```
ptr = &x :
printf ("%d", *ptr);        //invalid
```

you need to typecast the void pointer before dereferencing.

i.e.,

```
void *ptr
int x = 10;
ptr = &x;
printf("%d",*(int*)ptr);
                 └→typecasting
```

Here, (int*)ptr does type costing of void

(int) ptr dereferences the typecasted void pointer variable

- Pointer arithmetic in not possible on void pointer.
  i,e……. ptr+ +, + + ptr, – –ptr, ptr – –, ptr +1, ptr -1 is not allowed an void pointer.

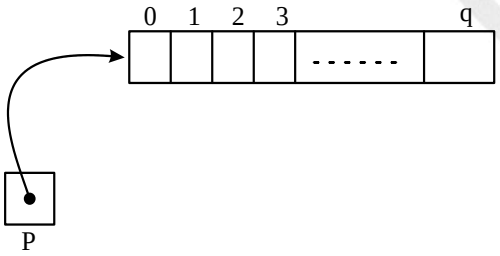
- An uninitialized pointer holds a garbage value while a NULL pointer holds a defined value but not a valid address.

4.1.10 Memory leakage problem

- Whenever a pragrammer allocated memory dynamically then it is the reponsibility of the programmer to free that memory after usage.
- Memory leakage problem occurs when a programmer allocates memory dynamically but does not de-allocate it after using it.
- It reduces the performance of the computer by reducing the amount of availbale memory.

4.1.11 Understanding Declartion

1. int *(P[10]) : P is an array of 10 pointer to integer
2. int (*P)[10] : P is a pointer to an array of 10 integers



3. int (*P)() : P is pointer to a function that takes no argument and returns an integer.
4. int (*P) (int, int) : P is a pointer to a function that takes 2 integer arguments and returns an integer.
5. int *P (char*) : P is a pointer to a function that takes a pointer to character argument and return a pointer to integer.

4.1.12 Pointer to Functions / Function Pointer

- Just like normal pointers we can have pointers to functions.

```
int(*p)(int, int)
```

P is a pointer to a function that take 2 integer arguments and returns an integer value.

- **Declaration of function pointer:** declaring a pointer to function is almost same as declaring the fuction except that in function pointer are prefix is with an asterik * symbol.
  For example: if the function is

  int add(int, int)

  Declaration of a fuction pointer for add() fuction is :

  int (*ptr)(int, int);
- How to call a function through function pointer : In order to all a function using function pointer, first we need to assign the address of function code to the pointer

Ex 1:

```
int add (int, int):
void main () {
        int (*ptr) (int, int);
        ptr = &add;       // ptr is a pointer to add() function
printf ("The sum of 10 and 20 is", (*ptr )(10, 20)); //calling add()
}
int add (int, int);
```

Ex 2:

```
void main() {
        int (*ptr) (int, int);
        ptr =  add ; // same as ptr = & add
printf (" The  sum of  10 and 20 is", (*ptr) (10, 20));
```

- A function can be called using following 4 ways using pointer to fuction.

int add (int int);
void main() {
int (*ptr) (int, int);
ptr = &add;
printf ("%d", (*ptr)(10, 20));
}
	int add (int int);
	void main() {
	int (*ptr) (int, int):
	ptr = add;
	printf ("%d", (*ptr) (10, 20));
	}

int add (int, int) ;
void main() {
int(*ptr) (int, int);
prt= &add;
printf ("%d", (*ptr)(10, 20));
}
	int add (int, int);
	void main() {
	int (*ptr) (int, int);
	prt= add;
	printf ("%d", (*ptr) (10, 20));
	}

- Using function poitnter we are able to pass a function as an argurment to other function and can also be returned from function.
  Array of function pointer
  int (*ptr[3])(int, int)
  Ptr is an array of 3 pointers to a function that takes 2 arguments and returns an integer.

❐❐❐

5 STRINGS

5.1 Strings

- Sequence of characters terminated by a null character '\0'.
  Syntax:
  char str_name[size];
- Initializing a string in c
  - char str[] = "Pankaj";
    Here, str is internally a pointer (holds the address of first character)
  - char str[20] = "Pankaj";
    Here, we predefine the size as 20 but we should always take one extra space along with size of string.
    i.e atleast 7 size must be there to store "Pankaj" (6+1)
  - char str[7] = {'P', 'a', 'n', 'k', 'a', 'j', '\0'};
    must set the end character as '\0'
  - chat str[] = {'p', 'a', 'n', 'k', 'a', 'j', '\0'};

5.1.1 Reading a string

- **scanf():**
  when we use scanf() to read, we use %s as a format specifier without using "&" to access the variable address because an array name acts as a pointer.

```
#include <stdio.h>
    int main (){
        char name [20];
        printf("enter your name \n");
        scanf("%s",name);
        printf("Hello %s",name);
          }
```

Output:

```
Enter your name
Pankaj Sharma
Hello Pankaj      // why not Hello Pankaj Sharma
```

The above code did not print Hello Pankaj Sharma as expected because scanf halts reading as soon as a whitespace or a newline character is encountered, and that is why it reads only Pankaj.

- In order to read a string containing space, we use gets() function. gets() ignores the whitespace & stops reading when a newline is found.

```
# include<stdio.h>
```

```
void main(){
        char name [20];
        printf("Enter your name :");
        gets(name);
        printf("Hello %s", name);
        }
Output :
        Enter your name: Pankaj Sharma
        Hello Pankaj Sharma
```

- A string literal always represent base address of a string. i.e address of first character.
- "Pankaj" represents address of character 'P'

Hence "Pankaj" [0] means 'P'

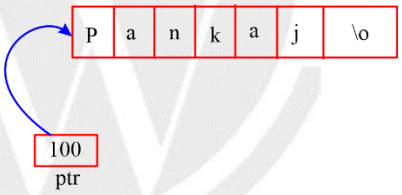
"Pankaj" [1] ≡ *("Pankaj"+1) ≡ *(Address of 'P'+1)
≡ *(Address of 'a')
≡ 'a'

5.2 Strings and pointers

- Array name represents the address of its first element. Similar to character arrays, we can create a character pointer to represent a string that will hold the starting address i.e address of first character of string.

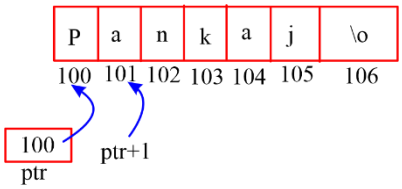
Example - 1

```
# include<stdio.h>
void main(){
        char *ptr = "pankaj"
        printf("%s",ptr);
        }
Output : pankaj
```



Example – 2

```
# include<stdio.h>
void main(){
        char *ptr = "Pankaj"
        printf("%s",ptr+1);
        }
Output: ankaj
```



5.3 Predefined functions for string

(1) strlen (): returns the length of the string passed as an argument.

(2) strcpy (): copies the contents of one string to another. strcpy (S_1,S_2) copies the content of string S_2 to string S_1.

(3) strcmp (): compares the first string with the second string. If both are same it returns 0.
strcmp (str1,str2) returns 0 if both strings are same.

(4) strcat (S_1,S_2): concat S_1 string with S_2 string and the result i.e concatenation of S_1,S_2 is stored in S_1.

5.4 Important concepts

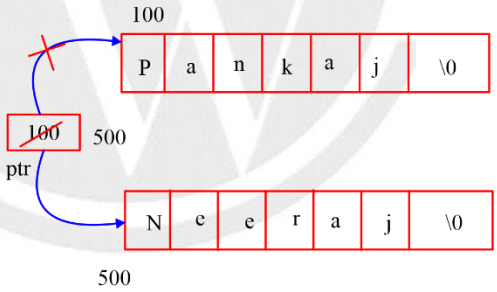
(1)
```
# include<stdio.h>
void main(){
        char name [20] = "Pankaj" ;
        name = "Neeraj" ;              // Error
        printf("%s", name);
                }
```

- Array name being a constant can't be presented at the left side of the assignment statement i.e it can't be L-value. You can't re-assign another string to array.

(2)
```
# include<stdio.h>
void main(){
                char name [20] = "Pankaj" ;
                name [1] = 'u' ;
                printf("%s", name);
                }
```
Output: Punkaj

- We are allowed to change content of array and hence we can update any character.

(3)
```
# include<stdio.h>
void main(){
                char * ptr = "Pankaj" ;
                ptr = "Neeraj" ;
                printf("%s", ptr);
                }
```
Output: Neeraj

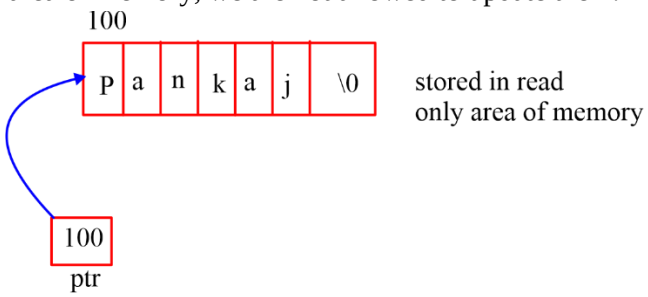


- Pointer ptr being a variable can hold different address at different time. Hence, we can assign another string to a character pointer any time.

(4)
```
# include<stdio.h>
void main()
        {
        char * ptr = "pankaj" ;
        ptr [1] = 'u' ;  // Invalid
        printf("%s", ptr);
        }
```

- String literals are stored in read only memory area i.e "pankaj" is stored first & then the address of character 'p' is given to ptr.

- As they are stored in read only area of memory, we are not allowed to update them.



❐❐❐

6 TYPES OF DATA STRUCTURE, ARRAY & LINKED LIST

6.1 Introduction

6.1.1 Linear data structure

Every element can have almost 2 neighbours. Ex. arrays, linked list, stack, queue.

6.1.2 Non-Linear data structure

Element can have more than 2 neighbours. Ex. Tree, graph.

6.2 Arrays

6.2.1 1-D array

Theoretically index can start from any integer value. Let A be a 1-D array of n elements and the size of each element is w bytes, then the address of A[i] element.

$$\text{Base Address}\,(A) + (i - \text{starting index}) * w$$

6.2.2 2-D Array

Analogous to matrix with rows and columns. Can be implemented in RMO (Row-major order) or CMO (Column-major order).

Let A[M] [N] be a 2-D array, where size of each element is w-bytes, then the address of A[i] [j] is :

(a) In RMO

Address (A[i] [j]) = Base Address (A) + [(j-starting index) + (i – starting index) × M]xw

(b) In CMO

Base Address (A) + [(i – starting index) + (j – starting index) × M] × w

6.2.3 Sparse Matrices

Most of the element of the matrix have 0 value and representing such matrix by a 2D array leads to wastage of memory and that is why, it is better than we will only store non-zero elements (Efficient method).

6.2.4 Lower Triangular Matrix

$n \times n$

$A_{ij} = 0 \quad \text{if} \quad i < j$

$A_{ij} \neq 0 \quad \text{if} \quad i > j$

For ex.

$$A=\begin{bmatrix} A_{11} & 0 & 0 & 0 \\ A_{21} & A_{22} & 0 & 0 \\ A_{31} & A_{32} & A_{33} & 0 \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

6.2.5 Upper triangular Matrix

$$A=\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ 0 & A_{22} & A_{23} & A_{24} \\ 0 & 0 & A_{33} & A_{34} \\ 0 & 0 & 0 & A_{44} \end{bmatrix}$$

6.2.6 Tri-diagonal Matrix

Matrix that has non-zero elements only in the main diagonal, diagonal just below main diagonal and diagonal just above main diagonal. All other elements are zero.

$$\begin{bmatrix} A_{11} & A_{12} & 0 & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 & 0 \\ 0 & A_{32} & A_{33} & A_{34} & 0 \\ 0 & 0 & A_{43} & A_{44} & A_{45} \\ 0 & 0 & 0 & A_{54} & A_{55} \end{bmatrix}$$

6.2.7 Efficient method to store sparse matrix

Using Row-major order for string lower triangular matrix, the element at index (i, j) can be represented as :

$$\text{Index of } A_{ij}=\left[\frac{i(i-1)}{2}+(j-1)\right]$$

Using column-major order for storing upper triangular matrix, the element at index (i, j) is sparse matrix can be represented as:

$$\text{Index of } A_{ij}=(i-j)+\left[(j-1)N-\frac{(j-i)(i-2)}{2}\right]$$

Matrix order

N × N

Using Row-major order for storing upper triangular matrix, the element at index (i, j) in sparse matrix can be represented as :

$$\text{Index of } A_{ij}=(j-i)+\left[(i-1)N-\frac{(i-i)(i-2)}{2}\right]$$

Where N : Number of rows/columns

Using column-major order for starting upper-triangular matrix the element at index (i, j) can be represented as

$$\text{Index of } A_{ij} = \left[(i-1) + \frac{j(j-1)}{2} \right]$$

Using Row-major order for storing tri-diagonal matrix,

Index of element A_{ij} = $2i + j - 3$

Using column-major order for storing tri-diagonal matrix,

Index of element A_{ij} = $i + 2j - 3$

6.3 Linked List

6.3.1 Linked List

Linked list is collection of elements called node, where each node contains atleast 2 fields.

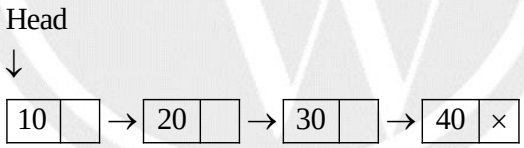
(i) Data field : that contains information.

(ii) Link field : contains address of next note.

6.4 Types of Linked List

6.4.1 Singly Linked List

Every node contains data and a pointer to the next node in the linked list, we can traverse the linked list only in forward direction.



- Head : Pointer that contains address of 1st node.
- Head contains NULL represent empty Linked List.
- If Ptr contains address of some note (Ptr is a pointer to node) then to go to the next node, we need.

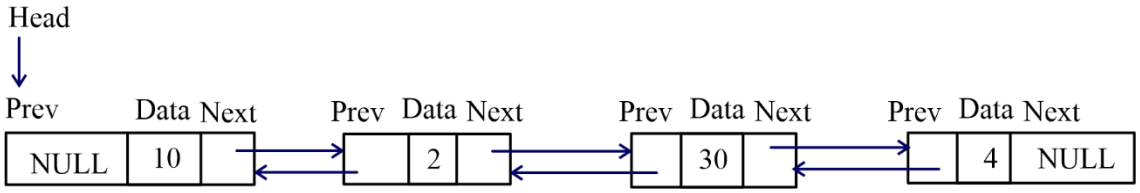
$$\text{Ptr} = \text{Ptr} \rightarrow \text{Link}$$

- Node can be implemented by structure in C

```
Struct Node {
        int data;
Struct Node    * Link;
        }
```

6.4.2 Doubly Linked List

A two-way linked list in which every node contains a pointer to the next node as well as pointer to previous node in the list we can traverse it in backward direction also.



Structure of a Node :

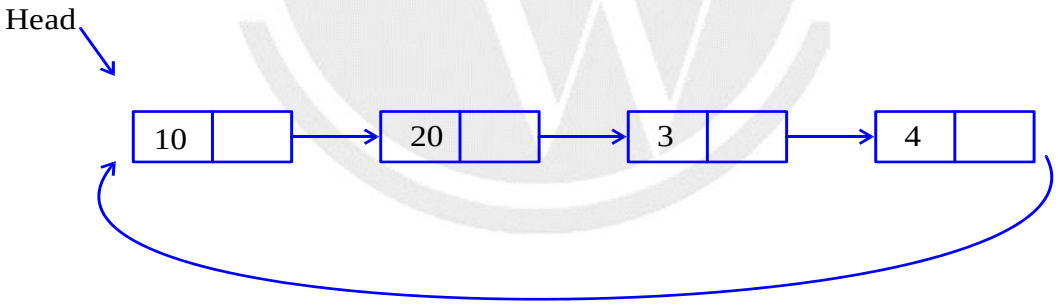
```
Struct Node {
        struct Node * Prev;
        int data;
        struct node * next;

        };
```

6.4.3 Circular Linked List

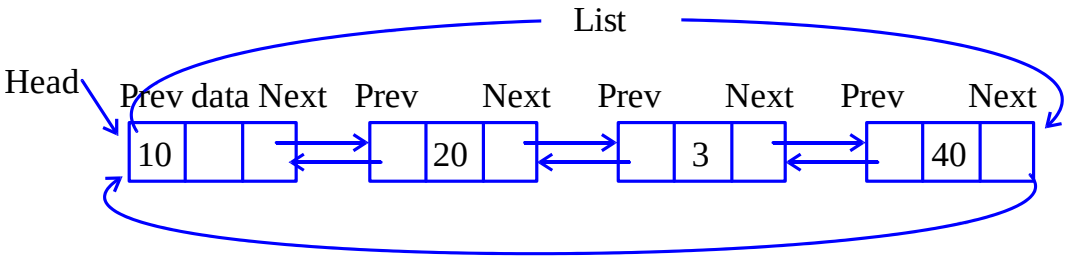
A linked list in which last node points back to the first node in the list.

- Circular linked list has no beginning and no end.



6.4.4 Doubly circular linked list

It is a 2-way circular linked list.

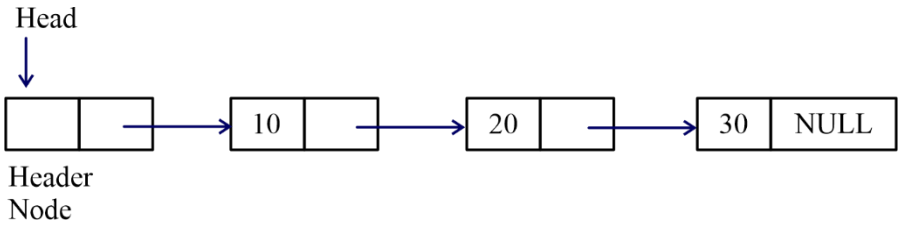


6.4.5 Header Linked List

A linked list that contains a special node called header node at the beginning of the list.

- Head will not point to first node.

- Head points to the header node.



6.4.6 Applications

- used to implement other data structure link stack, queue data structure.
- used for dynamic memory allocation.

6.4.7 Advantages and Disadvantage

- Insertion and deletion are efficient.
- Using pointers in every node, require more memory.
- Random accessing in not possible.
- Circular linked list can be used to implement Fibonacci Heap.

❑❑❑

7 STACK AND QUEUE

7.1 Introduction

Linear data structure in which both insertion and deletion operation are performed at one end called TOP of the stack. Works on LIFO (Last In First Out) Policy.

7.1.1 Application

To post-pone certain decision (To wait/To delay)

- **Stack Permutation :** Order of insertion of key is fixed but we can pop an element any time.

  Number of possible stack permutation with n elements $= \frac{^{2n}C_n}{n+1}$

- Infix to postfix
- Infix to prefix
- Prefix Evaluation
- Postfix Evaluation
- Recursion
- Tower of Hanoi

7.2 Queue Data Structure

Linear data structure in which insertion is done at one end called rear of the queue and deletion is performed at other end called front end of the queue.

Works on FIFO (First In First Out) policy.

7.2.1 Applications of Queue Data Structure

- Whenever a resource is being shared among many users.
- When data is transferred asynchronously.
- FCFS scheduling
- Algorithms like BFS.

7.3 Implementation

Can be implemented using arrays, linked list.

(a) Using Array :

- Easy to implement and memory efficient.
- Not dyanmic

(b) Using Linked List :

- Dynamic
- Require extra memory because of pointer field.

7.4 Standard Operations

7.4.1 Stack

(a) Push ( ) : Insertion of an element onto stack.

(b) Pop ( ) : Deletion of an element.

(c) Is_empty ( ) : Returns true if stack is empty otherwise return false.

7.4.2 Queue

(a) Enqueue : Insertion of an element.

Performed at rear end of the queue.

(b) Dequeue : Deletion of an element.

Performed at front end of the queue.

7.5 Circular Queue

Extended version of simple queue in which last element is connected to the first element.

In simple queue, even though space is available, we are not able to insert a new element and declaring it a overflow. Circular queue solves this problem.

Queue Full

(i) Front = = 0 & & Rear = = SIZE – 1 OR

(ii) Front = = (Rear + 1)% SIZE

7.6 Priority Queue

A queue in which a priority is associated with every element and elements are processed according to their priorities and if two elements have same priority then they are processed as per their arrival in the queue.

❑❑❑

8 TREE DATA STRUCTURE

8.1 Introduction

8.1.1 Terminology

(i) Internal Node : Node with atleast 1 child.

(ii) Leaf Node : Node without any child.

(iii) Root Node : Only node that does not have any parent.

(iv) Complete binary Tree : Binary tree in which nodes are inserted from left to right at every level and we can not insert at node at (K+ 1)th level until all levels upto K are fully filled.

(v) Full Binary Tree : A binary tree in which every internal node has exactly two children and all the leaf nodes are at some level.

Binary tree that contains maximum number of nodes.

(vi) Strict Binary Tree : A binary tree in which every internal node has exactly two children.

(vii) Complete K-ary tree : Tree in which every internal node has exactly K-childs.

8.1.2 Mathematical Result

(i) For a binary tree of height h

- Maximum number of nodes = $2^{(h+1)} - 1$
- Minimum number of nodes = h + 1

(ii) For a complete binary tree of height h

- Maximum number of nodes = $2^{(h+1)} - 1$
- Minimum number of nodes = 2^h

(iii) For a full binary tree :

Number of nodes = $2^{h+1} - 1$

(iv) Total number of unlabelled binary trees with n nodes = $\frac{{}^{2n}C_n}{n+1}$

(v) Total number of labelled binary tree with no keys = $\frac{{}^{2n}C_n}{n+1} \times n!$

(vi) Total number of binary trees with a given preorder (n length) = $\frac{^{2n}C_n}{n+1}$

(vii) Total number of binary trees with a given proorder/postorder/inorder = $\frac{^{2n}C_n}{n+1}$

(viii) Number of binary trees with a given preorder and inorder = 1

(ix) Number of binary trees with a given postorder and inorder = 1

(x) Number of leaf nodes = Number of nodes with two children + 1

8.2 Binary Tree Traversal

(1) Preorder :

(i) Process the root.

(ii) Traverse the left subtree of root in preorder.

(iii) Traverse the right subtree of root in preorder.

(2) Inorder

(i) Traverse the left subtree of root in inorder.

(ii) Process the root.

(iii) Traverse the right subtree of root in inorder.

(3) Post-order

(i) Traverse the left subtree of root in post-order.

(ii) Traverse the right subtree of root in post-order.

(iii) Process the root.

In-order

- **Morrison Traversal :**

  In-place traversal of a binary tree

  No recursion

  Constant extra space.

8.3 Binary Search Tree

A binary tree in which every node satisfy the property that all the keys in the left subtree of the node are smaller than node's key and all the keys in the right subtree of the node are greater than node's key.

- Inorder traversal of a binary search tree is ascending order of keys in the BST.
- Insertion in a BST
  (a) O (log n) best case
  (b) O (n) worst case.

- Deletion from a BST :

  (a) Deletion of leaf node : can be done directly

  (b) Deletion of node with one child : copy the child to the node and delete the child.

  (c) Deletion of a node with two children : first find the inorder successor of the node, copy the contents of this successor to the node and dlete the inorder successor.

  We can also use inorder predecessor.

  - TC : O (log n) best case

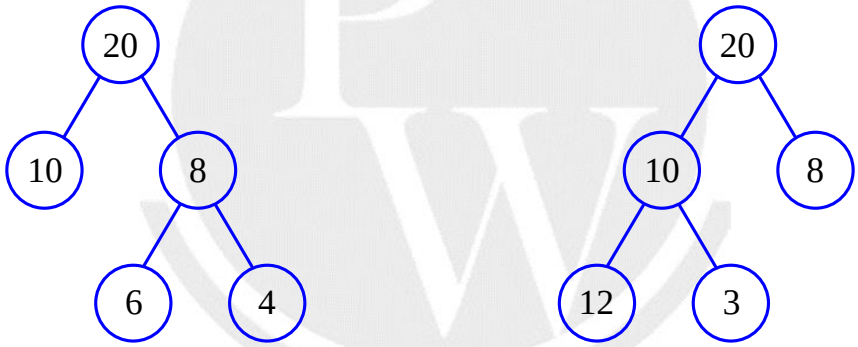
    O (n) worst case

8.4 Heap

A complete binary tree in which every node satisfy heap property (min heap or max heap)

(i) Min heap : A complete binary tree in which every node satisfy the property that the value of the node is smaller than both its children.

(ii) Max heap : A complete binary tree in which every node satisfy the property that the value in the node is greater than both its children.



Not a heap Not a heap

- **Construction of Heap :**

  (i) By inserting keys one after another : O (n log n)

  (ii) Using build-heap method/heapify algo : O(n)

- Number of min-heaps possible with n keys

  $T(n) = T(k) \cdot T(n - k - 1) \cdot {}^{n-1}C_k$

  K : Number of elements in the left subtree of root node.

- Deletion :

  (1) Swap A[1], A[n]

  (2) Apply Heapify on A[1] considering there are only (n – 1) nodes.

  T.C = O (log n)

- Heapsort (Assuming max heap)

  (i) Build Max Heap O(n)

  (ii) Replace root element with last element $\left(A[1] \leftrightarrow A[n]\right)$ reduce the heap size by 1 and then apply heapify at root node.

  (iii) Repeat step 2 while heap size is greater than 1.

8.5 AVL Search Tree

- Height balanced BST.
- Balance factor (B_f) is given by

$$B_f = |h_L - h_R| \leq 1$$

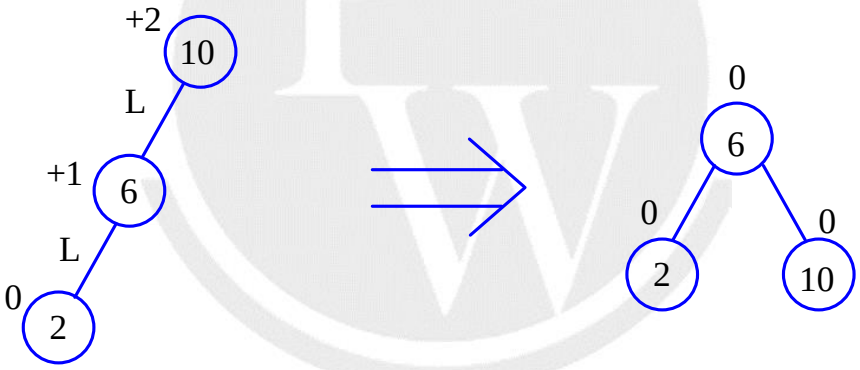
i.e., Balancing factor of a node can be +1, –1 or 0.

8.5.1 Insertion

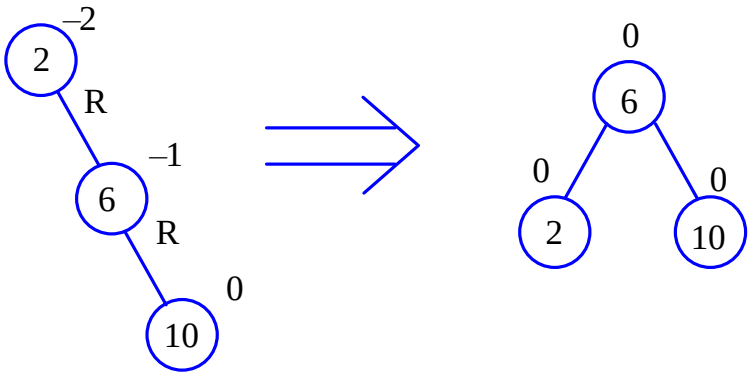
To ensure that the tree remains AVL tree, after insertion some re-balancing is performed i.e., certain rotations are needed.

8.5.2 Types of Rotation

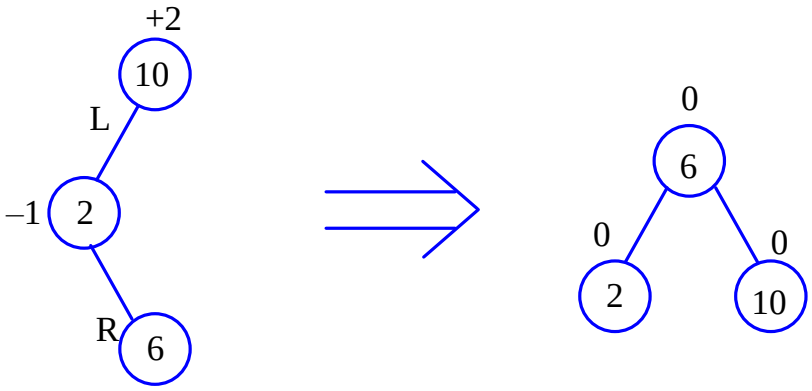
(i) Left-Left (LL) Rotation :



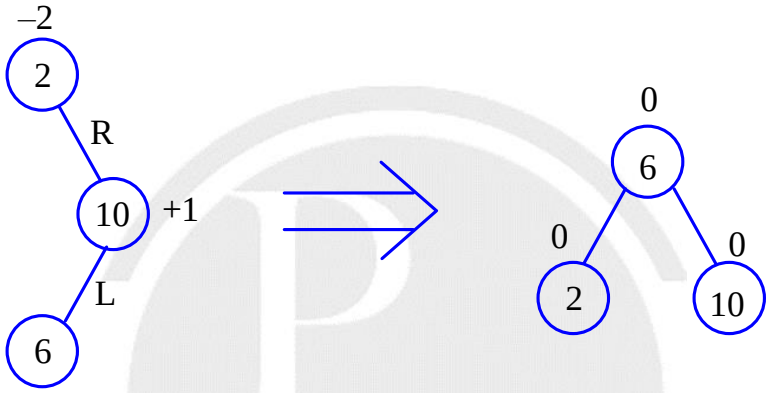
(ii) Right-Right (RR) Rotation :



(iii) Left-Right (LR) Rotation :



(iv) Right-Left (RL) Rotation :



- LL, RR are single rotation.
- LR, RL are double rotation.
- Just work no tri-node structure.
- Minimum number of nodes in an AVL tree of height h is given by recurrence.

  $n(h) = 1 + n(h - 1) + n(h - 2)$ $\quad h \geq 2$

  $n(0) = 1$

  $n(1) = 2$

  i.e., it also forms a Fibonacci series

  1, 2, 4, 7, 12, …….

❑❑❑

09 GRAPH TRAVERSAL

9.1 Introduction

9.1.1 Breadth First Search

- Uses a queue data structure.
- To find whether a graph is connected or not.
- To find number of connected components in a given graph (Breadth First traversal is used)
- To detect cycle in a graph.
- To find whether a given graph is bipartite or not.
- T.C. = O (V + E) using Adjacency List

  = $O(V^2)$ using Adjacency matrix.
- To find shortest path in undirected graph without weights.

9.1.2 Depth First Search

- Uses stack.
- To detect a cycle in a graph.
- To find whether a graph is connected or not.
- To find whether given graph is bipartite or not.
- To find number of connected components in the graph.
- To find bridges in the graph.
- To find articulation points in the graph.
- To find topological sort of a graph.
- To find biconnected components.
- To find strongly connected components.

❑❑❑

10 HASHING

10.1 Introduction

- Searching technique.
- Mapping keys into hash table using a hash function.
- Efficiency based on hash function used for mapping.

10.1.1 Terminology

(i) Collision : When two keys mapped to same location, collision occurs.

$H(k_1) = H(k_2)$, where H(k) is the hash function used.

(ii) Load factor $\lambda = \dfrac{\text{no. of keys}}{\text{Hash table size}}$

10.1.2 Collision resolution technique

(i) Open addressing (closed hashing)

(a) Linear probing
(b) Quadratic probing
(c) Double hashing

$\Big\} \lambda \le 1$

(ii) Separate chaining

10.1.3 Linear Probing

Whenever there is a collision at memory location 'L', then we will search for empty slot sequentially starting from 'L' then L + 1, L + 2, L + 3, …….

10.1.4 Problem with linear Probing

Linear probing suffers with primary clustering problem, consecutive keys forms a cluster and the time to find a free slot increases.

10.1.5 Quatratic Probing

Hash function $h(x) = x$ modm. It leads to a collision and it is i^{th} collision for key x then the collision resolution function is given as:

$H(x, i) = (h(x) + i^2)$ modm

Searching order : L, L + 1, L + 4, L + 9, ……

- Free from primary clustering
- Suffers with secondary clustering problem.

10.1.6 Double Hashing : Using 2 hash functions

(hash – 1 (key) + *i* * hash-2 (key))% m

M : table size

Hash-2 (key) : Secondary hash function must not give output 0.

* One of the best probing.
* Uniform distribution.
* Tree from primary and secondary clustering.
* Computation overhead because of two hash function computation.

10.2 Chaining

Uses the concept of linked list (chain) when more than one element are hashed to same location (slot) then elements are inserted into a singly linked list (chain).

- Deletion is easy compared to open addressing.
- Deleting in open addressing require rehashing remaining keys.

❑❑❑

GATE Exam 2025?

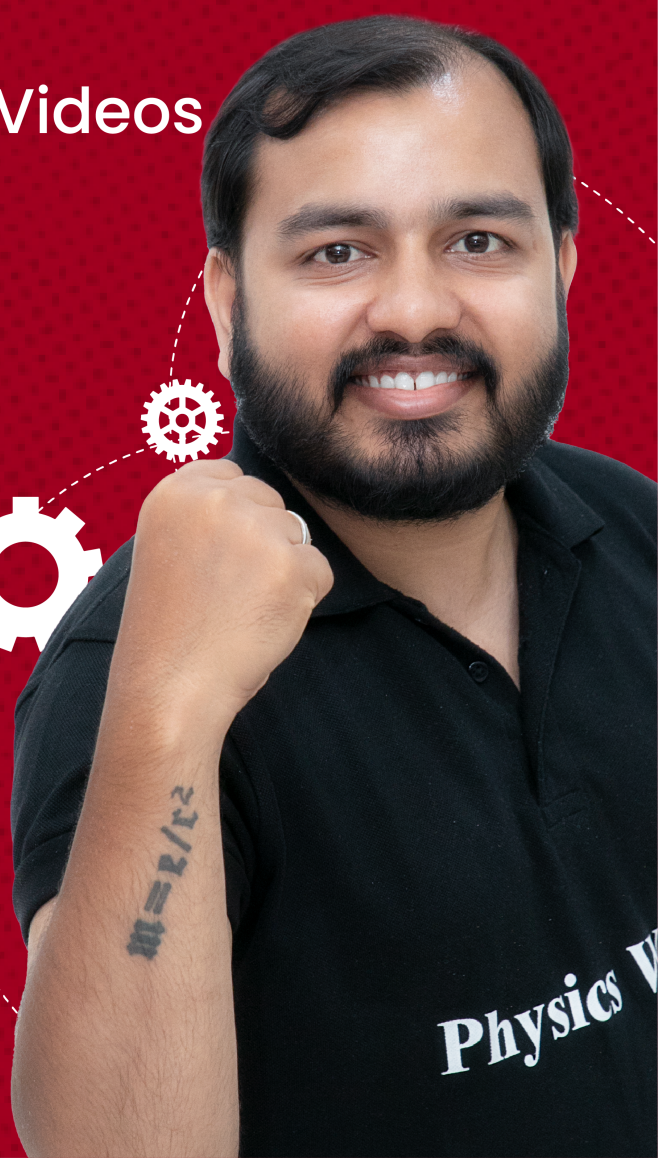
SHRESHTH BATCH

ME | CE | EE | EC | CS & IT

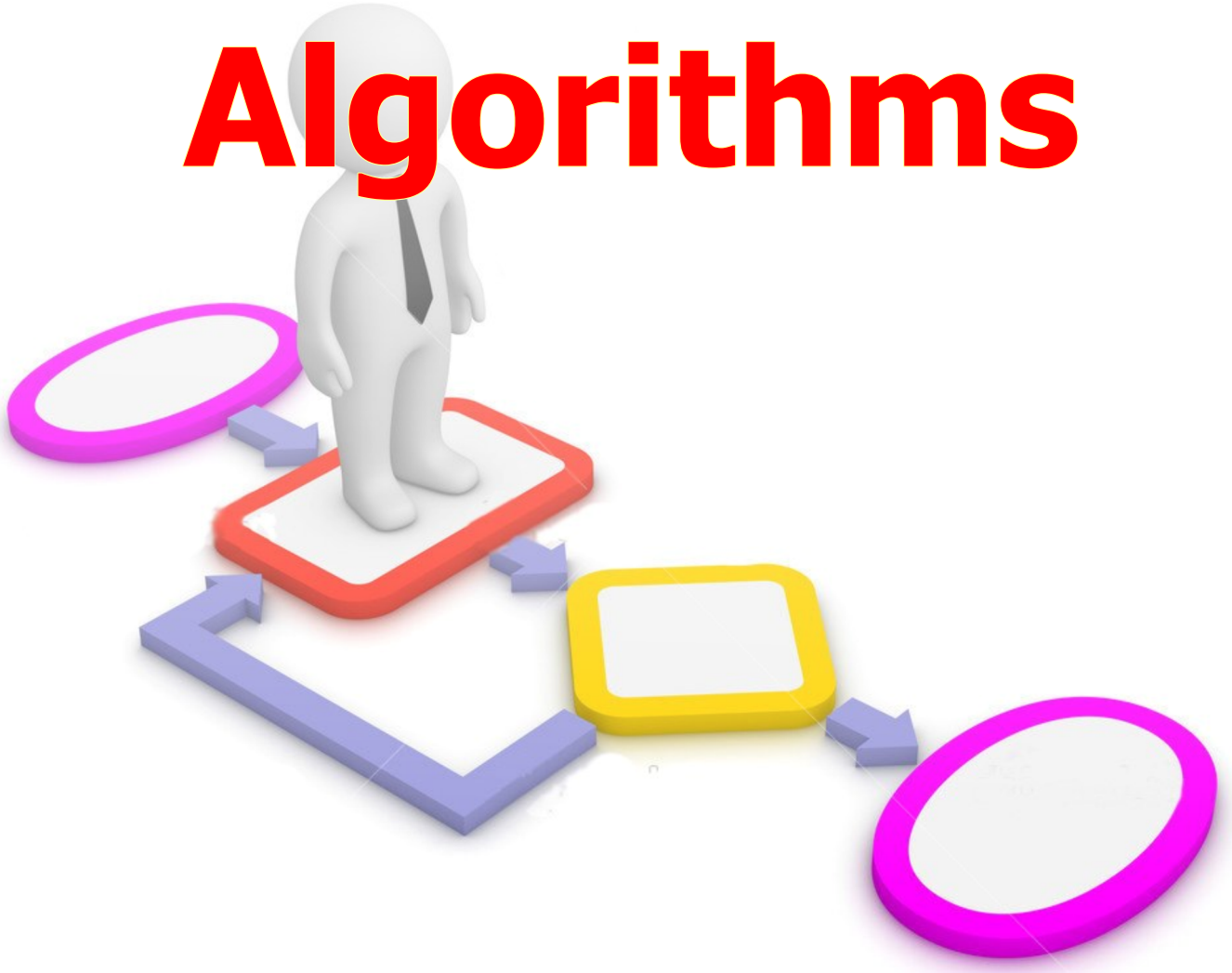
- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in ***Hindi***

Weekday & Weekend Batches Available

JOIN NOW!



Algorithms

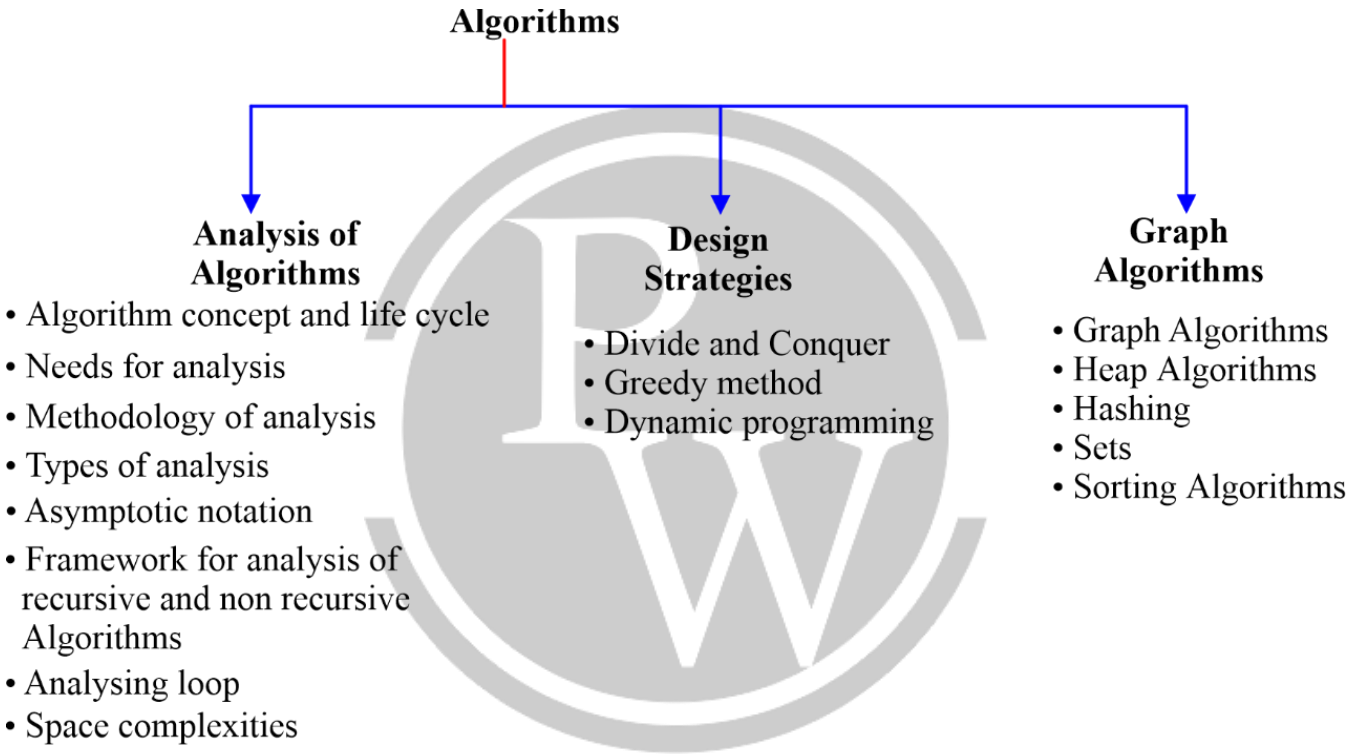


Algorithm

INDEX

1 ASYMPTOTIC NOTATION

1.1 Introduction of Course

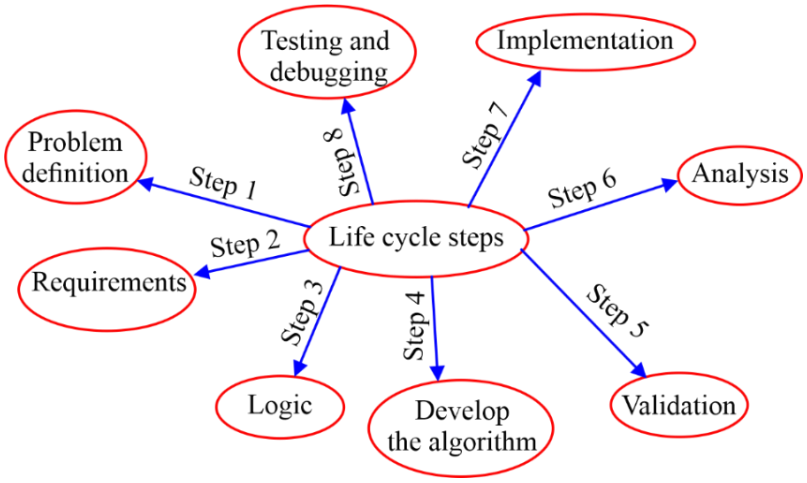


1.2 Algorithm Concept and Life Cycle Steps

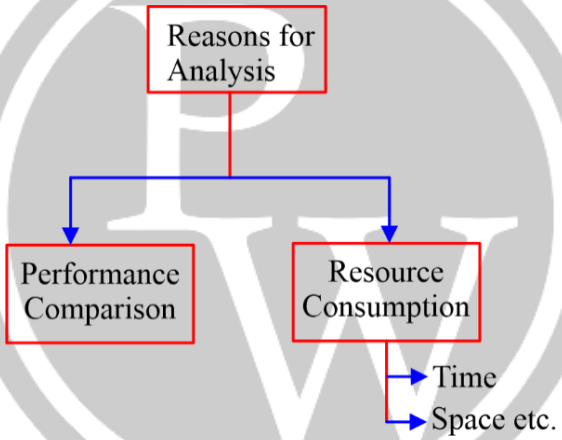
1.2.1 Algorithm

- An Algorithm consists finite number of steps to solve any problem.
- Every step involves some operations and each operation must be definite and effective.

1.2.2 Life Cycle Steps



1.3 Needs of Analysis



In performance comparison comparing different algorithms for optimal solution.

1.3.1 Time Complexity

Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the input size.

1.3.2 Space Complexity

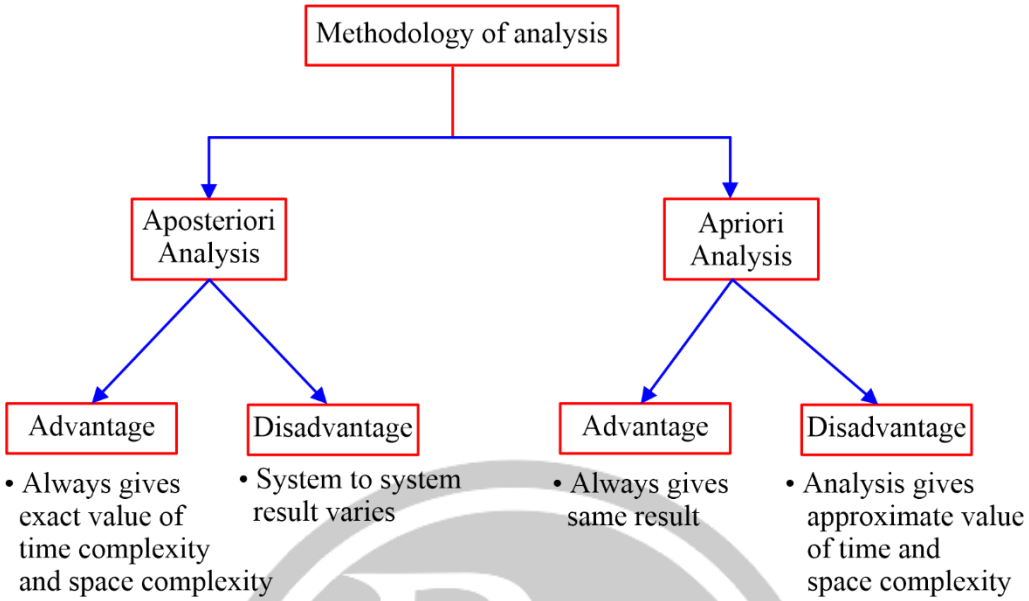
Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of input size.

Note:

To find the time complexity of an algorithm, find the loops and also consider larger loops.

Space complexity is dependent on two things input size and some extra space (stack space link, space list etc).

1.4 Methodology of Analysis



1.5 Types of Analysis

Worst Case

The input class for which the algorithm does maximum work and hence, take maximum time.

Best Case

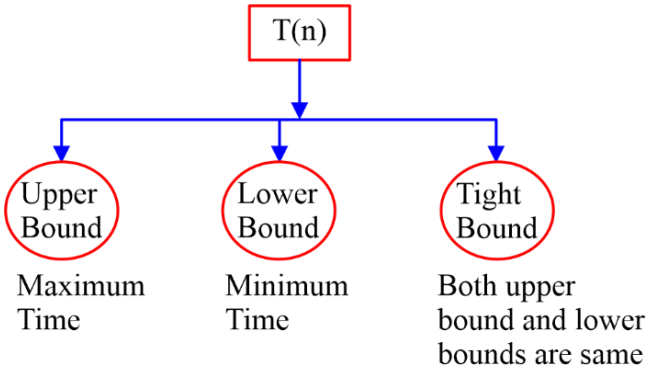
The input class for which the algorithm does minimum work hence, take minimum time.

Average Case

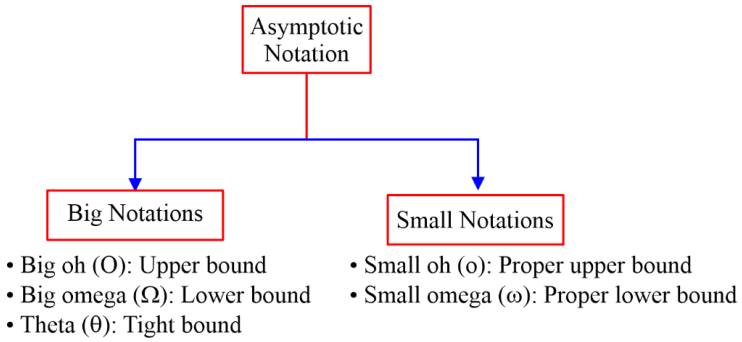
Average case can be calculated form best case to worst case.

1.6 Asymptotic Notations

Suppose, T(n) be a function of time for any algorithm.



1.7 Types of Asymptotic Notations



1.7.1 Big O – Notation

Two Functions $f(n), g(n)$

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

When the growth of $g(n)$ is same or higher than $f(n)$ like $a \le b$

Example:

$f(n) = 3n + 10, g(n) = n^2 + 2n + 5$

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

1.7.2 Ω - Notation

$f(n) = \Omega(g(n))$

$\therefore f(n) \ge C \cdot g(n) \qquad (a \ge b)$

Example: $3^n = \Omega(2^n)$

1.7.3 θ - Notation

If $f(n) \le g(n)$

And

$f(n) \ge g(n)$

$f(n) = g(n)$

$\therefore \quad f(n) = \theta(g(n))$

Example:

$f(n) = 2n^2, g(n) = n+10$

$f(n) > g(n)$ here

so, $f(n) = \Omega(g(n))$ or $g(n) = O(f(n))$

1.7.4. Properties with respect to asymptotic notations

	Reflexive	Symmetric	Transitive	Transpose symmetric
Big oh (O)	✓	×	✓	✓
Big omega (Ω)	✓	×	✓	✓
Theta (θ)	✓	✓	✓	×
Small oh (o)	×	×	✓	✓
Small omega (ω)	×	×	✓	✓

Example 1. Consider the following function

$$f(n) = \sum_{p=1}^{n} p^3 = q$$

Which of the following is/are true for 'q'

(a) $\theta(n^4)$ (b) $\theta(n^5)$ (c) $O(n^5)$ (d) $\Omega(n^3)$

Solution: (a, c, d)

$$f(n) = \sum_{P=1}^{n} P^3$$

$$= 1^3 + 2^3 + 3^3 + 4^3 \ldots\ldots\ldots\ldots + n^3$$

$$= \left(n\left(\frac{n+1}{2}\right)\right)^2$$

$$= O(n^4) \text{ or } \Omega(n^4)$$

$$= \theta(n^4)$$

Example 2. Consider the following functions:

$$f(n) = \sum_{P=1}^{n} P^{1/2} = q$$

Find the value of q in terms of asymptotic notation.

Solution: $f(n) = \sum_{P=1}^{n} P^{1/2}$

$$= 1 + (2)^{1/2} + (3)^{1/2} + \ldots\ldots$$

$$= \frac{2}{3}\left[n^{3/2} - 1\right]$$

$$= \frac{2}{3}n^{3/2} - \frac{2}{3}$$

$$= O(n^{1.5})$$

$$= O(n\sqrt{n})$$

Example 3. Arrange the following functions in increasing order.

$$f_1 = n\log n,\ f_2 = \sqrt{n},\ f_3 = 2^n,\ f_4 = 3^n,\ f_5 = n!,\ f_6 = n^n,\ f_7 = \sqrt{\log n},\ f_8 = 100n\log n$$

$$\rightarrow f_7 < f_2 < fl = f_8 < f_3 < f_4 < f_5 < f_6$$

Example 4. Arrange the following functions in increasing order.

$f_1 = 10,\ f_2 = \sqrt{n},\ f_3 = \log\log n,\ f_4 = (\log n)^2,\ f_5 = n^2$

$f_6 = n\log n,\ f_7 = n!,\ f_8 = 2^n,\ f_9 = n^n,\ f_{10} = n^2\log n$

$\rightarrow\ f_1 < f_3 < f_4 < f_2 < f_6 < f_5 < f_{10} < f_8 < f_7 < f_9$

Example 5. Arrange the following functions in increasing order.

$f_1 = \log\log n$ $\quad f_9 = n\log\log n$

$f_2 = \log n$ $\quad f_{10} = n^2\log n$

$f_3 = (\log n)^2$ $\quad f_{11} = n^3$

$f_4 = \sqrt{\log n}$ $\quad f_{12} = 2^n$

$f_5 = n^{1/10}$ $\quad f_{13} = e^n$

$f_6 = n$ $\quad f_{14} = n!$

$f_7 = n^2$ $\quad f_{15} = n^n$

$f_8 = n\log n$ $\quad f_{16} = n^{3/2}$

$f_1 < f_4 < f_2 < f_3 < f_5 < f_6 < f_9 < f_8 < f_{16} < f_7 < f_{10} < f_{11} < f_{12} < f_{13} < f_{14} < f_{15}$

$a^{\log_b c} \Leftrightarrow c^{\log_b a}$

$\therefore 2^{\log_2 n} \Leftrightarrow n^{\log_2 2} = n$

Example 6. Arrange the following functions in increasing order.

$f_1 = n!,\ f_2 = n^n$

$f_1 = n \times (n-1)(n-2) \times \ldots \times 3 \times 2 \times 1$

$f_2 = n \times n \times n \times n \times \ldots \times n \times n \times n$

$f_2 > f_1$

$\therefore f_1 = O(f_2)$

$\boxed{2^n < 3^n < 4^n < n! < n^n}$

Question.

Which of following is TRUE?

(1) $2^{\log_2 n} = O(n^2)$ TRUE

(2) $n^2 \cdot 2^{3\log_2 n} = O(n^5)$ TRUE

(3) $2^n = O(2^{2n})$ TRUE

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

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

Solution:

(1) $2^{\log_2 n} = O(n^2)$

$= n^{\log_2 2}$

$= n$

$= n = O(n^2)$

(2) $n^2 \cdot 2^{3\log_2 n} = O(n^5)$

$= n^2 \cdot n^{3\log_2 2}$

$= n^2 \cdot n^3$

$= n^5$

$= n^5 = O(n^5)$

(3) $2^n = 2^{2n}$

$2^n = 2^n.2^n$

$2^n \le 2^{2n}$

$2^n = O(2^{2n})$ True

(4) $\log n > \log\log n$

$\log n \neq O(\log n)$

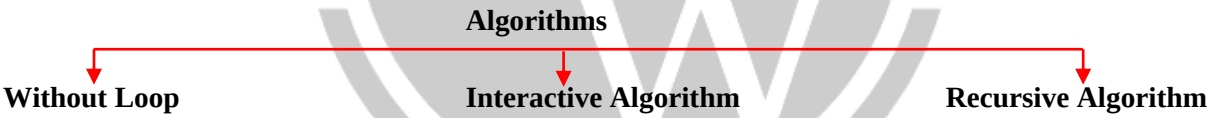
False

(5) $\log \log n \le n \log n$

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

True

1.8. Analysis of an Algorithm



1.8.1 Without loop

Example:

```
int fun (in + n)
{
    return n*(n+1)/2;
}
```

Solution.

Here 1 multiply, 1 division, 1 addition

∴ O (1) [no loops, no recursion]

1.8.2. Iterative Algorithm Analysis

Example 1:

```
for (i =1; i ≤ n; i=i*2)
printf("Sushil")
```

Solution.

$i=1, 2, 2^2, 2^3 ..., 2^k$

$\rightarrow$

$2k \le n$

$k \log 2 \le \log n$

$k \le \frac{\log n}{\log 2}$

$\therefore k \le \log_2 n$

$k = \lfloor \log_2 n \rfloor$

So, this will execute $\lfloor \log_2 n \rfloor + 1$ time and Complexity O ($\log_2 n$)

Example 2:

```
For (i=1; i ≤ n; i=i*3)
printf("Aaveg");
```

Solution.

So, this will execute $\lfloor \log_3 n \rfloor + 1$ time and complexity O ($\log_3 n$)

- $i = 1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow ... \rightarrow n$
  $i = n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow ... 1$

Example 3:

```
for (i = 1; i ≤ n; i++)
    {
        for (j=1; j ≤ 10; j++)
            {
                printf("Dhananjay");
            }
        }
```

Solution.

This will execute $10 \cdot n$ times and complexity O(n)

Example 4:

```
for (i = 1; i <= n; i = i*3)
    for (j = 1; j ≤ n; j++)
        printf("Prapti");
```

Solution.

Total $n\left(\lfloor \log_3 n \rfloor + 1\right)$ time execute and Complexity = O ($n \log_3 n$)

1.8.3. Recursive Algorithm Analysis

Example 1:

```
void fun (in + n)        T(n)
{
    if (n > 0)           1 compare; C1 time
    {
        if ("% d", n);   ←  C2 time
        fun (n - 1);     ←  T(n-1)
    }
}
```

Let T(*n*) be the Complexity time taken by algo for n size i/p

Solution.

$T(n) = C_1+C_2+T(n-1)$

$T(n) = T(n-1) + C$ $\quad n > 0$

$T(0) = C$ ← Constant

$T(n) = C;$ $\quad n = 0$

$T(n) = T(n - 1) + C;$ $\quad n > 0$

Example 2:

```
    void fun (in + n)        T (n)
{
    if (n > 0)   ← C1 time
    {
        for (i = 1; i <= n; i + 1)  ← n time
        printf("Hello");
        fun (n - 1);   ← T (n - 1)
    }
}
```

Solution.

$T(n) = C_1 + n - 1 + T(n - 1)$

$= T(n - 1) + n$ $\quad n > 0$

$T(0) = C$ $\quad n = 0$

Example 3:

```
void fun (in + n)        T(n)
{
  if (n > 0)   ← C1
    {
      for (i = 1; i < = n; i = i*2)  ← ⌊log2 n⌋
```

```
        printf("Divyajyoti");
    fun (n - 1);  <-------  T (n - 1)
    }
}
```

Solution. $T(n) = T(n-1) + O(\log_2 n); \quad n > 0$

or

$T(n) = T(n-1) + \log_2 n$

$T(0) = C$ } or $n = 0$

$T(0) = O(1)$

1.9 Solving Recurrence Relation

1.9.1 Substitution Method

Example: (1)

$$\mathbf{T(n) = T(n-1) + C}$$
$$\mathbf{T(1) = C}$$

n size on problem $n-1$ size x_1 convert them

$T(n) = T(n-1) + C$

$\downarrow$

$[T(n-2) + C] + C$

$T(n) = T(n-2) + 2C$

$\downarrow$

$= T(n-3) + 3C$

$T(n) = T(n-k) + kC$

$\therefore n - k = 1$

$T(n) = T(1) + (n-1)C$

$= C + (n-1)C$

$T(n) = O(n)$

Example (2)

$$\mathbf{T(n) = T(n-1) + C \cdot n}$$
$$\mathbf{T(1) = C}$$

Solution.

$\therefore T(n) = T(n-1) + C \cdot n$

$= [T(n-2) + C \cdot (n-1)] + C \cdot n$

$= [T(n-3) + C \cdot (n-2)] + C(n-1) + C \cdot n$

$= T(n-3) + (n-2) \cdot C + (n-1) \cdot C + n \cdot C$

$= T(n-k) + C(n-k+1) + C(n-k+2) + \ldots + C(n-k+k)$

$\therefore n - k = 1$

$T(n) = T(1) + T(2) + C(3) + C(4) + \ldots + C(n-1) + C(n)$

$= C + C(2) + (3)C + 4(C) + \ldots + (n-1)C + (n) \cdot C$

$= C[1 + 2 + 3 + \ldots + n]$

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

$= O(n^2)$

Example (3)

$T(n) = T(n/2) + C$

$T(1) = 1$

Solution.

$T(n) = T(n/2) + C$

$= [T(n/2^2) + C] + C$

$= T(n/4) + 2C$

$= T(n/2^3) + 3C$

$T(n) = T(n/2^k) + kC$

$= (n/2^k) = 2$

$T(n) = T(2) + (\log_2 n - 1) C$

$= 1 + (\log_2 n - 1) C$

$= O(\log n)$

Example (4)

$T(1) = 1$

$T(n) = 2T(n/2) + C$

Solution. $T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + C\right] + C$

$= 2^2 T\left(\frac{n}{2^2}\right) + 2^2 C + C$

$= 2^2\left[2T\left(\frac{n}{2^3}\right) + C\right] + 2C + C$

$= 2^3 T\left(\frac{n}{2^3}\right) + 2^2 C + 2C + C$

$= 2^k T\left(\frac{n}{2^k}\right) + 2^{k-1} C + 2^{k-2} C + \ldots + 2^1 \cdot C + C$

$\frac{n}{2^k} = 1 \quad \therefore n = 2^k$

$\rightarrow T(n) = nT(1) + 2^{k-1} \cdot C + 2^{k-2} \cdot C + \ldots + 2C + C$

$= 2^k + 2^{k-1} \cdot C + 2^{k-2} + \ldots + 2C + C$

$= 2^k + C(2^{k-1} + 2^{k-2} + \ldots + 2^1 + 2^0)$

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

$= 2^k + C(2^k - 1)$

$= 2^k + 2^k \cdot C - C$

$= n \cdot C$

$= O(n)$

1.9.2 Master's Method

$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k (\log n)^p)$

$a \geq 1, b > 1, k \geq 0$, p = real number

If $a > b^k$ or $\log_b a > k$

$T(n) = \Theta\left(n^{\log_b a}\right)$

Question 1. $\mathbf{T(n) = 2T\left(\frac{n}{2}\right) + (n)^0 \log n}$

Solution. $a = 2, b = 2, k = 0$

$a > b^k; 2 > 2^0; 2 > 1$

$T(n) = \Theta(n)$

Question 2. $\mathbf{T(n) = 2T\left(\frac{n}{2}\right) + n}$

Solution. $a = 2, b = 2, k = 1, p = 0$

$T(n) = \Theta(n \cdot \log n)$

Question 3. $\mathbf{T(n) = 2T\left(\frac{n}{2}\right) + n \log n}$

Solution. $a = 2, b = 2, k = 1, p = 1$

$\therefore T(n) = \Theta(n (\log n)^2)$

(b) If $p < 0$ then $T(n)$

$T(n) = O(n^k)$

Question 4. $T(n) = T\left(\frac{n}{2}\right) + C$

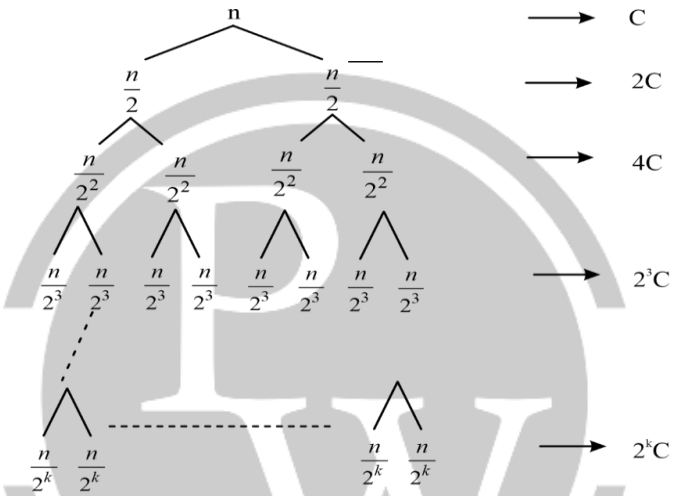
Solution.

$T(n) = \Theta(n^2 \log n)$

1.9.3. Recursive Tree

(1) $T(n) = 2T\left(\frac{n}{2}\right) + C$

$T(1) = C$



$\frac{n}{2^k} = 1 \rightarrow n = 2^k \rightarrow k = \log_2 n$

Total Work done $= C + 2C + 2^2C + 2^3C + \ldots + 2^kC$

$= C(1 + 2 + 2^2 + \ldots + 2^k)$

$= c\left(\frac{2^{k+1} - 1}{2 - 1}\right)$

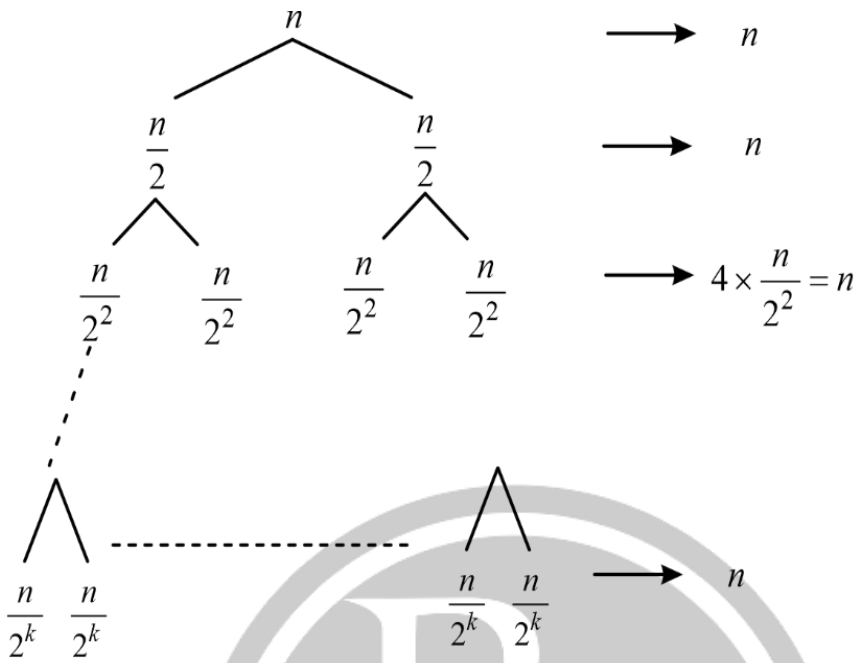
$= C(2^{k+1} - 1$

$= C(2 2^k - 1)$

$= C(2n - 1)$

$= O(n)$

(2) $T(n)=2T\left(\frac{n}{2}\right)+n$



$\frac{n}{2^k}-1; n-2^k; k-\log_2 n$

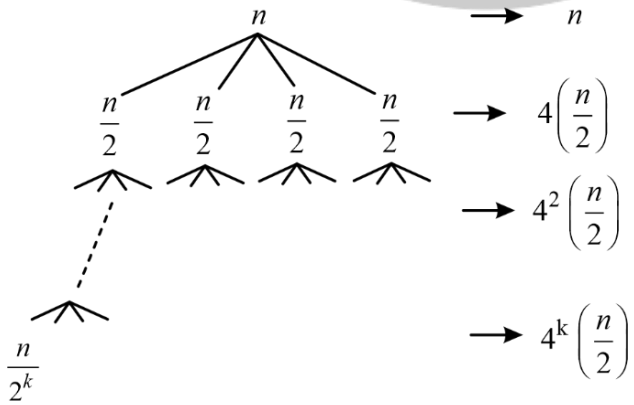
$\therefore n + n + n + \ldots + n$

$= \text{k } n$

$= \text{n } n\log_2 n$

$= O(n\log_2 n)$

(3) $T(n)=4T\left(\frac{n}{2}\right)+n$



$n = 2^k$, $k = \log_2 n$

$= n+4\left(\frac{n}{2}\right)+4^2\left(\frac{n}{2}\right)+\ldots+4^k\left(\frac{n}{2}\right)$

$$= n\left[1+2+2^2+2^3+\ldots+2^k\right]$$

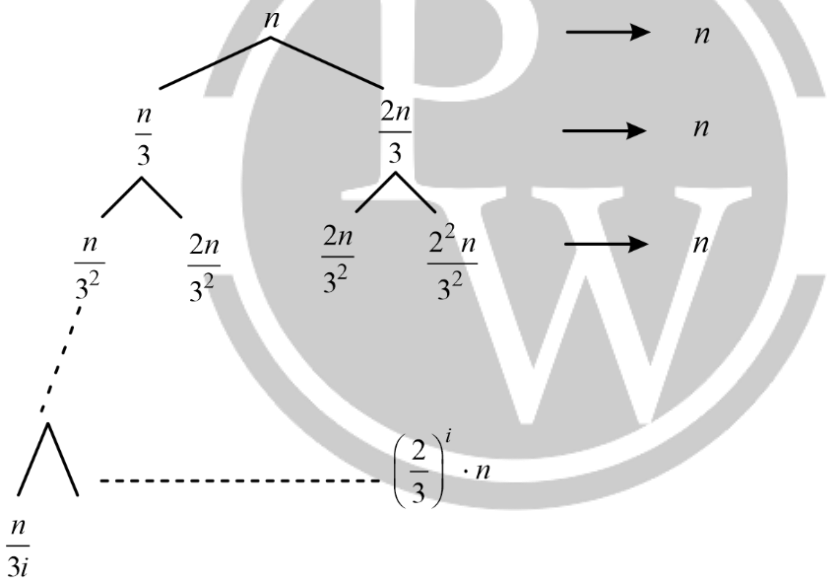
$$= n\left(\frac{2^{k+1}-1}{2-1}\right)$$

$$= n\left(2\cdot 2^{k-1}\right)$$

$$= n(2n)-1$$

$$= O\left(n^2\right)$$

(4) $T(n)=T\left(\frac{n}{3}\right)+T\left(\frac{2n}{3}\right)+n$ T(1) = 1



$\frac{n}{3i}-1;\ n-3i;\ i-\log_3 n$ [3]

$= n + n + \ldots + \log_3 n$ T (n)

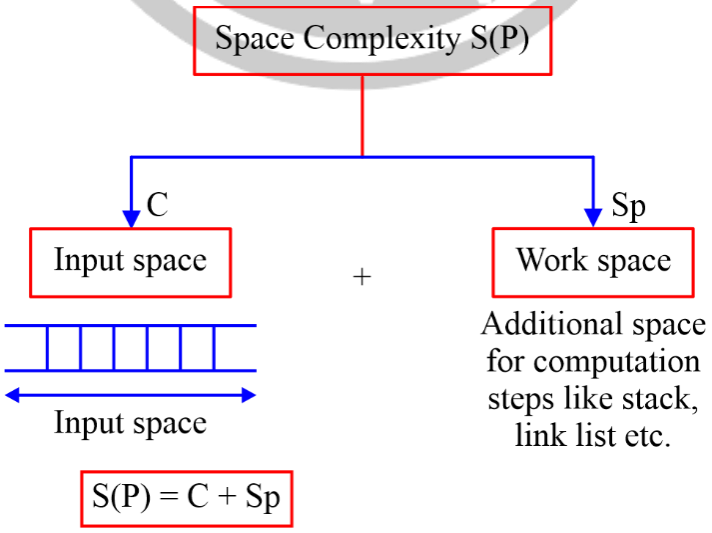
$= (n + n + n + \ldots + \log_3 n) \geq n + \ldots + \log_3 n$

$\Omega(n \log_{3/2} n)$

1.10 Recurrence Relations and their Time Complexity

$T(n) = C; n = 2$ $T(n) = 2\,T(\sqrt{n}) + C; n > 2$	O (logn)
$T(n) = C; n = 2$ $T(n) = T(n-1) + C; n > 2$	$O(n)$
$T(n) = C; n = 1$ $T(n) = T(n-1) + n + C; n > 2$	$O(n^2)$
$T(n) = C; n = 1$ $T(n) = 2T(n-1) + C; n > 1$	$O(2^n)$
$T(n) = C; n = 1$ $T(n) = 2T\left(\frac{n}{2}\right) + C; n > 1$	$\theta(n)$
$T(n) = C; n = 1$ $T(n) = 2T\left(\frac{n}{2}\right) + n; n > 1$	$\theta(n\log n)$
$T(n) = C; n = 1$ $T(n) = T\left(\frac{n}{2}\right) + C; n > 1$	$\theta(\log n)$
$T(n) = 1; n = 2$ $T(n) = T\left(\sqrt{n}\right) + C; n > 2$	$\theta(\log\log n)$

1.11 Space Complexities



```
Int n, A[n];
Algorithm Rsum(A, n)
{
  if (n = 1) return (A(1));
  else;
  return (A[n] + RSum(A, (n–1));
 }
```

- **Time Complexity** = O(n)
- **Space Complexity**
- We need stack space
- Stack is used to store activation records of function calls
- Size of activation records is trivial
- Stack size that we need = O(n)
- Space complexity = O(n)

```
Algorithm A(n)
{
   if (n = 1) return;
   else;
 {
   A(n/2);
  }
}
```

Recurrence relation

$$T(n) = C;\ n = 1$$

$$T(n) = T\left(\frac{n}{2}\right) + C;\ n > 1$$

Time Complexity = O(log n)

Space Complexity

- Space complexity will depend on number of activation record pushed into the stack

Suppose, n = 16

A (1)
A (2)
A (4)
A (8)
A (16)

For $n = 2^K$ we are pushing the 'K' activation record

∴ **Space Complexity**

$$n = 2^K$$
$$\log n = K\log_2 2$$
$$K = \log_2 n$$
$$\boxed{\text{Space Complexity} = O(\log n)}$$

Example 3

```
Algorithm A(n)
{
  if (n = 2) return;
  else;
  return (A √n );
}
```

Solution:

$$T(n) = 1;\ n = 2$$
$$T(n) = T\left(\sqrt{n}\right) + C;\ n > 2$$

Time Complexity = O (loglogn)

Space Complexity

Suppose n = 16

A(1)
A(2)
A(4)
A(16)

∴ For $2^{n/2^k}$ manner we are pushing in stack

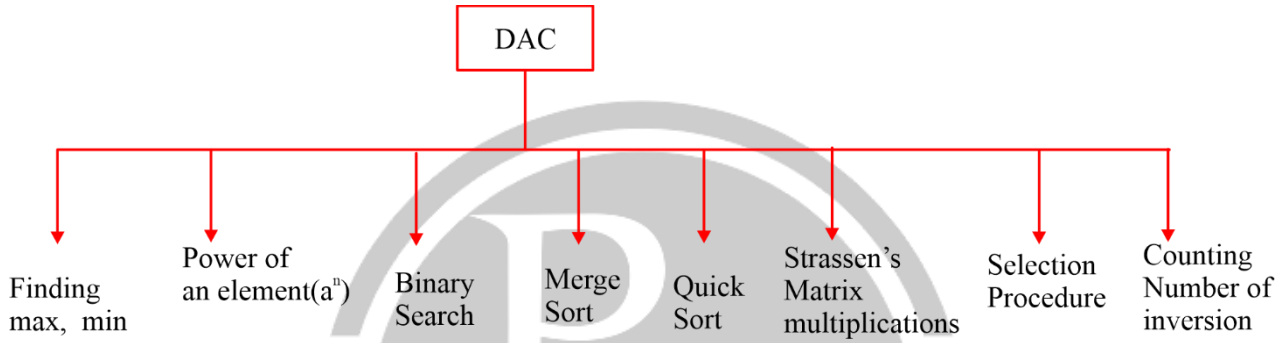
$$2^{n/2^k} \geq 2$$
$$\frac{n}{2^k}\log_2 2 \geq \log_2 2$$
$$n \geq 2^K$$
$$K \leq \log_2 n$$

$$\boxed{\text{Space complexity} = O\left(\log_2 n\right)}$$

❑❑❑

2 DIVIDE AND CONQUER

2.1 DAC Application



2.2 Finding Maximum Minimum element

Recurrence Relation:

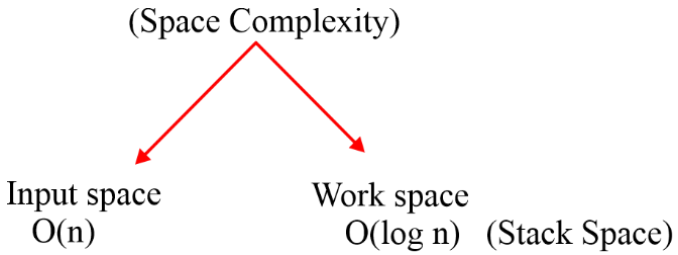
$$T(n)=\begin{cases}1 \text{ if } n=1 \text{ or } n=2\\ 2T\left(\frac{n}{2}\right)+1;\ n>2\end{cases}$$

Time Complexity:

$$T(n)=O(n)$$

- Time complexity is same for every case (Best case/Worst case).

Space Complexity:



$$\text{Space Complexity} = O(n) + O(\log n)$$
$$= O(n)$$

Number of comparisons to find maximum / minimum element on an given array of n elements:

$$\text{Comparison} = \frac{3n}{2} - 2$$

2.3 Power of an Element

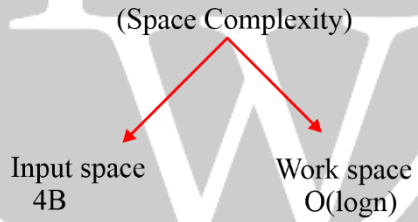
Recurrence relation:

$$T(n) = \begin{cases} 1 \text{ if } n = 1 \\ T\left(\frac{n}{2}\right) + 1;\ n > 1 \end{cases}$$

Time Complexity:

$$T(n) = T\left(\frac{n}{2}\right) + C$$
$$T(n) = O(\log n)$$

Space Complexity:



Space Complexity = 4B + O(logn)

= O(logn)

Number of multiplications to find a^n

Multiplication = O(logn)

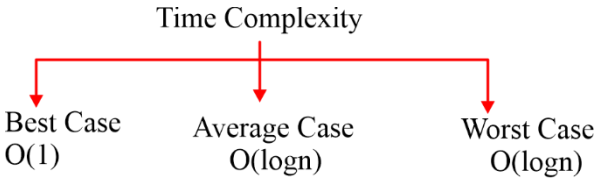
2.4 Binary Search

Given a sorted array and an element x, need to return the index of element x if it is present then 1, otherwise – 1.

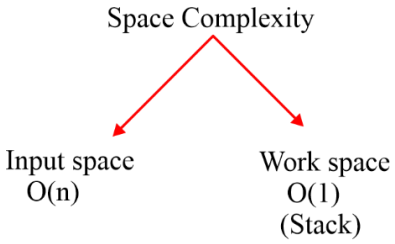
Recurrence relation:

$$T(n) = \begin{cases} 1; & n = 1 \\ T\left(\frac{n}{2}\right) + C;\ n > 1 \end{cases}$$

Time Complexity:



Space Complexity:



Space complexity = O(n) + O(1)

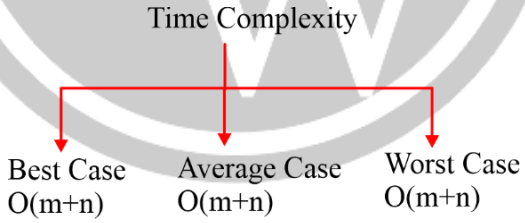
= O (n)

2.5 Merge Algorithm

- Merging two sorted sub arrays of input size m,n.
- Number of comparisons to merge two sorted sub arrays of size m,n.

Comparisons = m + n – 1 (worst case)

Number of moves =m + n (Outplace Algorithm)



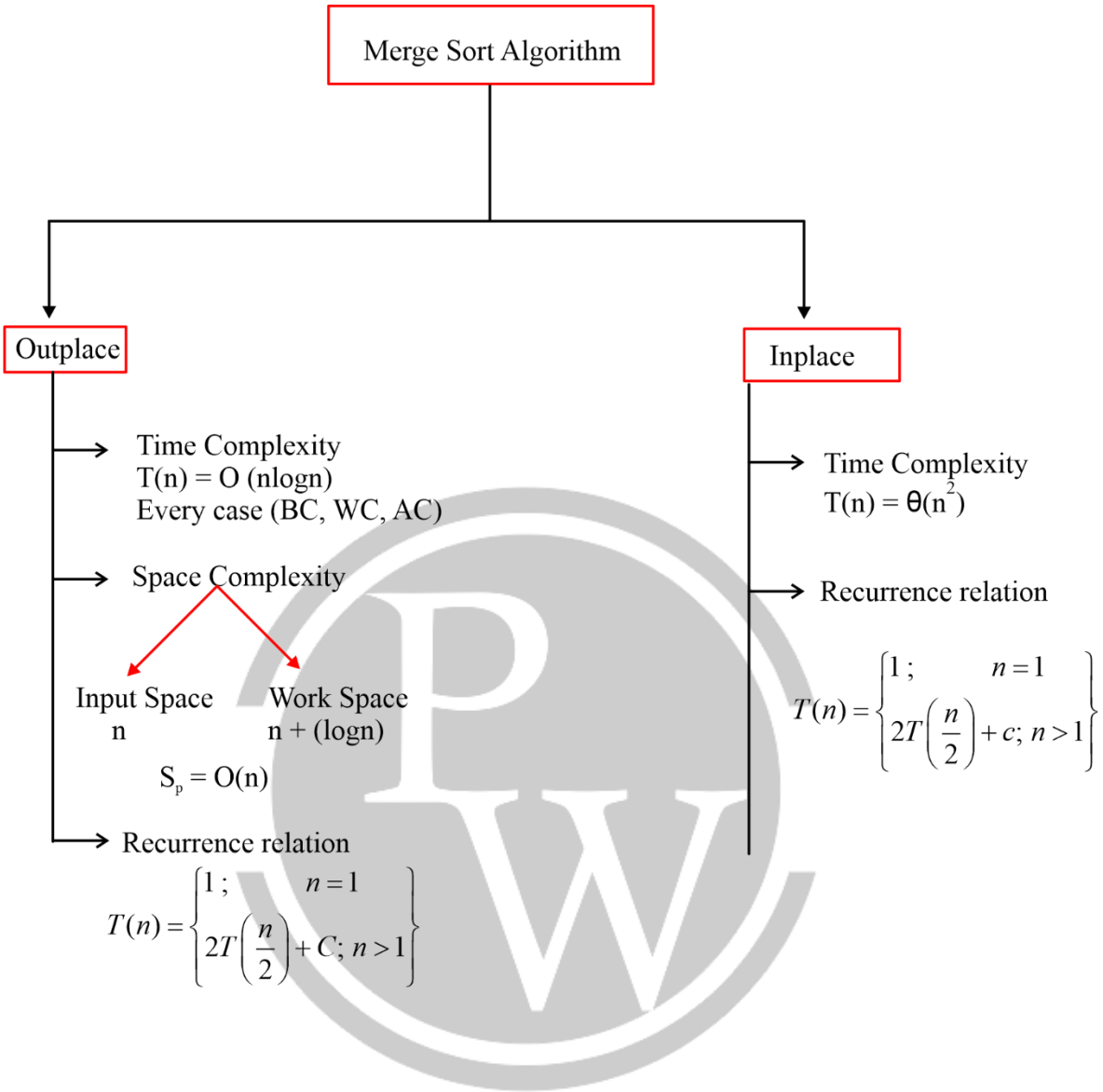
Number of comparisons in best case of merging two sorted subarrays of size m, n.

comparisons = min (m, n)

Moves = m + n (Always)

Note:
Best Case comes in comparisons no effect on moves.

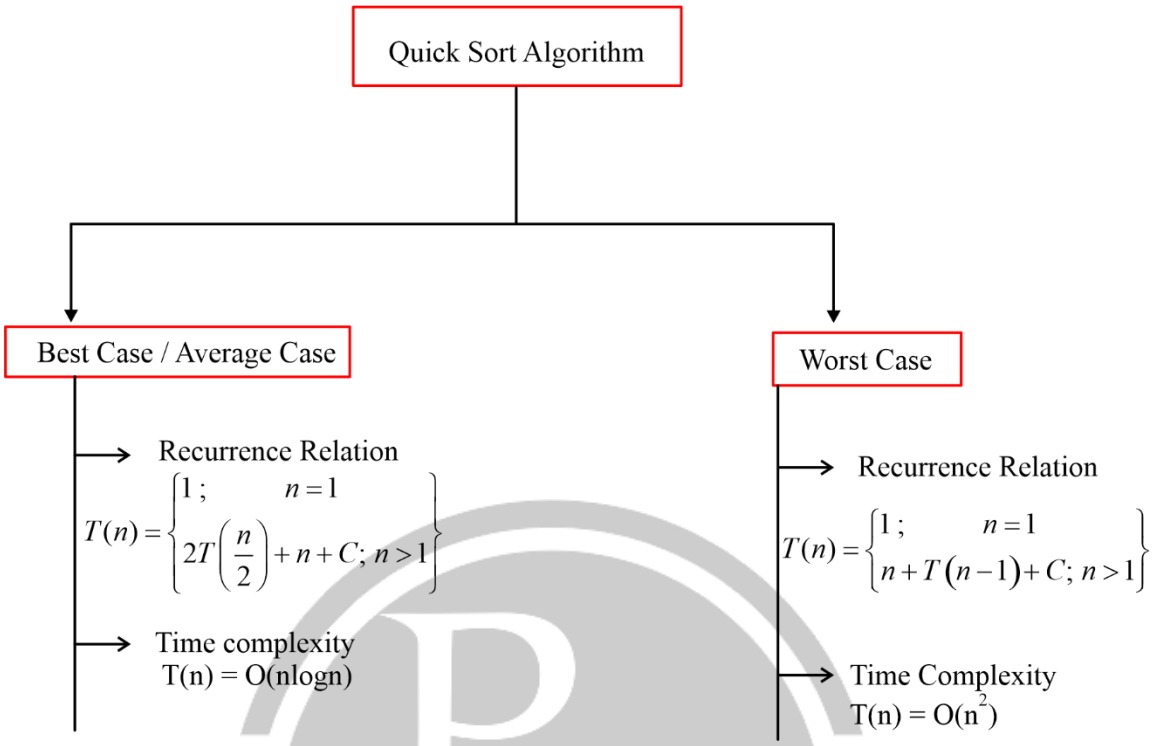
2.5.1 Merge Sort Algorithm:



Note:

- In GATE exam if merge sort given then always consider outplace.
- If array size very large, merge sort preferable.
- If array size very small, then prefer insertion sort.
- Merge sort is stable sorting technique.

2.6 Quick Sort Algorithm



Example 1: In Quick for sorting n elements, the $\left(\frac{n}{16}\right)^{th}$ smallest element is selected as pivot. what is the worst-case time Complexity?

Solution.

$$T(n) = T\left(\frac{n}{16}\right) + T\left(\frac{15n}{16}\right) + O(n)$$

= (solve by recursive tree method)

Example 2: The median of *n* elements can be found in O (n) time then, what is the time complexity of quick sort algo in which median selected as pivot?

Solution.

$T(n) = O(n) + C + O(n) + T(n/2) + T(n/2)$

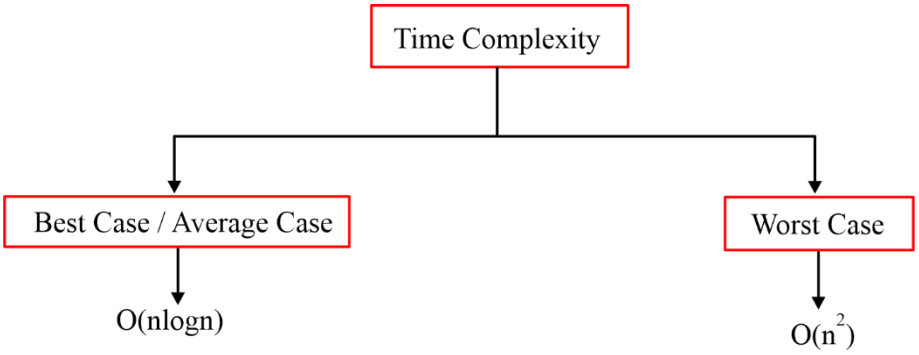
Find median — swap median with last — Partition algo

$= 2T(n/2) + C \cdot n$

$= O(n \log n)$

2.6.1 Randomized Quick Sort

- In Randomized quick short algorithm selection of pivot element can be taken randomly.



2.7 Counting Number of Inversion

- Counting number of inversion on given an array of an element.

  Time complexity T(n) = O(nlogn)

2.8 Selection Procedure

Find K^{th} smallest on given an array of an element and integer K.

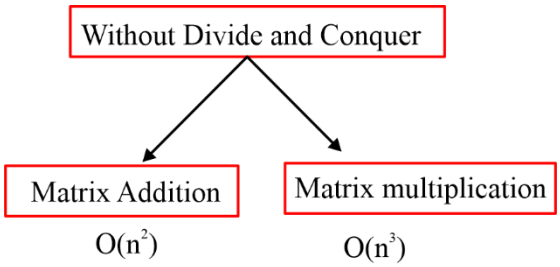
Time Complexity:

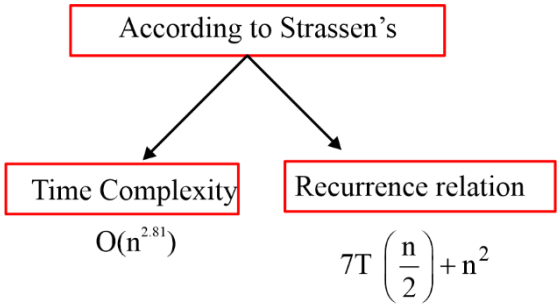
$T(n) = O(n^2)$

Space complexity:

Space Complexity = O(n)

2.9 Strassen's matrix Multiplication





2.10 Comparison Based Sorting Algorithms

Sorting Algorithm	Basic logic of sorting Algo	BC	AC	WC	Stable sorting	Inplace sorting
Quick sort	Choose pivot element place in correct position	$\theta(n\log n)$	$\theta(n\log n)$	$\theta(n^2)$	No	Yes
Merge sort	Divide to equal parts recursively sort each sub part & marge them	$\theta(n\log n)$	$\theta(n\log n) = n \log n$	$\theta(n\log n) = n \log n$	Yes	No
Heap sort	Build heap(max) delete max place	$\theta(n\log n)$	$\theta(n\log n)$	$\theta(n\log n)$	No	Yes
Bubble sort	Compare exchange	$\theta(n)$	$\theta(n^2)$	$\theta(n^2)$	Yes	Yes
Selection sort	Find position of min element from [1 to n]	$\theta(n^2)$	$\theta(n^2)$	$\theta(n^2)$	No	Yes
Insertion sort	Insert a [i + 1] into correct position	$\theta(n)$	$\theta(n^2)$	$\theta(n^2)$	Yes	Yes

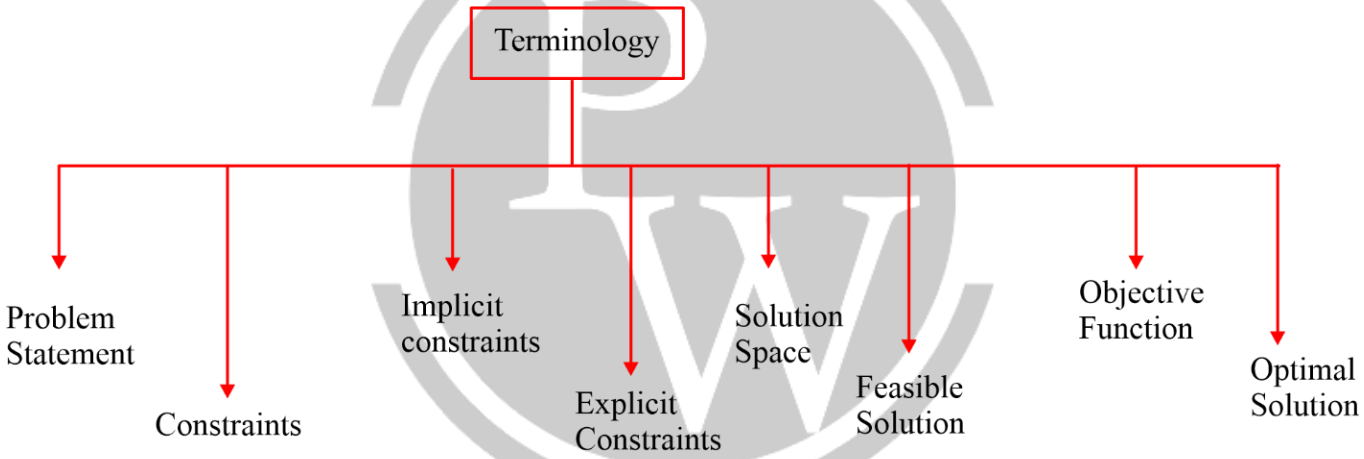
❑❑❑

3 GREEDY TECHNIQUE

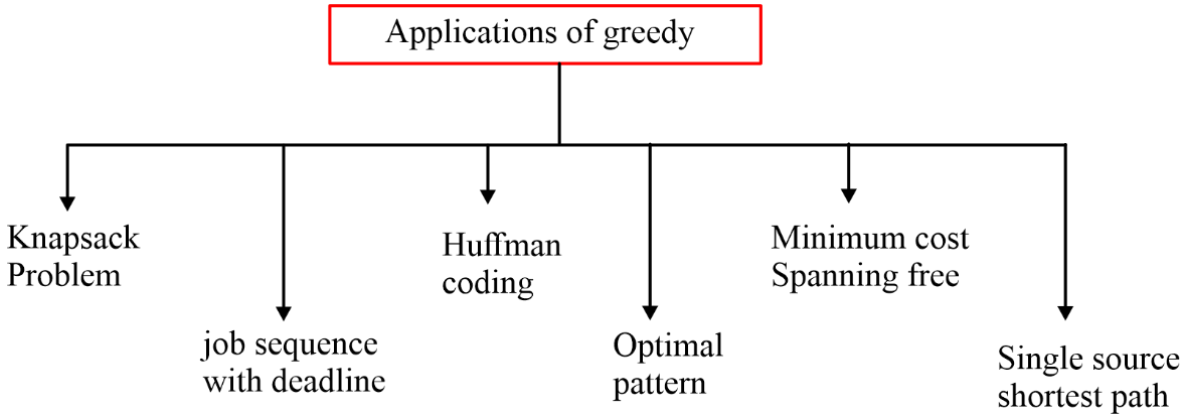
3.1 Greedy Technique

- Greedy method is an algorithm design strategy used for solving problems where solution are seen as result of making a sequence of decisions.
- A problem may contain more than one solution.

3.2 Terminology



3.3 Applications of greedy

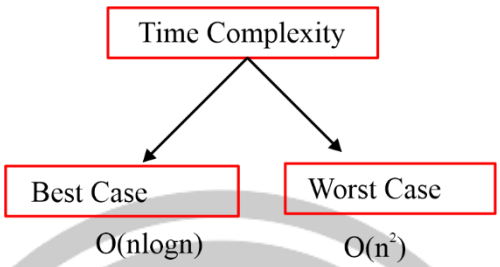


3.4 Knapsack Problem

Time complexity T(n) = O(nlogn)

3.5 Job Sequence with Deadline

- Single CPU only.
- Arrival time of each job is same.
- No pre-emption.



3.6 Optimal Merge Pattern

- This is a problem related to merging of files. Given a set of n-files in sorted order. It is required to merge them into a single sorted file with 2-way merging.
- This problem is like merging process in merge sort. In merge sort we were interested in number of comparisons but in optimal merge pattern we are interested in record movement (i.e moving a record from one file to another file).
    - If file F1 has 'n' records and file 'F2 ' has 'm' records then number of record movement will be 'm+n'. 1 2

  The problem of optimal merge pattern involves merging of n-files (n≥2).
- At any point choose two records with least weight merge them and put them in list and continue it until all records are merged.
- Time complexity T(n) = O(nlogn)
- Space complexity = O(n)

3.7 Huffman Coding

- Huffman coding is essentially a non-uniform encoding with convention that the character with higher frequency (probability) of occurrence will be enclosed with less number of bits.
- It comes under data compression technique.
- Time complexity T(n) = O(nlogn)

3.8 Minimum Cost Spanning Tree

3.8.1 Graph

Graph (V., E)

Set of vertices set of edges

- Let G(V, E) be a simple graph then

$$\text{Maximum edges} = \frac{V(V-1)}{2}$$

$$E \leq \frac{V(V-1)}{2}$$

$E \leq C.V^2$ C is constant

Note:

$E = O(V^2)$

$\text{Log } E = O(\log V)$

3.8.2 Graph Representation

Graph Representation

Adjacency matrix Adjacency list

- For more edges (Dense Graph) Adj. matrix is better (density more).
- For less edge (sparse graph) Adj list is better.

	Matrix	List
(1) Finding degree of vertex⇒ Time Complexity	O(V) Every Case	O(1) Best Case O(V1) Worst Case
(2)Finding total edges ⇒ Time Complexity	$O(V^2)$ Every Case	O(V+2E) Worst Case O(V) Best Case
(3) Finding 2-vertices adjacent (or)not ⇒ Time Complexity	O(1)	O(V-1) Worst Case O(1) Best Case
(4) G(V,E) ⇒ space	$O(V^2)$ Every Case	O(V+E) Every Case

3.8.3 What is Spanning Tree

A subgraph T(V, E') of G(V, E) where E' is the subset of ($E' \subseteq E$) is a spanning tree iff 'T' is a tree.
A sub graph G('V, E') of G(V, E) is said to be spanning tree.
(1) T' should contain all vertices of G
(2) T' should contain (V-1) edged where V is number of vertices without cycle.

(3) T' should connected.

3.8.4 Minimum Cost Spanning Tree

Minimum cost spanning tree is the one in which cost of the spanning tree formed should be minimum.

3.8.5 Prims Algorithm

- Select Any vertex

  Time complexity = V + VlogV + 2E + ElogV

  = O(E + V)logV

Using Sorted Array & Adjacency List

V + 2E + E × V = O(EV)

Using Sorted Array & Adjacency List

V × O(1) + V2 + E × V= O(EV)

3.8.6 Kruskal algorithm

- Take first minimum edge

  Time complexity = E log E + (V + E)

  = O(E log E) = O(ElogV)

  If edges are already sorted

  TC = O(E + V)

3.9 Single Source Shortest Path

3.9.1 Dijkstra Algorithm

- Using min heap & adjacency list = O(E + V)logV
- Using adjacency Matrix & Min heap = $O(V^2 ElogV)$
- Using adjacency list & Unsorted Array = $O(V^2)$
- Using adjacency list & Sorted Doubly Linked List = O(EV)

3.9.2 Bellman-Ford

- Time Complexity = O(EV)
- If negative edge weight cycle then for some vertices Incorrect answer.

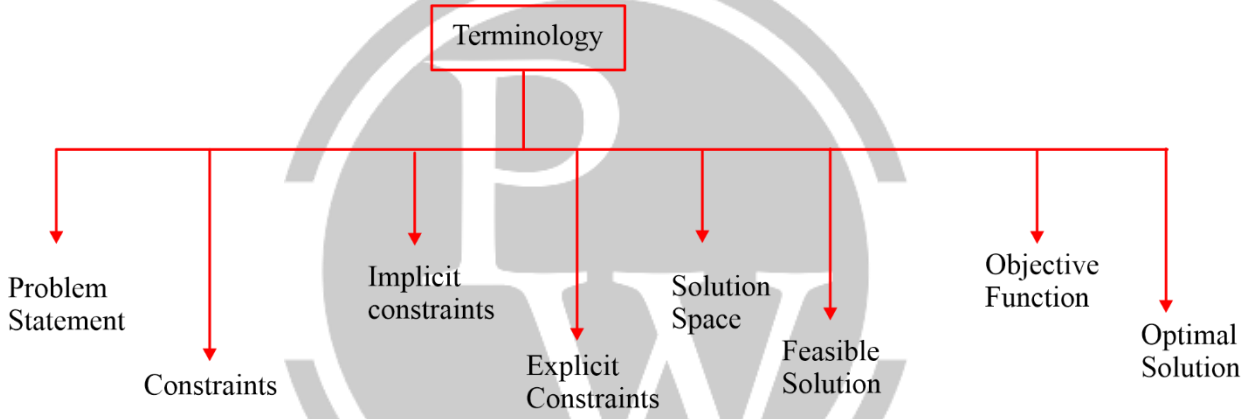
❑❑❑

4 DYNAMIC PROGRAMMING

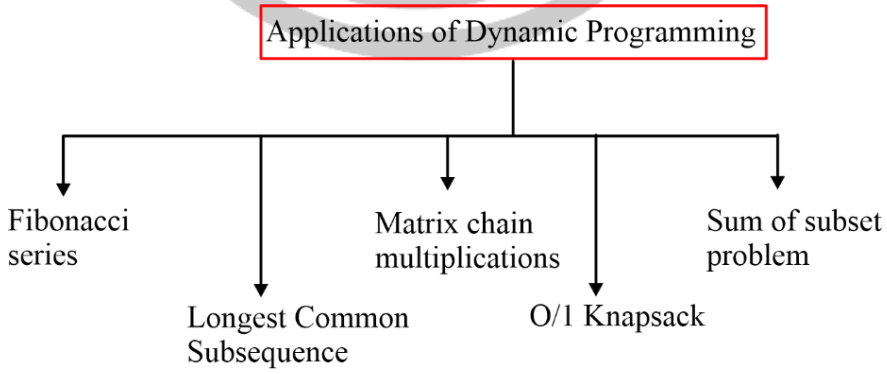
4.1 Dynamic Programming

In dynamic programming for optimal solution always computes distinct function calls.

4.2 Terminology



4.3 Application of Dynamic Programming

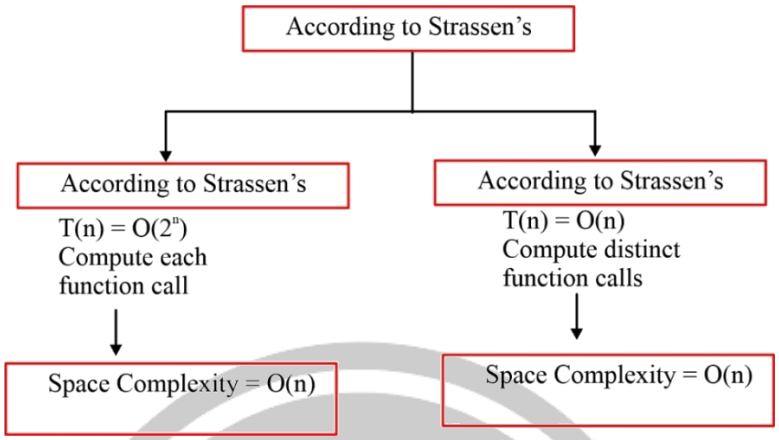


4.4 Fibonacci Series

- Time complexity T(n) = O(nlogn)
- Computes distinct function calls.

4.5 Job Sequence with Deadline

- Single CPU only
- Arrival time of each job is same
- No pre-emption



4.6 Longest Common Sub sequence (LCS)

- For common subsequence always consider two strings:
- P = <ABCDB> – Q = <BDCABA>
- Common subsequences for both 'p' are
- S = <A>
- S = <AB>
- S = <CAB>
- S = <BDAB>
- A common subsequence of longest length is known as longest common subsequence.
- For above problem longest common subsequence will be of length u.

4.6.1 Applications of LCS

1. Genomics
2. Software engineering applications
3. Plagiarism
4. Data gathering system of search engines

4.6.2 Algorithm for LCS

```
LCS (p,q)
{
1.  For   i ←0 to n −1
          L[i −1]=0
2.  For   j ←0 to m −1
          L[−1, j]=0
```

3. *For* $i \leftarrow 0$ *to* $n-1$

   For $j \leftarrow 0$ *to* $m-1$

   If $(p[i]=q[j])$ *then*

   $L[i, j]=1+L(i-1, j-1]$;

   else

   $L[i, j]=max\{L[i, j-1], L[i-1, j]\}$

   }

- Time complexity of step 1 = O(n)
- Time complexity of step 2 = O(m)
- Time complexity of step 3 = O(mn)
- Total Time complexity = O(n) + O(m) + O(mn)
  = O(mn)
- Space complexity = O[(M+1).(n+1)]
  = O(mn)

4.7 Matrix Chain Multiplications

Two matrices 'A' and 'B' are compatible if and only number of column of first matrix must be equal to number of rows of second matrix.

4.7.1 Brute force method

Number of parenthesizing for a given chain is given by Catalan number: $\left[\frac{1}{n+1}\ {}^{2n}C_n\right]$

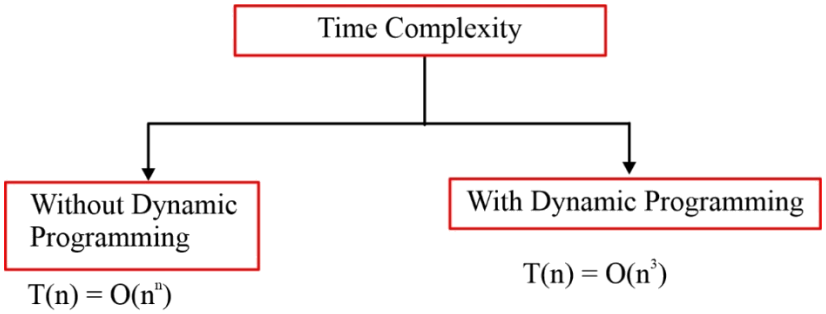
Time complexity = $O(n^n)$

Space complexity = O(n)

4.7.2 Algorithm For Matrix Chain Multiplication

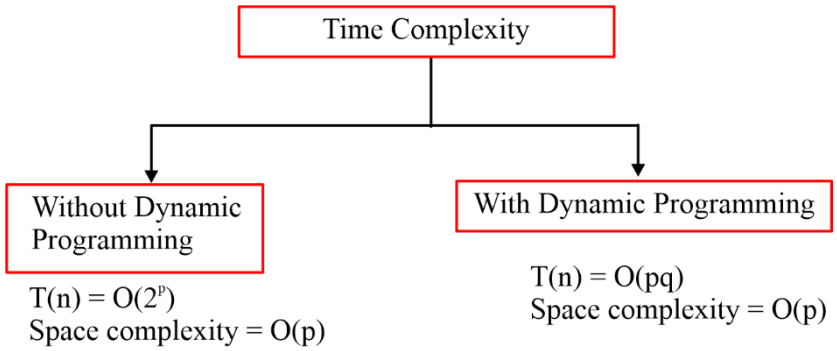
The time complexity of multiply the given chain of n matrices <A_1, A2 A3…. An> using dynamic programming (district function call) is $O(n^3)$

Space complexity = $O(n^2)$



4.8 O/1 Knapsack Problem

The maximum profit can be achieved by O/1 knapsack problem where capacity of problem is 'p' and number of objects are 'q'.



4.9 Sum of Subset Problem

- Given n-elements and an integer 'm', it is required to determine whether there exists a subset of given n elements, whose sum equal M.
- This is a decision problem (True/False).

4.9.1 Algorithm for Sum of Subset Problem

```
SoS(n, M, A)
// A [1 . . . n] is an array of elements
    {
1.  for i = 0 to n
        for j = 0 to M
        if (i > =0 and j = 0)
        SoS [i, j] = T
    else
        if (i = 0 and j > 0)
        SoS [i, j] = F;
    else
        if (A[i] > j)
        SoS [i, j] = SoS [i – 1, j]
    else
        SoS [i, j] = SoS [i – 1, j] or
        SoS [i – 1, j – A[i]]
    }
```

4.9.2 Time Complexity of SoS

Two for loops are there thus repeating for (n * m) times. Thus, time complexity = O(n * m)

Time complexity of SoS becomes exponential if $M = 2^n$

$$T.C = O(n \times 2^n)$$

❐❐❐

GATE Exam 2024?

PARAKRAM BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in ***Hindi***

Weekday & Weekend Batches Available

JOIN NOW!

Theory of Computation

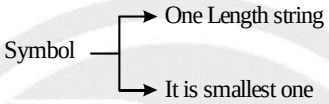
Theory of Computation

INDEX

1 BASICS OF THEORY OF COMPUTATION

1.1 Symbol

Symbol represents very unique in the world. Any small thing that never be broken into any other is called as symbol.



1.2 Alphabet (∑)

It is a set of finite number of symbols.

Example:

- English alphabet = {a, b, ... z}
- Binary alphabet = {0, 1}
- Decimal alphabet = {0, 1, 2, ... 9}
- We can create our own alphabet = {gate, cs, it, exam}
  where gate, cs, it, exam all are symbols

Alphabet (Σ) = {gate, cs, it, exam}
{ w , x , y , z }

1.3 String

- It is sequence of symbols defined over given alphabet.
  let ∑ = {a, b}
- Strings possible are ∈, a, b, aa, bb……
- Strings over English Alphabet: deva, gate, exam, etc.
- Strings over Binary Alphabet: 0010, 1011, 1101, 1111, etc.
- Strings over Decimal Alphabet: 3012, 2345, 5438, etc.

Note:

- Different length strings over given alphabet
  I Zero length string → Empty string / Null string ∈ or λ is used to denote empty string length of empty string. |∈| = 0.
  II One length string over ∑ = {a, b} are a, and b (2 strings).
  III Two length strings over ∑ = {a, b} are da, ab, ba, and bb (4 strings).

1.4 Operations on Strings

There are various operations on strings:

- Unary and Binary operations

1. **length of a string:** Length of a string is denoted as |w| and is defined as the number of positions for the symbol in the string.

   Example: w = aba
   |w| =|aba| = 3

2. **Reversal of a string:** It will reverse or changes the order of a given string w.

   Example: w = abb
   w^R = bba

1.4.1 Concatenation of Two Strings

Given two strings w_1 and w_2, we define the concatenation of w_1 and w_2 to be the string as w_1w_2.

Example:

w_1 = a, and w_2= ba

Then w_1w_2 = a.ba = aba.

1.4.2 Prefix of a String

A substring with the sequence of beginning symbols of a given string is called a "prefix".

Example:

(i) w = aaaa

Prefixes = {∈, a, aa, aaa, aaaa}

(ii) w = abcd

Prefixes = {∈, a, ab, abc, abcd}

Note:

If |w| = n then $|w^{Rev}|$ = n.

If length of w_1 is n_1 and length of w_2 is n_2 then length of w_1w_2 = $|w_1 w_2|$ = $n_1 + n_2$.

- Let w be n length string.

(i) Number of prefixes in w = n+1

(ii) Number of non-empty prefixes of w = n

(iii) Number of different length prefixes of w = n + 1

(iv) Number of different length prefixes of w excluding zero length = n

1.4.3 Suffix of a String

A substring with the sequence of ending symbols of a given string is called a "suffix".

Example:

(i) w = aaaa

Suffixes = {∈, a, aa, aaa, aaaa}

(ii) w = abcd

Suffixes = {∈, d, cd, bcd, abcd}

Note :

- Let w be n length string.
  (i) Number of suffixes of w = n + 1
  (ii) Number of non-empty suffixes of w = n
  (iii) Number of different length suffixes of w = n + 1
  (iv) Number of different length suffixes of w excluding zero length = n

1.4.4 Substring of a String

- Let w be n length string.

(i) Number of substrings of w
- minimum = n +1 (over 1 symbol)
- maximum = 1+ n + (n – 1) + ...+1 (all characters distinct) =1 + ∑n = ∑n +1

(ii) Number of non-empty substrings of w
- minimum = n
- maximum = ∑n

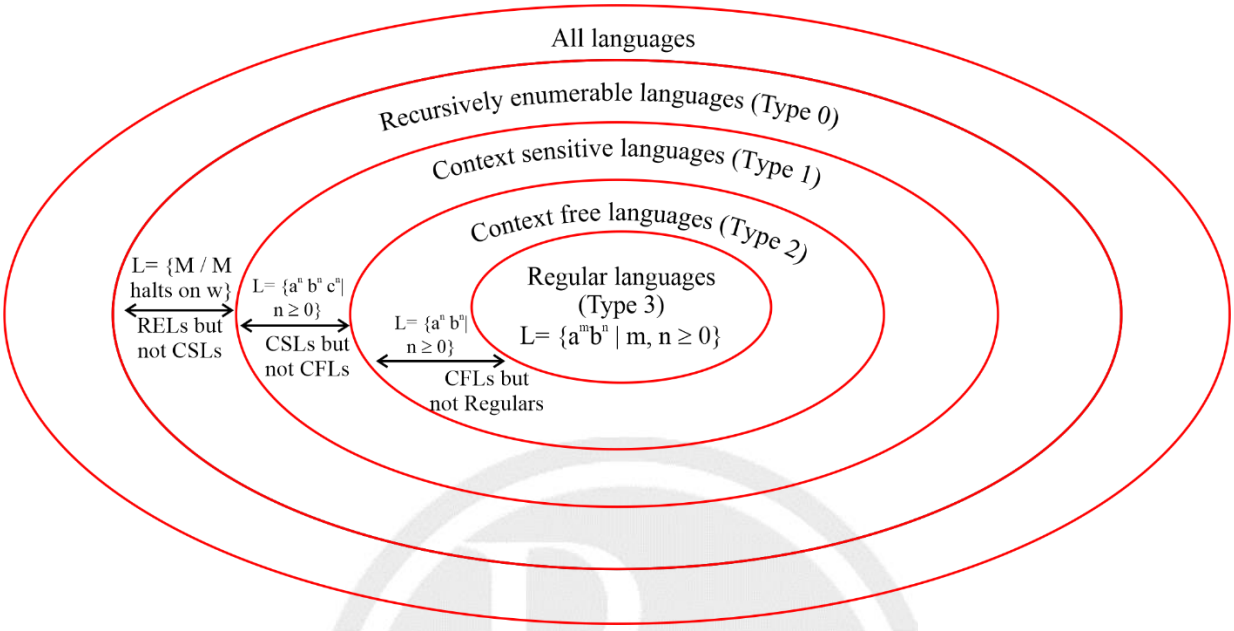
- Number of different length substrings of any given n length string =n+1
- Number of different length substrings of any given n length string excluding zero length=n.

1.5 Relation between Symbol, Alphabet, String and Language

Character	Symbol
⇓	⇓
Character set	Alphabet
⇓	⇓
Token	String
⇓	⇓
C Language	Language

1.5.1 Chomsky Hierarchy

- Chomsky hierarchy includes all the problems in the world classified into classes.



- Type 3 is the smallest class, and Type 0 is the biggest class.

	Type 3	Type 2	Type 1	Type 0
Language:	Regular	Context free	Context sensitive	Recursively Enumerable
Automata:	Finite Automata	Push down Automata	Linear Bounded Automata	Turing Machine
Grammar:	Regular	Context free	Context Sensitive	Unrestricted

1.6 Language

- Language is set of strings defined over alphabet (Σ).
- Let $\Sigma = \{a,b\}$. Then $\Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup$ …….
  = set of all strings
  = universal language
  = $\{\in, a, b, aa, ab, ba, ...\}$
- $\Sigma^2 = \Sigma^1 \bullet \Sigma^1 = \{a,b\}.\{a,b\} = \{aa, ab, ba, bb\}$
- Language: It is a subset of Σ^*

$\therefore$ $\boxed{L \subseteq \Sigma^*}$

- A language is a collection of strings that must be a subset of Σ^* where Σ^* is a universal language.

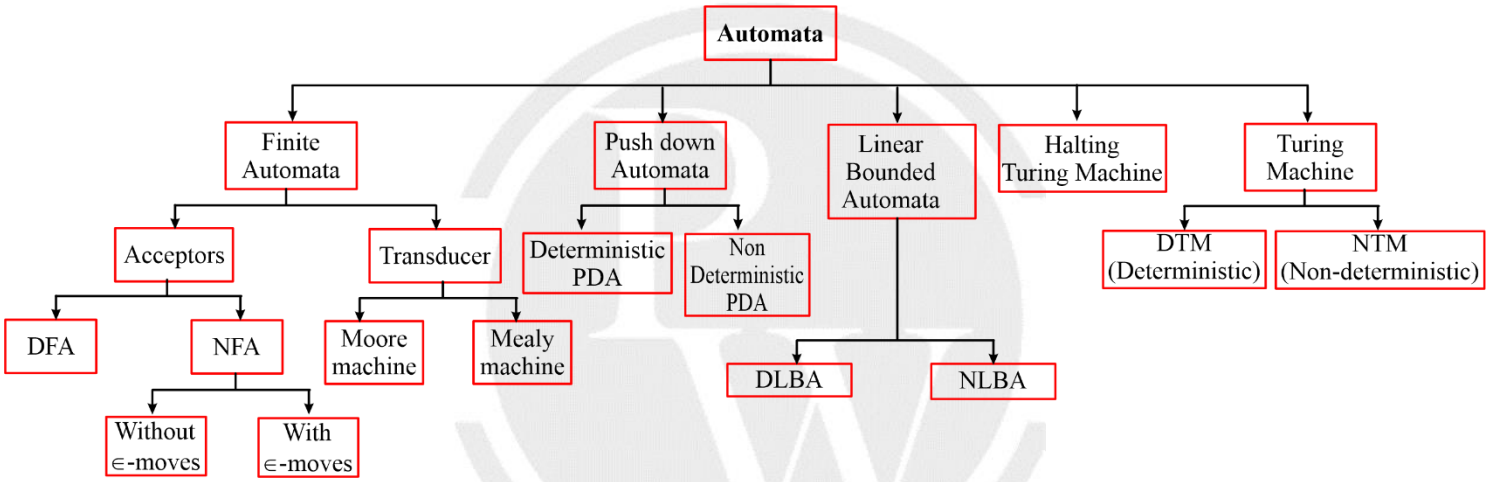
Example:

L = ab* = {a, ab, abb,.....}

1.7 Types of Languages

- Finite Language
- Infinite Language
- Regular language
- DCFL (Deterministic CFL)
- CFL
- CSL
- Recursive Language
- Recursive Enumerable Language (REL)

1.8 Types of automata



- Ralations between these machines:

$$FA < DPDA < PDA < LBA < HTM < TM$$

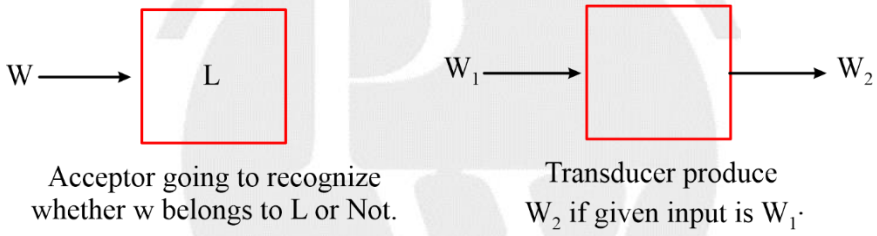
1. **Less power:** It can Represent less number of languages.
2. **More power:** It can Represent more number of languages Compare to all these machines.

❐❐❐

2 FINITE AUTOMATA

2.1 Introduction

- Finite Automata describes or represents a regular language.
- Finite Automata is of two types:
  (i) Acceptor: Accepts or rejects given string.
  (ii) Transducer: Produces output string for given input string.



Acceptor going to recognize whether w belongs to L or Not.

Transducer produce W_2 if given input is W_1.

Example:

1's complement of binary number: Input = 10100, Output = 01011

2.2 Finite Acceptor (Finite Automata)

- Finite Automata:

$FA = (Q, \Sigma, \delta, q_0, F)$

- F → Set of final states
- q_0 → Initial state
- δ → Transition function
- Σ → Set of input symbols
- Q → Set of finite number of states

Transition function (δ)

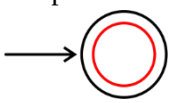
(1) DFA : ⟶ $\delta: Q \times \Sigma, \rightarrow Q$

(2) NFA: ⟶ Without $\in-$ moves $\delta: Q \times \Sigma, \rightarrow 2^Q$

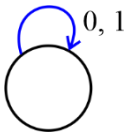
⟶ With $\in-$ moves $\delta: Q \times \Sigma \cup \{\in\} \rightarrow 2^Q$

2.3 Construction of DFA

1. If epsilon belongs to L, then initial state must be final in DFA.

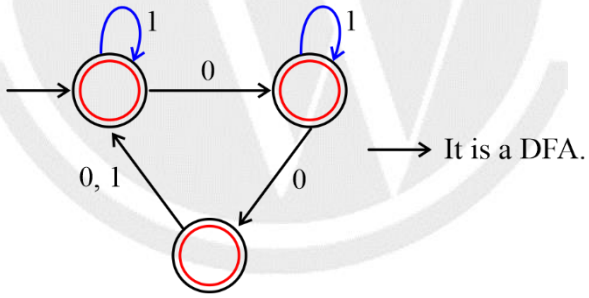


2. **Dead state:** It is non-final state but it never contain a path to final.

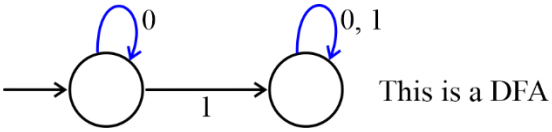


- Once we reach the dead state, there is no way to reach to the final state.
- If every state is final then every string accepted in the DFA.

3. If all states are finals in DFA then L(DFA) = $\sum^*$



4. If every state is non final in a DFA then L(DFA) = ϕ or L(DFA) = {},



$\boxed{\overline{L} = \sum^* - L}$

$\boxed{L \cup \overline{L} = \sum^*}$

$\boxed{L \cap \overline{L} = \phi}$

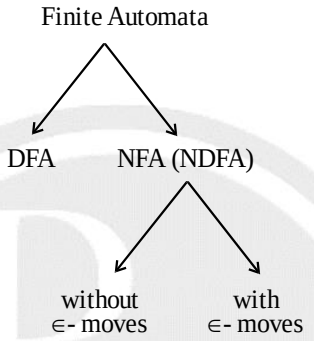
- If $|w| = n$ or $|w| \leq n$, then number of states in minimum DFA = (n + 2) states
- If $|w| \geq n$ then number of states in minimum DFA = n+ 1 states.
- Start condition, exactly, atmost length question requires Dead state but end condition, contain substring, atleast length questions do not require dead state.

- Over one symbol $\sum = \{a\}$. Length of string is equal to number of a's in string. $|w| = n_a(w)$.
- Let $L=\{w|w \in \{a,b\}^*, n_a(w)$ is divisible by m, $n_b(w)$ is divisible by n$\}$. Then Number of states in DFA= m×n.
- Let L={w | w belongs to {a. b}*, n^{th} symbol of w from begin is 'a'}. Then Number of states in DFA = (n + 2).
- Number of equivalence classes for any regular set L = Number of states in minimal DFA that accepts L.

2.4 Non-Deterministic Finite Automata

2.4.1 Introduction

- Finite Automata can be designed in two ways:



- All these finite state machines are equivalent. DFA ≅ . NFA
- We can convert one finite state machine to any other finite state machine.
- For every gular language, we can design infinite equivaent DFAs or infinite equivalent NFAs but minimum DFA is uique for given regular language.
- For every regular language, one or more minimum NFAs may exist.

Note : For every regular language:

(i) Unique minimum DFA exists.

(ii) One or more minimum NFAs exists.

2.5 Comparison of NFA and DFA

	$\|w\| = n$	$\|w\| \le n$	$\|w\| \ge n$
Number of states in NFA	n + 1	n + 1	n + 1
Number of states in DFA	n + 2	n + 2	n + 1

- If every string has n^{th} symbol from begin is 'a' over binary alphabet {a, b} then (n + 1) states in minimum NFA but (n+2) states in minimum DFA.

- If every string has n^{th} symbol from end is 'a' over binary alphabet {a, b} then 2^n states in minimum DFA and (n + 1) states in minimum NFA.

2.5.1 COMPARISON OF DFA AND NFA (NFA vs DFA)

	NFA	DFA
(1) Transition Function (δ)	$Q \times \Sigma \rightarrow 2^Q$	$Q \times \Sigma \rightarrow Q$
(2) Number of paths for string	For valid string: 1 path For invalid string: >=0 paths	For valid string: 1 path For invalid string: 1 path

2.5.2 Number of states in DFA and NFA for Regular Languages

Language	NFA states	DFA states
(1) {w | w ∈ {a, b}*, |w| = n}	n + 1	n + 2
(2) {w | w ∈ {a, b}*, |w| ≤ n}	n + 1	n + 2
(3) {w | w ∈ {a, b}*, |w| ≥ n}	n + 1	n + 2
(4) {w |w ∈ {a, b}*, w starts with a}	2	3
(5) {w| w∈ {a, b}*, n^{th} symbol from begin is a}	n + 1	n + 2
(6) {w| w ∈ {a, b}*, n^{th} symbol from end is 'a'}	n + 1	2^n

2.5.3 Finding number of states in DFA using NFA

NFA (n states)

⇓ Subset Construction

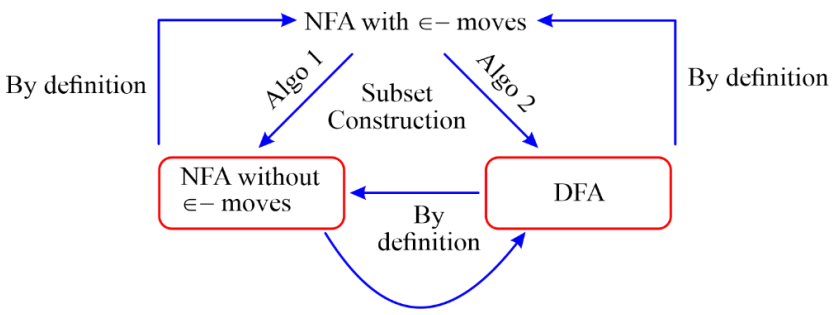
DFA (2^n sates exists)

⇓ Partition algorithm

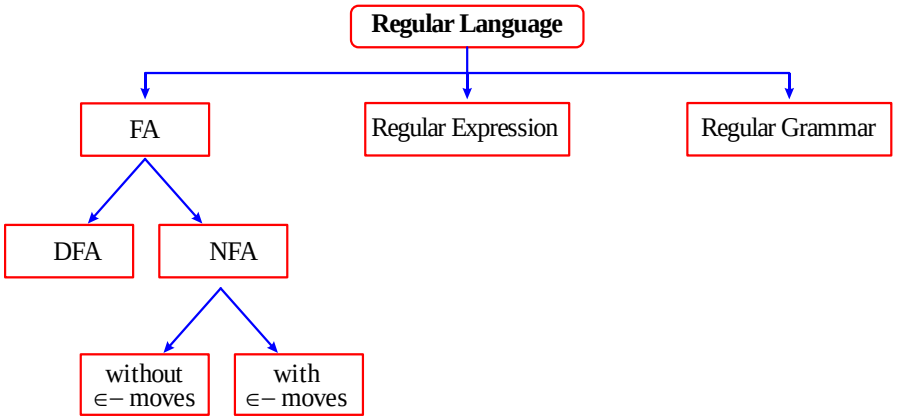
Minimum DFA (≤ 2^n states) atmost 2^n states

- Every DFA is NFA, but NFA need not be DFA.

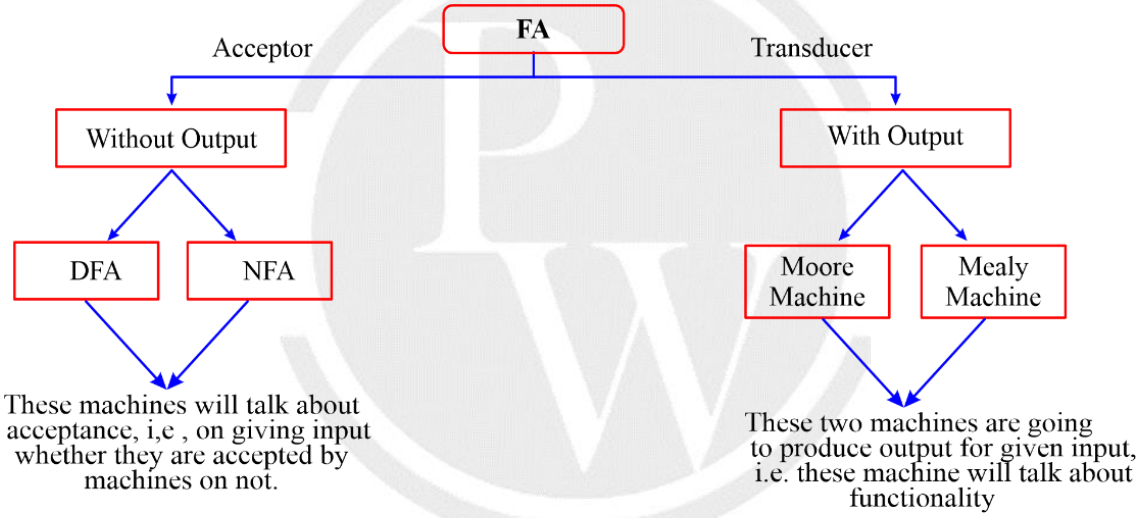
2.5.4 Relation between NFA with epsilon, NFA without epsilon, and DFA



2.6 Regular language Representation



2.7 FA Classification

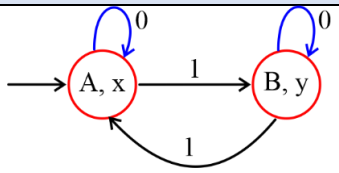


2.8 Moore machine and mealy machine

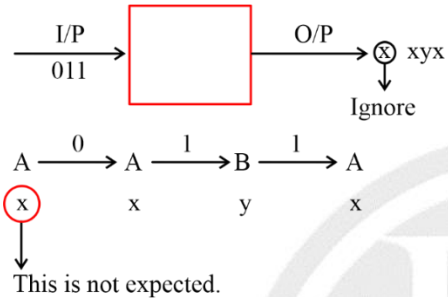
Moore machine	Mealy machine
Transition Function $\delta : Q \times \Sigma \rightarrow Q$ Output Function $\lambda : Q \rightarrow \Delta$ Outpput is asociated with every state.	$\delta : Q \times \Sigma \rightarrow Q$ $\lambda : Q \times \Sigma \rightarrow \Delta$ Output is associated with transition.

2.8.1 Example for Moore machine and Mealy machine

Moore

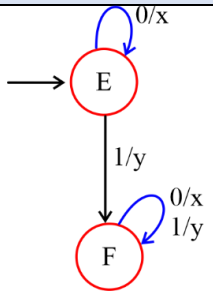


Q = {A, B}
∑ = {0, 1}
Δ = {x, y}
Outpput is asociated with state.

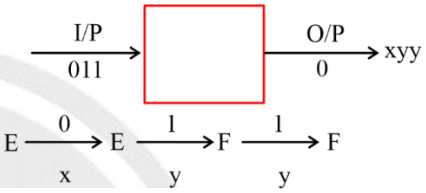


- Extra one output is produced other than desired output.
- So, we can ignore this extra output as this is the machine property.

Mealy



Q = {A, B}
∑ = {0, 1}
Δ = {x, y}
Output is associated with transition.

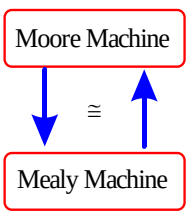


- No extra output is produced.
- For 3 length input, we are getting the 3 length output.

2.9 Difference between Moore machine and Mealy machine

		Moore machine	Mealy machine
1.	δ	$Q \times \Sigma \to Q$	$Q \times \Sigma \to Q$
2.	λ	$Q \to \Delta$	$Q \times \Sigma \to \Delta$
3.	Length of O/p	If n length I/P (assume 1 length O/P symbol is taken at each state) then O/P length is (n+1).	If n length input (assume 1 length O/P symbol is taken at each transition) is given then O/P length is n.
4.	By default	DFA no final state.	DFA no final state.

2.10 Constuction of FA with output



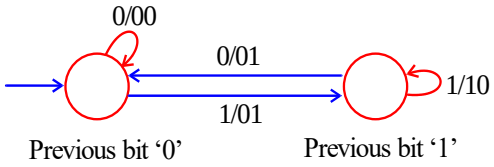
Note :

For every problem, if moore machine exists the we can also construct equivalent mealy machine.

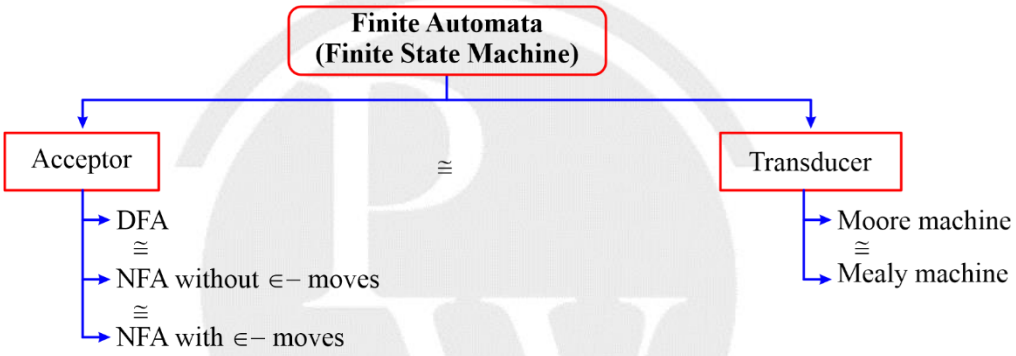
Example:

Sum of present bit and previous bit.

Mealy Machine



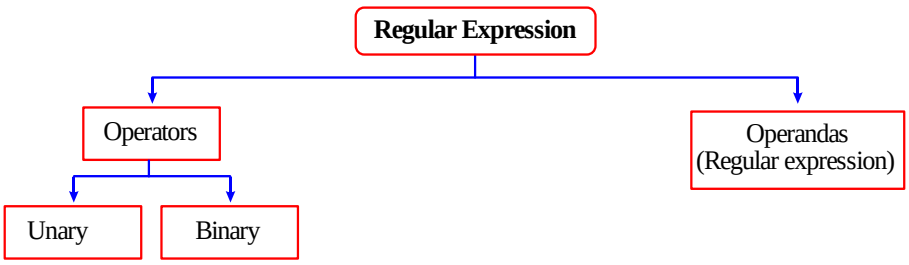
2.11 Classification of Finite State Machine (FSM)



2.12 Regular Expression

Definition:

- Regular expression represents a regular language.
- It describes a regular set.
- L (regular expression) is regular set.
- It is a kind of declarative way to represents a regular language.
- Regular expression generates a regular set.
- It uses 4 operators to represent a regular language.



2.13 Operators of Regular Expression

1. OR (+)
2. Concatenation(.)

} Binary Operator

3. Kleenestar (∗)
4. Kleenestar (+)

} Unary Operator

Regular Expression	Equivalent Regular Set
a + b	L(a + b) = {a, b}
a + a = a	L(a + a) = {a}
a + ϕ = a	L(a + ϕ) = {a}
a + ∈ = ∈ + a	L(a + ∈) = {a, ∈}
∈ + ∈ = ∈	L(∈+∈) = {∈}
ϕ + ϕ = ϕ	L(ϕ + ϕ) = {}
a · b = ab	L (a · b) = {ab}
a · ∈ = a	L (a · ∈) = {a}
∈ · ∈ = ∈	L(∈·∈) = {∈}
ϕ · ϕ = ϕ	L(ϕ · ϕ) = {ϕ}
a · a = aa = a^2	L (a · a) = aa = {a^2}

2.14 Kleene star/ kleene closure / closure

R* (Kleene closure of R):

$$R^* = R^\circ + R^1 + R^2 + R^3 + \ldots = \in + R + RR + RRR + \ldots.$$

Example:

$a^* = \{a^0, a^1, a^2, a^3, \ldots\} = \{\in, a, aa, aaa, \ldots\}$ = Set of all strings over a.

$\phi^* = \phi^0 + \phi^1 + \phi^2 + \phi^3 + \ldots. = \in + \phi + \phi + \phi + \ldots. = \in + \phi = \in$

2.15 Positive Closure (Kleeme Plus)

R^+ (Positive Closure of R):

$R^+ = R^1 + R^2 + R^3 + \ldots..$

$R^* = R^{\geq 0}$ i.e. repeart R any number of time

$R^+ = R^{\geq 1}$ i.e. repeat R atleast 1 time.

$R^* = R^+ + R^0$

Example:

(1) $a^+ = \{a, aa, aaa, \ldots\} = \{a^n \mid n \geq 1\}$

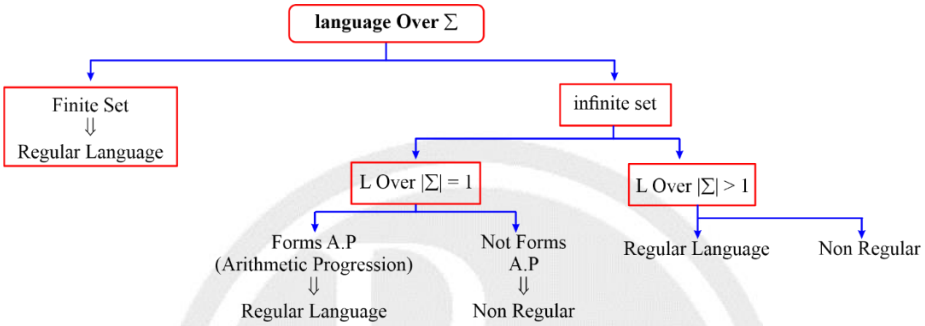
(2) $\in^+ = \in = \in^*$

(3) $\phi^+ = \phi$

Properties :

		OR	Concatenation
1.	Identity	ϕ	$\in$
2.	Associative	Yes	Yes
3.	Commutative	Yes	No
4.	Annihilatior	Σ^*	ϕ

2.16 Identification of Regular Languages



1. $L=\{a^nb^m \mid n, m \geq 0\} = a^*b^*$ (Regular language)
2. $L= \{a^nb^m \mid m<n<10\}$ (Finite Set, Regular language)
3. $L=\{a^nb^m \mid m > n > 10\}$ (Non Regular language)
4. $L= \{a^nb^m \mid m = n, m < 10\}$ (Regular language)
5. $L=\{ a^nb^m \mid m = n, m > 10\}$ (Non Regular language)
6. $L=\{a^mb^n \mid \gcd(m, n) = 1\}$ (Non Regular language)
7. $L=\{a^mb^n \mid LCM(m, n) = 1\}$ (Regular language)
8. $L= \{a^nb^n \mid n >= 0\}$ (Non Regular language)
9. $L=\{a^nb^{2n} \mid n \geq 0\}$ (Non Regular language)
10. $L=\{a^mb^n \mid m=even, n =odd\}$ (Regular language)
11. $L=\{a^mb^n \mid m = n = even\}$ (Non Regular language)

    OR

    $L = \{a^{2n}b^{2n} \mid n = even\}$
12. $L=\{a^*\}$ over $\Sigma= \{a\}$

    = Regular language
13. $L=\{a^{2n} \mid n >= 0\}$ over $\Sigma = \{a\}$.

    $L= a^{2n}=(aa)^*$

    = Regular language

14. L= $\{a^{Prime}\}$ over $\sum = \{a\}$

    = Non regular language

15. L=$\{ a^{n^2} \mid n \geq 0\}$ over $\sum = \{a\}$

    L={epsilon, a, aaaa, a^9, a^{16}...}, FA Not possible for L. So, it's Non-Regular language.

16. L= $\{ a^{2^n} \mid n > 0\}$over $\sum = \{a\}$

    Non Regular language

17. L= $\{ a^{2^n} \mid n \leq 10\}$

    = Regular language

18. L=$\{a^{n!} \mid n \geq 100\}$ over S = {a}

    = Non Regular language

19. L=$\{ a^{n^n} \mid n \geq 10\}$ over $\sum = \{a\}$

    = Non Regular language

20. L = $\{a^{Prime}\}^*$ over $\sum = \{a\}$

    L = complement of {a} = $\sum^* - \{a\} = \{\in, a^2, a^3\ a^4\ , a^5, a^6, a^7, ...\}$= Regular language

21. L = $\{a^{prime} \mid$ prime <100$\}$ is finite language (regular)

22. L = $\{w \# w \mid w \in a^*\}$

    L = $\{a^n \# a^n\}$

    dependency

    = Non Regular language

23. L = $\{w \# w \mid w \in (a+b)^*\}$

    = Non Regular language

24. L = $\{w \# w^R \mid w \in a^*\}$ is non regular

    L = $\{a^n \# a^n\}$

25. L = $\{w\ x\ w \mid w \in \{a, b\}^*, x \in \{a, b\}^+\}$

    Put w = $\in$, and x = $(a + b)^+$

    L = $(a + b)^+$

    = Regular language

26. L = $\{x\ w\ w \mid w \in \{a, b\}^*\ x \in \{a, b\}^+\}$

$L = (a + b)^+$

= Regular language

27. $L = \{w\ w^R\ x \mid w \in \{a, b\}^*, x \in \{a, b\}^+\}$

$L = (a + b)^+$

= Regular language

28. $L = \{w\ x\ w^R \mid w \in (a + b)^*, x \in \{a, b\}^+\}$

$L = (a + b)^+$

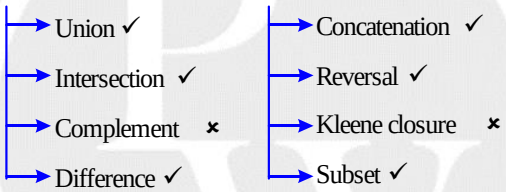
= Regular language

29. $L = \{x\ w\ w^R \mid w \in \{a, b\}^*, x \in \{a, b\}^+\}$

$L = (a + b)^+$

2.17 Closure Properties of Regular Languages

1. Closure properties for finite languages:



2.18 Table for FINITE sets

		Finite sets	Closed/Not Closed
(1)	Union ($\cup$)	$F_1 \cup F_2 \Rightarrow$ Finite	✓
(2)	Intersection ($\cap$)	$F_1 \cap F_2 \Rightarrow$ Finite	✓
(3)	Complement ($\overline{L}$)	$\overline{F} \Rightarrow$ NOT finite	×
(4)	$L_1 - L_2$	$F_1 - F_2 \Rightarrow$ Finite	✓
(5)	$L_1 . L_2$	$F_1 \cdot F_2 \Rightarrow$ Finite	✓
(6)	L^*, L^+	F^*, $F^+ \Rightarrow$ May or may not be finite	×
(7)	Subset (L)	Subset (Finite set) $\Rightarrow$ Finite	✓

2.18.1 Table for Infinite sets

		Infinite sets
(1)	Union ($\cup$)	Infinite $\cup$ Infinite $\Rightarrow$ Infinite
(2)	Intersection ($\cap$)	Infinite $\cap$ Infinite $\Rightarrow$ Need not be infinite
(3)	Complement ($\overline{L}$)	Complement of Infinite $\Rightarrow$ May or may not be infinite
(4)	$L_1 - L_2$	Need not be infinite
(5)	$L_1 . L_2$	Infinite
(6)	L^*, L^+	Infinite
(7)	Subset (L)	Need not be infinite

2.19 Closure Properties of Regulars

$L_i \rightarrow$ Regular

1.	$\cup$	2.	$\cap$
3.	$\overline{L}$	4.	$L_1 - L_2$
5.	$L_1 \cdot L_2$	6.	L^{Rev}
7.	L^+	8.	L^+
9.	Subset (L) is not closed for regular languages	10.	Prefix (L)
11.	Suffix (L)	12.	Substring (L)
13.	Substitution (L)	14.	Homomorphism (L)
15.	$\in$-free homomorphism (L)	16.	h^{-1} (L) (Inverse homomorphism (L))
17.	L_1/L_2 (Quotient)	18.	Symmetric difference
19.	Half (L)	20.	Second half (L)
21.	One-third (L) [1/3 (L)]	22.	Middle 1/3 (L)
23.	Last 1/3 (L)	24.	New $(L_1, L_2) = \text{Suffix}(L_1 \cup L_2^+)$
25.	Finite Union	26.	Finite Intersection
27.	Finite Difference	28.	Finite Concatenation

29.	Finite Subset	30.	Finite Substitution
31.	Infinite Union	32.	Infinite $\cap$
33.	Infinite Difference	34.	Infinite Concatenation
35.	Infinite Subset	36.	Infinite Substitution

- **Out of the given 36 closure properties, how many are closed for regular languages?**

  Subset operation, and 6 infinite operations are not closed, remaining all are closed for regular languages.

  Out of 36 operations total 7 operations are not closed.

2.19.1 Operations over regular languages

Examples:

(1) $\left.\begin{matrix} L_1 = a^* \\ L_2 = a^+ \end{matrix}\right\} \Rightarrow L_1 \,.\, L_2 = a^*$

(2) $\left.\begin{matrix} L_1 = \phi \\ L_2 = \text{Any} \end{matrix}\right\} \Rightarrow L_1 + L_2 = L_2$

(3) $\left.\begin{matrix} L_1 = a^* \\ L_2 = b^* \end{matrix}\right\} \Rightarrow L_1 + L_2 = a^* + b^*$

(4) $\left.\begin{matrix} L_1 = \Sigma^* \\ L_2 = \text{Any language over same } \Sigma \end{matrix}\right\} \quad L_1 + L_2 = \Sigma^* = L_1$

(5) $\left.\begin{matrix} L_1 = a\Sigma^* \\ L_2 = b\Sigma^* \end{matrix} \quad \Sigma = \{a,b\}\right\} \quad \begin{aligned} L_1 + L_2 &= a\Sigma^* + b\Sigma^* \\ &= (a+b)\Sigma^* \\ &= \Sigma^+ \end{aligned}$

Note :

1. $\left.\begin{matrix} L_1 = \phi \\ L_2 = \text{Any} \end{matrix}\right\} \Rightarrow L_1 \cdot L_2 = \phi$

2. $\left.\begin{matrix} L_1 = \Sigma^* \\ L_2 = \text{Any} \end{matrix}\right\} \Rightarrow L_1 + L_2 = L_2$

2.19.2 Properties of regular languages

I. If both L_1 and L_2 are regular sets then $L_1 \cap L_2$ is Regular.

II. IF $L_1 \cap L_2$ is regular set then

- L_1 "need not be regular"

- L_2 "need not be regular".

III. If L is regular then $\overline{L}$ is regular.

IV. If $\overline{L}$ is regular then L is regular.

V. L is regular iff $\overline{L}$ is regular

VI. L is not regular iff $\overline{L}$ is not regular.

Example:

$a^n b^n$ is not regular and $\overline{a^n b^n}$ is not regular.

2.19.3 Arden's Lemma and Kleene Method

Arden's Lemma:

If R = Q + RP and P does not contain $\in$ then $\boxed{R = QP^*}$

$R = Q + RP$ ...(1)

$= Q + (Q + RP)\,P = Q + QP + RP^2$ ...(2)

Substitute R one more time in equation (2)

$R = Q + QP + QP^2 + RP^3$

If we do repetitive substitution infinite time, we will get $\boxed{R = QP^*}$

Kleene Method:

Kleene Method

→ Dynamic programming $O(n^3)$

→ $\boxed{R_{ij}^{k}}$ where *i* is the initial state, *j* is the final state
k represents number of states

$$\boxed{R_{ij}^{k} = R_{ij}^{k-1} + R_{ik}^{k-1}\left(R_{kk}^{k-1}\right)^{*} R_{kj}^{k-1}}$$

Regular Grammar

→ It represents a regular set

→ It can be either left linear grammar (LLG) or right linear grammar (RLG)

LLG	RLG
Each production in LLG follows $V \rightarrow VT^*$ OR $V \rightarrow T^*$	Each production in RLG follows $V \rightarrow T^*V$ OR $V \rightarrow T^*$

2.20 Identify the Language Generated by Regular Grammar

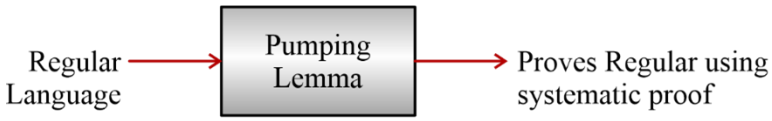
	Regular Grammar	Regular Language
1.	$S \rightarrow \in$	$L = \{\in\}$
2.	$S \rightarrow \in \mid a$	$L = \{\in, a\}$
3.	$S \rightarrow aa \mid abc \mid d$	$L = \{aa, abc, d\}$

4.	S → Aa A → b	By default S is a start symbol here. L = {ba}
5.	S → Aa	Useless production. It has no meaning. L = { } = ϕ
6.	$S \rightarrow \underset{\text{Useless}}{\boxed{Ab}} \mid b$	L = {b}
7.	↑ S → Aa | Ba A → a B → b Look from bottom to top	L = {aa, bb}
8.	↑ S → Aa A → ∈|b	L(A) = {∈, b} L = L(S) = L(A) · a = {a, ba}

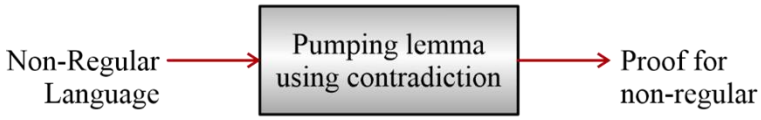
Regular Language	Regular Grammar
a^*	(I) S → Sa | ∈ OR (II) S → aS | ∈
a^+	(I) S → Sa | a OR (II) S → aS | a
$(a + b)^*$	(I) S → Sa | Sb | ∈ OR (II) S → aS | bS | ∈
$(a + b)^+$	(I) S → Sa | Sb | a | b OR (II) S → aS | bS | a | b
$(ab)^*$	(I) S → Sab | ∈ OR (II) S → abS | ∈

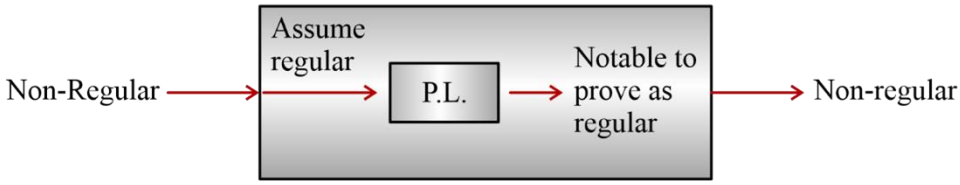
2.21 Pumping-Lemma (PL)

2.21.1 Pumping Lemma for Regular Languages



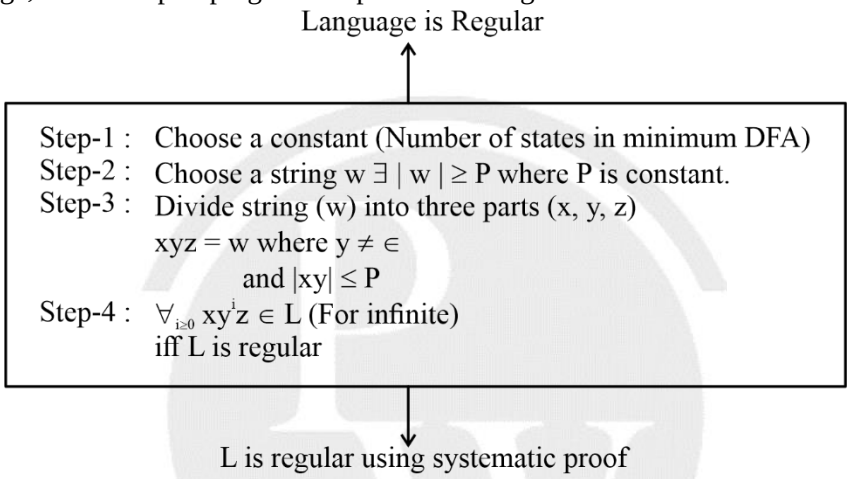
2.21.2 Pumping Lemma for Non-Regular Languages using contradiction





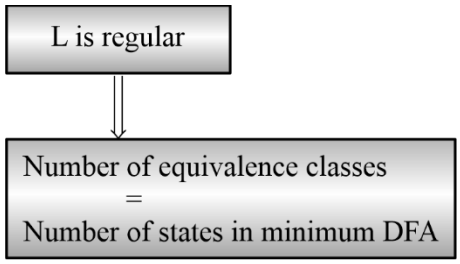
2.21.3 Proof for Regular Languages

- If L is Regular language, then how pumping lemma proves it as regular?



2.22 Equivalence Classes

(1) L is regular iff L has finite number of equivalence classes.
(2) L is not regular iff L has infinite equivalence classes.

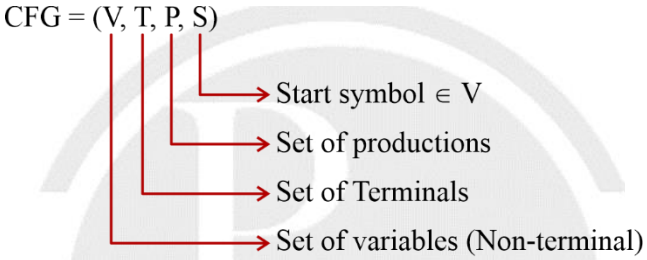


❐❐❐

3 PUSH DOWN AUTOMATA

3.1 Context Free Grammar

CFG: It represents a Context Free Language.



- Rule of each production in P:
  $V \rightarrow (V \cup T)^*$
  V = only 1 variable in LHS
- To derive a string, following derivations can be used.

1. **Linear Derivation:** Linear derivation is two types

   (a) Left Most Derivation (LMD)

   (b) Right Most Derivation (RMD)

2. **Non- linear Derivation:** Non- linear Derivation OR Parse Tree OR Derivation Tree

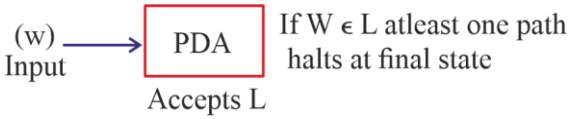
3.2 Types of Context Free Grammars

There are two types of CFG:

1. **Ambiguous CFG:** At least one string has more than one derivation.
2. **Unambiguous CFG:** Every string (w) generated by CFG has exactly one derivation.

3.3 Pushdown Automata (PDA)

PDA accepts context free language (CFL). PDA also called as NPDA.



3.4 PDA acceptance mechanisms

- PDA acceptance mechanisms are three types:
  1. PDA acceptance using final state.
  2. PDA acceptance using empty stack.
  3. PDA acceptance using both final state and Empty stack.
     All PDA acceptance mechanisms are equivalent.
- DPDA acceptance mechanism are two types:
  1. DPDA using final stack.
  2. DPDA using both final state and empty stack.

3.5 PDA configuration

PDA = $(Q, \sum, \delta, q_0, F, Z_0, \Gamma)$.

where Q = Set of states

$\sum$ = input alphabet

δ = Transition Function (PDA/NPDA $\delta : Q \times \sum \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$)

F = Set of Final state

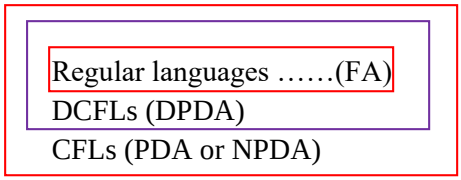
Z_0 = Bottom symbol or initially TOS

Γ = Stack Alphabet

- DPDA transition Function is [$\delta : Q \times \sum \times \Gamma \rightarrow Q \times \Gamma^*$]
- Difference between DPDA and PDA

DPDA	PDA
[1] $\delta : Q \times \sum \times \Gamma \rightarrow Q \times \Gamma^*$	$\delta : Q \times \sum \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$
[2] Every DPDA is PDA	Every PDA need not be DPDA
[3] DPDA acceptance with (a) Final state mechanism (b) Both Final and empty stack	PDA acceptance with (a) Final state mechanism (b) Empty stack mechanism (c) Both Empty stack and Final state

- Relation between Regular, DCFL and CFL.



I If L is regular language, then it is also DCFL and CFL.

II Every DCFL is CFL, but it need not be regular.

3.6 Closure Properties of CFLs

Operation	Closed / Not Closed
Union ($L_1 \cup L_2$)	✓
Intersection ($L_1 \cap L_2$)	×
Complement ($\overline{L}$)	×
Set difference (L_1-L_2)	×
Concatenation ($L_1.L_2$)	✓
Reversal (L^{Rev})	✓
Kleene Closure (L)	✓
Kleene Plus ($L^{\bullet}$)	✓
Subset (L)	×
Prefix (L)	✓
Suffix (L)	✓
Substring (L)	✓
Substitution (L)	✓
Homomorphism (L)	✓
$\in$ - free Homomorphism h(L)	✓
Inverse Homomorphism h^{-1} (L)	✓
quotient (L_1, L_2) = L_1/L_2	×
Symmetric difference (L_1, L_2)	×
Finite Union	✓
Finite Intersection	×
Finite difference	×
Finite concatenation	✓
Finite subset	✓
Finite substitution	✓
Infinite Union	×
Infinite intersection	×
Infinite difference	×
Infinite concatenation	×
Infinite Subset	×
Infinite Substitution	×
Union with Regular (L U Regular)	✓
L U Regular	✓
L – Reg	✓
Reg – L	×

Note :(i) CFL ∩ CFL = Need not be CFL

(ii) CFL ∩ Regular = CFL (Need not be DCFL)

(iii) CFL ∩ DCFL = May or may not be CFL

(iv) CFL ∩ Finite = Finite

(v) CFL ∩ infinite = Need not be CFL

3.7 Closure Properties of DCFLs

Operation	Closed / Not Closed
Union ($L_1 \cup L_2$)	×
Intersection ($L_1 \cap L_2$)	×
Complement ($\overline{L}$)	✓
Set difference (L_1-L_2)	×
Concatenation ($L_1.L_2$)	×
Reversal (L^{Rev})	×
Kleene Closure (L) = L*	×
Kleene Plus (L) = L^+	×
Subset (L)	×
Prefix (L)	✓
Suffix (L)	×
Substring (L)	×
Substitution (L)	×
Homomorphism (L)	×
∈ - free Homomorphism h(L)	×
Inverse Homomorphism h^{-1} (L)	✓
quotient (L_1, L_2)	×
Symmetric difference (L_1, L_2)	×
Finite Union	×
Finite Intersection	×
Finite difference	×
Finite concatenation	×
Finite subset	✓
Finite substitution	×
Infinite Union	×
Infinite intersection	×
Infinite difference	×
Infinite concatenation	×
Infinite Subset	×
Infinite Substitution	×
Union with Regular (L U Regular)	✓
L U Regular	✓
L – Regular	✓
Regular – L	✓

Note :

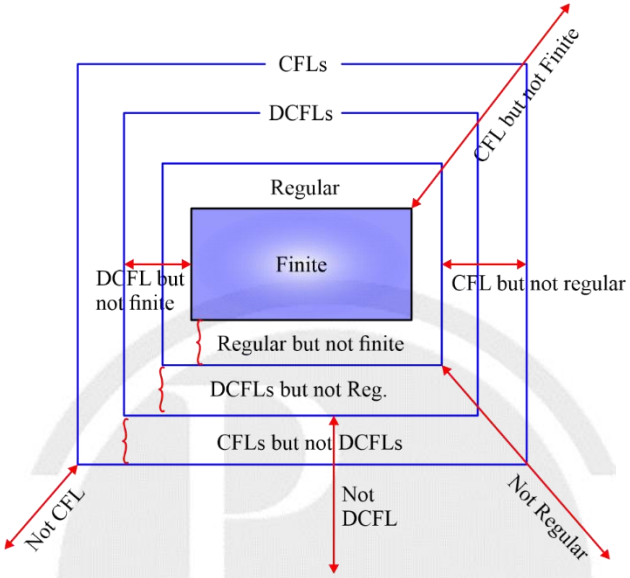
DCFL ∪ Regular : DCFL
DCFL ∩ Regular : DCFL
DCFL - Regular : DCFL
Regular - DCFL : DCFL
DCFL ∪ CFL : CFL (need not be DCFL)
DCFL ∩ CFL : Need not be CFL
DCFL - CFL : Need not be CFL
CFL - DCFL : Need not be CFL
DCFL ∪ Finite : DCFL
DCFL ∩ Finite : Finite
DCFL – Finite : DCFL
Finite - DCFL : Finite

3.8 Comparison of Regular Grammars and CFGs

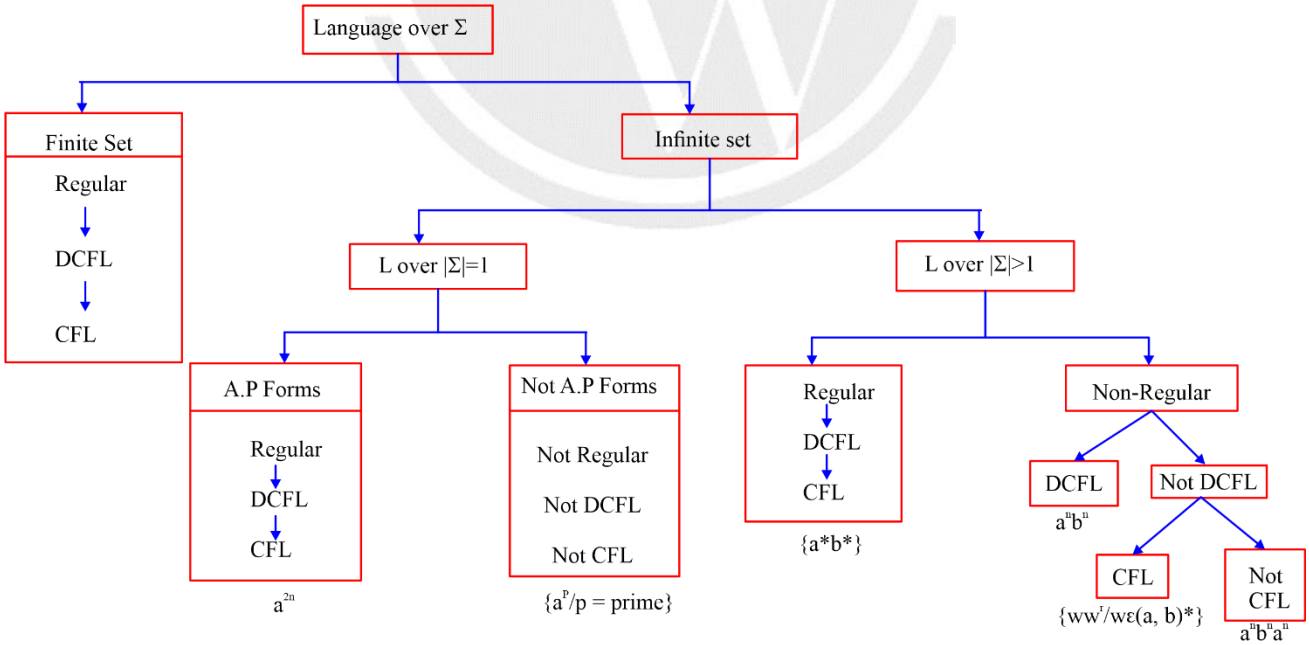
- CFL and CFG both are equivalent. CFL ≅ CFG
- LLG: Left Linear Grammar, RLG: Right Linear Grammar, RG: Regular Grammar, LG: Linear Grammar, CFG: Context Free Grammar.
- Every LLG is RG
- Every RLG is RG
- Every RG is LG
- Every RG is CFG
- Every LG is CFG
- Every RG need not be RLG
- Every RG need not be LLG
- Every LG need not be RG
- Every CFG need not be RLG
- Every CFG need not be LG

3.9 Context Free Languages and DCFLs

I. Comparison of various languages



II. Identification of Regulars, DCFLs, and CFLs



[1] $L = \{a^n b^n c^k \mid n, m, k \geq 0\}$
$= a^* b^* c^*$
= Regular

[2] $L = \{a^n b^n \mid n \geq 0\}$
L = DCFL but not regular
$S \rightarrow aSb \mid \in$

[3] $L = \{a^n b^{2n} \mid n \geq 0\}$
L = DCFL but not regular
$S \rightarrow aSbb \mid \in$

[4] $L = \{a^{2n} b^n \mid n \geq 0\}$
L = DCFL
$S \rightarrow aaSb \mid \in$

[5] $L = \{a^{2n} b^{2n} \mid n \geq 0\}$
L = DCFL
$S \rightarrow aaSbb \mid \in$

[6] $L = \{ a^n b^n c^* \}$ Assume always $n \geq 0$ in all examples
OR
$= \{ a^m b^n c^* \mid m = n\}$
DCFL but not regular

[7] $L = \{ a^n b^* c^n \}$ DCFL

[8] $L = \{ a^* b^n c^n \}$ DCFL

[9] $L = \{ a^m b^n c^* \mid m \neq n \}$
DCFL

[10] $L = \{ a^m b^n c^* \mid m < n \}$
DCFL

[11] $L = \{ a^m b^n c^* \mid m > n \}$
DCFL

[12] $L = \{ a^m b^n c^* \mid c \leq n \}$
DCFL

[13] $L = \{ a^m b^n c^* \mid m \geq n \}$
DCFL

[14] $L = \{ a^m b^n c^{m+n} \mid m, n \geq 0 \}$ is DCFL

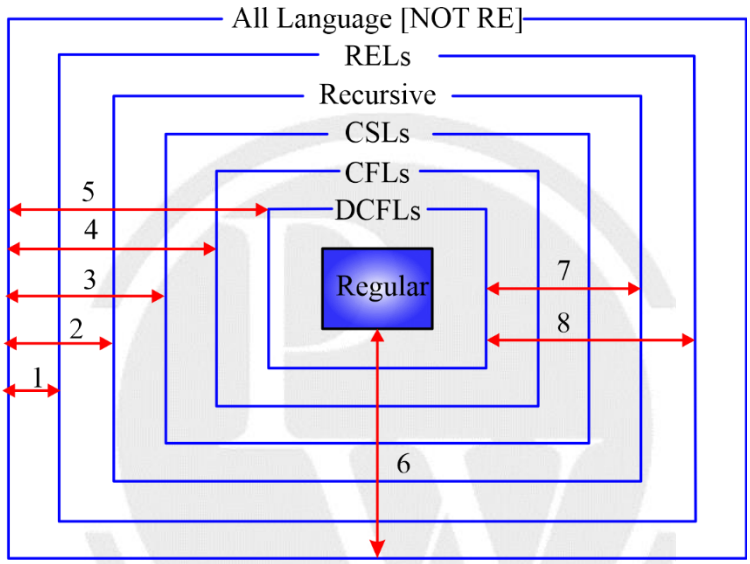
[15] L = { $a^{m+n}b^{n+k}c^{k+m}$ |m, n ≥ 0 } is CFL

[16] L = { ww^r |w belongs to {a, b}* } is CFL but not DCFL

❐❐❐

4 TURNING MACHINE

4.1 Classification of Languages



1. All languages which are not RELs
2. All not recursive languages
3. All not CSLs
4. All not CFLs
5. All not DCFLs
6. All not regulars
7. All recursive languages which are not DCFLs
8. All RELs which are not DCFLs

4.2 There are Two types of TM

(a) DTM (Deterministic TM)
(b) NTM (Non- Deterministic TM)

DTM

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \left\{ \underset{\text{Left}}{L}, \underset{\text{Right}}{R} \right\}$$

NTM

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

4.3 Unrestricted Grammar (UG) and Context Sensitive Grammar (CSG)

Unrestricted grammar (UG) also called as

- RE grammar
- Phase structure grammar

Context sensitive grammar (CSG) is also called as

- Non contracting grammar
- Bound restricted grammar

4.4 Grammars

Left Linear Grammar (LLG): $V \rightarrow VT^* \mid T^*$

Right Linear Grammar (RLG): $V \rightarrow T^*V \mid T^*$

Linear Grammar (LG): $V \rightarrow T^*VT^* \mid T^*$

Context Free Grammar (CFG): $LHS \rightarrow RHS, \quad |LHS| \leq |RHS|$

Unrestricted Grammar (UG): $LHS \rightarrow RHS$

4.5 Equivalence of various TMs

TM ≅ Single tape TM

TM ≅ One-way infinite tape TM

TM ≅ Two-way infinite tape TM

TM ≅ Multi tape and multi head TM

TM ≅ Universal TM

TM ≅ Two stack PDA

TM ≅ Multi stack PDA

TM ≅ FA with two stacks

TM ≅ FA + R/W tape + Bidirectional head

4.6 Restrictions on TM

(1) If TM tape is read only tape, this TM accepts regular.

TM ≅ FA (TM with read only tape)

(2) If TM head is unidirectional then L(TM) = Regular.

(3) If TM tape is read only and unidirectional head then L(TM) = Regular.

(4) If TM always halts then L(TM) is recursive language.

(5) If TM always halts and uses linear bound tape then L(TM) is CSL.

4.7 LBA, HTM and TM

HTM: It is a TM that always halts for every input.
TM: It halts for every valid string and for invalid strings either halts at "non final state" or "never halts".
LBA: It is HTM but length of the tape we use linearly bounded.

- LBA accepts CSL languages.
- HTM accepts recursive languages.
- TM accepts Recursive Enumerable (RE) languages.

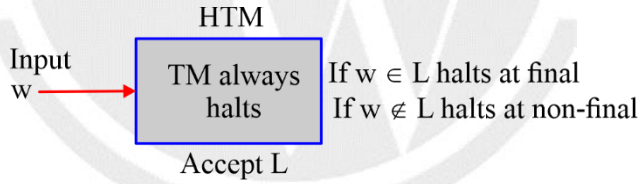
4.8 Recursively Enumerable Language

It is also called as:

- Enumerable language
- TM recognizable language
- TM Enumerable language
- Partially decidable language
- Semi-decidable language

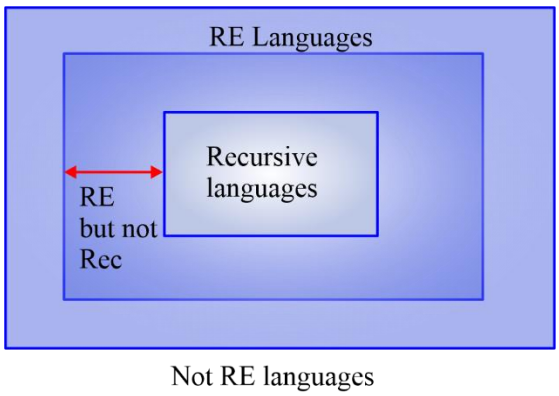
4.9 Recursive Language

- Recursive language is acceptable by HTM, and hence acceptable by TM.
- Recursive also called as decidable language.
- Recursive also called as Turing decidable language.
  If TM always halts, then TM is called as HTM.



4.10 Difference between Recursive and REL

- All Recursive languages are RE languages.



Note :

1. Union:
   - REL ∪ Finite ⇒ REL
   - REL ∪ Regular ⇒ REL
   - REL ∪ CFL ⇒ REL
   - REL ∪ Recursive ⇒ REL
   - $REL_1 \cup REL_2 \Rightarrow REL$
2. Intersection:
   - REL ∩ Finite ⇒ REL (Finite)
   - REL ∩ CFL ⇒ REL
   - REL ∩ Rec ⇒ REL
   - $REL_1 \cap REL_2 \Rightarrow REL$

4.11 Closure Properties of Recursive languages

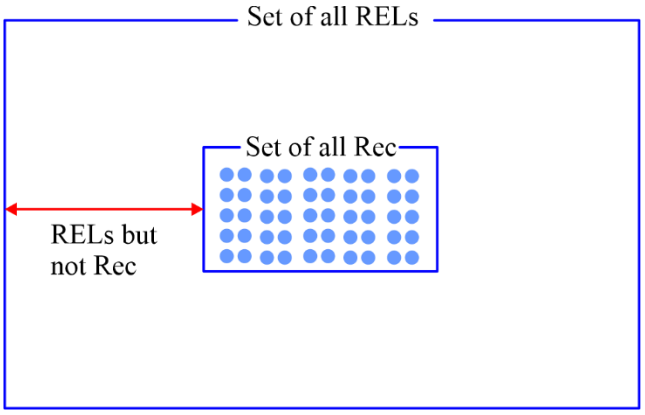
I. The following operations are not closed for recursive languages:

- Subset
- Substitution
- Homomorphism
- Finite substitution
- Infinite union
- Infinite intersection
- Infinite concatenation
- Infinite difference
- Infinite substitution

II. The following operations are closed for recursive languages:

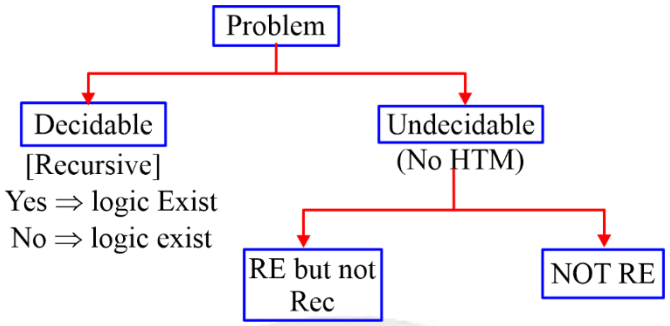
- Complement
- Difference
- Finite difference
- Infinite followed by ∪, ∩, -, •, ⊆, f
- Remember not closed operations

4.12 Complement of Recursive Vs Complement of REL



- Complement of Recursive set is Recursive.
- Complement of REL is either Recursive or non-REL.
- Complement of REL never be "REL which is not recursive".

4.13 Decidable and Undecidable



4.14 Decidable Vs Undecidable, RE Vs Not RE, Countable Vs Uncountable

[1] Decidable and Undecidable

↓ ↓

HTM exist (Rec language) | HTM not exist (not Rec language)

[2] RE and Not RE

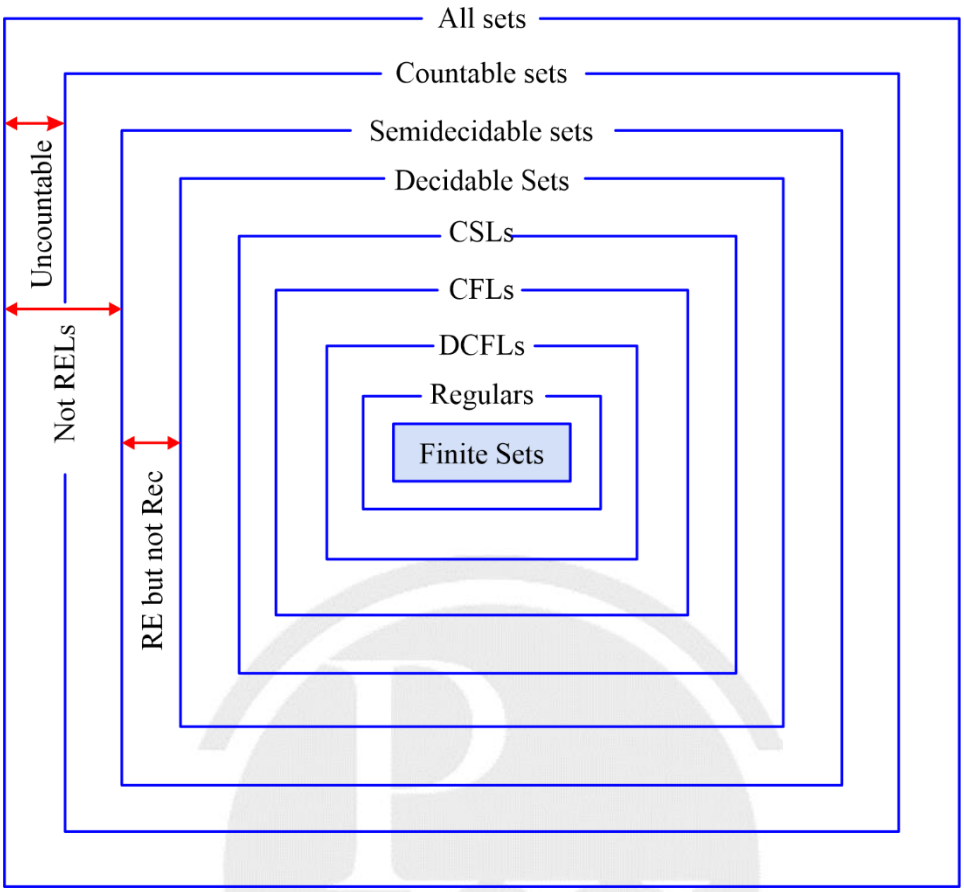
↓ ↓

TM exist | TM not exist

[3] Countable set and Uncountable set

↓ ↓

X is countable set iff $X \to f(X)$ is bijective where f(X) is known countable set. | (X is not countable set)



Note :

- If problem p is decidable then $\overline{p}$ is also decidable.
- If problem p is Undecidable then $\overline{p}$ is also UD.
- If problem p is RE but not recursive then $\overline{p}$ is not RE.
- If problem p is not RE then $\overline{p}$ is either "Not RE" or "RE but not Recursive".

4.15 Decision Properties Table

- D: Decidable
- UD: Undecidable

	FA	DPDA	PDA	LBA/HTM	TM
H (Halting)	D	D	D	D	UD
M (Membership)	D	D	D	D	UD
E_m (Emptiness)	D	D	D	UD	UD
F (Finiteness)	D	D	D	UD	UD

T (Totality)	D	D	UD	UD	UD
E_q (Equivalence)	D	D	UD	UD	UD
D (Disjoint)	D	UD	UD	UD	UD
S (Set Containment)	D	UD	UD	UD	UD

4.16 Decidable problem for DFA / NFA/ FA/ Regular

(1) Halting problem for FA / Reg/ DFA / NFA is decidable.
(2) Non-halting problem for FA / Reg/ DFA / NFA is decidable.
(3) Membership problem for FA / Reg/ DFA / NFA is decidable.
(4) Non-membership problem for FA / Reg/ DFA / NFA is decidable.
(5) Emptiness for FA / Reg/ DFA / NFA is decidable.
(6) Non-emptiness problem for FA / Reg/ DFA / NFA is decidable.
(7) Fitness problem for FA / Reg/ DFA / NFA is decidable.
(8) Non-fitness problem for FA / Reg/ DFA / NFA is decidable.
(9) Totality problem for FA / Reg/ DFA / NFA is decidable.
(10) Non-totality problem for FA / Reg/ DFA / NFA is decidable.
(11) Equivalence problem for FA / Reg/ DFA / NFA is decidable.
(12) Non-equivalence problem for FA / Reg/ DFA / NFA is decidable.
(13) Disjointness problem is decidable for FA / Reg/ DFA / NFA.
(14) N0n-disjointness problem is decidable for FA / Reg/ DFA / NFA.
(15) Set containment problem for FA / Reg/ DFA / NFA is decidable
(16) Non-set containment problem for FA / Reg/ DFA / NFA is decidable.

4.17 Decidable problems for CFLs/DCFLs

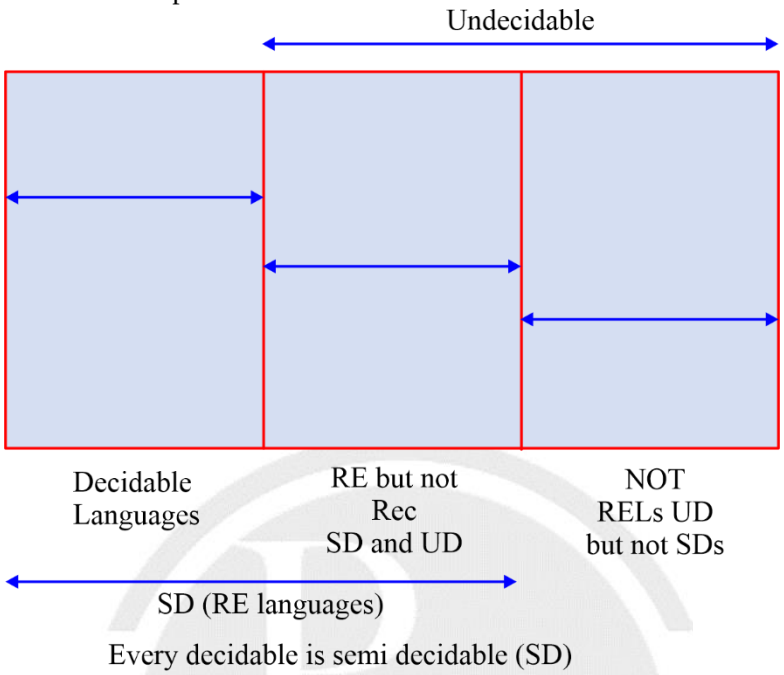
	Problems	DCFLs	CFLs
	Halting	D	D
	Non-Halting	D	D
CYK algo	Membership	D	D
CYK algo	Non-membership	D	D
Simplification algo	Emptiness	D	D
Simplification algo	Non-emptiness	D	D
Dependancy graph	Finiteness	D	D
Dependancy graph	Non-finiteness	D	D
	Totality	D	UD
	Non-totality	D	UD
	Equivalence	D	UD
	Disjointness	UD	UD
	Non-disjointness	UD	UD
	Set containment	UD	UD
	Non-set containment	UD	UD

4.18 Decidability problems for Recursive languages

	Problems	**Recursive**
1.	Halting	D
2.	Non-Halting	D
3.	Membership	D
4.	Non-membership	D
5.	Emptiness	UD [Not REL]
6.	Non-emptiness	UD [RE but not Rec] [SD but UD]
7.	Finiteness	UD [Not RE]
8.	Non-finiteness	UD [Not RE]
9.	Totality	UD [Not RE]
10.	Non-totality	UD [RE but not Rec] [SD but UD]
11.	Equivalence	UD [Not RE]
12.	Non-equivalence	UD [RE but not Rec]
13.	Disjointness	UD [Not RE]
14.	Non-disjointness	UD [RE but not Rec]
15.	Set containment	UD [Not RE]
16.	Non-set containment	UD [RE but not Rec]

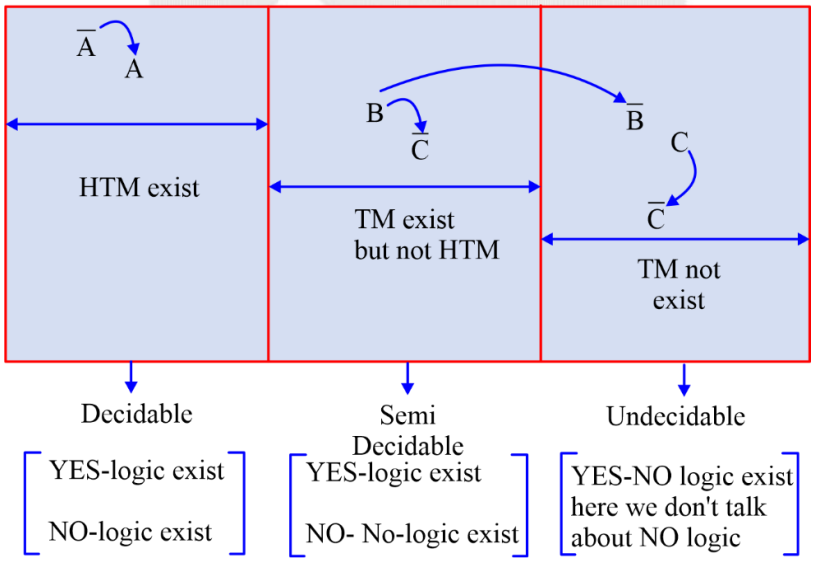
4.19 Classification of Languages based on Decidability

All RE languages can be classified into 3 important classes.



Every decidable is semi decidable (SD)

4.19.1 Decidability Vs Turing Machine



4.20 Decidable languages

(1) Finite set $\Rightarrow$ Decidable

(2) $\sum = \{a, b\} \Rightarrow$ Decidable

(3) $\sum \cong$ Set of finite number of symbols $\Rightarrow$ Decidable

(4) $\sum^*$ over alphabet $\sum = \{a, b\} \Rightarrow$ Decidable

(5) $\{M \mid M$ is DFA, M accepts $ab\} \Rightarrow$ Decidable

(6) {TM | Number of states in TM= 2} ⇒ Decidable

(7) {TM | TM reaches state q within 100 steps} ⇒ Decidable

(8) { TM | TM accepts REL} ⇒ Decidable

❐❐❐❐

GATE Exam 2024?

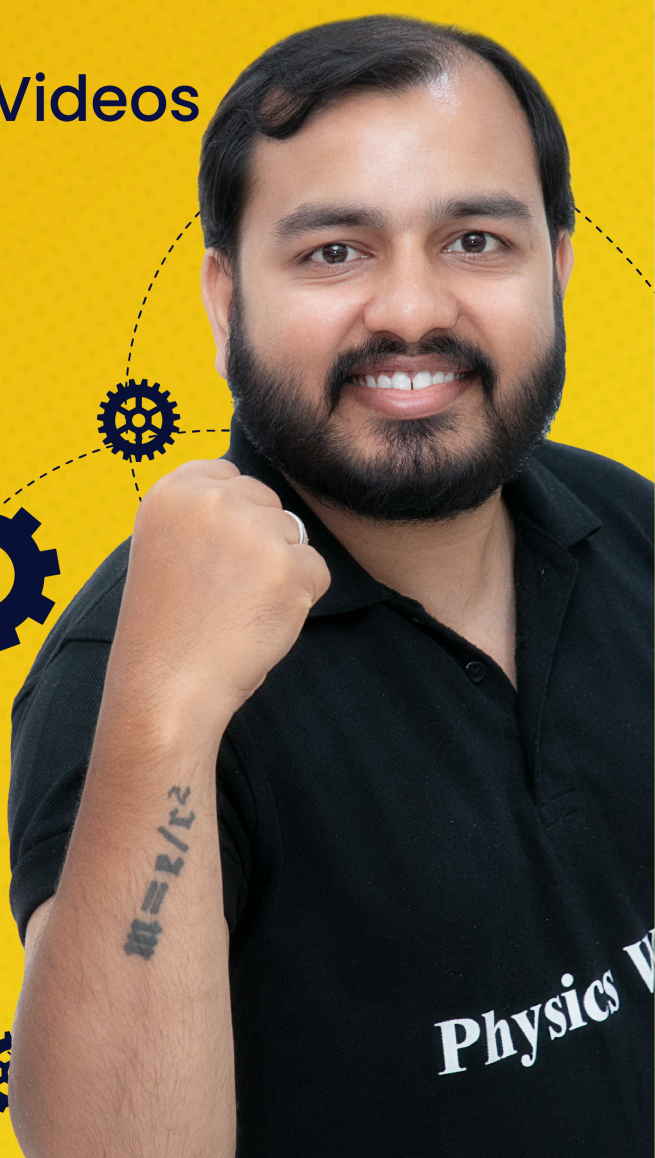
PARAKRAM BATCH

ME | CE | EE | EC | CS & IT

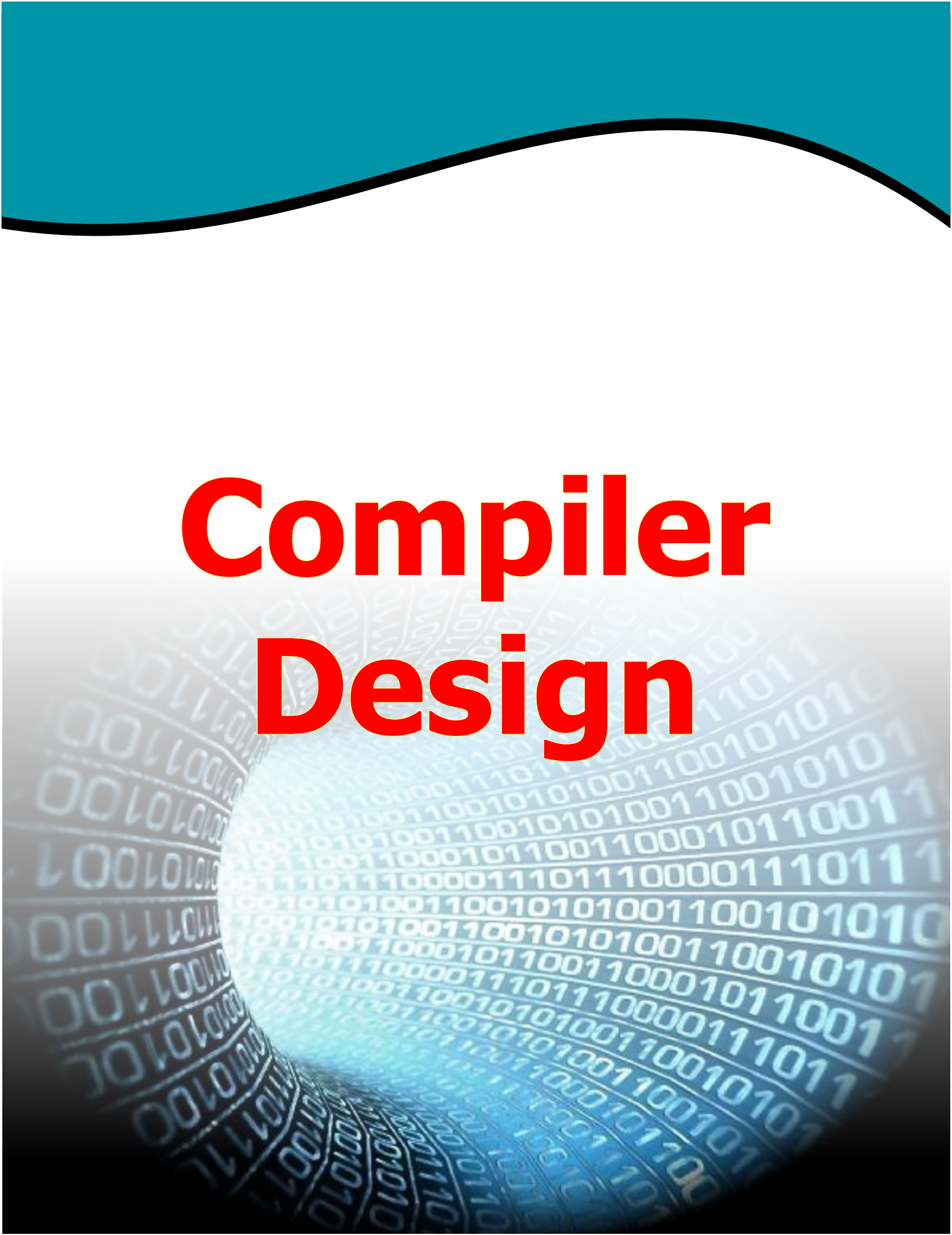
- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in ***English***

Weekday & Weekend Batches Available

JOIN NOW!



Compiler Design



Compiler Design

INDEX

1 INTRODUCTION AND LEXICAL ANALYSIS

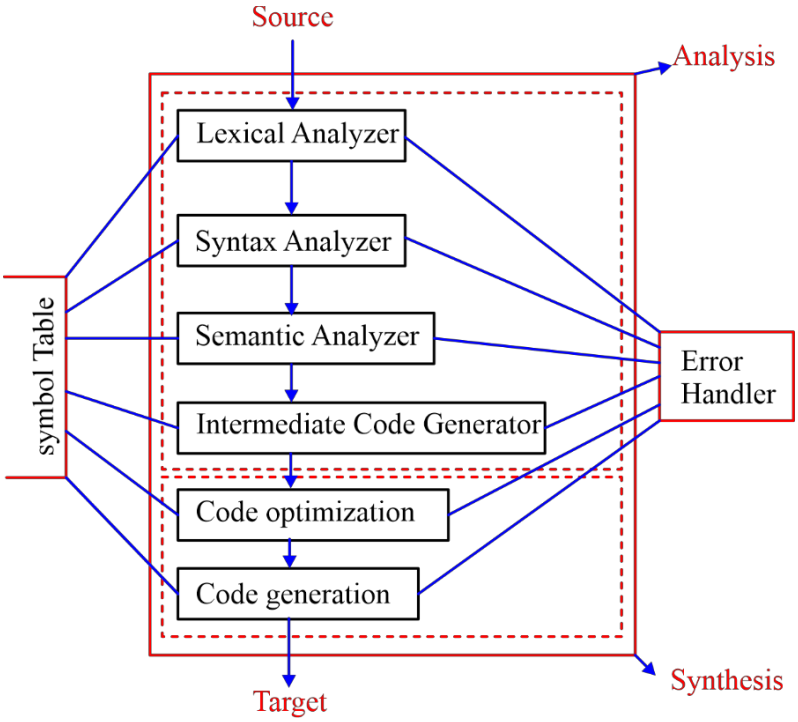
1.1 CD Introduction

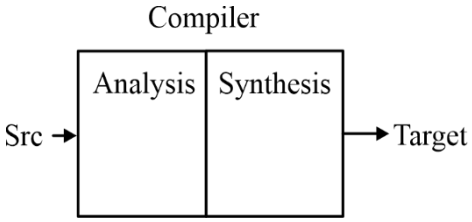
1.1.1 Definition

Convert High Level Language to Low Level Language.

- High Level Language can perform more than one operation in a single Statement.
- lol Level Language can perform atmost one operation in a statement.

1.2 Analysis and Synthesis model of Compiler





- There are 6 phases of the Compiler

1. **Lexical Analyzer:**

- Program of DFA, it checks for spelling mistakes of program.
- Divides source code into stream of tokens.

2. **Syntax Analyzer:**

- Checks grammatical errors of the program (Parser).
- Parser is a DPDA.

3. **Semantic Analyzer:**

Checks for meaning of the program.

Example:

Type miss match, stack overflow

4. **Intermediate Code Generation:**

- This phase makes the work of next 2 phases much easier.
- Enforces reusability and portability.

5. **Code optimization:**

- Loop invariant construct
- Common sub expression elimination
- Strength Reduction
- Function in lining

Deadlock elimination

6. **Symbol Table:**

(1) Data about Data (Meta data)

(2) Data structure used by compiler and shared by all the phrase.

❑❑❑

2 PARSING

2.1 CD - Grammar

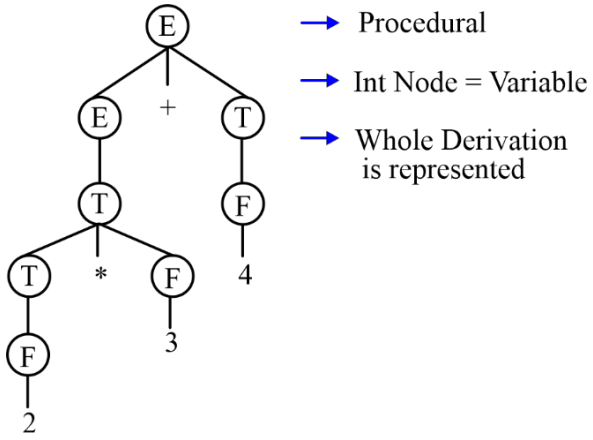
→ In compiler we only use: Type – 2 (CFG) and Type – 3 Regular grammar.
→ Compiler = Program of Grammar
→ Compiler = Membership algorithm
• Every programming Language is Context Sensitive Grammar (Context Sensitive Language)

2.1.1 Parse Tree and Syntax Tree

G: $E \to E + T / T$ $E \to E + T \to T + T \to T * F + T \to F * F + T \to 2 * F + T \to 2 * 3 + T \to 2 * 3 + F$
$T \to T * F / F$

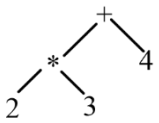
$E \to 2 * 3 + 4$

Parse Tree:



→ Procedural
→ Int Node = Variable
→ Whole Derivation is represented

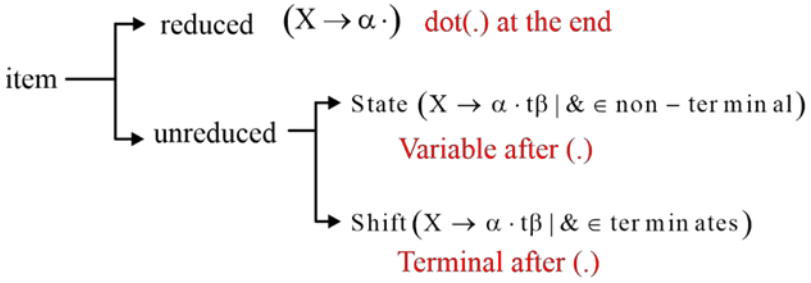
Syntax Tree:



→ non procedural
→ Int node = Operation
→ Discuss Concept only

- To check to priority / Associativity:
  Randomly derive till you have enough operators, then check which one is done first.
- If priority of 2 operators is same and both Left and Right associative → **Ambiguous Grammar** [Useless]

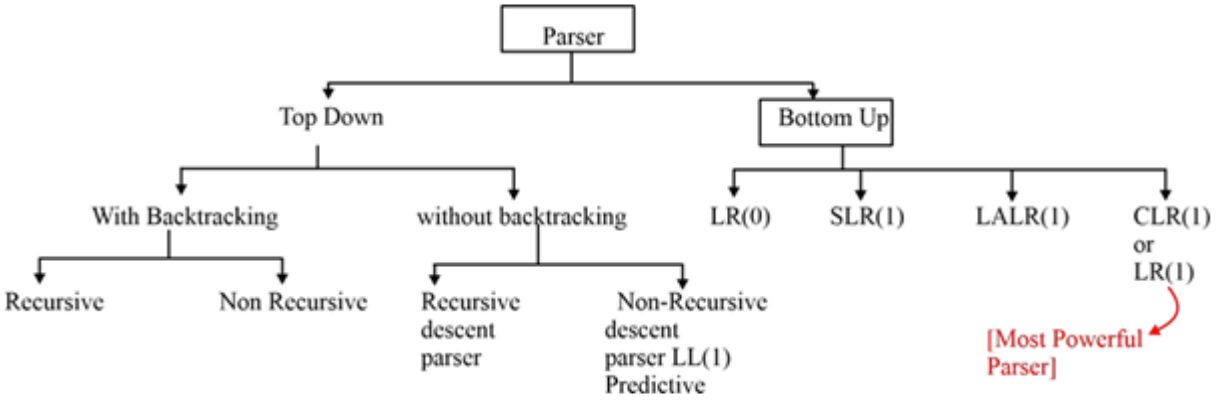
2.1.2 Types of Parsers in Bottom Up: [CD - Parser]

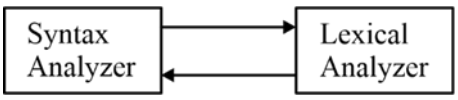


2.2 CD-Syntax Analysis / Parsing

Grammatical Errors are checked by the help of parsers.

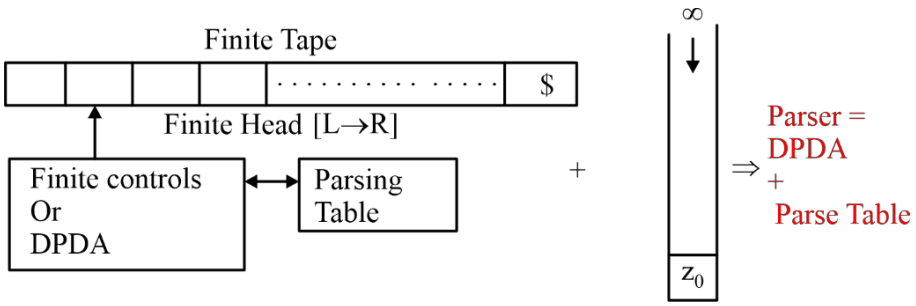
- Parsers are basically DPDA





- All of these parsers are table driven.

2.2.1 Mathematical Model of Parser



- Parsers generate Parse Tree, for a given string by the given grammar.

2.2.2 Top-Down Parser (LL (1))

- It uses LMD and is equivalent to DFS in graph.

2.2.3 Algorithm to construct Parsing Table

1. Remove Left Recursion if any.
2. Left Factor [Remove Common Prefix]
3. Find first and follow set
4. Construct the table.

- If we increase the look ahead symbol:

2.2.4 Removal of common Prefix: (Left Factor)

1. $S \rightarrow a \mid a\ b \mid a\ A\ S \rightarrow a\ Y$

   $Y \rightarrow \in \mid b \mid A$

2. $A \rightarrow a\ b\ A \mid a\ A \mid b$

   $A \rightarrow a \times \mid b \quad A \rightarrow a \times \mid b$

   $X \rightarrow b\ A \mid A \quad X \rightarrow b\ A \mid a\ x \mid b$

   $A \rightarrow a\ X \mid b$

   $X \rightarrow a\ X \mid b$

   $Y\ Y \rightarrow A \mid \in$

2.2.5 First and Follow

- First Set $\rightarrow$ extreme Left terminal from which the string of that variable starts.

  $\rightarrow$ it never contains variable, but may contain '$\in$'.

  $\rightarrow$ we can always find the first of any variable.

- Follow set $\rightarrow$ Follow set contains terminals and $.

  It can never contain variable and '$\in$'.

  How to find follow set?

1. Include $ in follow of start variable.
2. If production is of type $\rightarrow$

   $A \rightarrow \alpha B \beta$ $\alpha\beta \rightarrow$ strings of grammar symbol].

   follow (B) = first (β).

   If, $\beta \rightarrow \in$, i.e. $A \rightarrow \alpha B$, then follow(B) = follow(A).

- Productions like: $A \rightarrow \alpha A$ gives no follow set.

2.2.6 Example of first and follow set

1. S → AB | CD
   A → aA | a
   B → bB | b
   C → cC | c
   D → dD | d

	First	Follow
S	a, c	$
A	a	b
B	b	$
C	c	D
D	D	$

2.2.7 Entity into Table: Top Down

1. No of Rows = number of unique variable in Grammar.
2. No of columns = [Terminals + $]
3. For a Variable (Row) fill the column (Terminal) if its P, there in its first with the production required.
4. If ∈ is in first put V → ∈ under $ and its follow.

- If any cell has multiple items, then its not possible to have LL (1) Parser, since that will be ambiguous.
- In top down we do: Derivation
  In Bottom up we do: Production.

Question: Construct LL (1) Parsing Table for the given Grammar: E → E + T | T; T → T * F | F; F → (E) | id; ⇒ G_0

⇒ Removing Left Recursion:

E → TE'
E'→ + TE' | ∈
T → FT' G_1
T'→ *FT'| ∈
F → (E) | id

	First	Follow
E	C, id	$,)
E'	+, ∈	$,)
T	C, id	+, $,)
T'	*, ∈	+, $,)
F	C, id	*, +, $,)

- Left Factoring not Required:
  Construction of Table: [LL (1)]

	+	*	(	)	id	$
E	Error	Error	E→TE'	Error	E→ TE'	Error
E'	E'→TE'	Error	Error	E'→∈	Error	E'→∈
T	Error	Error	T→FT'	Error	T→ FT'	error
T'	T'→∈	T'→*FT'	Error	T'→∈	Error	T'→∈
F	Error	Error	F→(∈)	Error	F→ id	error

- Since for G_1, Table constructed with no multiple entries. Hence successfully completed. Hence G_1 is LL (1)

Question: Construct LL (1) Parsing Table for the following Grammar:
S → L = R | R; L → * R | id; R → L ⇒ G_0

Solution: Left Factoring:

$$S \to L = R \mid L \quad L \to *R \mid id \quad R \to L \Rightarrow \begin{matrix} S \to LX \\ \times \Rightarrow = R \mid \in \\ L \to *R \mid id \\ R \to L \end{matrix} \Rightarrow G_1$$

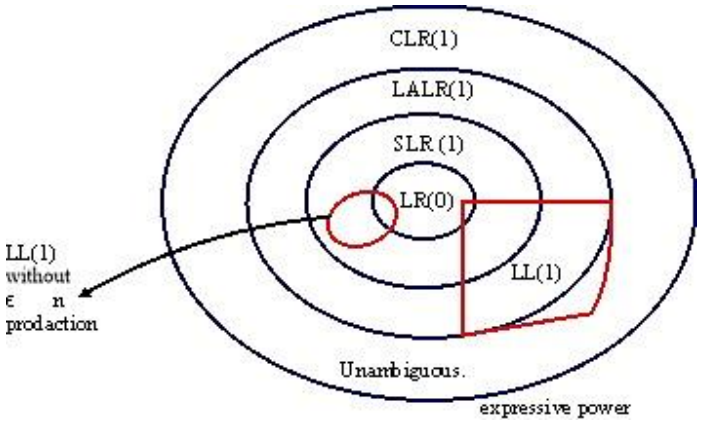
	First	Follow
S	*, id	$
×	=, ∈	$
L	*, id	$
R	*, id	$

Construction of Table:

	*	=	Id	$
S	S→LX	Error	S→L X	Error
L	L→*R	Error	L→id	Error
R	R→L	Error	R→L	Error
×	Error	X→ = R	Error	X→∈

- G_1 is a LL (1) Grammar.

2.2.8 Hierarchy of Parsers: [for ∈- free Grammar]



- For ∈ - producing grammars every LL (1) may not be LALR (1).

Note:
We can't construct any parser for ambiguous grammar. Except: Operator precedence, parser possible for some ambiguous grammar.

Example:

G: S → a S a | b S b | a | b (Unambiguous but no parser) L(G) = ω(a + b) ω R
(Odd palindrome)

- Every RG is not LL (1) as it may be ambiguous, or Recursive or Common Prefix.
- Parsers exists only for the grammar if its Language is DCFL.
- There are some grammars whose Language is DCFL but no parser is possible for it.

2.4.9 Operator Precedence Grammar

Format: (1) No 2 or more variable side by side
(2) No $\in$ production.

Example:

$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid id$
O.G.

or

$E \to E + E$
$E \to E * E$
$E \to a / b$
O.G

or

$S \to aSa \mid bSb \mid a \mid b$
O.G

Or

$S \to AB$
$A \to aA \mid \in$
$B \to bB \mid \in$

Not O.G

2.2.10 Checking LL (1) without Table

$A \to \alpha_1 \mid \alpha_1 \mid \alpha_1$ then $\to$

first $(\alpha_1) \cap$ first $(\alpha_2) = \varnothing$
first $(\alpha_1) \cap$ first $(\alpha_3) = \varnothing$
first $(\alpha_2) \cap$ first $(\alpha_3) = \varnothing$

$A \to \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \in$

first $(\alpha_1) \cap$ first $(\alpha_2) = \varnothing$
first $(\alpha_1) \cap$ first $(\alpha_3) = \varnothing$
first $(\alpha_2) \cap$ first $(\alpha_3) = \varnothing$
follow $(A) \cap$ first $(\alpha_1) = \varnothing$
follow $(A) \cap$ first $(\alpha_2) = \varnothing$
follow $(A) \cap$ first $(\alpha_3) = \varnothing$

2.2.11 Bottom-UP Parser

- It uses RMD in reverse and has no problem with:
  (a) Left recursion (b) Common Prefix
  SLR (1) LR (0) CLR (1) LR (1)
  LR (0) items LALR (1) items

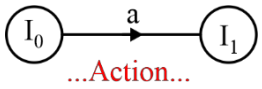
  LR (1) = LR(0) +1 Lookahead
- No parser possible for ambiguous grammar.
- There are some unambiguous grammars for which, there are no Parser.

2.2.12 Basic Algorithm for Construction

- Augment the grammar and expand it, and give numbers to it
- Construct LR (0) or LR (1) item set.
- From that fill the entries in the Table Accordingly.

2.2.13 Types of Entries

(1) Shift Entries: Transitions or Terminals



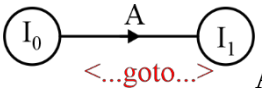
Entry:

	a
I_0	S_1

$a \in$ Terminal I_0, I_1: Item sets.

- Shift entries are some for all Bottom – up Parser.

(2) State Entry: Transition or non – terminal (variable)



$A \rightarrow$ Variable

Entry:

	A
I_0	1

- Same for all Bottom-up Parser.

(3) Reduce Entity: done for each separate production in the item set of type:

$i > \times \rightarrow \alpha$

where, $i \rightarrow$ Production Number

$\times \rightarrow$ Producing Variable

$\alpha \rightarrow$ Grammar String

(a) LR (0) Parser:

Put Ri in every cell of the set-in action table **(ALL).**

(b) SLR (1) Parser:

Put Ri only in the follow(x) form the Grammar. **(Follow(x))**

(c) LALR (1) and CLR (1):

Put Ri only in the look ahead of the production **(Lookaheads)**

2.2.14 Conflicts

LR (0) Parser:

SR: Shift Reduce Conflict

$S \rightarrow \quad X \rightarrow \alpha \cdot t\beta \quad \Rightarrow$ then SR

$R \rightarrow Y \rightarrow \delta \cdot$ conflict

RR: Reduce Reduce conflict

$R \rightarrow X \rightarrow \alpha \cdot \quad \Rightarrow$ then RR

$R \rightarrow X \rightarrow \beta \cdot$

SLR (1) Parser:

SR:

$S \rightarrow X \rightarrow \alpha \cdot t\beta$
$R \rightarrow Y \rightarrow \delta \cdot$
$t \in$ follow (Y)
} $ ⇒then SR, conflict

RR:

$X \rightarrow \alpha \cdot$
$Y \rightarrow \beta \cdot$
} $ ⇒then RR

Follow (x)n follow (y) $\neq \varnothing$

LALR (1) and CLR (1): Same as SLR (1), but instead

SR:

$X \rightarrow \alpha \cdot t\, \beta \cdot L_1$
$Y \rightarrow \delta \cdot L_2$
} $⇒then SR, conflict

$T \in L_2$

RR:

$X \rightarrow \alpha \cdot, L_1$
$Y \rightarrow \beta \cdot, L_2$
} $ then RR, conflict

$L_1 \cap L_2 \neq \varnothing$

2.4.15 Inadequate Static

A static having ANY conflict is called a conflicting state or independent state.

Note:
The state S' → S. or S' → S.,$ is accepted state, and this is not a reduction.

- The only difference between CLR (1) and LALR (1) is that, the states with the similar items, but different Lookaheads are merged together to Reduce space.

Important Points

1. If CLR (1) doesn't have any conflict, then conflict may or may not arise after merging in LALR (1).
2. If LALR (1) has SR – conflict, then we can conclude that CLR (1) also has SR – Conflict.
3. LALR (1) has SR – conflict if and only if CLR (1) also has SR.

- We can construct parser for every unambiguous regular grammar: [CLR (1) Parser].

	L Left to right scan of i/p	R Using reverse RMD	(0) No Lookhead	
S Simple	L L to R Scan	R Using Reverse RMD	(1) Lookahead	
L Look	A Ahead	L L to R Scan	R Revers RMD	(1) Lookahead
C Canonical	L L to R Scan	R reverse RMD	(1) Lookahead	

Very Important Point:

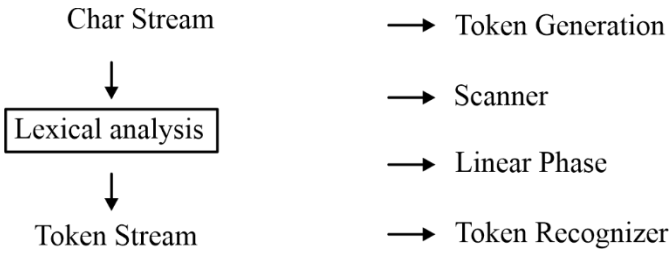
LALR (1) Parser can Parse non LALR (1) grammar, when only non-SR – Conflict by favouring shift over reduce.

Example: $E \rightarrow E + E \mid E * E \mid id \mid 2 + 3 * 5 \Rightarrow E + E \cdot * 5$

❑❑❑

3 LEXICAL ANALYSIS

3.1 Lexical Analysis



3.1.1 Definition

Scan the whole program, char by char and produces the corresponding token.

- Also produces /Lexical Errors (if any).
- Functions of Lexical Analyzer.

(1) Scans all the character of the program.
(2) Token recognizer.
(3) Ignores the comment & spaces.
(4) Maximal Munch Rule [Longest Prefix Match].

Note:
The Lexical Analyser uses, the Regular Expression.
Prioritization of Rules.
Longest Prefix match .

- Lexeme → smallest unit of program or Logic.
- Token → internal representation of Lexeme.

3.1.2 Types of Token

(1) Identifier
(2) Keywords
(3) Operators
(4) Literals/Constants
(5) Special Symbol

3.1.3 Token Separation

(1) Spaces
(2) Punctuation

3.1.4 Implementation

- LEX tool ⇒ Lex.yy.e

- All identifiers will have entry in symbol Table/ LA, gives entries into the symbol Table.
  Regular Expression → DFA → Lexical Analyzer

3.1.5 Find number of Tokens

(1) void main () {
Printf("gate");
}
{11 Tokens}
(2) int x, *P;
x = 10; p = &x; x++;
[18 Tokens]
(3) int x;
x = y;
x == y;
[11 Tokens]
(4) int 1x23; [Lexical Error]
(5) char ch = 'A';
[5 Token]
(6) char ch = 'A;
Lexical Error.
(7) char *p = "gate";
[6 Tokens]
(8) char * p = "gate;
Error.
(9) int x = 10;
/* comment x = x + 1; ERROR
x = x + 1; [11 Tokens]

❑❑❑

4 SYNTAX DIRECTED TRANSACTION

4.1 Syntax Directed Transaction

CFG + Transition
Syntax + Transition
} SDT

SDT: CFG + Transition → 1) Meaning 2) Semantic

4.1.1 Application of SDTL

(1) Used to perform Semantic Analysis.
(2) Produce Parse Tree.
(3) Produce intermediate representation.
(4) Evaluate an expression.
(5) Convert infix to prefix or postfix.

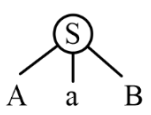
Example: $S \rightarrow S_1S_2$ [S.count = S_1.count + S_2.count]
$S \rightarrow (S_1)$ [S.count = S_1.count + 1]
$S \rightarrow \in$ [S.count = 0]

- Count is an attribute for non–terminal

4.1.2 Attributes

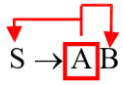
(1) Inherited Attribute
(2) Synthesized Attribute

$S \rightarrow AaB$



S is parent to A, a, B
A is child to S
A is Left Sibling to a, B

(1) Inherited Attribute: (RHS)

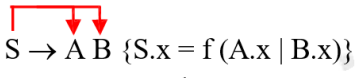


$S \rightarrow AB$ {A.x = f (B.x | S.x)}

- The computation at any node (non-terminal) depends on parent or siblings (S).

- In above Example, x is inherited attribute.

(2) Synthesized Attribute: (LHS)



x is synthesized attribute.

The computation at any node (non-terminal) depends on children.

4.1.3 Identifying Attribute Type

- Always check every Translation.

1) $D \rightarrow T : L$; {L.Type = T.Type} inherited
   $L \rightarrow L1, id$; {L1.Type = L.Type} inherited
   $L \rightarrow id$
   $T \rightarrow integer$ {T.Type = int} synthezised
   — Type is neither inherited nor synthesized

2) $E \rightarrow E1 + T$ {E.val = E1.val + T.val} synthesized
   $E \rightarrow T$ {E.val = T.val} Synthesized
   $T \rightarrow T1 – F$ {T.val = T1.val – F.val} Synthesized
   $F \rightarrow id$ {F.val = id.val} synthesized
   — Val is synthesized attribute

3) $S \rightarrow AB$ {Aa = B.x; S.y = A.x} x is inherited | y is synthesized
   $A \rightarrow a$ {A.y = a} y is synthesized
   $B \rightarrow b$ {B.y = a.y} y is synthesized
   — x ⇒ inherited
   y ⇒ synthesized

4) $D \rightarrow T L$ {L.in = T.type} inherited(in)
   $T \rightarrow int$ {T.type = int} /synthesized
   $L \rightarrow id$ {Add type(id.entry,L.in)}
   — in ⇒ inherited
   type ⇒ synthesized

4.1.4 Syntax Directed Definitions (SDDs): (Attribute Grammar)

(1) L-Attributed Grammar

→ Attribute is synthesized or restricted inherited. (1) Parent 2) Left sibling only).

→ Translation can be appended anywhere is RHS of production.

→ Example: $S \rightarrow AB$ {A.x = S.x + 2}

or, $S \rightarrow AB$ {B.x = f (A.x | S.x)}

or, $S \rightarrow AB$ {S.x = f (A.x | B.x)}

→ Evaluation: In Order (Topological).

(2) S-Attributed Grammar:

→ Attribute is synthesized only.

→ The transaction is placed only at the end of production.

→ Example: $S \rightarrow AB$ {S.x = f (Ax | Bx)}.

→ **Evaluation:** Reverse RMD (Bottom-Up Parsing).

4.1.5 Identify SDD

(1) $E \rightarrow E_1 + E_2$ {E.type = if (E_1.type = = int & & E.type = = int) then int} Synthesizer else type – error.

$E \rightarrow$ id {E.type = Lookup (id.entry)} synthesizer.

- type is synthesized, hence S-attribute and also L-attributed Grammar.
- Every S-attributed Grammar is also L-attributed Grammar.
- For L-attributed Evaluation, use the In-order of annotated Parser Tree.
- For S-attributed, reverse of RMD is used.

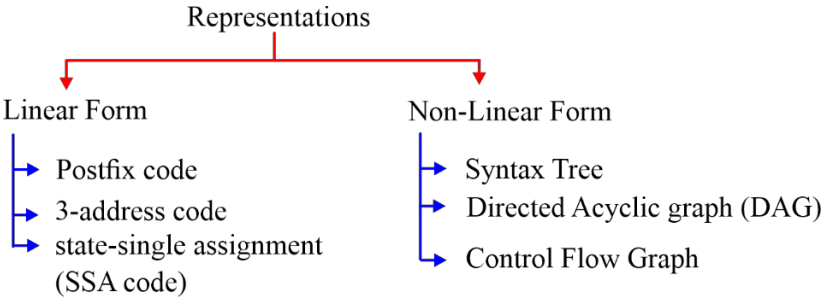
→ find RMD Order.

→ Consider its reverse.

❑❑❑

5 INTERMEDIATE, CODE OPTIMIZATION

5.1 Introduction



Example Expression:

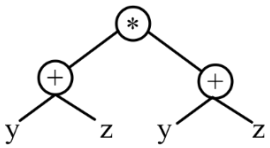
$(y + z) * (y + z)$

Post fix → $\boxed{yz + yz + *}$

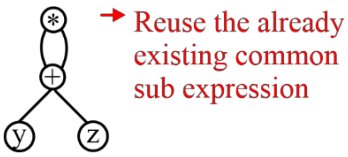
SSA → $\boxed{\begin{aligned} t_1 &= y + z \\ t_2 &= t_1 * t_2 \end{aligned}}$ ⇒f_1 and f_2 cannot be reassigned

3AC → $\boxed{\begin{aligned} t_1 &= y + z \\ t_2 &= t_1 \times t_2 \end{aligned}}$

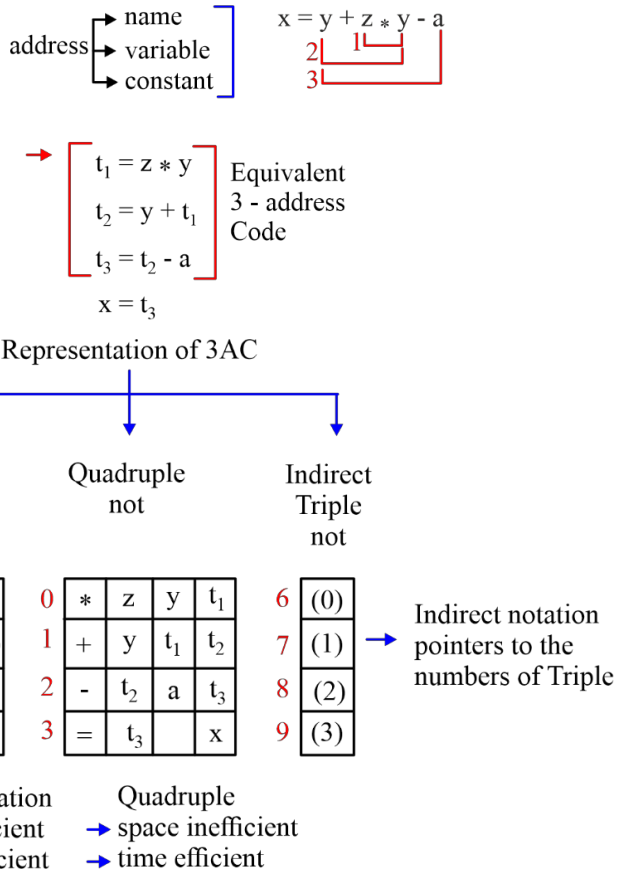
Syntax Tree →



DAG →



3-address Code: Code in which, at most 3 addresses. [including LHS]



- 3AC done using operator precedence.

Find minimum number of variables required in equivalent 3AC:

1.

$x = u - t$ $\qquad x * y \Rightarrow x * (t - z) \Rightarrow (y + w) * (t - z)$

$y = x * v \qquad \Rightarrow ((x * v) + w) * (t - z)$

$x = y + w \qquad \Rightarrow (((u - t) * v) + w) * (t - z)$

$y = t - z$

$y = *x * y$

7 variables

$$\begin{bmatrix} u = u - t & u = t - z \\ v = u * u & z - w * v \\ w = v + w & \end{bmatrix} \Rightarrow 5 \text{ variables only}$$

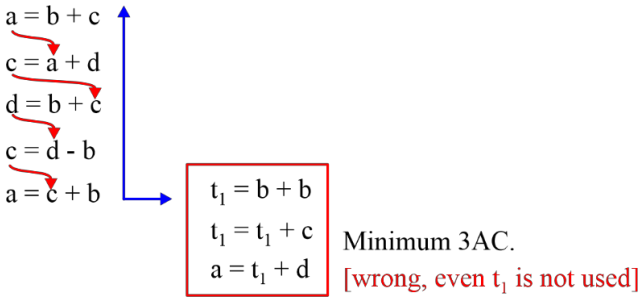
$\Rightarrow$ Evaluating the expression:

a $\Rightarrow c + b \Rightarrow d - b + b \Rightarrow b + c - b + b$

$\Rightarrow b + a + d - b + b$

$\Rightarrow b + b + c + d - b + b$

$\Rightarrow b + b + c + d$



∴ Minimum:

$$\left\{\begin{matrix} b = b + b \\ c = b + c \\ a = c + d \end{matrix}\right\} \Rightarrow \text{only 3 variables [most optimal]}$$

5.1.1 Static Single Assignment Code: (SSA Code)

Every variable (address) in the code has single assignment [single meaning] + 3AC.

1. x = u - t
   y = x * u
   x = y + w
   y = t - z
   y = x * y

   Find SSA?

⇒ [u, t, v, w, z] are already assigned so we can't use them.

Equivalent SSA Code:

x = u – t y = x * v

p = y + w

q = t – x

r = p * q

in use: x, y, p, q, r ⇒ additional

∴ Total Variable ⇒ 10.

2. p = a - b
   q = p * c
   p = u * v
   q = p + q

⇒ [a, b, c, u, v] are already assigned,

Equivalent SSA Code:

p = a – b

q = p * c

p_1 = v * u

$q_2 = p_1 + q$

in use: p, q, p_1, q_2

∴ Total Variable = 9

5.1.2 Control Flow Graphs

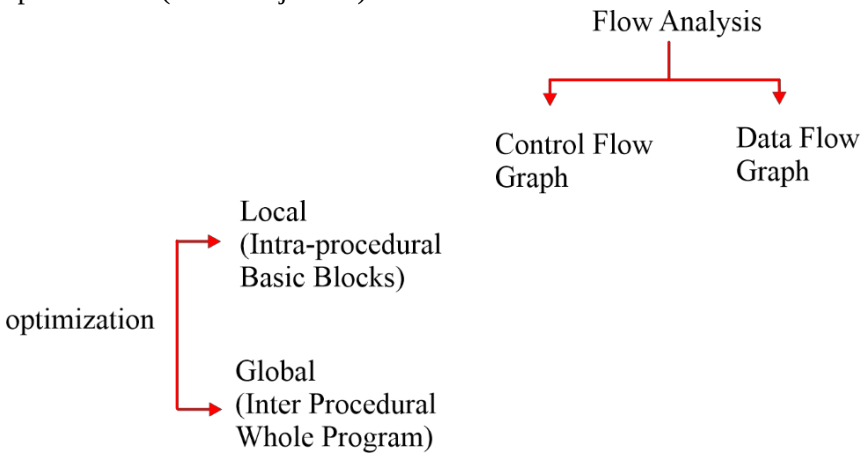
- CFG contain group of basic blocks and controls. CFG has nodes and edges to define basic blocks and controls.
- Basic Blocks: Sequence of 3-address code statements, in which control enters only from 1st statement (called as leader), and leaves from last statement.

Example:

1.	i = 1	LB_1
2.	j = 1	LB_2
3.	t1 = 5 * i	LB_3
4.	t2 = t1 + 5	
5.	t3 = 4*t2	
6.	t4 = t3	
7.	a[t4] = 1	
8.	j = j + 1	
9.	if j ≤ 5 goto 3	LB_4
10.	i = i + 1	LB_5
11.	if i < 5 goto 2	LB_6

5.2 Code Optimization

Saves space / time. (Basic Objective)



5.2.1 Optimization Methods

1. Constant folding
2. Copy propagation
3. Strength Reduction
4. Dead code elimination
5. Common sub expression elimination.
6. Loop Optimization
7. Peephole Optimization

1. Constant Folding

(i) x = 2 * 3 + y ⇒ x = 6 + y

Folding

ii) x = 2 + y * 3} can't fold the constant

2. Copy Propagation

i) Variable Propagation:

x = y;

z = y + 2;

⇒ z = x + 2;

ii) Constant Propagation:

x = 3

z = 3 + a;

⇒ z = x + a

3. Strength Reduction:

Replace expensive statement / instruction with cheaper one.

(i) x = 2 * y **costly** ⇒ x = y + y; **Cheap**

(ii) x = 2 * y ⇒ x = y < < 1; **Much Cheaper**

(iii) x = y / 8 ⇒ x = y > > 3;

4. Dead Code Elimination:

```
x = 2; FALSE                                          x = 2;
if (x > 2)                                            printf("optimization");
        printf ("code"); DEAD CODE CAN BE REMOVED
else
        printf("optimization");
```

- Hence, above dead code never executes during execution. We can always delete such code.

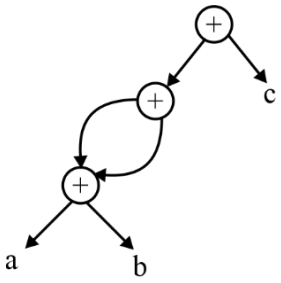
5. Common Sub Expression Elimination:

DAG is used to eliminate common sub expression.

Example: x = (a + b) + (a + b) + c;

⇒ $t_1 = a + b$

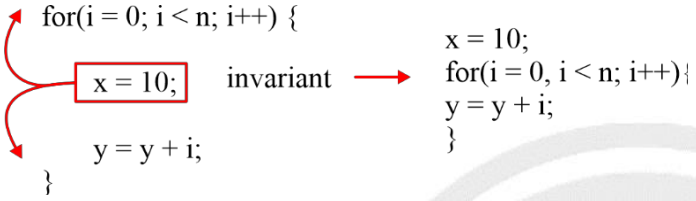
$x = t_1 + t_1 + c$



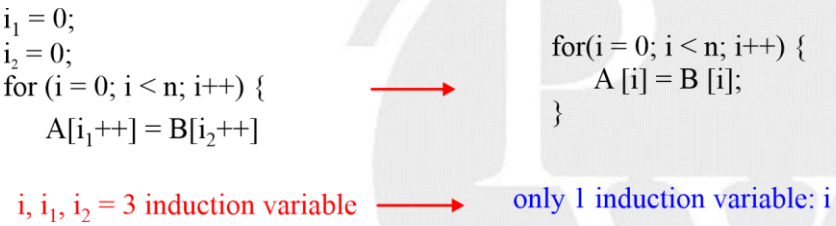
6. Loop Optimization:

(i) Code Motion – Frequency Reduction:

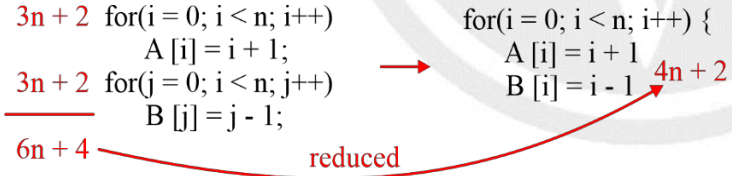
Move the loop invariant code outside of loop.



(ii) Induction Variable elimination:



(iii) Loop Merging / Combining: (Loop Jamming)



(iv) Loop Unrolling:

1)

```
for (i = 0; i < 3; i++)
    printf("CD");
```

3 × 3 + 2 = 11 Statements →

```
printf("CD");
printf("CD");
printf("CD");
```

3 statements

2)

```
for (i = 0; i < 2n; i ++) {
    printf("CD");
}
```

(2 × 3n + 2) = 6n + 2 →

```
for (i = 0; i < n; i++) {
    printf("CD");
    printf("CD");
}
```

(4n + 2)

7. **Peephole Optimization:**

Examines a short sequence of target instructions in a window (*peephole*) and replaces the instructions by a faster and/or shorter sequence when possible.

- Applied to intermediate code or target code
- Following Optimizations can be used:

- Redundant instruction elimination
- Flow-of-control optimizations
- Algebraic simplifications
- Use of machine idioms

❑❑❑

GATE Exam 2025?

SHRESHTH BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in ***English***

Weekday & Weekend Batches Available

JOIN NOW!

Operating System

Operating System

INDEX

1 INTRODUCTION AND BACKGROUND OF OS

1.1 What is OS

Operating system is an interface between the user applications and the computer hardware to develop and execute programs.

1.2 Goals of OS

- The primary goal of the operating system is to provide an easy-to-use environment to the user.
- The secondary goal of the operating system is the efficient use of the computer resources. (Operating system is also called as the resource allocator)
- Modularity
- Abstraction
- Ease of debugging

1.3 Functions of OS

- Process Management
- Memory management
- Resource Allocation
- File systems management
- Protection and Security

1.4 Types of OS

1.4.1. Batch Operating System

- Jobs with similar requirements are grouped into batches by the operating system.
- The idea is to execute the jobs onto the CPU one at a time. Another job cannot occupy the CPU time until the previous job has completed execution.
- Increased CPU idleness.

- Decreased throughput of the system.

1.4.2 Multi-programmed/multi-tasking operating system

- Multiple programs/jobs reside in the main memory for execution. The operating system selects and executes one of these jobs on to the CPU.
- If the job in execution requires an I/O operation, another job which is ready for execution is scheduled on the CPU.
- Increased CPU utilization.
- Increased throughput of the system.
- Multi-tasking is a logical extension of multi-programming systems. The jobs are executed on the CPU in time sharing mode.
- The main advantage of multi-tasking systems is good response time.

1.4.3 Real time operating system

- It has well-defined and fixed time constraints.
- Processing of the programs must be done in the defined constraints or the system will fail.
- They are categorized into-hard and soft real time systems.

1.4.4 Distributed operating system

- A distributed operating system handles jobs that are executed by multiple processors networked to each other.
- They are also termed as loosely coupled systems.
- The advantages of distributed systems include resource sharing, computation speed up and reliability.

1.5 Dual Mode Operations

A processor supports two modes of instruction execution:

- User mode/ non-privileged mode
- Kernel mode/ Privileged mode/ Monitor mode

The primary goal of the dual mode operation is to provide protection and security to the user application programs and the operating system from the unauthorized users. The mode bit is used to determine the particular mode in which an instruction is executing.

Mode bit: 0	Kernel mode
Mode bit: 1	User mode

- A process running in kernel mode has direct access to the hardware and full access to the machine instruction set. The operating system always runs in kernel mode.
- Other examples of privileged instructions include I/O operations, context-switching, clearing the memory map etc.

- A process running in user mode has no access to the hardware and limited access to the machine instruction set.
- Other examples of non-privileged instructions include reading the time of the clock, reading the status of the processor, generate trap etc.

1.6 System Call

- System calls provide the services of the operating system to the user application programs.
- They act as entry points into the kernel system.

1.6.1 Fork () System Call

- The fork () system call is used to create the child processes.
- It returns 0 to the newly created child process.
- It returns the process id of the child process to the parent process on successful creation of the child process.
- It returns negative value to the parent process on unsuccessful creation of the child process.
- If the program contains 'n' fork () system calls, then-

  Number of child processes created = $2^n - 1$.

❑❑❑

2 PROCESS CONCEPTS

2.1 Program Versus Process

Program	Process
Program is a set of instructions and data.	A program under execution is called a process.
It resides in the secondary memory.	It resides in the main memory.
It is passive.	It is active and dynamic.
It is not allocated any resources.	The operating system allocates resources to the process for its execution.

2.2 Process as ADT

The process image can be described as:

Process	
Process attributes (Process Control Block)	Context
Run time Stack	
Dynamic data	User address space
Static data	
Code section	

Any process has the following attributes:

- Process identification information
- Priority
- Process state information
- Program counter
- Memory limits
- List of files
- List of open devices
- Protection information

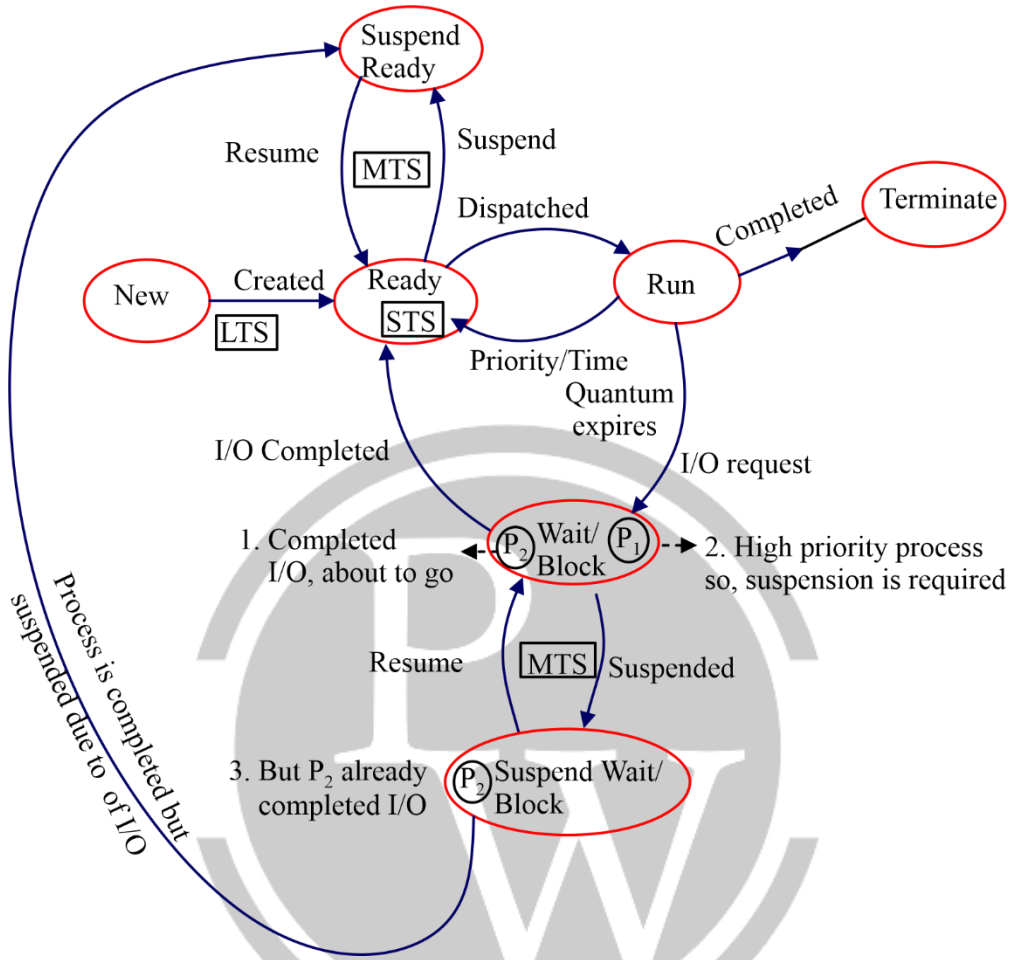
All the attributes of the process are stored in the Process Control Block (PCB). The features of Process Control Block are as follows:

- Every process has its own process control block.
- The process control blocks of all the processes are stored in the main memory.
- They are implemented using **double linked list** data structure.

The **context** of a process includes the process attributes and the stack information.

2.3 Process State Transition Diagram

A process passes through multiple states in its lifetime as follows:



- When the process is present in the Ready, Run or Wait state, it still resides in the main memory.
- When the process is present in the Suspend Ready or Suspend Block state, it resides in the secondary memory.
- There can be multiple processes present simultaneously in the Ready, Wait, Suspend Ready and Suspend Wait states.
- In the Run state, only one process can exist at a time.

2.4 Schedulers

The operating system deploys three kinds of schedulers:

- Long-term scheduler.
- Medium-term scheduler.
- Short-term scheduler.

2.4.1 Long-term scheduler

- It is responsible for the creation and bringing of new processes into the main memory.
- It controls the degree of multi-programming.
- It should generate a good mix of CPU and I/O bound process for efficient resource utilization.
- It is involved in the New → Ready state transition.

2.4.2 Medium-term scheduler

- The medium-term scheduler acts as the swapper by doing swap-out (suspending the process from the main to secondary memory) and swap-in (resuming the process by bringing it from the secondary to main memory) operations.
- It is involved in Ready ⇌ Suspend Ready and Block ⇌ Suspend Block state transitions.

2.4.3 Short-term scheduler

- The short-term scheduler selects a process from the ready state to be executed.
- It is involved in the Ready → Run state transition.

2.5 Dispatcher

- The dispatcher is responsible for loading the job (selected by the short-term scheduler) onto the CPU.
  It performs context switching. Context switching refers to saving the context of the process which was being executed by the CPU and loading the context of the new process that is being scheduled to be executed by the CPU.

❑❑❑

3 CPU SCHEDULING

3.1 Scheduling Criteria

CPU scheduling will occur when:

- A new process arrives into the Ready state.
- A process undergoes Wait → Ready state transition.
- A process undergoes Run → Wait state transition for an I/O request.
- A process undergoes Run → Ready state transition every q second where q is the time slice.
- The priority of a ready process is higher than the priority of a running process.

3.2 Goals of CPU Scheduling

- Maximize CPU utilization.
- Minimize the response time and waiting time of the processes.

3.3 Process times

3.3.1 Arrival Time (AT)

The time at which the process arrives into the Ready state is called arrival time of the process.

3.3.2 Burst Time (BT)

The time required by the process for its complete execution is called burst time/service time of the process.

3.3.3 Completion Time (CT)

The time by which a process completes its execution post arrival is called completion time of the process.

3.3.4 Turnaround Time (TAT)

The time difference between completion and arrival times of a process is called as the turnaround time of the process.
$TAT = CT - AT$

3.3.5 Waiting Time (WT)

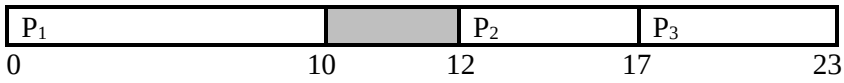
The time spent waiting for the CPU to complete the execution is called as waiting time of the process.
$WT = TAT - BT$

3.3.6 Response Time (RT)

The time difference between the first response and arrival times of a process is called the response time of the process.

3.4 Gantt Chart

Gantt chart shows the execution time period of the processes. For example:



- Process P_1 started execution at time t = 0 and finished just before time t = 10.
- The CPU was idle from t = 10 to t = 12.
- Process P_2 started execution at time t = 12 and finished at time t = 17.
- Process P_3 started execution at time t = 17 and finished at time t = 23.

3.5 Scheduling Algorithms

3.5.1 FCFS

- It is a non pre-emptive algorithm.
- Processes are assigned to the CPU based on their arrival times. If two or more processes have the same arrival times, then the processes are assigned based on their process ids.
- It is free from starvation.
- It suffers from ***Convoy effect.***

3.5.2 Shortest Job First (SJF)

- It is a non-pre-emptive algorithm.
- Processes are assigned to the CPU based on the shortest burst time. If two or more processes have the same burst times, then the processes are assigned to the CPU based on their arrival times.
- If all the processes have the same burst times, SJF behaves as FCFS.
- In SJF, the burst times of all the processes should be known prior execution.
- It minimizes the average response time of the processes.
- There is a chance of starvation.

3.5.3 Shortest Remaining Time First (SRTF)

- It is a pre-emptive algorithm.
- Processes are assigned to the CPU based on their shortest remaining burst times.
- If all the processes have the same arrival times, SRTF behaves as SJF.
- It minimizes the average turnaround time of the processes.
- If shorter processes keep on arriving, then the longer processes may starve.

3.5.4 Longest Remaining Time First (LRTF)

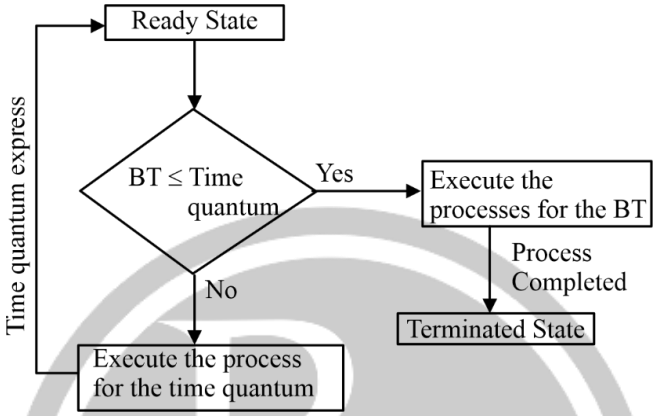
- It is a pre-emptive algorithm.
- Processes are assigned to the CPU based on their longest remaining burst times.
- It minimizes the average response time of the processes.
- If favours CPU bound processes.
- It is free from starvation.

3.5.5 Priority based Scheduling

- Priority scheduling can either be pre-emptive or non-pre-emptive.
- In pre-emptive priority scheduling, a process is voluntarily pre-empted whenever a higher priority process arrives.
- In non-pre-emptive priority scheduling, the scheduler picks up the highest priority process.
- If all the processes have equal priority, then priority scheduling behaves as FCFS.

3.5.6 Round Robin Scheduling

- RR scheduling is a pre-emptive FCFS based on the concept of time quantum or time slice.



- If the time quantum is too small, then the number of context switches (overhead) will increase and the average response time will decrease.
- If the time quantum is too large, then the number of context switches (overhead) will decrease and the average response time will increase.
- If the time quantum is greater than the burst times of all the processes, RR scheduling behaves as FCFS.

3.5.7 Highest Response Ratio Scheduling

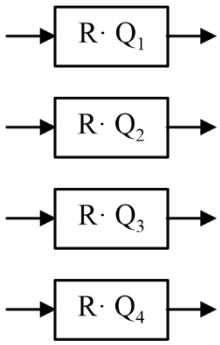
- It is a non pre-emptive algorithm.
- Processes are assigned to the CPU based on their highest response ratio.
- Response ratio is calculated as-

$$\text{Response Ratio} = \frac{WT + BT}{BT}$$

- It is free from starvation.
- It favours the shorter jobs and limits the waiting time of the longer jobs.

3.5.8 Multilevel Queue Scheduling

Multi-Level Queue Scheduling



- Depending on priority of the process, in which particular ready queue, the process has to be placed will be decided.
- High priority processes will be placed in top-level ready queue and low priority process will be placed in bottom level ready queue.
- Only after completion of all the processes, from top level ready queue the further level ready queue processes will be scheduled.
- If this is the strategy followed, then the processes which are placed in bottom level ready queue suffer from starvation.

If higher priority processes keep on arriving, the lower priority processes may starve. This problem can be solved with the help of aging. Aging is a technique which automatically increases the priority of the processes that have been waiting in the system for a very long time.

❑❑❑

4 MULTITHREADING

4.1 Threads

- A thread is a light-weight process.
- In multithreading, the threads of the same process share user address space, files, signal and signal handlers etc.
- In multithreading, each thread has its own stack, thread id, CPU state information (control and status registers, stack pointers) and scheduling information (thread state, priority etc.).

4.2 Benefits of Threads

- Improved response time.
- Faster context switches.
- Effective utilization of multiprocessor systems
- Enhanced throughput of the system.
- Economical.
- Effective resource sharing and utilization.

4.3 Types of Threads

4.3.1 Based on the number of threads

There are two types of threads:
(i) Single thread process
(ii) Multi thread process

4.3.2 Based on level

There are two types of threads:
(i) User-level threads
(ii) Kernel-level threads

4.3.3 User level threads versus kernel level threads

User level threads	Kernel level threads
These threads are managed at user level.	These threads are managed at kernel level.
These threads are not recognized by the kernel.	These threads are recognized by the kernel.
They are implemented as dependent threads.	They are implemented as independent threads.
All user-level threads of a process can run on one processor only and only one thread runs at a time.	The kernel-level threads of a process can run on different processors concurrently in a multi-processor environment.
Blocking one user-level thread of a process blocks the entire process.	Blocking one kernel-level thread of a process does not affect the other threads of the process.
These threads have less context.	These threads have more context.
Scheduling of user-level threads is done by the thread libraries.	Scheduling of kernel-level threads is done by the operating system.
No hardware support is required.	Hardware support is required.
Implementation is easy and simple.	Implementation is complicated and difficult.

4.4 Multithreading Models

4.4.1 Many-to-One

- Many user level threads are mapped to single kernel-level thread.

4.4.2 One-to-One

- Each user level thread is mapped to single kernel-level thread.

4.4.3 Many-to-Many

- Many user level threads are mapped to multiple kernel-level threads.
- It allows the OS to create a sufficient number of kernel threads

4.5 Thread Libraries

- Thread libraries provide programmer with API for creating and managing threads.
- There are two primary ways of implementing thread libraries:

(i) Library entirely in user space

(ii) Kernel-level library supported by the OS

- Examples include pthread libraries, java threads, green threads etc.

4.6 Disadvantages of Multithreading

- Blocking: Blocking one user-level thread of a process blocks the entire process. If the kernel thread is single threaded, then blocking the kernel thread will block the whole process. Consequently, CPU may remain idle during this period.
- Security: Since, there is an extensive sharing among threads, there is a potential problem of security.
- Maintaining Thread Control Block (TCB) is considered as overhead for the system.

❑❑❑

5 SYNCHRONIZATION

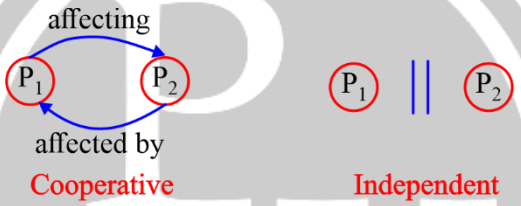
5.1 Synchronization

Processes are categorized into two types

Process

- Cooperative
- Independent

Execution of one process affects or affected by other process then those processes are said to be cooperative process. Otherwise, they are said to be Independent Process.



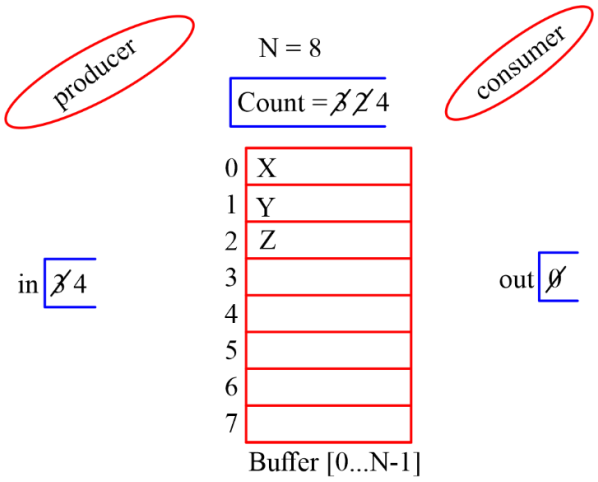
Processes must be cooperative to be synchronized.

5.2 Understanding Synchronization

(1) The problems arise not having synchronization between the processes.
(2) The conditions to be followed to achieve synchronization.
(3) The solutions (wrong and right).

5.3 Problems arises for not having synchronization between the processes

5.3.1 Producer Consumer




```
int count = 0; // Initially buffer will be empty and only producer will be allowed to execute first.
void producer (void)
    {
        int itemp;
        while(true)
        {
            produce_item(itemp);
            while (count = = N); // Buffer Full
            Buffer [in] = itemp;
            in = (in + 1)modN; // ← to repeat in value from 0 - 7
            count = count + 1;
        }
    }
```

Register of producer memory

I. Load R_p.m[count]
II. INCR R_p
III. Store m[count], R_p

```
void consumer(void)
    {
        int itemc;
        while(true)
        {
            while (count = = 0);
            itemc = Buffer [out];
            out = (out + 1) modN;
            count = count – 1;
            process_item(itemc);
        }
    }
```

I. Load R_c,m[count]
II. DECR R_c
III. Store m[count], R_c

- IN is a variable used by the producer to identify the next empty slot in the buffer.
- OUT is a variable used by the consumer to identify from where it has to consume the item.

- Count is a variable used both producer and consumer to identify no of items present in the buffer at any point of time.
- **Shared Resources:**
  (1) Count Variable
  (2) Buffer
- **Two conditions to be followed:**
  (1) If the buffer is full, producer is not allowed to produce the item into the buffer.
  (2) If the buffer is empty, consumer is not allowed to consume the item from the buffer.

5.3.2 Universal Assumption

The running process can get pre-empted at any point of time after completion of current instruction.

Analysis:

Itemp = 'A'
Itemc = 'X'

count
3 2 4

P → I Rp 3 in 3 4 out 0 1
P → II Rp 3 4

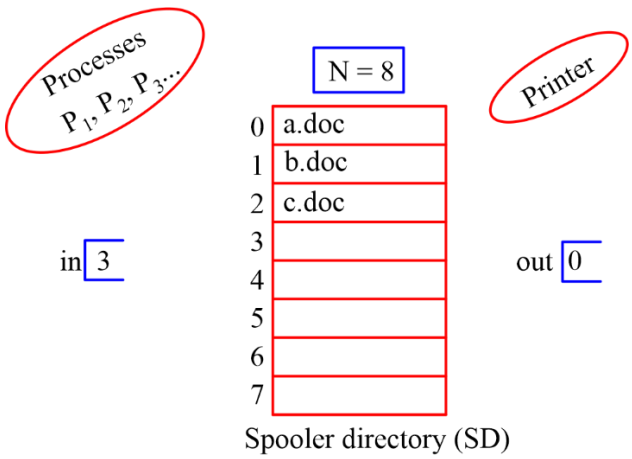
C → I
C → II Rc 3 2
C → III

P → III

Final count value = 4. Though actually three values are there.

(1) Inconsistency:
The producer and consumer are not properly synchronized while sharing the common variable "count". Hence it is leading to the problem of inconsistency.

5.3.4 Printer Spooler Daemon (Daemon → Background process)



5.3.5 Definitions

(1) Critical Section:

The portion of program text where shared variables or shared resources will be placed.

Example:

Count = Count + 1

OR

Count = Count – 1

(2) Non-Critical Section:

The portion of program text where the independent code of the processes will be placed.

Example:

Tn = (in + 1) modN

(3) RACE condition:

The final value of any variable depends on execution sequence of the processes. This type of condition is called as RACE condition.

- To avoid the RACE condition, only one process is allowed to enter into critical section.

5.3.6 Conditions to be followed to achieve Synchronization

(1) Mutual Exclusion (M.E):

- No two processes may be simultaneously present inside the critical section at any point of time.
- Only one process is allowed to enter into critical section at any point of time.

(2) Progress:

- No process running outside the critical section should block the other intersected process from entering into critical section when critical section when critical section is free.
- If there is only one process trying to enter into critical section then it should be definitely allowed to enter into critical section.
- If two or more processes are trying to enter into critical section then one process should be definitely allowed to enter critical section.

(3) Bounded Waiting:

- No process should have to wait forever to enter into critical section.
- There should be a bound on getting chance to enter into critical section.
- Some process is indefinitely waiting to enter into critical section because critical section is always busy by some other processes. This situation should not arise.
- If the bounded waiting is not satisfied then it is possible for starvation.

(4) No assumptions related to hardware and the processor speed:

- Number of processes.

Solutions:

I. Software Type:

(a) Lock Variables

(b) Strict alteration or Decker's Algorithm

(c) Petersons Algorithm

II. Hardware Type:

(a) TSL Instruction Set

(b) Test and Set Lock

III. OS Type:

(a) Counting semaphore
(b) Binary Semaphore

IV. Programming Language Compiler Support Type:

(a) Monitors

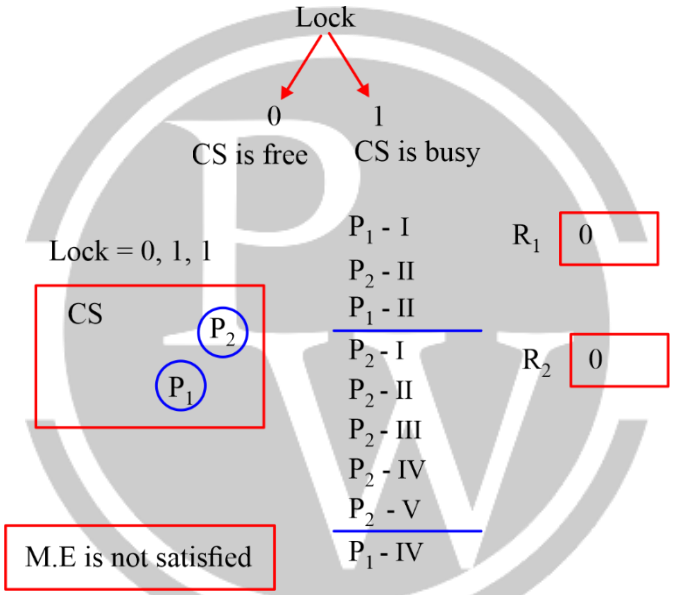
I. Software Types:

(a) Lock Variables:

Entry Section:

I. Load R_i, m[Lock] ($R_i \rightarrow$ Respective Process Register)
II. Cmp R_i, #0
III. JNZ to Step (I)
IV. Store m[Lock], #1
V. C.S
VI. Store m[Lock], #0

Analysis:



We have proved that both the processes P_1 and P_2 are entering into the critical section at the same time, Hence mutual exclusion is not satisfied and the solution is bound to be incorrect.

(b) Strict Alteration or Decker's Algorithm:

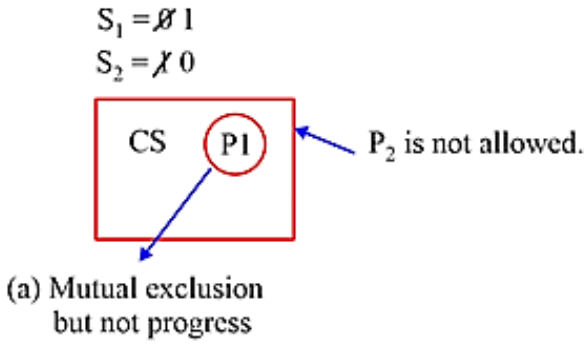
(Process takes 'Turn' to enter into C.S)

Process 'P_0' code	Process 'P_1' code

while (true)
{
non_cs ();
while (turn! = 0);
c.s
turn = 1;
}
	while (true)
	{
	non_cs();
	while(turn! = 1);
	c.s
	turn = 0;
	}

Important Points:

- The pre-emption is just a temporary stop and the process will come back and continue the remaining execution.
- If there is any possibility of solution becoming wrong by taking the pre-emption then consider the pre-emption.
- If any solution is having deadlock the progress is not satisfied.

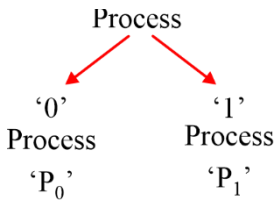


(a) Mutual exclusion but not progress

(c) Peterson's Algorithm:

(Two Process Solution)

```
#define N 2
#define TRUE 1
#define FALSE 0
    int turn;
    int interested [N];
    void enter_Region (int process)
    {
        1.  int other
        2.  other = 1 – process;
        3.  interested [process] = TRUE;
        4.  turn = process;
        5.  while (turn = = process && interested [other] = = TRUE);
    }
    C.S
    void leave_Region(int process)
    {
        interested [process] = FALSE;
    }
initially
interested [0] = FALSE;
interested [1] = FALSE
```

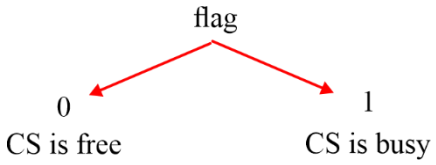


TURN is a shared variable used by both the processes P_0 and P_1, interested [N] is also shared by both the processes.

5.3.7 Hardware Type

(a) TSL Instruction Set: (Test and Set Lock):

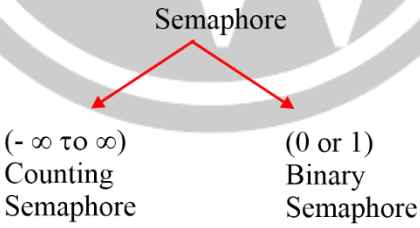
- **TSL Register Flag:**
  Copies the current value of flag into register and stores the value of '1' into flag in a single atomic cycle without any pre-emption.
- **Entry Selection:**
  1. TSL Ri, m[flag]
  2. Cmp Ri, ≠ 0
  3. JMP to step (1)
  4. c.s
  5. Store m[flag], ≠ 0



Algorithm	M.E.	Progress	Bounded Waiting
1. Lock Variable	×	✓	×
2. Strict alteration or Decker's Algorithm	✓	×	✓
3. Peterson's Algorithm	✓	✓	✓
4. TSL Instruction	✓	✓	×

5.3.8 OS Type

Semaphore:
Semaphore is an integer variable which is used by the various processes in a mutual exclusive manner to achieve synchronization.
Improper usage of semaphore will also give the wrong results.
Semaphore is categorised into 2 types:



The two different operations will be performed on the semaphore variable.
(1) Down, or wait (); or p ()
(2) up (); or signal (); or v (); or release ();

(a) Counting Semaphore:

```
Down (Semaphore s)
{
    s.value = s.value – 1;
    if (s.value < 0)
    {
        Block the process and
        place its PCB in the
```

```
            suspended list ();
        }
    }
    Up (Semaphore s)
    {
        s.value = s.value + 1;
        if (s.value ≤ 0)
        {
            Select a process from
            suspended list and
            wakeup ();
        }
    }
```

(b) Binary Semaphore:

```
Down (Semaphore S)
{
    if (S.value = = 1)
       S.value = 0;
    else
    {
        block the process
        and place its PCB
        in the suspended list ()
    }
}
```

```
Up (Semaphore S)
{
    if (Suspended list is empty)
    S.value = 1;
    else
    {
        select a process
        from suspended list
        and wakeup ();
    }
}
```

Notes: (Applicable for both counting and binary semaphore).

⇒ Every semaphore variable will have its own suspended list.

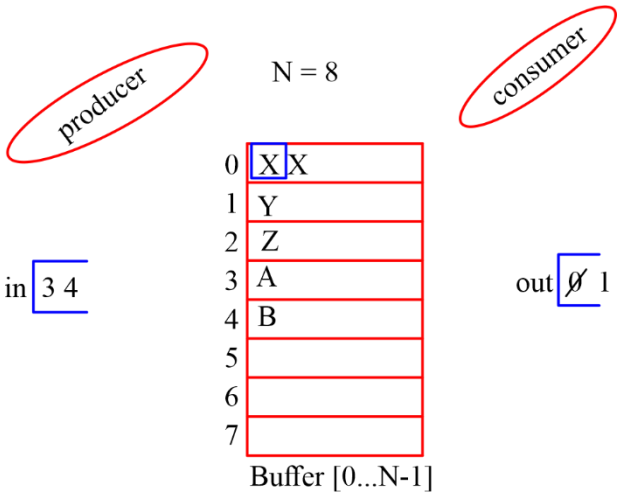
⇒ The down and up operations are atomic [OS will prevent the interrupts]

⇒ If more than one process is in the suspended list then every time when we perform one up operation one process will wakeup from the suspended list and this will be based on (FIFO → to ensure bounded waiting)

⇒ If two or more processes are in the suspended list and there is no other process to wakeup these processes then those processes are said to be involved in the deadlock.

5.4 Classical problems of IPC (Inter Process Communication)

- Producer consumer with semaphore:



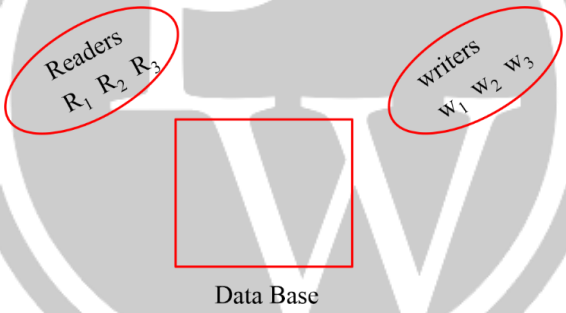

```
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;
void producer(void)
{
    int item p;
    while(true)
    {
        produce_item (item p);
        down(empty);
        down(mutex);
        buffer [in] = item p;
        in = (in + 1) mod N
        up(mutex);
        up(full);
    }
}

void consumer (void)
{
    int item c;
    while (true)
```

```
    {
        down (full);
        down (mutex);
        item c = buffer [out];
        out = (out + 1)mod N;
        up (mutex);
        up (empty);
        process_item (item c);
    }
}
```

- mutex is a binary semaphore used by the producer and consumer to access the buffer in a mutual exclusive manner.
- empty is counting semaphore variable represents number of empty slots in the buffer at any point of time.
- full is counting semaphore variable represents number of slots full in the buffer at any point of time.

5.5 READERS WRITERS PROBLEM




```
int rc = 0;
semaphore mutex = 1;
semaphore db = 1;
void reader (void)
{
    while (true)
    {
        down (mutex);
        rc = rc + 1;
        if (rc = = 1) down (db);
        up (mutex);
    }
```

```
}
void writer (void)
{
    while (true){
        down (db);
        D.B
        up (db);
    }
}
```

5.5.1 conditions to be followed

1) R - W ✗
2) R – R ✓
3) W – R ✗
4) W – W ✗

- rc is a integer variable represents readers count i.e number of readers present in the data base at any point of time.
- mutex is a binary semaphore used by the readers in mutual exclusive manner.
- db is a binary semaphore variable used by readers and writers in a mutual exclusive manner.

5.5.2 Dining Philosophers' problem

Dining Philosophers problem
- N philosophers and N forks
- Philosophers eat/think
  - Eating needs 2 forks
  - Pick one fork at a time

5.5.3 Solution 1 for Dining Philosophers problem

```
# define N5                     /* number of philosophers */

Void philosophers (int i)       /* i: philosophers' number from 0 to 4 */
{
    while (TRUE) {
    think ();                   /* philosopher is thinking */
     take_fork (i);             /* take left fork */
     take_fork ((i+1)%N)        /* take right fork; % is modulo operator */
     eat ();                    /* yum-yum spaghetti */
     put_fork (i);              /* put left fork back on the table */
     put_fork ((i+1)%N)         /* put right fork back on the table */

{
```

- This solution to dining philosopher problem suffers from Deadlock.
- Everyone picks up their left fork first, then waits for right fork… ⇒ this leads to Starvation!

- This solution to dining philosophers' problem suffers from Deadlock everyone picks up their left fork first, then waits for right fork => This leads to Starvation!

5.5.4 Solution 2 – state based

```
# define N                    5                    /* number of philosophers */
# define LEFT                 (i+N-1)%N            /* number of I's left neighbours  */
# define RIGHT                (i+1)%N              /* number of I's right neighbours */
# define THINKING             0                    /* philosopher is thinking */
# define HUNGRY               1                    /* philosopher is trying to get forks */
# define EATING               2                    /* philosopher is eating */
typedef int semaphore                              /*semaphores are a special kind of int */
int state [N];                                     /* array to keep track of everyone's state */
semaphore mutex = 1;                               /* mutual exclusion for critical regions */
semaphore s[N]:                                    /* one semaphore per philosopher */

void philosopher (int i)                           /* i: philosopher number, from 0 to N -1 */
 {        while (TRUE) {                           /* repeat forever */
                Think();                           /* philosopher is thinking */
                Take_forks(i);                     /* acquire two forks or block */
                Eat();                             /* yum-yum spaghetti */
                Put_forks(i);                      /* put both forks back on table */
          }
}
void take_forks(int i)                             /* i: philosopher number, from 0 to N -1 */
{
         down(&mutex);                             /* enter critical region */
         state [i] = HUNGRY;                       /* record fact that philosopher is hungry */
         test(i);                                  /* try to acquire two forks */
        up(&mutex);                                /* exit critical region */
         down(&s[i]);                              /* block if forks were not acquired */
}
void put_forks(i)                                  /* i: philosopher number, from 0 to N -1 */
{
         down(&mutex);                             /* enter critical region */
         state [i] = THINKING;                     /* philosopher has finished eating */
         test(LEFT);                               /* see if left neighbour can now eat */
         test(RIGHT);                              /* see if left neighbour can now eat */
         up(&mutex);                               /* exit critical region */
}
void test(i)                                       /* i: philosopher number, from 0 to N -1 */
{
       if (state[i] = = HUNGRY && state [LEFT] = EATING && state [RIGHT] = EATING) {
                state [i] = EATING;
                up(&s[i]);
        }
}
```

Made picking up left and right chopsticks an atomic operation ξ or, N – 1 philosophers but N chopsticks ξ …both changes prevent deadlock.

5.6 Monitors:

- Monitors is a programming language compiler support type of solution to achieve synchronization.
- Monitors is collection of variables and procedures combined together in a special kind of module or package.
- The process running outside the monitor cannot directly access the internal variables of the monitor but however they can call the procedures of the monitor.
- Monitors has an important property that only one process can be active inside the monitor at any point of time.

5.6.1 Syntax

```
Monitor example
{
    Variables;
        Condition variables;
        Procedure P1
        {
        }
        Procedure P2
        {
        }
}
```

5.6.2 Condition variables

```
Condition  x,y;
```

The two different operations performed on the condition variables of the monitor.

1. Wait operation: wait ()
2. Signal operation: signal ()

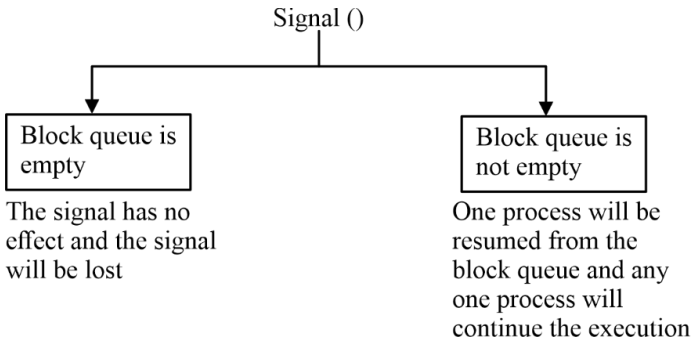
- **Wait ()**

  Ex – x.wait (); or wait (x)

  The process performing wait operation on any condition variable will be suspended and suspended process will be placed in the "block queue" of respective condition variable.

- **Signal ()**

  Ex – x.signal (); or signal(x)



5.6.3 Concurrent Programming

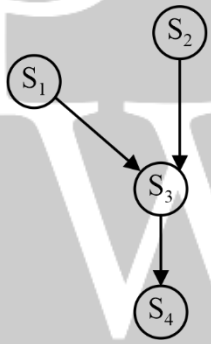
S_1: a = b + c ;
S_2: d = e * f ;
S_3: g = a / d ;
S_4: h = g * i ;

Read set = {b, c, e, f, a, d, g, i}

Write set = {a, d, g, h}

5.6.4 Precedence graph

S_1, S_2 can execute concurrently



- Any two statements S_i and S_j can be executed concurrently or parallel if they are following the conditions.

  (1) $R(S_i) \cap W(S_j) = \varphi$
  (2) $W(S_i) \cap R(S_j) = \varphi$
  (3) $W(S_i) \cap W(S_j) = \varphi$

- The real concurrent programming is possible only on the multiprocessor system.
- Concurrent has 3 different meanings
  - They can execute concurrently or parallel
  - They don't have any dependency

  Anyone can start first [for single processor this will be applicable]

❑❑❑

6 DEADLOCK

6.1 Concept of Deadlock

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example:
  - System has 2 disk drives.
  - P_1 and P_2 each hold one disk drive and each needs another one.

6.2 System Model

- Resource types $R_1, R_2, \ldots, R_m$
- Resources include CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - request
  - use
  - release

6.3 Deadlock Characteristics

6.3.1 Mutual Exclusion

There should be a one-to-one relationship between a resource and a process.

6.3.2 Hold and Wait

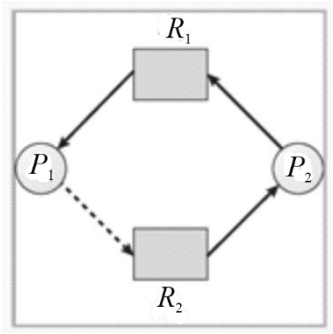
This condition arises when a process is requesting another resource while holding on to a resource.

6.3.3 No Pre-emption

The resource has to be voluntarily released by the process after its execution. It is not allowed to forcibly pre-empt a process to release its held resources.

6.3.4 Circular Wait

The processes are circularly waiting on each other for the resources.



Note:
If all the deadlock characteristics simultaneously exist in the system, then the system is in deadlock.

6.4 Deadlock Prevention

It ensures that the system will *never* enter into a deadlock state. Deadlock can be prevented by unsatisfying one of the mentioned deadlock characteristics.

- Mutual exclusion cannot be unsatisfied because of the concept of sharable and non-sharable resources.
- Hold and Wait characteristic can be unsatisfied through any of the following strategies:
  (i) A process should be assigned all the required resources before the start of its execution. This may lead to low device utilization.

  (ii) The process should release all the existing resources before making a new request. This may eventually lead to starvation.

- Pre-emption can be achieved as:
- Suppose a process P_1 is requesting for a resource R which is held by another process P_2. If P_2 is already in execution then P_1 has to wait; else, the resource R will be pre-empted from P_2 and allocated to P_1.
- Circular wait can be avoided by the following algorithm:
  (i) Assign unique numbers to resources. (assume there exists single instance of a resource).

  (ii) A process can request for a resource in either increasing or decreasing order of enumeration.

6.5 Deadlock Avoidance

Deadlock avoidance is achieved by implementing Banker's algorithm. Banker's algorithm ensures that the system never enters in unsafe state.

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in safe state if there exists a sequence $<P_1, P_2, \ldots, P_n>$ of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j.
- If a system is in safe state $\Rightarrow$ no deadlocks.
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock.

6.5.1 Data Structures for Banker's algorithm

Let n = number of processes and m = number of resources, then-

- ***Available***: Vector of length *m*. If available [*j*] = *k*, there are *k* instances of resource type R_j available.
- ***Max***: *n x m* matrix. If *Max* [*i,j*] = *k*, then process P_i may request at most *k* instances of resource type R_j.
- ***Allocation***: *n* x *m* matrix. If Allocation[*i,j*] = *k* then P_i is currently allocated *k* instances of R_j.
- ***Need***: *n* x *m* matrix. If *Need*[*i,j*] = *k*, then P_i may need *k* more instances of R_j to complete its task.
- *Need* [*i,j*] = *Max*[*i,j*] – *Allocation* [*i,j*].

6.5.2 Banker's Algorithm

- Algo Bankers(i , Request, Available, Allocation, Need){

```
    If(Need_i ≤ Max_i)
{
        If(Need_i ≤ Available)
          {
            Available = Available – Need_i ;
            Allocation = Allocation + Need_i ;
            Max_i = Max_i - Need_i ;
            Run safety Algorithm;
            If the system is in safe state, then grant the Need_i;
            Else block the process;
          }
 }
```

6.6 Deadlock Detection

- A cycle in the resource allocation graph represents deadlock only when resources are of single-instance type.
- If the resources are of multiple instance type, then the safety algorithm is used to detect deadlock.

6.7 Deadlock Recovery

A system can recover from deadlock through adoption of the following mechanisms:

- Process termination
- Resource Pre-emption
- Ostrich Algorithm:
- Ignore the problem and pretend that deadlocks never occurred in the system; used by most operating systems, including UNIX.

❑❑❑

7 MEMORY MANAGEMENT

7.1 Introduction

- In multiprogramming system, the task of subdividing the memory among the various processes is called memory management.
- The task of the memory management unit is the efficient utilization of memory and minimize the internal and external fragmentation.

7.2 Logical versus Physical Address

- The address generated by CPU is called the logical address.
- The address perceived by the memory unit is called physical address.

7.3 Memory Management Unit

- Hardware device that maps virtual to physical address.
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.

7.3.1 Loading

It is defined as bringing the program from the secondary to the main memory.
It is classified into three types:

(i) Absolute Loading

- A given program is always loaded into the same memory location whenever loaded for execution.

(ii) Relocatable Loading

- A given program is loaded into any desired memory location whenever loaded for execution.
- The compiler must generate relative address for the program.

(iii) Dynamic Loading

- Routine is not loaded until it is called better memory-space utilization (unused routine is never loaded, postponed until execution time).
- Useful when large amounts of code are needed to handle in frequently occurring cases.
- No special support from the operating system is required. It is implemented through program design
- Address translation is performed through the hardware.

7.3.2 Linking

Linking is the process of collecting and combining various pieces of code and data into a single file that can be loaded (copied) into memory and executed. It can be performed at compile time, load time or run time.

It is classified into two types:

- Static Linking
  Static linkers take as input a collection of relocatable object files and command-line arguments and generate as output a fully linked executable object file that can be loaded and run.
- Dynamic Linking
  A shared library is an object module that, at run time, can be loaded at an arbitrary memory address and linked with a program in memory. This process is dynamic linking.

7.3.3 Address Binding

Association of the program instructions and data into the actual physical memory locations is called as address binding. Address binding of instructions and data to memory addresses can happen at three different stages

- **Compile time**: If memory location known a priori, **absolute code** can be generated. It must recompile code if starting location changes.
- **Load time**: It must generate **relocatable code** if memory location is not known at compile time.
- **Execution time**: Address binding is delayed until run time if the process can be moved during its execution from one memory segment to another. It needs hardware support for address mapping (e.g., base and limit registers).

7.4 Memory Management Techniques

7.4.1 Contiguous Memory Allocation

It comprises of – Fixed Partition Scheme and Variable Partition Scheme

Fixed Partition Scheme / Static Partitioning

- The memory is divided into a fixed number of partitions of equal/unequal sizes.
- Every partition is associated with limit registers.
- The degree of multiprogramming is restricted by the number of partitions.
- Partition size may not be large enough for any waiting process.
- Internal fragmentation may be present.

Variable Partition Scheme / Dynamic Partitioning

- Initially, memory is available as a single continuous free block. When a process arrives, a hole large enough to accommodate it is created in the memory.
- There is no internal fragmentation.
- It suffers from external fragmentation.
- It is associated with the overhead of compaction.

Note

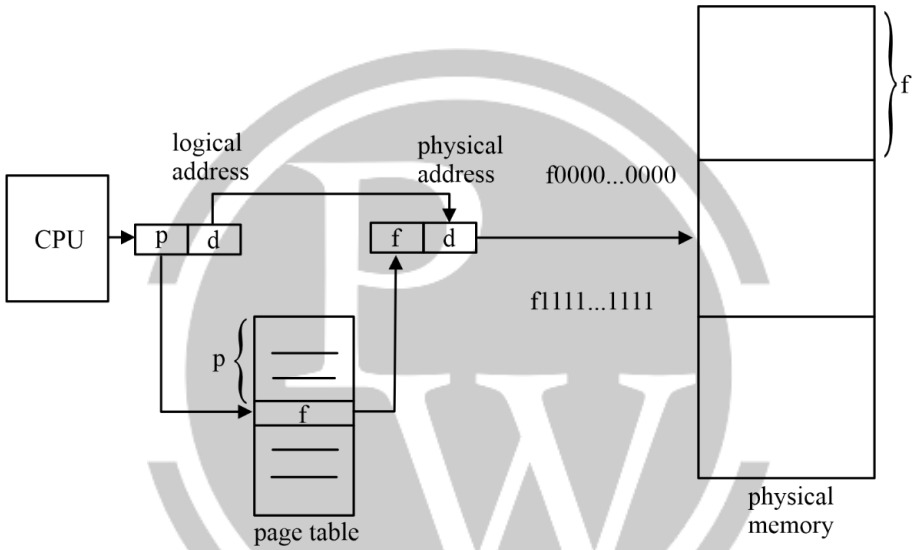
If more than one partition is sufficient to accommodate a process, then any of the following dynamic allocation methods can be adopted:

- First Fit: Allocate the first partition that is big enough starting from the beginning of the memory.
- Next Fit: It behaves similar to first fit except that it scans the memory after the ‘last allocation point’.
- Best Fit: It scans the entire memory to find the smallest sufficient partition to accommodate the process.
- Worst Fit: It scans the entire memory to find the largest sufficient partition to accommodate the process.

7.4.2 Non-Contiguous Memory Allocation

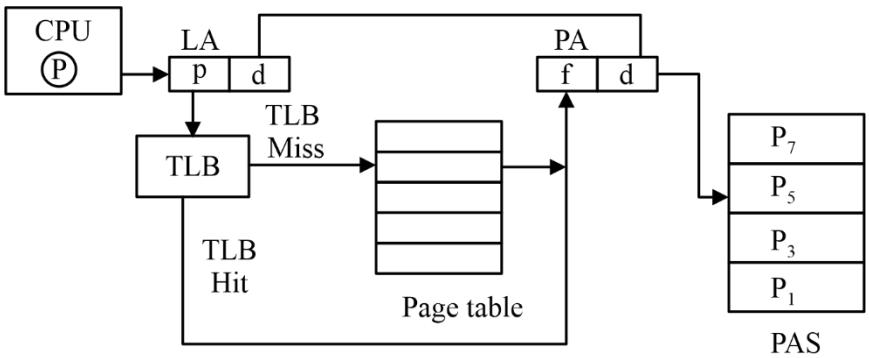
Paging

- The technique of mapping CPU generated logical address to physical address is called paging.
- Logical address space is divided into equal size pages. Physical address space is divided into equal size frames. In paging, the frame size is equal to the page size.
- When a process is created, paging will be applied on the process. A page table will be maintained in the main memory for a process. The base address of the page table will be stored in the process control block.
- The number of entries in the page table is equal to the number of the pages in the logical address space. Each page table entry contains the frame number. Hence, the page table is also known as the **address translation** table.
- There is no external fragmentation in paging. Internal fragmentation may exist in the last page and is formulated as $\left\lceil \frac{p}{2} \right\rceil$ where p is the page size.



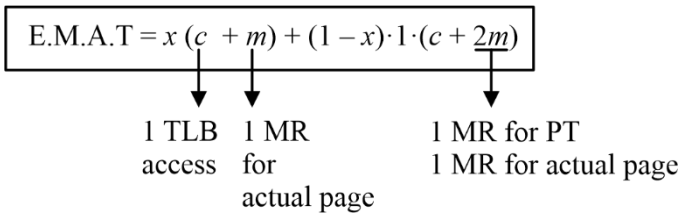
Paging with TLB

- **Translation Look Aside Buffer** is a hardware device implemented using associative registers.
- **TLB** is added to improve the performance of paging. The TLB access time will be very less compared to the main memory access time.
- **TLB** contains frequently referred page numbers and their corresponding frame numbers.



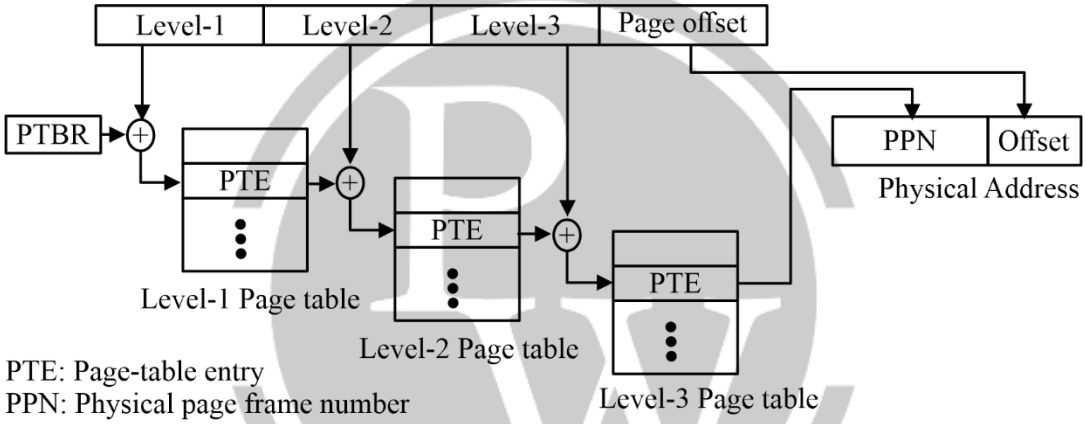
- TLB access time = c
- TLB hit ratio = x

Then the formula for effective memory access time:



Multilevel Paging

- To avoid the overhead of maintaining large size page tables, multilevel paging will be applied.
- In multilevel paging, pages of the page table are brought into the main memory.
- The page tables of all the levels of multi-level paging are kept in the memory.
- The entries of the level-1 page table are pointers to the level-2page table.
- The entries of the level-2 page table are pointers to the level-3 page table.
- The entries of the last level page table will have frame numbers of the actual page.
- All levels page table entry must contain frame number.
- Address translation is done as:



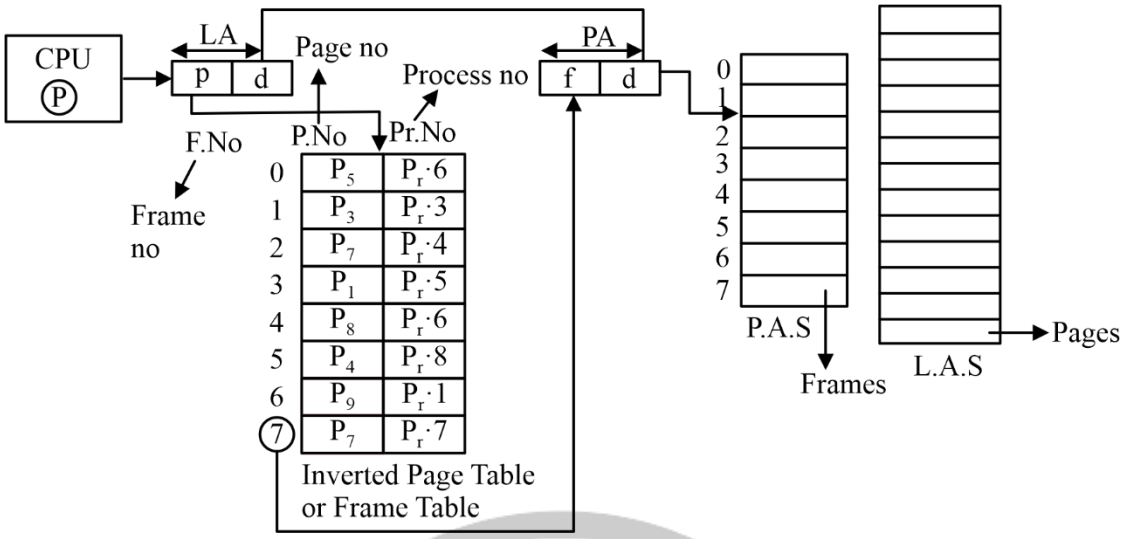
- Performance of multi-level paging with TLB:
  Assume, TLB hit ratio = p, TLB access time = c, Main Memory Access time = m
  If the system uses n-level paging, then effective memory access time is given as-

$$EMAT = p(c + m) + (1 - p)(c + (n + 1)m)$$

Inverted Paging

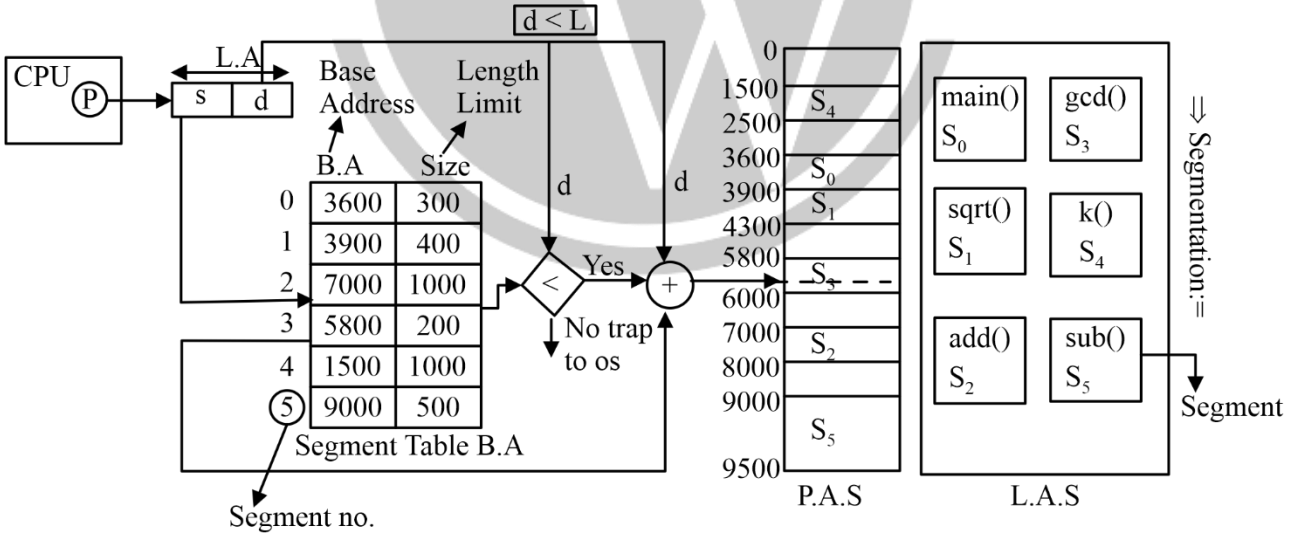
- Inverted paging maps physical frames to virtual pages.
- Instead of maintaining multiple page tables for different processes, an inverted page table can be maintained.
- The number of entries in the inverted page table is equal to the number of the frames in the physical address space.
- Each inverted page table entry contains a page number and a process identifier. It gives an idea about 'which page of which process is allocated in which frame'.
- The disadvantage of inverted paging is the increased lookup time and hard to implement.

- Address translation is done as:



Segmentation

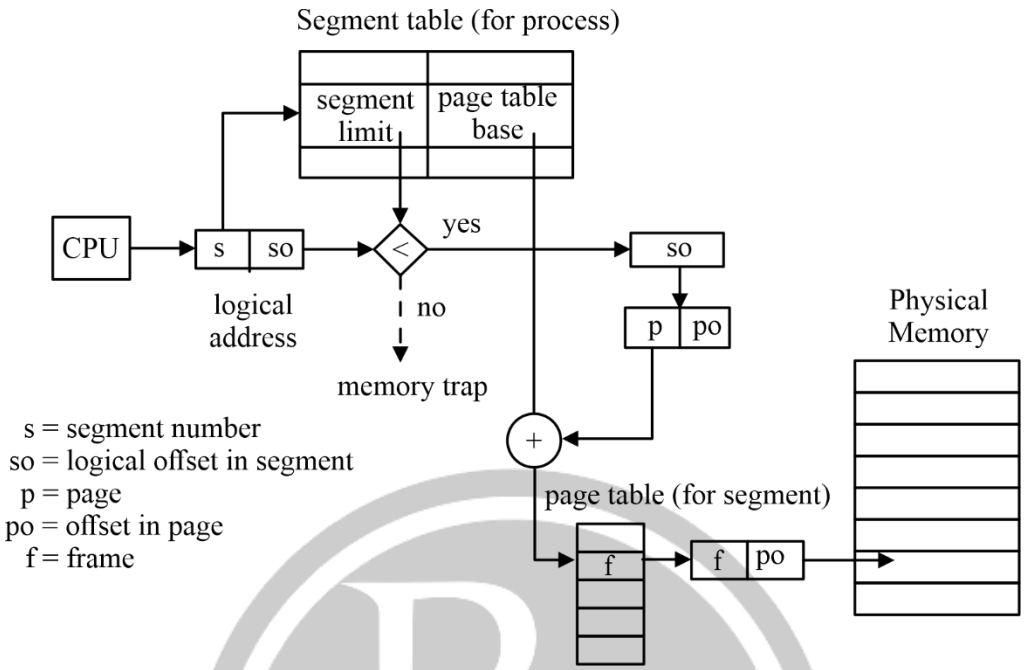
- Paging does not follow user's view of memory allocation. Instead of dividing the process into equal sized pages, it can be divided into variable length segments.
- A segment table is maintained for each process. The number of entries in the segment table is equal to the number of segments of a process.
- Each segment table entry contains the base (starting address) and limit(length) of the segment.
- Segmentation does not suffer from internal or external fragmentation.
- Address translation is done as:



Segmented Paging:

- To avoid the overhead of bringing large sized segments of a process into the main memory, paging will be applied on the segments.
- Pages of the segment will be loaded into the main memory.
- A page table will be maintained for each segment of the process.
- The number of entries in the page table is equal to the number of the pages of the segment in the logical address space.
- Lookup time increases.

- It suffers from internal fragmentation only.
- Address translation is done as:



❑❑❑

08 VIRTUAL MEMORY

8.1 VM Concept

- It gives an illusion to the programmer that programs having larger size than the physical memory can be executed.
- It allows address spaces to be shared by several processes.

8.2 VM Implementation

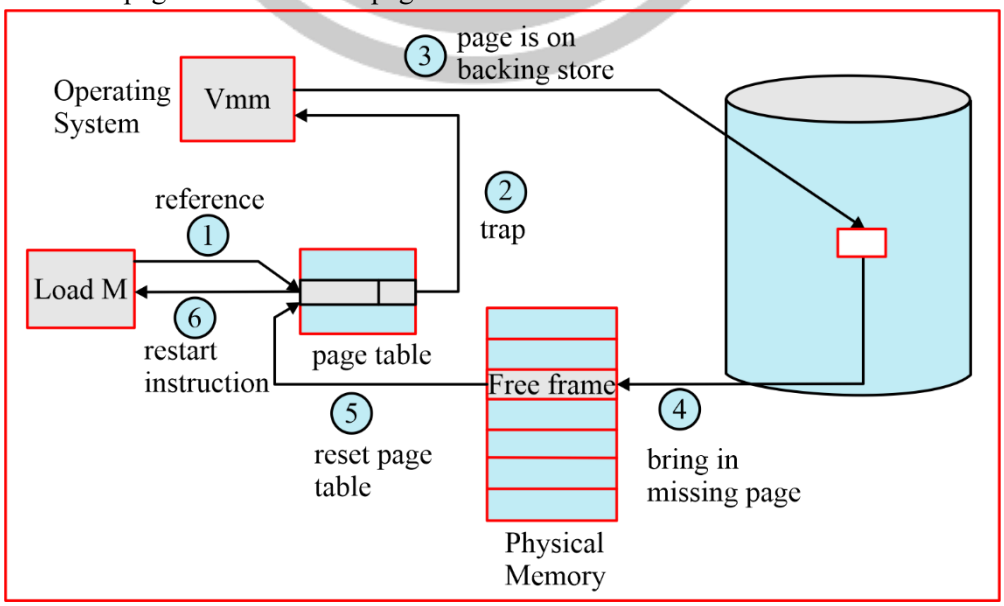
Virtual memory can be implemented through:

- Demand Paging
- Demand Segmentation

8.2.1 Demand Paging

Loading the pages from the secondary memory to the main memory on demand is called 'demand paging'.

- When the CPU tries to refer a page which is not available in the main memory, 'page fault' occurs. A signal is sent to the OS regarding the page fault. The OS locates the page into the Logical address space and loads it into the main memory frame. If the memory frames are full, appropriate page replacement algorithm is used. The respective page table entries are also updated accordingly.
- The time taken to service a 'page fault' is called 'page fault service time'.



8.3 Performance of virtual Memory

8.3.1 Temporal Issues

Let,

Page fault service time = 'S'

Main memory access time = 'M'

Page fault rate = 'P' where $0 \leq P \leq 1$

The effective memory access time is formulated as-

$$EMAT = P \times S + (1 - P) \times M$$

(Assume, page table access time is neglected)

8.3.2 Demand Paging with TLB

Assume, TLB hit ratio = 'h'

TLB access time = 'c'

Page fault service time = 'S'

Main memory access time = 'M'

Page fault rate = 'P' where $0 \leq P \leq 1$

The effective memory access time is formulated as-

$EMAT = h(c + m) + (1 - h)(c + (P \times S + (1- P) \times M))$

8.3.3 Page Replacement Algorithms

FIFO

- When a page fault occurs and all the memory frames are full, FIFO algorithm replaces the oldest page to allocate the page referred by the CPU.
- It is implemented using a queue or time-stamp on pages.
- Sometimes, even on increasing the number of frames, the page fault rate increases. This situation is called Belady's Anomaly.

LRU

- LRU algorithm replaces the page that has not been used for the longest period (least recently used page).
- It is implemented using a stack or counter.

Optimal

- Optimal algorithm replaces the page that will not be used for the longest period in future.
- For a fixed number of frames, optimal algorithm gives the least page fault rate.
- It cannot be implemented in real time as it requires future references.

Note

Reference String is a set of successively unique pages referred in a given list of virtual addresses.

8.3.4 Thrashing

When CPU utilization is low, the OS may increase the degree of multiprogramming. After a certain point, the throughput of the system will gradually decrease as the system spends more time in page replacements owing to the lack of frames. This phenomenon is called 'thrashing'.

8.3.5 Working Set Model

It is defined as the set of the unique pages referred during the past 'Δ' references.

- If Δ < (total number of frames), the OS can bring in more processes.
- If Δ > (total number of frames), the OS should instruct the mid-term scheduler to suspend some of the processes to avoid thrashing.

❑❑❑

9 FILE SYSTEMS

9.1 File

- Collection of logically related entities/records

9.1.1 Attributes

- File Name
- File Type
- File Size
- Location of file
- Generation date
- Last modified date
- Permission
- Owner/Author
- Password

9.2 File Context

File context is stored in file control block (FCB)

File will have various types–

- .doc
- .txt
- .pdf
- .exe
- .obj
- .dll
- .png
- .apk
- .xls/.xlsx
- .jpg
- .mp3
- .mp4
- .avi/.flv/.mkv/.3gp
- .c/.cpp/.java/.xml/.html

9.3 Operations performed on file

- create
- open
- write
- read
- hide
- save
- save as
- close
- copy
- paste
- cut
- delete
- rename
- send/share
- print

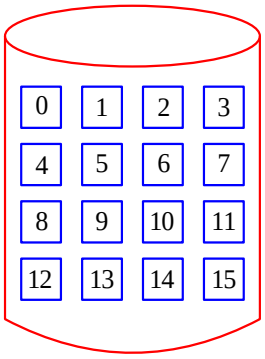
9.4 Access methods

- Sequential Access
- Random Access
- For better classification of files, files will be stored in directory.

9.5 Disk Space Allocation Methods

9.5.1 Contiguous Allocation

File	Starting Disk Block Address	Size w.r.t no. of DB
Abc.docx	2	4
Xyz.docx	9	5



- In contiguous allocation, whenever file is created, disk blocks are allocated in a continuous manner.
- Every file is associated with two parameters

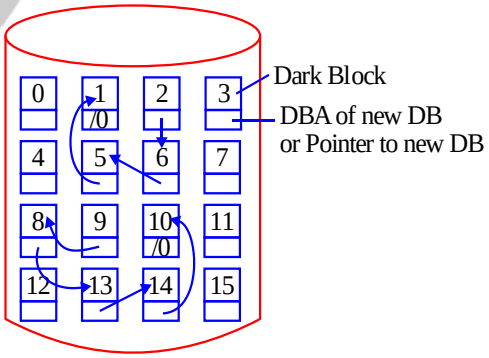
(1) Starting DBA
(2) Size

- Increasing the file size may not be possible always.
- It suffers from external fragmentation.
- Internal fragmentation may exist in the last disk block of the file.
- It supports both sequential and random access of file.

Starting disk block Address + Offset Value ⇒ Random location

9.5.2 Linked Allocation/Non-contiguous allocation

File	Starting DBA	Ending DBA
abc.docx	2	1
xyz.docx	9	10

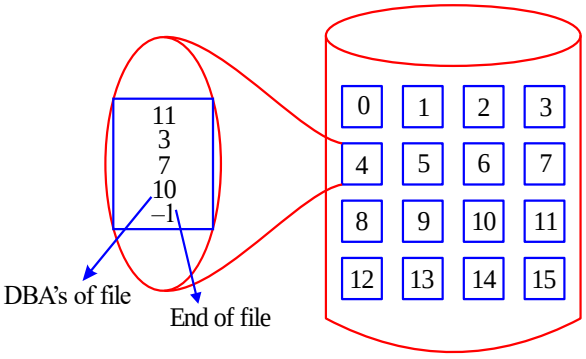


- In the linked allocation disk blocks are allocated in a non-continuous manner.
- Every file is associated with 2 parameters–

(1) Starting DBA
(2) Ending DBA

- Increasing file size is always possible if free disk block is available.
- There is no external fragmentation.
- Internal fragmentation may exist in last disk block of the file.
- There is an overhead of maintaining power in every disk blocks.
- If the pointer of any disk block is lost, the file will be truncated.
- It supports only sequential access of the file.

9.5.3 Indexed Allocation



File	Index Node
abc.dα	4

- In the indexed allocation, every file is associated with its own index node.
- Index node contains all the disk block address of the file.
- There is no external fragmentation but internal fragmentation may exist is the last disk block of file.
- If the file is very large, then one disk block may not be sufficient to store all the disk block addresses of the file.
- If the file is very small, then it is waste of using one entire disk blocks. Just to store, the addresses.

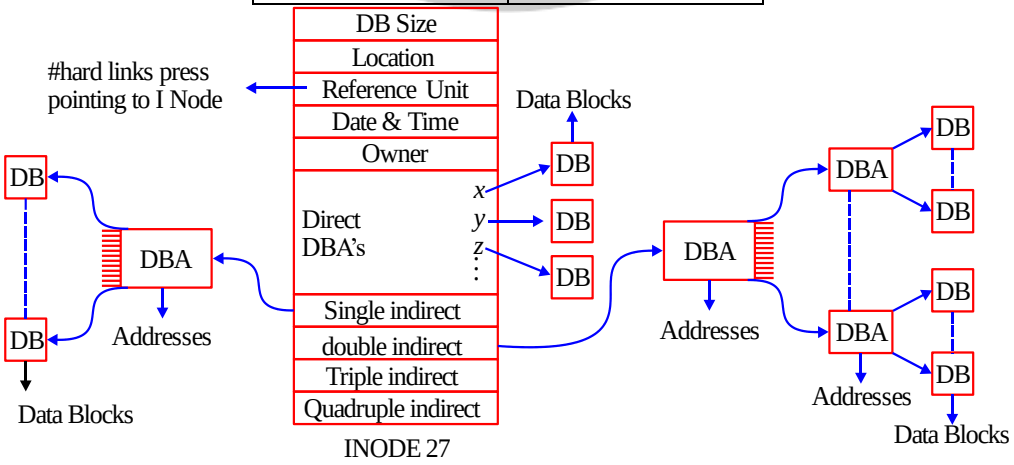
9.7 UNIX I-NODE IMPLEMENTATION

9.7.1 An extension of indexed allocation

$$\text{Total size of file system} = \left[\#\text{Direct DBA's} + \left(\frac{\text{DB Size}}{\text{DBA}}\right) + \left(\frac{\text{DB Size}}{\text{DBA}}\right)^2 + \left(\frac{\text{DB Size}}{\text{DBA}}\right)^3 + \ldots\ldots\right] \times \text{DB Size}$$

$\Downarrow$ #Single indirect. DBA $\Downarrow$ #Double Index DBA $\Downarrow$ #Triple Index DBA

File	I-NODE
abc.docx	27

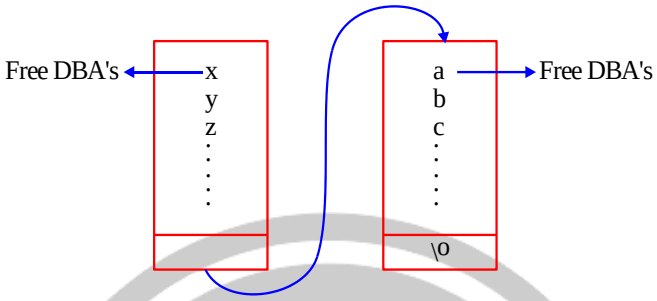


9.8 DISK FREE SPACE MANAGEMENT

Size of Disk = 20 MB
DB Size = 1 KB
DBA = 16 Bits = 2 B

Disk blocks available on given disk = $\frac{\text{Size of disk}}{\text{DB size}} = \frac{20 \times 2^{20} B}{2^{10} B}$ =20 K

9.8.1 Free List Approach



- In the free list approach, some disk blocks are used just to store the free disk block addresses.

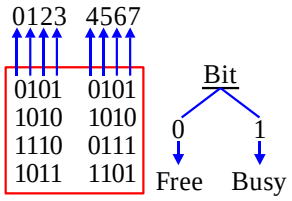
  # Disk block addresses possible to store in one disk block = $\frac{\text{DB Size}}{\text{DBA}} = \frac{2^{10}}{2} = 2^9$

  2^9 addresses $\rightarrow$ 1 disk block

  20 K addresses $\rightarrow \frac{1}{2^9} \times 2^{11} \times 10 \rightarrow 2^2 \times 10 = 40$ disk block

9.8.2 Bit Map Approach

- In Bit map approach, every disk block were be mapped with 1 binary bit



Free Disk Blocks

0, 2, 4, 6, 9, 11, ...

Busy Disk Blocks

1, 3, 5, 7, 8, 10, ...

Disk blocks we can map in 1 disk block = 1 K × 8 bits = 8 K bits.

8 K bits $\rightarrow$ 1

20 K bits $\rightarrow \frac{1}{8\cancel{K}} \times 20\cancel{K} = 2.4 = 3$

❑❑❑

10 DISK SCHEDULING

10.1 Goal of Disk Scheduling

- Enhanced throughput.
- Minimize the average seek time of the disk.
- Fair chance to all the I/O requests.

10.2 Disk scheduling Algorithms

10.2.1 FCFS

It serves the first request in the queue.

10.2.2 SSTF

It selects the request with minimum seek time for the current head position.

10.2.3 SCAN or Elevator Algorithm

The disk arm starts from the current head position and moves towards the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

10.2.4 C–SCAN

The disk arm starts from the current head position and moves towards the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and immediately returns to the beginning of the disk without servicing any requests at the return trip. The servicing then continues from the beginning of the disk again.

10.2.5 C–LOOK

It is similar to C-SCAN where the arm goes as far as the last request in each direction, then reverses the direction immediately, without first going all the way to the end of the disk.

Note:

- Performance depends on the number and types of requests.
- I/O requests can be influenced by the file allocation methods.
- Either SSTF or LOOK is a reasonable choice for the default algorithm.

10.2.6 Comparison Among the Disk Scheduling Algorithms

Algorithm	Advantages	Disadvantages
First Come First Served	• Easy to implement • Fair chance to all the I/O requests • It doesn't suffer from starvation	• Seek time is not optimized. • It doesn't maximize throughput.
Shortest Seek Time First	• It tends to minimize the arm movement. • It has better throughput than FCFS.	• It may suffer from starvation.
SCAN/LOOK	• It eliminates starvation. • It works well with light to heavy loads.	• It is complex to implement. • Increased overhead. • It needs a directional bit to keep track of the head direction.
C-SCAN/C-LOOK	• No directional bit is required. • It works well with light to heavy loads.	• It is complex to implement. • Increased overhead.

❑❑❑

OUR
YOUTUBE CHANNELS

GATE WALLAH
EE, EC & CS

GATE Wallah

GATE WALLAH
ME, CE & XE

GATE Wallah (English)

ACCESS QUALITY CONTENT
For Free

Database Management Systems

Database Management Systems

INDEX

1 DATABASE DESIGN

1.1 Introduction

A database is an organized collection of structured data or related data.

1.2 Limitations of File System

- The physical details of the data to access the database are managed by the user.
- File system can be used to manage small database.
- When database is large, the operating system fails to control the concurrency.
- Only single user can access the whole data of the file system at a time.

1.3 Integrity Constraints Of RDBMS

- According to codd, data in database file must be stored in tabular format (set of row's and column's).
- According to codd, no two row's/records of the table should be equal.

1.3.1 Arity

Number of attributes of database table.

1.3.2 Cordinality

Number of records of database table.

1.4 Keys in database

- **Condidate Key:** A minimal set of attributes that differentiate records/row's of DB table uniquely.
- **Primary Key :** One of the candidate key whose field value cannot be null.
- **Simple Candidate Key:** A candidate key with only one attribute.
- **Compound Candidate Key:** A candidate key with atleast two attributes.
- **Overlapped Candidate Key:** Two or more candidate key with some common attribute.
- **Prime attribute:** The attribute that belongs to some candidate keys of a relation.
- **Non-prime attribute:** The attribute that does not belongs to any of the candidate keys of the relation.

Example :

Consider a relation R (ABCDE)

1. Assume candidate key : {AB, BC}

2. The above candidate keys are compound and overlapped.
3. Prime attribute {A, B, C}
4. Non-prime attribute : {D, E}

1.4.1 Difference between Primary key and Alternative key

Primary Key	Alternative Key
1. Any one candidate key whose field value can not be null. 2. Atmost one primary key allowed for any DB table.	1. All candidate key of relation except primary key, whose field value can be null. 2. Many (o or more) alternative keys are allowed for DB table.

Note:
For any RDBMS table there must be atleast one candidate key, whose field value can not be null.

(Unique + Not Null) ≠Primary key

1.4.2 Super key

The set of attributes which can differentiate records/tuples uniquely or super set of candidate key.

Example 1: R (ABCD) with CK = {A}

∴ Super key = CK · [Any subset of other attributes (BCD)]

= A · [2^3] = 8 Super key.

Example 2 : Let R be the relational schema with n-attibutes, R (A_1, A_2, ... A_n) then number of superkeys possible:

(I) With (A_1) as candidate key : 2^{n-1}

(II) With {A_1, A_2} as candidate key : $2^{n-1} - 2^{n-2} + 2^{n-1}$

(III) With {A_1A_2, A_3A_4} as candidate key: $2^{n-2} - 2^{n-4} + 2^{n-2}$

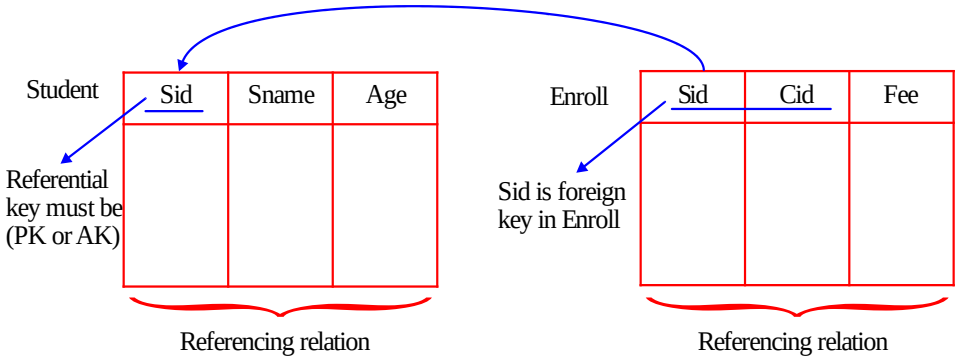
(IV) The maximum number of super keys possible when each attribute of R is candidate key : 2^{n-1}

(V) The minimum number of super key possible when all the attributes combined form single candidate key:
1 {A_1A_2 ..... A_n : candidate key}

1.5 Referential Integrity Constraints

1.5.1 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 :** [May cause violation]
   (a) On delete no action : Means if it cause problem on delete then deletion is not allowed on table.
   (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 value from referenced table then it will try to set the null values in place of that value in referencing table.

Note:
If foreign key field is not null attribute then "On delete set null" is same as" on delete no action."

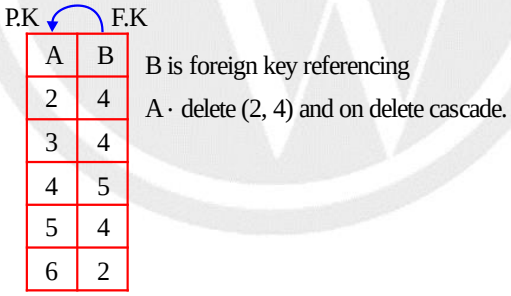
3. **Updation :** [May cause violation]
   (a) On update no action
   (b) On update cascade
   (c) On update set null

Referencing Relation

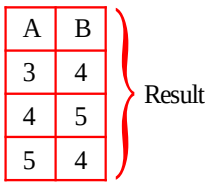
1. **Insertion :** [May cause violation]
2. **Deletion :** No violation
3. **Updation :** [May cause violation]

Note
If integrity violation occurs because of insertion or updation in referencing table then restrict insertion and updation.

Example:



So, If we delete (2, 4) then PK "2". gets deleted from the table and all the tuples in which B is referencing PK.2" also gets deleted.



1.6 Database Design Goals

1. 0% redundancy
2. Loss-less join
3. Dependency preservation

1.6.1 Functional Dependency

Let R be the relational shcema and x, y be the non-empty set of attributes. Consider t_1, t_2 are any tuples of relation then $x \to y$ (y functionally determined by x):

If $\forall t_1, t_2\ t_1 \cdot x = t_2 \cdot x$ then $t_1 \cdot y = t_2 \cdot y$

1.6.2 Types of Functional Dependency

1. Trivial Functional Dependency

Consider relational schema R(ABCD)

A FD $x \to y$ is trivial FD only if $x \supseteq y$.

Example :

$AB \to A$

$A \to A$

$B \to B$

2. Non-trivial Functional Dependency

Consider relational schema R(ABCD)

A FD $x \to y$ is non-trivial only if

$x \cap y = \phi$ means no common attributes between x and y attribute sets.

Example :

$A \to B$

$AB \to CD$

3. Semi-Non-Trivial Functional Dependency

A combination of both trivial FD and non-trivial FD.

Example :

$A \to AB \equiv \{A \to A, A \to B\}$

1.6.3 Attribute Closure (X^+)

The set of all possible attributes determined by x.

Example :

R (ABCD) $\{A \to B, B \to C, C \to D\}$

$\therefore$ $(A)^+ = \{A, B, C, D\}$

1.6.4 Armstrong's Axioms

1. **Reflexive :** If $x \supseteq y$ then $x \to y$ or $x \to x$
2. **Transitivity :** If $x \to y$ and $y \to z$ then $x \to z$
3. **Augmentation :** If $x \to y$ then $xz \to yz$
4. **Splitting :** If $x \to yz$ then $x \to y$, $x \to z$
5. **Union :** If $x \to y$ and $x \to z$ then $x \to yz$
6. **Pseudo transitivity :** If $x \to y$, $yw \to z$ then $xw \to z$

1.6.5 Finding Candidate Key [Minimal Super Key]

X is candidate key of R
If and only if

1. x must be super key of relation R
   $(x)^+$ = {determine all attributes of R}
2. No proper subset of 'x' is super key of relation R.
   $\forall\ y \subset x : (y)^+$ = {not determine all attributes of R}

- $x \rightarrow y$ is a non-trivial FD in R with y is a prime attribute then relation R has atleast two candidate key.

(x → y) : Non – trivial FD
↓
Prime-attribute

1.7 Membership of FD

A FD $x \rightarrow y$ is member (logically implied) of FD set F if and only if $(x)^+$ should determine 'y' in FD set F.

Example : Given FD Set F : { ….}
↓
x → y FD belongs to F or not ?
↓
$x^+ = \{\ldots y \ldots\}$
then x → y is member of F.

1.7.1 Testing of Two FD Sets whether they are equal or not

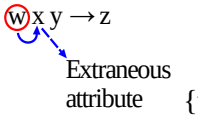
F and G FD sets equality test:
F and G FD sets logically equal if and only

1. **F cover's G :** All FD's of G set must be member's of F set.
   $F \supseteq G$
2. **G cover's F :** All FD's of F set must be member's of G set.
   $F \subseteq G$

F Cover G	G Cover F	Result
True	True	$F \equiv G$
True	False	$F \supset G$
False	True	$F \subset G$
False	False	F & G are not comparable

1.7.2 Minimal Cover or Canonical Cover

- Minimal Cover of given FD Set (F) is minimum possible FD's (Fm), which is logically equal to 'F' : (F = Fm)
- Remove extraneous attribute (useless attribute) from each determinant of FD set F.



$\{wxy \rightarrow z, w \rightarrow x\} \equiv \{wy \rightarrow z, w \rightarrow x\}$

- Every FD must be simple (RHS of any FD should have single attribute).

Example : F = {A → CD} then {A → C, A→ D}

- FD set must be non-redundant FD. (eliminating unnecessary FD's)

Example : F = {A → B, B → C, A→ C} then

F = {A → B, B→ C} because A → C is redundant.

Note:

Minimal Cover of FD Set (F) may not unique, but all minimal cover's logically equal.

$\therefore \quad \boxed{Fm_1 \equiv Fm_2 = F}$

1.8 Normalization

Normalization used to eliminate/reduce redundency in DB relations.

The normalization is especially meant to eliminate the following anomalies:

- Insertion anomaly
- Deletion anomaly
- Update anomaly
- Join anomaly

1.8.1 Normalization of DB table

Decompose relation into two or more sub-relations in-order to reduce or eliminate redundancy and DB anomalise.

1.8.2 Properties of Decomposition

- **Loss-less Join decomposition:** Relational Schema R decomposed into R1, R2, R3, ... Rn sub-relations.
- In general [R1 ⋈ R2 ⋈ R3 ..... Rn] ⊇ R

1. If [R_1 ⋈ R_2 ⋈ R_3.....⋈ R_n] ≡ R then loss-less join decomposition.
2. If [R_1 ⋈ R_2 ⋈ R_3 ..... ⋈ R_n] ⊃ R then lossy join decomposition.

Example :

Let R be the relational schema decomposed into R_1 and R_2.

Given decomposition is loss-less join if–

1. $R_1 \cup R_2 = R$ (all attributes covers)
2. $R_1 \cap R_2 \neq \phi$
3. $\underbrace{R_1 \cap R_2 \to R_1}_{R_1 \cap R_2 \text{ is SK of } R_1}$ or $\underbrace{R_1 \cap R_2 \to R_2}_{R_1 \cap R_2 \text{ is SK of } R_2}$

1.8.3 Dependency Preserving Decompostion

Relational Schema R with FD set F decomposed into R_1, R_2, ... R_n sub relations

Assume F_1, F_2, ... F_n FD sets of sub relations respectively.

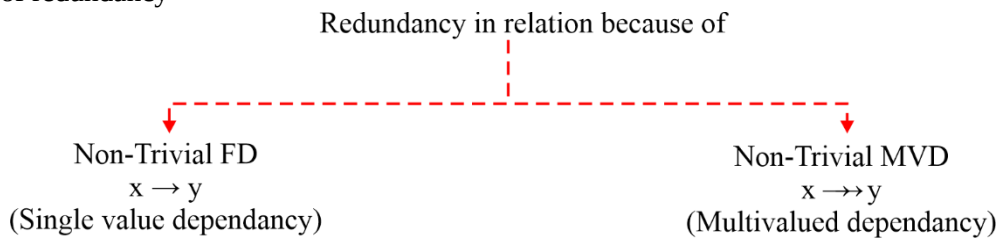
In general

[$F_1 \cup F_2 \cup$ ..... F_n] ⊆ F

1. If [$F_1 \cup F_2 \cup$ ..... F_n] ≡ F then dependency preserving decomposition.
2. If [$F_1 \cup F_2 \cup$ ..... F_n] ⊂ F then not dependency preserving decompostion.

1.9 Normal Forms

Used to find degree of redundancy



- BCNF relations have 0% redundancy over FD's where as 4NF relations have 0% redundancy over FD and MVD.
- To Eliminate redundancy over Non-Trivial FD, relation should decompose into BCNF but BCNF may not avoid the redundancy due to MVD.
- To eliminate redundancy over non-trivial MVD, relation should decompose into 4NF (4^{th} Normal Form).

1.9.1 First Normal Form

- Default NF of RDBMS relations.
- Relation (DB Table) R is in 1 NF if and only if no multivalued attributes in R. [Every attribute of R must be atomic/single valued].

Note:
Degree of redundancy is very high in 1NF relation.

Important Point 1

- $x \rightarrow y$ FD forms redundancy in relational schema R if and only if –
  (i) Non-Trivial FD
  and
  (ii) X is not super key.
- $x \rightarrow y$ FD not forms any redundancy in relation R if and only if–
  (i) Trivial FD [$x \supseteq y$] or Semi-trivial
  or
  (ii) x : super key

1.9.2 Second Normal Form (2NF)

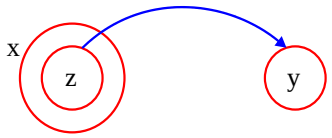
Relational Schema R is in 2 NF iff

- R should be 1NF
- R should not contain any partial dependency

Partial Dependency

Let R be the relational schema and x, y, z are non-empty set of attributes.

X : Any candidate key of R.
Y : Non-prime attribute
Z : Proper subset of candidate key



$\therefore$ $z \rightarrow y$ is said to be partial dependency iff–

- z is proper subset of candidate key
- y should be non-prime attribute.

Important Point 2

$$\underbrace{[\text{Proper subset of Candidate key}]}_{\text{Not super key}} \rightarrow [\text{Non-Prime attribute}]$$

The above FD forms redundancy in R.

1.9.3 Third Normal Form (3NF)

Relational schema R is in 3NF iff every non-trivial FD $x \rightarrow y$ in R with–

(i) x must be super key (SK)
or
(ii) y must be prime attribute.

$\therefore$ {SK $\rightarrow$ Prime/Non-Prime, (Not SK) $\rightarrow$ Prime attribute}

Important Point 3

3NF allow FD set:
[Proper subset of candidate key] $\rightarrow$ [Proper subset of other candidate key]
which forms redundancy in R.

1.9.4 Boyce codd Normal Form (BCNF)

Relational schema R is in BCNF iff every non-trivial FD "$x \rightarrow y$" with x must be super key.
$\therefore$ Prime/ Non-prime attributes must be determine by super key.

Important Point 4

If R is in 3NF but not BCNF then [Proper subset of candidate key] $\rightarrow$ [Proper subset of other candidate key] must exists in R.

Important Point 5

If relational shcema are with only simple candidate key then R always in:

I. 2NF
II. May or may not 3NF/BCNF

Reason : [Proper subset of candidate key] $\rightarrow$ [Non-prime attribute]
From the above statement, we can conclude that "partial dependency" not possible if all CK's are simple candidate key.

Important Point 6

If relational schema R with only prime attribute (No non-prime attribute in R) then R always in:

I. 3NF
II. May or may not BCNF

Important Point 7

If relational schema R with no non-trivial FD's then R always in BCNF.

1.9.5 Database design Table

DB design goal	1NF	2NF	3NF	BCNF
1. Loss-less jon decomposition	Yes	Yes	Yes	Yes
2. Dependency preserving decomposition	Yes	Yes	Yes	May not
3. O% redundency	No	No	No	Yes [Over FD]

- Every relation possible to decompose into 1NF, 2NF, 3NF, BCNF with loss-less join decomposition.
- Every relation possible to decompose into 1NF, 2NF, 3NF with Dependency preserving decomposition.
- Not every relation can decompose into BCNF and 4NF with dependency preserving decompostion.
- Most accurate normal form to design simple database is 3NF because every relation is possible to decompose into 3NF with loss-less join and dependency preserving.

❑❑❑

2 ER MODEL

2.1 Introduction

Entity relationship diagram used to represent diagrametic design (High level design) of DB.

2.2 Main components in ER Diagram

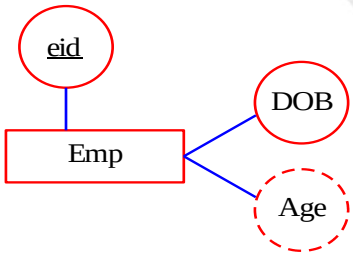
(i) Attributes
(ii) Entity sets
(iii) Relationship sets

(a) Entity sets

It is a set of entities of the same type denoted by a rectangular box in ER diagram. Entity can be identified by a list of attributes which are placed in ovals.

Represent by : 

Example :



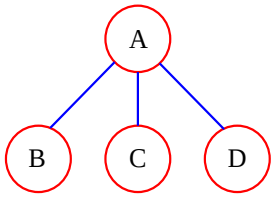
(b) Attributes

(i) Attribute : 

(ii) Key attribute : 

(iii) Derived attribute : 

(iv) Composite attribute: Attribute which can be represented as two or more attributes.

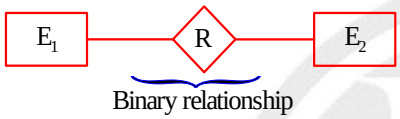


(v) Multivalued attribute: 

(c) Relationship set: It is used to relate two or more entity set.

Represented by : 

Example:



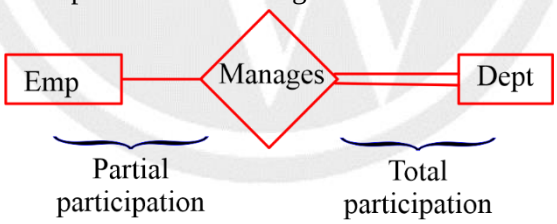
2.3 Participation

- If every entities of entity set are participated with relationship set then it is total participation (100% participation) otherwise it will be partial participation (< 100% participation)

Example :

Consider Emp and Dept entity set.

Manages relationship set such that each dept must have manager.

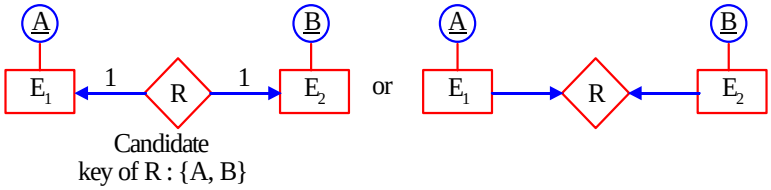


2.4 Mapping [Cardinality constraints of relationship set]

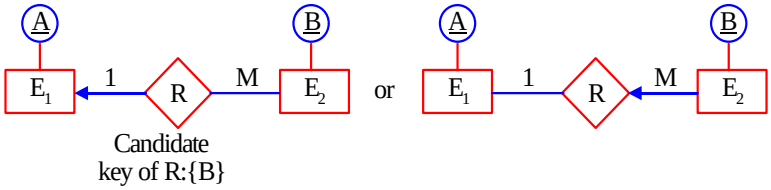
One mapping : Atmost one (0 or 1)

Many mapping : 0 or more (0 ..... *)

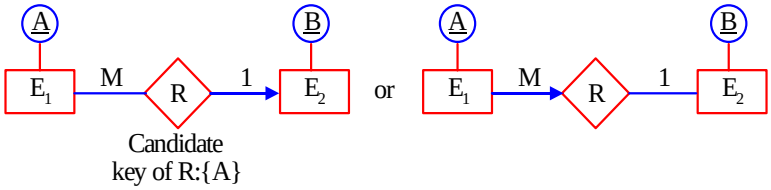
1. Binary Relationship Mapping (One : One)



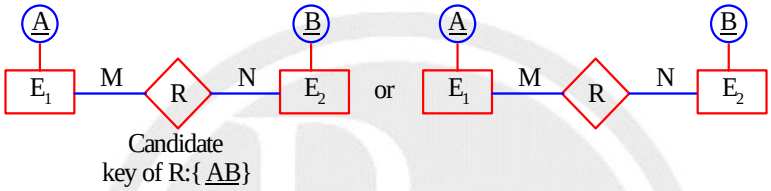
2. Binary Relationship Mapping (One : Many)



3. Binary Relationship Mapping (Many to One)



4. Binary Relationship Mapping (Many to Many)



2.5 RDBMS Design of Given ER Diagram

(For binary relationship)

1. Partial participation on both side of binary relationship

- **One to Many :** Merge relationship set towards many side. So, 2 relational tables.
- **Many to one :** Merge relationship set towards many side. So, 2 relational tables.
- **One to one :** Merge relationship set any one side. So, 2 relational tables.
- **Many to Many :** Separate table for each entity set and relationship set. so, 3 relational tables.

2. Full participation on "one" side of many to one relationship

Merge the entities and relationship set into single relational table. So, 1 table.

3. Full participation on "Many" side of Many-to-one relationship

Merge relationship set towards many side. So, 2 relational tables.

4. Full participation on any "one" side in one-to-one relationship

Merge the entity sets and relationship set into single table. So, 1 table.

5. Full participation on any "Many" side of Many-to-Many relationship

Merge relationship set towards any "Many" side of relationship. So, 2 table.

6. Full participation on both side of relationship

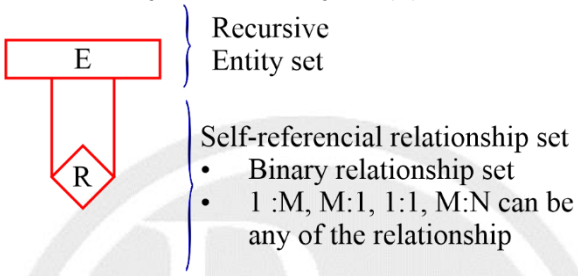
- 1:1
- 1:M
- M:1
- M:N

Merge the entity sets and relationship set into single relational table so, 1 relational table

2.6 Self-Referencial Relationship Set

(Recursive entity set)

Entities of entity set (E) related to some other entity of same entity set (E).



2.7 Weak Entity Set and Weak Relationship Set

The entity set with no key. (Attributes of weak entity sets are not sufficient to differenciate entities uniquely).

Points:

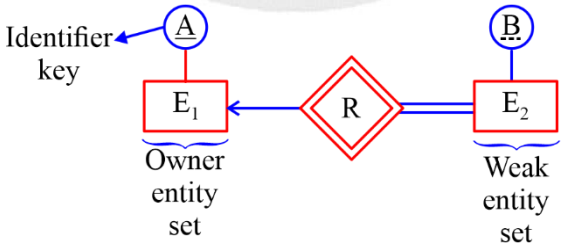
(a) For each weak entity set there must be owner entity set, which is strong entity set.

(b) Relationship set between weak entity set and identifier entity set is also "weak relationship set".

(c) The participation towards weak entity set end must be "total participation".

(d) The mapping between identifier entity set and weak entity set must be one : many (1 : M)

Example:



Note:

Weak entity set and multivalued attributes allowed to represent in ER diagram but not allowed in RDBMS table.

❑❑❑

3 RELATION MODEL AND SQL

3.1 Procedural Query Language and Non-procedural Query Language

Procedural Query Language	Non-procedural Query Language
Formulation of how to access data from the database table and what data required to retrieve from DB tables. "Relational Algebra"	Formulation of what data retrieve from DB tables. "Relational Calculus"

3.2 Relational Algebra

(Always generate distinct tuples)

Relational algebra refers to a procedural query language that takes relation instances as input and returns relation instances as output.

3.2.1 Basic operators

π : Projection operator

σ : Selection operator

× : Cross-product operator

U : Union

– : Set difference

ϱ : Rename operator

3.2.2 Derived operators

∩ : Intersection {using "_"}

⋈ : Join {using X, σ}

/ or ÷ : Division {using π, x, –}

I. π : Projection

- $\pi_{attribute\_name}$ (R): It is used to project required attribute from relation R.
- $\sigma_{Condition(P)}$ (R): It is used to select records from relation R, those satisfied the condition (P).

Example:

R

A	B	C
8	4	5
2	4	5
7	4	6
3	5	5

$\pi_{B,C}(R)$:

B	C
4	5
4	6
5	5

$\sigma_{A>S}(R)$:

A	B	C
8	4	5
7	4	6

II. Cross product (×):

- R × S : It result all attributes of R followed by all attributes of S, and each record of R paired with every record of S.
- Degree (R × S) = Degree (R) + Degree (S)
- $|(R \times S)| = |R| \times |S|$

Note:
- Relation R with n tuples and
- Relation S with 0 tuples then

number of tuples in R × S = 0 tuples

3.3 Join (⋈)

I. Natural join (⋈)

$R \bowtie S \equiv \pi_{\text{distinct attributes}}(\sigma_{\text{equality between common attributes of R and S}}(R \times S))$

Example:

- T_1 (ABC) and T_2 (BCDE)

$$\therefore \quad T_1 \bowtie T_2 = \pi_{ABCDE}\left(\begin{matrix}\sigma_{T_1.B=T_2.B}(T_1 \times T_2) \\ \cap T_1.C=T_2.C\end{matrix}\right)$$

- T_1 (AB) and T_2 (CD)

$\therefore \quad T_1 \bowtie T_2 \equiv T_1 \times T_2 = \pi_{ABCD}(T_1 \times T_2)$

Note:
Natural join equal to cross-product if join condition is empty.

II. Conditional Join ($\bowtie_c$)

- $R \bowtie_c S \equiv \sigma_c (R \times S)$

III Outer Joins:

(a) LEFT OUTER JOIN

R ⟕ S : It produces

(R ⋈ S) ∪ {Records of R those are failed join condition with remaining attributes null}

(b) RIGHT OUTER JOIN (⟖)

R ⟖ S : It produces]

(R ⋈ S) ∪ {Records of S those are failed join condition with remaining attributes null}

(C) FULL OUTER JOIN ($\bowtie$)

$R \bowtie S = (R \bowtie S) \cup (R \bowtie S)$

3.4 Rename operator (ϱ)

- It is used to rename table name and attribute names for query processing.

Example:

(I) Stud (Sid, Sname, age)

ϱ(Temp, Stud) : Temp (Sid, Sname, age)

(II) $\varrho_{I,N,A}$ (Stud) : Stud (I, N, A)

All attributes renaming

(III) $\varrho_{\substack{Sid \to I \\ age \to A}}$ (Stud) : Stud (I, Sname, A)

$_{age \to A}$

Some attribute renaming

Example:

Retrieve eicls of female employees whoose salary more than every male employee?

Result :

$$\pi_{eid}\left(\sigma_{(gen=Female)}^{(Emp)}\right) - \pi_{eid}\left(E \bowtie \delta_{I,S,G}\left(emp\right)\right)$$

$$\begin{pmatrix} gen = female \\ \cap \\ G = male \\ \cap \\ Sal \le S \end{pmatrix}$$

3.5 Division

- It is used to retrieve attribute value of R which has paired with every attribute value of other relation S.
  $\pi_{AB}(R)/\pi_{B}(S)$: It will retrieves values of attribute 'A' from R for which there must be pairing 'B' value for every 'B' of S.

Expansion of '/' by using basic operator

- Example: Retrieve sid's who enrolled every course.

Result:

π_{sidcid}(Enroll)/ π_{cid}(Course)

Step 1: Sid's not enrolled every course of course relation.

(Sid's enrolled proper subset of course)

$\pi_{sid}((\pi_{sid}(Enroll) \times \pi_{cid}(course)) - \pi_{sidcid}(Enroll))$

Step 2:

[sid's enrolled every course] = [sid's enrolled some course] – [sid's not enrolled every course]

$\therefore \quad \pi_{sidcid}(E) / \pi_{cid}(c) = \pi_{sid}(E) - \pi_{sid}((\pi_{sid}(E) \times \pi_{cid}(C) - \pi_{sidcid}(E))$

3.6 Set operator

U: Union operator

– : Except minus

$\cap$: Intersection operator

- To apply set operations relations must be union compatable.
- R and S relations union compatable
- If and only if-

  (i) Arity of R equal to Arity iof S and

  (ii) Domain of attributes of R must be same as domain of attributes of s respectively.

Example 1:

$\pi_{Sid\ Sname}(.....) \cap \pi_{Sid}(.....)$

{Arity not same so, set operation not allowed}

Example 2:

$\pi_{Sid\ S\ name}(.....) \cap \pi_{Sid\ marks}(.....)$

{Arity same but Sname domain is different from marks so, not allowed}

Example 3:

$\pi_{Sid\ Sname}(.....) \cap \pi_{Stud\ ID,\ Stud\ name}(.....)$

{Arity and domains are same so, allowed for set operation}

- 1. Set operation on relation:

R

A
2
2
2
3

S

B
2
2
4

$R \cup S : \{x / x \in R\ V\ x \in s\} \equiv$

A
2
3
4

$R - S : \{x / x \in R \wedge x \notin S\} \equiv$

A
3

$R \cap S : \{x / x \in R \wedge x \in S\} \equiv$

A
2

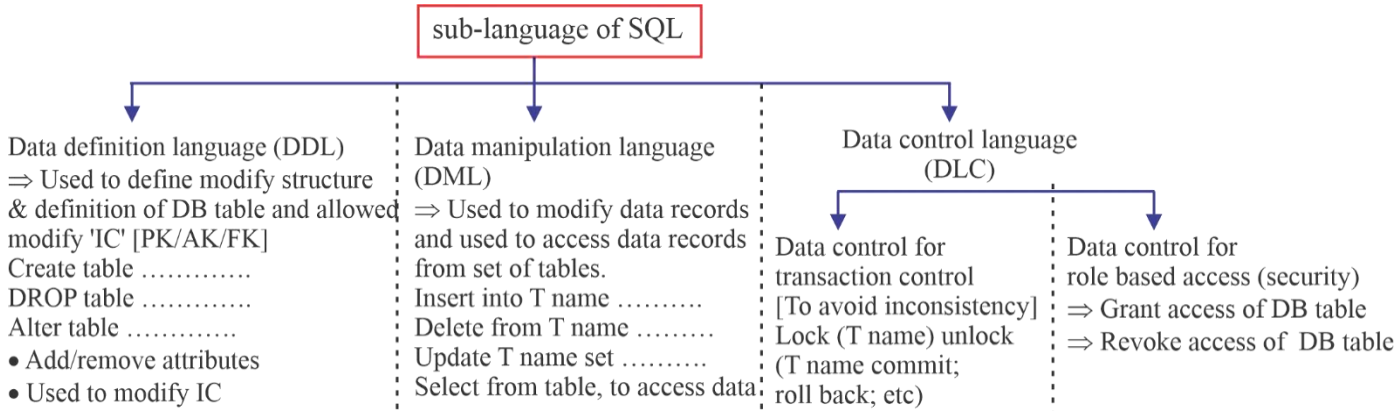
3.7 Cardinality Table

Assume relation R with n Distinct tuples and relation S with m distinct tuples:

RA Expression	Cardinality
R × S	n * m tuples
R ⋈ S	{0 to n * m tuples}
R =⋈ S	{n to n* m tuples}
R ⋈= S	{m to n*m tuples}
R =⋈= S	{max (n,m) to n*m tuples}
R∪S	{max(n,m) to n + m}
R∩ S	{0 to min (m, n}
R–S	{0 to n tuples}
R/S	{0 to [n/m] tuples}

3.8 SQL

3.8.1 [Structure Query Lang]



3.8.2 Relational Query and SQL query

- SQL : SELECT DISTINCT $A_1, A_2, \ldots\ldots A_n$

FROM $R_1, R_2, \ldots\ldots R_n$
WHERE P;

Equivalent RA : $\pi_{A_1, A_2 \ldots\ldots A_n}\left(\sigma_p\left(R_1 \times R_2 \times \ldots\ldots R_n\right)\right)$

- SELECT ≡ projection of RA (π)
  FROM ≡ Cross-product (x)
  WHERE ≡ Selection operator of RA (σ)

3.8.3 Basic SQL Clauses

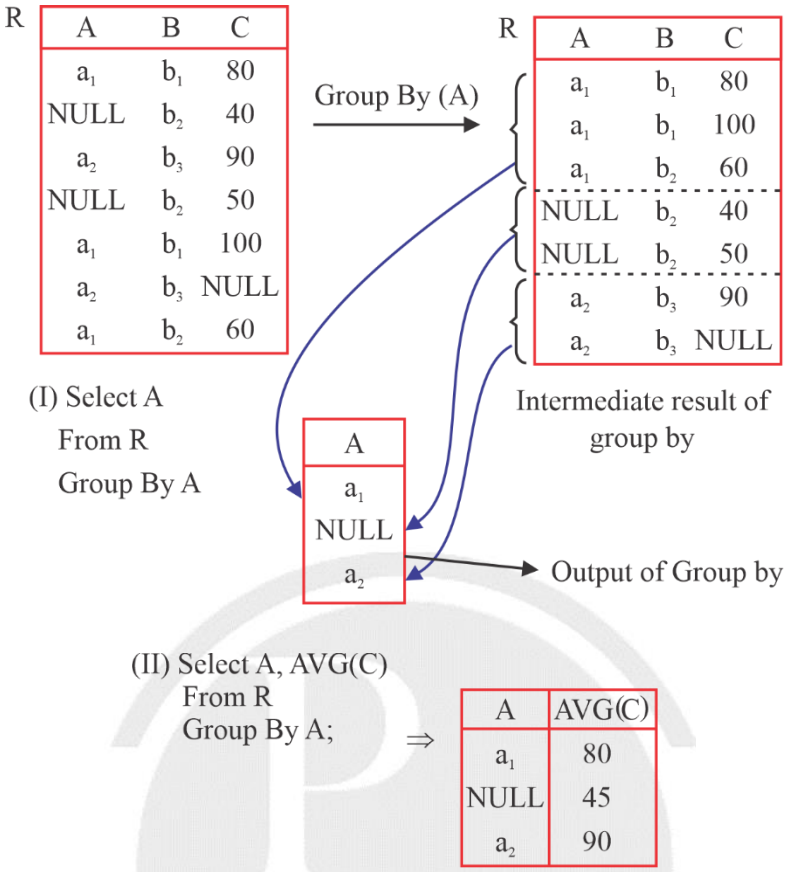
SELECT DISTINCT $A_1, A_2, \ldots\ldots A_n$
FROM $R_1, R_2, \ldots\ldots R_n$
WHERE condition
GROUP BY (attributes)
HAVING condition
ORDER BY (attributes) (DESC)

3.8.4 Execution Flow

FROM ⇒ cross-product of relations
↓
WHERE ⇒ Selection operator (σ) to apply condition for each record
↓
GROUP BY
↓
HAVING
↓
SELECT
↓
DISTINCT
↓
ORDER BY

3.8.5 GROUP BY:

- It is used to group records data a based on specific attribute.
- It GROUP BY clause used then
  (a) Every attribute of GROUP BY clause must be selected in SELECT clause.
  (b) Not allowed to select any other attribute in SELECT clause.
  Allowed to select aggregate function along with GROUP BY attribute in SELECT Clause.



3.8.6 HAVING Clause

- HAVING clause must be followed by GROUP BY clause.
- HAVING clause used to select groups those are satisfied having clause condition.
- HAVING clause condition must be over aggregation function such as some ( ), Every ( ) etc. but not allowed direct attribute comparison

Example:

1.
```
SELECT A
FROM R
GROUP BY A
HAVING AVG (c) > 60;
```

2.
```
SELECT A, AVG (c)
FROM R
GROUP BY A
HAVING some (c) > 50;
```

3.
```
SELECT A
FROM R
GROUP BY A
HAVING C> 60;
```
} Error

⇩
Not allowed

Note:
WHERE clause condition tested for each record but HAVING clause condition tested for each GROUP.

3.8.7 WITH Clause

- WITH clause is used to create sub-query result that can be re-used many times in query processing.

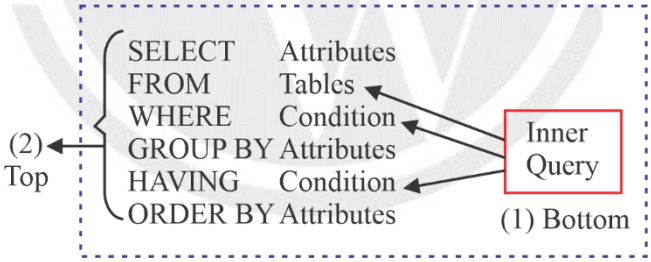
Example:

WITH Temp (A_1, A_2) as
(SELECT T_1 . A_1, T_2 . A_2
FROM Emp T_1, Emp T_2
WHERE T_1 . A_1 < T_2 . A_1)

Note:
Every Query which is written by using HAVING clause can be re-written by using WHERE clause.

3.8.8 Nested Query Without Co-relations

- Inner query independent of outer query.



- Execution flow must be Bottom to Top mean first bottom (Inner query) evaluated then outer query executed.

3.8.9 Co-related Nested Query

- In Nested Co-related query inner query uses attributes from outer query tables.
- In Co-related Nested query inner query allowed in WHERE, HAVING clause of outer query.

Example:

```
SELECT A
FROM R
WHERE (SELECT count(*)
        FROM S
        WHERE S.B < R.A) < 5;
```

Important point 1

- If co-relation in WHERE clause then inner query re-computes for each record of outer query From clause.
- If correlation in HAVING clause then inner query re-computes for each group of outer query.

3.9 Function used for nested Queries

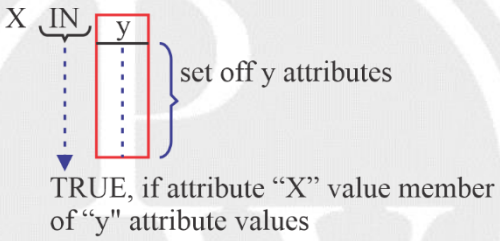
1. IN/ NOT IN
2. ANY
3. ALL

} Best suitable for non- co-related query

4. EXISTS/NOT EXISTS } Best suitable for co-related query

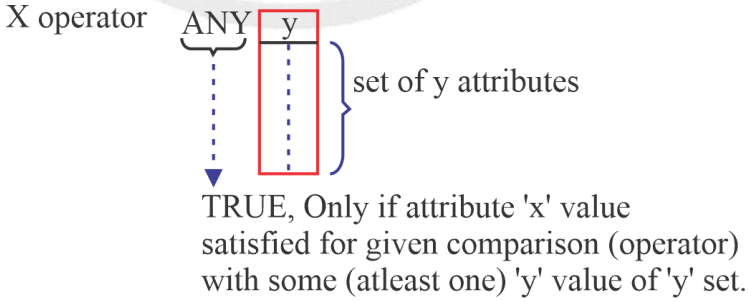
1. IN Function

It is used for membership testing.

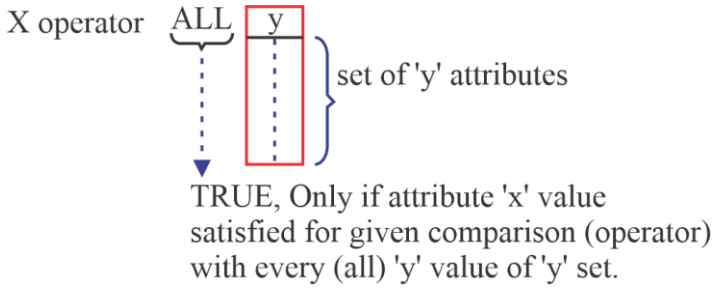


Note:
Queries of "equi Join" can implement by IN function ≡ (R ⋈ S)

2. ANY Function (operator "<, ≤, >, ≥, =, < >")



3. ALL Function



Note:
ANY function can be used as queries of conditional join query.

Example:

$$\pi_A\left(R \underset{R.A > S.B}{\bowtie} S\right) \equiv \text{SELECT A FROM R WHERE R.A > ANY (SELECT B FROM S)}$$

$$\pi_A(R) - \pi_A\left(R \underset{R.A \le S.B}{\bowtie} S\right) \equiv \text{SELECT A FROM R WHERE R.A > ALL (SELECT B FROM S)}$$

Important point 2

- IN function equal to '= ANY' function

$$X \text{ IN } Y \equiv X = \text{ANY } Y$$

- IN function not equal to '= ANY' if more than one attribute used for IN comparison.

$$(A,B) \text{ IN } (C\ D) \neq (A,B) = \underbrace{\text{ANY}}_{\text{Not allowed}} (C\ D)$$

- NOT IN function equal to "< > ALL".

$$X \text{ NOT IN } Y \equiv X <> \text{ALL } Y$$

4. **EXISTS** clause
   - It is used to test result of inner query is empty or not empty.
   -

EXISTS (Query)

TRUE, if result of inner query is non-empty that is atleast one record in result of inner query.

Example:

```
SELECT R.A
FROM R
WHERE EXISTS (SELECT *
              FROM S
              WHERE R.A > S.B);
```

Atleast one record of S relation satisfy R.A > S.B

Note:
EXISTS function can use as conditional join queries with join condition in inner query WHERE clause.

Emp (eid sal dno. Gen) Retrieve eids of femal's whose salary more than (some) male emp.

$$\text{RA: } \pi_{eid}\, \sigma_{gen\ =\ Female}(Emp) \bowtie_{Sal > S} \rho_{I,S,D,G}\left(\sigma_{gen=male}(emp)\right)$$

SQL [JOIN Query]

```
SELECT DISTINCT T1 . eid
FROM EMP T1, Emp T2
WHERE
T1.gen = Female and
T2.gen = male and
 T1.sal > T2 . sal
```

SQL [Mested Query]

```
SELECT Eid
FROM Emp
WHERE gen = Female
and sal > ANY (SELECT sal
                FROM Emp
                WHERE gen = male);
```

SQL [Co – related Nested Query]

```
SELECT Eid
FROM Emp T1
WHERE T1.gen = Female and
EXISTS (SELECT *
        FROM Emp T2
        WHERE T2.gen = male and T1. Sal > T2 . sal);
```

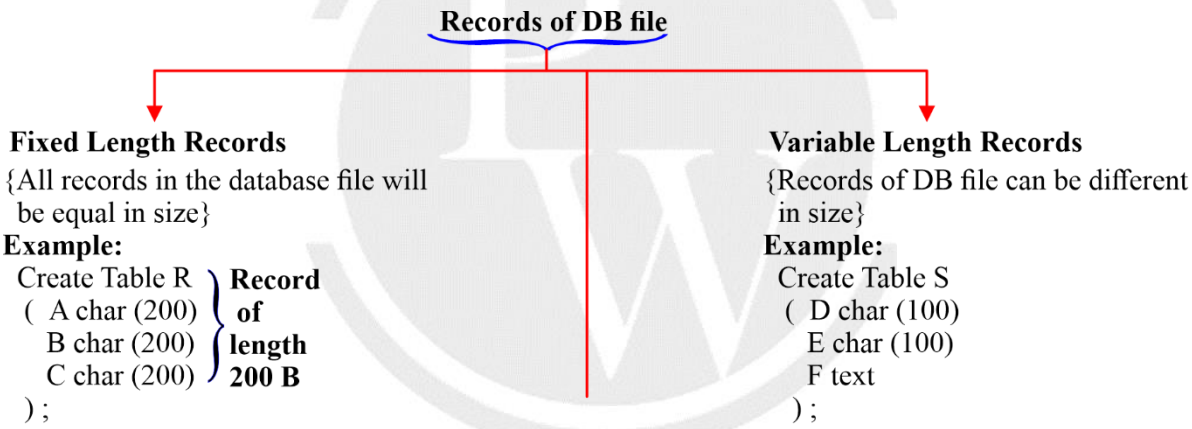
❑❑❑

4 File Organization and Indexing [B and B+ Tree]

4.1 File Organization

- Database is collection of files [Tables].
- File is collection of blocks [Pages].
- Block is collection of records.

Note:
Data access from disk (DB file) to main memory block by block.



4.2 Records organization of DB file

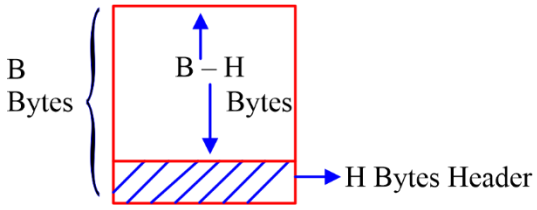
Spanned Organization	Unspanned Organization
1. Record allowed to span in more than one block.	1. Complete record must be in one block.
2. **Advantage:** Possible to allocate file without any internal fragmentation.	2. **Advantage:** less access cost and easy to organize DB file.
3. **Disadvantage:** More access cost to access spanned records.	3. **Disadvantage:** May not possible to allocate DB file without any internal fragmentation.

Note:
1. Spanned organization prefer to store database record of the file with variable length record.

2. Unspanned organization prefer to store DB record of file with fixed length record.

4.3 Block factor [BF]

The maximum possible records which can store in block (maximum record per block).



Consider record size = R bytes

1. $\therefore$ Block Factor $= \left\lfloor \frac{\text{Block size} - \text{Header size}}{\text{Record size}} \right\rfloor$

   For unspanned organization

2. $\therefore$ Block Factor $= \left\lceil \frac{\text{Block size} - \text{Header size}}{\text{Record size}} \right\rceil$

   For spanned organization

4.4 Categories of Index

4.4.1 Dense Index

For each DB record of DB file there must be index entry in index file.

4.4.2 Sparse Index

For set of DB records (blocks) of DB file there exists index entry in index file.

Important Point I

- Sparse index possible only over ordered field of DB file.
- In Sparse indexing

$$\left\lceil \begin{matrix}\text{No. of index} \\ \text{file entries}\end{matrix} \right\rceil < \left\lceil \begin{matrix}\text{No. of records} \\ \text{of DB file}\end{matrix} \right\rceil$$

4.4.3 Block Factor of Index

Assume Block size = B Bytes
Header size = H Bytes
Search key = K Bytes
Pointer size = P Bytes

$\therefore$ Block Factor of index $= \left\lfloor \frac{B-H}{K+P} \right\rfloor$

Note:
By default header size will be 0 bytes, if not given in problem.

4.5 I/O Cost [Access cost]

The number of secondary memory block (DB file block) required to transfer from disk to main memory in order to access required record.

1. I/O cost to access data record without index from DB file of n blocks

(a) Over ordered field:
Access cost = $[\log_2 n]$ block

(b) Over unordered field:
Access cost = n blocks

2. If Dense Index Used:

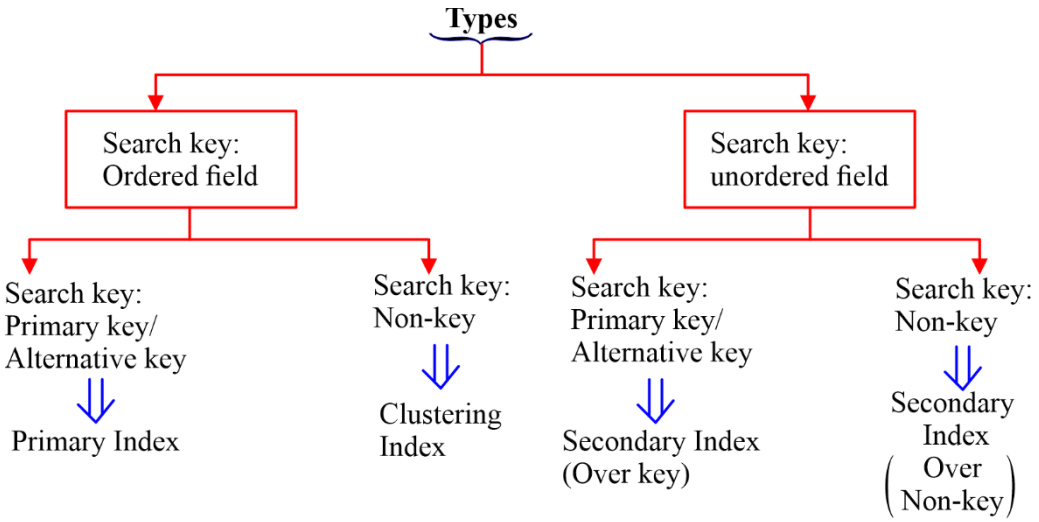
- Number of Dense Index block = $\left\lceil \frac{\text{number of records}}{\text{Block Factor of Index}} \right\rceil$
- Access cost (Number of Block access)
  $= \left\lceil \log_2 \left(\text{number of Dense index blocks at } 1^{st} \text{ level} \right) \right\rceil + 1$

3. If sparse Index Used:

- Number of DB blocks = $\left\lfloor \frac{\text{number of records}}{\text{BF of DB}} \right\rfloor$
- Number of sparse Index block (At 1^{st} level)
  $= \left\lceil \frac{\text{number of DB blocks}}{\text{BF of Index}} \right\rceil$
- Access cost (Number of block access)
  $\left\lceil \log_2 \left(\text{number of sparse index blocks at } 1^{st} \text{ level} \right) \right\rceil + 1$

4.6 Types of Index

- Search key: Fields of DB file which is used for indexing



4.6.1 Primary Index

Search Key: Ordered field and key (candidate key)

- I/O cost to access record using primary index with multilevel indexing is (k + 1) blocks.
  Where k is level of indexing.
- For any database relation at most one primary index is possible because, search key must be ordered field.
- Primary index can be either dense or sparse but sparse primary index preferred.

4.6.2 Clustered Index

Search Key: Ordered field and non-key.

- I/O cost:

  $$\text{Single level index blocks} : \left\lceil \frac{\text{No. of distinct values over non-key}}{\text{BF of Index}} \right\rceil$$

  $$\therefore \text{ Access cost} = \left\lceil \log_2(\text{single level index blocks}) \right\rceil + 1$$

- I/O cost to access cluster of record using clustered index with multilevel index is:

  $$k + \underbrace{(\text{one or more block of DB})}_{\text{Until next cluster begins}}$$

- For any DB relation at most one clustering index is possible because search key is ordered field.

> **Note:**
> For any DB relation either primary index or clustering index is possible but not both simultaneously.

4.6.3 Secondary Index [over key]

Search Key : Unordered field and key.

- Secondary index is alternative index to access data records even primary or clustering index already exists.
- Secondary index over key is always dense index.
- I/O cost to access record using secondary index over key with multilevel index: (k + 1) blocks.

4.6.4 Secondary Index [over non-key]

Search Key: Unordered field and non-key.

- I/O cost to access record using secondary index over non-key with multilevel index = [k blocks from k level of index] + [some more DB blocks]

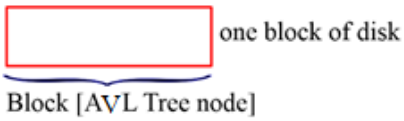
> **Note:**
> Many secondary index's possible for DB file.

4.7 Balanced Search Tree [Height Restricted Search Tree]

- The maximum height of search tree for n distinct keys should not exceed O (log n)
- Worst case search cost to search element is : O (log n)

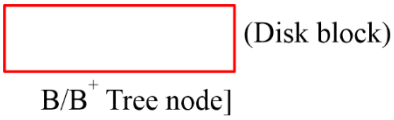
4.7.1 Why B/B+ Tree Index preferred rather than balanced BST/AVL for DB index

(a) AVL: If AVL tree (balanced BST) used for DB index:



- Disadvantage: Each index block with only one key so the number of levels of index is high. (I/O cost to access data is also very high).
- Wastage of disc space, which is allocated for index (only one key stored per block)

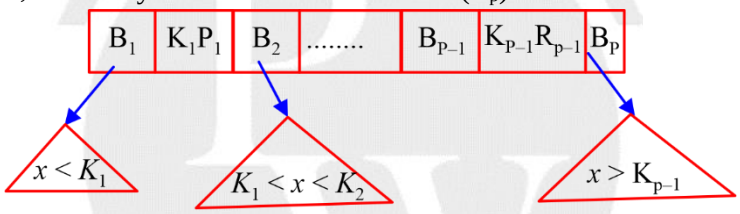
(b) B/B$^+$ Tree:



- The access cost (I/O cost) is less because many keys store in one node (number of levels of index will be less).
- The wastage of disk space is less.

4.7.2 B Tree Definition

- Order P (degree): The maximum possible child printers can be stored in B Tree node.
  Node structure: P child pointer, P – 1 Key and P – 1 Record Pointer (R_p).



$\therefore$ order of node $\Rightarrow$

$P \times$ (size of block pointer) + (P - 1) (size of key field + size of record pointer) $\leq$ Block size

1. Every internal node except root must be atleast $\left\lceil \frac{p}{2} \right\rceil$ block pointer and $\left\lceil \frac{p}{2} \right\rceil$ - 1 keys and at most P block pointer and (P – 1) keys.
2. Root can have at least 2 child/block pointer and 1 keys and at most P block pointer and (p - 1) keys.
3. Every leaf node must be at same level.

- **Advantages:**

B Tree index best suitable for random access of any one record.

Example:

```
SELECT *
FROM R
WHERE A = 15;
```

$\therefore$ Worst case access cost = (k + 1) blocks
Best case access cost = 2 blocks

- **Disadvantages:**

B Tree index not suitable for sequential access of range of records.

4.7.3 B+ Tree

Order P: The maximum possible pointers can be store in B^+ tree node.

1. Node Structure:

- **Leaf node:** (set of (key, R_P) pairs and one block pointer)

K_1R_1	K_2R_2	…….	$K_{P-1}R_{P-1}$	B_P

→Block Pointer {pointer to next leaf}

∴ Order of leaf node = $(P-1)[K + R_P] + 1 * B_P \leq$ Block size.

- **Internal Node:**
  Order of internal node = $P * B_P + (P-1) * K \leq$ Block size.
- B^+ Tree best suitable for random access of any one record and sequential access of range of records.
- I/O cost to access range of 'x' records sequential:

$$K + \frac{x}{\left\lceil \frac{P}{2} \right\rceil - 1} + x = 0(K + x)$$

To reach leaf

No. of times leaf accessed

DB block

Important point 2

If disk block allocation for B and B^+ tree nodes are equal size then

1. Order P of B tree < order P of B^+ Tree.
2. [number of index blocks of B tree index for n keys] ≥ [number of index blocks of B^+ Tree nodes for n keys]
3. I/O cost ≥ I/O cost

Important point 3

If order P of B Tree is equal to order P of B^+ Tree then

1. [number of B Tree node n distinct keys] ≤ [number of B^+ Tree nodes for n distinct keys]
2. [B Tree level for n keys] ≤ [B^+ Tree levels for n keys]

❑❑❑

5 TRANSACTIONS AND CONCURRENCY CONTROL

5.1 Transaction

A set of logically related operation to perform unit of work.

5.2 Degree of concurrency

The number of users (Transactions) using data bases simultaneously (concurrently).

5.2.1 Flat File System

In flat file system 1 file is a resource so, only one user allowed at a time.

5.2.2 DBMS File System

- In DBMS file system 1 record is a resource so, it allows as many users as the number of records.
- More degree of concurrency.

5.3 Main Operations in Transactions

5.3.1 Read (A)

Access the data item ‘A’ from DB file ‘Disk’ to programmed variable (main memory) in order to use current value of ‘A’ in transaction logic.

5.3.2 Write (A)

Modification of data item ‘A’ in DB file.

5.4 ACID Properties

To preserve integrity (correctness) each transaction must satisfy ACID properties

DBMS Software	A:Atomicity, D:Durability} Recovery management component of DBMS software take cares of atomicity and durability
	I : Isolation} Concurrency control component of DBMS software is responsible for isolation.
DBA User	C : Consistency} User (DBA or DB developer) is responsible for consistency.

5.5 Atomicity

Execute all operations of transaction including commit or execute none of the operation of transaction by the time of transaction termination.
Recovery management component should roll back, if transaction fails anywhere before commit.

5.5.1 Redo Operation

- It performs all the modification of database file because of transaction commit.
- Clean all the log entries of committed transaction.
- Redo is not performed after every commit but after every check point.

5.5.2 Undo Operation

- Undone all the write operation performed by the transaction.
- Convert dirty block (updated block) into clean block.

5.5.3 Check point

- Check point issued by DBMS software in regular interval.
- If checkpoint issued
  (I) Performs Redo operation on all the committed transaction until previous checkpoint.

Example :

9 : 00 AM	DB Started
:	
9 : 05 AM	1st checkpoint
:	
9 : 10 AM	2nd checkpoint
:	
9 : 15 AM	System Crash

5.5.4 System Crash

If System crash/failure happen, required operation to recover are
(I) All committed transaction until previous checkpoint will perform Redo.
(II) All uncommitted transaction in entire system will perform undo.
(III) Clean all log entries.

5.5.5 Roll back (Abort)

Undo modification of database file which are done by failure transaction.

5.6 Durability

- Durability maintained by Recovery management component of DBMS Software.
- The transaction should able to recover under any case of failure.
- Transaction fails because of

1. Power failure
2. Software crash
   - OS restarted
   - DBMS restarted
3. OS/DBMS concurrency controller may kill transaction.
4. Hardware Crash (Disk failure)
   RAID architecture of disk design is used to overcome the problem.

5.7 Consistency

- User responsible for consistency.
- Database should be consistent before and after the execution of the transaction.
- DB operations requested by user (SQL queries/transaction operation) must be logically correct.

5.8 Isolation

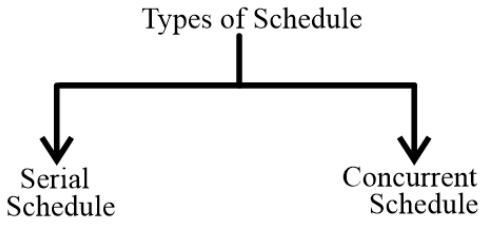
- Isolation maintained by concurrency control component.
- Isolation means concurrent execution of 2 or more transaction result must be equal to result of some serial schedule.

5.9 Schedules

Time order execution sequence of two or more transactions.

Example :

S : $R_2(A)$ $R_1(A)$ $W_3(A)$



5.9.1 Serial Schedule

- After Commit of one transaction, begins (Start) another transaction.
- Number of possible serial Schedules with 'n' transactions is "n!"
- The execution sequence of Serial Schedule always generates consistent result.

Example

S : $R_1(A)$ $W_1(A)$ Commit (T_1) $R_2(A)W_2(A)$ commit (T_2).

5.9.2 Advantage

- Serial Schedule always produce correct result (integrity guaranteed) as no resource sharing.

5.9.3 Disadvantage

- Less degree of concurrency.
- Through put of system is low.
- It allows transactions to execute one after another.

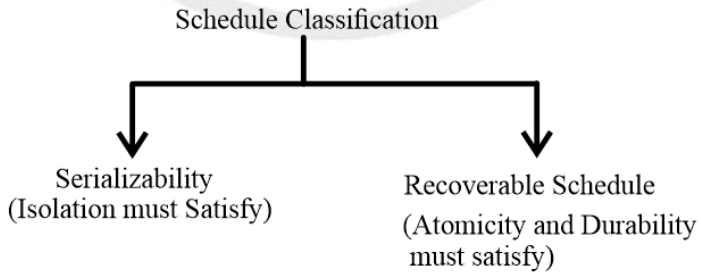
5.10 Concurrent Schedule

Transactions can execute concurrently or simultaneously

- May result inconsistency.
- Better through put and less response time.
- To maintain consistency transaction should satisfy ACID property.
- If T_1 and T_2 transaction with 'n' and 'm' operations each, then

  No. of concurrent Schedule = $\frac{(n+m)!}{n!\ m!}$

5.11 Schedule Classification

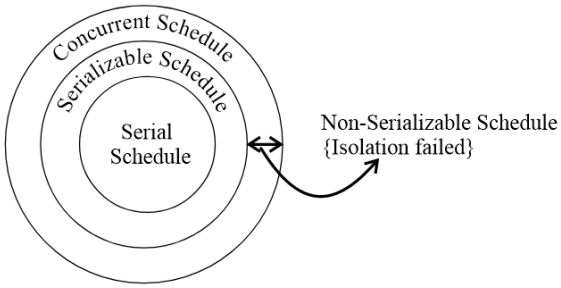


5.12 Serializable Schedule

A Schedule is serializable Schedule if it is equivalent to a Serial Schedule.

(i) Conflict Serializability

(ii) View Serializability



5.13 Conflict Serializability

5.13.1 Topological order

Graph traversal algorithm for directed graph

(i) Visit vertex (v), whose indegree is 'O' and delete 'V' from the graph.

(ii) Repeate (i) for all the vertices of graph.

5.13.2 Conflict Pairs

Two operations form Schedule (s) are conflict pair if and only if

(i) Atleast one write operation.

(ii) Operation on different data item.

(iii) Operations are from different transactions.

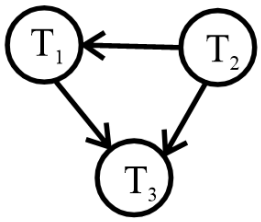
$$\left.\begin{array}{l} S: r_i(A).....w_j(A) \\ S: w_i(A).....r_j(A) \\ S: w_i(A).....w_j(A) \end{array}\right\} \text{Conflict Pairs}$$

5.13.3 Precedence Graph

A graph G in which vertex (v) represent the transaction of Schedule and edges (E) represent conflict pair precedence's.

Example:

S : $R_2(A)$ $R_2(B)$ $W_1(A)$ $W_2(B)$ $W_3(A)$ $W_3(B)$



5.13.4 Conflict Equal Schedule

Schedule S_1 and S_2 are conflict equal Schedule if and only if Schedule S_2 can derived by inter changing consecutive non-conflict pairs of Schedule S_1.

$$S_1 \xleftarrow[\text{Consecutive non-conflict pair}]{\text{Interchange}} S_2$$

Note:

If Schedule S_1 and S_2 have

(i) Same set of transactions, and

(ii) Same precedence graph and

(iii) A cyclic precedence graph then

S_1 and S_2 are conflict equal Schedule.

Important Point 1:

1. If S_1, S_2 Schedule are conflict equal then precedence graph of S_1 and S_2 must be same.
2. If S_1 and S_2 have same precedence graph then S_1 and S_2 may or may not conflict equal.

5.14 Conflict Serializable Schedule

Schedule (s) is conflict serializable Schedule if and only if some serial Schedule (S) must be conflict equal to Schedule (S).

$$\underbrace{\text{Schedule (s)}}_{\text{Conflict Serializable Scchedule}} \xleftrightarrow[\text{Equal}]{\text{Conflict}} (S') \text{Serial Schedule}$$

Important Point 2:

Schedule (s) is conflict serializable Schedule if and only if

1. Precedence graph is a cyclic and
2. Conflict equal serial schedule are to apological order of precedence graph.

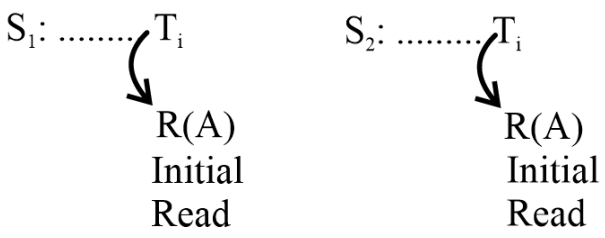
Note:

[No. of serial schedule conflict equal to schedule s] = [No. of topological order of schedules precedence graph]

5.15 View Serializable Schedule

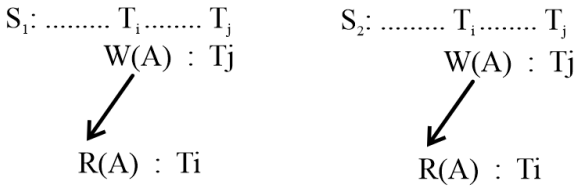
- Schedule (s) is view serializable if and only if some serial schedule (s') view equal to schedule (s).
- Let S_1 and S_2 be two schedule with the same set of transactions, S_1 and S_2 are view equal view equivalent if the following three conditions are met:

5.15.1 Initial Read



For each data item A, if transaction T_i reads the initial value of A in Schedule S_1, then transaction T_i must, in Schedule S_2, also read the initial value of A.

5.15.2 Updated Read



For each data item A if transaction T_i perform Read (A) in Schedule S_1 and that value was produced by transaction T_j, then transaction Ti must in Schedule S_2 also read the value of A that is produce by the transaction Tj.

5.15.3 Final Write

S_1 : ..... Ti .....

Final write of A : $w(A)$ (Last write)

S_2 : ..... Ti....

Final write of A : $w(A)$ (Last write)

For each data item A, the transaction that perform the final write (A) operation in schedule (S_1) must perform the final write (A) operation in schedule S_2.

Important Point 3

- If schedule S is conflict serializable schedule, then S is also view serializable schedule.
- If schedule s is not conflict serializable then it may or may not view serializable
- Every view serializable schedule that is not conflict serializable.

5.16 Classification based on Recoverability

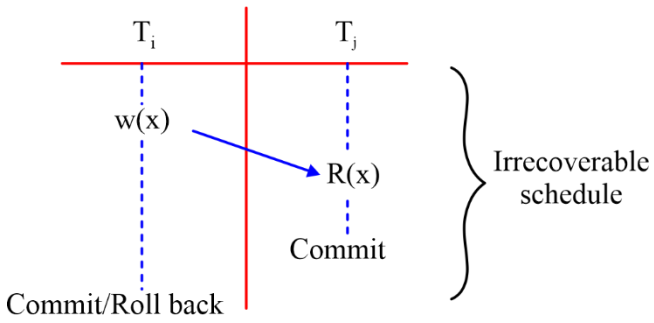
The concurrent execution may leads:

1. Irrecoverable schedule
2. Cascading rollback problem
3. Lost update problem

The above problems can occur even if the schedule is serializable (Integrity satisfied)

5.16.1 Irrecoverable Schedule

- A schedule(s) is irrecoverable if and only if transaction T_j reads data item 'x', which is updated by T_i and commit of T_j before commit/rollback of transaction T_i.
- Irrecoverable means unable to recover or rollback.



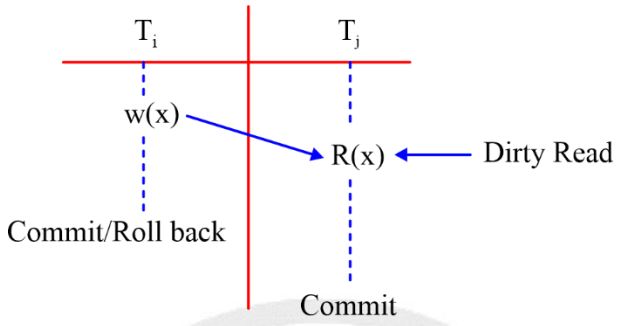
5.16.2 Recoverable Schedule

A schedule(s) is recoverable if and only if

(I) No uncommitted read (no dirty ready in schedule(s).

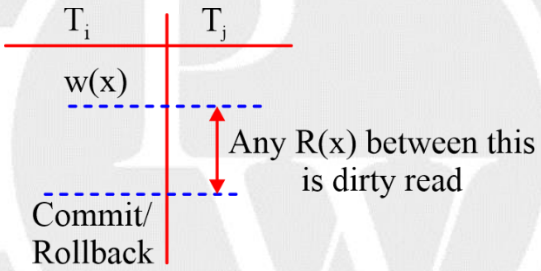
Or

(II) If transaction T_j reads data item 'x' which is updated by transaction 'T_i', then commit of T_j must be delayed until commit/rollback of T_i.



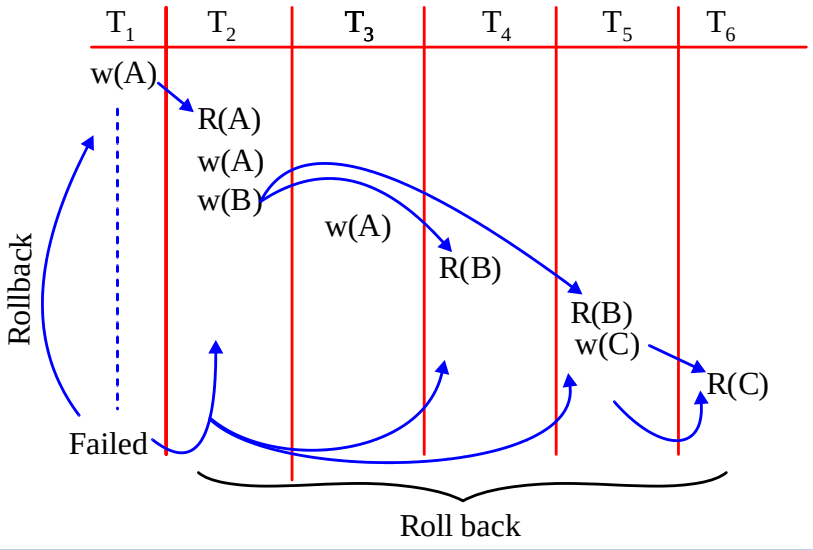
5.16.3 Uncommitted Read (Dirty Read)

Transaction T_j reads data item 'x' which is updated by uncommitted transaction. T_i



5.17 Cascading Rollback Schedule (Problem)

When some transaction reads data items which is updated by some other transaction then because of failure of the transaction that updated the data item may rollback all the dependent transaction.

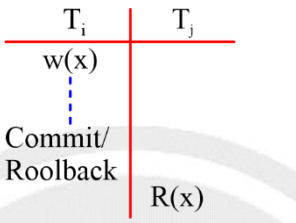


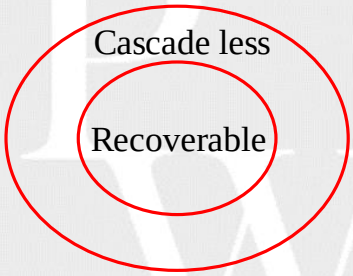
Failure of T_1 forced to rollback T_2, T_4, T_5 and T_6.

- **Disadvantage of Cascading Rollback**
  1. It waste the CPU execution time.
  2. Wastage of I/O access cost.

5.17.1 Cascadeless Rollback Recoverable Schedule

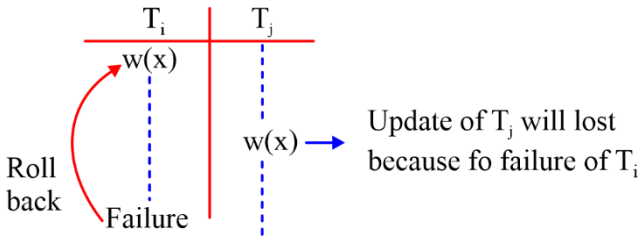
- No dirty read in the schedule.
- Transaction T_j should delay read (x) which is updated by T_i, until T_i commit or rollback.





5.18 Lost Update Problem

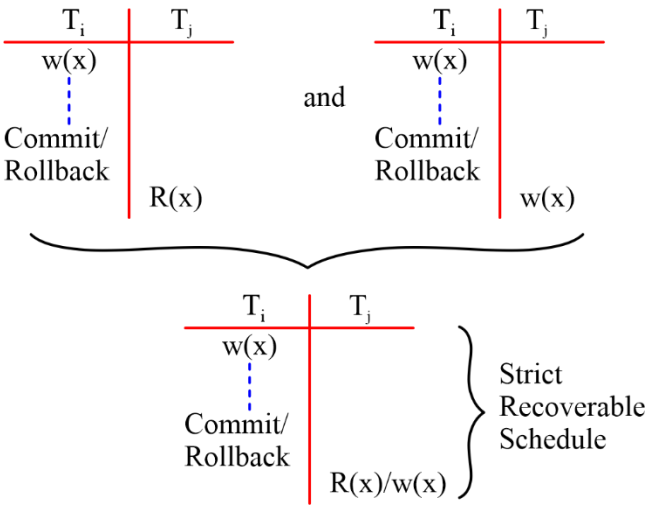
Lost update problem occur if T_j write (x) which is already written by uncommitted transaction T_i.

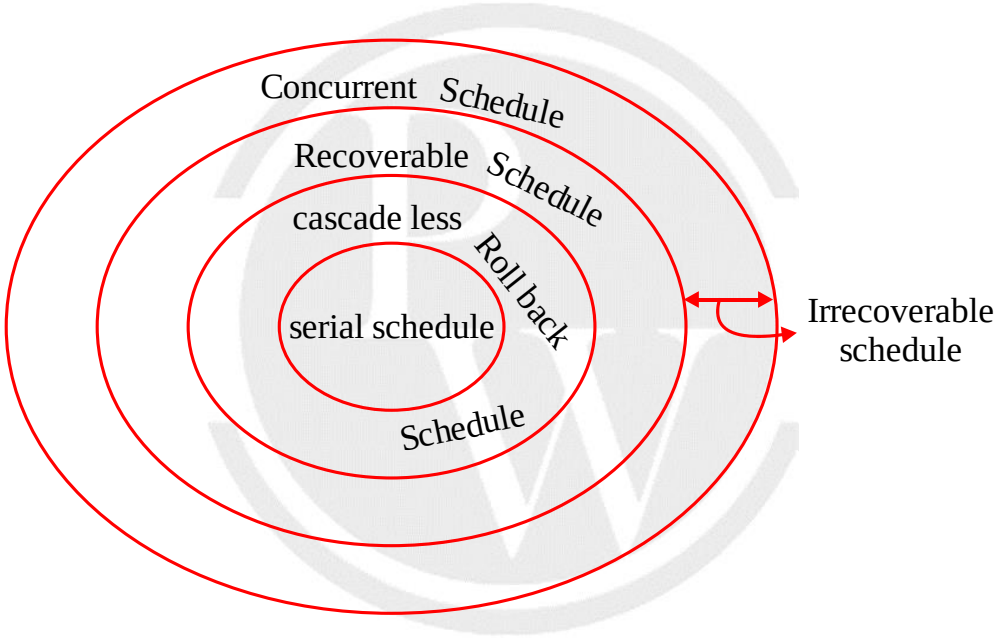


5.19 Strict Recoverable schedule

A schedule must satisfy

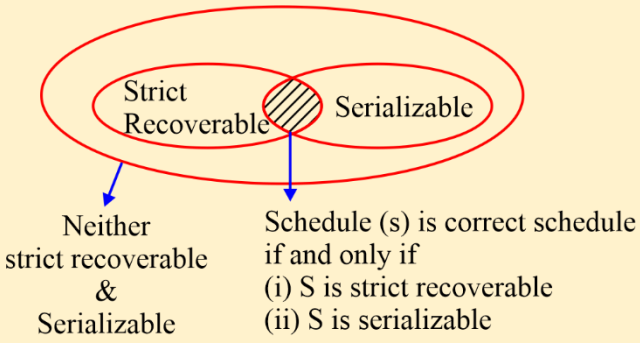
1. Cascadeless rollaback recoverable schedule and
2. If T_i writes (x) then other transaction T_j write (x) must be delayed until T_i commit or rollback.





Note:

1. Strict recoverable schedule may or may not be serializable.
2. Serializable schedule may or may not be recoverable.



5.20 Concurrency control protocol

Concurrency control protocol should not allow to execute:

1. Non-serializable schedule (Violate Isolation).
2. Non-strict recoverable schedule (Violate atomicity and Durability).

5.21 Locking Protocol

Lock is as variable used to identify the status of data item.

Transaction (T_1)

Lock (A): Granted by concurrency controller

R(A)

W(A)

Lock(B): Denied by concurrency controller

5.22 Types of Lock

1. Shared Lock(s): Read only lock.
2. Exclusive lock (x): Read/write lock.

1. Shared (s mode)

- Exists when concurrent transaction granted READ access.
- Produce no conflict for Read only transactions.
- Issued when transaction wants to read and exclusive lock not held on item.

2. Exclusive (x) mode

- Exists when access reserved for locking transaction.
- Used when potential for conflict exists.
- Issued when transaction wants to updated unlocked data.

Example:

Transaction (T_1)

S(A) : Granted shared lock

R(A) } only read allowed

X(B) : Granted exclusive lock

R(B) }
W(B) } Read and write

5.22.1 Lock compatible table

Data item A	S	X ← Holded by T_i
Requested by T_j — S	Yes	No
Requested by T_j — X	No	No

5.23 Phase Locking Protocol (2PL)

- It guaranteed serializability.
- Transaction (T) allowed to request lock on any data item in any mode (x/s) until first unlock of transaction (T).

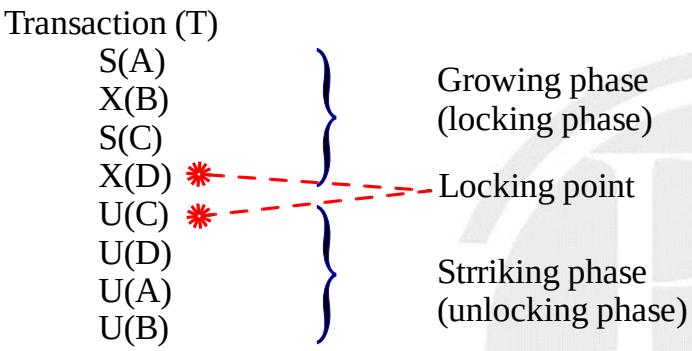
1. Growing Phase

Acquires all the required locks without unlocking any data. Once all locks have been acquired, the transaction is in it locked point.

2. Shrinking phase

Release all locks and can not obtain any new lock governing rules of 2 PL.

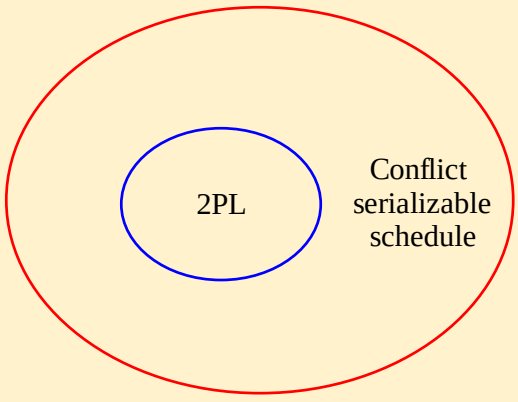
Example:



Important point 1

1. If schedule (s) is executed by 2PL then schedule guaranteed is conflict serializable schedule.
2. Conflict equal schedule is based on lock points order of the transaction.
3. If schedule is not conflict serializable schedule (cyclic precedence graph) then schedule not allowed to execute by 2 PL.

Note: Every schedule which is allowed by 2PL is always conflict serializable, but not every conflict serializable schedule is allowed by 2PL.



5.23.1 Limitations of 2PL

1. 2PL restriction may leads to deadlock.
2. 2PL restriction may leads to starvation.
3. 2PL condition not sufficient to avoid
   (I) Ir-recoverable schedules
   (II) Cascading rollback problem
   (III) lost updated problem

5.24 Strict 2PL protocol

- **Basic 2PL:** Lock request of transaction (T) not allowed in shrinking phase of transaction T

  and

  Strict recoverable: All exclusive lock of transaction must hold until commit/Rollback of transaction T.

5.24.1 Strict 2PL protocol guarantees

(a) Serializability.
(b) Strict recoverable.

Disadvantage:

It is not free form deadlock and starvation.

Example:

Transaction (T)

$$\left.\begin{array}{l} S(A) \\ X(B) \\ S(C) \\ X(D) \end{array}\right\} \text{growing phase}$$

$$\left.\begin{array}{l} U(A) \\ U(B) \\ \text{Commit} \\ U(B) \\ U(D) \end{array}\right\} \text{Strinking phase}$$

5.25 Rigorous 2PL

Basic 2 PL : A lock request allowed only in growing phase of transaction and

all locks (shared/Exclusive) of transaction T must be hold until commit or Rollback.

5.26 Conservative 2PL

Transaction (T) must lock all required data itmes in starting phase.

If locks are not available then unlock all data items and re-request all data item lock again.

	Basic 2PL	Strict 2PL	Rigorous 2PL	Conservative 2PL
Guaranteed Serializability	Yes	Yes	Yes	Yes
Guaranteed strict Recoverable	No	Yes	Yes	No
Free from dead lock	No	No	No	Yes
Free from starvation	No	No	No	NO

5.27 Time stamp ordering protocol

Assign global unique time stamp value to each transaction and produces order for transaction submission.

The time stamp values of transaction can be used for transaction identification and to set priority between the transaction.

5.27.1 Time stamp value for each data item

- Assume data item is A

I. **Read TS value (A):** It is the highest transaction time stamp value that has executed R(A) successfully.

II. **Write TS value (A):** It is the highest transaction TS value that has executed W(A) successfully.

1. Basic Time stamp ordering protocol:

a. If transaction T issues R(A) operation.

If (WTS(A)) > TS(T)

then Rollback T

else

{Allow to execute R(A) successfully

set RTS (A) = max {RTS(A), TS(T)}}

b. If transaction T issues W(A) operation
if (RTS(A)) > TS (T)
{then Rollback trans (T)}
else if (WTS(A) > TS(T))
{then Roll back T}
Else
{Allow to execute W(A) successfully
set WTS(A) = {TS(T)}}

2. Thomas write rule stamp ordering protocol

a. If transaction T issues R(A operation
if (WTS(A)) > TS(T)

{then Rollback transaction T}

else

{allow to execute R(A) successfully

set RTS (A) = max {TS(T), RTS (A)}

b. If transaction T issue W(A) operation
if (RTS (A) > TS(T))

{then roll back T}

else if (WTS(A)) > TS(T)

{then ignore w(A) of transaction T and continue.}

else

{Allow to execute w(A) successfully

set WTS(A) = TS(T)}

3. Strick time stamp ordering protocol

A transition T_2 that issues a R(A) or W(A) such that TS (T_2) > WTS (A) has it read operation delayed until the transaction T_1 that write the value x has committed or rolled back.

Basic TS ordering protocol	Thomas write TS ordering protocol	Strict TS ordering protocol
1. Ensue conflict serialization	1. Ensure view serialization	1. Ensure conflict serialization
2. Free from deadlock	2. Free from deadlock	2. Free from deadlock
3. Not free from starvation	3. Not free from starvation	3. Not free from starvation
4. Not guaranteed strict recoverable	4. Not guaranteed strict recoverable	4. Guaranteed strict recoverable

❑❑❑

GATE Exam 2025?

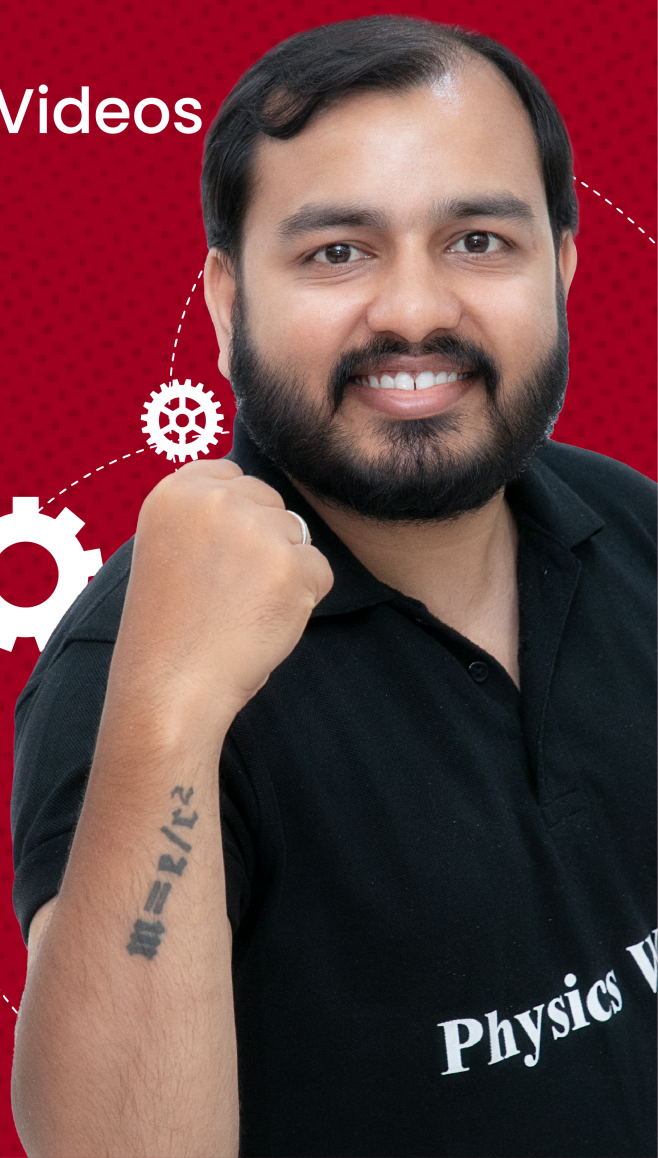
SHRESHTH BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in ***Hindi***

Weekday & Weekend Batches Available

JOIN NOW!



Computer Networks

Computer Networks

INDEX

1 IP ADDRESSING, SUBNETTING & SUPERNETTING

1.1 IP Addressing

Class A : 0 → (1 - 126), No. of IP Addresses = 2^{31}
Class B : 10 → (128 - 191), No. of IP Addresses = 2^{30}
Class C : 110 → (192 - 223), No. of IP Addresses = 2^{29}
Class D : 1110 → (224 - 239), No. of IP Addresses = 2^{28}
Class E : 1111 → (240 - 255), No. of IP Addresses = 2^{28}

1.2 Default subnet Mask

Class A : 255.0.0.0
Class B : 255.255.0.0
Class C : 255.255.255.0

1.3 Private Addresses Range

10.0.0.0 to 10.255.255.255 → 1 class A Network.
172.16.0.0 to 172.31.255.255 → 16 class B Network.
192.168.0.0 to 192.168.255.255 → 256 class C Network.

Class	Number of Networks	Number of hosts per Network
Class A	$2^7 - 2 = 126$	$2^{24} - 2 = 1,67,77,214$ hosts
Class B	$2^{14} = 16,384$	$2^{16} - 2 = 65,534$ hosts
Class C	$2^{21} = 20,97,125$	$2^8 - 2 = 254$ hosts
Class D	No NID and HID, all 28 remaining bits are used to define multicast address	
Class E	No NID and HID, it is meant for research and future purpose	

Note:
The IP address 127.x.y.z is known as loop back address and it is used to check the connectivity.

1.4 Types of Communication

(i) Unicast communication (1 : 1)

(ii) Broadcast communication (1 : All)

(iii) Multicast Communication (1: Many)

1.5 Unicast Communication

1. Transmitting the data from one computer to another computer is called as unicast communication.
2. It is one to one transmission.
3. In Unicast communication both source and destination either present in the same network or in the different network.

1.6 Broadcast Communication

1.6.1 Limited Broadcasting

1. Transmitting data from one computer to all other computer in the same network is called as Limited Broadcasting.
2. Limited Broadcast Address = 255.255.255.255
3. Limited broadcast address can't be used as a source IP Address.
4. Limited broadcast Address will always be used as a Destination IP.

1.6.2 Direct Broadcasting

1. Transmitting data from one computer to all other computer in the different network is called as Direct Broadcasting.
2. Direct broadcast address can't be used as a source IP Address.
3. Direct broadcast Address will always be used as a Destination IP.

	NID	HID		
1.	–	0's	→	Network ID
2.	–	1's	→	Direct Broadcast Address (DBA)
3.	1's	1's	→	Limited Broadcast Address (LBA)
4.	0's	–	→	Host with in the Network
5.	1's	0's	→	Network Mask or Subnet Mask

1.7 Multicast communication

Transmitting a packet from one computer to many computers (0 or more) is called Multicast communication.

1.8 CIDR Rules

1. All the IP Address in the Block must be contiguous.
2. Block size must be a power of 2.
3. First IP address of the block must be divisible by size of the block.

1.9 Supernetting

The process of combining two or more network to get a single network is called as supernetting.

1.10 Advantage of Supernetting

1. Super netting Reduce Routing table entry.
2. Router will take less time for processing the packet.
3. It improve flexibility of IP Address Allotment i.e. If someone required 500 Address, then no need to purchase class B network we can combine two class C network.

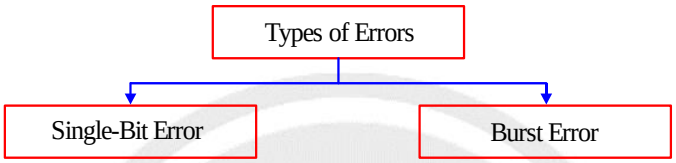
1.11 Rules of Supernetting

1. Network ID must be contiguous.
2. Size of the Network must be same and number of Network must be a power of 2.
3. First Network ID must be divisible by size of the supernet.

❑❑❑

2 ERROR CONTROL

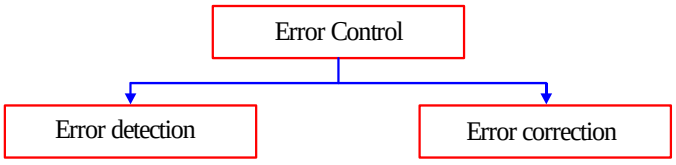
2.1 Error Control



Note:

- The number of corrupted bits or affected bits depends on the data rate and duration of noise.
- The number of corrupted bits or affected bits =Data rate * Noise duration.
- Burst error is more likely to occur than a single bit error.
- Error correction is more difficult than error detection.

	Error Detection	Error Correction
1.	Once noticed error simply discard.	Capability of correcting error.
2.	Ask for retransmission.	Does not required retransmission.



1.	Simple Parity	1.	Hamming code
2.	2D parity		
3.	Check sum		
4.	CRC		

2.1.1 Hamming distance

Hamming distance between two Binary string of same size is the number of differences between corresponding bits.
Hamming distance between two Binary string is denoted by d(x, y)

d(000, 011) = 2

d(100, 011)=3
d(10101, 11110)=3

Hamming distance can easily be found if we apply XOR operation (⊕) on the two words and count the Number of 1's in the result.

2.1.2 Minimum Hamming distance

In a set of codewords, the minimum Hamming distance is the smallest Hamming distance between all possible pairs of code words.

Valid code word			
0	1	0	(a)
1	0	1	(b)
1	1	0	(c)
0	0	1	(d)

d (a, b) = 3
d (a, c) = 1
d (a, d) = 2
d (b, c) = 2
d (b, d) = 1
d (c, d) = 3

} Minimum Hamming distance = 1

2.1.3 Minimum Hamming distance for Error detection

To detect 'd' bit error minimum hamming distance required =d+1

2.1.4 Min. Hamming distance For Error Correction

To Correct 'd' bit error minimum hamming distance required =2d+1

2.2 Simple Parity Check Code

Simple parity

In the Simple parity concept one extra bit (parity bit) is added to each dataword.
Simple parity check can detect all single bit error .
Simple parity check can not detect an even number of errors.
Simple parity check can detect an odd number of errors .

2.3 2D Parity Check Code

2D parity

Two dimensional parity check can detect and correct all single bit error and detect two or three bit error that occur any where in the matrix.
However only some pattern with four or more Errors can be detected.
In a 2D-parity check code, the information bits are organized in a matrix consisting of rows and columns.
For each row and each column one parity check bits is calculated.

2.4 CRC

Length of the dataword = n
Length of the divisor = k
Append (k-1) Zero's to the original message
Perform modulo 2 division
Remainder of division = CRC
Codeword = dataword with Appended (k-1) Zero's + CRC

Note:

1. CRC must be (k-1) bits.
2. If the generator has more than one term and coefficient of x^0 is 1, all single bit error can be detected.
3. If a generator can't divide $x^t + 1$ (t between 0 and n – 1) then all isolated Double error can be detected.
4. The generator that contains a Factor of x + 1 can detect all odd numbered errors.

2.4.1 A good polynomial generator needs to have the following characteristics

1. It should have at least two terms.
2. The coefficient of the term x^0 should be 1.
3. It should not divide $x^t + 1$, for t between 2 and n – 1.
4. It should have the factor x + 1.

2.5 Hamming Code

1. Hamming code is used for error correction.
2. Hamming code can correct 1 bit error only.
3. Hamming code can detect upto 2 bit error.

   m = Massege bits

   r = redundant bits or Check bits or parity bits or extra bits

   n = m + r (n = codeword)

According to the hamming code, number of redundant bits

$$\boxed{m + r + 1 \leq 2^r}$$

where r = lower limit

❑❑❑

3 FLOW CONTROL

3.1 Delay in Computer Network

(1) Transmission Delay (T_d)
(2) Propagation Delay (P_d)
(3) Queuing Delay (Q_d)
(4) Processing Delay (P_{rd})

3.2 Transmission Delay

Amount of time taken to transfer a packet on to the outgoing link is called as Transmission delay.

$$\text{Transmission delay}\,(T_d) = \frac{\text{Length of packet}}{\text{Bandwidth}}$$

$$T_d = \frac{L}{B}$$

3.3 Propagation Delay

Amount of time taken to reach a packet from one (sender) point to another (receiver) point is called as propagation delay.

$$\text{Propagation delay}\,(P_d) = \frac{\text{distance}}{\text{velocity}}$$

$$P_d = \frac{d}{v}$$

3.4 Stop wait Protocol

3.4.1 Sender Side Rule

Rule 1 : Sender can send one data packet at a time.
Rule 2 : Sender can send the next data packet only after receiving the ACK of the previous packet.

3.4.2 Receiver Side Rule

Rule 1: Receiver will receive and consume the data packet.
Rule 2 : After consuming the data packet, Ack need to be sent.

3.5 Efficiency OR Line utilization OR Link utilization OR Sender utilization

$$\text{Efficiency} = \frac{\text{Useful time}}{\text{Total time}}$$

$$\text{Efficiency} = \frac{T_d(\text{frame})}{T_d(\text{frame}) + 2*P_d + Q_d + P_{rd} + T_d(\text{ACK})}$$

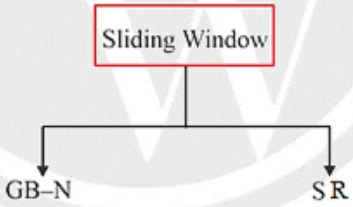
3.6 Throughput Or Effective Bandwidth Or Bandwidth Utilization Or Maximum Data Rate Possible

$$\text{Throughput} = \frac{\text{Length of the Frame}}{\text{Total time}}$$

$$\boxed{\text{Throughput} = \frac{L}{T_d(\text{frame}) + 2*P_d + Q_d + P_{rd} + T_d(\text{ACK})}} \quad \text{OR} \quad \boxed{\text{Throughput} = \eta * B}$$

3.7 Sliding Window

In the sliding window concept instead of sending one packet and wait for the acknowledgement, we send 'w' packet and wait for the Acknowledgement. Where 'w' is the sender window size.



3.7.1 GB–N(N>1)

1. In the GB – N the sender window size is N itself.
2. In the GB-N the receiver window size is equal to one always.

Note:

(1) Out of order packet is not received by Receiver.

(2) Timer is maintained only for the first frame (Rightmost) in window because if its timer expires then sender assume that rest of the frame are also not received by receiver (because out of order delivery is rejected).

Note:

GB–N uses cumulative Acknowledgement and Acknowledgement number defines the number of the next expected frame.

Ack timer < Time out timer

Window Receiver (W_R) size:

In the GB-N the window receiver size is equal to one always irrespective of window sender size (Wr=1).

Window Sender (W_S) Size:

Window sender size is calculated based on the following formula:

$Ws + W_R \leq$ Available Sequence Number

$Ws + 1 \leq$ Available Sequence Number

$Ws \leq$ Available Sequence Number – 1

3.7.2 Efficiency and Throughput in GBN

$$\text{Efficiency} = \frac{\text{Useful time}}{\text{Total time}}$$

$$\boxed{\text{Efficiency} = \frac{N*T_d(frame)}{T_d(frame)+2*P_d+Q_d+P_{rd}+T_d(ACk)}}$$

$$\text{Throughput} = \frac{N*\text{Length of Frame}}{\text{Total time}}$$

$$\boxed{\text{Throughput} = \frac{N*\text{Length of Frame}}{T_d(frame)+2*P_d+Q_d+P_{rd}+T_d(ACk)}} \quad \text{OR} \quad \boxed{\text{Throughput} = \eta * B}$$

3.8 Selective Repeat ARQ

3.8.1 Selective Repeat/Selective Reject ARQ

(1) In SR Protocol window sender size is equal to window receiver size ($W_S = W_R$).

(2) SR Protocol uses independent acknowledgement, and acknowledgement number defines number of error free packet received.

(3) SR receiver can receive out of order packet but packets are delivered to upper layer in sorted order.

(4) In SR protocol searching and sorting logic is required. Searching is done by sender and sorting is done by receiver.

(5) Timer is maintained for each and every frame in the window at sender side.

(6) For 1st out of order delivery or if packet received is corrupted then Negative acknowledgment (NACK) for respective packet is sent by receiver to sender.

(7) When sender receive NACK 3 then it will search in the window for packet 3 & immediately packet 3 is retransmitted even though its timer is not expired.

3.8.2 Efficiency and Throughput of SR

$$\text{Efficiency} = \frac{\text{Useful time}}{\text{Total time}}$$

$$\boxed{\text{Efficiency} = \frac{Ws \times T_d(frame)}{T_d(frame)+2*P_d+Q_d+P_{rd}+T_d(ACK)}}$$

$$\boxed{\text{Throughput} = \eta * B}$$

$$\text{Throughput} = \frac{W_s * \text{Length of the frame}}{\text{Total time}}$$

$$\boxed{\text{Throughput} = \frac{W_s * \text{Length of the frame}}{T_d(frame)+2*P_d+Q_d+P_{rd}+T_d(ACK)}}$$

3.9 Comparison among Stop and Wait, GBN and SR protocols

	Stop & wait	GBN	SR
Efficiency	$\eta = \frac{\text{Useful time}}{\text{Total time}}$ or $\eta = \frac{T_d(frame)}{\text{Total time}}$	$\eta = \frac{\text{Useful time}}{\text{Total time}}$ or $\eta = \frac{N*T_d(frame)}{\text{Total time}}$	$\eta = \frac{\text{Useful time}}{\text{Total time}}$ or $\eta = \frac{W_s*T_d(frame)}{\text{Total time}}$
Throughput	$\frac{\text{Length of frame}}{\text{Total time}}$ or $\eta * B$	$\frac{N*\text{Length of the frame}}{\text{Total time}}$ or $\eta * B$	$\frac{W_s*\text{Length of the frame}}{\text{Total time}}$ or $\eta * B$
Buffer	1 + 1	N + 1	N + N
Seq No.	2	N + 1	2N
Seq. No. = K bit		$W_s = 2^K - 1$ $W_R = 1$	$W_S = 2^{K-1}$ $W_R = 2^{K-1}$

$$\boxed{\text{RTT or Total Time} = T_d(frame) + 2*P_d + T_d(ACK) + P_{rd} + Q_d}$$

❑❑❑

4 IPv4 HEADER

4.1 IPv4 Header

VER (4 bits)	HL (4 bits)	Services (8 bits)	Total Length (16 bits)
Identification number (16 bits)		Flags (3 bits)	Fragment offset (13 bits)
Time to Live (8 bits)		Protocol (8 bits)	Header checksum (16 bits)
Source IP Address (32 bits)			
Destination IP Address (32 bits)			
Option (0-40 bytes)			

4.1.1 Version (4 Bit)

It is used to indicate IPv4 or IPv6.

4.1.2 Header Length(4 Bit)

Header length is a 4 bit field that contains the length of header.

Minimum Header size is 20 byte.

Maximum Header size is 60 byte.

4.2 Services [8 bit]

In this Interpretation the first 3 bit are called precedence bit (Priority bit) and Next 4 bit are called types of services bits and last bit is Not used.

4.2.1 Priority

It is a 3-bit subfield ranging from 0 to 7 (000 to 111 in binary). Priority field is needed if a router is congested and need to discard some datagram, those datagrams which have the lowest priority are discarded first.

4.2.2 Types of Services

It is a 4 bit subfield. Each bit having a special meaning, although a bit can be 0 or 1. One and only one of the bits can have the value 1 in each datagram.

4.3 Total length (16 bits)

Total length = Data + Header

4.4 Identification Number (16 bits)

1. Each datagram is associated with a sequence number is called as datagram number or identification number.
2. It is used to identify all the fragment of same datagram.
3. All the fragment of same datagram will have the same identification number.

4.5 Flags

It is the 3 bit field shown in the figure.

X	D F	M F
Not Used	Don't Fragment	More Fragment

4.6 Fragment offset (13 bits)

Fragment offset indicate no of data byte ahead of this fragment in that particular packet.

4.7 TTL (8 bits)

1. TTL is used to avoid infinite looping.
2. TTL field is used to control the maximum number of hops visited by datagram.
3. When a source host sends a datagram, it stores a number in this field. Each router that process the datagram decrements this number by one. If TTL field reaches zero before the datagram arrives at its destination, then the datagram is discarded and an ICMP message is sent back to sender.

4.8 Protocol (8 bits)

1. This 8 bits field tell us which protocol is encapsulated in the IP packet.
2. At the time of traffic, some packet must be discarded. In this case it will be advantageous to know which protocol data it contains.
3. The order in which router eliminate the datagram from buffer is-
   ICMP>IGMP>UDP>TCP
   (01) (02) (17) (06)

4.9 Header Checksum

1. It is calculated only for header part not the data because rest of the component in packet already covered by TCP checksum.
2. Header checksum is calculated at each and every Router because IP Header might be change when packet is moving from one router to another.
3. Every router makes one modification i.e. TTL so Header checksum is calculated at every Router.
4. Fragment offset, MF, Total length, option all may be changed at a Router.

4.10 Source Address (32 bits)

This 32 bit defines the IPv4 address of source. This field remain unchanged during the time the IPv4 datagram travel from the source Host to destination Host.

4.11 Destination Address (32 bits)

This 32 bits Field defines the IPv4 address of the destination. This field remain unchanged during the time the IPv4 datagram travel from source host to destination host.

4.12 Option

The Header of IPv4 data gram is made of two parts a fixed part and a variable part. The fixed part is 20 bytes long and variable part that can be maximum of 40 bytes.

There are 5 options

1. Strict source Routing
2. Loose source Routing
3. Record Routing
4. Time stamp
5. Padding

4.12.1 Strict Source Routing

A strict source routing is used by the source to predetermine a route for data gram as it travel through the internet.

4.12.2 Loose Source Routing

A loose source route option is similar to strict source route but it is less rigid. Each router in the list must visited, but the data gram can visit other router as well.

4.12.3 Record Routing

A record route option is used to record the internet routers that handle the data gram. It can list up to 9 router Address.
All the Router are supposed to record their IP Address on their IP packets.

Note:

First 16 bits (2 byte) are reserved for option type (8 bit) and length (8 bit). Out of 40 bytes only 38 bytes are remaining for storing IPv4 addresses. In 38 bytes we can store 9 IPv4 addresses as each IPv4 address is of 4 byte

4.12.4 Time Stamp

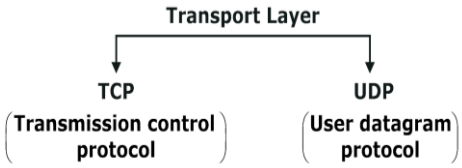
It is used to find out delays at each router. Every router should record incoming time and outgoing time.

❑❑❑

5 TCP & UDP

5.1 Introduction

Transport Layer can be connection oriented or connection less.



1. TCP is reliable process to process delivery of entire message.
2. TCP is a connection oriented.
3. TCP connection are full duplex and point to point.
4. TCP connection has 3 phases
   i. Connection establishment
   ii. Data transfer
   iii. Connection termination
5. Each TCP connection is identified uniquely by Source Port + Destination Port + Source IP + Destination IP.
6. Each TCP connection is associated with Four window.
7. TCP uses three way “Handshake” to establish TCP connection.
8. TCP is not useful for Broadcasting and Multicasting.
9. TCP Header size is 20 bytes but if option (40 bytes) is added it will become 60 bytes.
10. TCP provide end to end error control and flow control.
11. Data will be received at the destination in order.
12. Data may arrive out of order and be temporarily stored by receiving TCP, but TCP guarantee that no out of order data delivered to the process.

5.2 TCP header

<table>
<tr><td colspan="8">Source Port (16 bits)</td><td>Destination Port (16 bits)</td></tr>
<tr><td colspan="9">Sequence number (32 bits)</td></tr>
<tr><td colspan="9">Acknowledgement number (32 bits)</td></tr>
<tr><td>Header Length (4 bits)</td><td>Reserved bits (6 bits)</td><td>URB</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size (Advertisement Window) (16 bits)</td></tr>
<tr><td colspan="8">Check sum (16 bits)</td><td>Urgent Pointer (16 bits)</td></tr>
<tr><td colspan="9">Options
(0 - 40 bytes)</td></tr>
</table>

Port Number	Name
0 – 1023	Well known port No.
1024 – 49151	Registered Port No.
49152 – 65535	Dynamic Port No.

$$\text{Wrap Around time (WAT)} = \frac{\text{Total sequence No.}}{\text{Bandwidth}\left[\text{Bytes / Sec}\right]}$$

- Minimum sequence number required to Avoid wrap Around time with in Life time = B × LT
- Minimum number of bits required to Avoid wrap Around time with in LT = $\lceil \log_2 B * LT \rceil$

SYN = 1 → Consume one sequence number.

Ack = 1 → Consume zero sequence number.

FIN = 1 → Consume one sequence number.

1 Data byte → Consume one sequence number.

SYN	Ack	Meaning
1	0	request
1	1	reply
0	1	Ack
0	0	Data

Time out timer in TCP

Basic Algorithm	Jacobson's Algorithm
Time Out Timer = 2 * RTT Next Round Trip Time (NRTT) = α(IRTT) + (1 – α)ARTT $0 \le \alpha \le 1$	Time Out Timer = 4 * ID + RTT Next Round Trip Time (NRTT) = α (IRTT) + (1 – α) ARTT $0 \le \alpha \le 1$ Actual Deviation (AD) = |IRTT – ARTT| Next Deviation (ND) =α (ID) + (1 – α) AD

5.3 Congestion Control

1. $W_s = \min (W_C, W_R)$

Slow start	Congestion Avoidance	Congestion Detection
1. If ACK Arrives $W_c = W_c + 1$	1. IF ACK Arrives $W_c = W_c + \frac{1}{W_c}$	1. Time out
2. After one RTT $W_c = 2 * W_C$	2. After one RTT $W_c = W_c + 1$	2. 3 duplicate ACK

5.4 Token Bucket

- Token bucket algorithm allows ideal hosts to accumulate credit for the future in the form of tokens.
- In regular interval, tokens are thrown into the bucket.
- Bucket has a maximum capacity.
- If there is a ready packet a token is removed from bucket and packet is sent.
- If there is no token in the bucket, the packet cannot be sent.

Let the capacity of token bucket is 'C' token and token enter into the bucket at rate of 'r' tokens per second.
The maximum number of packet that can be enter into the network during the time interval 't' is

Maximum number of packet = $C + rt$

Maximum average rate for token bucket $M = \frac{C + rt}{t}$

$$Mt = C + rt$$
$$Mt - rt = C$$
$$(M - r)t = C$$
$$t = \frac{C}{M - r}$$

C → token Bucket capacity
r → Token Arrival rate

5.5 UDP

1. UDP is message oriented connection less Datagram protocol.
2. It is unreliable Transport protocol.
3. It does not provide Flow control and Error control & congestion control.
4. It does not add anything to the services except process to process delivery of data.
5. Header is simple and fixed in size i.e. 8 byte.

UDP Header

Source port (16 bit)	Destination port (16 bit)
Length (16 bit)	Checksum (16 bit)

Note:
Unlike TCP, the checksum calculation is not mandatory in UDP. No error control or flow control is provided by UDP. Hence UDP depends on IP and ICMP for error reporting.

5.5.1 Optional inclusion of checksum

The sender of UDP packet can choose not to calculate the checksum. In this case the checksum field is filled with all 0's before being sent.

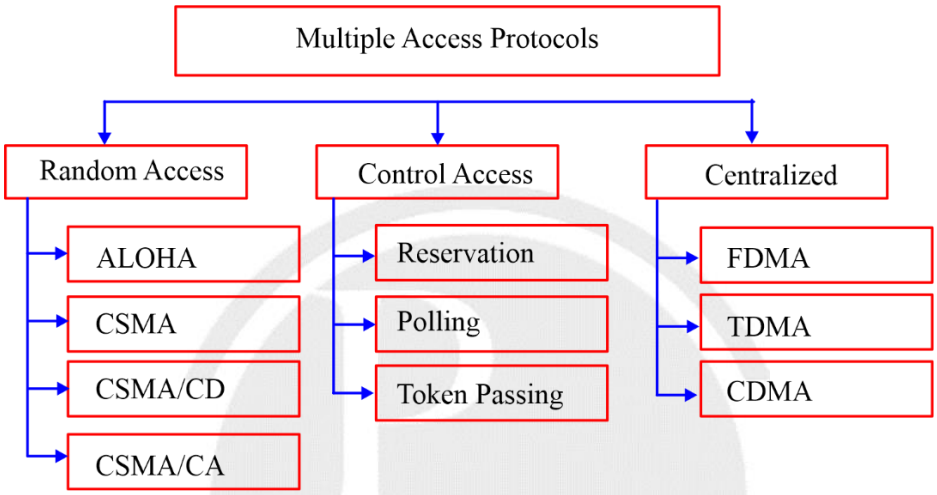
5.5.2 Need of UDP

[1] The application that required one request one reply. TCP is not suitable hence we use UDP.
[2] Application that required constant dataflow TCP is not suitable hence we use UDP.
[3] Application that required multimedia data transfer we cannot use TCP hence we use UDP.
[4] Application that required fastness and then reliability TCP is not suitable hence we use UDP.
[5] UDP used for management process such as SNMP (simple network management protocol).
[6] UDP is used for some route updating protocol such as RIP.
[7] For broadcasting & multicasting application TCP is no suitable hence we use UDP.
[8] UDP is normally used for interactive real time applications.
[9] UDP is suitable for a process with internal flow and error control mechanisms. For example, the Trivial File Transfer protocol(TFTP) process include flow and error control. It can easily use UDP

TCP	**UDP**
Connection-oriented	Connectionless
Reliability in delivery of message	Not reliable
Sequence Number.	No sequence number.
ACK number.	No ACK number.
Overhead is high(Header size 20-60 Bytes)	overhead is less(Header size = 8 Bytes)
Keep track of order (sequence)	No order
Protocols: HTTP, FTP, SMTP, POP	Protocol: DNS, SNMP, TFTP, DHCP, All real time protocol

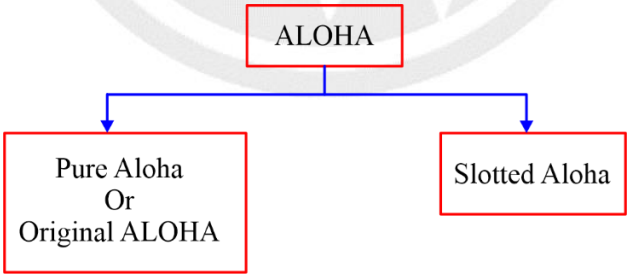
❑❑❑

6 MEDIUM ACESS CONTROL[MAC]



6.1 Aloha

Aloha was developed at university of Hawaii in 1970's.
It was designed for wireless LAN but it can be used in any shared medium.
Each station sends equal size frame.



Pure Aloha	Slotted Aloha
Any station transmits the data at any time.	Any station can transmit the data at the beginning of any time slot.
Vulnerable time in which collision may occur = 2 * T_f (T_f - Transmission time for single frame)	Vulnerable time in which collision may occur = T_f
Throughput of pure aloha = $G * e^{-2G}$	Throughput of slotted Aloha = $G * e^{-G}$
Maximum throughput s_{max} = 18.4 % (When G = 1/2)	Maximum throughput s_{max} = 36.8 % (When G = 1)
The main advantage of pure aloha is it simplicity in implementation	The main advantage of slotted aloha is that it reduces the number of collisions to Half and double the throughput of pure aloha

6.2 CSMA (Carrier Sense multiple access)

CSMA requires that each station, first sense the carrier before transmitting the data.
Vulnerable time for CSMA = Propagation time.

6.3 Persistence methods in CSSMA

Persistent
Non-persistent
P-persistent

6.4 CSMA/CD (Carrier Sense multiple access/Collision Detection)

1. Minimum size of frame to detect the collision in Ethernet (CSMA/CD)
   $T_d \geq 2 * P_d + T_d$(Jam signal)
2. Backoff Algorithm
   Waiting time = K * Slot duration
   = K * RTT
   = K * 2 * P_d

   K is any random number in between 0 to 2^n -1.
   n is collision number (Collision number is respect to data packet).

3. Efficiency in Ethernet (CSMA/CD)

$$\eta = \frac{1}{1+6.44a} \text{ or } \eta = \frac{\text{useful time}}{\text{Total time}} = \frac{T_d}{\text{Collision time}+T_d+P_d}$$

4. $P(1-P)^{N-1}$ → Probability of success for single station(Throughput of Host)

$NP(1 - P)^{N-1}$ → Probability of success for any station among all stations [Throughput of channel]

5. Ethernet Frame Structure:

Preamble	SFD	DA	SA	Length Of Data	Data	CRC
7B	1B	6B	6B	2B	(46B – 1500B)	4B

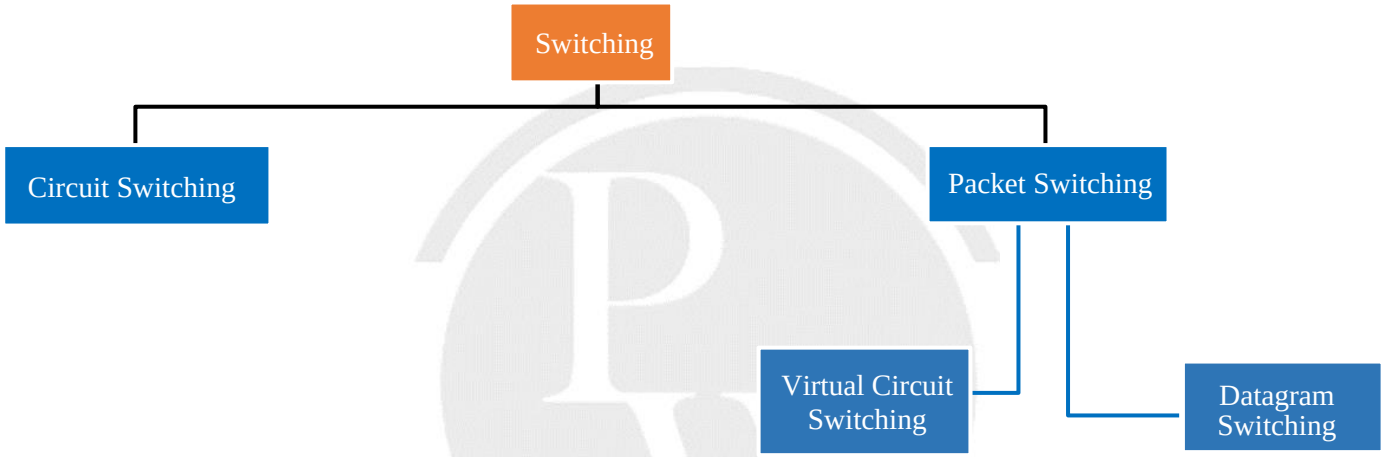
PL DLL DLL

Ethernet uses Manchester encoding technique for converting data bits into signal.
(Baud rate = 2 ∗ bit rate)
Bit rate = 1/2 baud rate

❑❑❑

7 ROUTING ALGORITHMS, SWITCHING & IP SUPPORT PROTOCOL

7.1 Switching



Circuit Switching	Packet switching
(I) It has three phases-connection establishment, Data transfer and Connection termination.	It has only one phase-Data transfer
(2) Physical path between source and destination	No physical path
(3) All packets use same path	Packet may follow different path (travel independently)
(4) Reserves the entire bandwidth in advance	Does not reserve
(5) Bandwidth wastage	No Bandwidth wastage
(6) No store and forward transmission	Supports store and forward transmission
(7) Congestion can happen during connection establishment phase	Congestion can happen during data transfer phase

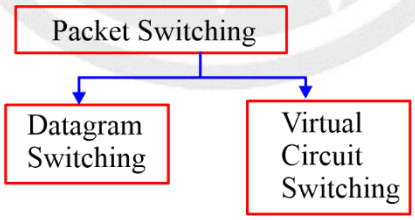
(8) It is reliable	Not reliable
(9) Better for sending large messages	Better for sending small messages
(10) Not fault tolerant technique	Fault tolerant technique
(11) Circuit switching is implemented at physical layer.	Packet switching is implemented at network layer
Total time= Setup time + T_d + P_d + Tear down time $TT=S+\frac{L}{B}+X.\frac{d}{V}+T$	For X Hop and N packet Total time= $X\,[T_d + P_d] + X-1[P_{rd} + Q_d] + N-1(T_d)$

7.2 Generalized Formula for optimal packet size(P)

M = Message size
h = Header size
p = Payload/Packet data size
No. of Hops = X

$p = \sqrt{\frac{Mh}{X-1}}$

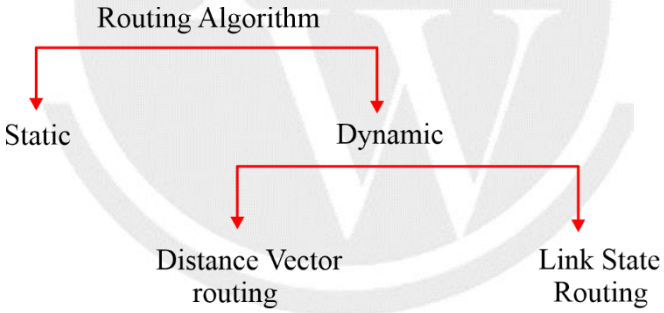
So optimum packet size P = p + h



Datagram Switching	Virtual circuit switching
(1) It is a connection less service.	It is connection-oriented service.
(2) All the packets may follow different path.	All the packet follows the same dedicated path.
(3) Data may appear out of order at the destination since the packets may follow different paths.	Data appears in order at the destination since all the packets take the same dedicated path.
(4) It is not reliable since packets may be discarded.	It is highly reliable.

(5) Cheap	Costly
(6) No resource reservation is done.	First packet reserves the resources (CPU, bandwidth and buffer) for the subsequent packets.
(7) All the packets require a global header which contains full information about the destination.	Only first packet requires a global header which identifies the path from one end to another end. All the following packets require a local header which identifies the path from hop to hop.
(8) The packets may be discarded at intermediate switches if sufficient resources are not available to process the packets.	The packets are never discarded at intermediate switches and immediately forwarded since resources are reserved for them.
(9) IP Networks uses datagram switching.	ATM (Asynchronous Transfer Mode) uses virtual circuit switching.
(10) Datagram switching is normally implemented at network layer.	Virtual circuit switching is normally implemented at data link layer.

7.3 Routing Algorithm



Distance vector Routing	Link state Routing
1. 1980's	1. 1990's
2. Bandwidth required is very less because we sent only distance vector packet.	2. Band width required is high because we sent entire link state packet
3. Local knowledge	3. Global knowledge
4. Bellman Ford algorithm	4. Dijkstra algorithm
5. Traffic is very less	5. Traffic is very high

6. Convergence is very low	6. Convergence is faster
7. Count to infinity Problem	7. No problem of count to infinity
8. Persistent Loops	8. Transient Loops
9. RIP	9. OSPF

The maximum Hop count allowed For RIP is 15 and Hop count of 16 is considered as Destination unreachable.

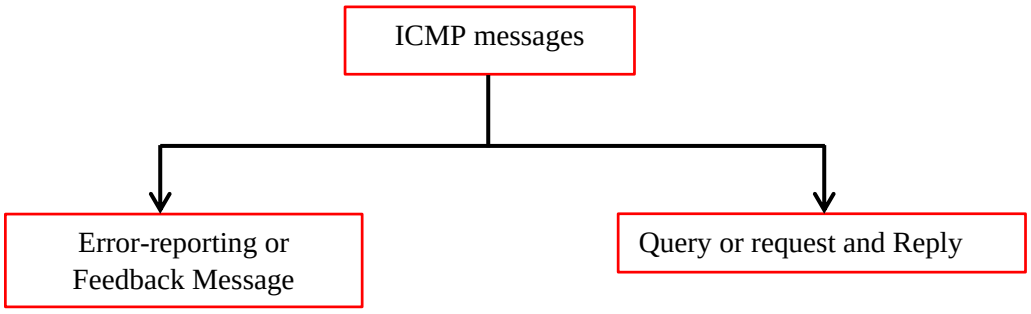
Note:
RIP uses UDP as its transport protocol with the port number – 530

7.4 IP Support Protocol

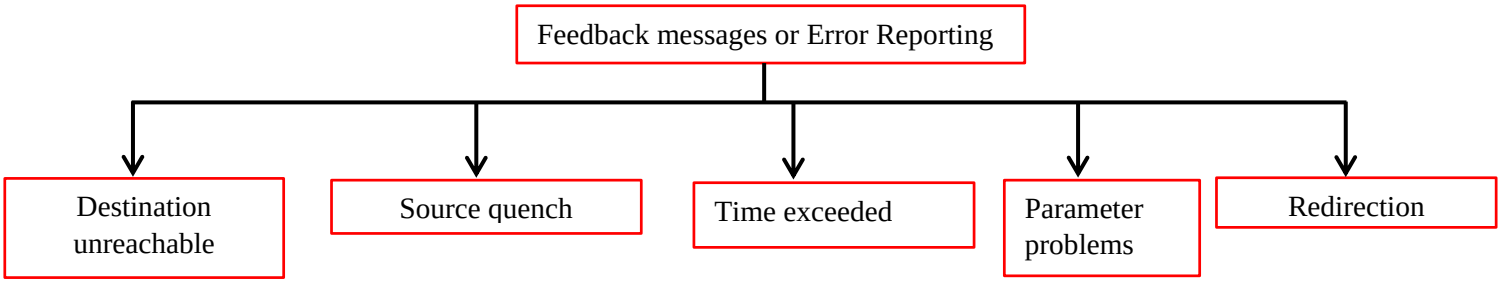
7.4.1 ARP

1. Address Resolution Protocol(ARP) is used to find the MAC(Media Access Control) address of a device from its IP address.
2. **ARP request:** ARP request is broadcasting
3. **ARP response/reply:** ARP reply is unicasting.

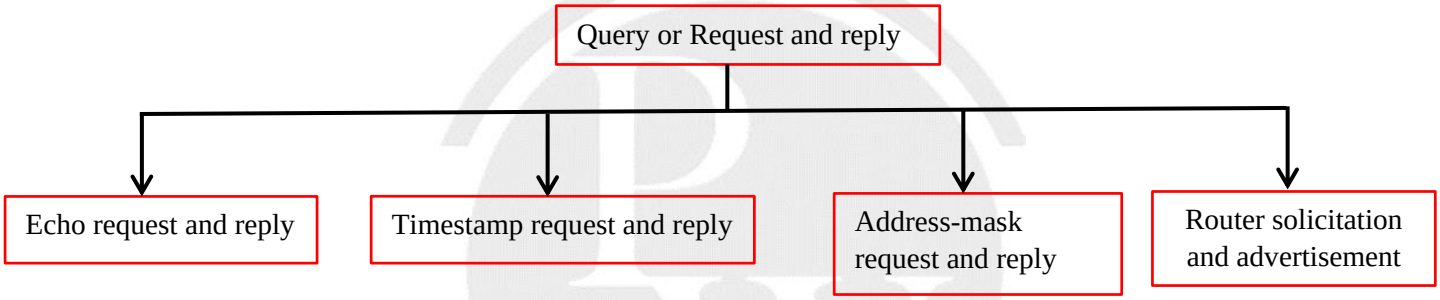
7.4.2 Internet Control Message Protocol (ICMP)



7.4.3 Internet Control Message Protocol (ICMP)



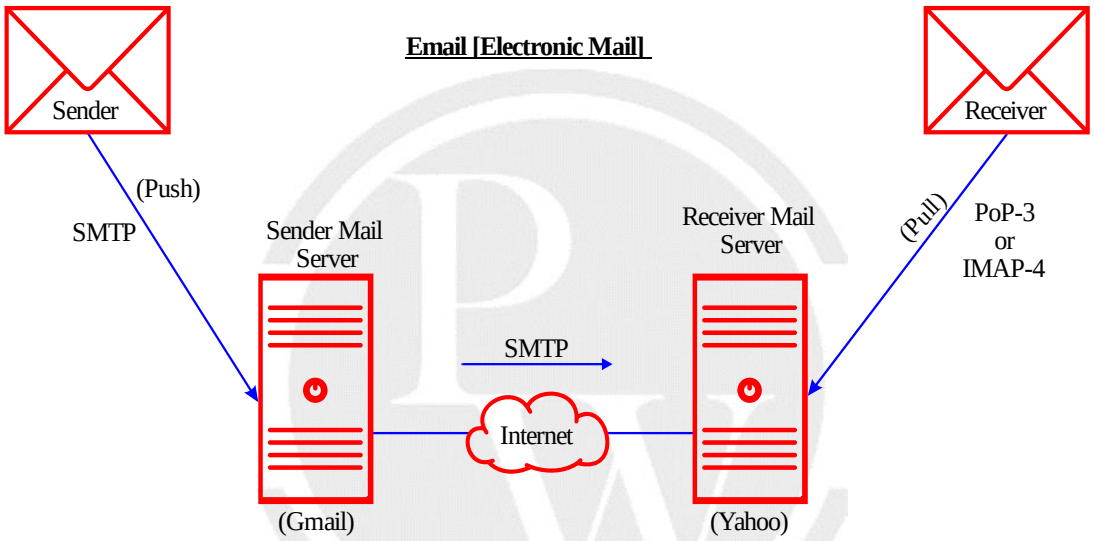
7.4.4 Internet Control Message Protocol (ICMP)



❑❑❑

8 APPLICATION LAYER PROTOCOL

8.1 E-MAIL



SMTP transfer the mail from sender's mail server to receiver's mail server.
While sending the mail ,SMTP is used two times-.

(i) Between the sender and sender's mail server
(ii) Between the sender's mail server and receiver's mail server

8.1.1 To receive or download the email,

Another protocol is needed between the receiver mail server and receiver.
The most commonly used protocols are POP3 and IMAP4.

8.1.2 SMTP(Simple mail transfer protocol)

1. The objective of SMTP is to transfer the email reliably and efficiently.
2. It uses port number-25 at TCP.
3. In SMTP there are two components:
   (i) User Agent (UA)
   (ii) Mail transfer Agent (MTA)
4. User Agent prepares the message, creates the envelope and put the message in the envelope.
5. Mail transfer Agent transfer the mail across the internet i.e. Actual mail transfer is done through MTA.
6. To send mail, system must have a client MTA and to receive the mail. it must have a server MTA.

7. SMTP is text based protocol.
8. With the help of SMTP & POP we can send only text messages.
9. SMTP can only handle the message containing 7 bit ASCII text.
10. SMTP cannot transfer other types of data like images, video, audio, etc.
11. SMTP cannot transfer binary files or executable files.
12. SMTP cannot transfer the text data for the language other than English (such as French, Japanese, and Chinese etc.).
13. Only SMTP is not sufficient to send binary files or to send videos or audio so we require MIME (Multipurpose Internet mail extension).
14. MIME is a supplementary protocol that allows non-ASCII data to send through SMTP.
15. MIME is a set of software function that transforms non-ASCII data to ASCII data or viceversa.
16. MIME is used to convert non text data to text data and text data to non text data.
17. SMTP is stateless protocol. It does not maintain any information of user. If an e-mail is asked to be sent twice, then server resends it without saying that e-mail has already been sent.
18. SMTP is a connection-oriented protocol.
19. SMTP uses persistent TCP connections, so it can send multiple e-mail at once.
20. SMTP is an “In-Band” protocol.
21. SMTP is used for Push the e-mail.
22. SMTP Pushes the mail from client to server on other hand, It needs a pull protocol(Download).
23. POP3 and IMAP4 are used for Pulling the e-mail.

8.2 POP3(Post office Protocol version-3)

1. It is a message access protocol.
2. It is a pull protocol.
3. POP3 uses port number-110 at TCP.
4. POP3 is a connection-oriented protocol.
5. POP3 uses persistent TCP connection.
6. POP3 is a stateful protocol.
7. POP3 is an “In-Band” protocol.
8. POP3 does not allow users to partially check the content of the mail before downloading.
9. POP3 does not allow user to organize the mail on the mail server.

8.2.1 IMAP-4(Internet Mail Access Protocol version-4)

1. IMAP-4 is similar to POP3 but it has more features. IMAP-4 is more powerful and more complex.
2. IMAP-4 provides the following extra functions.
3. A user checks the email header prior to downloading.
4. A user can search the content of the email for a specific string of characters prior to downloading.
5. A user can partially download the email.

6. A user can create, delete, or rename the mail box on the mail server.
7. A user can create a hierarchy of mailbox in a folder for email storage.

8.2.2 Characteristics of IMAP

1. IMAP is a pull protocol.
2. IMAP uses port number-143 at TCP.
3. IMAP is a connection-oriented protocol.
4. IMAP uses persistent TCP connection.
5. IMAP is a tasteful protocol.
6. IMAP is an "In-Band" Protocol.

POP3	IMAP
(1) Mails can only be accessed from a single device.	Mails can be accessed from multiple device.
(2) Download the email from server to a single computer and the copy at the server is deleted.	The email message is stored on the mail server itself.
(3) User cannot organize the mails in the mail box of the mail server.	User can organize mails on the mail server.
(4) It does not allow user to sync emails.	It allows user to sync their emails.
(5) It is unidirectional i.e all the changes made on a device does not effect the content present on the server.	It is bidirectional i.e all the chances made on server or device are made on the other side too.

8.3 File Transfer protocol (FTP)

8.3.1 FTP (File Transfer Protocol)

1. File transfer protocol is a standard internet protocol for transferring files b/w computers over TCP/IP connection.
2. It uses port number - 20 & 21 on TCP.
3. It has two types of connection
   (i) Control connection (port number. - 21)
   (ii) Data connection (port number - 20)
4. Control connection remains connected during the entire interactive FTP session.
5. The data connection is opened and closed for each file transfer activity.
6. When user starts an FTP session, the control connection opens. While the control connection is open, the data connection can be opened and closed multiple times if several files are transferred.
7. FTP uses persistent TCP connections for control information.
8. FTP uses Non-persistent TCP connections for data information.
9. FTP is a connection-oriented protocol.
10. FTP is an "out of band" protocol as data and control information flow over different connection.
11. Some protocols send their request and data in the same TCP connection for this reason they are called as In-bound protocol.
12. HTTP & SMTP are In-Band protocol.
13. FTP is state full protocol.

8.3.2 Transmission mode

FTP can transfer a file across the data connection using one of the following three transmission modes.

(i) Stream mode
(ii) Block mode
(ii) Compressed mode

8.3.3 File Type

FTP can transfer one of the following file types across the data connection:

(i) ASCII file
(ii) EBCDIC file (File format used by IBM)
(ii) Image file

8.3.4 Data structure

FTP can transfer a file across the data connection using one of the following interpretation of the structure of the data:

(i) File structure
(ii) Record structure
(iii) Page structure

8.4 HTTP Protocol

1. HTTP protocol is used mainly to access data on world wide web (www).
2. It is client server protocol using port number - 80 on TCP.
3. HTTP is “In-Band” protocol i.e. both request and data we will send only in one connection.
4. HTTP is a stateless protocol i.e. It does not maintain any information of user.
5. There are two types of HTTP protocol
   (i) Non persistent (1.0)
   (ii) Persistent (1.1)

8.5 Non Persistent (1.0)

In a Non persistent connection one TCP connection is ade for each request/response. This strategy follow the following steps :-

(i) The client opens a TCP connection and sends a request.
(ii) Server sends the response and closes the connection.
(iii) In this strategy , If a file contains link to N-different pictures in different files(all located on same server) the connection must be opened and closed N+1 times.

8.6 Persistent (1.1)

1. In a persistent connection the server leaves the connection open for more request after sending a response.
2. The server closes the connection at the request of client or time out has been reached.

8.6.1 Important Table

SHORT TRICK	DNS	HTTP	SMTP	POP	IMAP	FTP
Stateful/ Stateless	Stateless	Stateless	Stateless	Stateful	Stateful	Stateful
Transport Protocol Used	UDP	TCP	TCP	TCP	TCP	TCP
Connectionless/ Connection oriented	Connection less	Connection less	Connection oriented	Connection oriented	Connection oriented	Connection oriented
Persistent/Non-persistent	Non-persistent	HTTP 1.0 is non persistent HTTP 1.1 is persistent.	Persistent	Persistent	Persistent	Control connection is persistent. Data connection is non-persistent.
Push/Pull	-	-	Push	Pull	Pull	Can't
Port Number Used	53	80	25	110	143	20 for data connection. 21 for control connection.
In band/ Out-of-band	In band	In band	In band	In band	In band	Out-of- band

Application	Port Number.	Transport Protocol
DNS	53	UDP
HTTP	80	TCP
FTP	20 (Data connection) 21(Control connection)	TCP
SMTP	25	TCP
POP	110	TCP
SNMP	161, 162	UDP
TFTP	69	UDP
IMAP	143	TCP
Telnet	23	TCP
DHCP	67 (DHCP Server) 68 (DHCP Client)	UDP

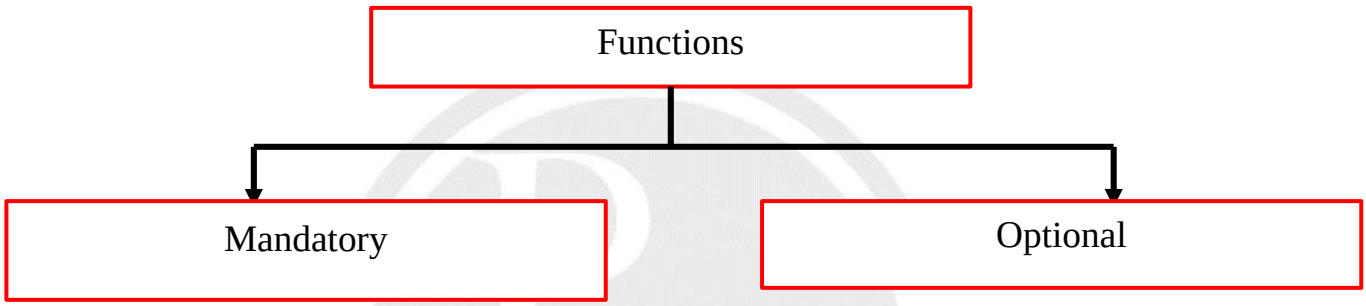
8.6.2 Commands

HTTP	FTP	SMTP
GET	USER	HELO
HEAD	PASS	MAIL FROM
PUT	ACCT	RCPT TO
POST	CWD	DATA
TRACE	REIN	QUIT
DELETE	QUIT	RSET
CONNECT	PORT*	VRFY
OPTIONS	PASV	NOOP
	TYPE	TURN
	MODE	EXPN
	PROMPT	HELP
	STRU	SEND FROM
		SMOL FROM
		SMAL FROM

❑❑❑

9 OSI AND TCP/IP PROTOCOL STACK

9.1 Functions of Computer Network



- Error control
- Flow control
- Access control
- Multiplexing and Demultiplexing
  :
  :

- Encryption & Decryption
- Check pointing
- Routing
  :
  :

9.2 OSI/ISO

OSI : Open systems Interconnection model.
ISO : International Standards Organization. It is a multinational body dedicated to worldwide agreement on international standard.

9.2.1 OSI Mode

- This model has been proposed by ISO.
- An open system is a set of protocols that allows any two different systems to communicate regardless of their underlying architecture (Hardware/Software).
- The purpose is to show how to facilitate communication between different systems without requiring changes to the logic of the underlying hardware & software.
- This model has got 7 separate but related layers.

9.3 Functions of Physical layer

9.3.1 Physical Layer

Physical Layer is responsible for movement of individuals bits from one Hop to next Hop.

9.3.2 Functions of physical Layer

1. It is used to define electrical, mechanical, functional and procedural characteristic of physical link. (physical Link)
   Copper → Electrical signal
   Fiber → Light signal
   Wireless comm. → Electromagnetic signal.

2. It defines transmission mode:
   a. Simplex
   b. Half duplex
   c. Full duplex

3. It defines topology configuration:
   - Bus topology
   - Star topology
   - Mesh Topology
   - Tree Topology

4. It is totally Hardware layer.

5. It defines link configuration:
   i Point to Point Link
   ii Broadcast Link
6. It defines Encoding.
7. Bits Synchronization.
8. Bit rate control.

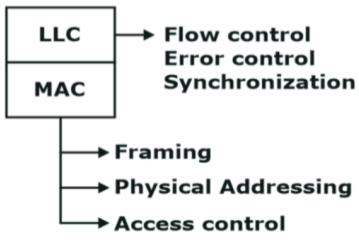
9.4 Data Link Layer

Data link layer is responsible for moving frames From One Hop (Node) to Next Hop (Node).

9.4.1 Function of data link layer

1. Flow control
2. Error control
3. Access control
4. Framing
5. Physical Addressing

Data Link Layer is divided into two parts



9.5 Network Layer

The network layer is responsible for the delivery of individual packet from source to destination.

9.5.1 Function of Network Layer

1. Host to Host connectively
2. Logical Addressing
3. Switching
4. Routing
5. Fragmentation
6. Congestion control:

9.6 Transport Layer

Transport Layer is responsible for process to process delivery. A process is an application program running on a host.

9.6.1 Function of Transport Layer

1. End to end connectively
2. Service point Addressing
3. Flow control
4. Error control
5. Segmentation and Reassembly
6. Congestion control
7. Connection Control
8. Multiplexing and Demultiplexing.

9.7 Session Layer

Session layer also known as **network dialog controller.** It establishes, maintains, synchronizes and terminates the interaction b/w sender and receiver.

9.7.1 Function of Session Layer

1. Authentication & Authorization
2. Check point or synchronization
3. Dialog control

9.8 Presentation Layer

This layer take care of syntax and semantics of the information exchange in between two communicating systems.

9.8.1 Function of Presentation Layer

1. Character translation
2. Encryption/Decryption
3. Compression

9.9 Application Layer

Application Layer is responsible for providing services to users. Users such as:

1. Mail services
2. File sharing
3. File transfer and many more

9.10 TCP/IP Model

This mode has got 5 different layer

Physical : No specific protocol

Data Link : No specific protocol

Network : ARP, RARP, ICMP, IGMP

Transport : TCP, UDP,SCTP

Application : DNS, SMTP, HTTP, FTP, POP,IMAP, SNMP , Telnet etc.

❑❑❑

GATE Exam 2024?

PARAKRAM BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in ***Hindi***

Weekday & Weekend Batches Available

JOIN NOW!

GENERAL APTITUDE

General Aptitude

INDEX

1 PERCENTAGES

1.1. Understanding Percentages

The word percent can be understood as follows:

Per cent ⇒ for every 100.

So, when percentage is calculated for any value, it means that that you calculate the value for every 100 of the reference value.

Why Percentage?

Percentage is a concept evolved so that there can be a uniform platform for comparison of various things. (Since each value is taken to a common platform of 100.)

Example:

- To compare three different students depending on the marks they scored we cannot directly compare their marks until we know the maximum marks for which they took the test. But by calculating percentages they can directly be compared with one another.
- Before going deeper into the concept of percentage, let u have a look at some basics and tips for faster calculations:

1.2. Calculation of Percentage

$$\text{Percentage} = \left(\frac{\text{Value}}{\text{Total value}}\right) \times 100$$

Example: 50 is what % of 200?

Solution: $\text{Percentage} = \left(\frac{50}{200}\right) \times 100 = 25\%$.

1.2.1. Calculation of Value:

$$\text{Value} = \left(\frac{\text{Percentage}}{100}\right) \times \text{ total value}$$

Example: What is 20% of 200?

Solution: $\text{Value} = \left(\frac{20}{100}\right) \times 200$

Note: Percentage is denoted by "%", which means "/100".

Example: What is the decimal notation for 35%?

Solution: $35\% = \frac{35}{100} = 0.35$.

For faster calculations we can convert the percentages or decimal equivalents into their respective fraction notations.

1.3. Percentages – Fractions Conversions:

The following is a table showing the conversions of percentages and decimals into fractions:

Percentage	Decimal	Fraction
10%	0.1	$\frac{1}{10}$
12.5%	0.125	$\frac{1}{8}$
16.66%	0.1666	$\frac{1}{6}$
20%	0.2	$\frac{1}{5}$
25%	0.25	$\frac{1}{4}$
30%	0.3	$\frac{3}{10}$
33.33%	0.3333	$\frac{1}{3}$
40%	0.4	$\frac{2}{5}$
50%	0.5	$\frac{1}{2}$
60%	0.6	$\frac{3}{5}$
62.5%	0.625	$\frac{5}{8}$
66.66%	0.6666	$\frac{2}{3}$

Similarly we can go for converting decimals more than 1 from the knowledge of the above cited conversions as follows:

We know that $12.5\% = 0.125 = \frac{1}{8}$

Then, $1.125 = \frac{[8(1)+1]}{8} = \frac{9}{8}$ (i.e., the denominator will add to numerator once, denominator remaining the same.

Also, $2.125 = \frac{[8(2)+1]}{8} = \frac{17}{8}$ (here the denominator is added to numerator twice)

$3.125 = \frac{[8(3)+1]}{8} = \frac{25}{8}$ and so on.

Thus we can derive the fractions for decimals more than 1 by using those les than 1.

We will see how use of fractions will reduce the time for calculations:

Example: What is 62.5% of 320?

Solution: $\text{Value} = \left(\frac{5}{8}\right) \times 320 \left(\text{since } 62.5\% = \frac{5}{8}\right) = 200.$

1.4. Percentage Change

A change can be of two types – an increase or a decrease.

When a value is changed from initial value to a final value,

% change = (Difference between initial and final value/initial value) × 100

Example: If 20 changes to 40, what is the % increase?

Solution: $\%\ increase = \frac{(40-20)}{20} \times 100 = 100\%.$

Note:
1. If a value is doubled the percentage increase is 100.
2. If a value is tripled, the percentage change is 200 and so on.

1.5. Percentage Difference

% Difference = (Difference between values/value compared with) × 100.

Example: By what percent is 40 more than 30?

Solution: $\%\ \text{difference} = \frac{(40-30)}{30} \times 100 = 33.33\%$

(Here 40 is compared with 30. So 30 is taken as denominator)

Example: By what % is 60 more than 30?

Solution: $\%\ \text{difference} = \frac{(60-30)}{30} \times 100 = 100\%.$

(Here is 60 is compared with 30.)

Hint: To calculate percentage difference the value that occurs after the word "than" in the question can directly be used as the denominator in the formula.

1.6. Important Points to Note

1. When any value increases by

(a) 10%, it becomes 1.1 times of itself. (since 100+10 = 110% = 1.1)

(b) 20%, it becomes 1.2 times of itself.

(c) 36%, it becomes 1.36 times of itself.

(d) 4%, it becomes 1.04 times of itself.

Thus we can see the effects on the values due to various percentage increases.

2. When any value decreases by

(a) 10%, it becomes 0.9 times of itself. (Since 100 – 10 = 90% = 0.9)

(b) 20%, it becomes 0.8 times of itself

(c) 36%, it becomes 0.64 times of itself

(d) 4%, it becomes 0.96 times of itself.

Thus we can see the effects on a value due to various percentage decreases.

Note:
1. When a value is multiplied by a decimal more than 1 it will be increased and when multiplied by less than 1 it will be decreased.
2. The percentage increase or decrease depends on the decimal multiplied.

Example: 0.7 ⇒ 30% decrease, 0.67 ⇒ 33% decrease, 0. 956 ⇒ 4.4% decrease and so on.

Example: When the actual value is x, find the value when it is 30% decreased.

Solution: 30% decrease ⇒ 0.7 x.

Example: A value after an increase of 20% became 600. What is the value?

Solution: 1.2x = 600 (since 20% increase)

⇒ x = 500.

Example: If 600 is decrease by 20%, what is the new value?

Solution: New value = 0.8 × 600 = 480. (Since 20% decrease)

Thus depending on the decimal we can decide the % change and vice versa.

Example: When a value is increased by 20%, by what percent should it be reduced to get the actual value?

Solution: (It is equivalent to 1.2 reduced to 1 and we can use % decrease formula)

$$\% \text{ decrease} = \frac{(1.2 - 1)}{1.2} \times 100 = 16.66\% .$$

3. When a value is subjected multiple changes, the overall effect of all the changes can be obtained by multiplying all the individual factors of the changes.

Example: The population of a town increased by 10%, 20% and then decreased by 30%. The new population is what % of the original?

Solution: The overall effect = 1.1 × 1.2 × 0.7 (Since 10%, 20% increase and 30% decrease)

= 0.924 = 92.4%.

Example: Two successive discounts of 10% and 20% are equal to a single discount of ___

Solution: Discount is same as decrease of price.

So, decrease = 0.9 × 0.8 = 0.72 $\Rightarrow$ 28% decrease (Since only 72% is remaining).

❐❐❐

2 AVERAGES & AGES

2.1. What is Average?

The concept of average is equal distribution of the overall value among all the things or persons present there. So the formula for finding the average is as follows:

Average, $A = \frac{\text{Total of all things, } T}{\text{Number of things, } N}$

Therefore, Total, $T = AN$

If any person joins a group with more value than the average of the group then the overall average increases. This is because the value in excess than the average will also be distributed equally among all the members.

Similarly when any value less than the average joins the group the overall group decreases as the deficit is divided equally among all the people present there.

Example:

Consider three people A, B and C with total of Rs. 30/-. Their average becomes Rs. 10/- for each. If another person D joins them with Rs. 50/- then he has Rs. 40/- more than actual average of Rs. 10/-.

So this Rs. 40/- will get distributed among those four and each gets Rs. 10/-. Thus the average becomes Rs. 20/- each.

Example:

The average age of a class of 30 students is 12. If the teacher is also included the average becomes 13 years. Find the teacher's age.

Solution:

- When the teacher is included there are totally 31 members in the class and the average is increased by 1 year. This means that everyone got 1 extra year after distributing the extra years of the teacher.
- So extra years of the teacher are as follow: 31 × 1 = 31 years.
- Age of the teacher = actual avg + extra years = 12 + 31 = 43 years.

3 PROFIT AND LOSS

3.1. What is Profit?

When a person does a business transaction and gets more than what he had invested, then he is said to have profit. The profit he gets will be equal to the additional money he gets other than his investment.

So profit can be understood as the extra money one gets other than what he had invested.

Example: A person bought an article for Rs. 100 and sold it for Rs. 120. Then he got Rs. 20 extra and so his profit is Rs. 20.

3.2. What is Loss?

When a person gets an amount less than what he had invested, then he is said to have a loss. The loss will be equal to the deficit he got than the investment.

Example: A person bought an article at Rs. 100 and sold it for Rs. 90. Then he got a deficit of Rs. 10 and so his loss is Rs. 10.

3.3. Cost Price (CP)

- The money that the trader puts in his business is called Cost Price. The price at which the articles are bought is called Cost Price.
- In other words, Cost Price is nothing but the investment in the business.

3.4 Selling Price (SP)

- The price at which the articles are sold is called the Selling Price. The money that the trader gets from the business is called Selling Price.
- In other words, Selling Price is nothing but the returns from a business.

3.5. Marked/Market/List Price (MP):

- The price that a trader marks or lists his articles to is called the Marked Price.
- This is the only price known to the customer.

3.6. Discount

The waiver of cost from the Marked Price that the trader allows a customer is called Discount.

Note:

1. Profit or loss percentage is to be applied always to the Cost Price only.
2. Discount percentage is to be applied always to the Marked Price only.

3.7. Relationship Among CP, SP and MP:

A trader adds his profit to the investment and sells it at that increased price.
Also he allows a discount on Marked Price and sells at the discounted price.
So, we can say that,

- SP = CP + Profit. (CP applied with profit is SP)
- SP = MP – Discount. (MP applied with discount is SP)

3.8. Understanding Profit and Loss:

So, by now we came to know that if CP is increased and sold it would result in profit and vice versa.

Also whatever increase is applied to CP, that increase itself is the profit.

For Rs. 10 profit, CP is to be increased by RS. 10 and the increased price becomes SP.

For 10% profit, CP is to be increased by 10% and it is the SP.

(From previous chapter we know that any value increased by 10% becomes 1.1 times.)

So, for 10% profit, CP increased by 10% $\Rightarrow$ 1.1CP = SP.

- $SP = 1.1CP \Rightarrow \frac{SP}{CP} = 1.1 \Rightarrow$ 10% profit
- $SP = 1.07CP \Rightarrow \frac{SP}{CP} = 1.07 \Rightarrow$ 7% profit
- $SP = 1.545CP \Rightarrow \frac{SP}{CP} = 1.545 \Rightarrow$ 54.5% profit and so on.

Similarly,

- $SP = 0.9CP \Rightarrow \frac{SP}{CP} = 0.9 \Rightarrow$ 10% loss (Since 10% decrease)
- $SP = 0.7CP \Rightarrow \frac{SP}{CP} = 0.76 \Rightarrow$ 24% loss and so on.

So, to calculate profit % or loss %, it is enough for us to find the ratio of SP to CP.

Note:

1. If SP/CP > 1, it indicates profit.
2. If SP/CP < 1, it indicates loss.

3.9. Multiple Profits or Losses

A trader may sometimes have multiple profits or losses simultaneously. This is equivalent to having multiple changes and so all individual changes are to be multiplied to get the overall effect.

Examples: A trader uses a 800gm weight instead of 1 kg. Find his profit %.

Solution: (He is buying 800 gm but selling 1000 gm.

So, CP is for 800 gm and SP is for 1000 gm.)

$$\frac{SP}{CP}=\frac{1000}{800} = 1.25 \Rightarrow 25\% \text{ profit.}$$

Examples: A trader uses 1 kg weight for 800 gm and increases the price by 20%. Find his profit/loss %.

Solution: 1 kg weight for 800 gm $\Rightarrow$ loss (decrease) $\Rightarrow$ 800/1000 = 0.8

20% increase in price $\Rightarrow$ profit (increase) $\Rightarrow$ 1.2

So, net effect = (0.8) × (1.2) = 0.96 $\Rightarrow$ 4% loss.

Examples: A milk vendor mixes water to milk such that he gains 25%. Find the percentage of water in the mixture.

Solution: To gain 25%, the volume has to be increased by 25%.

So, for 1 lt of milk, 0.25 lt of water is added $\Rightarrow$ total volume = 1.25 lt

$$\% \text{ of water} = \frac{0.25}{1.25}\times 100 = 20\%.$$

Examples: A trader bought an item for Rs. 200. If he wants a profit of 22%, at what price must he sell it?

Solution: CP=200, Profit = 22%.

So, SP = 1.22CP = 1.22 × 200 = 244/-.

Examples: A person buys an item at Rs. 120 and sells to another at a profit of 25%. If the second person sells the I tem to another at Rs. 180, what is the profit % of the second person?

Solution: SP of 1st person = CP of 2nd person = 1.25 × 120 = 150.

SP of 2nd person = 180.

$$\text{Profit \%} = \frac{SP}{CP} = \frac{180}{150} = 1.2 \Rightarrow 20\%.$$

❐❐❐

4 RATIOS AND PROPORTIONS

4.1. What is a Ratio?

A ratio is a representation of distribution of a value present among the persons present and is shown as follows:

If a total is divided among A, B and C such that A got 4 parts, B got 5 parts and C got 6 parts then it is represented in ratio as A:B:C = 4:5:6.

So, 4:5:6 means that the total value is divided into 4+5+6 = 15 equal parts and then distributed as per the ratio.

Example 1: Divide Rs. 580 between A and B in the ratio of 14:15.

Solution: A:B = 14:15 $\Rightarrow$ 580 is divided into 29 equal parts $\Rightarrow$ each part = Rs. 20.

So A's share = 14 parts = 14 × 20 = Rs. 280

B's share = 15 parts = Rs. 300.

Example 2: If A:B = 2:3 and B:C = 4:5 then find A:B:C.

Solution: To combine two ratios the proportions common for them shall be in equal parts. Here the common proportion is B for the given ratios.

Making B equal in both ratios they become 8:12 and 12:15 $\Rightarrow$ A:B:C = 8:12:15.

Example 3: Three numbers are in the ratio of 3: 4 : 8 and the sum of these numbers is 975. Find the three numbers.

Solution: Let the numbers be $3x$, $4x$ and $8x$. Then their sum = $3x + 4x + 8x = 15x = 975 \Rightarrow x = 65$.

So the numbers are $3x = 195$, $4x = 260$ and $8x = 520$.

Example 4: Two numbers are in the ratio of 4 : 5. If the difference between these numbers is 24, then find the numbers.

Solution: Let the numbers be $4x$ and $5x$. Their difference = $5x - 4x = x = 24$ (given).

So the numbers are $4x = 96$ and $5x = 120$.

Example 5: Given two numbers are in the ratio of 3 : 4. If 8 is added to each of them, their ratio is changed to 5 : 6. Find two numbers.

Solution: Let the numbers be a and b.

$A:B = 3:4 \Rightarrow \frac{A}{B} = \frac{3}{4}$. Also, $\frac{(A+8)}{(B+8)} = \frac{5}{6}$.

Solving we get, A=12 and B = 16.

❐❐❐

5 TIME AND DISTANCE

5.1. Speed

We have the relation between speed, time and distance as follows:

$$\text{Speed} = \frac{\text{Distance}}{\text{Time}}$$

So the distance covered in unit time is called speed.

This forms the basis for Time and Distance. It can be re-written as Distance = Speed X Time or

$$\text{Time} = \frac{\text{Distance}}{\text{Speed}}.$$

5.1.1. Units of Speed

The units of speed are kmph (km per hour) or m / s.

$$1 \text{ kmph} = \frac{5}{18} \text{ m/s}$$

$$1 \text{ m/s} = \frac{18}{5} \text{ kmph}$$

5.1.2. Average Speed

When the travel comprises of various speeds then the concept of average speed is to be applied.

$$\text{a Average Speed} = \frac{\text{Total distance covered}}{\text{Total time of travel}}$$

Note: In the total time above, the time of rest is not considered.

Example 1: If a car travels along four sides of a square at 100 kmph, 200 kmph, 300 kmph and 400 kmph find its average speed.

Solution: $\text{Average Speed} = \frac{\text{Total distance}}{\text{Total time}}$.

Let each side of square be x km. Then the total distance = $4x$ km.

The total time is sum of individual times taken to cover each side.

To cover x km at 100 kmph, time $= \frac{x}{100}$.

For the second side time $= \frac{x}{200}$.

Using this we can write average speed $= \frac{4x}{\left(\frac{x}{100}+\frac{x}{200}+\frac{x}{300}+\frac{x}{400}\right)} = 192$ kmph.

Example 2: A man if travels at $\frac{5}{6}$ th of his actual speed takes 10 min more to travel a distance. Find his usual time.

Solution: Let s be the actual speed and t be the actual time of the man.

Now the speed is $\left(\frac{5}{6}\right)$s and time is (t +10) min. But the distance remains the same.

So distance 1 = distance 2 $\Rightarrow s \times t = \left(\frac{5}{6}\right)$s × (t+10) $\Rightarrow$ t = 50 min.

Example 3: If a person walks at 30 kmph he is 10 min late to his office. If he travels at 40 kmph then he reaches to his office 5 min early. Find the distance to his office.

Solution: Let the distance to his office be d. The difference between the two timings is given as 15 min $= \frac{1}{4}$ hr.

Now if *d* km are covered at 30 kmph then time = d/30. Similarly second time = d/40.

So, $\frac{d}{30} - \frac{d}{40} = \frac{1}{4} \Rightarrow d = 30$ km.

Note: When two objects move with speeds s_1 and s_2

- In opposite directions their combined speed = $s_1 + s_2$
- In same direction their combined speed = $s_1 \sim s_2$.

Example 4: Two people start moving from the same point at the same time at 30 kmph and 40 kmph in opposite directions. Find the distance between them after 3 hrs.

Solution: Speed = 30 + 40 = 70 kmph (since in opposite directions)

Time = 3 hrs

So distance = speed × time = 70 × 3 = 210 km.

Example 5: A starts from X to Y at 6 am at 40 kmph and at the same time B starts from Y to X at 50 kmph. When will they meet if X and Y are 360 km apart?

Solution: Distance = 360 km, Speed = 40 + 50 = 90 kmph.

Time $= \frac{\text{distance}}{\text{speed}} = \frac{360}{90} = 4$hrs from 6 am $\Rightarrow$10 am.

❐❐❐

6 TIME AND WORK

6.1. Introduction

If a person can complete a work in 'n' days then he can do $\frac{1}{n}$ part of the work in one day.

The amount of work done be a person in 1 day is called his efficiency.

Example: A can do a work in 10 days. Then the efficiency of A is given by $A = \frac{1}{10}$.

Note: Number of days required to do a work = work to be done/work per day.

Example 1: If A can do a work in 10 days, B can do it in 20 days and C in 30 days in how many days will the three together do it?

Solution: The efficiencies are $A = \frac{1}{10}, B = \frac{1}{20}$ *and* $C = \frac{1}{30}$

So work done per day by the three $= \frac{1}{10} + \frac{1}{20} + \frac{1}{30} = \frac{11}{60} \Rightarrow No\ of\ days = \frac{60}{11} = 5.45\ days$.

Example 2: If A and B can do a work in 10 days, B and C can do it in 20 days and C and A can do it in 40 days in what time all the three can do it?

Solution: $A + B = \frac{1}{10}$

$B + C = \frac{1}{20}$

$C + A = \frac{1}{40}$

Adding all the three we get $2(A + B + C) = \frac{7}{40} \Rightarrow A + B + C = \frac{7}{80} \Rightarrow No\ of\ days = \frac{80}{7}\ days$.

Note: If all the people do not work for all the time then the principle below can be used:

$$mA + nB + \mathbf{oC = 1}. \quad (1 \text{ is the total work})$$

Here, m = no of days A worked

n = no of days B worked

o = no of days C worked

A, B, C = efficiencies

Example 3: If A can do a work in 12 days, B can do it in 18 days and C in 24 days. All the three started the work. A left after two days and C left three days before the completion of the work. How many days are required to complete the work?

Solution: Let the total no of days be x.

A worked only for 2 days, B worked for x days and C worked for $x - 3$ days.

So, mA + nB + oC = 1

$$\Rightarrow 2\left(\frac{1}{12}\right) + x\left(\frac{1}{18}\right) + (x-3)\left(\frac{1}{24}\right) = 1$$

$$\Rightarrow 12 + 4x + 3(x-3) = 72$$

$$\Rightarrow x = \frac{69}{7} \text{ days}.$$

Note: The ratio of dividing wages = ratio of efficiencies = ratio of parts of work done

Example 4: A can do a work in 10 days and B can do it in 30 days and C in 60 days. If the total wages for the work is Rs. 1800 what is the share of A?

Solution: Ratio of wages = $\frac{1}{10} : \frac{1}{30} : \frac{1}{60} = 6 : 2 : 1$ (Multiplying each term by LCM 60)

So total 9 equal parts in Rs. 1800 $\Rightarrow$ each part = Rs. 200 $\Rightarrow$ share of A = 6 parts = Rs. 1200.

Note: When pipes are used filling the tank they are treated similar to the men working but some outlet pipes emptying the tank are present whose work will be considered negative.

Example 5: A pipe can fill a tank in 5 hrs but because of a leak a the bottom it takes 1 hr extra. In what time can the leak alone empty the tank?

Solution: Let the filling pipe be A.

$$A = \frac{1}{5}$$

But with the leak L, $A - L = \frac{1}{6}$ (A-L because leak is outlet)

So, $\frac{1}{L} = \frac{1}{5} - \frac{1}{6} = \frac{1}{30} \Rightarrow$ Leak can empty the tank in 30 hrs.

❐❐❐

7 CLOCKS

7.1. Introduction

In a clock the most important hands are the minutes hand and the hours hand. Whatever may be the shape of the dial they move in a circular track.

The total angle of 360 degrees in a watch is divided into 1 sectors, one for each hour.

So one hour sector = $\frac{360}{12} = 30$ degrees.

For every one hour (60 min),

- The minutes hand moves through 360 deg.
- The hours hand moves through 30 deg.

So for every minute,

- The minutes hand moves through 6 deg
- The hours hand moves through 0.5 deg.

They move in same direction. So their relative displacement for every minute is 5.5 deg.

This 5.5 deg movement constitutes the movements of both the hands.

So for every minute both the hands give a displacement of 5.5 deg.

Note:

1. Between every two hours i.e., between 1 and 2, 2 and 3 and so on the hands of the clock coincide with each other for one time except between 11, 12 and 12, 1.
   In a day they coincide for 22 times.
2. Between every two hours they are perpendicular to each other two times except between 2, 3 and 3, 4 and 8, 9 and 9, 10.
   In a day they will be perpendicular for 44 times.
3. Between every two hours they will be opposite to each other one time except between 5, 6 and 6, 7.
   In a day they will be opposite for 22 times.

Examples: At what time between 5 and 6 will the hands of the clock coincide?

Solution: At 5 the angle between the hands is 150 deg.

To coincide, they collectively have to travel this distance. Every minute they travel 5.5 deg.

So no. of minutes required to coincide = $\frac{150}{5.5} = \frac{300}{11} = 27\frac{3}{11}\ min$.

Examples: At what time between 6 and 7 will the hands be perpendicular?

Solution: At 6 the angle between the hands is 180 deg.

To form 90 deg they have to cover 90 deg (out of 180 if 90 is covered 90 will remain)

So no. of minutes required = $\frac{90}{5.5} = \frac{180}{11} = \frac{16\,4}{11} min$.

But they will be perpendicular for two times. The second one will happen after the minutes hand crosses the hours hand and then for 90 deg.

So it has to travel 180 + 90 = 270 deg.

So time = $\frac{270}{5.5} = \frac{540}{11} = 49\frac{1}{11} min$.

Examples: What is the angle between the hands of the clock at 3.45?

Solution: At 3, the angle between the hands = A = 90 deg.

In 45 min the hands will move angle of B = 45 × 5.5 deg (since 5.5 deg for 1 min)

B = 247.5 deg.

Required angle = A ~ B = 157.5 deg.

Examples: What is the angle between the hands at 4.40?

Solution: At 4 the angle between the hands, A = 120 deg.

In 40 min, B = 40 × 5.5 = 220 deg.

The required angle = A ~ B = 100 deg.

Examples: A clock loses 5 min for every hour and another gains 5 min for every hour. If they are set correct at 10 am on Monday then when will they be 12 hrs apart?

Solution: For every hour watch A loses 5 min and watch B gains 5 min.

So for every hour they will differ by 10 min.

For 12 hrs (720 min) difference between them the time required = $\frac{720}{10} = 72\ hrs$

So they will be 12 hrs apart after 3 days i.e., at 10 am on Thursday.

❐❐❐

8 CALENDARS

8.1. Calendars

Here you mainly deal in finding the day of the week on a particular given date.

The process of finding this depends on the number of odd days.

Odd days are quite different from the odd numbers.

- **Odd Days:** The days more than the complete number of weeks in a given period are called odd days.
- **Ordinary Year:** An year that has 365 days is called Ordinary Year.
- **Leap Year:** The year which is exactly divisible by 4 (except century) is called a leap year.

Example: 1968, 1972, 1984, 1988 and so on are the examples of Leap Years.

1986, 1990, 1994, 1998, and so on are the examples of non leap years.

Note: The Centuries divisible by 400 are leap years.

Important Points:

- An ordinary year has 365 days = 52 weeks and 1 odd day.
- A leap year has 366 days = 52 weeks and 2 odd days.
- Century = 76 Ordinary years + 24 Leap years.
- Century contain 5 odd days.
- 200 years contain 3 odd days.
- 300 years contain 1 odd day.
- 400 years contain 0 odd days.
- Last day of a century cannot be Tuesday, Thursday or Saturday.
- First day of a century must be Monday, Tuesday, Thursday or Saturday.

Explanation:

100 years = 76 ordinary years + 24 leap years

= 76 odd days + 24 × 2 odd days

= 124 odd days = 17 weeks + 5 days

∴ 100 years contain 5 odd days.

No. of odd days in first century = 5

∴ Last day of first century is Friday.

No. of odd days in two centuries = 3

∴ Wednesday is the last day.

No. of odd days in three centuries = 1

∴ Monday is the last day.

No. of odd days in four centuries = 0

∴ Sunday is the last day.

Since the order is continually kept in successive cycles, the last day of a century cannot be Tuesday, Thursday or Saturday.

So, the last day of a century should be Sunday, Monday, Wednesday or Friday.

Therefore, the first day of a century must be Monday, Tuesday, Thursday or Saturday.

8.2. Working Rules

Working rule to find the day of the week on a particular date when reference day is given:

Step 1: Find the net number of odd days for the period between the reference date and the given date (exclude the reference day but count the given date for counting the number of net odd days).

Step 2: The day of the week on the particular date is equal to the number of net odd days ahead of the reference day (if the reference day was before this date) but behind the reference day (if this date was behind the reference day).

Working rule to find the day of the week on a particular date when no reference day is given

Step 1: Count the net number of odd days on the given date

Step 2: Write:

For 0 odd days – Sunday

For 1 odd day – Monday

For 2 odd days – Tuesday

⋮ ⋮ ⋮

For 6 odd days - Saturday

Examples: If 11th January 1997 was a Sunday then what day of the week was on 10th January 2000?

Solution: Total number of days between 11th January 1997 and 10th January 2000

= (365 – 11) in 1997 + 365 in 1998 + 365 in 1999 + 10 days in 2000

= (50 weeks + 4 odd days) + (52 weeks + 1 odd day) + (52 weeks + 1 odd day) + (1 week + 3 odd days)

Total number of odd days = 4 + 1 + 1 + 3 = 9 days = 1 week + 2 days

Hence, 10th January, 2000 would be 2 days ahead of Sunday i.e. it was on Tuesday.

Examples: What day of the week was on 10th June 2008?

Solution: 10th June 2008 = 2007 years + First 5 months up to May 2008 + 10 days of June

2000 years have 0 odd days.

Remaining 7 years has 1 leap year and 6 ordinary years $\Rightarrow$ 2 + 6 = 8 odd days

So, 2007 years have 8 odd days.

No. of odd days from 1st January 2008 to 31st May 2008 = 3+1+3+2+3 = 12

10 days of June has 3 odd days.

Total number of odd days = 8+12+3 = 23

23 odd days = 3 weeks + 2 odd days.

Hence, 10th June, 2008 was Tuesday.

❐❐❐

9 BLOOD RELATIONS

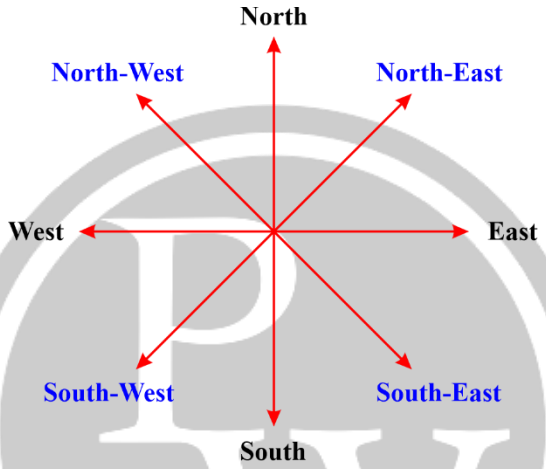
9.1. Introduction

The standard definitions of relations are given below

A/An ⇓	is related to a PERSON as ⇓
Grandfather	The father of his/her mother or father
Grandmother	The mother of his/her mother or father
Grandson	The son of his/her daughter/son
Granddaughter	The daughter of his/her daughters/son
Uncle	The brother of his/her mother or father
Aunt	The sister of his/her mother or father
Nephew	The son of his/her brother or sister
Cousin	The son or daughter of his/her aunt of uncle
Niece	The daughter of his/her brother or sister
Spouse	as her husband or his wife
Father-in-law	the father of his/her spouse
Mother-in-law	the mother of his/her spouse
Sister-in-law	the sister of his/her spouse
Brother-in-law	the brother of his/her spouse
Son-in-law	the spouse of his/her daughter
Daughter-in-law	the spouse of his/her son

10 DIRECTIONS

10.1. Introduction

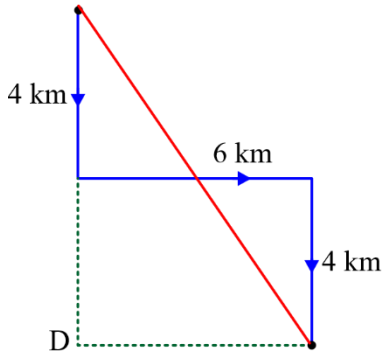


Examples:

Example 1: Ravi traveled 4 km straight towards south. He turned left and traveled 6 km straight, then turned right and traveled 4 km straight. How far is he from the starting point?

(a) 8 km (b) 10 km (c) 18 km (d) 12 km

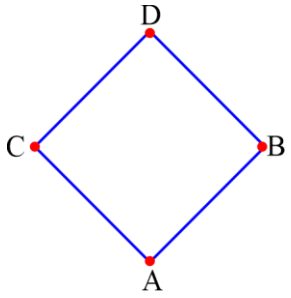
Solution. 10 km. B is the finishing point and A is starting point. The distance of A from B is



Example 2. A is to the South-East of C, B is to the East of C and North-East of A. If D is to the North of A and North-West of B. In which direction of C is D located?

(a) North-West (b) South-West (c) North-East (d) South-East

Solution. North-East D is located to the North-East of C.

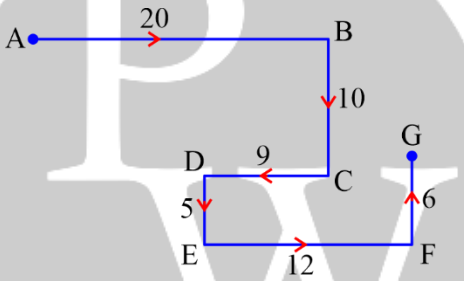


Example 3. A rat runs 20' towards East and turns to right, runs 10' and turns to right, runs 9' and again turns to left, runs 5' and then runs to left, runs 12' and finally turns to left and runs 6'. Now, which direction is the rat facing?

(a) East (b) West

(c) North (d) South

Solution. North. The movements of the rat from A to G are as shown in the fig. It is clear, rat is walking in one direction FG, *i.e.*, North.

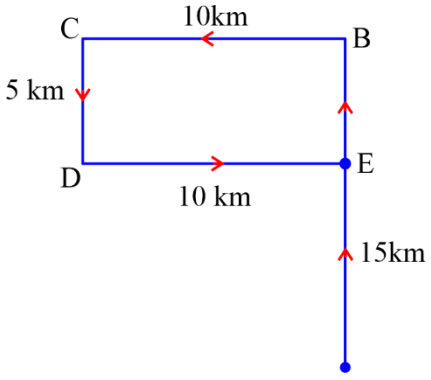


Example 4. From his house, Lokesh went 15 km to the North. Then he turned west and covered 10 km, then he turned South and covered 5 km. Finally, turning to East, he covered 10 km. In which direction is he from his house?

(a) East (b) South

(c) North (d) West

Solution. North. Starting point is A and ending point is E. E is to the north of his house at A.



❐❐❐

11 DATA INTERPRETATION

11.1. Data Interpretation

- It deals with careful reading, understanding, organizing and interpreting the data provided so as to derive meaningful conclusions.
- Mostly used tools for interpretation of a data are
  - Ratio
  - Percentage
  - Rate
  - Average

11.2. Types of Data Interpretation:

The numerical data pertaining to any event can be presented by any one or more of the following methods.

1. Tables
2. Line Graphs
3. Bar Graphs or Bar Charts
4. Pie Charts or Circle Graphs

11.2.1. Tables

It is the systematic presentation of data in tabular form to understand the given information and to make clear the problem in a certain field of study. It has six elements namely:

- **Title:** It is the heading of the table.
- **Stule:** It is the section of the table containing row headings
- **Column Captions:** It is the heading of each column
- **Body:** It consists the numerical figures
- **Footnotes:** It is for further explanation of the table
- **Source:** It is the authority of the data

Example: Study the following table and answer the questions given below it.

Annual Income of Five Schools

Figures in '00 rupees

Sources of Income	School A	School B	School C	School D	School E
Tuition Fee	120	60	210	90	120
Term Fee	24	12	45	24	30
Donations	54	21	60	51	60
Funds	60	54	120	42	55
Miscellaneous	12	3	15	3	15
Total	270	150	450	210	280

Example: The income by way of donation to school D is what per-cent of its miscellaneous?

Solution: Required percentage $= \frac{5100}{300} = 27\%$

11.2.2. Line Graph

A line graph indicates the variation of a quantity w.r.t two parameters calibrated on X and Y-axis respectively.

Note:

1. Any part of the line graph parallel to X-axis represents no change in the value of Y parameter w.r.t the value of X parameter.
2. The steepest or maximum part of the line graph indicates maximum percentage change of the value during the two consecutive period in which the related part lies.
3. If the steepest part is a rise slope, then it is the highest percentage growth.
4. If the steepest part is a decline slope, it will represent a maximum percentage fall of the value calibrated in the other axis.

11.2.3. Bar Graph

Bar graphs are diagrammatic representation of a discrete data.

Types of Bar Graphs:

- **Simple Bar Graphs:** A simple bar graph relates to only one variable. The values of the variables may relate to different years or different terms.
- **Sub-divided Bar Graph:** It is used to represent various parts of sub-classes of total magnitude of the given variable.
- **Multiple Bar Graphs:** In this type, two or more bars are constructed adjoining each other, to represent either different components of a total or to show multiple variables.

11.2.4. Pie Chart

In this method of representation, the total quantity is distributed over a total angle of 360° which is one complete circle or pie.

❐❐❐

12 DATA SUFFICIENCY

12.1. Introduction

Data sufficiency questions are designed to measure your ability to analyze a quantities problem, recognize which given information is relevant, and determine at what point there is sufficient information to solve a problem. In these questions, you are to classify each problem according to the five or four fixed answer choice, rather than find a solution to the problem.

Each Data sufficiency question consists of a question, often accompanied by some initial information, and two statements, labeled (1) and (2), which contain additional information. You must decide whether the information in each statement is sufficient to answer the question or- if neither statement provides enough information –whether the information in the two statements together is sufficient. It is also possible that the statements in combination do not give enough information answer the question.

Begin by reading the initial information and the question carefully. Next, consider the first statement. Does the information provided by the first statement is sufficient to answer the question? Go on the statement. Try to ignore the information given in the first statement when you consider the second statement. Now you should be able to say, for each statement, whether it is sufficient to determine the answer.

Next, consider the two statements in tandem. Do they, together, enable you to answer the question?

Give our answers as per the following statements

A Statement (1) alone is sufficient but
Statement (2) alone is not sufficient

B Statement (2) alone is sufficient but
Statement (1) alone is not sufficient

C Both statements together are sufficient
but neither statement alone is sufficient

D Each statement alone is sufficient

E Both statement together are still not sufficient.

❐❐❐

OUR
YOUTUBE CHANNELS

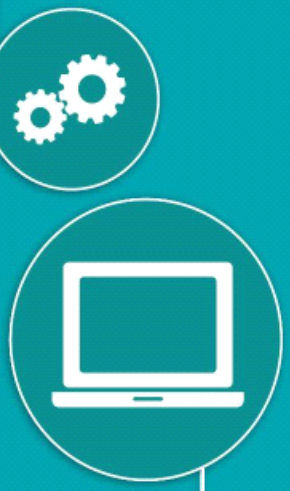
GATE WALLAH
EE, EC & CS

GATE Wallah

GATE WALLAH
ME, CE & XE

GATE Wallah (English)

ACCESS QUALITY CONTENT
For Free



Thank you