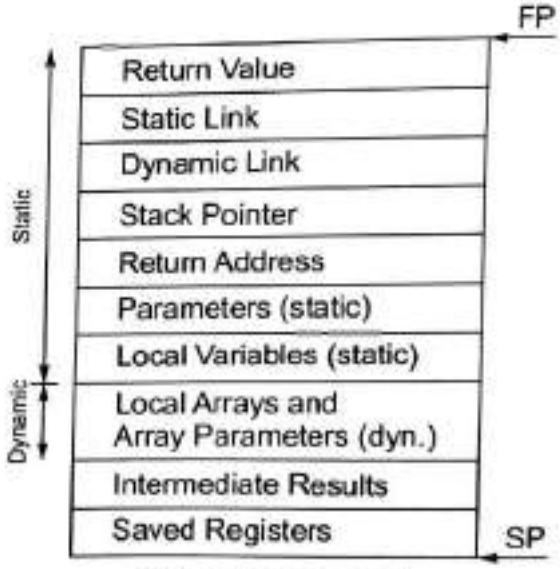
Scanned by CamScanner

Activation Records

- Information needed by a single execution of a procedure is managed using a contiguous block of storage called "activation record".
- An activation record is allocated when a procedure is entered and it is deallocated when that procedure is exited.
- It contains temporary data, local data, machine status, optional access link, optional control link, actual parameters and returned value.
  - (*i*) Program Counter (PC) whose value is the address of the next instruction to be executed.
  - (*ii*) Stack pointer (SP) whose value is top of the (top of stack, tos).
  - (*iii*) Frame Pointer (FP) which points to the current activation record.



Activation Record Contents



Activation Record

Note: .....

- Activation records are allocated from one of static area (like Fortran 77), stack area (like C or Pascal) and heap area (like lisp).

.....

■■■

Scanned by CamScanner

Intermediate, Target Code Generation and Code Optimization

5

Intermediate Code

- Intermediate codes are machine independent codes, but they are close to machine instructions.
- Syntax trees, Postfix notation, 3-address codes, DAG can be used as intermediate language.
- Three-address code:
  - (*i*) Quadruples (4 fields: Operator, Operand1, Operand2, Result)
  - (*ii*) Triples (3 fields: Operator, Operand1, Operand2)
  - (*iii*) Indirect triples.

Need for Intermediate Code Generation

1. Suppose we have n-source languages and m-target languages:
   - (*i*) **Without Intermediate Code,** we will change each source language into target language directly. So, for each source-target pair we will need a compiler. Hence, we need (n × m) compliers.
   - (*ii*) **With Intermediate Code,** we need n-compilers to convert each source language into intermediate code and m-compilers to convert intermediate code into m-target languages. Thus, we need only (n + m) compilers.
2. Re-targeting is facilitated; a compiler for a different machine can be created by attaching a back-end (which generate target code) for the new machine to an existing front-end (which generate intermediate code).
3. A machine independent code-optimizer can be applied to the intermediate code.
4. Intermediate code is simple enough to be easily converted to any target code.
5. Complex enough to represent all the complex structure of high level languages.

Scanned by CamScanner

Code Generation

A code generator has three primary tasks: instruction selection, register allocation and assignment, and instruction ordering.

1. The code generator is given the intermediate text in any one of its form.
2. But in specific algorithms, we deal with quadruples or in some case parse trees as the intermediate forms.
3. Necessary semantic checking is done.
4. The data areas and offsets have been determined for each name that information is available from the symbol table.

Code Optimization

- Optimizations applied on target code (assembly code) is machine dependent optimization.
- Optimization applied on 3-address code is machine independent optimization.
- **Basic Block:** Sequence of consecutive statements in which context enters at beginning and leaves at end without halt or possibility of branching except at the end.
- The optimization which is performed within the block is local optimization.

Machine Independent Optimizations

1. **Loop Optimization:** Process of optimizing within the loop.

   Characteristics of loop optimization:

   (*i*) Code motion/Loop invariant elimination/Frequency reduction:
   Reduce the evaluation frequency of expressions.
   Bring loop-invariant statements out of the loop

   (*ii*) **Loop Unrolling:** To execute less number of iterations.

   (*iii*) **Loop Jamming/Loop Fusion/Loop Combining**: Combine the bodies of two loops whenever they are sharing same index variable.
2. **Constant Folding:** Replacing the value of constant during compilation is called as folding.
3. **Constant propagation:** Replacing the value of an expression during compile time is known as constant propagation.

Scanned by CamScanner

4. **Copy Propagation:**

Example:

$x = y \quad \Rightarrow \quad x = y$

$z = 1 + x \qquad z = 1 + y$

5. **Dead Code Elimination:** Removes instructions without changing the behaviour (using DAG)

6. Common Subexpression Elimination/Redundant Code Elimination:

Example:

$x = (a + b) - (a + b)/4 \quad \Rightarrow \quad c = a + b$

$x = c - c/4$

7. **Induction Variables and Strength Reduction:**

An induction variable is used in loop for the following kind of assignment

$i = i +$ constant

Strength reduction means replacing costly operator by simple (cheap/low strength) operator.

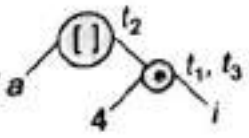
Example: $y = 2 * x \Rightarrow y = x+x$

Directed Acyclic Graph (DAG)

- DAG is used to eliminate the common subexpression
- **Procedure to construct DAG:** Whenever we are going to create a node, first verify that whether node is already created or not. If it is already created then make use of the existing one instead of going for new creation.

Example:

$$\left.\begin{array}{l} t_1 = 4*i \\ t_2 = a[t_1] \\ t_3 = 4*i \end{array}\right\} \Rightarrow$$



■■■

Scanned by CamScanner

A Handbook on
Computer Science

8

Operating System

CONTENTS

1. Processes and Threads 241
2. IPC, Concurrency and Synchronization 244
3. Deadlock 250
4. CPU Scheduling 253
5. Memory Management and Virtual Memory 257
6. I/O and File Systems, Protection and Security 267

Scanned by CamScanner

Processes and Threads

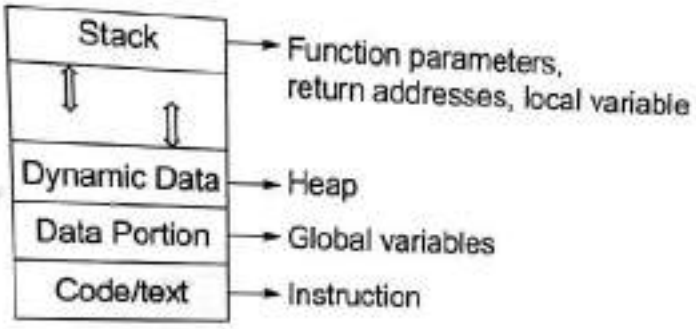
1

Process Concepts

- **System call:** It is a request made by a user program to the operating system in order to get any kind of service.
- **Process:** Program under execution. It is an instance of a program.

Program	Process
Passive entity	Active entity
Resides on disk	Resides in main memory

- **Abstract view of process:**



- **Operations on process:**
  (*i*) Create a process
  (*ii*) Scheduling/running, block a process, suspend, resume
  (*iii*) Terminate a process
- **Dual modes of operation in operating system:**
  (*i*) User Mode/Non privileged Mode
    (a) Preemption is always possible
    (b) User program executes in this mode
  (*ii*) Kernel Mode/Supervisory/System/Privileged Mode
    (a) No preemption (atomic execution)
    (b) OS components executes in this mode
- **Process Control Block (PCB):** Each process is represented by a PCB. It is associated with every process to maintain process ID, starting address of page table, open file information, CPU information, security information, etc.

  Attributes of PCB:
  (*i*) Process ID (PID) and Parent Process ID (PPID)

Scanned by CamScanner

(*ii*) Process State
(*iii*) Program Counter
(*iv*) CPU Registers
(*v*) CPU Scheduling Information
(*vi*) Memory Management Information
(*vii*) Accounting Information
(*viii*) Address space for the process
(*ix*) User Identification
(*x*) I/O status
(*xi*) List of open files

Process State Diagram

- In the wait state, there will be multiple number of processes that w perform I/O operations simultaneously.
- When the process is in ready running or wait state, it is residing in ma memory, otherwise the process is in secondary memory.

Threads

Problems with Traditional Processes

- Traditional processes have separate address spaces.
- Process creation and switching is expensive
- Sharing memory areas among processes is non-trivial.

Scanned by CamScanner

Thread

Thread is a basic unit of CPU utilization that shares along with the other threads of the same process.

- The address space consists of code section, data section and other common resources like files, signals, semaphore etc.
- Threads also contain thread specific data which is not shared along with the peer threads of the same process.
- Each thread maintains its own register set and a stack to monitor the control within the code section of the process.
- Thread is a light weight process.
- Context switch overhead is less in thread based systems compared to process based systems.

Difference between ULT and KLT

User Level Threads (ULT)	Kernel Level Threads (KLT)
(a) ULTs are created by the application packages or libraries at the user level without taking any support of Operating System (OS).	(a) Scheduling must be through OS.
(b) Thread management is done at user mode.	(b) Thread management is done at Kernel mode
(c) Fastest context switching	(c) Lowest context switching
(d) Entire process gets blocked when a thread requires an I/O device or any system call.	(d) No other process get blocked except the specific thread which needs I/O will be blocked.

Benefits of Multithreading

1. Responsiveness
2. Resource sharing
3. Economical Design (contest switch overhead is less).
4. Utilization of multiprocessor architectures.

Multithreading Models

1. **Many-to-one model:** It maps many user level threads to one Kernel thread, multiple threads cannot run in parallel on multiprocessors.
2. **One-to-one model:** It maps each user thread to a Kernel thread. Creating a user thread requires creating the corresponding Kernel thread.
3. **Many-to-many model:** It multiplexes many user threads to a smaller or equal number of Kernel threads.

■■■

Scanned by CamScanner

IPC, Concurrency and Synchronization

2

Inter-Process Communication

- **Independent processes:** Two processes can execute independently and they will not affect the output with the order of execution.
  - (*i*) No states shared with other processes.
  - (*ii*) Order of scheduling does not matter.
  - (*iii*) output does not depend on order of execution of processes.
- **Co-operating processes:** Two or more processes are said to be co-operating iff they get affected by or affects the execution of other process.
  - (*i*) Co-operating processes require an Inter-Process Communication (IPC) mechanism that will allow them to exchange data and information like messages, pipes, shared memory, shared variables.
  - (*ii*) Shared resources, speed-up and modularity are the advantages of co-operating processes.
- **Competing processes:** Two or more processes are said to be competing iff they race to access a shared resource concurrently (simultaneously).

IPC Models

- Shared Memory.
- Message Passing.
  - (*i*) Synchronous or Asynchronous Communication
  - (*ii*) Direct or Indirect Communication

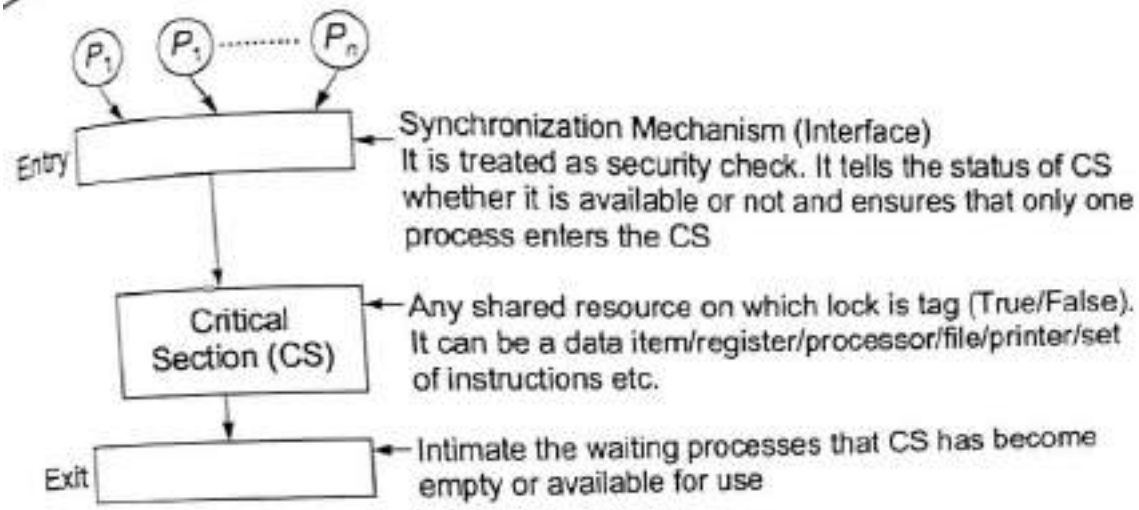
Concurrency

- Able to run multiple applications at the same time.
- Allows better resource utilisation.
- Better average response time of applications.
- Achieves better performance.

Synchronization

Synchronization is to use atomic operations to ensure co-operation between processes.

Scanned by CamScanner



- **Critical Section:** That portion of the program text where shared resources are accessed.
- **Race Condition:** The final output of any variable depends on the execution sequence of the process. This type of condition is called as race condition.
- **Requirements for Synchronization Mechanism**
  - (*i*) **Mutual Exclusion:** It states that no two processes may be simultaneously present in the critical section (i.e. at the same time).
  - (*ii*) **Progress:** No process running outside the critical section should block the other interested process from entering into critical section when critical section is free.
  - (*iii*) **Bounded Waiting:** Due to mutual exclusion waiting time of a process must be definite (This is achieved by writing release of locks without forgetting).
- **Lack of Synchronization leads to:**
  - (*i*) Inconsistency
  - (*ii*) Loss of data
  - (*iii*) Deadlock
- **Locking Mechanisms:**
  - (*i*) The test and set operation
  - (*ii*) Busy waiting
  - (*iii*) Spin Locks

Semaphore

- It is special variable whose value indicates information about lock.
- **Types of Semaphore:**
  - (*i*) Counting Semaphore
  - (*ii*) Binary Semaphore

Scanned by CamScanner

- Operations on Semaphore:
  - (*i*) **Wait (P) (Down):**
    - (a) It is used before entering critical section.
    - (b) It decrements semaphore value by 1.
    - (c) If resource is not free then block the process.
  - (*ii*) **Signal (V) (Up):**
    - (a) It is used after critical section to release the acquired lock.
    - (b) It increments semaphore value by 1.

Wait (P)
Access Critical Section
Signal (V)

⇒ Semaphore operations are executed atomically

- **Binary Semaphore:**

Semaphore *b*;
initial value *b* = 1;

```
wait (b)
{
    while (b==0);
    b - -;
}
```

```
signal (b)
{
    b++;
}
```

- **Limitations:**
  - (*i*) There is busy wait called spin locks in binary semaphore.
  - (*ii*) Binary semaphore can work on a single instance of resource.
  - (*iii*) Binary semaphore wait operation should complete for a single process only.
- **Counting Semaphore:**

Semaphore c;
initial value c ≡ Number of instances of type *R*

```
wait (c)
{
c- -;
if(c < 0)
{
place current process
 in waiting queue;
}
}
```

```
signal (c)
{
c++;
if(c <= 0)
{
wake-up one waiting process on 'c';
}
}
```

Scanned by CamScanner

ote:

- Negative value of counting semaphore indicates number of request in waiting queue waiting on type *R*.
- Positive value indicate number of instance of type *R* currently free for allocation and there is no request in waiting queue on type *R*.
- When we use multiple semaphores then it is highly important to follow proper order of semaphores (Sometimes it unnecessarily blocks the critical section).

lassical Problems on Synchronization using Semaphore

. Producer Consumer Problem.

. Reader Writer Problem:

(*i*) N readers are allowed simultaneously
(*ii*) If a reader is reading
  (a) New reader is allowed
  (b) New written is NOT allowed
} Shared Lock

(*iii*) If a writer is writing
  (a) New reader
  (b) New writer
  } Not allowed
} Exclusive Lock

(*iii*) Dining Philosopher Problem: Number of philosopher N ≥ 2.

Solution:

(a) **(*N* – 1) and 1**

- Left then right fork
- Right then left fork (Maximum concurrency)

(b) ***N***

- **ODD** — First Left then Right
- **EVEN** — Right then Left fork

Peterson Solution for Two Process and One shared Resource

```
# define N 2
# define TRUE 1
# define FALSE 0
int turn;
int interested [N] = {FALSE};
void entry-section (int process)
{
    int other;
    other = 1 – process;
```

Scanned by CamScanner

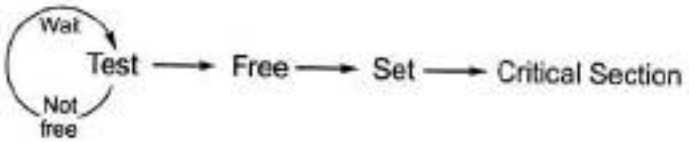
```
        interested [process] = TRUE;
        turn = process;
        while (interested [other] == TRUE && turn ==process);
    }
        Critical section
        void exit section (int process)
        {
          interested [process] = FALSE;
        }
```

- Peterson solution can work for only two processes and one shared resource.
- Peterson solution has a drawback called busy wait or spinlock.
- Peterson solution needs operating system intervention so that only one waiting process enters into critical section.
- It suffers from priority inversion problem.

Test and Set Hardware Instructions

Test: To check whether resource is free or not

Set: Sets LOCK




```
Boolean Test and set( )
  {
    if(Locked == T)
    {                    } Test
      Return(T)
    }
  else
    {
      Locked = T         } Set
      Return(F)
    }
  }
```

```
While(1)
  {
    while (Test and Set( ))
    {
      Do-nothing
    }
  Access Critical Section
  Locked = F  // Release of Lock
  }
```

Monitor

- Monitors can be used to separate "use locks for mutual exclusion" and "condition variables for scheduling constraints".
- Monitor is a special kind of module or package which includes variables and data structures that are all grouped together at one place.

Scanned by CamScanner

- Procedures running outside the monitor cannot access the monitor's internal variable and data structures. However they can activate monitor's internal procedures.
- Monitors have an important property that only one process can be active in the monitor at any time.
- A monitor is a lock with zero or more conditional variable for managing concurrent access to share data. The lock provides mutual exclusion to the shared data. A conditional variable allows a queue of waiting processes while being inside a critical section. This property is ensured by using a binary semaphore at the beginning and at the end of monitor.

Spinlock and Busy Waiting

Spinlock is a lock which causes a thread trying to acquire it to simply wait in a loop ("spin") while repeatedly checking if the lock is available. Since the thread remains active but is not performing a useful task, the use of such a lock is a kind of **busy waiting**.

■■■

Scanned by CamScanner

Deadlock

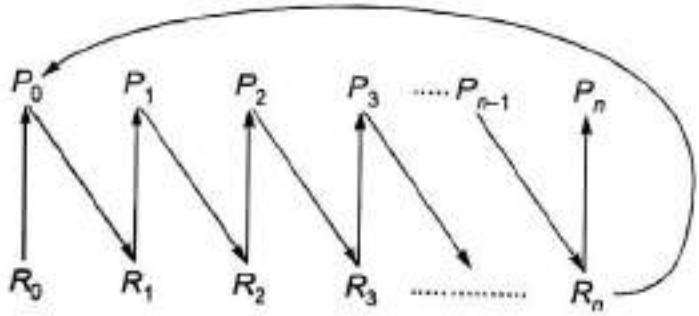
3

Introduction

Two processes are involved in dead lock iff they wait for happening of an event which would never happen. Minimum two processes are required for deadlock.

Necessary Conditions for Deadlock

1. **Mutual Exclusion:** Only one process at a time can use the resource.
2. **Hold and Wait:** A process *'A'* holding a resource and waiting for another resource held by process *'B'*.
3. **No Preemption:** A resource can be released only voluntarily by the process holding it after completion of task.
4. **Circular Wait:** Let $P_0, P_1, ..., P_n$ be the processes and $R_0, R_1, ..., R_n$ be the resources. P_0 is holding R_0 and waiting for R_1 which is held by P_1.

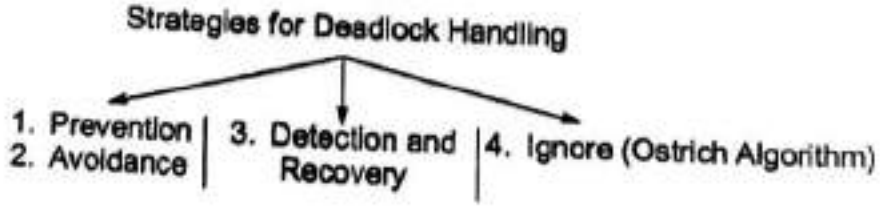


Note: .....

- If Resource Allocation Graph (RAG) has single instance and multiple instances or only multiple instances, cycle is only necessary condition for deadlock.
- If RAG has single instance resource cycle is necessary and sufficient condition for deadlock.

.....

Strategies for Deadlock Handling



Scanned by CamScanner

Prevention

It involves the following:

- Infinite resources
- No sharing
- No waiting
- Preempt resources
- Allocate all resources at the beginning
- Make everyone use the same ordering in accessing resources.
- A combination of techniques.

Avoidance through Banker's Algorithm

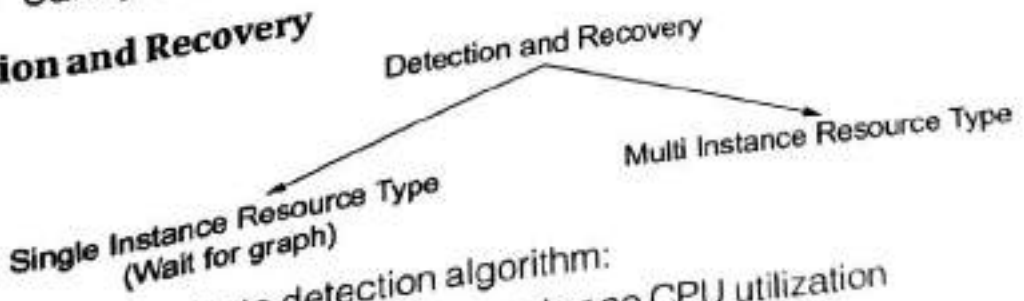
- Single instance of resources using resource allocation graph.

Note:

- A system is said to be safe iff the needs of all the processes could be satisfied with the available resources in some order.
- The order is known as safe sequence
- There may be multiple safe sequences at any given time.
- Need = Max – Allocation

- Multiple Instance of resources:
  - (*i*) Resource Request
  - (*ii*) Safety Algorithm

Detection and Recovery



- When we activate detection algorithm:
  - (*i*) CPU utilization has decreased or no CPU utilization
  - (*ii*) Most of the processes are blocked.
- Deadlock detection and Recovery involves the following approach:
  - (*i*) Scan the resource allocation graph
  - (*ii*) Detect a cycle in graph
  - (*iii*) Recovery From Deadlock

Scanned by CamScanner

Deadlock Recovery

1. **Process Termination:** Two methods are available
   (*i*) Abort all deadlocked processes
   (*ii*) Abort one process at a time until the deadlock cycle is eliminated.
2. **Resource Preemption:** To eliminate deadlock using resource preemption, we successfully preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken. Here three issues need to be addressed:
   (*i*) Selecting a Victim: Which resource should be preempted.
   (*ii*) Rollback: Rollback to safe state and restart from that state.
   (*iii*) Starvation

Note: .....

$$\boxed{r \geq p(n-1)+1;}$$

r = number of available resources (deadlock free)

n = maximum number of resources needed by each process.

p = number of processes.

■■■

Scanned by CamScanner

CPU Scheduling

4

Introduction

Multiprogramming

- The objective of multiprogramming is to have some process use CPU cycles at all times, to maximize CPU utilization.

Time Sharing

- The objective of time sharing is to switch the CPU time among processes so frequently that users can interact with each program while it is running.
- The response time should be short.

Scheduling Queues

- **Job Queue:** As processes enter the system, they are put into job queue.
- **Ready Queue:** The process that are residing in main memory and are ready and waiting to execute are kept in ready queue.
- **Device Queue:** List of processes waiting for a particular I/O device. Each device has its own device queue.

Scheduler

A scheduler handles the removal of the running process from the CPU and the selection of the next running process.

A scheduler is implemented to achieve the following:

1. To minimize the waiting time.
2. To maximize the throughput (number of completed jobs per unit time).
3. To achieve fairness.

Types of Schedulers

1. **Long-term/Job scheduler:** Job scheduler controls the degree of multiprogramming (number of processes in memory). It brings processes from new state to ready state.
2. **Short-term/CPU scheduler:** CPU scheduler decides which process should run on CPU from ready queue.
3. **Medium-term scheduler:** The key idea behind a medium term scheduler is that sometimes it is advantageous to remove processes from memory and thus reduce the degree of multiprogramming. Later, the process

Scanned by CamScanner

can be reintroduced into memory, and its execution can be continued where it is left-off. It brings processes from running state to suspend ready state or from wait state to suspend wait stage.

Context Switch

- Switching the CPU to another process requires performing a '**state save**' of the current process and a '**state restore**' of a different process. This take is known as a context switch.
- Context-switch is pure overhead, because system does no useful work while switching.

Dispatcher

- The dispatcher is the module that gives control of the CPU to the process selected by CPU scheduler.
- The function involves switching context, switching to user mode and jumping to the proper location in the user program to restart the program.
- The time taken by the dispatcher to stop one process and start another running, is known as the "Dispatch Latency".

Scheduling Criteria

- **CPU Utilization:** CPU should be as busy as possible
- **Throughput:** The number of processes that are completed per unit time
- **Turnaround Time:** Completion Time – Arrival Time
- Waiting Time = Turnaround Time – Burst Time
- Response Time = First Response Time – Arrival Time

Starvation

- Indefinite waiting of a process to get CPU cycles.

CPU Scheduling Algorithms

The following algorithms are priority/non-priority based and preemptive/non-preemptive based.

1. **First Come First Serve (FCFS)/ First-in First-out (FIFO):** It assigns the processor in the order of arrival of requests. This algorithm is non-preemptive. In this policy, the disadvantage is shorter jobs can get stacked behind the long running jobs.

   Convoy Effect: If the first process is having large burst time, then it will have major impact on average waiting time of other processes.
2. **Shortest Job First (SJF)/Shortest Job Next (SJN):** It favours shorter jobs, it is used as a benchmark in order to compare its performance

Scanned by CamScanner

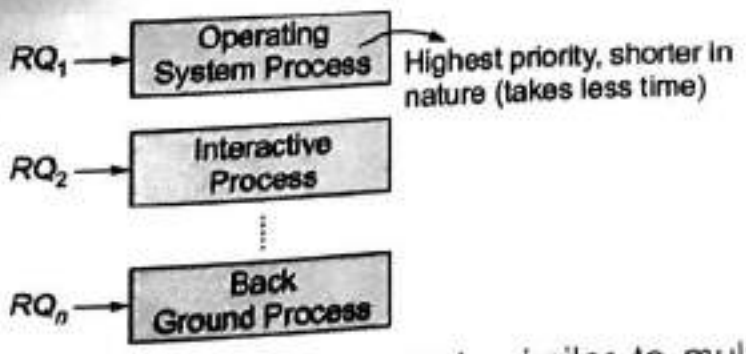
with other process. This algorithm is non-preemptive. In this policy, the advantage is "turnaround time can be minimised". The disadvantage is it can lead to starvation of processes.

"Aging": It is a technique of gradually increasing the priority of processes that wait in the system for a long time. It avoids starvation.

3. **Shortest Job Remaining Time First (SRTF) scheduling:** Preemptive SJF scheduling is called Shortest Remaining Time First (SRTF).
4. **Round Robin Scheduling:** Designed for *time sharing* system CPU is allocated to each process for certain time interval called time *quantum* (time slice). It is preemptive algorithm. If time quantum decreases, no of context switch increases, response time decreases.
5. **Longest Job First:** It is a non preemptive algorithm, completely opposite to SJF.
6. **Longest Remaining Time First:** LRTF is preemptive LJF.
7. **Priority based scheduling** can be preemptive or non preemptive.
8. **Highest Response Ratio Next (non preemptive):** Selects the highest **'response ratio'** process

$$\text{Response Ratio} = \frac{\text{Waiting time} + \text{Burst time}}{\text{Burst time}}$$

9. **Multilevel Queue Scheduling:** As only one ready queue is there so only one scheduling technique is used. Therefore, to use more than one scheduling can be used with multilevel queue scheduling.
   Each queue can maintain different priority.



10. **Multilevel Feedback Queues:** It is similar to multilevel queue scheduling.
    Multilevel feedback queues adjust each job's priority as follows:
    (*i*) Job starts in the highest priority queue.
    (*ii*) If time slice expires, drop the job by one level.
    (*iii*) If time slice does not expire, push the job up by one level.

Scanned by CamScanner

11. **Lottery Scheduling:** Lottery scheduling is another adaptive scheduling approach that addresses the fairness problem of multilevel feedback queues. The lottery scheduler gives every job some number of lottery tickets, and on each time slice, it randomly picks a winning ticket. On average, the CPU time allotted is proportional to the number of tickets given to each job.

Scheduling algorithm	Starvation possibility
FCFS	No
SJF	Yes
SRTF	Yes
Round Robin	No
LJF	Yes
LRTF	Yes
Non preemptive priority based	Yes
Preemptive priority based	Yes
Highest response ratio next	No
Multilevel queue	Yes
Multilevel feedback queue	No

■■■

Scanned by CamScanner

Memory Management and Virtual Memory

5

Memory Management

Dynamic Loading

- It refers to loading library into the memory during load or run time.
- A routine is not loaded until called and all routines are kept in "relocatable format".

Dynamic Linking

- It refers to the linking that is done during load or runtime and not when .exe is created.
- It requires support from operating system.
- Library which link dynamically is called Dynamic Link Library (DLL).

Static Linking

All modules of program are stored in memory before calling.

Memory Management Techniques

Memory Management Techniques

Contiguous (CG)

Program is stored at only one place.
This approach is centralized.

1. Multiprogramming with Fixed Task (MFT)
2. Multiprogramming with Variable Task (MVT)
3. Overlays
4. Buddy System (Dynamic Merging)

Non-contigous (NCG)

Program is not centralized, it is distributed in memory as per availability.

1. Paging
2. Segmentation
3. Segmented paging
4. Virtual memory (Demand paging)

Partitions

The following **functions** can be used for partitions:

1. Allocation
2. Protection (using limit registers)
3. Free space management
4. Deallocation

Scanned by CamScanner

The following **goals** should be achieved for partitions:

1. Minimum wastage of memory (fragmentation)
2. Run (manage) larger programs in smaller memory area.

Contiguous Memory Allocation

Two ways which support the above goals are overlays and virtual memory so that degree of multiprogramming will increase, there is more CPU utilization, throughput increases.

Overlays

- It is a very old memory management technique focus is on a single process information retained in RAM at a time.
- It will suit to sequential processing but multitasking, multiprogramming cannot be supported.
- Overlaying is possible when the program is divided into modules and the modules are independent.

Allocation Policies

1. **First Fit:** Allocation of first available free space.
2. **Best Fit:** Allocation of best available free space.
3. **Worst Fit:** Allocation of largest available free space.
4. **Next Fit:** Same as first fit, but it starts from last allocated process.

Multiprogramming with Fixed Task

- Maximum process size $\propto$ Maximum partition size.
- Degree of multiprogramming $\propto$ Number of partitions.
- Suffers from both Internal and External fragmentation.

Multiprogramming with Variable Tasks

- Degree of multiprogramming varies.
- Suffers from only External Fragmentation.
- Worst Fit is the best allocation policy.

Buddy System

- The "buddy system" allocates memory from a fixed size segment consisting of physically contiguous pages.
- Memory is allocated from this segment using a power-of-2 allocator, which satisfies requests in units sized as a power of 2.

Scanned by CamScanner

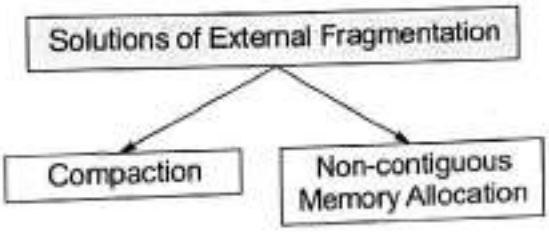
- An advantage of the buddy system is how quickly adjacent buddies can be combined to form larger segments using a technique known as "coalescing".

Internal Fragmentation

In fixed size partitioning if process size is smaller than partition size some unused free space is left within the partition and these are called memory fragments/holes/waste memory/unused free space and their sum is called internal fragmentation.

External Fragmentation

In variable size partitioning initially processes are contiguous but as processes leave their space these partitions may not suit the new processes for allocation. Those unused partitions is called External Fragmentation.



Compaction

- Processes are on one side and free space is on other side in the memory.
- During memory compaction:
  - (*i*) Block processes for compaction
  - (*ii*) Move them towards lower address space
  - (*iii*) Allocate memory for pending request
  - (*iv*) Resume blocked process w.r.t. compaction
- It consumes CPU time (overhead)
- Program need to support the dynamic allocation.

Non-contiguous Allocation

Size of Logical Address Space (LAS) generated by CPU is greater than or equal to size of Physical Address Space (PAS) generated by Main Memory. So that there will be efficient use of multiprogramming, can run larger programs in smaller memory area.

Scanned by CamScanner

Paging

Logical Address Space

- Logical Address Space is divided into equal size pages.
- Page size (PS) is generalisation of power of 2.
- Number of pages in LAS (N) = $\frac{LAS}{PS}$
- Page No (P) bits = $\lceil \log_2 N \rceil$
- Page offset depends on page size
- Page No dependents on number of pages
- Page offset (d) bits = $\lceil \log_2 PS \rceil$

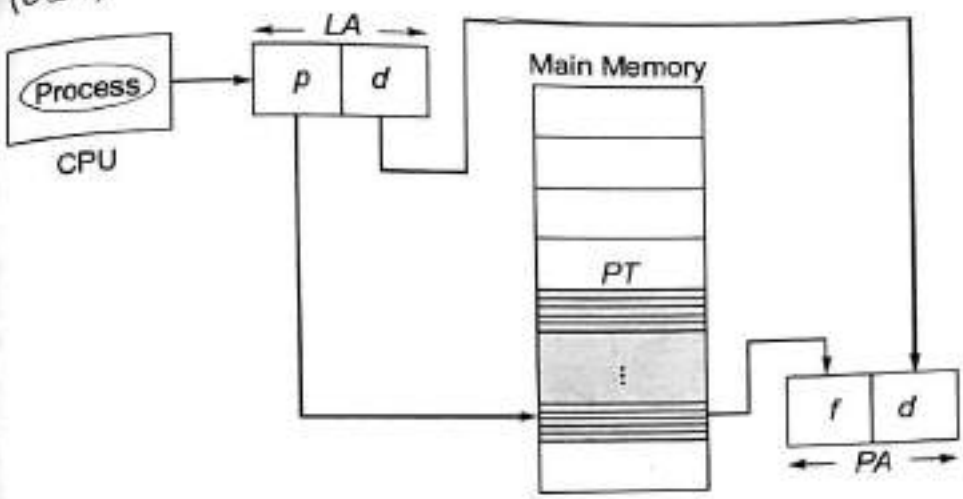
Physical Address Space

- Physical Address Space is divided into equal size frames (Page frames)
- Frame size = Page size
- Number of frames (M) = $\frac{PAS}{PS\,(FS)}$
- Frame No (f) bits = $\lceil \log_2 M \rceil$
- Frame offset = Page offset = d

Page Table

- Each process has its own pagetable stored in memory.
- Pagetable is organised as series of entries.
- Number of entries in page table = Number of pages in logical address space (single level paging)
- Page table entry contain frame number.
- Page table implementation maintains a four-bit per page table entry.
  - (*i*) **Use bit:** Set when a page is referenced, cleared by the clock algorithm
  - (*ii*) **Modified bit:** Set when page is modified, cleared when a page is written to disk
  - (*iii*) **Valid bit:** Set when a program can make legitimate use of this page table entry
  - (*iv*) **Read-only:** Set for a program to read the page, but not to modify it.

Scanned by CamScanner

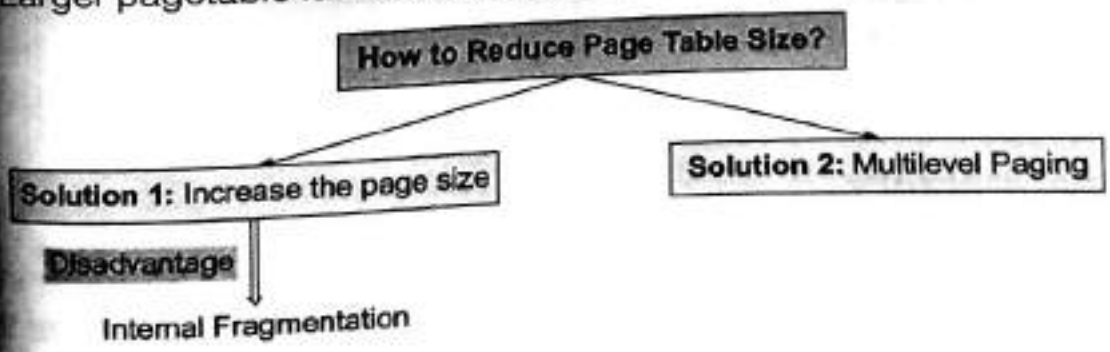
- Pagetable (PT) size in bytes = $N \times e$; e = page table entry size in bytes ($e \geq 1$)
- 

Performance of Paging

- **Paging without TLB:**
  (*i*) Main Memory Access Time = 'm'
  (*ii*) Effective Memory Access Time (EMAT) = '$2m$'
- **Paging with TLB:**
  (*i*) TLB Access Time = 'c' [$c <<< m$]
  (*ii*) TLB Hit Ratio(x) = $\frac{\text{Number of hits}}{\text{Total References}}$
  (*iii*) Effective Memory Access Time = $x(c + m) + (1 - x)(c + 2m)$

ltilevel Paging

Larger pagetable is not desirable because they take lot of memory.



imal Page Size

LAS = 'S' bytes
PT entry size = 'e' bytes
Page size (PS) = P

Scanned by CamScanner

- Page Table (PT) size = $\frac{S}{P} \times e$
- Internal Fragmentation (IF) = $P/2$
- Total overhead = PT size + IF = $\left(\frac{P}{2} + \frac{S}{P} \times e\right)$ = Should be minimum
- Differentiating w.r.t. to P

$$\frac{d}{dP}\left(\frac{P}{2} + \frac{S}{P} \times e\right) \Rightarrow \frac{1}{2} + \left(\frac{-1}{P^2} \times Se\right) = 0$$

$$\Rightarrow \frac{Se}{P^2} = \frac{1}{2}$$

$$\Rightarrow P = \sqrt{2\,Se}$$

- Performance of multilevel paging:

$$EMAT = x(c + m) + (1 - x)\,(c + (n+1)\,m)$$

where n is number of levels in paging, c is cache access time and m is memory access time.

Difference between Paging and Segmentation

Paging	Segmentation
(a) Paging suffers from Internal Fragmentation only in the last page.	(a) There is no Internal Fragmentation
(b) Paging does not preserve users view of memory allocation	(b) Segmentation is visible to the programmer
(c) There is no External Fragmentation	(c) There is External Fragmentation

Virtual Memory

- Virtual memory gives an illusion to the programmer that a huge amount of memory is available for waiting and executing programs greater than size of available physical address space.
- **Page Fault Service:**
  - (*i*) Process gets blocked
  - (*ii*) Change the mode bit

    Virtual Memory Manager (VMM) runs in Kernel mode

Scanned by CamScanner

(iii) VMM → Device Manger → Device Driver → Disk (Device)

(iv) Cases:

(a) Empty frame may be available

(b) No empty frame is available ⇒ Use page replacement algorithm

- More number of page faults, Page Fault Service Time (PFST) is more, execution time increases and throughput decreases.
- Virtual memory is implemented by demand paging.

Performance of Virtual Memory

Effective Memory Access Time (EMAT):

$$EMAT_{DP} = P \times s + (1 - P)m$$

Main Memory Access Time = 'm' [in nsec/μsec]

PFST = 's' [$s >>> m$] (in msec)

Page fault rate = 'P'

Page hit ratio = $1 - P$

Demand Paging

- It allows the pages that are being referenced actively to be stored in memory.
- It allows bigger virtual address spaces and provides the illusion of infinite physical memory.
- It allows more processes to fit in the physical memory, and to be running at the same time.
- Demand paging means that page tables need to sometimes point to the disk locations as opposed to memory locations.
- Page table needs an additional *present* or *valid* bit.

(i) If present, the page table entry points to a page in memory.

(ii) If not present, a reference to an invalid page ***(page fault)*** will cause the hardware to trap, and the OS performs the following steps while running another process in the meantime.

(a) Choose an old page in memory to replace

(b) If the old page has been modified, write contents back to disk

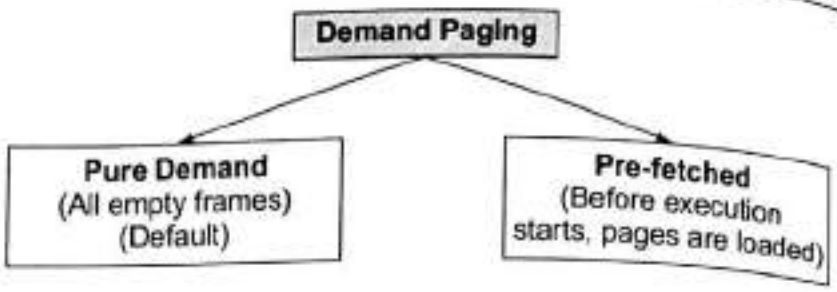
(c) Change the corresponding page table entry and TLB entry

(d) Load new page into memory from disk

(e) Update page table entry

(f) Continue the process

Scanned by CamScanner



Page Replacement

- **Page reference string:** Set of successively unique pages referred in given list of logical address or virtual address.
- **Frame Allocation Policies:** All frames are not allocated, less than maximum demand are allocated.

  Total number of frames = 'm'

  Total number of processes = 'n' $[P_1, P_2, \ldots, P_n]$

  Demand of each process = s_i

  Total demand of all processes = $S = \sum_{i=1}^{n} s_i$ $(s >>> m)$

Note:

- If process requires 'n' frames and operating system allocate 'n' frames to it at once, then there is no need of page replacement.

Page Replacement Algorithms

First-In First-Out (FIFO)

- In general within the increase in page frames the page fault rate of the process should decrease (natural characteristic).
- Sometimes with increase in number of page frames of the process, the page fault rate also increases. This unnatural behavior is called **Belady's Anamoly.**
- FIFO and FIFO based page replacement algorithms (Second Chance/ Clock Algorithm as it degenerates to FIFO) are bound to suffer from Belady's Anamoly.

Optimal

- Replace the page that will not be used for the longest period of time.
- It does not follow Beladys Anamoly.
- It is not implementable. It is used as a Benchmark to measure the performance of other page replacement algorithms.

Scanned by CamScanner

Least Recently Used (LRU)

- Replace the page that has not been used for the longest period of time.
- Performance of LRU is closer to optimal as it is practical approximation to optimal page replacement.
- Criteria is Time of Reference.

Most Recently Used (MRU)

- Replace the page that is used most recently.
- Criteria is Time of Reference.

Counting Algorithms

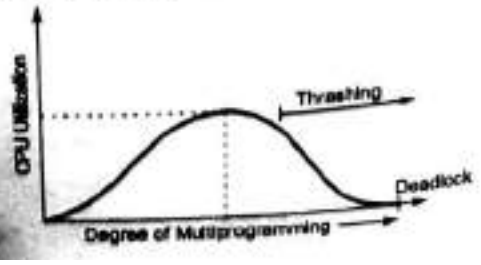
- Least Frequently Used (LFU): Replace the page with the smallest count.
- Most Frequently Used (MFU).
- Implementation is expensive.

Variation of LRU/LRU Approximation

- LRU approximation are not exactly LRU, they approximation to the behaviour of LRU (i.e. they work like LRU).
- Reference bit (R)
  - 0 → Page is not referred so far during the current epoch
  - 1 → Page is referred atleast once during the current epoch (a period of time)
- **Second Chance/Clock Algorithm:** It degenerates to FIFO. Criteria is Arrival Time and Reference Bit.
- **Enhanced Second Chance/Not Recently Used:** Criteria is Reference Bit and Modify/Dirty Bit
  - Modify Bit
    - 0 → Page is clean
    - 1 → Page is modified

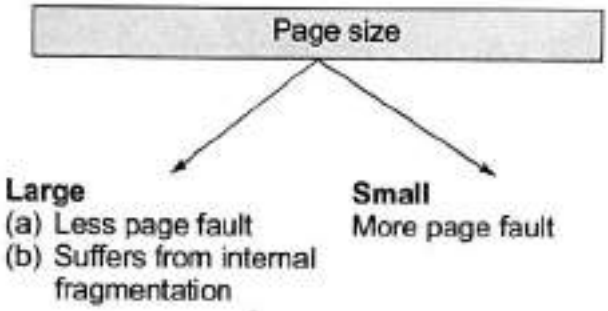
Thrashing

- High paging activity is called thrashing. A process is thrashing if it is spending more time paging than executing.



Scanned by CamScanner

- **Thrashing** occurs when the memory is over committed, and pages are tossed out while still needed.
- CPU utilization is very low when the page fault rate is high.
- Reasons:
  - (*i*) Primary Reasons:
    - (a) Lack of memory frames
    - (b) High degree of multiprogramming
  - (*ii*) Secondary Reasons:
    - (a) Page replacement policy
    - (b)



- Control strategies:
  - (*i*) **Prevention:** Controlling the degree of multiprogramming.
  - (*ii*) Detection:
    - (a) Low CPU utilization
    - (b) High degree of multiprogramming
    - (c) High paging disk utilization
  - (*iii*) Recovery:
    - (a) Process suspension
    - (b) Increase RAM

Inverted PageTable

- An inverted page table has one entry for each real page (or frame) of memory.
- Each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- Only one page table is in the system and it has only one entry for each page of physical memory.
- Each inverted page table entry is a pair < process-id, page-number> where the process id assumes the role of the address space identifier.

■■■

Scanned by CamScanner

I/O and File Systems, protection and Security

6

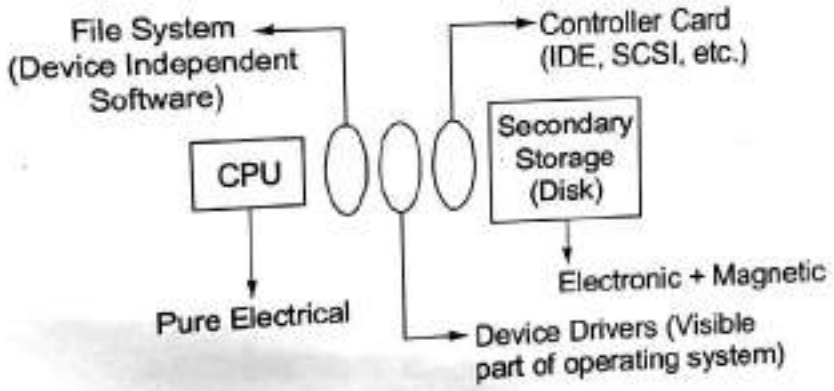
File System

There are three ways to access a file.

1. **Sequential access:** Bytes are accessed in order.
2. **Random access:** Bytes are accessed in any order.

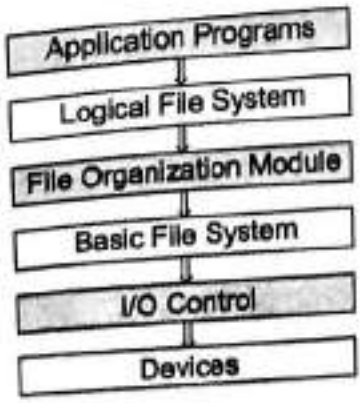
A file system consists of the following major components:

- **Disk management** organises disk blocks into files.
- **Naming** provides file names and directories to users, instead of track and sector numbers.
- **Protection** keeps information secure from other users
- **Reliability** protects information loss due to system crashes.



File System Structure

- A file system is used to allow the data to be stored, located, and retrieved easily.
- The file system itself is generally composed of many different levels:



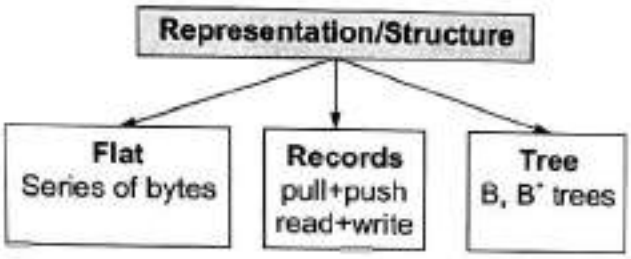
Scanned by CamScanner

File Usage Patterns

- Most files are small and most file references are to small files.
- Large files use up most of the disk space.
- Large files account for most of the bytes transferred between memory and disk.

File

- A file is logically related records. From programmers perspective file is a DATA STRUCTURE that has definition, representation/structure, operations and attributes.



- **Operations:** Create, Open, Read, Write, Modify, Seek, Copy, Move, Rename, Close, Delete.
- **Attributes:** Name, Type, Size, Owner, Path, Location etc.
- **Directory:** It is also a file which contains information about files. It is a metadata of files.

Logical Structure of Disk

- **Partitions:**
  - (*i*) **Primary/Bootable:** In which operating system and data can be stored. Atleast one primary partition should be there in disk.
  - (*ii*) **Secondary/Non Bootable:** extended (1 or more logical drives)
- **File Control Block (FCB):** It contains information about the file, including ownership, permission, location etc.
  - (*i*) **UNIX:** UNIX FILE SYSTEM (Inode block)
  - (*ii*) **Windows:** FAT, FAT32,WTFS (information within master file table)
  - (*iii*) **Linux:** Extended File System (ext2 and ext3)
- **Allocation Methods:**
  - (*i*) **Contiguous Allocation:**
    - (a) There is internal and external fragmentation
    - (b) Inflexible to increase file size
    - (c) Random access (Faster as it is direct access).

Scanned by CamScanner

(*ii*) **Linked Allocation:**

(a) Each file block on disk is associated with a pointer to the next block.

(b) Flexibility to increase file size

(c) Sequential access so it is slow

(d) Space is consumed for links

(e) Suffers from external fragmentation

(f) MS-DOS file system, which uses the File Attribute Table (FAT) to implement linked -list files

(*iii*) **Indexed Allocation:**

(a) Uses an index to directly track the file block locations.

(b) No fragmentation

(c) Flexibility to increase file size

(d) Suffers from wasted space, supports direct access.

(*iv*) **Multilevel Indexed Allocation:**

(a) Index entries point to index blocks as opposed to data blocks.

(b) The file header, (*i*_node data structure), holds index pointers.

(c) The first pointers point to data blocks.

(d) The pointer points to a **single indirect block,** which contains additional pointers to data blocks.

(d) The pointer in the file header points to a **double indirect block,** which contains pointers to single indirect blocks.

(e) The pointer points to a **triple indirect block,** which contains pointers to double indirect blocks.

- **Free Space Management:**

(*i*) **BitMap/Bit vector:**

(a) A bit vector of size equal to the number of disk blocks is used.

(b) If i^{th} block is free then bit vector value at i^{th} location will be 0 otherwise 1.

(c) Memory requirements for storing bit vector also increases with the increase in the disk size.

(d) Advantages of this approach include: Relatively simple and efficient to find the first free blocks or n-consecutive free blocks on the disk.

(e) To find the fist non-zero value, the calculation of the block number is:

Block Number = (Number of bits per word × Number of zero-value words) + offset of first 1 bit

Scanned by CamScanner

Consider a disk with 'B' blocks, 'F' free, 'D' – Disk Block Address (DBA), 'S' – Disk Block size in Bytes (DBS).

$$\left.\begin{array}{l}\text{Actual Disk size} = BS \\ \text{Maximum possible Disk size} = 2^D \times S\end{array}\right\} \therefore \boxed{B \le 2^D}$$

Condition in which free list uses less space than BitMap.

BitMap → 'B' bits

Free List → $F \times D$ bits

$$\boxed{FD < B}$$

(*ii*) **Linked list approach:**

(a) It is to link together all the free disk blocks.

(b) It is not easy to traverse the free list.

(*iii*) **Grouping:**

(a) The addresses of n-free blocks are stored in the fist free block.

(b) The first $(n-1)$ of these blocks are actually free.

(c) The last block contains addresses of another n-blocks.

(d) The importance of this approach is that the addresses of a large number of free blocks can be found quickly.

(*iv*) **Counting:** We usually allocate and free a number of contiguous blocks at the same time.

(a) Rather than have the free-space list contain just block numbers.

(b) The free-space list can contain paris (block number, count)

(c) Although each entry is larger, the list, in general, will take less space.

Disk Scheduling → we access cylinder no.

First-come First-serve (FCS)

- Requests are served in the order of arrival.
- This policy is fair among requesters, but requests may land on random spots on disk.
- The seek time may be long.
- It does not provide the fastest service.

Shortest Seek Time First (SSTF)

- It picks the request that is closest to the current disk arm position.
- Selects the request with minimum seek time from the current head position

Scanned by CamScanner

- It includes the rotational delay in calculation, since rotation can be as long as seek.
- SSTF is good at reducing seeks, but may result in starvation.

SCAN (Elevator)

- It takes the closet request in the direction of travel.
- The head continuously scans back and forth across the disk.
- It guarantees no starvation, but retains the flavor of SSTF.

LOOK

- Read write head exactly stops at the last head i.e., look at last readwrite (r/w) operation and go back.

C-SCAN

- The disk arm always serves requests by scanning in one direction.
- Once the arm finishes scanning for one direction, it quickly returns to the 0th track for the next round of scanning.
- Head is imagined to move in circular motion (which is not practically possible).

Disk Access Time

The disk access time to read or write a disk section includes three components.

- **Seek time:** The time to position heads over a cylinder.
- **Rotational delay:** The time to wait for the target sector to rotate underneath the head.
- **Transfer time:** The time to transfer bytes.
- Disk access time = Seek time + Rotational delay + Transfer time.

Protection and Security

- **Protection** refers to the actual mechanisms implemented to enforce the specified policy.
- **Security** refers to the policy of authorising accesses. Security aims to prevent intentional misuses of a system, while protection aims to prevent either accidental or intentional misuses.
- A secure system tries to accomplish three goals:
  - (*i*) **Data confidentiality:** secret data remains secret.
  - (*ii*) **Data integrity:** unauthorised users should not be able to modify any data without the owner's permission.
  - (*iii*) **System availability:** nobody can disturb the system to make it unusable.

Scanned by CamScanner

- There are three components of security:
  (*i*) **Authentication** determines who the user is.
  (*ii*) **Authorization** determines who is allowed to do what.
  (*iii*) **Enforcement** makes sure that people do only what they are supposed to do.

Security Attacks

Eavesdropping

- **Eavesdropping** is the listener approach.
- One can tap into the serial line on the Ethernet, and see everything typed in; almost everything goes over network unencrypted.

Abuse of Privilege

If the superuser is evil, there is nothing you can do.

Imposter

- An imposter breaks into the system by pretending to be someone else.
- A countermeasure against the imposter attack is to use **behavioral monitoring** to look for suspicious activates.

Trojan Horse

- A **Trojan horse** is a seemingly innocent program that contains code that will perform an unexpected and undesirable function.
- A countermeasure against the Trojan horse is **integrity checking**.

Salami Attack

- The idea is to build up a chunk one-bit at a time.
- A countermeasure is for companies to have ***code* reviews** as a standard practice.

Logic Bombs

- A programmer may secretly insert a piece of code into the production system.
- A countermeasure is to have **code reviews.**

Denial-of-service Attack

- Denial-of-service attacks refer to attacks on system availability.
- A handful of compromised machines can flood a victim machine with network packets to disrupt its normal use.

■■■

Scanned by CamScanner

A Handbook on Computer Science

9

Databases

CONTENTS

1. ER-model and Relational Model 274
2. Database Design 279
3. SQL 286
4. File Structures (B and B* Trees) 293
5. Transactions and Concurrency Control 296

Scanned by CamScanner

ER-model and Relational Model

1

ER Model

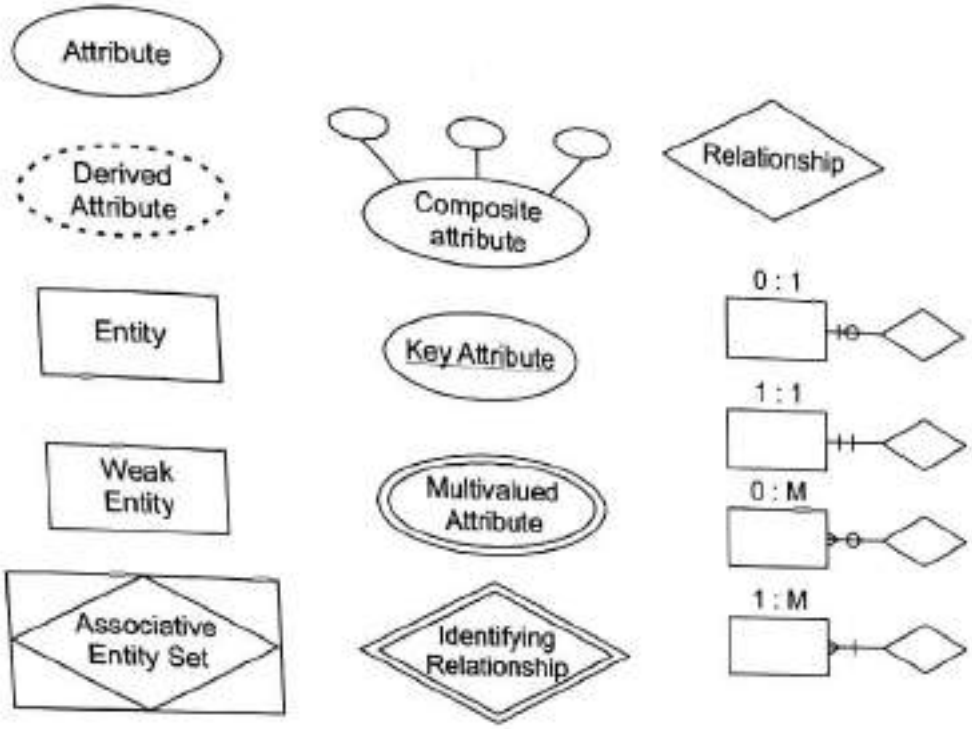
- It is a logical tool which is used for database scheme design.
- It does not include implementation details.
- It is described by an ER (Entity-Relationship) diagram.
- **Entity:** An entity is a thing that has an independent existence. It is described by its attributes and determined by its instantiations (are particular values for its attributes).
- **Entity Set:** It is a set of entities of the same type, denote by a rectangular box in ER diagram. Entity can be identified by a list of attributes which are placed in ovals.
- **Relationship:** It is an association among several entities.
- **Relationship Set:** It is a set of relationship of the same type.
- **Superkey:** A superkey is a set of one or more attributes which collectively allow us to identify uniquely an entity in the entity set.
- **Candidate key:** A superkey for which no subset is a super key is called a candidate key.
- **Primary key:** Primary key is a candidate key chosen by the DB designer to identify entities in an entity set.
- **Weak entity set:** An entity set that does not possess sufficient attributes to form a primary key is called a weak entity set. Weak entity set is an entity set whose existence is dependent on one or more other strong entity sets.
- **Strong entity set:** An entity set that does have a primary key is called a strong entity set.

Structural Constraints

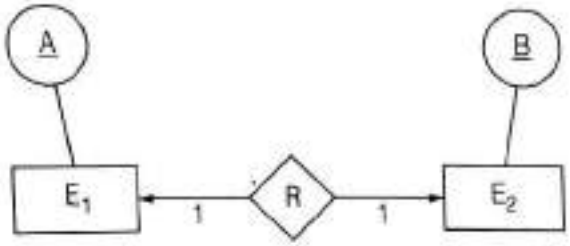
- **Degree:** Number of participating entity sets.
- **Cardinality constraints:** One-to-one, one-to-Many and Many-to-Many
- **Participation constraints:** Partial or total.

Scanned by CamScanner

ER Diagram Symbols

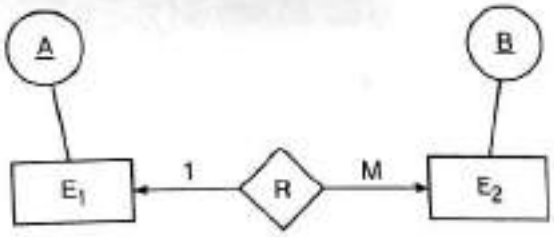


One-to-one



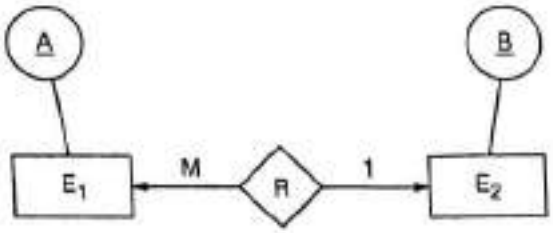
Candidate key of R : A, B
One-to-one

One-to-many



Candidate key of R : B
One-to-many

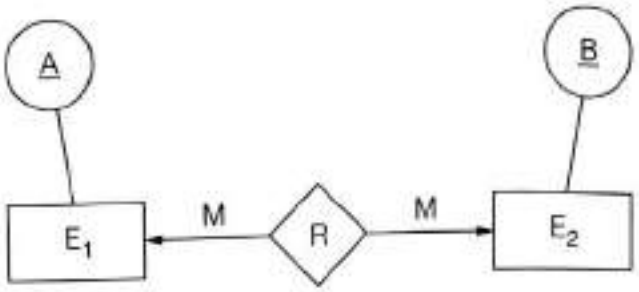
Many-to-one



Candidate key of R : A
Many-to-one

Scanned by CamScanner

Many-to-many



Candidate key of R : AB

Many-to-many

- If we have cardinality one to many or many to one then, we can mix relational table with many involved table.
- If cardinality is many to many we cant mix any two tables.
- If we have one to one relation and we have total participation of one entity then we can mix that entity with relation table and if we have total participation of both entity then we can make one table by mixing two entities and their relation.

Commands

DDL Commands

CREATE, ALTER, DROP, GRANT, REVOKE, TRUNCATE AND COMMENT.

DOL Commands

COMMIT, SAVEPOINT, ROLLBACK, and SET TRANSACTION.

DML Commands

SELECT, INSERT, UPDATE, DELETE, CALL, LOCK TABLE AND EXPLAIN PLAN.

Relational Commands

Cartesian Product (×)

- Deg $(R \times S) = \deg(R) + \deg(s)$ and
- $|(R \times S)| = |R| \times |S|$

Natural Join ($\bowtie$)

- It is same as cross product + $\sigma_{\text{equality of common attribute}}$
- $\deg(R_1 \bowtie R_2) = \deg(R_1) + \deg(R_2)$ – number of common attribute.
- Number of n-ary relation that can be formed over n-domains having n elements: n^n

Scanned by CamScanner

- Number of equivalent representation of a relation having degree m and cardinality n are m! × n!

Orderby

Default ascending order.

Aggregation Operations

SUM, COUNT, AVG, MIN, MAX

Selection Operation (Restriction) (σ)

- $\deg(\sigma_c(R)) = \deg(R)$ (Number of attributes)
- $0 \le |\sigma_c(R)| \le |R|$ (Number of tuples)
- $\sigma_{c_1}(\sigma_{c_2}(R)) = \sigma_{c_2}(\sigma_{c_1}(R)) = \sigma_{c_1,c_2}(R)$

Projection Operation (π)

- $0 \le \deg(\pi_A(R)) \le \deg(R)$
- $|\pi_{A_i}(R)| = |R|$

Rename (ρ)

$\text{Deg}(\rho_s(R)) = \text{degree}(R)$ and $|(\rho_s(R))| = |R|$

Union Operation

- Two relations are union compatible if they are of same degree and same domain.
- $\deg(R \cup S) = \deg(R) = \deg(S)$
- $\max(|R|, |S|) \le |R \cup S| < (|R| + |S|)$

Intersection Operation

- Two relation should be union compatible.
- $\deg(R \cap S) = \deg(R) = \deg(S)$
- $0 \le |(R \cap S)| \le \min(|R|, |S|)$

Set Difference (–)

- Two relation should be union compatible
- $\deg(R-S) = \deg(R) = \deg(S)$
- $0 \le |(R - S)| \le |R|$

Scanned by CamScanner

Operation	Relational Algebra	Tuple Relational Calculas
Selection	σ_{cond} (R)	{T | R(T) AND Cond (T)}
Projection	$\Pi_{A_1, A_2, \ldots, A_k}$ (R)	{T | T.A_1, T.A_2,...,T.A_k | R(T)}
Cartesian Product	R × S	{T | $\exists T_1 \in R$, $\exists T_2 \in R$(T.A_1=T_1.A_1 AND,...,T.A_n=T_1.A_n AND T.B_1=T_2.B_1 AND,...,T.B_m=T_2.B_m)}
Union	R ∪ S	{T | R(T) AND S(T)}
Set Difference	R – S	{T | R(T) AND $\forall T_1 \in S$, (T_1<>T)} where T<>T_1 is T.A_1<>T_1A_1OR...T.A_n<>T_1.A_n

■■■

Scanned by CamScanner

Database Design

2

Database Design

Limitations of File System

1. To access database user should know about the physical details of the data.
2. Operating system fails to control concurrency when database is large.
3. Once got access user can access the whole data of the file system.

Database Design Goals

1. 0% redundancy
2. Loss-less join
3. Dependency preservation.

According to Codd

No two records of the table should be equal.

Difference between Primary Key and Alternative Key

1. At most one primary key is allowed for any relation but more than one alternative keys are allowed.
2. Null values are not allowed in primary key. Null values are allowed in alternative keys.
3. Primary keys used to design primary key, unique keys are used to design alternative keys.

Primary key: Unique + not null

Super key: Set of attributes used to differentiate all the tuples of the relation or super set of candidate key.

Example: Let R be the relational schema with n-attributes, $R(A_1, A_2 \ldots A_n)$.

Number of super keys possible

(a) With only candidate key $A_1 \rightarrow 2^{n-1}$

(b) With only candidate key $A_1, A_2 \rightarrow 2^{n-1} - 2^{n-2} + 2^{n-1}$

(c) With only candidate key $A_1A_2, A_3A_4 \rightarrow 2^{n-2} - 2^{n-4} + 2^{n-2}$

Scanned by CamScanner

Functional Dependency

Let R be the relational schema and x, y be the non-empty set of attributes. t_1, t_2 are any tuples of relation then $x \rightarrow y$ (y functionally determined by x) if $t_1.x = t_2.x$ then $t_1.y = t_2.y$

Trivial Functional Dependency

If $x \supseteq y$ then $x \rightarrow y$ is trivial dependency.

Example:

- Sid → Sid
- Sid Same → Sid

Non-trivial Functional Dependency

If $x \cap y = \phi$ and if $x \rightarrow y$ satisfy functional dependency definition.

Example:

Cid → Cname.

Armstrong's Axioms

1. Reflexive: if $x \supseteq y$ then $x \rightarrow y$
2. Transitivity: If $x \rightarrow y$ and $y \rightarrow z$ then $x \rightarrow z$
3. Argumentation: If $x \rightarrow y$ then $xz \rightarrow yz$
4. Splitting: If $x \rightarrow yz$ then $x \rightarrow y$, $x \rightarrow z$
5. Union: If $x \rightarrow y$ and $x \rightarrow z$ then $x \rightarrow yz$
6. Pseudo transitivity: If $x \rightarrow y$, $yw \rightarrow z$ then $xw \rightarrow z$

Attribute Closure (x^+)

Set of all attributes functionally determined by X.

Example:

R(ABCD) $\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

$(A)^+ = \{ABCD\}$

Functional Dependency Set closure (F^+)

Set of all trivial and non-trivial FD's derived from given FD set F.

$x \rightarrow y$ is logically applied in given FD set F only if x^+ should determine y.

Let R be the relational schema decomposed into R_1 and R_2. Given decomposition is loss-less only if

1. $R_1 \cup R_2 = R$

Scanned by CamScanner

2. $R_1 \cap R_2 \neq \phi$
3. $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$.

Example:

R(ABCDE)

$\{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$

$D = \{ABC, ACDE\} \rightarrow$ Loss-less.

Properties of Decomposition

Loss-less Join Decomposition

Because of decomposition there should not be generation of any additional tuples.

$$\text{i.e. } R \equiv R_1 \bowtie R_2$$

Dependency Preserving Decomposition

Because of decomposition there should not be any loss of any dependency. Let R be the relational schema with functional dependency set F is decomposed into $R_1, R_2, \ldots R_n$ with FD sets $F_1, F_2, \ldots F_n$ respectively. In general $F_1 \cup F_2 \cup F_3 \ldots F_n$ can be $\subseteq F$.

If $F_1 \cup F_2 \ldots \cup F_n \equiv F$ then decomposition is preserving dependency.

NORMALIZATION

The normalization is especially meant to eliminate the following anomalies:

- Insertion anomaly
- Deletion anomaly
- Update anomaly
- Join anomaly

Goals of Normalisation

- Integrity
- Maintainability

Side Effects of Normalization

- Reduced storage space required (usually, but it could increase)
- Simpler queries (sometimes, but some could be more complex)
- Simpler updates (sometimes, but some could be more complex)

Scanned by CamScanner

Example: F = {A → BC}, here F = {A → B, A → C} is simple.

- It is left reduced (removal of extraneous symbols)

  Example: F = {AB → C, B → C}, here B is extraneous attribute. So F = {A → C, B → C}.

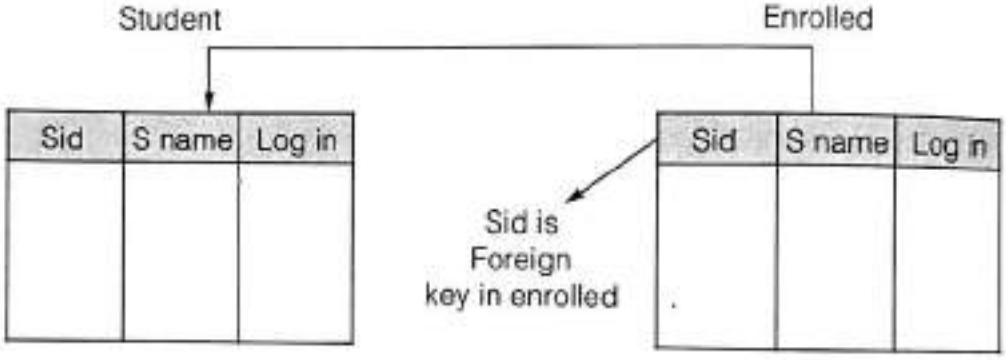
- It is non-redundant (eliminating unnecessary FDs)

  Example: F = {A → B, B → C, A → C}, here A → C is redundant. Hence, F = {A → B, B → C}.

Referential Integrity Constraints

Foreign Key

Foreign Key is a set of attributes that references primary key or alternative key of the same relation or other relation.



Referenced Relation

1. **Insertion:** no violation.
2. **Deletion:**
   (a) **On delete no action:** Means if it causes problem on delete then not allowed to delete.
   (b) **On delete cascade:** If we want to delete primary key value from referenced table then it will delete that value from referencing table also.
   (c) **On delete set null:** If we want to delete primary key from referenced table then it will try to set the null values in place of that value in referencing table.
3. **Updation:**
   (a) On update no action
   (b) On update cascade
   (c) On update set null

Scanned by CamScanner

Referencing Relation

- Insertion: May cause violation
- Deletion: No violation
- Updation: May cause violation

Example:

A	C
2	4
3	4
4	3
5	4
6	2

C is foreign key referencing A
Delete (2, 4) and on delete cascade

A	C
3	4
4	3
5	4

← Result

■■■

Scanned by CamScanner

SQL

3

Introduction

- SQL stands for structured query language.
- SQL lets you access and manipulate databases.
- SQL is an ANSI (American National Standards Institute) standard.

Functions of SQL

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures and views.

RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS is the bases for SQL, and for all modern database systems such as MS SQL server, IBM DB2, Oracle, MySQL and Microsoft Access.
- The data in RDBMS is stored in database objects called tables.
- A table is a collection of related data entries and it consists of columns and rows.

Type of SQL

Two types of SQL:

- **DML:** Data Manipulation Language (SELECT)
- **DDL:** Data Definition Language (CREATE TABLE)

Scanned by CamScanner

SQL Commands

- **SELECT Statement:**
  - (*i*) SELECT statement defines WHAT is to be returned (separated by commas)
  - (*ii*) "*" mean all columns from all tables in the FROM statement.

    Example: SELECT state-code, state-name.
- **FROM Statement:**
  - (*i*) Defines the Table(s) or View(s) used by the SELECT or WHERE statements.
  - (*ii*) Multiple Tables/Views are separated by commas.
- **WHERE Clause:**
  - (*i*) Defines what records are to be included in the query, it is optional.
  - (*ii*) Uses conditional operators:
    - (a) =, >, >=, <=, !=(<>)
    - (b) BETWEEN x AND y
    - (c) IN (list)
    - (d) LIKE '% string' ("%" is a wild-card)
    - (e) IS NULL
    - (f) NOT (BETWEEN/IN/LIKE/NULL)
- **SELECT DISTINCT Statement:**
  - (*i*) The SELECT DISTINCT statement is used to return only distinct (different) values.
  - (*ii*) In a table, a column may contain many duplicate values; and sometimes you only want to list the different (distinct) values.
  - (*iii*) **Syntax:** SELECT DISTINCT column_name, column_name FROM table_name;
- **AND & OR:**
  - (*i*) "AND" means all conditions are TRUE for the record
  - (*ii*) "OR" means at least one of the conditions is TRUE.
  - (*iii*) Multiple WHERE conditions are linked by AND/OR statements.
- **ORDER BY statement:**
  - (*i*) Defines how the records are to be sorted
  - (*ii*) Must be in the SELECT statement to be ORDER BY
  - (*iii*) Default is to order in ASC (Ascending) order. To sort the records in a descending order, use the DESC keyword.

Scanned by CamScanner

(*iv*) **Syntax:** SELECT column_name, column_name FROM table_name ORDER BY column_name, column ASC/DESC;

- **Group Functions:**
  (*i*) Performs common mathematical operations on a group of records.
  (*ii*) Must define what constitutes a group by using the GROUP BY clause.
  (*iii*) All non-group elements in the SELECT statement must be in the GROUP BY clause.
- **INSERT INTO statement:**
  (*i*) Used to insert new records in a table.
  (*ii*) **Syntax:** INSERT INTO table_name VALUES (value1, value2,...);

  or

  INSERT INTO table_name (column1, column2,...) VALUES (value1, value2,...)
- **UPDATE statement:**
  (*i*) Used to update records in a table
  (*ii*) **Syntax:** UPDATE table_name SET column1=value1, column2=value2,... WHERE some_column = some_value

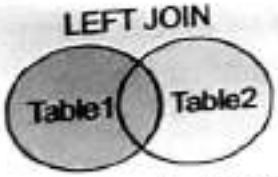
Note:

- The WHERE clause specifies which record or records that should be updated. IF WHERE clause is omitted, all records will be updated.

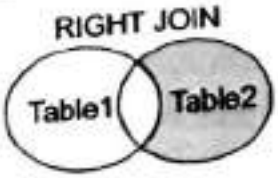
- **DELETE statement:**
  (*i*) Used to delete rows in a table.
  (*ii*) **Syntax:** DELETE FROM table_name WHERE some_column = some_value;
- **LIKE operator:**
  (*i*) The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
  (*ii*) **Syntax:** SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern;
- **IN operator:**
  (*i*) The IN operator allows you to specify multiple values in a WHERE clause.
  (*ii*) **Syntax:** SELECT column_name(s) FROM table_name WHERE column_name IN (value1, value2, ...)

Scanned by CamScanner

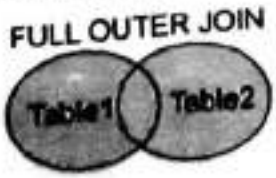
- **BETWEEN operator:**
  (i) It selects values within a range. The values can be numbers, text, or dates.
  (ii) **Syntax:** SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 and value2;
- **Aliases:**
  (i) SQL aliases are used to temporarily rename a table or a column heading.
  (ii) SQL aliases are used to give a database table, or a column in a table, a temporary name.
  (iii) Basically aliases are created to make column names more readable.
  (iv) **Syntax:** SELECT column_name AS alias_name FROM table_name AS alias_name;
- **Joining Tables:**
  (i) Joins between tables are usually based on primary/foreign keys.
  (ii) Make sure joins between all tables in the from clause exist.
  (iii) List joins between tables before other selection elements.
  (a) **Left outer join keyword:** It returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.



  (b) **Right outer join keyword:** It returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.



  (c) **Full outer join keyword:** It returns all rows from the left table (table1) and from the right table (table2). It combines the result of both LEFT and RIGHT joins.



Scanned by CamScanner

- **SQL constraints:**
  - (*i*) SQL constraints are used to specify rules for the data in a table.
  - (*ii*) If there is any violation between the constraint and the data action, the action is aborted by the constraint.
  - (*iii*) Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).
  - (*iv*) In SQL, we have the following constraints:
    - **(a) NOT NULL:** Indicates that a column cannot store NULL value. It enforces a field to always contain a value. This means that one cannot insert a new record, or update a record insert a new record, or update a record without adding a value to this field.
    - **(b) UNIQUE:**
      - Ensures that each row for a column must have unique value.
      - The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.
      - A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

Note:

There might be many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

- **(c) PRIMARY KEY:** A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table easily and quickly.
- **(d) FOREIGN KEY:** Ensure the referential integrity of the data in one table to match values in another table.
- **(e) CHECK:** Ensures that the value in a column meets a specific condition. It is used to limit the value range that can placed in a column.
- **(f) DEFAULT:** Specifies a default value when specified none for this column.

- **ALTER TABLE statement:**
  - (*i*) It is used to add, delete, or modify columns in an existing table.

Scanned by CamScanner

(*ii*) Syntax:

(a) To add a column in a table:

ALTER TABLE table_name
ADD column_name datatype

(b) To delete a column from a table:

ALTER TABLE table_name
DROP COLUMN column_name

- **Aggregate Functions:**

(*i*) Aggregate functions perform a calculation on a set of values and return a single value.

(*ii*) Aggregate functions return the same value any time that they are called by using a specific set of input values.

(*iii*) Aggregate functions can be used as expressions only in the following:

(a) The select list of a SELECT (either a subquery or an outer query).

(b) A HAVING clause.

- **AVG( ) Function:**

The AVG( ) function returns the average value of a numeric column.

Syntax: SELECT AVG (column_name) FROM table_name

- **COUNT( ) Function:**

The COUNT( ) function returns the number of rows that matches a specified criteria.

COUNT (column_name) syntax:

It returns the number of values (NULL values will not be counted) of the specified column:

SELECT COUNT (column_name) FROM table_name;

COUNT (*) Syntax:

It returns the number of records in a table:

SELECT COUNT (*) FROM table_name;

COUNT (DISTINCT column_name) Syntax:

It returns the number of distinct values of the specified column:

SELECT COUNT (DISTINCT column_name) FROM table_name;

Scanned by CamScanner

- **MAX( ) Function:**

  The MAX( ) function returns the largest value of the selected column.

  Syntax: SELECT MAX (column_name) FROM table_name;

- **MIN( ) Function:**

  The MIN( ) function returns the smallest value of the selected column.

  Syntax: SELECT MIN (column_name) FROM table_name;

- **SUM( ) Function:**

  The SUM( ) function returns the total sum of a numeric column.

  Syntax: SELECT SUM (column_name) FROM table_name;

- **LEN( ) Function:**

  The LEN( ) function returns the length of the value in a text field.

  Syntax: SELECT LEN (column_name) FROM table_name;

- **HAVING Clause:** The HAVING clause was added to SQL because the WHERE keyword could be used with aggregate functions.

- **GROUP BY Statement:** The GROUP BY statement is used in conjunction with the aggregate functions to group the result_set by one or more columns.

■■■

Scanned by CamScanner

File Structures (B and B+ Trees)

4

Sequential File

- Blocking factor : $\left\lfloor \frac{\text{Block size}}{\text{Record size}} \right\rfloor$
- Number of record blocks : $\left\lceil \frac{\text{Total no. of records}}{\text{Blocking factor}} \right\rceil$
- Average number of blocks accessed by linear search:

$$\left\lceil \frac{\text{\# of record blocks}}{2} \right\rceil$$

- Average number of blocks accessed by binary search:

$$\lceil \log_2 \text{ \# of record blocks} \rceil$$

Index File

- Index blocking factor = $\left\lceil \frac{\text{\# of record blocks} + 1}{2} \right\rceil$
- First (single) level index blocks: $\left\lceil \frac{\text{\# of record blocks}}{\text{index blocking factor}} \right\rceil$
- Number of block accesses = $\lceil \log_2 \text{ (first level index blocks)} \rceil + 1$

Indexing Types

1. **Single level Index:**
   - (a) **primary index (sparse):** Ordered with key field.
   - (b) **clustered index (sparse):** Ordered with non key field.
   - (c) **secondary index (dense):** Ordered with either key or non key field.
2. **Multilevel Index:**
   - (a) Indexed sequential access method
   - (b) B-tree index
   - (c) B+-tree index

Scanned by CamScanner

Single Level Index

Dense Index

- For every data base record there exists entry in the index file.
- Number of data base records = Number of entries of the index file.

Sparse Index

- For set of database records there exists single entry in the index files.
- Number of index file entries < Number of database records
- Number of index file entries = Number of database blocks.

Clustered Index

- Search key should be used to physically order the DB.
- Search key is non-key.
- Atmost one clustering index is possible.
- **Single level index blocks:** $\left\lceil \frac{\text{No. of distinct values over non key field}}{\text{Index blocking factor}} \right\rceil + 1$
- **Number of block accesses:** $\lceil \log_2 (\text{single level index blocks}) \rceil + 1$

Secondary Index

- Search key is used to non ordered the DB file.
- Search key is primary or alternative key.
- Dense indexing is used.
- Number of block accesses $\lceil \log_2 (\text{single level index block}) \rceil + 1$
- Index blocking factor same for all indexes.

Multilevel Index (Static level index)

- Second level index is always sparse.
- Level 1 = "first level index blocks" computed by index.

$$\text{Level 2} = \left\lceil \frac{\text{\# of blocks in level (1)}}{\text{index blocking factor}} \right\rceil$$

$$\vdots$$

$$\text{Level n} = \left\lceil \frac{\text{\# of blocks in level (n} - 1)}{\text{index blocking factor}} \right\rceil = 1$$

Scanned by CamScanner

- Number of levels = n
- Number of blocks = $\sum_{i=1}^{n}$(Number of blocks in level i)
- Number of blocks access = n + 1

B-tree (Bayes's/Balanced Search Tree)

- **Root node:** 2 to n children (pointers)
- **Internal node:** $\lceil n/2 \rceil$ to n children
- leaf nodes are all at same level.
- **Block Size** = p × (Size of block pointer) + (p − 1) × (Size of key field + size of record pointer)
- Minimum number of nodes = $1+\left(\frac{2\left[(p/2)^h-1\right]}{(p/2)-2}\right)$
- Maximum number of nodes = $\frac{p^{h+1}-1}{p-1}$
- Minimum possible height = $\lceil \log_p \ell \rceil$ (ℓ: no. of leaves)
- Maximum possible height = $\lfloor 1+\log_{p/2} \ell/2 \rfloor$

B+ Tree

It is same as B-tree.
All the records are available at leaf (last) level.
It allows both random and sequential access, but in B-tree only random access allowed.
All the leaf nodes are connected to next leaf node by block pointers [every leaf node has one block pointer]
Order of non-leaf Node:
[p × size of block pointer] + [(p − 1) × size of key field] ≤ Block size.
Order of Leaf Node:
[(p_{leaf} − 1) × (size of key field + size of record pointer) + p × (size of block pointer) ≤ Block size.]

■■■

Scanned by CamScanner

Transactions and Concurrency Control

5

Transactions and Concurrency Control

Transaction Properties (ACID)

- **Atomicity:** Means execute all the operation or none of them.
- **Consistency:** Database should be consistent before and after the execution of the transaction.
- **Isolation:** Concurrent execution of two or more transaction.
- **Durability:** After commit, effect of transaction should persist.

Schedules

Sequences that indicate the chronological order in which instructions of concurrent transactions are executed.

Serial Schedule

- After commit of one transaction begins another transaction.
- Number of possible serial schedules with n transactions is n!
- No inconsistency.

Concurrent Schedule

- Simultaneous execution of two or more transaction.
- may result inconsistency.
- Better throughput and less response time.
- To maintain consistency transaction should satisfy ACID property

Conflicts in Concurrent Execution

- **WR problem:**

T_i	T_j
W(A)	⋮
⋮	R(A) → uncommited read

- **RW problem:**

T_i	T_j
R(A)	⋮
⋮	W(A)

Scanned by CamScanner

- WW problem:

T_i	T_j
W(A)	⋮
⋮	W(A)

Classification of Schedule based on Serializability

- **Serializability:** A schedule is serialisable if it is equivalent to a serial schedule.
  (*i*) Conflict serialisability
  (*ii*) View serialisability
- **Conflict serialisability:** Instructions I_i and I_j of transactions T_i and T_j respectively, **conflict** if and only if there exists some item Q accessed by both I_i and I_j, and at least one of these instructions wrote Q.
  (*i*) I_i = read (Q), I_j = read (Q). I_i and I_j don't conflict.
  (*ii*) I_i = read (Q), I_j = write (Q). They conflict.
  (*iii*) I_i = write (Q), I_j = read (Q). They conflict.
  (*iv*) I_i = write (Q), I_j = write (Q). They conflict.
- **Conflict equivalent:** If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S' are **conflict equivalent.**

  Conflict serialisable: We say that a schedule S is conflict serialisable if it is conflict equivalent to a serial schedule.

  View serialisability: Let S and S' be two schedules with the same set of transactions. S and S' are view equivalent if the following three conditions are met:
  (*i*) For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S', also read the initial value of Q.
  (*ii*) For each data item Q if transaction T_i executes **read** (Q) in schedule S, and that value was produced by transaction T_j (if any), then transaction T_i must in schedule S' also read the value of Q that was produced by transaction T_j.
  (*iii*) For each data item Q, the transaction (if any) that performs the final **write** (Q) operation in schedule S must perform the final **write**(Q) operation in schedule S'.

Scanned by CamScanner

Remember:

- Every conflict serialisable schedule is also view serialisable.
- Every view serialisable schedule that is not conflict serialisable has blind writes.

- **Irrecoverable Schedule:** Its not possible to roll back after the commitment of a transaction as the initial data is no where.

  Example:

	T_1	T_2
Roll back ↑	R (A)	
	W (A)	
	⋮	R (A)
		Commit
Failed ←		

- **Recoverable Schedule:** A schedule is recoverable if a transaction T_j reads a data items previously written by transaction T_i, the commit operation of T_i appears before the commit operation of T_j.

  Example:

T_i	T_j
R (A)	
W (A)	
⋮	
Commit	R (A)
	W(B)
	Commit

- **Cascadeless Recoverable Schedule:** For each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i the commit operation of T_i appears before the read operation of T_j. Every cascadeless schedule is also recoverable.

  Example:

T_i	T_j
R (A)	
W (A)	
⋮	
Commit	
	R (A)
	Commit

Cascading rollback: A single transaction failure leads to a series of transaction rollbacks.

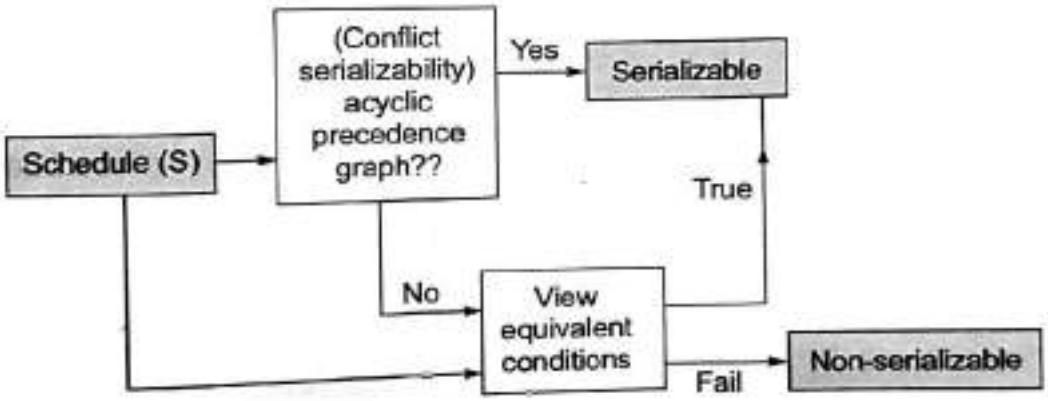
Scanned by CamScanner

- **Strict Recoverable Schedule:** If transaction T_i updates the data item A, any other transaction T_j not allowed to R(A) or W(A) until commit or roll back of T_i.

Example:

T_i	T_j
W (A)	
Commit or rollback	
	R (A) or W(A)

- S: $r_3(x)\, r_1(x)\, w_3(x)\, r_2(x)\, w_1(y)\, r_2(y)\, w_2(x)\, c_3 c_1 c_2$ is recoverable schedule but not cascadeless schedule.
- S: $r_1(x)\, r_2(z)\, r_1(z)\, r_3(x)\, r_3(y)\, w_1(x)\, w_3(y)\, r_2(y)\, w_2(z)\, w_2(y)\, c_1 c_2 c_3$ is irrecoverable.
- **Serializability checking:**



Concurrency Control with Locks

- Lock guarantees current transaction exclusive use of data item.
- Acquires lock prior to access.
- Lock released when transaction is completed.
- A locking protocol is a set of rules followed by all transactions while requesting and releasing locks.
- Locking protocols restrict the set of possible schedules.

Lock Granularity

Lock granularity indicates level of lock use: database, table, page, row, or field (attribute).

Scanned by CamScanner

Database Level

- The entire database is locked.
- Good for batch processes, but unsuitable for online multi-user DBMSs.

Table Level

- The entire table is locked.
- If a transaction requires access to several tables, each table may be locked.
- Table-level locks are not suitable for multi-user DBMSs.

Page Level

- A page has a fixed size and a table can span several pages while a page can contain several rows of one or more tables.
- Page-level lock is currently the most frequently used multi-user DBMS locking method.

Row-Level

- It allows concurrent transactions to access different rows of same table even if the rows are located on the same page.
- It improves the availability of data, but requires high overhead cost for management.

Field-Level

- It allows concurrent transactions to access the same row, as long as they require the use of different fields.
- The most flexible multi-user data access, but cost extremely high level of computer overhead.

Lock Types

Binary Locks

- Two states: locked (1) or unlocked (0).
- Locked objects are unavailable to other objects.
- Unlocked objects are open to any transaction.
- Transaction unlocks object when complete.
- Every transaction requires a lock and unlock operation for each data item that is accessed.
- It locks before use of data item and release the lock after performing operation.
- Problems with binary locks: Irrecoverability, Deadlock and Low Concurrency level.

Scanned by CamScanner

Shared/Exclusive Locks

1. **Shared (S Mode):**
   (*i*) Exists when concurrent transactions granted READ access.
   (*ii*) Produces no conflict for read-only transactions.
   (*iii*) Issued when transaction wants to read and exclusive lock not held on item.
2. **Exclusive (X Mode):**
   (*i*) Exists when access reserved for locking transaction.
   (*ii*) Used when potential for conflict exists.
   (*iii*) Issued when transaction wants to update unlocked data.

Request j ↓ \ Hold i →	S	X
S	Y	N
X	N	N

Note:

- **Lock-compatibility matrix:**
  (a) A transaction may grants a lock on item if the requested lock is compatible with locks already held on item by other transactions.
  (b) Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive on the no other transaction may hold any lock on the item.
- **Problems with Locking:**
  (a) **Transaction schedule may not be serialisable:** Managed through two-phase locking.
  (b) **Schedule may create deadlocks:** Managed by using deadlock detection and prevention techniques.

Two-Phase Locking

Two-phase locking defines how transactions acquire and relinquish locks.

1. **Growing phase:** Acquires all the required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked point.
2. **Shrinking phase:** Releases all locks and cannot obtain any new lock.

Governing rules of 2PL:

(*i*) Two transactions cannot have conflicting locks.
(*ii*) No unlock operation can precede a lock operation in the same transaction.
(*iii*) No data are affected until all locks are obtained.

Scanned by CamScanner

Basic 2PL Protocol

- Equal serial schedule based on lock point.
- **Problems with basic 2PL:** Irrecoverability, Deadlock and Starvation.

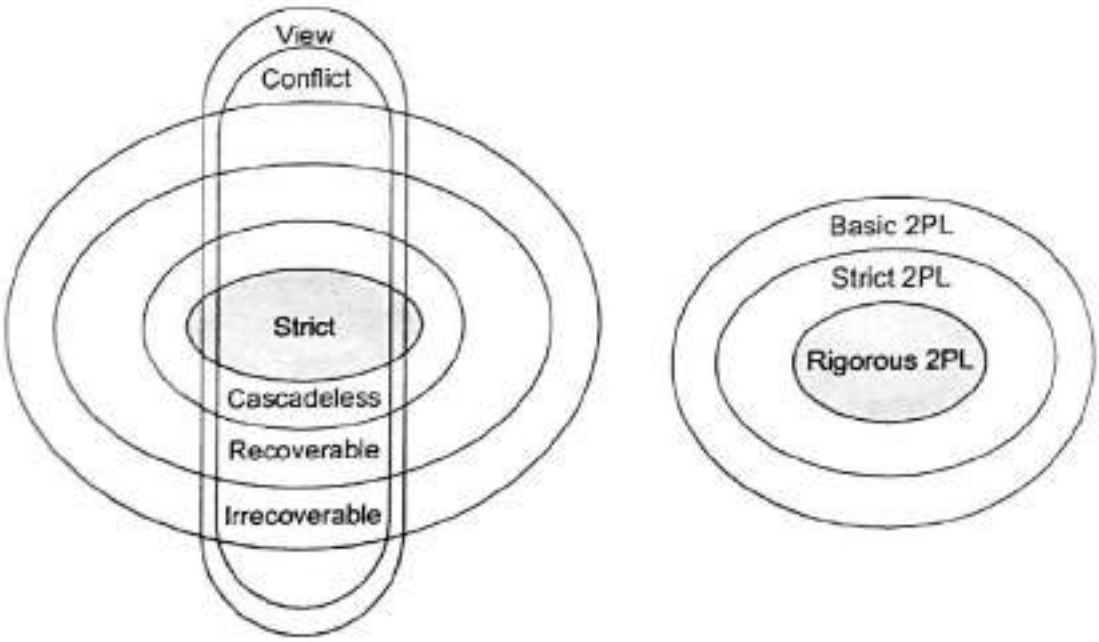
Strict 2PL Protocol

- Basic 2PL with all exclusive locks should be hold until commit/roll back.
- It ensures serializability strict recoverable.
- **Problems with strict 2PL:** Starvation and irrecoverability.

Rigorous 2PL protocol

- Basic 2PL with all locks (S/X) should be hold until commit.
- Equivalent serial schedule based on order of commit.

Class Diagram of Schedules with Serializability



Time Stamp Ordering Protocols

Assigns global unique time stamp to each transaction and produces order for transaction submission. Advantages with time stamping: Free from Deadlock and Ensuring serializability.

Basic Time Stamp Ordering Protocol

1. **Transaction T issues R(A) option:**
   (a) If WTS(A) > TS(T) then roll back T
   (b) Otherwise allow to execute R(A) successfully.
   Set RTS(A) = max {TS(T), RTS(A)}

Scanned by CamScanner

2. Transaction T issues W(A) operation:
   (a) If RTS (A) > TS (T) then roll back.
   (b) If WTS (A) > TS (T) then roll back
   (c) Otherwise execute W(A) option set WTS (A) = {TS(T)}

Thomas Write Rule Stamp Ordering Protocol

1. Transaction T issues R(A) option:
   (a) If WTS (A) > TS (T) then roll back T
   (b) Otherwise execute successfully.
   Set RTS (A) = max {TS (T), RTS (A)}
2. Transaction T issues W (A) operation:
   (a) If RTS (A) > TS (T) then roll back T.
   (b) If WTS (A) > TS (T) then ignore W(A) and continue execution of trans T.

Strict Time Stamp Ordering Protocol

A transaction T_2 that issues a R(A) or W(A) such that $TS(T_2) > WTS(X)$ has its read write option delayed until the transaction T_1 that write the value X has committed or rolled back.

Wait Die Protocol

- Write the transaction in ascending order of time stamp values.
- If T_1 required resource that is hold by T_2, T_1 wait for T_2 to unlock.
- If T_2 required resource that is hold by T_1, then roll back T_2 and restart with same time stamp value.

Wound Wait Protocol

- Write the transaction in ascending order of TS values.
- If T_1 required resource that is hold by T_2 then roll back T_2 and restart with same time stamp value.
- If T_2 required resource that is hold by T_1 than T_2 wait for T_1 to unlock.
- Both wait die and wound wait protocols may have starvation.

■■■

Scanned by CamScanner

A Handbook on Computer Science

10

Information Systems and Software Engineering

CONTENTS

1. Information Gathering, Requirement & Feasibility Analysis 305
2. DFDs and Process Specifications 308
3. I/O Design, Process Life Cycle 310
4. Planning and Managing the Project 314
5. Design, Coding, Testing, Implementation and Maintenance 316

Scanned by CamScanner

Information Gathering, Requirement & Feasibility Analysis

1

Software Requirements Specification

- A Software Requirements Specification (SRS), a requirements specification for a software system.
- It is a complete description of the behavior of a system to be developed and may include a set of use cases that describe interactions the users will have with the software.
- It also contains non-functional requirements. Non functional requirements impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints).
- The software requirements specification document enlists all necessary requirements that are required for the project developed.
- To derive the requirements we need to have clear and thorough understanding of the products to be developed. This is prepared after detailed communications with the project team and customer.

Purpose of SRS

1. **Establish the basis for agreement between the customers and the suppliers on what the software product is to do:** The complete description of the functions to be performed by the software specified in the SRS will assist the potential user to determine if the software specified meets their needs or how the software must be modified to meet their needs.
2. **Provide a basis for developing the software design:** The SRS is the most important document of reference in developing a design.
3. **Reduce the development effort:** The preparation of the SRS forces the various concerned groups in the customer's organisation to thoroughly consider all of the requirements before design work begins.

   A complete and correct SRS reduces effort waisted on redesign, re-coding and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings and inconsistencies early in the development cycle when these problems are easier to correct.

Scanned by CamScanner

4. **Provide a basis for estimating costs and schedules:** The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates.
5. **Provide a baseline for validation and verification:** Organisations can develop their test documentation much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured
6. **Facilitate transfer:** The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organisation and suppliers find it easier to transfer it to new customers.
7. **Serve as a basis for enhancement:** Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued product evaluation.

Requirement Analysis

- Software requirement document (SRD) describes in precise details what the customer wants.
- **SRD provides:**
  - (*i*) Agreement between customer and supplier
  - (*i*) Costing and scheduling
  - (*ii*) Validation and verification
  - (*iii*) All form of maintenance
- **SRD characteristics:**
  - (*i*) Defines all software requirements
  - (*ii*) Must be unambiguous
  - (*iii*) Must be complete
  - (*iv*) Must be verifiable
  - (*v*) Must be consistent
  - (*vi*) Must be modifiable
  - (*vii*) Must be traceable

Scanned by CamScanner

- **Techniques for gathering requirements:**
  - (*i*) One-to-one interview
  - (*ii*) Group interview
  - (*iii*) Prototyping
  - (*iv*) Requests for proposals
  - (*vi*) Questionnaires
  - (*vii*) Use cases
  - (*viii*) Brain storming

■■■

Scanned by CamScanner

DFDs and Process Specifications

2

Data Flow Diagram (DFD)

- It represents the flow of data between different process in a business.
- DFD involve 4 symbols:

  (*i*) **Process:** Transform of incoming data flow to outgoing flow

  (*ii*) **Data flow:** Movement of data in the systems.

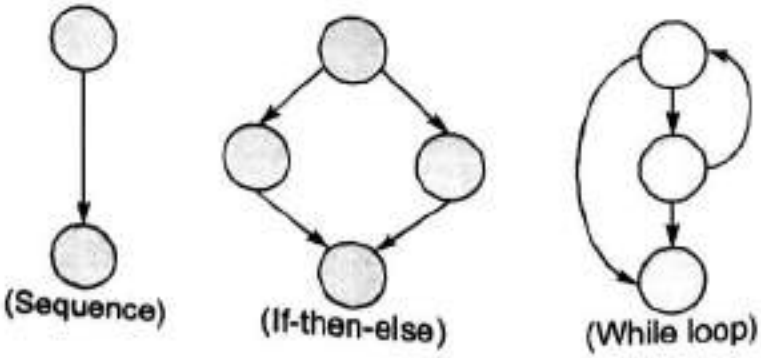
  (*iii*) **Data store:** Data repositories for data that are no moving.

  (*iv*) **External entity:** Sources of destinations outside the specified system boundary.

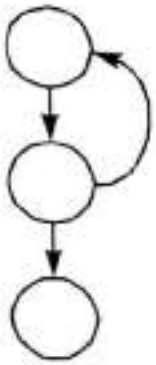
- DFDs are easy to understand, check and change data (advantage).
- In DFDs: Modification of data layout may cause entire layout to be changed (disadvantage).

Flow Graph

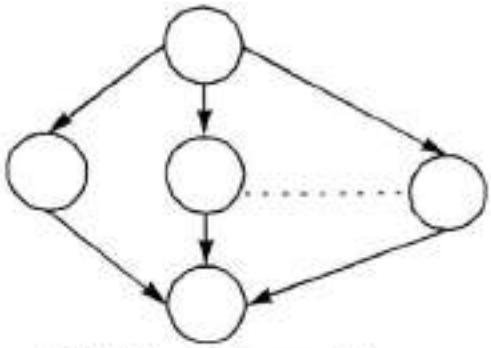
The flow graph is a directed graph in which nodes are statements and edges represent flow of control.



Scanned by CamScanner



(Do-while loop)



(Switch Statement)

Independent Path

An independent path must move along atleast one edge that has not been traversed before the path is defined.

Cyclomatic Complexity V(G)

It provides a quantitative measure of the logical complexity of a program. This approach is used to find the number of independent paths through a program.

$\boxed{V(G) = e - n + 2K}$ where e = Edges

n = Nodes/vertices

K = Connected components

$\boxed{V(G) = P + 1}$; P = Number of predicates (decisions)

$\boxed{V(G) = \text{No. of Regions}}$

$$\boxed{\text{Reachability Measure} = \frac{\text{Total no. of paths for all nodes}}{\text{Number of nodes}}}$$

■■■

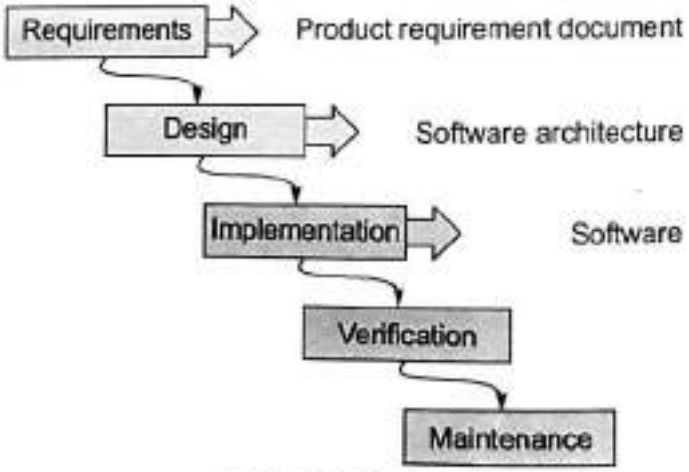
Scanned by CamScanner

I/O Design, Process Life Cycle

3

Software Life Cycle Models

1. **Waterfall Model:**
   (*i*) The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation, and Maintenance.
   (*ii*) The waterfall developed model originates in the manufacturing and construction industries; highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible.
   (*iii*) Also called as linear sequential process model and classical life cycle process model.
   (*iv*) Follows a disciplined approach.
   (*v*) Performance and Interface Risk.



Waterfall Model

2. **Prototyping Model:**
   (*i*) The prototyping model is a systems development method (SDM) in which a prototype (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed.
   (*ii*) The model can be used when the requirements cannot or will not be specified.

Scanned by CamScanner

(*iii*) The user can experiment with the system to improve the requirements.

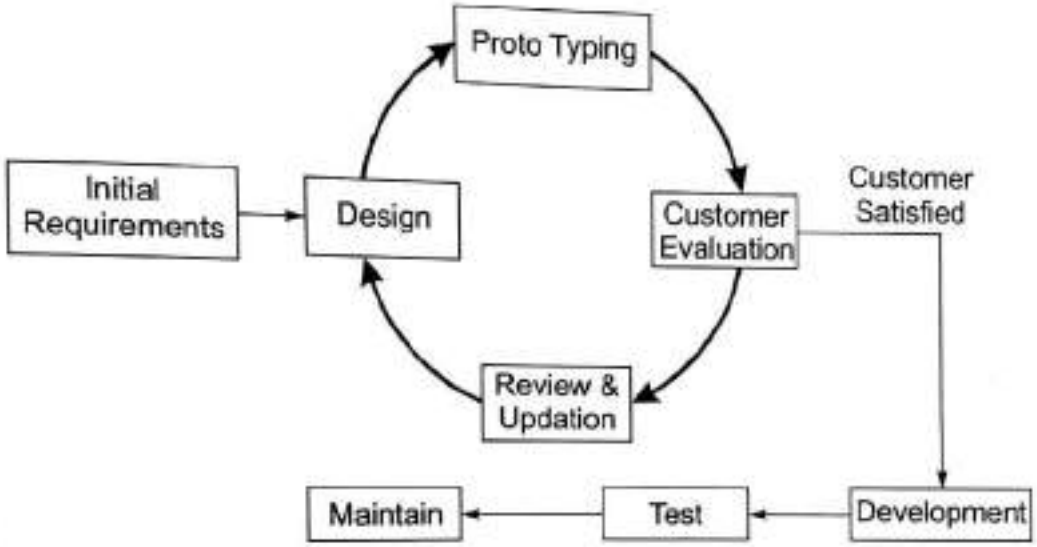
(*iv*) Use of the method is exploratory in nature and therefore constitutes a high-risk.

(*v*) Used when complete set of requirements are not available

(*vi*) A part of product is visible at an early stage

(*vii*) Less technical risks but expensive and time consuming

(*viii*) Scope of accommodating new requirements

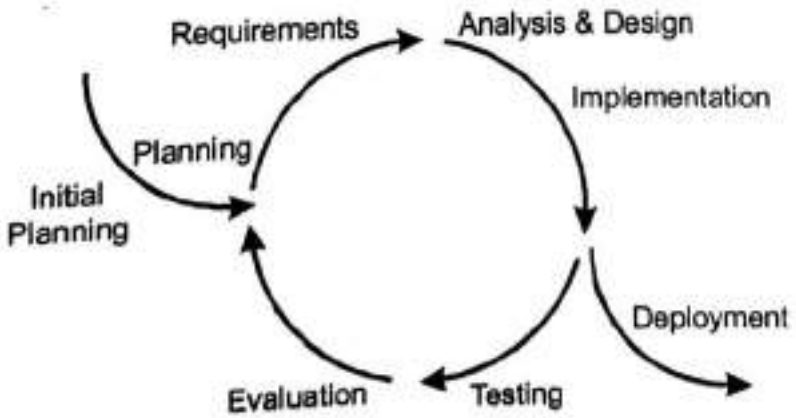


Prototyping Model

Iterative Enhancement Model:

(*i*) Delivers an operational quality product at each release

(*ii*) Complete product is divided into releases.



Iterative Model

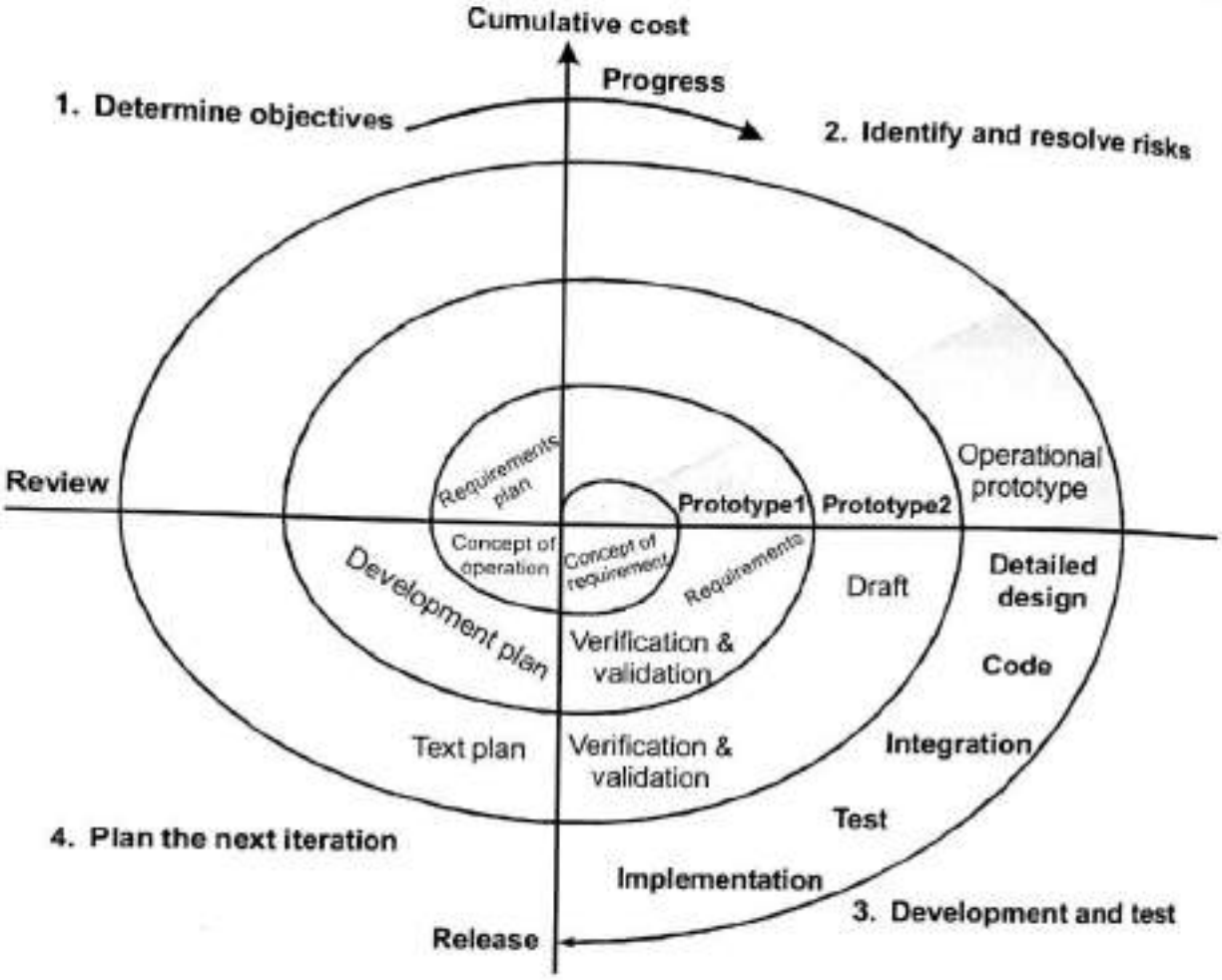
Spiral Model:

(*i*) Requires good expertise in risk management and project planning.

(*ii*) Iterative and realistic

(*iii*) Captures potential risks at an early stage.

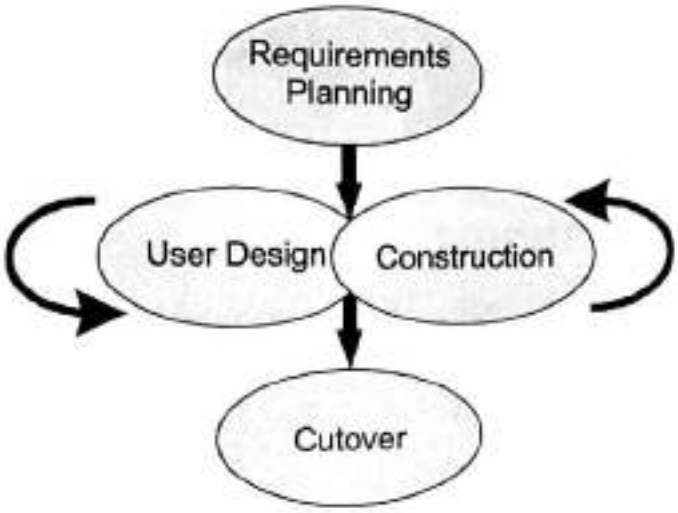
Scanned by CamScanner



Spiral Model

5. **Evolutionary Development Model:**
   (*i*) Similar to Iterative Model but this model doesn't require a usable product at the end of each cycle.
   (*ii*) Requirements are implemented by category rather than by priority.
6. **Rapid Application Development (RAD) Model:**
   (*i*) RAD is a software development methodology that uses minimal planning in favor of rapid prototyping.
   (*ii*) The "planning" of software developed using RAD is interleaved with writing the software itself.
   (*iii*) The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements.
   (*iv*) This model is used when there are tight deadlines, high pressure from customer, users have to be involved throughout.
   (*v*) Each major function is addressed by a separate RAD team.

Scanned by CamScanner



RAD Model

Agile Methodology

- Development team focuses on construction than design and documentation.
- Intended to deliver software quickly.

■■■

Scanned by CamScanner

Planning and Managing the Project

4

Software Project Planning

The project planning must incorporate the major issues like:

1. Size and Cost Estimation
2. Scheduling
3. Project Monitoring
4. Resource Requirement
5. Risk Management

Remember:

Estimated Value $EVx = (S_{optimistic} + 4S_{likely} + S_{pessimistic})/6$

Lines of Code (LOC)

Any line of a program text that is not a comment or blank line, regardless of number of statements or fragments of statements on the line.

Basic Metric

1. Effort = Size/Productivity
2. Productivity = Size/effort
3. Quality = Errors per LOC
4. Cost of Line = Cost per LOC
5. Cost of Software = Effort × Pay

Function Count

It measures functionality from users point of view.

$FP = UFP \times CAF$, where

FP = Function Point

UFP = Unadjusted Function Point

CAF = Complexity Adjustment Factor

Scanned by CamScanner

Cost Constructive Model (COCOMO)

- The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry W.Boehm. The model uses a basic regression formula with parameters that are derived from historical project data and current as well as future project characteristics.
- COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. The first level, *Basic COCOMO* is good for quick, early, rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes (*Cost Drivers*). *Intermediate COCOMO* takes these Cost Drivers into account and *Detailed COCOMO* additionally accounts for the influence of individual project phases.

Basic COCOMO

Basic COCOMO computes software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of source lines of code (SLOC), (KLOC).

COCOMO applies to three classes of software projects:

- **Organic projects:** "small" teams with "good" experience working with "less than rigid" requirements.
- **Semi-detached projects:** "Medium" teams with mixed experience working with a mix of rigid and less than rigid requirements.
- **Embedded projects:** Developed within a set of "tight" constraints. It is also combination of organic and semi detached projects. (hardware, software, operational, ...).

The basic COCOMO equations take the form

$$E = a(KLOC)^b \qquad D = c(E)^d$$

E = Effort in person-months

D = Development time in months

Modes	Standard Constant			
	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

■■■

Scanned by CamScanner

Design, Coding, Testing, Implementation and Maintenance

5

Software Design

Design

In this phase, the designer plans how a software system should be developed in order to make it functional, reliable, understandable, modifiable and maintainable.

The purpose of design phase is to produce a solution to a problem given in SRS document.

Module Coupling

Coupling is the measure of degree of interdependence between modules.

Note:

- Loosely coupled systems are made up of modules which are relatively independent.
- Highly coupled systems share a great deal of dependent between modules.
- Uncoupled modules have no interconnections at all.
- A good design will have low coupling. Thus, interfaces should be carefully specified in order to keep low value of coupling.

Most Desirable ← → Least Desirable

Procedural	Low	Inclusion	Import	External	Data	Stamp	Control	Communication	Content

Coupling Spectrum

Module Cohesion

Cohesion is a measure of degree to which elements of a module are functionally related.

Least Desirable → Most Desirable

Coincidental	Logical	Temporal	Procedural	Communicational	Sequential	Functional	Informational

Cohesion Spectrum

Maximize cohesion and minimize coupling.

Scanned by CamScanner

Software Testing

- Testing is the process of executing a program with the intent of finding errors.
  Complete or exhaustive testing is just not possible.
- **Verification and Validation:**
  (*i*) *Verification:* Checking the software with respect to specifications
  (*ii*) *Validation:* Checking the software with respect to customer's expectations.
- **Acceptance Testing:** Testing is conducted by the customer to validate all requirements.
- **Alpha and Beta Testing:**
  (*i*) Used when the software is developed as a product for anonymous customer.
  (*ii*) The alpha tests are conducted at the developer's site by customer.
  (*iii*) The Beta tests are conducted by the customer at their site.

Note: .....

Testing and debugging are two different processes which occurs at same time which identifies bug and the other removes the bug (respectively).

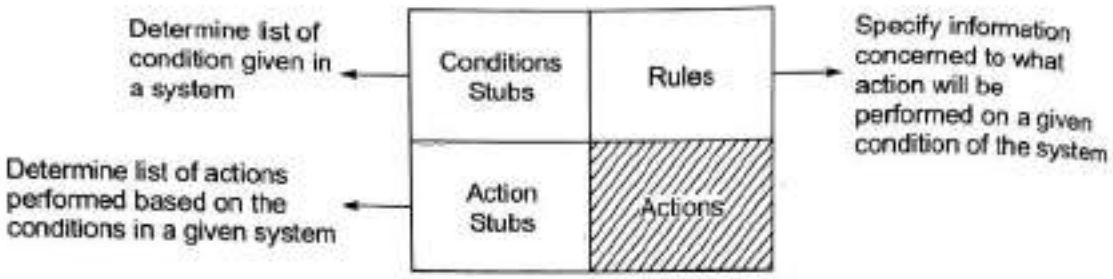
.....

- **Unit Testing:** Tests the functionality within the module.
- **Integration Testing:** Tests are focused more on interaction between modules.
- **System Testing:** Tests the software as part of the bigger system for which it was created.
- **Regression Testing:** Ensures that in process of modifying the existing system, the original functionality of the system was not disturbed.
- **Functional Testing:** It is also referred as "Black Box Testing" which involves only observation of the output for certain input values.
  (*i*) **Boundary Value Analysis:** The basic idea of boundary value analysis is to use input variable values at their minimum, nominal value and maximum.
  (*ii*) **Equivalence Class Testing:** Input domain of a program is partitioned into a finite number of equivalence classes such that one can reasonably assume that the test of a representative value of each class is equivalent to a test of any other value.

Scanned by CamScanner

(*iii*) **Decision Table Based Testing:** A decision table is most widely adapted graphical tool for representing process specifications.

Decision tables are useful for describing situations in which a number of combination of actions are taken under varying set of conditions. These are used to represent and analyze complex logical relationship.



- **Structural Testing:** A complementary approach to functional testing is called structural/white box testing. It premises us to examine the internal structure of the program.
  - (*i*) **Static White Box Testing:** If we want to test the program without running it.
  - (*ii*) **Dynamic White Box Testing:** If we want to test the program by running it.

White Box Testing → Flow Graph Approach
White Box Testing → Graph Matrix

- **Loop Testing (Logic Coverage):** Simple loops, Nested loops, concatenated loops, unstructured loops.
- **Stress Testing:** Upto which level a product will work, beyond it product will not work.

Note: .....

It is better to prevent the injection of defects rather than finding them and fixing them.

.....

- **Integration Testing:**
  - (*i*) **Top Down:** Top level modules are developed and tested first. Dummy bottom level module called "stub" is required.
  - (*ii*) **Bottom Up:** Bottom level modules are developed and tested first. Dummy loop level module called "Driver" is required.
  - (*iii*) **Sandwitch:** Combines both bottom up and top down integration and a layer is identified in between.

Scanned by CamScanner

(*iv*) **Big Bang:** Test all modules independently and then integrate the modules to form the software and finally the integrated software is tested as a whole.

Note: .....

Testing phase requires largest manpower.

.....

Maintenance

Any work done to change the software after it is in operation is considered to be maintenance work.

Types of Maintenance

1. **Corrective Maintenance:** This refers to modifications initiated by defects in the software.
2. **Adaptive Maintenance:** It includes modifying the software to match changes in the everchanging environment.
3. **Perfective maintenance:** It means improving processing efficiency or performance or restructuring the software to improve change ability.
4. **Preventive Maintenance:** There are long term effects of corrective, adaptive and perfective changes. This leads to increase in the complexity of the software. It is required to maintain it or reduce it which is done by preventive maintenance.

Software Configuration Management (SCM)

SCM tool helps in maintaining different versions of the configurable items.

Maintenance

- Mean Time Between Failure (MTBF) = MTTF + MTTR

$$\text{Availability} = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR}$$

- Software Maturity Index (SMI) = $\dfrac{[M_T - (f_a + f_d + f_c)]}{M_T}$

M_T = Number of modules released
f_a = Number of modules added
f_d = Number of modules deleted
f_c = Number of modules changed

Scanned by CamScanner

- Annual Cost Tariff (ACT) = $\frac{\text{No. of modules added + No. of modules modified}}{\text{Total Size}}$

Note:

In maintenance if size of product increases then the duration of maintenance decreases whereas in development it is directly proportional.

■■■

Scanned by CamScanner

A Handbook on Computer Science

11

Computer Networks

CONTENTS

ISO/OSI Stack, LAN Technologies (Ethernet, Token Ring) 322

IP (v4), (v6) and Routing Algorithms 332

TCP/UDP and Application Layer Protocols 340

Network Security (Cryptography) 348

Scanned by CamScanner

ISO/OSI Stack, LAN Technologies (Ethernet, Token Ring)

1

OSI Model

- Open Systems Interconnection Reference Model, developed in 1984 by the International Standards Organization (ISO).
- It is a way of sub-dividing a communications system into smaller parts called layers.
- A layer is a collection of conceptually similar functions that provide services to the layer above it and receives services from the layer below it.
- Provides a set of design standards for equipment manufacturers so they can communicate with each other. It is basic guideline for protocol development.

1. Physical Layer

- Conveys the bit stream through the network at the electrical and mechanical level.
- Defines physical means of moving data over network devices.
- Interfaces between network medium and devices.
- Defines optical, electric and mechanical characteristics: voltage levels, timing of voltage changes, physical data rates, transmission distances and physical connection.

2. Data Link Layer

- Takes a string of bits and delivers it across a link.
- Conveys the bit stream through the network at the electrical and mechanical level (i.e., Layer 1).
- Turns packets into raw bits and bits into packets.

Scanned by CamScanner

Framing and Error detection :

(*i*) Break the bit stream up into frames.

(*ii*) Compute an error-detection code.

(*iii*) Transmit each frame separately.

3. Network Layer

- Translates logical network address and names to their physical address (e.g., device name to MAC address).
- Responsible for :
  - (*i*) Addressing.
  - (*ii*) Determining routes for sending.
  - (*iii*) Managing network problems such as packet switching, data congestion and routing.
- Breaks the data into smaller unit and assembles data.
- Shields higher layers from details of how the data gets to its destination.

4. Transport Layer

- Divides streams of data into chucks or packets
- Reassembles the message from packets
- Provide error-checking to guarantee error-free data delivery, with no losses or duplications.
- Provides acknowledgment of successful transmissions.
- Requests retransmission if some packets don't arrive error-free.
- Provides flow control and error-handling.

5. Session Layer

- Establishes, maintains and ends sessions across the network.
- Responsible for name recognition (identification) so only the designated parties can participate in the session.
- Provides synchronization services by planning check points in the data stream.
- If session fails, only data after the most recent checkpoint need be transmitted.
- Manages who can transmit data at a certain time and for how long.

6. Presentation

- Translates from application to network format and vice-versa.

Scanned by CamScanner

- All different formats from all sources are made into a common uniform format that the rest of the OSI can understand.
- Responsible for protocol conversion, character conversions, data encryption/decryption, expanding graphics commands and data compression.
- Sets standards for different systems to provide seamless communication from multiple protocol stacks.

7. Application Layer

- Used for applications specially written to run over the network.
- Allows access to network services that support applications.
- Directly represents the services that directly support user applications (e.g., file transfer and email).
- What the user sees or does.

	PROTOCOLS
Application	NNTP, SIP, SSI, DNS, FTP, Gopher, HTTP, NFS, NTP, SMPP, SMTP, DHCP, SNMP, Telnet, Netconf
Presentation	MIME, XDR, TLS, SSL
Session	Named Pipes, NetBIOS, SAP, SIP, L2TP, PPTP
Transport	TCP, UDP, SCTP, DCCP
Network	IP (IPv4, IPv6), ICMP, IPsec, IGMP, IPX, AppleTalk
Data Link	ATM, SDLC, HDLC, ARP, CSLIP, SLIP, PLIP, IEEE 802.3, Frame Relay, ITU-T G.hn DLL, PPP, X.25
Physical	EIA/TIA-232, EIA/TIA-449, ITU-T V-Series, I.430, I.431, POTS, PDH, SONET/SDH, PON, OTN, DSL, IEEE 802.3 IEEE 802.11, IEEE 802.15, IEEE 802.16, IEEE 1394, ITU-T G.hn PHY, USB, Bluetooth

Token Ring (IEEE 802.5)

- **Characteristics of Token Ring**
  - (*i*) It offers connectionless communication.
  - (*ii*) It uses ring topology.
  - (*iii*) It uses token passing as an access control method.
  - (*iv*) There are no collisions, priorities are possible, offers deterministic service.

Scanned by CamScanner

- **Token Holding Time (THT):**
  (*i*) The maximum time a token frame can be with held by a station, by default it is set to 10 msecs.
  (*ii*) No station can keep the token beyond THT (It solves monopolization problem)
- **Monitor:**
  (*i*) The station with highest priority/MAC address and which generates the token frame is called monitor.
  (*ii*) Monitor maintains Minimum TRT and Maximum TRT.

  Min TRT = Propagation delay + (Number of stations × Delay at each station)
  Max TRT = Propagation delay + (Number of stations × THT)
- **Frame format of 802.5:**
  (*i*) To bypass a faulty station two data rates are there
  (a) 4 Mbps (minimum ring length of 1200 meters)
  (b) 16 Mbps (minimum ring length of 300 meters)
  (*ii*) Frame format of 802.5
  (a) Data Frame

	SD	AC	FC	DA	SA	Data	CRC	ED	FS
Bytes	(1)	(1)	(1)	(2/6)	(1)	(0-4500)	(4)	(1)	(1)

Starting Delimiter (SD): JK0JK000 [J&K are non data symbols]

Access Control (AC): PPPTMRRR $\begin{bmatrix} T = 1 \text{ for data} \\ = 0 \text{ for token} \end{bmatrix}$

Ending Delimeter (ED): JK1JK1IE [I = Intermediate frame Indication,E = Error detection bit]

Frame Status (FS): ACrrACrr [r = unused bit]

A (Destination)	*C* (Frame Acceptance)
0	0 → Destination not present and frame not accepted
0	1 → Not posible
1	0 → Destination present and frame not accepted
1	1 → Destination present and frame accepted

(b) Token Frame

SD	AC	ED
(1)	(1)	(1)

Scanned by CamScanner

(c) Abort Frame

SD	ED
(1)	(1)

- **Ring Latency (RL):** Time taken for a bit to travel around ring.

$$RL = \frac{d}{v} + \frac{Mb}{R} \text{ (seconds)} = \frac{dR}{v} + Mb \text{ (bits)}$$

where d : Length of link (meters), v : Velocity (m/s),
M : Number of station, R : Data rate of link (bps)

- Data transfer rate $= \dfrac{f}{t_1 + t_2 + \dfrac{f}{B}}$;

where f : Frame length, B : Channel capacity or bandwidth,
t_1 : Ring latency time, t_2 : token observing time

- Length of link (in bits) $= R \times \dfrac{d}{v}$

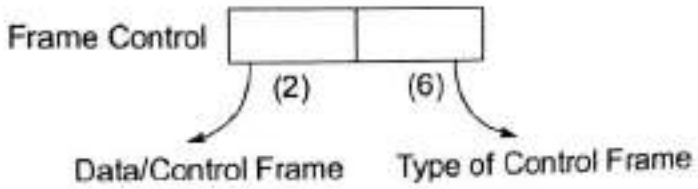
$$\text{Utilization} = \frac{T_1}{T_1 + T_2} = \frac{1}{1 + \dfrac{a}{N}}; \text{ if } a < 1$$

$$= \frac{1}{a\left(1 + \dfrac{1}{N}\right)}; \text{ if } a > 1$$

where T_1 : Average time to transmit a data frame
T_2 : Average time to pass a token
N : Number of stations

- **For minimum Token Size:** Propagation delay = Transmission delay
- **Types of Control Frame:**
  (*i*) **Becon Frame:** (000010) to find major faults in ring
  (*ii*) **Claim Token:** (000011) used in election process of monitor
  (*iii*) **Purge:** (000100) used to clear unwanted signals
  (*iv*) **Active monitor present:** (000101) to inform all the stations that monitor is alive
  (*v*) **Stand by monitor present:** (000110) to carryout neighbour identification process
  (*vi*) **Duplicate address test:** (000000) For virtual PC, physical address may change

Scanned by CamScanner



Delay Token Ring Reinsertion Strategy (DTR)	Early Token Ring Reinsertion Strategy (ETR)
(a) Token is reinserted after getting the entire data packet (b) Efficiency is low and always only one packet is available on the link (c) Reliability is high. THT = 10 ms (d) It is used under no load conditions (e) Cycle time = $a + b + c + d$	(a) Token is reinserted as soon as data transmission is over (b) Efficiency is high and many packets can be available on the link (c) Reliability is low, no meaning of THT (d) It is used under high load conditions (e) Cycle time = $a + c + d$ a : Packet transmission time b : Ring latency c : Token transmission d : Propagation delay between stations

Ethernet (IEEE 802.3)

- Characteristics of Ethernet:
  - (*i*) It offers connectionless communication
  - (*ii*) No acknowledgment, no flow and error control
  - (*iii*) It uses bus topologies for its operation
  - (*iv*) It uses CSMA/CD as accessing technology

Maximum duration for contention slot = 2 × Propagation delay

$$\text{Contention Period} = \frac{2 \times \text{Propagation delay}}{A}; A = 1/e$$

- Frame Format:

Preamble	SFD	DA	SA	Length	Data	CRC
7	1	6	6	2	(46 to 1500)	4

Bytes 7 → 1010....10; 1 → 10101011

Minimum Ethernet Frame Size:

$DA + SA$ + Length + Data (min) + CRC

$= 6 + 6 + 2 + 46 + 4$ bytes $= 64$ bytes

Maximum Ethernet Frame Size:

$DA + SA$ + Length + Data (max) + CRC

$= 6 + 6 + 2 + 1500 + 4$ bytes $= 1518$ bytes

Scanned by CamScanner

- Channel efficiency (utilization η) = $\dfrac{1}{1+\dfrac{2BLe}{vf}} = \dfrac{1}{1+5.44a}$

$$a = \frac{\text{Propagation delay}}{\text{Transmission delay}} = \frac{d/v}{f/B}$$

If length of packet increases η also increases

where f : Frame length, v : Speed of signal propagation

e : Contention slots/frame, L : Cable Length (d)

B : Network bandwidth, a : Propagation delay

$2T$: Duration of each slot = 2 × propagation delay

A : Slot probability = $(1/e)$

- The probability of one station succeeding in putting its traffic on a network of 'n' stations is given as $nP(1-P)^{n-1}$
- **CSMA/CD for ethernet:**

$$\text{Utilization } \eta = \frac{1}{1+6.44a}\begin{bmatrix}\text{used when ack information}\\ \text{is not provided}\end{bmatrix}$$

$$C.P. = \text{Number of slots} \times \text{Slot duration} = e \times 2 \times (d/v)$$

$$\text{Utilization} = \frac{T.P.}{T.P.+C.P.} = \frac{\left(\frac{1}{2a}\right)}{\left(\frac{1}{2a}\right)+\left(\frac{1-A}{A}\right)}$$

where, $\dfrac{1}{2a}$ = Transmision Interval

$\dfrac{1-A}{A}$ = Contention Interval

$$A = \left(1-\frac{1}{N}\right)^{N-1}; \; N = \text{Number of stations}$$

Flow Control

- The procedures to be followed by the transmitter sender and receiver for efficient transmission and reception is called as flow control.
- Two approaches
  - **Stop and wait – Error control (stop and wait ARQ)**
  - **Sliding Window Protocol (SWP)**

Scanned by CamScanner

Stop and Wait ARQ

- Only one frame at a time on the link ⇒ poor utilization ⇒ poor efficiency
- Efficiency $\eta = \dfrac{t_{transmit}}{(t_{transmit}) + (2\, t_{propagation})}$

$$= \frac{f/B}{f/B + RTT} = \frac{1}{1+2a} \quad \left[\because \; a = \frac{t_{propagation}}{t_{transmission}}\right]$$

- It is an example of closed loop control protocol.
- Positive ACKs are numbered in Stop and Wait.
  Negative ACKs are not numbered
- **Throughput = 1 window/RTT = 1 Packet/RTT**
- It is a **special category of SWP** of window size = 1

Go-back N (GBN) ARQ

- Receiver window size (RWS) = 1
- Sender window size (SWS) = $2^K - 1$ where, K is number of bits received for window size in the header.
- Efficiency = $(2^K - 1) \times \dfrac{t_{transmission}}{t_{transmission} + \text{RTT}}$
- $W_S + W_R \leq$ ASN (Available Sequence Number)
- Uses cumulative/Piggybacking acknowledgments
- GBN is called as "conservating protocol".

Selective Repeat (SR)

- $W_S = W_R = \dfrac{N}{2}$; $N \rightarrow$ maximum ASN (2^K)
- It uses piggybacking/cumulative/Independent acknowledgments.
- It accepts out of order of packets.
- Efficiency = $(2^{K-1}) \times \dfrac{t_{transmit}}{t_{transmit} + \text{RTT}}$
- With piggyback throughput = $\dfrac{2 \times \text{Packet}}{\text{RTT}}$
- Round Trip Time (RTT): It is minimum acknowledgment waiting time.
  RTT = 2 × Propagation delay
- **Time Out:** It is the maximum acknowledgment waiting time

Scanned by CamScanner

Wireless IEEE 802.11:

- **Wireless hosts:** In the case of wired networks, hosts are the end-system devices that run applications.
- **Wireless links:** A host connects to a base station (defined below) or to another wireless host through a wireless communication link.
- **Base station:** A base station is responsible for sending and receiving data (e.g., packets) to and from a wireless host that is associated with that base station. Access points in 802.11 wireless LANs are examples of base stations.

  Two Types of mode:

  (*i*) **Infrastructure mode:** since all traditional network are provided by the network to which a host is connected via the base station.

  Types of Infrastructure mode:

  (a) **Single-hop, infrastructure-based:** Base station is used in these networks which is connected to a larger wired network where all communication is between this base station and a wireless host over a single wireless hop.

  (b) **Multi-hop, infrastructure-based:** A base station is present that is wired to the larger network where some wireless nodes may have to relay their communication through other wireless nodes in order to communicate via the base station.

  (*ii*) **Adhoc networks:** wireless hosts have no such infrastructure with which to connect. In the absence of such infrastructure, the hosts themselves must provide for services.

  Types of Adhoc mode:

  (a) **Single-hop, infrastructure-less:** In this no base station that is connected to a wireless network.

  (b) **Multi-hop, infrastructure-less:** There is no base station in these networks, and nodes may have to relay messages among several other nodes in order to reach a destination.

- **Basic service set (BSS):** A BSS contains one or more wireless stations and a central base station, known as an access point (AP) in 802.11.
- Each 802.11 wireless station has a 6-byte MAC address also each AP also has a MAC address for its wireless interface.
- 802.11b and 802.11g use the 2.4 Ghz ISM band offering only 33 non-overlapping channels.
- 802.11a uses 55GHz ISM band offering at-least 23 non-overlapping channels.

Scanned by CamScanner

- 802.11n can use either the 2.4 GHz or the 55 GHz band while 802.11c uses only the 55 GHz band.
- **WiFi jungle:** Any physical location where a wireless station receives a sufficiently strong signal from two or more APs.
- **Scanning:** The process of scanning channels and listening for beacon frames is known as **Passive scanning**. In **Active scanning**, wireless station broadcasting a probe frame that will be received by all APs within the wireless host's range. APs respond to the probe request frame with a probe response frame. The wireless host can then choose the AP with which to associate from among the responding APs.

  (*i*) **Passive scanning:**
  - (a) Beacon frames sent from Aps.
  - (b) Association request frame sent: Host to selected AP.
  - (c) Association response frame sent: Selected AP to Host.

  (*ii*) **Active scanning:**
  - (a) Probe request frame broadcast from Host.
  - (b) Probes response frame sent from APs.
  - (c) Association request frame sent: Host to selected AP.
  - (d) Association response frame sent: Selected AP to Host.
- Instead of using collision detection, 802.11 uses collision-avoidance techniques. Second, because of the relatively high bit error rates of wireless channels, 802.11 (unlike Ethernet) uses a link-layer acknowledgment/retransmission (ARQ) scheme.
- The 802.11 MAC protocol uses **link-layer acknowledgments SIFS and DIFS.**
- Binary exponential backoff uses.
- **Hidden Terminal Problem** is avoided by using RTS (Request to Send) and CTS (Clear to Send).

Frame Format:

2 bytes	2 bytes	6 bytes	6 bytes	6 bytes	2 bytes	6 bytes	0 to 2312 bytes	2 bytes
FC	D	Address-1	Address-2	Address-3	SC	Address-4	Frame body	FCS

Frame Format

Frame Control										
Protocol version	Type	Subtype	To DS	From DS	More flag	Retry	Power management	More data	WEP	Rsvd
2 bits	2 bits	4 bits	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit

Subfield of Frame Control

■■■

Scanned by CamScanner

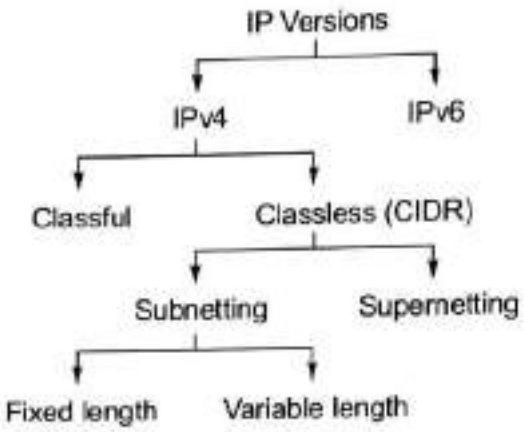
IP (v4), (v6) and Routing Algorithms

2

IPv4

IP Address

Used for global unique identification



IPv4 Address

- For flexibility 32 bit address is divided into 4 chunks each of 8 bits so that readability increases.
- IP address has 2 parts (*i*) Network ID (*ii*) Host ID.
- **Classfull Address:**
  - (*i*) Identification of classes in decimal notation
    Class *A*: (0-127), **Class *B*:** (128-191), **Class *C*:** (192-223), **Class *D*:** (224-239), **Class *E*:** (240-255)
  - (*ii*) Identification of a class in binary notation is done with the help of Network Identification Bit (NIB).

Class *A* : 0	Unicast (One-to-One)	*N·H·H·H*/8
Class *B* : 10		*N·N·H·H*/16
Class *C* : 110		*N·N·N·H*/24
Class *D* : 1110	Multicast	
Class *E* : 1111	R&D	

(a) **Network id:** All the host bits are made '0'. It is address of the network to which host belongs to.

(b) **Broadcast id:** Used to broadcast any packet to all machines in the network. All host bits are made '1'.

Scanned by CamScanner

(*iii*) Class A:

Number of network bits = 7, Number of host bits = 24

Number of hosts = $2^{24} - 2$, Number of networks = $2^7 - 2$

Note:

- Having all zeros/all ones either in network id or host id is ruled out and they are meant for special purpose.

(*iv*) Class B:

Number of network bits = 14, Number of host bits = 16

Number of hosts = $2^{16} - 2$, Number of networks = 2^{14}

(*v*) Class C:

Number of network bits = 21, Number of host bits = 8

Number of hosts = $2^8 - 2$, Number of networks = 2^{21}

- Special Purpose Addresses:

(*i*) Loop back Address:

(a) To verify the TCP/IP protocol suite of own machine i.e. used for self connectivity checking.

(b) It is also extensively used for interprocess communication.

(c) 127.X.X.X [X ranges from 0 to 255]

But 127.0.0.0 and 127.255.255.255 not allowed.

(*ii*) Default Route: If a packet destination address is not known, then if the default route is present in the routing table of the router, then the packet is not discarded but forwarded to the next router.

0.0.0.0

(*iii*) Direct Broadcasting: Packet sent to all in remote network.

N.255.255.255, N.N.255.255, N.N.N.255

(*iv*) Limited Broadcasting: Packet has to be broadcasted within same network. 255.255.255.255

- Private Address: This IP cannot be routed in the internet, but only in intranet range:

(*i*) 10.X.X.X, (*ii*) 172.16.X.X to 172.31.X.X, (*iii*) 192.168.X.X

- Limitations of Logical Addressing: No flexibility, No security, Not permanent to system.

- Rules to Deploy A Router:

Rule 1: All the interfaces of the router must belong to different networks.

Rule 2: The LANs interconnected must belong to different networks.

Scanned by CamScanner

Rule 3: The ethernet interface and the LAN must belong to the same network.

Rule 4: The routers sharing a same link must belong to the same network.

Solutions are: Supernetting, Subnetting, Physical addressing system.

Subnetting	Supernetting
(I) Dividing a network into multiple subnets is subnetting. (ii) Bits are borrowed from host id. (iii) It is applicable for single network. (iv) Number of 1s in subnet mask is always equal to more than network id bits.	(i) Aggregation of two or more n/ws to generate single IP address for the group is supernetting. (ii) Bits are borrowed from network id. (iii) It is applicable for two or more networks. (iv) Number of 1s always less than n/w id bits. (v) It reduces routing table entries. (vi) Network ids of the network must be in sequence.

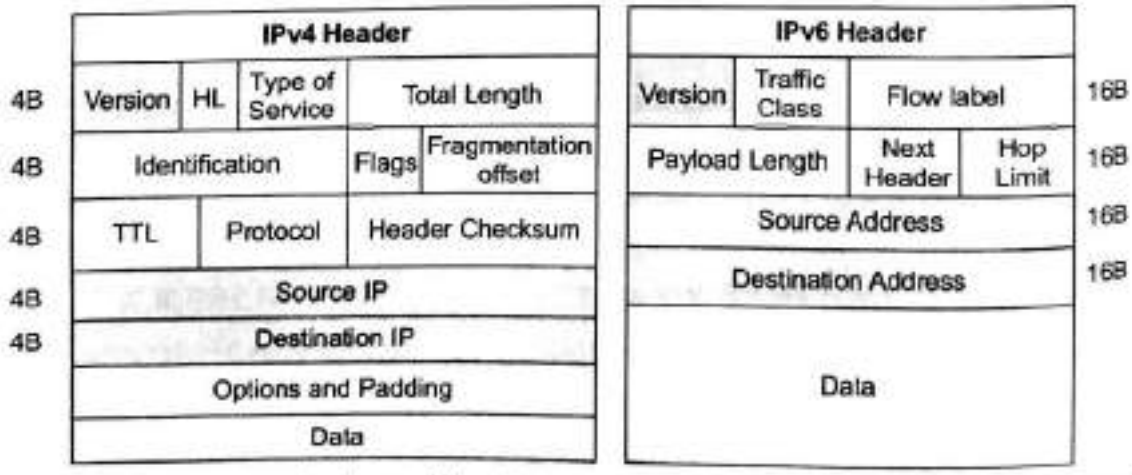
IP Datagram

IPv4

- It is **best effort** service and **connectionless** approach
- IP datagram

(20 to 60 Bytes)	
Header	Data (Transport Layer Header + Data)

- **Header Format (IPv4 and IPv6):**



- At source, header length is divided by 4 and at destination multiplied by 4. During fragementation, constant scale factor '8' must be used (fragments must be divisible by 8 except last segment).

Scanned by CamScanner

Options

- The header of the IPv4 datagram is made of two parts: a fixed part and a variable part.
- The fixed part is 20 bytes long and the variable part can be a maximum of 40 bytes.
- Options are not required for a datagram. They can be used for network testing and debugging.
- **End of Option:** An end-of-option option is a 1-byte option used for padding at the end of the option field. It, however, can only be used as the last option.
- **No Operation:** A **no-operation option** is a 1-byte option used as a filler between options.
- **Record Route:** A record route option is used to record the Internet routers that handle the datagram. It can list up to **nine router addresses.** It can be used for debugging and management purposes.
- **Strict Source Route:** A strict source route option is used by the source to predetermine a route for the datagram as it travels through the Internet.
- **Loose Source Route:** A loose source route option is similar to the strict source route, but it is less rigid. Each router in the list must be visited, but the datagram can visit other routers as well.

Timestamp: A timestamp option is used to record the time of datagram processing by a router.

IPv6

- **IPv6 Header** is always present and is a fixed size of 40 bytes.
- **Extenson Headers:** Zero or more extension headers can be present and are of varying lengths. A Next Header field in the IPv6 header indicates the next extension header. Within each extension header is a Next Header field that indicates the next extension header. The last extension header indicates the upper layer protocol (such as TCP, UDP, or ICMPv6) contained within the upper layer protocol data unit.
- **Upper Layer Protocol Data Unit:** The upper layer protocol data unit (PDU) usually consists of an upper layer protocol header and its payload (for example, an ICMPv6 message, a UDP message, or a TCP segment).
- The IPv6 packet payload is the combination of the IPv6 extension headers and the upper layer PDU. Normally, it can be up to 65,535 bytes long. Payloads greater than 65,535 bytes in length can be sent using the Jumbo Payload option in the Hop-by-Hop Options extension header.

Scanned by CamScanner

Values of the Next Header Field:

Value (in decimal)	Header
0	Hop-by-Hop Options Header
6	TCP
17	UDP
41	Encapsulated IPv6 Header
43	Routing Header
44	Fragment Header
46	ResourceReSerVationProtocol
50	Encapsulating Security Payload
51	Authentication Header
58	ICMPv6
59	No next header
60	Destination Options Header

- **IPv6 Extension Headers:** IPv6 extension headers that must be supported by all IPv6 nodes: (*i*) Hop-by-Hop Options header, (*ii*) Destination Options header, (*iii*) Routing header, (*iv*) Fragment header, (*v*) Authentication header, (*vi*) Encapsulating Security Payload header (*vii*) tension header must fall on a 64-bit (8-byte) boundary. Extension headers of variable size contain a Header Extension Length field and must use padding as needed to ensure that their size is a multiple of 8 bytes.
- It is recommended that extension headers be placed in the IPv6 header in the following order: (*i*) Hop-by-Hop Options header, (*ii*) Destination Options header (for intermediate destinations when the Routing header is present), (*iii*) Routing header (*iv*) Fragment header (*v*) Authentication header (*vi*) Encapsulating Security Payload header (*vii*) Destination Options header (for the final destination).
- When an IPv6 packet is fragmented, it is initially divided into unfragmentable and fragmentable parts:
- The unfragmentable part of the original IPv6 packet must be processed by each intermediate node between the fragmenting node and the destination. This part consists of the IPv6 header, the Hop-by-Hop Options header, the Destination Options header for intermediate destinations, and the Routing header.

Scanned by CamScanner

- The fragmentable part of the original IPv6 packet must only be processed at the final destination node. This part consists of the Authentication header, the Encapsulating Security Payload header, the Destination Options header for the final destination, and the upper layer PDU.

ARP (Address Resolution Protocol)

- Mapping from Logical to Physical Address
  - (*i*) When user knows IP address (obtained from the DNS if the sender is the host or it is found in a routing table if the sender is a router) but not the MAC address.
  - (*ii*) ARP Request (includes the physical and IP addresses of the sender and the IP address of the receiver) is broadcasted but ARP Reply(includes the recipient's IP and physical address) is uni-casted.
  - (*iii*) ARP can be useful if the ARP reply is cached (kept in cache memory for a while) because a system normally sends several packets to the same destination. A system that receives an ARP reply stores the mapping in the cache memory and keeps it for 20 to 30 minutes unless the space in the cache is exhausted. Before sending an ARP request, the system first checks its cache to see if it can find the mapping.
  - (*iv*) An ARP packet is encapsulated directly into a data link frame. An ARP packet is encapsulated in an Ethernet frame.

  Note that the type field indicates that the data carried by the frame are an ARP packet.

  Cases in which the services of ARP can be used:
  - (*i*) The sender is a host and wants to send a packet to another host on the same network.
  - (*ii*) The sender is a host and wants to send a packet to another host on another network.
  - (*iii*) The sender is a router that has received a datagram destined for a host on another network. It checks its routing table and finds the IP address of the next router.
  - (*iv*) The sender is a router that has received a datagram destined for a host on the same network.
- **RARP (Reverse Address Resolution Protocol):** mapping from Physical to Logical Address

Scanned by CamScanner

(*i*) Finds the logical address for a machine that knows only its physical address (Present on NIC).

(*ii*) The requesting machine must be running a RARP client program; the responding machine must be running a RARP server program.

(*iii*) A RARP request is created and broadcast on the local network. Another machine on the local network that knows all the IP addresses will respond with a RARP reply.

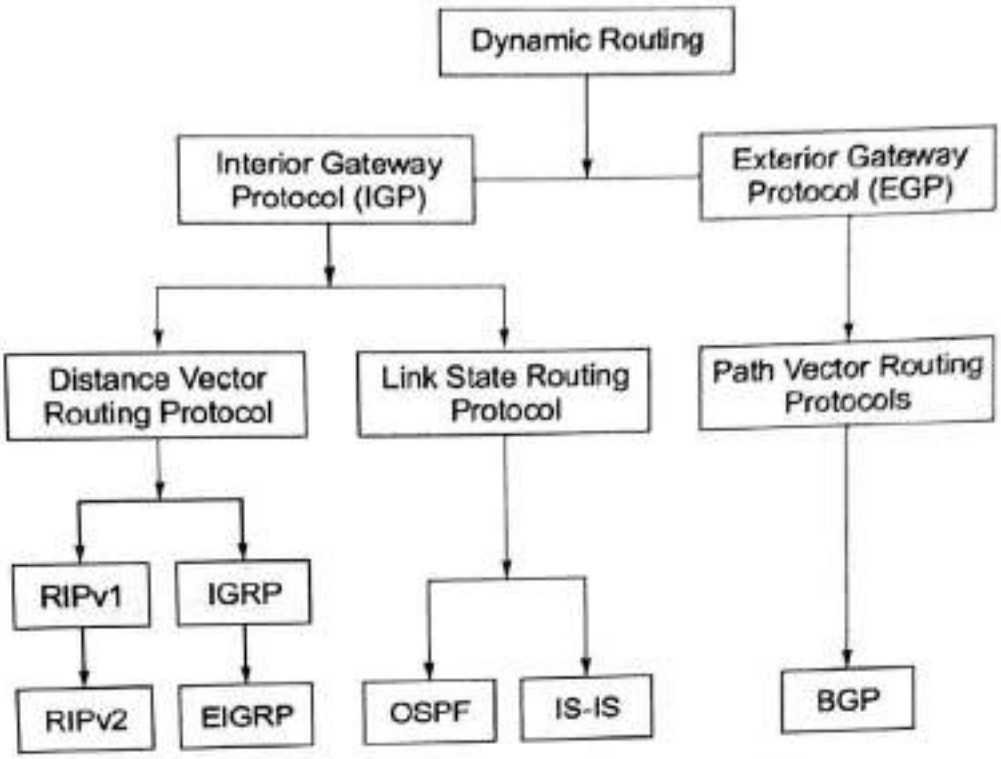
Problem with RARP: Broadcasting is done at the data link layer. The physical broadcast address, all 1's in the case of Ethernet, does not pass the boundaries of a network. This means that if an administrator has several networks or several subnets, it needs to assign a RARP server for each network or subnet.

Routing Algorithms

1. **Flooding:** Every incoming packet is sent out on every outgoing line except the one it arrived on.
2. **Broadcast Routing:** **(a)** Flooding, **(b)** Multidimensional routing, **(c)** Sink tree (spanning tree) and **(d)** Reverse path forwarding

   It sends the packets from a host to all other remaining hosts in the network simultaneously.
3. **Multicast Routing:** It sends the packets from a host to specific groups. **(a)** Shared tree, **(b)** Source based tree, **(c)** Least unit cost path tree and **(d)** Reverse path multicasting (uses prune message)
4. **Distance Vector Routing:** Routing only to neighbours and knowledge about whole network. RIP, IGRP uses split horizon technique.
5. **Link State Routing (OSPF) [Dijkstra's Algorithm]:** It discovers the neighbour of each router and learn their network addresses, cost of reaching each neighbours, sends the packet to all other routers.
6. **Path Vector Routing (BGP):** It lists all autonomous system visited inorder to reach the destination network by this route.
7. **Hierachical Routing (Intra region and Inter region routing):** Routing is based on regions and it maintains two tables namely "Full Table (for all routers in all regions) and "Hierachical Table" (for all routers within the region and knows about other regions but not routers of other regions).

Scanned by CamScanner



- **A problem** with distance vector routing is instability.
  - (*i*) **Two-node loop problem**
  - (*ii*) **Defining Infinity:** Most implementations of the distance vector protocol define the distance between each node to be 1 and define 16 as infinity.
- **A Solution: Split Horizon and Split Horizon and Poison Reverse.**
- **RIP (Routing Information Protocol):**
  - (*i*) RIP is an intra-domain routing protocol.
  - (*ii*) The metric used by RIP is very simple; the distance is defined as the number of links (networks) to reach the destination. For this reason, the metric in RIP is called a hop count.
  - (*iii*) **Infinity is defined as 16,** which means that any route in an autonomous system using RIP cannot have more than 15 hops.

  OSPF (Open Shortest Path First or OSPF):
  - (*i*) OSPF protocol is an intradomain routing protocol based on link state routing.
  - (*ii*) Its domain is also an autonomous system.
  - (*iii*) **Types of Links:** (a) Point-to-point link, (b) Transient, (c) Stub link, (d) Virtual link.
- **Path Vector Routing:** Path vector routing proved to be useful for interdomain routing.

■■■

Scanned by CamScanner

TCP/UDP and Application Layer protocols

3

User Datagram Protocol

- It is **connectionless** (No Acknowledgment) and **unreliable** approach.
- It is iterative compared to concurrent TCP.
- It is used for Broadcasting and Multicasting, all real time application (or) services utilises UDP with help of RTP (Video conference, Live Telecast) because it has fixed header size.
- It is message oriented and there is no flow control.
- **Pseudoheader** (Consist Source IP, Destination IP, Protocol used in IP, TCP/UDP segment length) of UDP is same as TCP.
- **Header Format:**

8 Bytes	Source Port (16)	Destination Port (16)
	Total Length (16)	Checksum (16) **optional**
	DATA	

- Applications that require bulk data transfer and fastness (than reliability) uses UDP.
- Protocols which takes services of UDP are NFS, SNMP-161, RIP-520, TFTP-69, DNS-53 and Ping (ICMP)-7.

Transmission Control Protocol

- It is reliable, port to port, byte/stream transport layer protocol.
- It supports **full duplex, connection oriented** (cumulative acknowledgment) approach.
- Not useful for broadcasting and multicasting.
- It is slow start protocol
- It uses sliding window protocol for flow control, the window size is set and controlled by receiver.
- TCP connections have three phases: (*i*) Connection establishment, (*ii*) Data Transmission and (*iii*) Connection Termination.

Scanned by CamScanner

- Header Format:

Source Port (16)								Destination Port (16)	4B
Sequence Number (32)									4B
Acknowledgment Number (32)									4B
Header Length (4)	Reserved (6)	URG (1)	PUSH (1)	ACK (1)	SYN (1)	FIN (1)	RST (1)	Advertised Window	4B
Checksum (16)								Urgent Pointer (16)	4B
Options									
Data									

Note:

- TCP uses random initial sequence number. After exhausting all the sequence number, sequence number are repeated.
- Probability of getting a number = $1/2^{32}$.

- Protocols which takes services of TCP are:
  HTTP (80), FTP (Data (20), Control (21)), Telnet (23), SMTP (25), POP3(110)

Congestion Control

Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened.

1. **In Open-Loop Congestion Control** policies are applied to prevent congestion (handled by either the source or the destination) before it happens.

- **Retransmission Policy (sometimes unavoidable)**
  (*i*) If the sender feels that a sent packet is lost or corrupted, the packet needs to be retransmitted, which may increase congestion in the network.
  (*ii*) The retransmission policy and the retransmission timers must be designed to optimize efficiency and at the same time prevent congestion.
- **Window Policy:** The Selective Repeat window is better than the Go-Back-N window for congestion control.

Scanned by CamScanner

- Acknowledgment Policy (policy imposed by the receiver).
- **Discarding Policy**: A good discarding policy by the routers may prevent congestion and at the same time may not harm the integrity of the transmission
- **Admission Policy**: An admission policy, which is a quality-of-service mechanism, can also prevent congestion in virtual-circuit networks.

2. **Closed-Loop Congestion Control**: Closed-loop congestion control mechanisms try to alleviate congestion after it happens.

   (*i*) **Backpressure:**

   (a) Is a node-to-node congestion control that starts with a node and propagates, in the opposite direction of data flow, to the source.

   (b) In backpressure, the warning is from one node to its upstream node, although the warning may eventually reach the source station.

   (*ii*) **Choke Packet:**

   (a) A choke packet is a packet sent by a node to the source to inform it of congestion.

   (b) In the choke packet method, the warning is from the router, which has encountered congestion (intermediate nodes are not warned), to the source station directly.

   (*iii*) **Implicit Signaling:**

   (a) No communication between the congested node or nodes and the source.

   (b) The source guesses that there is a congestion somewhere in the network from other symptoms.

   (*iv*) **Explicit Signaling:** The node that experiences congestion can explicitly send a signal to the source or destination.

   (*v*) **Backward Signaling:** A bit can be set in a packet moving in the direction opposite to the congestion warn the source that there is congestion and that it needs to slow down to avoid the discarding of packets.

   (*vi*) **Forward Signaling:** A bit can be set in a packet moving in the direction of the congestion warn the destination that there is congestion.

Traffic Shaping is a technique which involves in making a **Bursty Incoming Traffic** into an average output or a regulated output.

Scanned by CamScanner

Two Approaches
- (i) Leaky Bucket
- (ii) Token Bucket

M = Maximum regulated output rate
C = Capacity of bucket
ρ = Token filling rate
t = Time period of incoming traffic
$\therefore$ $C = (M - \rho)t$

- **TCP congestion Control:** Two approaches are used
  - (*i*) Slow Start and Additive Increase
  - (*ii*) **Multiplicative Decrease:** Congestion window always starts MSS = 1 and increases exponentially upto threshold value, then increases linearly upto Sender Window Size (SWS) and when timeout occurs, congestion window reduced to 1 MSS and threshold value reduced to half of congestion window size.
- **TCP Error Control:** 3 parameters are used
  - (*i*) **Checksum:** It detects errors.
  - (*ii*) **ACK:** There is no negative acknowledgment in TCP, as well as there is No ACK for received ACK.

    If any segment is corrupted (found through checksum), such segments are not acknowledged.
  - (*iii*) **Time-out:** Transport layer uses dynamic RTT for time out calculation. Different timers are deployed for error control like Time Awaited Timer, Keep-Alive Timer, Persistence Timer, Retransmission Timer.
- **Purpose of Pseudoheader:** It is used to identify whether packet is received by the correct destination or not. It is prepared at the source with source values, again it is prepared at destination with destination values.
- **TCP State Transition Diagram:** The functionality of TCP connection setup, communication phase and termination phase can be easily depicted by the state transition diagram where the TCP will be only at one state at a time w.r.t. server or client.

  Limitations:
  1. Error control procedures are not depicted.
  2. Retransmissions are not shown.

Socket

- The socket is the software abstraction used to represent the "terminals" of a connection between two machines.

Scanned by CamScanner

- Client and servers establish connections and communicate via sockets, which are communication links that are created over the Internet using TCP (or UDP).
- Sockets are the endpoints of internet communication stream or channel (logical).
- Clients create client sockets and connect them to server sockets.
- If two processes wish to communicate they must each create a socket. Once created, the socket must then be bound to a specific network address.
- One process must initiate the connection (This process is called the client). The other process must wait for a connection request (This process is the server). The client start by trying to establish a connection with the listening socket at a server.

BOOTP (Bootstrap Protocol)

- Client/server protocol designed to provide physical address to logical address mapping.
- BOOTP is not a dynamic configuration protocol.
- BOOTP is an application layer protocol. The administrator may put the client and the server on the same network or on different networks.
- BOOTP messages are encapsulated in a UDP packet, and the UDP packet itself is encapsulated in an IP packet.
- Client can send an IP(uses all 0's as as the source address and all 1's as the destination address).
- The BOOTP request is broadcast because the client does not know the IP address of the server.

  Problem with RARP solved by BOOTP: A broadcast IP datagram cannot pass through any router. One of the hosts can be used as a relay agent which knows the unicast address of a BOOTP server. After receives BOOTP broadcasting packet, it encapsulates the message in a unicast datagram and sends the request to the BOOTP server.

DHCP (Dynamic Host Configuration Protocol)

- The Dynamic Host Configuration Protocol (DHCP) has been devised to provide **static and dynamic** address allocation that can be manual or automatic.
- **Static Address Allocation:** In this case DHCP acts as BOOTP does. It is backward compatible with BOOTP. A DHCP server has a database that statically binds physical addresses to IP addresses.

Scanned by CamScanner

- **Dynamic Address Allocation:** DHCP has a second database with a pool of available IP addresses(which makes it dynamic).
  - (*i*) **The dynamic aspect of DHCP** is needed when a host moves from network to network or is connected and disconnected from a network. DHCP provides temporary IP addresses for a limited time.
  - (*ii*) The DHCP server issues a **lease** for a specific time. When the lease expires, the client must either stop using the IP address or renew the lease. The server has the option to agree or disagree with the renewal. If the server disagrees, the client stops using the address.
  - (*iii*) **One major problem with the BOOTP protocol** is that the table mapping the IP addresses to physical addresses needs to be manually configured while DHCP, on the other hand, allows both manual and automatic configurations.

Application Layer Protocols

Domain Name System (DNS)

- It is used to keep track of computers, services, resources both in LAN and WAN environment.
- It is used to map an URL to an IP address.
- It uses both TCP as well as UDP services on port 53.
- It uses 4 servers for application: (*i*) Root server, (*ii*) Top level domain server, (*iii*) Authoritative server and (*iv*) Local DNS.
- It offers 4 services: (*i*) Name Translation, (*ii*) Host Aliasing, (*iii*) Mail Aliasing and (*iv*) Load Balancing.
- It uses distributed databases to store information in terms of records.

File Transfer Protocol (FTP)

- It takes services of TCP < Data (20), Control (21)
- It has set of status codes and error codes.
- It has 3 modes of operation: (*i*) Active, (*ii*) Passive, and (*iii*) Extended passive mode.
- There are 2 modes of access:

Scanned by CamScanner

Two Modes of Access

FTP	TFTP
(i) For authorized users (ii) Username, password required (iii) Upload (iv) Uses TCP	(i) For anonymous users (ii) Username, password not required (iii) Download (iv) Uses UDP on port 69

Hyper Text Transfer Protocol (HTTP)

- It takes services of TCP on **port 80.**
- It is **"stateless protocol"** since it does not have any mapping from one transaction onto other and treats a request and reply as a pair everytime.
- It offers security through Secure Sockets Layer (a protocol for transmitting private documents via internet).
- It is used to access webpages from www.
- It has 8 methods for it's operation. They are: ***(i)*** Head, ***(ii)*** Get, ***(iii)*** Put, ***(iv)*** Post, ***(v)*** Delete, ***(vi)*** Trace, ***(vii)*** Connect and ***(viii)*** Options.
- There are two types of connection in HTTP:
  - ***(i)*** **Non Persistent:** For every process the connection is established newly.
  - ***(ii)*** **Persistent:** A single TCP connection is set on which multiple request and response can be made.

Simple Mail Transfer Protocol (SMTP)

- It uses port 25 on TCP.
- Components of SMTP are:
  - *(i)* User Agent (UA)
  - *(ii)* Mail Transfer Agent (MTA)
  - *(iii)* Multipurpose Internet Mail Extension (MIME)
  - *(iv)* Mail Access Protocol (MAP):
    - (a) POP3
    - (b) Internet MAP (IMAP4)

Internet Control Messaging Protocol (ICMP)

- It is used for error reporting and query messages which help in network debugging.

Scanned by CamScanner

- It is encapsulated into an IP datagram and then transmitted into the network, if the protocol field in IP datagram is 1 then the IP datagram is said to be carrying ICMP message.
- It uses the services of TCP and UDP on port 7 as ping command.
- **Types of Messages:**
  - (*i*) **Error reporting:** **(a)** Destination unreachable, **(b)** Source quench, **(c)** Parameter problem, **(d)** Time exceeded and **(e)** Redirection.
  - (*ii*) **Query Message:** **(a)** Router Solicitation and Router Advertisement, **(b)** Address Mask Request and Reply, **(c)** Time Stamp Echo Request and Reply and **(d)** Echo Request and Reply

Post Office Protocol Version 3 (POP 3)

- It is a pull protocol which takes services of TCP on **port 110.**
- At present IMAP4 is most used for MAP services

■■■

Scanned by CamScanner

Network Security (Cryptography)

4

Cryptography

- **Plaintext** (The original message, before being transformed) and **Ciphertext** (After the message is transformed).
- An **encryption algorithm** transforms the plaintext into ciphertext while a decryption algorithm transforms the ciphertext back into plaintext.

Types of cryptography

1. **Symmetric Key (Secrete key Cryptography):** Same key (shared) is used by the sender (for encryption) and the receiver (for decryption).

   Types of Symmetric Key Cryptography:

- **Substitution Cipher:** A substitution cipher substitutes one symbol with another: (a) Monoalphabetic cipher, (b) Polyalphabetic cipher.
- **Shift Cipher:** The simplest monoalphabetic cipher is probably the shift cipher.
- **Transposition Ciphers:** There is no substitution of characters; instead, their positions change. In a transposition cipher, the key is a mapping between the position of the symbols in the plaintext and cipher text.
- **Rotation Cipher:** Input bits are rotated to the left or right. The rotation cipher can be keyed or keyless.
  - (*i*) **In keyed rotation**, the value of the key defines the number of rotations (If the length of the original stream is N, after N rotations, we get the original input stream). The decryption algorithm for the rotation cipher uses the same key and the opposite rotation direction.
  - (*ii*) **In keyless rotation** the number of rotations is fixed.
- **Block Ciphers:** These ciphers are referred to as block ciphers because they divide the plaintext into blocks and use the same key to encrypt and decrypt the blocks.
  - (*i*) **Data Encryption Standard (DES):**
  - (a) The algorithm encrypts a 64-bit plaintext block using a 64-bit key.
  - (b) DES has two transposition blocks (P-boxes) and 16 complex round ciphers (they are repeated).

Scanned by CamScanner

(c) The initial and final permutations are keyless straight permutations that are the inverse of each other. The permutation takes a 64-bit input and permutes them according to predefined values.

DES Function is the **heart of DES**.

(*ii*) Triple DES:

(a) Uses three DES blocks where encrypting block uses an encryption-decryption-encryption combination of DESs, while the decryption block uses a decryption-encryption-decryption combination.

(b) Two different versions of 3DES are in use: (a) 3DES with two keys, (b) 3DES with three keys.

(c) To make the key size 112 bits and at the same time protect DES from attacks such as the man-in-the-middle attack, 3DES with two keys was designed.

(*iii*) Advanced Encryption Standard (AES):

(a) The Advanced Encryption Standard (AES) was designed because DES's key was too small.

(b) Triple DES increased the key size, the process was too slow.

(c) AES is a very complex round cipher. AES is designed with three key sizes: 128, 192, or 256 bits.

2. **Asymmetric-Key Cryptography (Public key Cryptography):** Two keys: a private key(kept by the receiver) and a public key(announced to the public).

- **RSA (Rivest, Shamir, and Adleman algorithm):**

(*i*) It uses two numbers, e and d, as the public and private keys.

(*ii*) RSA is a public-key cryptosystem that is often used to encrypt and decrypt symmetric keys.

(a) Bob chooses two very large prime numbers p and q. Remember that a prime number is one that can be divided evenly only by 1 and itself.

(b) Bob multiplies the above two primes to find n, the modulus for encryption and decryption. In other words, $n = p \times q$.

(c) Bob calculates another number $\phi(n) = (p - 1) \times (q - 1)$.

(d) Bob chooses a random integer e. He then calculates d so that $d \times e = 1 \bmod \phi$.

- Bob announces e and n to the public; he keeps $\phi(n)$ and d secret.

Scanned by CamScanner

(*i*) In RSA, e and n are announced to the public; d and $\phi(n)$ are kept secret.

(*ii*) Although RSA can be used to encrypt and decrypt actual messages, it is very slow if the message is long. RSA, therefore, is useful for short messages such as a small message digest or a symmetric key to be used for a symmetric-key cryptosystem.

(*iii*) RSA is used in digital signatures and other cryptosystems that often need to encrypt a small message without having access to a symmetric key.

(*iv*) RSA is also used for authentication.

- **Diffie-Hellman:**

(*i*) Diffie-Hellman, on the other hand, was originally designed for key exchange.

(*ii*) In Diffie-Hellman cryptosystem, two parties create a symmetric session key to exchange data without having to remember or store the key for future use.

(*iii*) The **symmetric key** for the session is K.

$(g^x \bmod p)^Y \bmod p = (g^Y \bmod p)^x \bmod p = g^{xy} \bmod p$.

(*iv*) **Problems arise in Diffie-Hellman:** Man-in-the-Middle Attack, It is also known as a bucket brigade.

Solution: The man-in-the-middle attack can be avoided if Bob and Alice first authenticate each other: **(a)** Message Confidentiality, **(b)** Message Integrity, **(c)** Message Authentication, **(d)** Message Nonrepudiation and **(e)** Entity Authentication.

(*v*) For a long message, symmetric-key cryptography is much more efficient than asymmetric-key cryptography.

- **Message Integrity**: Encryption and decryption provide secrecy, or confidentiality, but not integrity.

- **Message Digest Functions:** A message digest is a cryptographic hash function containing a string of digits created by a one-way hashing formula. Message digests are designed to protect the integrity of a piece of data or media to detect changes and alterations to any part of a message.

(*i*) Every bit of the message digest function's output is potentially influenced by every bit of the function's input.

(*ii*) If any given bit of the function's input is changed, every output bit has a 50 percent chance of changing.

Scanned by CamScanner

(*iii*) Given an input file and its corresponding message digest, it should be computationally infeasible to find another file with the same message digest value.

- **MD2(Message digest 2):**
  (*i*) This message digest is probably the most secure of Rivest's message digest functions, but takes the longest to compute.
  (*ii*) MD2 produces a 128-bit digest.
- **MD4(Message digest 4):**
  (*i*) This message digest algorithm was developed as a fast alternative to MD2.
  (*ii*) MD4 produces a 128-bit digest.
- **MD5(Message digest 5):**
  (*i*) MD5 is a modification of MD4 that includes techniques designed to make it more secure.
  (*ii*) MD5 and SHA-1 are both used in SSL and in Microsoft's Authenticode technology.
  (*iii*) MD5 produces a 128-bit digest.
- **SHA(Secure Hash Algorithm):** SHA produces a 160-bit digest.

SHA-I (Secure Hash Algorithm 1):

(*i*) SHA-I is a revised version of SHA.
(*ii*) Each creates a digest of length N from a multiple-block message.
(*iii*) Each block is 512 bits in length.
(*iv*) SHA-I has a message digest of 160 bits (5 words, each of 32 bits).

Message Authentication:

(*i*) A hash function guarantees the integrity of a message (guarantees that the message has not been changed).
(*ii*) The digest created by a hash function is normally called a modification detection code (MDC). The code can detect any modification in the message.

MAC (Message Authentication Code):

(*i*) To provide message authentication we need message authentication code.
(*ii*) An MDC uses a keyless hash Function a MAC uses a keyed hash function.
(*iii*) A keyed hash function includes the symmetric key between the sender and receiver when creating the digest.

Scanned by CamScanner

(*iv*) If the two MACs are identical, the message has not been modified and the sender of the message is verified.

- **Digital Signature:**

(*i*) A digital signature use a pair of asymmetric keys (a public one and a private one).

(*ii*) The sender can sign the message digest using receiver's public key, and the receiver can verify the message digest using own private key.

(*iii*) A digital signature can provide message integrity, message authentication, and non-repudiation.

■■■

Scanned by CamScanner

A Handbook on
Computer Science

12

Web Technologies

CONTENTS

1. HTML 354
2. XML 357
3. Basic Concepts of Client-Server Computing 362

Scanned by CamScanner

HTML

1

Introduction

- HTML is a tag based language. All the tags are predefined.
- Tags are interpreted by browser.
- HTML is used for developing webpages, it is compatible with http.
- HTML is formatting or presentation language.
- HTML is written in plaintext, therefore it is neutral or independent of hardware and software.
- HTML is standardized by world wide web console (wwwc).

Attributes

There are properties of tags. By changing the value of attribute the working of tag changes. Value of an attribute is enclosed in double quotes.

Example: <P align= "left"> Hi </P>

Basic HTML Tags

The basic structure for all HTML documents is simple and should include the following minimum elements or tags:

<HTML> The main container for HTML pages

<HEAD> The container for page header information

<TITLE> The title of the page

<BODY> The main body of the page

Body Tags or Formatting Tags

1. ** ** : Starts a new paragraph, tag is used to end the paragraph
2. **
** : Starts a new line without a blank line in between
3. **<b>** : **Bold** face any text between the tags.
4. ***<i>*** : *Italic* face any text between the tags.
5. **<u>** : Underline any text between the tags. Not widely used, because most underlined text on HTML pages is hyperlinks.
6. **<sup>** : Superscript any text between the tags.
7. **<sub>** : Subscript any text between the tags.
8. **<tt>** : Any text between the tags is uniformly spaced: good for aligning columns.

Scanned by CamScanner

9. **<hr>** : Draws a horizontal line. The attribute size = "50%" (any percent will do) draws a line of that percentage of the screen width. The attribute align = can be used to move the line to left, centre or right.
10. **<basefont>** : Sets a basic font size for normal browser text.
11. **<big>** : Increases the font size of text.
12. ** ** : Centers all the enclosed content.
13. **<comment>** : Inserts a comment into markup.
14. **<del>** : Identifies deleted content.
15. **<plaintext>** : Tells browser to treat all following text as plain text.
16. **<q>** : Defines an in-line quotation.
17. **<small>** : Sets a reduced font size for the enclosed text.
18. **** : Signifies strongly emphasized content.
19. **<var>** : Defines a variable part of a phrase or example.
20. **<wbr>** : Suggests where in a word a break should occur.
21. **<xmp>** : Example text.
22. **<s>** : Renders text with a horizontal strike (line) through the middle.
23. **<noscript>** : Provides alternative content for use when scripts aren't supported or are switched off.
24. **<ins>** : Identifies inserted content.
25. **<pre>** : Text between these tags is "preformatted". Spaces and line breaks appear as you entered them.

ists and Tables Tags

<ul></ul> : Begin and end an un-numbered list. Each individual item on the list begins with <li> and ends with </li>. Each item appears as a bullet point.

2. **<ol></ol>** : Begin and end a numbered list. Each individual item on the list begins with <li> and ends with </li>. Each item appears as a numbered point on a separate line.
3. **<table></table>** : Begin and end a table. Tables are built row-by-row, with each row starting with <tr> and ending with </tr>. Within the rows, individual cells start with <td> and end with </td>. In cells: text, images, links, forms, etc. can be inserted.
4. **<table border> </table>** : Begin and end a table that is surrounded by a border. This is the way to get a table that looks like an actual table.

Scanned by CamScanner

Form Tags

1. **<form action = "path/file" method = "post"> </form>** : A form begins and ends with these tags. When the submit button is clicked, the listed file (in a sub-directory of cgi-bin) will be executed. The target = new attribute is optional and means that the output of the CGI program will be sent to a new browser window.
2. **<text area rows = "8" cols = "60" name = "parameter"> </text area>** : This tag creates a text box that is 8 rows high and 60 columns wide. These boxes are typically used to cut-and-paste text into. Whatever text the programmer put between the tags will be printed in the box.
3. **<select name = "parameter"> <option> Label 1 </option> <option selected> Label 2 </option> </select>** : This tag creates a select box (a drop down box). The "selected" statement causes that option to be the default. When an option is selected by the user, the label between the option tags is sent as the value of the named parameter. If you want to allow the user to select multiple values, put the word "multiple" in the select tag.
4. **<input type = "submit" value = "button_label">** : This tag creates a submit button. There is no associated parameter; all this button does is send the form to the program for processing.
5. **<input type = "check box" name = "parameter">** : This tag creates a checkbox. The parameter is given a value of TRUE if the box is checked.

■■■

Scanned by CamScanner

XML

2

Introduction

- XML is a tag based language like HTML.
- HTML is used for displaying data whereas XML is used for transferring and storing data.
- XML doesnot have predefined tags, all tags are user defined.
- XML is case sensitive, written in plaintext and is compatible with http.
- XML is a hardware and software independent tool for transferring and storing data.
- XML parser is used to extract data from XML.

HTML	XML
1. It is a markup language that dictates how the text looks in the browser (thus for formatting and layout) *Example:* <sachin>	1. It is a cross platform, software and hardware independent tool for transmitting information. It describe the data and how it is organised. *Example:* <mydata> <name> sachin </name> <city> delhi </city> </mydata>
2. It has a finite set of pre-defined tags	2. In xml be can create our own tags.

Attributes in XML

XML tags can also have attributes but it is recommended to use tag instead of attributes, since attributes are not extensible.

Example:

```
<?xml version = "1.0"?>
<student>
<name> Sachin </name>
<dob> 5/6/91 </dob>
⋮
</student>
<?xml version = "1.0"?>
<student>
```

Scanned by CamScanner

```
<name> Sachin </name>
<dob>
<day> 5 </day>
<month> 6 </month>
<year> 91 </year>
</dob>
⋮
</student>
```

Syntax of XML

There are five syntactic rules given by XML that must be followed by every XML document.

1. Every XML document must have a root tag
2. Tags are case sensitive

   Example: <Roll> 10 </roll> ☒

   <roll> 10 </roll> ☑
3. Every opening tag must have a closing tag. It tag is empty then opening tag must have '/' at the end.

   Example: <name> Sachin </name>

4. Tags must be nested properly

   Example: <*i*> Hi </*i*> ☒

    <*i*> Hi </*i*> ☑
5. Value of an attribute must be enclosed in double quotes.

   <student dob = "5/6/91">

For XML parser we need to follow the following syntax:

1. **Well-formed:** If a XML document is following XML syntax then it is called as well formed. If a document is not well formed then XML parser cannot extract data so document is not useful.
2. **Valid:**
   (*i*) The structure of a XML document can be defined using DTD/XML scheme
   (*ii*) If a XML is following its structure then it is valid.
   (*iii*) Well formed condition is compulsory, valid is optional.

Scanned by CamScanner

Languages Developed from XML

1. **WML:** Wireless Markup Language is used for developing web pages using mobiles.
2. **WAP:** Wireless Application Protocol is used to access websites using mobile phones.
3. **RDF:** Resource Description Framework is used to share the resources like audio, video, ebooks etc.
4. **RSS:** Really Simple Syndicate is used to share news in the web.
5. **SMIL:** Synchronized Multimedia International Language is used for developing multimedia applications.
   Flex, Silver Light uses SMIL.

Languages Meant for XML

1. **XSL:** XML Style Sheet is used for defining how to display XML data.
2. XSLT : XML Style Sheet Transformation
   XSL - FO : XML Style Sheet Formatting objects
   } XML to HTML
3. **X-Query and X-Path:** X-Query retrieve data from XML but X-Path can retrieve data from single document whereas X-Query can retrieve data from multiple documents and it is like SQL.
4. **X-Link:** It is used to define relationship between XML documents. It plays a role like primary key and foreign key in SQL.
5. **X-pointer:** It is used to define relationship within same XML document.

Document Type Definition (DTD)

It is used to define structure of a xml document so that validation can be done.

Example:

student. dtd

```
<!ELEMENT student (name, roll, add)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT roll (#PCDATA)>
<!ELEMENT add (#PCDATA)>
```

Building Blocks of DTD

1. **ELEMENT** : It is used for defining tag

Scanned by CamScanner

2. **ATTLIST** : It defines attributes
3. **ENTITY** : Shortcuts for special symbols. It starts with '&' and ends with ';'

Entity	Symbol
& lt;	<
& gt;	>
& amp;	&
& nbsp;	space
& copy;	©
& reg;	®
& trade	™

Entities of html are also used in xml. We can define our own entities (shortcuts) in DTD and can use then in xml.

Example: ***DTD***

<!ENTITY WHO "World Health Organisation">

XML

<org> &WHO; </org>

4. **CDATA**: CDATA stands for character data. XML can have sections that contain characters of any kind that are not parsed.
5. **PCDATA**: PCDATA means parsed character data. i.e. if we have a character data element declared as PCDATA then all characters or text or data inside the XML tags will be parsed by the XML parser.

Types of Elements/Tags

1. **Optional Child Element :**

<!ELEMENT product (name, cost | price)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT cost (#PCDATA)>
<!ELEMENT price (#PCDATA)>

2. **Empty Element** : It donot contain data, it must have one attribute then only it is useful.

Example : <payment/> <!ELEMENT payment EMPTY>

3. **Multiple Occurence of a Child Element:**

<customer>
<name> Sachin </name>
<phone no.> 4444422222 </phone no.>

Scanned by CamScanner

```
<phone no.> 99119596699 </phone no.>
<phone no.> 6666633333 </phone no.>
</customer>
<!ELEMENT product (name, phone no.+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT phone no. (#PCDATA)>
```

Note:

- + One or more
- * Zero or more
- ? Zero or one

XML Schema

The XML schema file, like the DTD file, contains only metadata. In addition to the definition and structure of elements and attributes, an XML schema contains a description of the type of elements and attributes found in the associated XML file.

- It is used to define structure of xml document.
- It is alternate of DTD
- It is written in form of xml
- It supports data type, therefore it is easy for mapping data from database to xml document and it is reusable.
- It support name spaces to avoid name conflict.
- It is possible to impose restrictions on data and to refine the pattern of data.

■■■

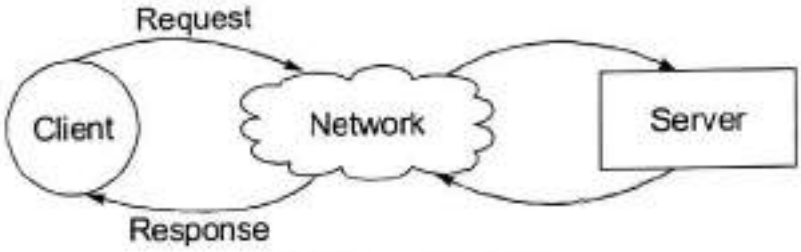
Scanned by CamScanner

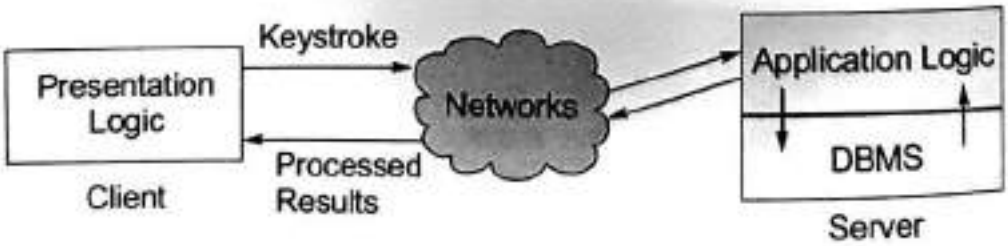
Basic Concepts of Client-Server Computing

3

Client Server Model

- A simple definition of client server is "server software accepts requests for data from client software and turns the results to the client".
- Client application program running on the local machine requests a service from another application program-server-running on the remote machine.
- Commonly server provides service to any client, not a particular client.
- Generally, a client application program that requests a service should run only when it is needed. A server program providing service should run all the time, as it does not know when its services will be needed.
- In client-server computing major focus is on SOFTWARE.
- **Elements of client-server computing:** Request-response may be repeated several times, the process is finite.



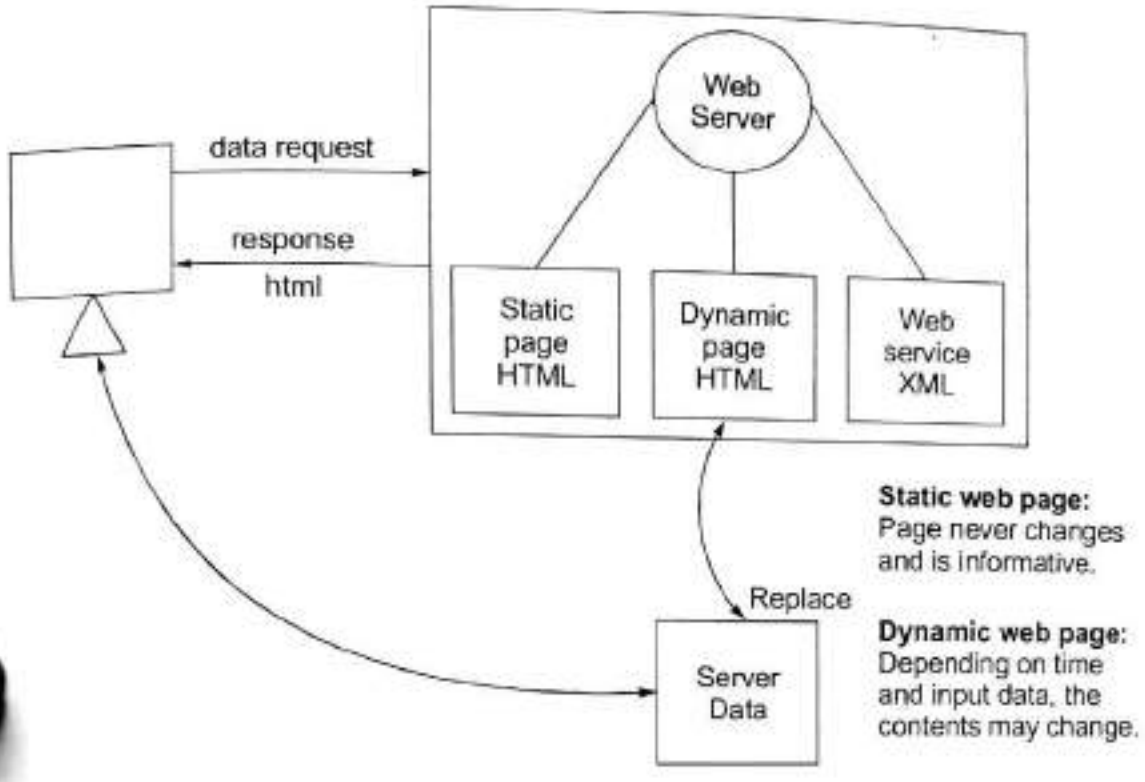


Client-Server Model

- Client-server computing is distributed access, not a distributed computing.
- Most typical use of technology in client-server is data base server. It accepts requests for data, retrieves the data from its database (or requests data from another node) and passes the results back.

Scanned by CamScanner

Server Side Programming/Dynamic HTML



Servlet

- It is a Java program which runs within the context of web servers, so only one process is there and no IPC mechanism.
- Runtime environment of a servlet is called servlet container. There are two types of containers:
  - (*i*) **Generic servlet** : any protocol.
  - (*ii*) **http servlet** : http protxocol.

Servlet	Applet
(1) Lifetime may be years (2) Servlet runs inside the web server to generate dynamic html or server side (3) Methods init ( ), service (request, response), destroy ( ) are called by web server	(1) Lifetime is more than 1 day (2) Applet runs inside the browser to make the page dynamic or server side (3) Methods init ( ), start ( ), stop ( ), destroy( ), paint ( ) are called by browser

Servlet Methods

1. **init ( )** : It is called when the servlet is deployed. It is used for starting and initialising the **servlet**.
2. **Service ( )** : It is called when a servlet gets a request. If there are multiple request, service method will run in multiple threads.

Scanned by CamScanner

3. **Destroy ( )** : It is called when the service is undeployed or removed from the servlet.

Types of Contents on a Website

1. **Webpages (html)**: Webpages are primary things. Following types of pages are possible:
   (*i*) **Static:** HTML
   (*ii*) **Dynamic:** HTML + Programming language/server side script.
   (*iii*) **Client side dynamic:** HTML + Javascript
   (*iv*) **Both sides dynamic:** HTML + Programming language, server side script + Javascript.
2. **Images/Audio/Video**: Audio/Video will play on the client side only.
3. **CSS:** Cascading Style Sheets are used for defining the styles of tags in html. CSS improves html.
4. **Script:** Script is embedded within html. Browser executes the script. Server side dynamic pages are returned using ASP, Servlet, JSP, PHP, ASP.NET whereas client side dynamic pages are returned using Javascript, VBscript.

Note: .....

- Javascript can communicate with server to get some data or information for updation but not the complete html page.
- Javascript can bypass the browser and send a request to a server side program, server side program will respond to program in form of xml and Javascript will update the data on a page.

.....

5. **XML:** It is used for representing the data where HTML is used for formatting the data.
6. **AJAX:** Asynchronous Javascript XML, a new technique used to make pages dynamic on both sides.
   The purpose of AJAX is to achieve multitasking, since there is not only one script running all task rather multiple scripts are used.
7. Client side components
8. Server side components

Note: .....

- A component is a reusable piece of software which is reused at runtime, it provides its functionality through set of functions known as interface.

.....

■■■

Scanned by CamScanner