U18PCCS402 - Design & Analysis of Algorithm

Unit I

1. Introduction:

→ An algorithm is a finite set of instruction that if followed, accomplishes a particular task.

→ An algorithm is a sequence of steps to solve a problem.

→ Algorithms are the building blocks of computer programs.

⇒ Why do we need to study algorithm?

→ It tells the computer exactly what it should do in order to accomplish a task.

→ It is the process of finding the best way to solve a problem.

⇒ Advantages of algorithm:

→ It is a stepwise representation of a solution to a given problem, which makes it easy to understand.

→ An algorithm uses a definite procedure.

Scanned by CamScanner

→ It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.

→ Every step in an algorithm has its own logical sequence so it is easy to debug.

→ By using algorithm, the problem is broken down into smaller piece or steps, hence it is easier for programmer to convert it into an actual program.

Why design and analysis of algorithm?

→ Designing & analysis of algorithm is necessary before writing the program code, as the algorithm explains the logic even before the code is developed.

→ We design & analyse them for time complexity, space complexity, efficiency, etc.

→ If you are going to develop programs, you need to have knowledge of how to design and analyse algorithms to make an effective algorithm yourself.

Scanned by CamScanner

1.1 Characteristics of algorithm:

1. Input: An algorithm has zero or more inputs, ie quantities which are given to it initially before the algorithm begins.
2. Output: An algorithm has one or more outputs. ie quantities which have a specified relation to the input.
3. Finiteness: An algorithm must always terminate after a finite number of steps.
4. Definiteness: Each step of an algorithm must be precisely defined, the actions to be carried out must be rigorously and unambiguously specified for each case.
5. Effectiveness: Every instruction must be very basic. It is not enough that each operation be definite, it must be feasible. It should produce successfull result.

Scanned by CamScanner

2. Analysis of algorithm:

→ The term analysis of algorithms was coined by Donald Knuth.

→ Analysis of algorithms is the determination of the amount of time & space resources required to execute it.

→ The efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps, known as time complexity or volume of memory known as space complexity.

Need for analysis:

→ Algorithms are often quite different from one another, though the objective of these algorithms are same.

→ For example, we know that a set of numbers can be sorted using different algorithms.

→ Number of comparisons performed by one algorithm may vary with others for the same input.

→ Hence, time complexity of those algorithms may differ.

→ At the same time, we need to calculate the memory space required by each algorithm.

Scanned by CamScanner

→ Analysis of algorithm is the process of analyzing the problem solving capability of the algorithm in terms of the time and size required.

→ The main concern of analysis of algorithms is the required time or performance.

→ We perform the following types of analysis

Worst case: The maximum number of steps taken on any instance of size a.

Best case: The minimum number of steps taken on any instance of size a.

Average case: An average number of steps taken on any instance of size a.

2.1. Asymptotic analysis of complexity bounds:

→ When we study algorithm, we are interested in characterizing them according to their efficiency.

→ Usually we are interested in the order of growth of running time of an algorithm & called as asymptotic running time

Scanned by CamScanner

→ Asymptotic notation gives us a method for classifying functions according to their rate of growth.

Asymptotic Notations:

⇒ Big Oh Notation - O.
⇒ Omega Notation - Ω
⇒ Theta Notation - θ.

Big Oh Notation - O :

→ Big oh Notation is denoted by 'O'.

→ It is a method of representing the upper bound of algorithm's running time.

→ Using Big oh notation, we can provide longest amount of time taken by the algorithm to complete.

Definition :

Let $f(n)$ and $g(n)$ be the two non negative function

Let n_0 and c constant are two integers such that n_0 denotes some value of input & $n > n_0$. c is some constant $c > 0$.

$$f(n) \leq c * g(n)$$

$$f(n) \in O(g(n))$$

$\therefore f(n)$ is big oh of $g(n)$.

Scanned by CamScanner

∴ $f(n)$ is less than $g(n)$, if $g(n)$ is multiple of some constant c.

→ The value of $f(n)$ always lies on or below $cg(n)$.

→ n_0, c – always +ve constant.

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

examples:

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

$f(n) = n^2+n \qquad g(n) = n^3$

$n \geq 1, \quad n \leq n^3$

$n^2 \leq n^3$

$n^2+n \leq n^3$

$n^2+n = O(n^3) \quad n_0 = 1, \; c = 2.$

2. $f(n) = 2n+2, \quad g(n) = n^2.$

check $f(n) \leq c\,g(n).$

$n=1, \quad f(n) = 2(1)+2 = 4, \qquad g(n) = 1^2$

$n=2, \quad f(n) = 2(2)+2 = 6, \qquad g(n) = 4$

$n=3, \quad f(n) = 2(3)+2 = 8, \qquad g(n) = 9.$

$\therefore f(n) \leq g(n)$

Scanned by CamScanner

Omega Notation :

→ It is denoted by Ω.

→ It is used to represent the lower bound of algorithm's running time

→ Using Ω we can denote the shortest amount of time taken by an algorithm.

Definition:

A function $f(n)$ is said to be $\Omega(g(n))$ if $f(n)$ is bounded below by some positive constant multiple of $g(n)$, such that

$$f(n) \geq c\,g(n) \qquad n \geq n_0.$$

$$\therefore f(n) \in \Omega(g(n))$$

→ The value of $f(n)$ always lie on or above $g(n)$.

→ n_0 & c – +ve constant

→ As n value increases, $f(n)$ grows no slower than $g(n)$

examples:

1. $f(n) = 2n^2+5$ $\quad g(n) = 7n$

$n = 0$, $f(n) = 2(0)+5 = 5$, $g(n) = 7(0) = 0$

$n = 1$, $f(n) = 2(1)+5 = 7$, $g(n) = 7(1) = 7$.

$f(n) = g(n)$

Scanned by CamScanner

$n=2$, $f(n) = 2(2)^2+5 = 13$, $g(n) = 7(2) = 14$

$n=3$, $f(n) = 2(3)^2+5 = 23$, $g(n) = 7(3) = 21$

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

$$f(n) \geq c\, g(n)$$

$$\therefore 2n^2+5 \in \Omega(n).$$

2. $5n^3 + 4n = \Omega(n^2)$ verify.

$f(n) = 5n^3+4n$ $\quad g(n) = n^2$

$n=1.$ $\quad 5(1)+4 = 9.$ $\quad g(n)=1.$

$f(n) \geq c\,g(n)$ $\quad c=2,\ n_0=0,\ n=1.$

Theta Notation:

→ It is denoted as θ.

By this method, running time is found between upper and lower bound.

Definition:

$f(n)$ & $g(n)$ be non negative functions & c_1, c_2 are two positive constant.

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$\therefore f(n) \in \theta(g(n)).$$

Scanned by CamScanner

→ $f(n)$ always lie between $c_1 g(n)$ & $c_2 g(n)$.

→ As n increases, $f(n)$ grows at same rate as $g(n)$.

examples:

1. $f(n) = 2n+8$, $g(n) = 7n$, $n \geq 2$, $f(n) = \Theta g(n)$

$5n < 2n+8 < 7n$

$c_1 = 5$, $c_2 = 7$ $n_0 = 2$.

2. $n^3 + 10^6 n^2 = \Theta(n^3)$

$f(n) = n^3 + 10^6 n^2$ $g(n) = n^3$

$n^3 \leq n^3 + 10^6 n^2 \leq 10^7 n^3$

$c_1 = 1$, $c_2 = 10^7$, $n_0 = 1$.

3. Performance measurements of algorithm

→ By measuring the performance of an algorithm we can determine which algorithm is better than other.

→ Performance measurement is concerned with obtaining the space and time requirements of a particular algorithm.

→ Performance of algorithm is usually represented by Big Oh notation (O)

Scanned by CamScanner

→ These quantities depend on the compiler and options used as well as on which the algorithm is running.

3.1. Order of growth:

→ Measuring the performance of an algorithm in relation with the input size 'n' is called order of growth.

n	log n	n log n	n^2	2^n
1	0	0	1	2
2	1	2	4	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65536
32	5	160	1024	4294967296

→ It is clear that logarithmic function is the slowest growing function & Exponential function 2^n is the fastest function!

3.2. Properties of Big Oh:

→ Following are some important properties of big oh notations.

→ If there are two functions $f_1(n)$ & $f_2(n)$ such that $f_1(n) = O(g_1(n))$ & $f_2(n) = O(g_2(n))$, then

$$f_1(n) + f_2(n) = \max\{(O(g_1(n))), (O(g_2(n)))\}$$

Scanned by CamScanner

→ If there are two functions $f_1(n)$ & $f_2(n)$ such that $f_1(n) = O(g_1(n))$ & $f_2(n) = O(g_2(n))$. then

$$f_1(n) * f_2(n) = O(g_1(n)) * (g_2(n)).$$

→ If there exists a function f_1 such that $f_1 = f_2 * c$. c is constant, then f_1 and f_2 are equivalent.

$$O(f_1 + f_2) = O(f_1) = O(f_2).$$

→ If $f(n) = O(g(n))$ & $g(n) = O(h(n))$,

$$f(n) = O(h(n)).$$

→ In a polynomial, the highest power term dominates other terms.

For example, $3n^3 + 2n^2 + 10$,

its time complexity is $O(n^3)$.

→ Any constant value leads to $O(1)$ time complexity, ie if $f(n) = c$, then $\in O(1)$ time complexity.

3.3. Basic Efficiency classes:

Name of efficiency class	order of growth	Description	Example
constant	1	As the input size grows then we get constant running time	Scanning array elements.
Logarithmic	log n	When we get logarithmic running time then it is sure that the algorithm does not consider all its inputs. The	

Scanned by CamScanner

		problem is divided into smaller parts on each iteration	Perform binary search operation.
Linear	n	The running time of algorithm depends on the input size n.	Performing sequential search operation.
N logn	n logn	Some instance of input is considered for the list of size n.	Sorting the elements using merge sort or quick sort.
Quadratic	n^2	When the algorithm has two nested loop then this type of efficiency occurs.	Scanning matrix elements.
cube	n^3	When the algorithm has three nested loops then this type of efficiency occurs.	Performing matrix multiplication.
Exponential	2^n	When the algorithm has very faster rate of growth then this type of efficiency occurs	Generating all subsets of n elements.
Factorial	n!	When the algorithm is computing all the permutations then this type of efficiency occurs.	Generating all permutations

Scanned by CamScanner

Examples:

Linear:

```
for (i=0; i<n; i++)
    m+ = i;
```

Time complexity = $O(n)$.

Quadratic:

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        sum[i]+= entry[i][j];
```

Time complexity = $O(n^2)$.

cubic:

```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        for (k=1; k<=n; k++)
            printf ("AAA");
```

Time complexity = $O(n^3)$.

Logarithmic:

```
for (i=1; i<n; i= i*2)
    printf ("AAA");
```

Time complexity = $O(\log n)$

Linear Logarithmic: (N log n)

```
for (i=1; i<n ; i= i*2)
    for (j=1; j<=n; j++)
        printf ("AAA");
```

Time complexity = $O(n \log n)$.

Scanned by CamScanner

Performance of program:

→ The performance of a program is measured based on the amount of computer memory and time needed to run a programs.

→ There are two approaches which are used to measure performance of a program.

⇒ Performance analysis (Analytical method)

⇒ Performance measurement (Experimental method)

Time: Amount of time required by an algorithm to execute.

Space: Amount of storage required by an algorithm.

4. Time & space trade off:

4.1. Time complexity:

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

→ The time complexity can be measured by measuring the time taken by the program when it is executed.

Scanned by CamScanner

Reason for estimating the time complexity:

→ We need real time response for many applications ie a faster execution of program is required for many applications.

→ If the time complexity is estimated beforehand, then modifications can be done to the program to improve the performance.

→ It is used to specify the upper limit for time of execution for some programs. It is used to avoid infinite loops.

Time complexity $T(P) = c + T_P$.

c ↑ compile time

T_P ↑ execution time.

⇒ c - compile time - time taken by the compiler to produce intermediate code.

⇒ T_P - execution time or run time. Time taken for execution of program.

→ compile time does not depends on instance characteristics

→ Run time depends on the instance characteristics.

Scanned by CamScanner

example:
frequency - total no. of times executed.

Statement	Frequency	total steps
Algorithm sum(a,n)	-	0
{	-	0
s := 0.0;	1	1
for i := 1 to n do	n+1	n+1
s := s + a[i]	n	n
return s;	1	1
}	-	0
	total.	2n+3

4.2. Space Complexity:

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

Reason for estimating the space complexity:

→ We should know in advance, whether or not sufficient memory is present in the computer.

→ If this is not known & the program will execute directly & there is possibility that the program may consume more memory.

Scanned by CamScanner

→ This leads to insufficient memory & system may crash.

→ In multiuser system, we prefer the programs of lesser size, because multiple copies of the program run when multiple user access the system. Hence if the program occupies less space during execution, then more no. of users can be accomodated.

→ The space needed by each of these algorithms is seen to be the sum of the following components.

fixed part: ⇒ It is independent of the characteristics (eg. no, size) of input & output.

⇒ It includes instruction space (space for code) space for simple variables & fixed size components variables, space for constant.

Variable part: ⇒ Which the size is dependent on instance characteristics, the recursion stack.

$$S(P) = c + S_p$$

space complexity of program P.

c ↑ fixed / constant space

S_p ↑ variable space.

Scanned by CamScanner

example:

```
int square (int a)
{
return a*a;
}
```

int a = 2 bytes
return = 2 bytes
4 bytes
constant space complexity.

```
int sum (int A[], int n)
{
int sum = 0, i;
for (i=0; i<n; i++)
    sum = sum + A[i];
return sum;
}
```

n = 2
a[] = n*2
sum = 2
i = 2
return = 2
2n+8
linear space complexity.

5. Analysis of best case, worst case and average case:

Linear search:

→ It is a method for finding a target value within a list.

→ It sequentially checks each element of the list for the target value until a match is found.

Scanned by CamScanner

Implementation of linear search:

```
// Linearly search x in arr[], if x is
present it'll return the index value
// Otherwise return -1.

int search (int arr[], int n, int x)
{                  ↑ array     ↑ size     ↑ search element
int i;
for (i=0; i<n; i++)
{
   if (arr[i] == x)
      return i;
}
return -1;
}
```

example:

0	1	2	3	4	5	6
2	6	17	1	4	9	11

Find 6 is present in the array.
If present - returns element (index)
If not present - returns -1.

Worst case Analysis:

→ We calculate upper bound on running time of an algorithm.

→ This causes maximum no. of operations to be executed.

Scanned by CamScanner

→ For linear search, the worst case happens when the element to be searched is not present in the array or the search element is present at n^{th} location.

→ For these cases the search() function compares with all the elements of array arr[] one by one.

→ The worst case time complexity of linear search is $O(n)$.

Best case Analysis:

→ We calculate the lower bound on running time of an algorithm.

→ This causes minimum no. of operations to be executed.

→ In linear search, the best case occurs when x is present at the first location.

→ The no. of operations in the best case is constant.

→ Time complexity is $\Omega(1)$.

Average case analysis:

→ Average case analysis gives information about the behaviour of an algorithm on random input.

→ Let the algorithm is for linear search.

Scanned by CamScanner

→ Let p be the probability of getting successfull search.

→ n be the total no. of elements in the list.

→ The first match of the element will occur at i^{th} location.

→ Hence the probability of occuring first match is p/n for every i^{th} element.

→ probability of getting unsuccessfull search is $(1-p)$

$$\theta(n) = \text{Probability of successfull search} + \text{Probability of unsuccessfull search}$$

$$= \left(1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n}\right) + n(1-p)$$

$$= \frac{p}{n}\left(1 + 2 + \cdots + i + \cdots + n\right) + n(1-p)$$

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

→ This is for general formula for computing average case time complexity.

→ Suppose if $p = 0$, (the element is not found)

$$\theta(n) = 0\left(\frac{n+1}{2}\right)n + n(1-0)$$

$$\boxed{\theta(n) = n.}$$

Scanned by CamScanner

→ If $p=1$, we get a successful search.

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

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

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

6. Types of problems:

6.1 Sorting
6.2 Searching
6.3 String Processing
6.4 Graph problems.
6.5 combinatorial problems
6.6 Geometric problems
6.7 Numerical problems.

6.1. Sorting:

→ The process of arranging the given items in ascending or descending order.

→ Sorting is the operation of arranging the record of a table according to the key value of the each record.

Properties:

Stable: It preserves the relative order of any two equal elements in its input.

Inplace: An algorithm is said to be in place if it doesnot require extra memory.

Scanned by CamScanner

classification:

Internal sorting: data sorting process take place within the main memory.

→ Bubble sort
→ Selection sort
→ Quick sort.

External sorting: data sorting take place in the external memory such as tape or disk.

→ Merge sort

Advantages:

→ Easy to retrieve the information in an application.

→ Easy to search an element in the list.

6.2. Searching

→ Finding a search key in given set of items.

Sequential search:

→ The algorithm starts its search with the first available record and proceeds to the next available record repeatedly, until the required data item to be searched ~~its~~ is found.

Advantages:

→ It is a straight forward technique.
→ It doesnot require sorted order.
→ Searching operation is simple.

Scanned by CamScanner

Disadvantages:

→ It takes more time.

→ It cannot be suitable for large no. of records.

Binary search:

→ To search for an element in the list, the binary search algorithm split the list and locate the middle element of the list.

→ It requires the list to be sorted in ascending order.

→ First compare middle key 'K' with a given key 'S'.

→ If (K_1 == S) then the element is found.

Advantages:

→ Less time consumption

→ The processing speed is fast.

→ The no. of iteration is less, it take n/2 iterations.

→ Best case analysis is easy.

→ For successful search only one element comparison is needed.

→ For unsuccessful search (log n) comparisons are needed.

Scanned by CamScanner

6.3. String processing:

String: It is a sequence of characters from an alphabet.

Types:

→ **Text string**: It contains letters, numbers, special characters.

→ **Bit string**: Bit string contain zeros and ones.

→ **Gene sequence**: It can be modeled by string of characters from the 4 character alphabet {A, C, G, T}.

String matching:

→ The process of searching for an occurrence of word in a text is called string matching or pattern matching.

String matching algorithm:

→ simple pattern matching algorithm

→ Pattern matching using morris Pratt algorithm.

String operation:

→ string concatenation

→ finding string length

→ substring operations

→ string copy

→ string comparison.

Scanned by CamScanner

6.4 Graph algorithm:

Graph $G = \{V, E\}$ contain collection of nodes & edges.

Graph Traversal:

→ It is the process of determining the possible way for visiting the nodes of a graph.

⇒ Depth first search - DFS

⇒ Breadth first search - BFS

Shortest path:

→ It is used to find shortest distance between source node to all the other nodes in the graph.

→ It is used to find best route between 2 cities.

Topological sorting:

→ Topological sort is an ordering of vertices in a directed acyclic graph, such that if there is path from v_i to v_j then v_j appears after v_i in the ordering.

Travelling salesman problem:

→ It is the problem of finding the shortest tour through n cities that visits every city exactly once.

Scanned by CamScanner

Graph coloring problem:

It is the problem of assigning the smallest no. of colours to vertices of a graph so that no two adjacent vertices are the same colour.

Advantages:

→ Graph can be used for modeling a wide variety of real life applications like transportation, communication networks, project scheduling & games.

→ It is used to find maximum no. of links between web pages.

6.5. Combinatorial problems:

→ These are problems that used (explicitly or implicitly) to find a combinatorial object such as permutation, a combination or a subset that satisfies certain constraints & has some desired property. (maximise or minimizes cost)

→ combinatorial problems are difficult problems.

6.6. Geometric problems:

→ Problems deals with geometric objects such as points, lines & polygons.

Scanned by CamScanner

Advantages: It is used in computer graphics, robotics & tomography.

Geometric algorithm:

closest pair problem: Given n points in a plan, it find the closest pair among them.

convex hull problem: It find the smallest convex polygon that would include all the points of a given set.

6.7. Numerical problems:

Problems that involve mathematical objects of continuous nature solving computing & system of equations, function definite integrals, evaluating & so on.

Advantages:

→ It is used in many scientific & engineering applications.

→ It is also used in business applications.

Disadvantages:

→ It can be solved approximately.

→ It manipulates real numbers, which can be represented in computer only approximately.

→ Large numbers of arithmetic operations performed approximately.

Scanned by CamScanner

7. Basics of Data structure:

→ Data structure refers to a scheme for organizing data or an arrangement of data in computer's memory in such a way that it could make the data quickly available to the processor for required calculations.

Classification:

Linear data structure:

⇒ Arrays
⇒ Linked list
⇒ stacks
⇒ queue.

Non linear data structure:

⇒ Trees
⇒ Graphs
⇒ Heaps.

Arrays: An array is a collection of data items having similar data types.

Linked list: Linked list is a collection of nodes, where each node has a data element and a reference to the next node in the sequence.

Stack: It is a first in last out data structure where the element that is added first will be deleted last.

Scanned by CamScanner

Queue: It is a first in first out data structure where the elements is added first will be deleted first.

Tree: Tree is a collection of nodes where these nodes are arranged hierarchically & form a parent - child relationship.

Graph: Graph is a collection of a finite number of vertices & edges. Edges connect the vertices & represent the relationship between the vertices that connect these vertices.

8. **Analysis of non recursive algorithm**:

Recursion: The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called recursive function.

Non recursion: A non recursive technique is that doesn't use recursion. ex: Insertion sort.

Non recursive algorithm

It is executed only once to solve the problem.

Scanned by CamScanner

8.1 General plan for analyzing efficiency of non recursive algorithm:

→ Decide on a parameter indicating an input's size.

→ Identify the algorithm's basic operation.

→ check whether the no. of times the basic operation is executed depend only on the size of an input.

→ Sum = no. of time basic operation is executed.

→ Using standard formulas & rules for sum manipulation, either find a closed form formula for the count establish its order.

example:

8.2 Finding the value of largest element in a list of n numbers.

Algorithm:

```
Algorithm max element (A [0 .... n-1])
//Determines the value of the largest
//element in a given array
// An array A[0... n-1] of real numbers.
//output - The value of the largest
element in A.

maxval = A[0]
    for i = (1 to n-1) do
        if (A[i] > maxval)
            maxval = A[i]
return maxval
```

$T_c = O(n)$.

Scanned by CamScanner

9. Analysis of recursive algorithm through recurrence relations:

→ Recurrence equation is an equation that defines a sequence recursively

$$\boxed{T(n) = T(n-1) + n} \quad , \quad n > 0.$$

$\boxed{T(0) = 0}$, initial condition.

→ To solve recurrence equation

1) Substitution method
2) Master's method
3) Recursive tree method.

9.1 Substitution method:

→ Forward substitution
→ Backward substitution

Forward substitution:

→ It makes use of initial condition in the initial term & the value for next term is generated.

→ The process is continued untill some formula is guessed.

examples:

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

$T(0) = 0.$

$n=1, \quad T(1) = T(1-1) + 1$

$= T(0) + 1$

$\boxed{T(1) = 1}$

Scanned by CamScanner

$n = 2, \quad T(2) = T(2-1) + 2$

$= T(1) + 2$

$= 1 + 2$

$\boxed{T(2) = 3}$

$n = 3, \quad T(3) = T(3-1) + 2$

$= T(2) + 2$

$= 3 + 3$

$\boxed{T(3) = 6.}$

$\therefore 1, 3, 6, \ldots n.$

$$T(n) = \frac{n(n+1)(n+2)}{6}$$

$$= n^3 + n^2.$$

$$\boxed{T(n) = O(n^3).}$$

Backward substitution:

→ This method uses backward values in order to derive some formula.

example:

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

$n = n-1, \quad T(n-1) = T(n-1-1) + n-1$

$$\boxed{T(n-1) = T(n-2) + n-1.}$$

$n = n-2.$

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

$$\boxed{T(n-2) = T(n-3) + n-2.}$$

Scanned by CamScanner

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

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

$= T(n-3) + n-2 + 2n-1$

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

$$\boxed{T(n) = T(n-3) + (n-2) + (n-1) + n}$$

$$\boxed{T(n) = T(n-k) + (n-k+1) + (n-k+2) + \cdots + n.}$$

$\boxed{k = n.}$

$T(n) = T(0) + T(1) + T(2) + \cdots + n.$

$= 0 + 1 + 2 + \cdots + n.$

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

$$\boxed{T(n) = \frac{n^2+n}{2}}$$

$$\boxed{T(n) = O(n^2)}$$

2. $T(n) = 2T(n/2) + n.$ $T(1) = 1.$

$n = n/2$

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

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

Scanned by CamScanner

$n = n/4$

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

$$= 2T(n/8) + n/4$$

$n = n/8$

$$T(n/8) = 2T\left(\frac{n/8}{2}\right) + \frac{n}{8}$$

$$= 2T(n/16) + \frac{n}{8}$$

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

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

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

$$= 4\left[2T\left(\frac{n}{8} + \frac{n}{4}\right)\right] + 2n.$$

$$= 8T\left(\frac{n}{8} + n\right) + 2n$$

$$= 8T\left(\frac{n}{8}\right) + n + 2n$$

$$T(n) = 8T\left(\frac{n}{8}\right) + 3n$$

$$= 8\left(2T\left(\frac{n}{16}\right) + \frac{n}{8}\right] + 3n.$$

$$= 8\left[2T\left(\frac{n}{16}\right) + \frac{n}{8}\right] + 3n.$$

Scanned by CamScanner

$$= 16\ T\left(\frac{n}{16}\right) + n + 3n$$

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

$$\boxed{\begin{array}{l} k = \log n. \\ 2^k = n. \end{array}}$$

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

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

$$= n\ T(1) + n \log n.$$

$$T(n) = n + n \log n.$$

$$\boxed{T(n) = O(n \log n).}$$

9.2 Master's Method:

I) method:

$$T(n) = a\ T(n/b) + F(n).$$

1. $T(n) = \Theta(n^d)$, $a < b^d$.
2. $T(n) = \Theta(n^d \log n)$, $a = b^d$.
3. $T(n) = \Theta(n^{\log_b a})$, $a > b^d$.

Scanned by CamScanner

1. $T(n) = 4\,T(n/2) + n$

$a = 4 \qquad b = 2 \qquad d = 1 \qquad f(n) = n^1$

$4 > 2^1$

$a > b^d$

$T(n) = \theta\left(n^{\log_2 4}\right)$

$T(n) = \theta(n^2)$

$\log_2 4 = \log_2 2^2 = 2(1) = 2.$

$\therefore \boxed{T(n) = \theta(n^2).}$

2. $T(n) = 2\,T(n/4) + \sqrt{n} + 4^2$

$a = 2 \qquad b = 4 \qquad d = 1/2 \qquad f(n) = \sqrt{n} = n^{1/2}$

$2 = 4^{1/2}$

$2 = 2^{2\,1/2}$

$2 = 2.$

$a = b^d$

$\therefore \boxed{T(n) = \theta(\sqrt{n}\,\log n)}$

II method:

$T(n) = a\,T(n/b) + f(n).$

case i: $f(n) = O\left(n^{\log_b a}\right).$

$f(n) < n^{\log_b a}, \qquad T(n) = \theta\left(n^{\log_b a}\right).$

Scanned by CamScanner

case II:

$$f(n) = \Theta(n^{\log_b a} \log n)$$

$$f(n) = n^{\log_b a}$$

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

case III:

$$f(n) = \Omega(n^{\log_b a})$$

$$f(n) > n^{\log_b a}$$

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

Steps:

→ Find a, b, f(n) from given recurrence relation.

→ Find $n^{\log_b a}$

→ compare $f(n)$ & $n^{\log_b a}$.

1) $T(n) = 2T(n/2) + n.$

$a = 2$ $b = 2$ $f(n) = n$

$$n^{\log_b a} = n^{\log_2 2} = n^1 = n.$$

$$f(n) = n^{\log_b a}.$$

$$\boxed{T(n) = \Theta(n \log n).}$$

2. $T(n) = 9T(n/3) + n.$

$a = 9$ $b = 3$ $f(n) = n$

$$n^{\log_b a} = n^{\log_3 9} = n^{\log_3 3^2}$$

$$= n^{2(1)}$$

$$= n^2.$$

$T(n) = \Theta(n^{\log_b a})$ $n < n^2$ $\therefore$ $\boxed{T(n) = \Theta(n^2)}$

Scanned by CamScanner

9.3. Recursion tree method:

→ In recursion tree method each node represents the cost of single sub problem.

→ The recursion tree method helps to make a good guess of time complexity.

1. $T(n) = 2\,T(n/2) + 4n$.

recursive part ; root node.

height $= h$
level $= 2^i$

level	nodes	Tree	Rowsum.
$T(n)$	1	$4n$	$4n$
$T(n/2)$	2^1	$4\,n/2$ $\quad$ $4\,n/2$	$4n$.
$T(n/4)$	2^2	$4\frac{n}{4}$ $\quad$ $4\frac{n}{4}$ $\quad$ $4\frac{n}{4}$ $\quad$ $4\frac{n}{4}$	$4n$
$T(n/8)$	2^3	$4\frac{n}{8}$ $\quad$ $4\frac{n}{8}$ $\quad$ $4n/8$ $\quad$ $4n/8$	$4n$.

$$\sum_{i=0}^{\log_2 n} 4n$$

$$= 4n \sum_{i=0}^{\log_2 n} (1)$$

$$= 4n\left[1 + \cdots + \log_2 n\right]$$

$$= 4n + 4n \log_2 n.$$

$$\boxed{\begin{aligned} \frac{n}{2^i} &= 1 \\ n &= 2^i \\ \log_2 n &= i \end{aligned}}$$

$$\boxed{T(n) = O(n \log_2 n)}$$

Scanned by CamScanner

2. $T(n) = T(n/3) + T(2n/3) + n.$

levels	nodes	Tree	Rowsum
$T(n)$	1	n	n
	2^1	$n/3 \quad 2n/3$	$n.$
	2^2	$n/9 \quad 2n/9 \quad 2n/9 \quad 4n/9$	n
	$\vdots$		
	2^i		$n.$

$$\frac{n}{2^i} = 1$$

$$n = 2^i$$

$$\log_2 n = i$$

$$\sum_{i=0}^{\log_2 n} n$$

$$= n \sum_{i=0}^{\log_2 n} (1)$$

$$= n\left[1 + \log_2 n\right]$$

$$= n + n\log_2 n.$$

$$\boxed{T(n) = O(n\log_2 n).}$$

Scanned by CamScanner

Important questions Unit I.

2m:

1. What is an algorithm?
2. What is the need for analysing algorithm?
3. Explain Big Oh Notation.
4. Explain Omega Notation.
5. Explain Theta Notation.
6. What is time & space complexity?
7. Explain linear and non linear data structure
8. Explain Master's theorem.
9. What is recursion?
10. What is asymptotic Notation?
11. Explain analysis of algorithm.

6m:

1. What are the characteristics of algorithm?
2. Explain asymptotic analysis of complexity bounds with example.
3. Explain the order of growth of asymptotic notations.
4. Explain basic efficiency classes with example.
5. Analysis of non-recursive algorithms
6. problems on substitution method & master's theorem.

Scanned by CamScanner

2. 10 m:

1. Explain time & space complexity with examples.
2. Explain in detail about the analysis of worst case, best case & average case
3. Explain types of problems in detail.
4. Problems on recursive tree method

Scanned by CamScanner

1. Brute force Approach:

→ It is a straight forward approach to solve a problem based on problem's statement and definition of the concepts involved.

→ It is considered as one of the easiest approach.

→ It is useful for solving small size instance of problem.

→ It is considered as design strategy.

→ Brute force approach doesnot include any shortcuts to improve performance, but instead relies on sheer computing power to try all possibilities until the solution to a problem is found.

1.1 Greedy Algorithm:

→ The greedy approach suggests that constructing a solution through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is obtained.

→ problems have n inputs and require us to obtain a subset that satisfies some constraints.

Scanned with CamScanner

→ Important terms in greedy algorithm or greedy technique

Feasible: Any subset that satisfies problem constraint is called feasible.

objective function: To find a feasible solution that either maximizes or minimizes a given objective function.

optimal solution: It has to be the best choice among all the feasible solution.

Irrevocable: once made, it cannot be changed on subsequent steps of the algorithm.

1.1.1) An activity selection problem:

→ There are n different activity are given with their starting and ending time.

→ select maximum no. of activity to solve by a single person.

→ sort the activity with their ending time.

→ find compatible activity and add to list.

Scanned with CamScanner

Algorithm:

```
Greedy-activity (S, f)
n ← length (S)
A ← {1}
J ← 1
for i ← 2 to n
    do if S_i ≥ f_J
        then A ← A ∪ {i}
             J ← i
Return A.
```

Time complexity.

sort - n

assigning - $n \log n$.

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

example:

1. Given 9 activity along with start and finish time.

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
S_i	1	2	4	1	5	8	9	11	13
f_j	3	5	7	8	9	10	11	14	16.

$A = \{A_1, A_3, A_6, A_8\}$

~~$n \leftarrow 9$~~
~~$A \leftarrow \{A_1\}$~~
~~$J \leftarrow 1$~~
~~$i = 2$~~
~~$S_2 \geq f_1$~~

Steps:

$n = 9$
$A = \{A_1\}$
$j = 1,\ i = 2.$
$S_2 \geq f_1$
$2 \geq 3$ ✗

$i = 3,\ j = 1.$
$\boxed{S_3 \geq f_1}$
$4 \geq 3$ ✓
Add A_3

$i = 4 \quad j = 3.$
$S_4 \geq f_3$
$1 \geq 7$ ✗

$i = 5 \quad j = 3$
$S_5 \geq f_3$
$5 \geq 7$ ✗

$i = 6 \quad j = 3$
$S_6 \geq f_3$
$8 \geq 7$ ✓
Add A_6.

$i = 7 \quad j = 6$
$S_7 \geq f_6$
$9 \geq 10$ ✗

$i = 8 \quad j = 6$
$S_8 \geq f_6$
$11 \geq 10$ ✓
Add A_8

$i = 9 \quad j = 8$
$S_9 \geq f_8$
$13 \geq 14$ ✗

Scanned with CamScanner

∴ The optimal solution is

$\{A_1, A_3, A_6, A_8\}$

1.1.2. Elements of greedy strategy:

→ A greedy algorithm obtains an optimal solution to a problem by making a sequence of choices.

→ For each decision point in the algorithm, the choice that seems best at the moment is chosen.

Steps involved in greedy method:

→ Determine the optimal substructure of the problem.

→ Develop a recursive solution

→ Prove that at any stage of recursion, one of the optimal choices is the greedy choice. Thus it is always safe to make the greedy choice.

→ Show that all but one of the subproblems induced by having made the greedy choice are empty.

→ Develop a recursive algorithm that implements the greedy strategy

→ Convert the recursive algorithm to an iterative algorithm.

Scanned with CamScanner

1.1.3. Huffman code

→ Huffman coding is a lossless data compression algorithm.

→ The idea is to assign variable length code to input characters, length of the assigned codes are based on the frequencies of corresponding characters.

→ The most frequent character gets the smallest code and the least frequent character gets the largest code.

→ The variable length codes assigned to input characters are 'prefix code' means the codes are assigned to one character is not prefix of code assigned to any other character.

→ This is how Huffman coding makes sure that there is no ambiguity when decoding the general bit stream.

→ There are two major parts in Huffman code.

⇒ Build a Huffman tree from input characters.

⇒ Traverse the Huffman tree and assign codes to characters

Scanned with CamScanner

Steps to build Huffman tree:

→ Create a leaf node for each unique character and build a min heap of all leaf nodes.

→ Extract two nodes with the minimum frequency from the min heap.

→ Create a new internal node with frequency equal to the sum of two nodes frequencies.

→ Repeat the steps #2 & #3 until the heap contains only one node.

→ The remaining node is the root node & the tree is complete.

example:

character	frequency.
a	5
b	9
c	12
d	13
e	16
f	45.

Scanned with CamScanner

a - 5
b - 9
c - 12
d - 13
e - 16
f - 45.

c - 12
d - 13
IN1 - 14
e - 16
f - 45.

4

IN1 - 14
e - 16
IN2 - 25
f - 45

IN2 - 25
IN3 - 30
f - 45

f - 45.
IN4 - 55.

f - 1
c - 000
d - 001
e - 011
a - 0100
b - 0100.

Scanned with CamScanner

Step 1:

~~c - 12~~
~~IN1 - 14~~
~~d -~~

c - 12
d - 13
IN1 - 14
e - 16
f = 45

Step 2:

IN1 - 14
e - 16
IN2 - 25
f - 45

Step 3:

IN1 - 14
IN2 - 25
IN3 - 30
IN4 - 55
f - 45

Step 4:

IN1 - 14
IN2 - 25
IN3 - 30
f - 45
IN4 - 55

Scanned with CamScanner

Step 5: Maintain the auxiliary array. While moving to the left child, write 0 to array, while moving to the right child, write 1 to array. Print the array when left node is encountered.

character	code word
f	0
c	100
d	101
a	1100
b	1101
e	111

Scanned with CamScanner

Algorithm:

Input: no. of characters with frequency count.

output: Huffman tree

Begin

Let Q be the priority queue,

Q = { initialize the priority queue with frequencies of all characters }

Repeat n-1 times

Create a new node Z

X = extract_min (Q)

Y = extract_min (Q)

Frequency (Z) = Frequency (X) + Frequency (X);

Insert (Z, Q)

End repeat

Return (extract_min (Q))

End.

Analysis of complete complexity:

For creating n leaves = $O(n)$

Turning H into heap using build heap = $O(n)$

for loop = n-1

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

$\therefore O(n) + O(n) + O(n \log n)$

$\therefore \boxed{O(n \log n)}$

Scanned with CamScanner

1.1.5. Task Scheduling problem:

→ Given array of jobs where every job has deadline and associated profit if the job is finished before deadline.

→ It is also given that every job takes single unit of time.

→ So the minimum possible deadline for any job is 1.

Steps:

→ Sort all jobs in decreasing order of profit.

→ Initialize the result sequence as first job in sorted jobs.

→ Do the following for remaining n−1 jobs.

⇒ If the current job can fit in the current result sequence without missing the deadline, add current job to the result, else ignore the current job.

example:

1.

Job	A	B	C	D	E
Deadline	2	1	2	1	3
Profit	100	19	27	25	15

2.

Job	A1	A2	A3	A4	A5
dead line	2	1	3	2	1
Profit	60	100	20	40	20

1 2 2 3 3
100 60 40 20 20
A2 A1 A4 A5 A3

Scanned with CamScanner

Sort all jobs in decreasing order of profit.

Job	A	B	C	D	E
Deadline	2	2	1	1	3
Profit	100	27	25	19	15

∴ Jobs A, C, E are scheduled with profit of (100+25+15) 140.

Algorithm:

```
for i=1 to n do
set K= min (dmax , Deadline(i))
while K>=1 do
        if timeslot [K] is empty then
            timeslot [K] = job [i]
        break;
    end if
    set K=K-1
  end while
end.
```

Analysis:

Time complexity = $O(n^2)$.

Scanned with CamScanner

6a.

step1: job A2

$i = 1$

$K = \min(dmax, deadline(i))$

$K = \min(3, deadline(1))$

$K = \min(3, 1)$

$K = 1.$

timeslot	1	2	3
	A2	A1	A3

$K >= 1$ ✓

timeslot [1] is empty

timeslot[1] = A2. (Job[1]).

step2: $i = 2.$ Job A1.

$i = 2.$

$K = \min(dmax, deadline(i))$

$= \min(3, deadline(2))$

$= \min(3, 2)$

$K = 2.$

$K >= 1$ $2 >= 1$ ✓

timeslot[2] is empty

timeslot [2] = Job[2]

timeslot[2] = A1.

step3:

$i = 3$ Job A4.

$K = \min(dmax, deadline(i))$

$= \min(3, deadline(3))$

$= \min(3, 2)$

$K = 2$

$K >= 1$

$2 >= 1$ ✓

timeslot [2] is empty ✗

Scanned with CamScanner

$k = k - 1$

$k = 2 - 1$

$\boxed{k = 1.}$

$i = 3$, ~~Job~~

$k >= 1$

$1 >= 1$ ✓.

timeslot [1] is empty ✗

$k = k - 1$

$i = 3, k = 0$

$k >= 1$ ✗

$0 >= 1$ ✗

so move to next job.

<u>step 4</u>: Job 3. $i = 4$.

$k = \min(d_{max}, \text{deadline}(i))$

$= \min(3, \text{deadline}(4))$

$k = \min(3, 3)$

$k = 3$

$k >= 1$

$3 >= 1$ ✓

timeslot [3] is empty ✓

timeslot [k] = job [4]

timeslot [3] = A_3.

∴ timeslot is full.

∴ The required job sequence is

$A_2 \rightarrow A_1 \rightarrow A_3$

Profit = 180.

Scanned with CamScanner

1.1.6. Knapsack problem:

→ In knapsack problem objects are given with corresponding profits and weights. Knapsack capacity is also given.

→ The main objective is to fill the knapsack with objects containing maximum profit and minimum weights.

⇒ n objects/items are given

⇒ w_i - weight of ith object ($w_i > 0$).

⇒ P_i - profit of ith object ($P_i > 0$)

⇒ capacity of the knapsack is W.

x_i of ith object $0 \le x_i \le 1$.

objective function:

$$\text{Maximize} \sum_{n=1}^{N} x_i P_i$$

problem constraint:

$$\sum_{n=1}^{N} x_i, W_i \le W.$$

example:

W = 60.

Object	A	B	C	D
profit	280	100	120	120
weights	40	10	20	24
P_i/w_i	7	10	6	5

Scanned with CamScanner

sort based on $\frac{P_i}{w_i}$.

	B	A	C	D
Profit	100	280	120	120
weight	10	40	20	24
Ratio $\frac{P_i}{w_i}$	10	7	6	5

$W = 60$.

Weight:

$10 + 40 + 20/20 = 60$

Profit.

$100 + 280 + 120 \times \frac{10}{20} = 440$.

Algorithm.

```
Knapsack (w[1....n], P[1...n], W)
for i = 1 to n
    do x[i] = 0
end for
Load = 0
i = 1
while ((Load < W) & (i <= n)) do
    if (wi + Load ≤ W) then
        Load = Load + wi
        X[i] = 1 // Entire object has been inserted
        i = i + 1
    else
        r = W - Load
        Load = Load + r/wi * wi
        X[i] = r/wi // 1/2 part of object
    end if
end while
return (X)
End.
```

Scanned with CamScanner

Steps:

$i=1$ $\quad i=2$ $\quad i=3$ $\quad i=4$

$x[1]=0$ $\quad x[2]=0$ $\quad x[3]=0$ $\quad x[4]=0$

Load = 0, 10,

$i = 1, 2$ 40

while ($\overset{10}{0} < 60$) & ($\overset{2}{i} <= 4$)

if ($10 + 0 \leq 60$)

Load = 10

$x[\underset{1}{i}] = 1$ // considering whole item 1.

~~Load = 10~~.

Load = 10

$i = 2$.

while ($40 < 60$) && ($2 <= 4$)

if ($10 + 40 \leq 60$)

Load = 50

$x[\underset{2}{i}] = 1$

$i = 3$

Load = 20.

while ($50 < 60$) & ($3 <= 4$)

if ($50 + 20 \leq 60$) ×

else $\quad 60 - 50$

$r = W - Load$ // $r = 10$

$Load = Load + w \times \frac{r}{w_i}$ $\quad$ ½. // $Load = Load + w \times \frac{w_i}{w_i}$

$= 50 + \frac{10}{20} \times w_i$ (20) $\quad$ $= 50 + 20 \times \frac{10}{20}$.

$= 50 + 10$

$x[i] = \frac{r}{w_i}$ // $\frac{10}{20}$. // ½ of the 3rd obj.

Analysis:

Time complexity = $O(n \log n)$.

Scanned with CamScanner

sort based on $\frac{P_i}{w_i}$

7a:

Profit	B 100	A 280	C 120	D 120
weight	10	40	20	24
Ratio $(\frac{P_i}{w_i})$	10	7	6	5.

Weight:

$$10 + 40 + \frac{20}{2} = 60.$$

Profit:

$$100 + 280 + 120 \times \frac{10}{20} = 440.$$

Algorithm:

```
Algorithm Bound (Cp, Cw, k)
//Cp is the current profit total,
//Cw is the current weight total.
//k is the index of the last removed
item and M = Knapsack capacity.
{
    b = Cp,  c = Cw.
    for i = (k+1) to n do
    {
        c = c + w[i]
        if (c < m) then b = b + p[i]
        else return b + (1 - (c-m)/w[i]) * p[i]
    }
    return b
}.
```

Scanned with CamScanner

example:

$n = 8$, $m = 110$,

$P = \{11, 21, 31, 33, 43, 53, 55, 65\}$ $\quad 0 \le x_i \le 1$.

$W = \{1, 11, 21, 23, 33, 43, 45, 55\}$

objects i	P_i	W_i	P_i/w_i	x_i	C_w	$m - C_w$	$P_i \times x_i$
1	11	1	11	1	1	109	11
2	21	11	1.909	1	12	98	21
3	31	21	1.476	1	433	~~6~~77	31
4	33	23	1.434	1	56	~~9~~54	33
5	43	33	1.303	1	89	21	43
6	53	43	1.232	21/43	110	0	25.88
7	55	45	1.222	0	110	–	
8	65	55	1.181	0	110	–	

$w = 110$.

objects $= \{1, 2, 3, 4, 5, 6\,(21/43)\}$.

profit $=$ ~~254.88~~.

$= 164.88$.

Scanned with CamScanner

1.2. Dynamic Programming:

→ Dynamic Programming is the most powerful design technique for solving optimization problems.

→ Dynamic Programming is used when the sub-problems are not independent, eg. When they share the same subproblems.

→ Dynamic Programming solves each subproblems just once and stores the result in a table, so that it can be repeatedly retrieved if needed again.

→ It is a Bottom up approach, we solve all possible small problems and then combine to obtain solutions for bigger problems.

→ Dynamic Programming works when a problem has the following features:

optimal substructure: If an optimal solution contains optimal sub-problems then a problem exhibits optimal substructure.

overlapping subproblems: When a recursive algorithm would visit the same sub-problems repeatedly, then a problem has overlapping subproblems.

Scanned with CamScanner

→ If a problem has optimal substructure, then we can recursively define an optimal solutions, then we can improve on a recursive implementation by computing each subproblem only once.

→ If a problem doesn't have optimal substructure, there is no basis for defining a recursive algorithm to find the optimal solutions

→ If a problem doesn't have overlapping sub problems, we don't have anything to gain by using dynamic programming.

Elements of Dynamic programming:

1. **Substructure**: Decompose the given problem into smaller subproblems. Express the solution of the original problem in terms of the solution for smaller problems.

2. **Table structure**: After solving the sub problems, store the results to the subproblems in a table.

→ This is done because subproblem solutions are reused many times, & we don't want to repeatedly solve the same problem over and over again.

Scanned with CamScanner

3. **Bottom up computation**: Using table combine the solution of smaller subproblems to solve larger subproblems & eventually arrives at a solution to complete problem.

Bottom up means:

→ Start with smallest subproblems

→ Combining their solutions obtain the solution to sub problems of increasing size.

→ Until solving at the solution of the original problem.

Components of dynamic programming:

Stages: The problem can be divided into several sub problems, which are called stages.

→ A stage is a small portion of a given problem.

States: Each stage has several states associated with it.

Decision: At each stage, there can be multiple choices out of which one of the best decisions should be taken.

→ The decision taken at every stage should be optimal.

Scanned with CamScanner

optimal policy: It is a rule which determines the decision at each stage, a policy is called an optimal policy, if it is globally optimal. This is known as Bellman principle of optimality.

→ Given the current state, the optimal choices for each of the remaining states does not depend on the previous states or decision.

→ There exist a recursive relationship that identify the optimal decisions for stage j, given that stage j+1, has already been solved.

→ The final stage must be solved by itself

Applications of dynamic programming:

→ 0/1 knapsack problem
→ All pair shortest path problem
→ Longest common subsequence (LCS)
→ Flight control & robotics control

Scanned with CamScanner

1.2.3

Assembly line scheduling:-

→ For solving assembly line scheduling through dynamic programming consider car manufacturing company.

→ From the raw materials, they manufacture car by passing the raw materials through several stages.

→ Each stage contains cost and they contain entry and exit at each stage.

→ The main objective is to manufacture car with minimum cost.

example:

s_1 - Engine fit

s_2 - checking engine

s_3 - light fit

s_4 - finishing

s_5 - Quality test.

level 1, level 2 } - Assembly line.

objective : minimum cost

Scanned with CamScanner

	1	2	3	4	5	
F_{1j}	11	21	24	29	38	
F_{2j}	14	19	26	30	36	37

levels.

	2	3	4	5	
L_{1j}	1	2	1	1	
L_{2j}	1	2	1	2	2.

minimum cost = 37

The path

Scanned with CamScanner

Algorithm:

```
int min (int a, int b)
{
return a < b ? a : b;
}
int carAssembly(int a[][N-S], int t[][N-S],
        int *e, int *x)
{
int T1 [NS], T2[NS], i;
T1[0] = e[0] + a[0][0];
T2[0] = e[1] + a[1][0];
for (i = 1; i < N-S ; ++i)
{
T1[i] = min (T1 [i-1] + a[0][i],
            T2 [i-1] + t[1][i] + a[0][i]);
T2[i] = min (T2 [i-1] + a[1][i],
            T1 [i-1] + t[0][i] + a[1][i]);
}
return min (T1 [N-S - 1] + x[0],
            T2 [N- S -1] + x[1]);
}
int main ()
{
int a[][NS] = { {8, 10, 4, 5, 9},
                {9, 6, 7, 5, 6} };
```

Scanned with CamScanner

```
int t[][N-S] = {{0, 2, 3, 1, 3, 2},
                {0, 2, 1, 2, 2, 1}};
int e[] = {3, 5}, x[] = {2, 1};
printf("%d", carAssembly(a, t, e, x));
return 0;
}.
```

output:

37.

Analysis:

Time complexity $= \Theta(n)$.

solution:

example 2:

	1	2	3	4	5
F_{1j}	14	19	22	24	42
F_{2j}	14	24	24	28	35

	2	3	4	5
L_{1j}	1	1	1	1
L_{2j}	2	1	2	2

solution

cost 35.

Scanned with CamScanner

1.2.4 Matrix chain Multiplication:

→ Given a sequence of matrices, find the most efficient way to multiply these matrices together.

→ The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

→ The cost of matrices depends on the times of multiplication to be performed.

→ The objective function is to minimize the cost of multiplication.

Example:

A_1	A_2	A_3	A_4
5×4	4×6	6×2	2×7

check: $((A_1 \cdot A_2) \cdot A_3) \cdot A_4$

or

$(A_1 \cdot A_2) \cdot (A_3 \cdot A_4)$

//There are multiple ways to perform multiplication.

n = node

$$\boxed{T(n) = \frac{{}^{2n}C_n}{n+1}}$$

$\Rightarrow T(3) = 5$ trees are possible.

Scanned with CamScanner

Tabulation method (bottom up approach).

m.

	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84.
4				0

s.

	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

m[1,1] A1

m[2,2] A2

m[3,3] A3

m[4,4] A4.

m[1,2]

A1 . A2

5x4 4x6.

= 5x4x6

= 120.

m[2,3]

A2 . A3

4x6 6x2

= 4x6x2

= 48

m[3,4]

A3. A4

6x2 2x7

= 6x2x7

= 84

m[1,3]

A1 .(A2. A3). or (A1. A2). A3

5x4 4x6 6x2 5x4 4x6 6x2

m[1,1] + m[2,3] + 5x4x2

= 0 + 48 + 40

= 88

m[1,2] + m[3,3] + 5 x6x2.

= 120+0+60

= 180.

$\boxed{m[1,3]=88}$

Scanned with CamScanner

$m[2,4]$

$A_2 . (A_3 . A_4)$ or. $(A_2 . A_3) . A_4$.

$4\times6 \quad 6\times2 \quad 2\times7$ $\qquad$ $4\times6 \quad 6\times2 \quad 2\times7$.

$m[2,2] + m[3,4] + 4\times6\times7$

$= 0 + 84 + 168$

$= 252$

$m[2,3] + m[4,4] + 4\times2\times7$

$= 48 + 0 + 56$

$= 104$.

$$\boxed{m[2,4] = 104.}$$

$A_1(A_2 . A_3 . A_4)$

$$m[1,4] = \min\{ \underset{(A_1 . A_2)\ (A_3 . A_4)}{m[1,1] + m[2,4] + 5\times4\times7},$$

$$m[1,2] + m[3,4] + 5\times6\times7,$$

$$\underset{(A_1 . A_2 . A_3) . A_4}{m[1,3] + m[4,4] + 5\times2\times7}\}$$

$$= \min\{0 + 104 + 140,\ 120 + 84 + 210,$$

$$88,\ 0 + 70\}$$

$$= \min\{244,\ 414,\ 158\}$$

$$\boxed{m[1,4] = 158.}$$

$A_1 \quad A_2 \quad A_3 \quad A_4$

$5\times4 \quad 4\times6 \quad 6\times2 \quad 2\times7$

$d_0 \quad d_1 \quad d_2 \quad d_3 \quad d_4$

$$m[i,j] = \min\{m[i,k] + m[k+1,j] + d_{i-1}\times d_k \times d_j\}$$

Scanned with CamScanner

$(A_1, A_2 . A_3) . (A_4)$. // $s[1,4] = 3$. $s[1,4]=3$.

$(A1 . (A2 . A3)) (A4)$.

//solution.

Analysis:

generating elements $= \frac{n(n+1)}{2}$ or $\frac{n(n-1)}{2}$.

since multiplying with all the elements $= n \times n^2$ // n^2.

$= n^3$.

$\therefore \Theta(n^3)$.

Algorithm:

```
main()
{ int n=5;
  int P[ ] = {5, 4, 6, 2, 7};
  int m[5][5] = {0};
  int s[5][5] = {0};
  int j, min, q;
  for(int d=1; d<n-1; d++) // find the (cols) difference
  {
```

i j	i j	i j
1 2	1 3	1 4
2 3	2 4	d=3
3 4	d=2	
d=1		

Scanned with CamScanner

```
for (int i=1; i < n-d; i++) //moves i.
{
  j = i+d;                          //finding minimum.
  min = 32767;
  for(int k=1; k <= j-1; k++)
  {
    q = m[i][k] + m[k+1][j] + P[i-1] * P[k] * P[j];
    if (q < min)
    {
      min = q;
      s[i][j] = k;
    }
  } m[i][j] = min;
}
}
cout << m[1][n-1];
}.
```

1.2.5. Longest common subsequence:

→ Given two sequences, find the length of the longest subsequence present in both of them.

→ A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

Scanned with CamScanner

example:

string 1: a b d a c e

string 2: b a b c e.

subsequence: "b, a, c, e".

"abce".

example:

string A:

1	2
b	d

string B:

a	b	c	d
0	1	2	3

B

		a 1	b 2	c 3	d 4
	0	0	0	0	0
b	1	0	1	1	1
d	2	0	1	1	2

A

b d.

∴ Longest common subsequence = "bd".

Algorithm :

```
if (A[i] = B[j]) // add 1 to previous diagonal val
{
    LCS[i,j] = 1 + LCS[i-1, j-1]
}
else
{
    LCS[i,j] = max(LCS[i-1,j], LCS[i,j-1])
}
```

Scanned with CamScanner

example:

Str1 : s t o n e //m size.

Str2 : l o n g e s t //n

		l	o	n	g	e	s	t
	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
s 1	0	0	0	0	0	0	1	1
t 2	0	0	0	0	0	0	1	2
o 3	0	0	1	1	1	1	1	2
n 4	0	0	1	2	2	2	2	2
e 5	0	0	1	2	2	3	3	3
			o	n		e		

∴ The longest common subsequence size = 3.
"one"

Analysis:

Time complexity: $\Theta(m \times n)$

1.2.6. Optimal Binary search tree:

→ An optimal binary search tree is a binary search tree for which the nodes are arranged on levels such that the tree cost is minimum.

→ for n keys = $\frac{{}^{2n}C_n}{n+1}$ optimal binary trees are possible.

$$w(0,n) = \sum_{i=1}^{n} f(i)$$

Scanned with CamScanner

example:

0	1	2	3	← index / no. of nodes
10	12	16	21	← node name
4	2	6	3	← cost / frequency.

	0	1	2	3
0	4	$8^{(0)}$	$20^{(2)}$	$26^{(2)}$
1		2	$10^{(2)}$	$16^{(2)}$
2			6	$12^{(2)}$
3				3

case 1: $l=1$. considering only one node.

(0,0)	(1,1)	(2,2)	(3,3).
4	2	6	3.

case 2: $l=2$, considering two nodes.

$\Rightarrow 4+2 + \min \begin{cases} 2 \leftarrow 0 \\ 4 \leftarrow 1. \end{cases}$ root

$= 4+2+2$

$= 8^{(0)}$. (0,1).

$\Rightarrow 2+6 + \min \begin{cases} 6 \leftarrow 1 \\ 2 \leftarrow 2 \end{cases}$ root

$= 8+2$

$= 10^{(2)}$ (1,2)

Scanned with CamScanner

2	3
16	21
6	3

$\Rightarrow 6+3+\min \begin{cases} 3 \leftarrow 2 \quad \text{root} \\ 6 \leftarrow 3 \end{cases}$

$= 6+3+3$

$= 12^{(2)} \quad (2,3).$

case 3: $l=3$, consider three nodes.

0	1	2
10	12	16
4	2	6

$\Rightarrow 4+2+6+\min \begin{cases} (1,2)\ 10 \leftarrow 0. \quad \text{root} \\ (0,0)+(2,2)\ 4+6 \leftarrow 1 \\ (0,1)\ 8 \leftarrow 2 \end{cases}$

$\Rightarrow 12+\min \begin{cases} 10 \\ 10 \\ 8^{(2)} \end{cases}$

$\Rightarrow 20^{(2)} \quad (0,2).$

1	2	3
12	16	21
2	6	3

$\Rightarrow 2+6+3+\min \begin{cases} (2,3)\ 12 \leftarrow 1 \quad \text{root} \\ (1,1)+(3,3)\ 2+3 \leftarrow 2 \\ (1,2)\ 10 \leftarrow 3 \end{cases}$

$= 11 + 5$

$= 16^{(2)} \quad (1,3)$

case 4: $l=4$, considering all nodes.

0	1	2	3
10	12	16	21
4	2	6	3

$\Rightarrow 4+2+6+3+\min \begin{cases} (1,3)\ 16 \leftarrow 0 \quad \text{root} \\ (0,0)+(1,3)\ 4+12 \leftarrow 1 \\ (0,1)+(3,3)\ 8+3 \leftarrow 2 \\ (0,2)\ 20 \leftarrow 3 \end{cases}$

Scanned with CamScanner

$$= 15 + \min \begin{cases} 16 \leftarrow 0 \\ 16 \leftarrow 1 \\ 11 \leftarrow 2 \\ 20 \leftarrow 3 \end{cases}$$

$= 15 + 11$

$= 26^{(2)}$ $(0,3)$.

$\therefore$ The tree after finding minimum cost/ frequency is

$\Rightarrow 6 \times 1 + 4 \times 2 + 3 \times 2 + 12 \times 3$

$= 6 + 8 + 6 + 6$

$= 26$. // cost.

Analysis:

Time taken for left subtree $= n$.
Time taken for right subtree $= n$
Time taken for root node $= n$.

$\therefore$ Time complexity $= O(n^3)$.

Scanned with CamScanner

example 2: for optimal binary search:

	1	2	3	4.
Key	10	20	30	40
prob	4	2	6	3.

i \ j	0	1	2	3	4
0	0	4	8 (1)	20 (3)	26 (3)
1		0	2	10 (3)	16 (3)
2			0	6	12 (3)
3				0	3
4.					0

$l = j - i = 0$. // consider no nodes.

$0-0=0$

$1-1=0$

$2-2=0$

$3-3=0$

$4-4=0$

$c[0,0]=0$ $c[2,2]=0$ $c[4,4]=0$

$c[1,1]=0$ $c[3,3]=0$

$l = j - i = 1$. // consider one node

$1-0=1$ $(0,1)$

$2-1=1$ $(1,2)$

$3-2=1$ $(2,3)$

$4-3=1$. $(3,4)$.

$c[0,1]=4$ $c[2,3]=6$

$c[1,2]=2$ $c[3,4]=3$

$l = j - i = 2$. // consider two nodes.

$2-0=2$ $(0,2)$

$3-1=2$ $(1,3)$

$4-2=2$ $(2,4)$.

$c[0,2]=8^{(1)}$ $c[2,4]=12^{(3)}$

$c[1,3]=10^{(3)}$

$c[0,2]$

1	2
10	20
4.	2

8 10

c[1,3]

2	3
20	30
2	6

$2+12=14.$ $6+4=\underline{\underline{10.}}$

c[2,4].

3	4
30	40
6	3

$12+3=15$ $6+6=\underline{\underline{12.}}$

$l=j-i=3.$ // consider three nodes.

$3-0=3\ (0,3)$ $c[0,3]=20^{(3)}$

$4-1=3\ (1,4)$ $c[1,4]=16^{(3)}.$

c[0,3]

1	2	3
10	20	30
4	2	6

1 - root

$=4+4+18=26^{(1)}$ $=4+12+6=32^{(1)}$

2 - root

$2+8+12=22^{(2)}$

3 - root

$6+4+4$ $=6+8+6$

$=14^{(3)}$ $=20^{(3)}$

Scanned with CamScanner

c[1,4]

2	3	4
20	30	40
2	6	3.

2-root

$= 2+12+9 = 23^{(2)}$

3- root

$= 6+4+6$

$= 16^{(3)}$

4- root

$= 3+12+5$

$= 20^{(4)}$

$= 3+4+18$

$= 25^{(4)}$

$l=4.$

$= i-j = 4.$ // consider all four nodes.

1	2	3	4
10	20	30	40
4	2	6	3

$= 4+2+6+3+\min$

(4,4) root

$12^{(3)} \leftarrow 1$

$(0,1)+(2,4)$

$4+12 \leftarrow 2$

$(0,2)+(3,4) \leftarrow 3$

$8+3 \leftarrow 4$

$(4,4)$

$10.$

$= 15+\min \begin{cases} 12 \leftarrow 1 \\ 16 \leftarrow 2 \\ 13 \leftarrow 3 \\ 20 \leftarrow 4. \end{cases} = 26^{(3)}$

Scanned with CamScanner

∴ The tree after finding minimum cost / frequency is

$= 6\times1 + 4\times2 + 3\times2 + 2\times3$

$= 26.$ //cost.

Algorithm:

```
optimalBST(P[1... n])
for i=1 to n do
    c[i, i-1]=0          //initial values
    c[i, i]= P[i]
    R[i, i] = i
    c[n+1, n] = 0
for d=1 to n-1 do //diagonal count
for i=1 to n-d do
    j= i+d
    for k= i to j do
    if (c[i, k-1] + c[k+1, j]) < minval
        minval = c[i, k-1] + c[k+1, j]
        kmin= k.
R[i, j]= k
sum = P[i]
```

Scanned with CamScanner

Unit - III

Graph and Tree Algorithms

1.1. **Graph:** A graph G may be defined as a finite set V of vertices and a set E of edges (pair of connected vertices.

Graph $\boxed{G = (V, E)}$

→ The set of vertices of graph is $V = \{1, 2, 3, 4, 5\}$

→ The set of ~~vertices~~ edges for the graph is $E = \{(1,2), (1,5), (1,3), (5,4), (4,3), (2,3)\}$.

→ The elements of E are always a pair of elements.

Graph.

Directed graph: If every edge (a,b) in a graph is marked by a direction from a to b, then it is directed graph (digraph).

Scanned with CamScanner

Undirected graph: If the directions are not marked on the edges, it is called undirected graph.

Graph Terminologies:

Adjacent Vertices: Two vertices a and b are said to be adjacent, if there is an edge connecting a and b.

ex: 3 & 4 are adjacent.

Path: A path is defined as a sequence of distinct vertices, in which each vertex is adjacent to the next.

ex: Path from 1 to 4 can be defined as a sequence of adjacent vertices (1,5), (5,4).

cycle: → A graph contains cycles if there is a path of non-zero length through the graph.

→ A cycle can be defined as the path which has no repeated edges or vertices except the first & last vertices.

Scanned with CamScanner

Connected graph: A connected graph is the one in which some path exists between every two vertices (u, v) in V. There are no isolated nodes in connected graph.

Complete graph: → A complete graph is the one in which every node is connected with all other nodes.

→ A complete graph contain $\frac{n(n-1)}{2}$ edges, where n is the number of nodes in the graph.

Weighted graph: → In a weighted graph, each edge is assigned with some data such as length or weight.

→ The weight of an edge e can be given as w(e) which must be a positive (+) value indicating the cost of traversing the edge.

Loop: An edge that is associated with the similar end points can be called as loop.

Degree of the node: It is the number of edges that are connected with that node. A node with degree 0 is called as isolated node.

Scanned with CamScanner

1.2. Trees:

→ Tree is a hierarchical data structure which stores the information naturally in the form of hierarchy style.

→ Tree is one of the most powerful and advanced data structures.

→ It is non linear data structure compared to arrays, linked list, stack and queue

→ It represents the nodes connected by edges.

structure of tree

Root: → Root is a special node in a tree.

→ The entire tree is referenced through root.

→ It doesnot have any parent.

Scanned with CamScanner

1.3. Representation of graphs:

→ It is technique which is to be used in order to store some graph, into computer's memory.

→ There are two ways for graph representation

1.3.1. Sequential Representation.

1.3.2. Linked Representation.

1.3.1. Sequential Representation.

→ We use adjacency matrix to store the mapping represented by vertices & edges.

→ In adjacency matrix, the rows and columns are represented by the graph vertices.

→ A graph having n vertices, will have a dimension $n \times n$.

→ An entry M_{ij} in the adjacency matrix representation of an undirected graph G will be 1 if there exists an edge between v_i & v_j.

Scanned with CamScanner

Undirected graph

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 1 & 0 \\ B & 1 & 0 & 1 & 1 & 0 \\ C & 0 & 1 & 0 & 0 & 1 \\ D & 1 & 1 & 0 & 0 & 1 \\ E & 0 & 0 & 1 & 1 & 0 \end{array}$$

Adjacency Matrix

Directed graph

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 0 & 0 \\ B & 0 & 0 & 1 & 1 & 0 \\ C & 0 & 0 & 0 & 0 & 1 \\ D & 1 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 1 & 0 \end{array}$$

Adjacency Matrix

Weighted directed graph

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 4 & 0 & 0 & 0 \\ B & 0 & 0 & 2 & 1 & 0 \\ C & 0 & 0 & 0 & 0 & 8 \\ D & 5 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 10 & 0 \end{array}$$

Adjacency Matrix

Scanned with CamScanner

1.3.2. Linked Representation

→ In the linked representation, an adjacency list is used to store the graph into the computer's memory.

Undirected graph

Adjacency List

→ An adjacency list is maintained for each node present in the graph which stores the node value and a pointer to the next adjacent node to the respective node.

→ If all the adjacent nodes are traversed then store the NULL in the pointer field of the last node of the list.

→ The sum of lengths of adjacency lists is equal to the twice of the number of edges present in an undirected graph.

Scanned with CamScanner

In case of weighted directed graph, each node contains an extra field ie called the weight of the node. (5)

weighted directed graph

Adjacency list

2. Traversal Algorithms

→ Tree traversal is known as tree search, is a form of graph traversal and refers to the process of visiting (checking / updating) each node in a tree data structure, exactly once.

→ Such traversals are classified by the order in which the nodes are visited

→ Tree traversal algorithms can be classified broadly into two categories

⇒ Depth First Search Algorithm
⇒ Breadth First Search Algorithm

Scanned with CamScanner

2.1. Depth First Search - (DFS) Algorithm:

→ It starts with the root node & first visits all nodes of one branch as deep as possible of the chosen Node. & before backtracking, it visits all other branches in a similar fashion.

→ There are three sub types under depth first search

2.1.1. Inorder Traversal

2.1.2. Preorder Traversal

2.1.3. Postorder Traversal.

2.1.1. Inorder Traversal

In this traversal method, the left subtree is visited first, then the root and the right sub tree.

⇒ D, B, E, A, F, C, G.

Algorithm:

1) Recursively traverse left subtree.
2) Visit root node.
3) Recursively traverse right subtree.

Scanned with CamScanner

2.1.2. Preorder Traversal.

In this traversal method, the root node is visited first, then the left subtree and finally right subtree.

⇒ A, B, D, E, C, F, G.

Algorithm:

1) Visit root node.
2) Recursively traverse left subtree
3) Recursively traverse right subtree

2.1.3. Post order traversal:

In this traversal method, the root node is visited last. First traverse the left subtree, then the right subtree and finally the root node.

D, E, B, F, G, C, A.

Scanned with CamScanner

Algorithm:

1) Recursively traverse left subtree
2) Recursively traverse right subtree
3) Visit root node.

2.2. Breadth First Search Algorithm:

→ It starts from root node & visits all nodes of current depth before moving to the next depth in the tree.

2.2.1. Level Order Traversal

→ The breadth of the tree takes priority first and then move to depth.

→ Visit all the nodes present at the same level one by one from left to right and then move to the next level, to visit all the nodes of that level.

⇒ A, B, C, D, E, F, G, H, I, J.

Scanned with CamScanner

2.3. Transitive closure:

→ Transitive closure of graph is the reachability matrix to reach from vertex u to vertex v of a graph.

→ If a graph is given, we have to find a vertex v which is reachable from another vertex u, for all vertex pairs (u, v).

$$\begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 1 & 1 & 1 & 1 & 1 \\ b & 0 & 1 & 0 & 0 & 0 \\ c & 0 & 1 & 1 & 0 & 0 \\ d & 0 & 1 & 1 & 1 & 1 \\ e & 0 & 1 & 1 & 0 & 1 \end{array}$$

Reachability matrix

3. Topological Sort:

→ Topological sorting for directed Acyclic graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering.

→ Topological sorting for a graph is not possible if the graph is not DAG.

→ Every DAG will have atleast one topological ordering.

Scanned with CamScanner

Steps:

→ Identify a node with no incoming edges

→ Add that node to the ordering.

→ Remove it from the graph.

→ Repeat.

example:

in degree ⇒

① = 0

② = 1

③ = 2

④ = 2

⑤ = 1.

ordering:

1

ordering:

1 2

1 2 4 3 5

or

1 2 4 5 3

∴ The topological sort for given graph is

{ 1 2 4 3 5
1 2 4 5 3.

example:

← Topological sorting cannot be done because it contains cyclic path.

example:

ordering

A C D B.

⋮

C A D B.

∴ The topological sorting for the given graph is

{ A C D B
C A D B }.

Analysis:

Time complexity = $O(M+N)$

no. of edges

no. of nodes.

Scanned with CamScanner

4. Strongly Connected Components:

→ Connectivity in an undirected graph means that every vertex can ~~be~~ reach every other vertex via any path.

→ If the graph is not connected the graph can be broken down into connected components.

→ Strong connectivity applies only to directed graphs.

→ A directed graph is strongly connected if there is a directed path from any vertex to every other vertex.

→ Strongly connected components can be solved by Kosaraju's algorithm

visited

stack based on finish time

Scanned with CamScanner

Reverse graph.

visited

I H G J
K B A C D
F E

~~I~~
~~J~~
~~G~~
~~H~~
~~K~~
~~B~~
~~D~~
~~E~~
~~F~~
~~C~~
~~A~~

stack by finish time

strongly connected components:

① ⇒ I H G J

② ⇒ K.

③ ⇒ B A C

④ ⇒ D F E

Kosaraju's Algorithm:

→ Create an empty stack 'S'.

→ DO DFS traversal of a graph. In DFS traversal, after calling recursive DFS for adjacent of a vertex, push the vertex to stack.

→ Reverse directions of all arcs to obtain the transpose graph.

→ One by one pop a vertex from S (stack) while S is not empty.

Scanned with CamScanner

~~Let~~

→ Let the popped vertex be 'v'. Take v as source and do DFS call on v. The DFS starting from v points strongly connected component of v.

Analysis:

$O(V+E)$ V= vertices
E= edges

5. Minimum Spanning Tree:

→ A spanning tree is a sub graph of an undirected and a connected graph, which includes all the vertices of the graph having a minimum possible no. of edges.

→ If a vertex is missed, then it is not a spanning tree

→ The edges may or may not have weights assigned to them.

→ The total no. of spanning trees with n vertices that can be created from a complete graph is equal to $n^{(n-2)}$

→ The cost of the spanning tree is the sum of the weights of all the edges in the tree.

Scanned with CamScanner

→ Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees.

example:

Tree

$n = 3$

∴ The possible spanning tree for given tree is

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

5.1. Properties of spanning tree:

→ A spanning tree must contain only $(V-1)$ edges where $G(V, E)$.

→ Remove $(E - V + 1)$ edges.

→ A spanning tree should be maximally acyclic.

→ It must be minimally connected.

→ $V^{(V-2)}$ is maximum possible spanning tree.

Scanned with CamScanner

5.2. Algorithms:

5.2.1. Kruskal's Algorithm

5.2.2. Prim's Algorithm

5.2.1. Kruskal's Algorithm:

→ It is used to determine the minimum cost spanning tree.

→ It plays vital role in computer networks.

→ In kruskal's algorithm, we choose the edge with the minimum cost. But it is not necessary that selected optimum edge is adjacent.

→ It generates the spanning tree in the increasing order of cost or weight associated with each edge.

example:

Step 1: starting with minimum cost edge

Scanned with CamScanner

step 2:

step 3:

step 4:

step 5:

forms cyclic path.

step 6:

step 7:

∴ minimum cost = 10 + 26 + 22 + 12 + 16 + 14

∴ = 100.

Scanned with CamScanner

examples:

cost = 14

cost = 96.

Algorithm:

Kruskal (G).

// Kruskal's algorithm for constructing a minimum spanning tree.

// Input: A weighted connected graph $G = (V, E)$

// Output: E_T, set of edges composing a minimum spanning tree of G.

sort E in non decreasing order of the edge weights $w(e_1) \leq .. \leq w(e_{|E|})$

$E^T \leftarrow \phi$: encounter $\leftarrow 0$ // Initialise set of edges & its size.

$K \leftarrow 0$ // initialise the no. of processed edges.

while encounter $< |V| - 1$

$K \leftarrow K + 1$

if $E_T \cup \{e_{i_K}\}$ is acyclic

$E_T \leftarrow E_T \cup \{e^*\}$;

encounter $\leftarrow$ encounter $+1$

return E_T.

Scanned with CamScanner

Analysis:

$\theta(E \log E)$ E - total no. of edges in graph.

5.2.2. Prim's Algorithm:

→ Our goal is to find the minimum cost spanning tree but with different approach.

→ We need to continue with edge which contains minimum cost.

example:

step 1:

step 2:

step 3:

step 4:

step 5:

Scanned with CamScanner

Step 6:

minimum cost

$10+26+14+22+16+12$

$=100.$

examples:

Algorithm:

Prim's (G)

{

input: A weighted connected graph $G=(V,E)$

output: E_T, set of edges composing a minimum spanning tree.

$V_T = \{V_0\}$ // set of tree vertices can be initialised with any vertex

$E_T = \phi$

for $i=1$ to $|V|-1$ do

{

$e^* = (u^*, v^*)$ among all edges(u,v)

$V_T = V_T \cup \{v^*\}$

$E_T = E_T \cup \{e^*\}$

}

return E_T

}

Scanned with CamScanner

Analysis :

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

Applications of spanning tree :

→ Designing efficient routing algorithm.

→ Network design.

6. Single source shortest path Algorithm:

→ Given a source vertex s from set of vertices V in a weighted graph where all its edges weights $w(u,v)$ are non negative, find the shortest path weights $d(s,v)$ from given source s for all vertices v present in the graph

6.1. Dijkstra Algorithm

6.2. Bellman Ford Algorithm.

6.1. Dijkstra Algorithm:

Scanned with CamScanner

Relaxation :

if $(d[u] + c[u,v] < d[v])$

$d[v] = d[u] + c(u,v)$.

V	d[V]
2	2
3	3
4	8
5	6
6	9

∴ The minimum cost from source to destination is 9.

Analysis

$\Theta(n^2)$

Scanned with CamScanner

selected vertex	2	3	4	5	6.
4	50	45	10	∞	∞
5	50	45	10	25	∞
2	45	45	10	25	∞
3	45	45	10	25	∞
6	45	45	10	25	∞

∴ All the vertex given in graph is visited with the minimum cost.

6.2. Bellman Ford Algorithm:

Scanned with CamScanner

→ Relax the vertex upto $|V| = n-1$ times.

$\therefore n = 7.$ $|V| = n-1 = 7-1 = 6.$

→ Relaxation: (u, v)

if $(d[u] + c(u,v) < d[v])$

$d[v] = d[u] + c(u,v).$

→ List of edges:

$(1,2), (1,3), (1,4), (2,5), (3,5), (4,6), (6,7)$

$(3,2), (4,3), (5,7)$

v	d[v]
1	0
2	1
3	3
4	5
5	0
6	4
7	3

Scanned with CamScanner

Analysis: $O(n^2)$.

In complete graph $\Rightarrow O(n^3)$.

$n = 4$.

$|V| = n - 1 = 3$.

edge list $\Rightarrow$ (1,2), (1,4), (4,3), (3,2).

V	d[V]
1	0
2	-2
3	8
4	5

7. All pair shortest path:

→ The all pair shortest path algorithm is also known as Floyd-Warshall algorithm is used to find all pair shortest path problem from a given weighted graph.

→ As a result of this algorithm, it will generate a matrix, which will represent the minimum distance from any node to all other nodes in the graph.

Scanned with CamScanner

→ All pair shortest path algorithm is used to find the shortest path between every pair of nodes in graph by using intermediate node.

example:

No intermediate node

A^0

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 11 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & \infty & 0 \end{array}$$

1- intermediate node

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 11 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{array}$$

2- intermediate node

A^2

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 6 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 7 & 0. \end{array}$$

Scanned with CamScanner

3 - intermediate node.

A^3

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 6 \\ 2 & 5 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{array}$$

∴ The shortest path for the given directed graph using all pair shortest path (Floyd Warshall) is

Algorithm:

```
AllpairshortestPath (cost, A, n)
 for i=1 to n
     for j=1 to n
         A[i, j] = cost [i, j]
     end for
 end for
 for k=1 to n
     for i=1 to n
         for j=1 to n
         A [i, j] = min {A[i, j], A[i, k] + A[k, j]}
         end for
     end for
 end for
```

Scanned with CamScanner

Analysis :

$\Theta(V^3)$ V = vertices.

8. Network Flow Algorithms:

→ Given a graph which represents a flow network where every edge has a capacity also given two vertices source s. and sink t. in the graph.

→ Find out the maximum possible flow from s to t with the following constraints.

⇒ Flow on an edge doesn't exceed the given capacity of the edge

⇒ In-flow is equal to out-flow for every vertex except s and t

Algorithm :

Ford - Fulkerson Algorithm

→ Start with a initial flow as 0.

→ While there is augmenting path from source to sink add two path flow to flow.

→ Return flow.

Scanned with CamScanner

Terminologies:

Residual graph: It's a graph which indicates additional possible flow. If there is such path from source to sink then there is a possibility to add flow.

Residual capacity: It's original capacity of the edge minus flow.

Minimal cut: Also known as bottle neck capacity, which decides maximum possible from flow source to sink through an augmented path.

Augmenting path: It can be done in two ways.

→ Non full forward edges.
→ Non empty backward edges.

example:

source = A
sink = F

Scanned with CamScanner

Augmenting paths	Bottle neck capacity
1. S→A→D→T	8
2. S→C→D→T	2
3. S→C→D→A A→B→T	4
4. S→A→D→B→T	2
5. S→C→D→B→T	3

∴ The maximum flow = 19.

Scanned with CamScanner

Analysis:

Time complexity = $O(E * F)$

edges

flow

Scanned with CamScanner

* Backtracking:

1 → Backtracking is a general algorithm for finding all the solutions to some computational problem.

2. → In backtracking method we begin by defining a solution space for the problem.

2.0 → In backtracking technique, we try to build a solution incrementally.

→ Whenever the algorithm needs to decide between the two alternatives to next component of solution, it simply tries both options recursively.

→ On reaching a dead end, it backtracks.

→ solution space is called as state space tree.

→ state space tree – tree of choice.

→ Depth first search method is used to explore state space tree.

Applications:

a → n Queen's problem

b → Hamiltonian circuit problem.

c -->Sum of subsets

d -->Graph Colouring

Scanned by CamScanner

* 8- Queen problem:

concept:

→ Placing Queen on 8X8 chess board, we have to check the following conditions.

→ NO 2 Queens on the same row

→ NO 2 Queens on same column

→ NO 2 Queens on same diagonal.

	1	2	3	4	5	6	7	8
Q_1			Q_1					
Q_2						Q_2		
Q_3		Q_3						
Q_4							Q_4	
Q_5	Q_5							
Q_6				Q_6				
Q_7								Q_7
Q_8					Q_8			

Algorithm:

Algorithm N Queens (K, n)

//using backtracking, this procedure prints all possible place for n queens on nxn, so that non attacking.

```
{
for ( i = 1 ; i <= n ; i++)
```

Scanned by CamScanner

3. → ~~It~~ We construct one candidate solution at a time & evaluate the partially constructed solution.

4. → If no potential values of the remaining components can lead to a solution, the remaining part of solution is not generated.

5. → It uses state space tree for constructing feasible sol.

6. → It terminates a node as soon as it can be guaranteed that no solution to the problem can be obtained.

7. → The state space tree is generated using DFS (Depth first search)

8. → DFS stops exploring subtree rooted at nodes that cannot lead to a solution and backtracks to such a node's parent to continue the search.

9. → The search terminates when we found the answer

<u>example</u>:

8 Queen's problem.

Scanned by CamScanner

```
{
  if place (k, i)
  {
    x[k] = i;
      if (k == n)
          Print (x[1 : n]);
      else
          N Queens (k+1 ; n);
  }
}
}
```

complexity: $n! = 8! = 64$.

* Hamiltonian circuit problem.

→ Let $G = (V, E)$ be a connected graph with 'n' vertices.

→ A Hamiltonian cycle is a round trip path along n edges of G that visits every vertex once and return to its starting position.

~~Procedure~~.

~~A~~ The ~~graph~~

ex:

Scanned by CamScanner

solution:

⇒ 1-2-8-7-6-5-4-3-1

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

ex. 2.

⇒ a-b-f-e-c-d-a

a-c-b-f-e-d-a

a-c-d-e-f-b-a.

ex: 3

⇒ The graph does not contain Hamiltonian cycle.

Algorithm:

Algorithm Hamiltonian (k)

// This algorithm uses the recursive formulation of backtracking.

{

repeat

{ //generate values for x[k]

Next value (k); // assign a legal next value k.

if (x[k] = = 0)

return ;

```
if (k==n)
    point (x[1:n]); // it points all the vertices
else
    Hamiltonian (k+1);
}
Until (false);
}.
```

<u>complexity</u> :

$O(n)$.

Scanned by CamScanner

Sum of subsets:

→ The problem is to find subset of a given set. $S = \{s_1, s_2 \ldots s_n\}$ of n positive integers whose sum is equal to a given positive integer 'd'.

→ Arrange the sets in increasing order.

→ The set should not be fixed size.

<u>example:</u> $S = \{1, 3, 4, 5\}$. $d = 8$.

$S_1 = \{1, 3, 4\}$.

$S_2 = \{3, 5\}$

<u>state space tree:</u>

Scanned by CamScanner

$\therefore \; S_1 = \{1, 3, 4\}$

$S_2 = \{3, 5\}$.

example:

$S = \{5, 10, 12, 13, 15, 18\}$

$d = 30$.

Scanned by CamScanner

Algorithm:

```
if (subset is satisfying the constraint)
{
  print subset
  exclude the current element &
  consider next element.
}
else
{
  generate the nodes of present
  level along breadth of tree &
  recur for next levels
}
}.
```

Scanned by CamScanner

Graph colouring.

→ The problem is to assign colors to certain elements of a graph subject to certain constraints.

→ Graph colouring problem is provided with m colours, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using same color.

example:

colours.
m = 3
R, G, B

→ m colouring decision problem:

⇒ Whether the given colors can be used for coloring the vertices

Scanned by CamScanner

→ **m - coloring optimization problem:**

⇒ Minimum how many colors required for coloring the graph.

example:

$m = 3$

$= \{R, G, B\}$

⇒ State space tree:

$S_1 = \{R, G, R, G\}$

$S_2 = \{R, G, R, B\}$

$S_3 = \{R, G, B, G\}$

$S_4 = \{R, B, R, G\}$

$S_5 = \{R, B, R, B\}$

Scanned by CamScanner

solutions = {R, G, R, G}

{R, G, R, B}

{R, G, B, G}

{R, B, R, G}

{R, B, R, B}

Algorithm:

While there are untried conflicts configurations

{

generate the next configuration

If no adjacent vertices are colored with same color

{ print the configuration

}

}

Time complexity : $O(n^m)$.

Scanned by CamScanner

Knapsack problem by Backtracking:

example:

	1	2	3	4
weight	2	3	4	5
Profit	3	5	6	10

capacity
$m = 8$

Scanned by CamScanner

$\therefore$ 2 & 4 $\Rightarrow w \Rightarrow 8$

$x\begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 1 \end{pmatrix}$. $\quad p \Rightarrow 15$

Scanned by CamScanner

* Branch & bound:

→ Branch & bound is a systematic method for solving optimization problems that applies where the greedy method & dynamic programming fail.

→ Branch & bound is an approach developed for solving discrete & combinatorial optimization problems.

→ B & B is a problem solving technique in which optimal solution is obtained by means of state space tree.

→ state space is the set of all paths from root node to all other node.

Scanned by CamScanner

→ Branching is done at each node of the tree.

→ Bound value is computed at each node of tree.

→ compare the nodes bound values with the value of the best solution obtained till now.

→ ~~non~~ ~~promising node~~: (dead node): If the bound value is not better than the other node, node is not expanded further. such a node is called non promising node.

→ Live node: It is a node in state space tree which is a promising node & it can be further expanded inorder to get the solution to the given problem.

Scanned by CamScanner

* Knapsack problem:

statement:

→ Given 'n' objects and a knapsack in which objects is to be placed.

→ The knapsack has the capacity 'c'.

→ The objective is to obtain filling of knapsack with maximum profit, but it should not exceed weight 'w' of the knapsack.

→ compute upper bound of the tree.

$$\boxed{UB = V + (c - W)(V_{i+1} / W_{i+1})}$$

ex:

		weight w	value v	v/w
1)	I_1	4	40	10
	I_2	7	42	6
	I_3	5	20	4
	I_4	3	12	4.

$c = 10$

maximum profit so finding upper bound.

Initially $w = 0$, $v = 0$, $v_{i+1} = v_1 = 40$, $w_{i+1} = w_1 = 4$.

$v/w = 10$.

1). $UB = 0 + (10 - 0)(40/4) = 10\left(\frac{40}{4}\right) = 10 \times 10 = 100.$

Scanned by CamScanner

weight value

$w=0,\ v=0.$
$UB=100..$

$UB = V+(C-W)$ (with, wit

with I_1 — without I_1.

$w_1=4 \quad v_1=40.$
$UB = 76.$

$w_1=4 \quad w_2=7$
$v_1=40. \quad v_2=42$

$UB = 40+(10-4)(42/7)$
$= 40+(6)(6)$
$= 40+36$
$= 76$

$w_1=0 \quad v_1=0$
$UB=60.$

$w_1=0, \quad w_2=$
$v_1=0, \quad v_2=42.$

$UB = 0+(10-0)(42/7)$
$= 0+10(6)$
$= 60.$

with I_2 — w/o I_2.

$w=7+4=11 \quad v=40+42=62$
$UB=$

$UB = w=11$ greater than knapsack capacity cannot proceed.

$w=4,\ v=40$
$UB=64.$

$UB = 40+(10-4)(60/4)$
$= 40+6(4)$
$= 40+24$
$= 64.$

with I_3 — w/o I_3

$UB = 60+(10-9)(12/3)$
$= 60+(1)(4)$
$= 64.$

$w=9,\ v=60$
$UB=64$

value is maximum.

$w=4,\ v=40$
$UB=64.$

4

with I_4 — w/o I_4.

greater $12>10$, ✗

$w=12,\ v=72$

$w=9,\ v=60$
$UB=64.$ ⇒ feasible solution

I_1 & I_3 $\boxed{w=9,\ v=60,\ UB=64.}$

Scanned by CamScanner

2)

	weight w	value v.	v/w
I_1	4	40	10
I_2	7	42	6
I_3	5	25	5
I_4	3	12	4.

$C = 10$

$$\boxed{UB = v + (c-w)(v_{i+1}/w_{i+1})}$$

Initially $w=0,\ v=0,$ $\quad v_{i+1} = v_1 = 40,\ w_{i+1} = w = 4.$ $\quad W=10.$

$UB = 0 + (10-0)(10) = 100.$

$\boxed{UB = 100.}$

Scanned by CamScanner

* Travelling salesman problem:

Given:

→ A graph showing a number of cities connected by edges.

→ cost of each edge is given using cost matrix

Aim of problem:

Find the shortest path to cover all the cities & come back to the same city.

constraints:

→ All the cities should be covered.

→ each city should be visited only once.

→ The starting & ending point should be the same.

ex:

step 1:

	1	2	3	4
1	∞	20	5	10
2	20	∞	15	25
3	5	15	∞	5
4	10	25	5	∞
	~~5~~	~~15~~		

Row minimization

→ each row consider minimum value & subtract with other cost.

∴

Step 2:

	1	2	3	4	
1	∞	15	0	5	5
2	5	∞	0	10	15
3	0	10	∞	0	5
4	5	20	0	∞	5
	0	10	0	0	

row minimization

row reduced matrix

Scanned by CamScanner

step 3:

Column minimization:

	1	2	3	4	
1	∞	5	0	5	0
2	5	∞	0	10	0
3	0	0	∞	0	0
4	5	10	0	∞	0
	0	10	0	0	

Total cost = row min + col min

$= (5+15+5+5) + (10)$

$= 30 + 10$

$= 40.$

∴ 1 – 2 – 3 – 4 – 1.

cost = 50.

Scanned by CamScanner

1-2:

	1	2	3	4	
1	∞	∞	∞	∞	
2	∞	∞	0	10	0
3	0	∞	∞	0	0
4	5	∞	0	∞	
	0	∞	0	0	$\hat{r}=0$

$c(1,4) = c(1,4) + r + \hat{r}$

$= 45 + 40 + 0$

$= 45.$

1-3:

	1	2	3	4
1	∞	∞	∞	∞
2	50	∞	~~∞~~ 105	50
3	∞	0	∞	0
4	50	10	∞	∞
	5	0	0	$\hat{r}=10$

$c(1,3) = c(1,3) + r + \hat{r}$

$= 0 + 40 + 10$

$= 50$

1-4:

	1	2	3	4	
1	∞	∞	∞	∞	
2	5	∞	0	∞	
3	0	0	∞	∞	$\hat{r}=0$
4	∞	10	0	∞	

$c(1,4) = c(1,4) + r + \hat{r}$

$= 5 + 40 + 0$

$= 45$

1-2-3: (2-row, 3-col = ∞)

	1	2	3	4	
1	∞	~~∞~~ 0	∞	~~∞~~	5
2	∞	∞	∞	∞	
3	~~∞~~	~~∞~~	∞	~~∞~~	0
4	~~5~~ 0	~~∞~~ 5	∞	∞	5
	5			$\hat{r}=5$	

$c(1,2,3) = c(1,3) + r + \hat{r}$

$= 0 + 45 + 5$

$= 50.$

1-2-4: 2-row, 4-col = ∞

	1	2	3	4
1	∞	∞	∞	∞
2	∞	∞	∞	∞
3	0	∞	∞	∞
4	5	∞	0	∞
				$\hat{r}=0$

$c(1,4) = c(2,4) + r + \hat{r}$

$= 45 + 10 + 0$

$= 55.$

1-2-3-4.

	1	2	3	4
1	∞	∞	∞	∞
2	∞	∞	∞	∞
3	∞	∞	∞	∞
4	5	∞	∞	∞

0. $c(1,2,3,4) = 50.$

Scanned by CamScanner

TSP

ex: 2.

step 1:

	1	2	3	4
1	∞	10	15	20
2	5	∞	9	10
3	6	13	∞	12
4	8	8	9	∞

step 2:
row minimization/ row reduced cost matrix

	1	2	3	4	
1	∞	0	5	10	10
2	0	∞	4	5	5
3	0	7	∞	6	6
4	0	0	1	∞	8
	0	0	1	5	

step 3:
column minimization / column reduced cost matrix

	1	2	3	4
1	∞	0	4	5
2	0	∞	3	0
3	0	7	∞	1
4	0	0	0	∞

step 4:
Total reduced cost = Row cost + col cost

$= (10+5+6+8) + (0+0+1+5)$

$= 29 + 6$

$\boxed{r = 35}$

Scanned by CamScanner

Step 4: contd:

Sol:

1 – 2 – 4 – 3 – 1.

= 35.

1-2:

	1	2	3	4
1	∞	∞	∞	∞
2	∞	∞	3	0
3	0	∞	∞	1
4	0	∞	0	∞

$\Rightarrow c(1,2) + r + \hat{r}$.

$= 0 + \cancel{\phi} + 35 + 0$.

$\hat{r} = 0$.

1-3:

	1	2	3	4
1	∞	∞	∞	∞
2	∞	∞	∞	0
3	∞	∞	∞	1
4	0	0	∞	∞
	0	0	∞	1

$\Rightarrow c(1,3) + r + \hat{r}$

$= 4 + 35 + 1$

$\hat{r} = 1$. $= 40$.

Scanned by CamScanner

1-4:

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty \\ 3 & 0 & 7 & \infty & \infty \\ 4 & \infty & 0 & 0 & \infty \\ \hline & & 0 & 0 & \end{array}$$

$= c(1,4) + r + \hat{r}$

$c(1-4) = 5 + 35 + 0$

$= 40.$

1-2-3: (use matrix of node 2)

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \infty & \infty & \infty & \infty \\ 2 & \infty & \infty & \infty & 0 \\ 3 & 0 & \infty & \infty & 1 \\ 4 & 0 & \infty & \infty & \infty \end{array}$$

(2 - row, 3rd - col - ∞)

2-3:

$= c(2,3) + r + \hat{r}$

$= 3 + 35 + 0.$

$= 38.$

1-2-4: (1-3) (2 - row, 4th - col = ∞).

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \infty & \infty & \infty & \infty \\ 2 & \infty & \infty & \infty & 0 \\ 3 & \infty & 7 & \infty & \infty \\ 4 & 0 & 0 & \infty & \infty \end{array}$$

$\hat{r} = 1$

2-4

$= \text{cost}(2,4) + r + \hat{r}$

$= 0 + 35 + 0$

$= 35$

1-2-4-3:

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \infty & \infty & \infty & \infty \\ 2 & \infty & \infty & \infty & 0 \\ 3 & 0 & 7 & \infty & \infty \\ 4 & 0 & \infty & \infty & \infty \end{array}$$

$\hat{r} = 0$

$\text{cost}(4-3) + r + \hat{r}$

$= 0 + 35 + 0$

$= 35$

Scanned by CamScanner

NP & P Problems: NP- Hard & NP- complete

Polynomial Time

→ Linear search – n

→ Binary search – $\log n$

→ Insertion sort – n^2

→ merge sort – $n \log n$

→ matrix multiplication – n^3

Exponential time (time consuming algo.)

→ 0/1 Knapsack – 2^n

→ Travelling salesman – 2^n

→ Hamiltonian cycle – 2^n

n^{10}

n^{100}

① → we need to try solving exponential time problems in polynomial time to get solution, if you didn't get solution, try doing some deterministic / non deterministic solution by.

NP complete problem:

→ NP complete is a decision problem in which there is a question in some formal system that can be posed as a yes-no question depends on input values.

→ P problem: The set of polynomially solvable problems are known as P problems.

ex: polynomial time.

→ NP – non polynomial (exponential).

→ NP problems include problems with exponential algorithm but have not proved that they cannot have polynomial time algorithm.

Scanned by CamScanner

UNIT 5

1. Tractable and Intractable problems

1.1 Polynomial Time (P)

- Most of the algorithms we have looked at so far have been polynomial-time algorithms
- On inputs of size n, their worst-case running time is O(n k) for some constant k
- The question is asked can all problems be solved in polynomial time?
- From what we've covered to date the answer is obviously no.
- There are many examples of problems that cannot be solved by any computer no matter how much time is involved
- There are also problems that can be solved, but not in time O(n k) for any constant k

1.2 Non-Polynomial Time (NP)

- Another class of problems are called NP problems
- These are problems that we have yet to find efficient algorithms in Polynomial Time for, but given a solution we can verify that solution in polynomial time
- Can these problems be solved in polynomial time?
- It has not been proved if these problems can be solved in polynomial time, or if they would require super polynomial time
- This so-called P != NP question is one which is widely researched and has yet to be settled.

1.3 Tractable and Intractable

- Generally, we think of problems that are solvable by polynomial time algorithms as being tractable, and problems that require super polynomial time as being intractable.
- Sometimes the line between what is an 'easy' problem and what is a 'hard' problem is a fine one.
- For example, "Find the shortest path from vertex x to vertex y in a given weighted graph". This can be solved efficiently without much difficulty.
- However, if we ask for the longest path (without cycles) from x to y, we have a problem for which no one knows a solution better than an exhaustive search Programming and Data Structures

1.4 Deterministic v Non-Deterministic

- Let us now define some terms – P: The set of all problems that can be solved by deterministic algorithms in polynomial time
- By deterministic we mean that at any time during the operation of the algorithm, there is only one thing that it can do next
- A nondeterministic algorithm, when faced with a choice of several options, has the power to "guess" the right one.
- Using this idea we can define NP problems as, – NP: The set of all problems that can be solved by nondeterministic algorithms in polynomial time.

1.5 Is P = NP?

- Obviously, any problem in P is also in NP, but not the other way around
- To show that a problem is in NP, we need only find a polynomial-time algorithm to check that a given solution (the guessed solution) is valid.
- But the idea of nondeterminism seems silly.
- A problem is in NP if we can 'guess' a solution and verify it in polynomial time!!
- No one has yet been able to find a problem that can be proven to be in NP but not in P
- Is the set P = NP? We don't know. If it is, then there are many efficient algorithms out there just waiting to be discovered.
- Most researchers believe that P != NP, but a proof remains to be shown

1.6 NP-Completeness

- NP-complete problems are set of problems that have been proved to be in NP
- That is, a nondeterministic solution is quite trivial, and yet no polynomial time algorithm has yet been developed.
- This set of problems has an additional property which does seem to indicate that P = NP
- If any of the problems can be solved in polynomial time on a deterministic machine, then all the problems can be solved in NP(Cook's Theorem)
- It turns out that many interesting practical problems have this characteristic

1.7 Examples of NP-Complete

Travelling Salesman Problem: Given a set of cities and distances between all pairs, find a tour of all the cities of distance less than M.

Hamiltonian Cycle: Given a graph, find a simple cycle that includes all the vertices.

Partition: Given a set of integers, can they be divided into two sets whose sum is equal?

Integer Linear Programming: Given a linear program is there an integer solution?

Vertex Cover: Given a graph and an integer N, is there a set of fewer than N vertices which touches all the edges?

1.8 NP Hard

A problem is in the class NPC if it is in NP and is as **hard** as any problem in NP. A problem is **NP-hard** if all problems in NP are polynomial time reducible to it, even though it may not be in NP itself.

If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable. These problems are called **NP-complete**. The phenomenon of NP-completeness is important for both theoretical and practical reasons.

Definition of NP-Completeness

A language **B** is ***NP-complete*** if it satisfies two conditions

- **B** is in NP

- Every **A** in NP is polynomial time reducible to **B**.

If a language satisfies the second property, but not necessarily the first one, the language **B** is known as **NP-Hard**. Informally, a search problem **B** is **NP-Hard** if there exists some **NP-Complete** problem **A** that Turing reduces to **B**.

The problem in NP-Hard cannot be solved in polynomial time, until **P = NP**. If a problem is proved to be NPC, there is no need to waste time on trying to find an efficient algorithm for it. Instead, we can focus on design approximation algorithm.

NP-Complete Problems

Following are some NP-Complete problems, for which no polynomial time algorithm is known.

- Determining whether a graph has a Hamiltonian cycle
- Determining whether a Boolean formula is satisfiable, etc.

NP-Hard Problems

The following problems are NP-Hard

- The circuit-satisfiability problem
- Set Cover
- Vertex Cover
- Travelling Salesman Problem

2. Cooks theorem

Stephen Cook presented four theorems in his paper "The Complexity of Theorem Proving Procedures". These theorems are stated below. We do understand that many unknown terms are being used in this chapter, but we don't have any scope to discuss everything in detail.

Following are the four theorems by Stephen Cook −

Theorem-1

If a set **S** of strings is accepted by some non-deterministic Turing machine within polynomial time, then **S** is P-reducible to {DNF tautologies}.

Theorem-2

The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {tautologies}, {DNF tautologies}, D3, {sub-graph pairs}.

Theorem-3

- For any $T_Q(k)$ of type **Q**, TQ(k)k√(logk)2TQ(k)k(logk)2 is unbounded
- There is a $T_Q(k)$ of type **Q** such that TQ(k)⩽2k(logk)2TQ(k)⩽2k(logk)2

Theorem-4

If the set S of strings is accepted by a non-deterministic machine within time $T(n) = 2^n$, and if $T_Q(k)$ is an honest (i.e. real-time countable) function of type **Q**, then there is a constant **K**, so **S** can be recognized by a deterministic machine within time $T_Q(K8^n)$.

- First, he emphasized the significance of polynomial time reducibility. It means that if we have a polynomial time reduction from one problem to another, this ensures that any polynomial time algorithm from the second problem can be converted into a corresponding polynomial time algorithm for the first problem.
- Second, he focused attention on the class NP of decision problems that can be solved in polynomial time by a non-deterministic computer. Most of the intractable problems belong to this class, NP.
- Third, he proved that one particular problem in NP has the property that every other problem in NP can be polynomially reduced to it. If the satisfiability problem can be solved with a polynomial time algorithm, then every problem in NP can also be solved in polynomial time. If any problem in NP is intractable, then satisfiability problem must be intractable. Thus, satisfiability problem is the hardest problem in NP.
- Fourth, Cook suggested that other problems in NP might share with the satisfiability problem this property of being the hardest member of NP.

3. Approximate Algorithms

Introduction:

An Approximate Algorithm is a way of approach **NP-COMPLETENESS** for the optimization problem. This technique does not guarantee the best solution. The goal of an approximation algorithm is to come as close as possible to the optimum value in a reasonable amount of time which is at the most polynomial time. Such algorithms are called approximation algorithm or heuristic algorithm.

- For the traveling salesperson problem, the optimization problem is to find the shortest cycle, and the approximation problem is to find a short cycle.
- For the vertex cover problem, the optimization problem is to find the vertex cover with fewest vertices, and the approximation problem is to find the vertex cover with few vertices.

Performance Ratios

Suppose we work on an optimization problem where every solution carries a cost. An Approximate Algorithm returns a legal solution, but the cost of that legal solution may not be optimal.

For Example, suppose we are considering for a **minimum size vertex-cover (VC)**. An approximate algorithm returns a VC for us, but the size (cost) may not be minimized.

Another Example is we are considering for a **maximum size Independent set (IS)**. An approximate Algorithm returns an IS for us, but the size (cost) may not be maximum. Let C be the cost of the solution returned by an approximate algorithm, and C* is the cost of the optimal solution.

We say the approximate algorithm has an approximate ratio P (n) for an input size n, where

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq P(n)$$

Intuitively, the approximation ratio measures how bad the approximate solution is distinguished with the optimal solution. A large (small) approximation ratio measures the solution is much worse than (more or less the same as) an optimal solution.

Observe that P (n) is always ≥ 1, if the ratio does not depend on n, we may write P. Therefore, a 1-approximation algorithm gives an optimal solution. Some problems have polynomial-time approximation algorithm with small constant approximate ratios, while others have best-known polynomial time approximation algorithms whose approximate ratios grow with n.

U18PCCS401
CLOUD COMPUTING

UNIT I
UNDERSTANDING CLOUD COMPUTING

Cloud Computing : History of CC – Cloud Architecture – Cloud Storage – Why Cloud Computing Matters – Advantages CC – Disadvantages CC – Companies in the Cloud Today – Cloud Services - Web-Based Application.

UNIT-II
CLOUD ARCHITECTURE AND MODEL

Technologies for Network Based system - System Models for Distributed and Cloud Computing - NIST Cloud Computing Reference Architecture

Cloud models : Characteristics - Cloud Models (IaaS, PaaS, SaaS)

Public vs. Private Cloud-Cloud Solutions-Cloud ecosystem-Service Management-Computing on demand.

UNIT-III
CLOUD INFRASTRUCTURE

Architectural Design of compute and Storage Clouds-Layered Cloud Architecture Development-Design Challenges-Inter Cloud Resource Management-Resource Provisioning and Platform Deployment-Global Exchange of Cloud Resources.

UNIT-IV
PROGRAMMING MODEL

Parallel and Distributed Programming Paradigms-Map Reduce-Twister and Iterative Map Reduce-Hadoop Library from Apache-Mapping Applications-Programming Support-Google App Engine, Amazon AWS-Cloud Software Environments-Eucalyptus, Open Nebula, Open Stack, Aneka, CloudSim.

UNIT-V
CLOUD SECURITY

Security Overview-Cloud Security Challenges and Risks-Software-as-a-Service-Security Security Governance-Risk Management-Security Monitoring-Security Architecture Design-Data Security-Application Security-Virtual Machine Security-Identity Management and Access Control-Autonomic Security.

UNIT I
UNDERSTANDING CLOUD COMPUTING

- History of CC
- Cloud Architecture
- Cloud Storage
- Why Cloud Computing Matters
- Advantages CC
- Disadvantages CC
- Companies in the Cloud Today
- Cloud Services
- Web-Based Application

History of CC

In 1963, DARPA (the Defense Advanced Research Projects Agency), presented MIT with $2 million for Project MAC. The funding included a requirement MIT develop technology allowing for a "computer to be used by two or more people, simultaneously." In this case, one of those gigantic, archaic computers using reels of magnetic tape for memory and was the precursor to what has now become collectively known as Cloud Computing. It acted as a primitive Cloud with two or three people accessing it. The word "Virtualization" was used to describe this situation, though the word's meaning later expanded.

In 1969, J. C. R. Licklider helped develop the ARPANET (Advanced Research Projects Agency Network), a "very" primitive version of the Internet. JCR, or "Lick" was both a psychologist and a computer scientist, and promoted a vision called the "Intergalactic Computer Network," in which everyone on the planet would be interconnected by way of computers, and able to access information from anywhere. (What could such an unrealistic, impossible-to-pay-for, fantasy of the future look like?) The Intergalactic Computer Network, otherwise known as the Internet, is necessary for access to the Cloud.

The meaning of Virtualization began shifting in the 1970s, and now describes the creation of a virtual machine, that acts like a real computer, with a fully functional operating system. The concept of Virtualization has evolved with the Internet, as businesses began offering "virtual" private networks as a rentable service. The use of virtual computers became popular in the 1990s, leading to the development of the modern Cloud Computing infrastructure.

The Late 1990s

In its early stages, the Cloud was used to express the empty space between the end user and the provider. In 1997, Professor Ramnath Chellapa of Emory University defined Cloud Computing as the new "computing paradigm, where the boundaries of computing will be determined by economic rationale, rather than technical limits alone." This somewhat ponderous description rings true in describing the Cloud's evolution.

The Cloud gained popularity as companies gained a better understanding of its services and usefulness. In 1999, Salesforce became a popular example of using Cloud Computing successfully. They used it to pioneer the idea of using the Internet to deliver software programs to the end users. The program (or application) could be accessed and downloaded by anyone with Internet access. Businesses could purchase the software in an on-demand, cost-effective manner, without leaving the office.

The Early 2000s

In 2002, Amazon introduced its web-based retail services. It was the first major business to think of using only 10% of their capacity (which was commonplace at the time) as a problem to be solved. The Cloud Computing Infrastructure Model gave them the flexibility to use their computer's capacity much more efficiently. Soon after, other large organizations followed their example.

In 2006, Amazon launched Amazon Web Services, which offers online services to other websites, or clients. One of Amazon Web Services' sites, called Amazon Mechanical Turk, provides a variety of Cloud-based services including storage, computation and "human intelligence." Another of Amazon Web Services' sites is the Elastic Compute Cloud (EC2), allowing individuals to rent virtual computers and use their own programs and applications.

In the same year, Google launched the Google Docs services. Google Docs was originally based on two separate products, Google Spreadsheets and Writely. Google purchased Writely, which offers renters the ability to save documents, edit documents, and transfer them into blogging systems. (These documents are compatible with Microsoft Word.) Google Spreadsheets (acquired from 2Web Technologies, in 2005) is an Internet-based program allowing users to develop, update, and edit spreadsheets, and to share the data online. An Ajax-based program is used, which is compatible with Microsoft Excel. The spreadsheets can be saved in an HTML format.

In 2007, IBM, Google, and several universities joined forces to develop a server farm for research projects needing both fast processors and huge data sets. The University of Washington was the first to sign up and use resources provided by IBM and Google. Carnegie Mellon University, MIT, Stanford University, the University of Maryland, and the University of California at Berkeley, quickly followed suit. The universities immediately realized computer experiments can be done faster and for less money, if IBM and Google were supporting their research. Since much of the research was focused on problems IBM and Google had interests in, they also benefitted from the arrangement. 2007 was also the year when Netflix launched it's streaming video service, using the Cloud, and provided support for the practice of "binge-watching."

Eucalyptus offered the first AWS API compatible platform, which was used for distributing private Clouds, in 2008. In the same year, NASA's OpenNebula provided the first open-source software for deploying Private and Hybrid Clouds. Many of its most innovative features focused on the needs of major businesses.

In 2011, IBM introduced the IBM SmartCloud framework, in support of Smarter Planet (a cultural thinking project). Then, Apple launched the ICloud, which focuses on storing more personal information (photos, music, videos, etc.). Also, during this year, Microsoft began advertising the Cloud on television, making the general public aware of its ability to store photos, or video, with easy access.

Oracle introduced the Oracle Cloud in 2012, offering the three basics for business, IaaS (Infrastructure-as-a-Service), PaaS (Platform-as-a-Service), and SAAS (Software-as-a-Service).

Cloud Computing tutorial provides basic and advanced concepts of Cloud Computing. Our Cloud Computing tutorial is designed for beginners and professionals.

Cloud computing is a virtualization-based technology that allows us to create, configure, and customize applications via an internet connection. The cloud technology includes a development platform, hard disk, software application, and database.

What is Cloud Computing

The term cloud refers to a network or the internet. It is a technology that uses remote servers on the internet to store, manage, and access data online rather than local drives. The data can be anything such as files, images, documents, audio, video, and more.

There are the following operations that we can do using cloud computing:

- Developing new applications and services
- Storage, back up, and recovery of data

- Hosting blogs and websites
- Delivery of software on demand
- Analysis of data
- Streaming videos and audios

Cloud Architecture

Cloud computing technology is used by both small and large organizations to **store the information** in cloud and **access** it from anywhere at anytime using the internet connection.

Cloud computing architecture is a combination of **service-oriented architecture** and **event-driven architecture**.

Cloud computing architecture refers to the components and subcomponents required for cloud computing. These components typically consist of a front end platform (fat client, thin client, mobile device), back end platforms (servers, storage), a cloud based delivery, and a network (Internet, Intranet, Intercloud). Combined, these components make up cloud computing architecture.

Cloud computing architecture is divided into the following two parts -

- Front End
- Back End

The below diagram shows the architecture of cloud computing -

Components of Cloud Computing Architecture

There are the following components of cloud computing architecture -

1. Client Infrastructure

Client Infrastructure is a Front end component. It provides GUI (Graphical User Interface) to interact with the cloud.

2. Application

The application may be any software or platform that a client wants to access.

3. Service

A Cloud Services manages that which type of service you access according to the client's requirement.

Cloud computing offers the following three type of services:

i. Software as a Service (SaaS) – It is also known as **cloud application services**. Mostly, SaaS applications run directly through the web browser means we do not require to download and install these applications. Some important example of SaaS is given below –

Example: Google Apps, Salesforce Dropbox, Slack, Hubspot, Cisco WebEx.

ii. Platform as a Service (PaaS) – It is also known as **cloud platform services**. It is quite similar to SaaS, but the difference is that PaaS provides a platform for software creation, but using SaaS, we can access software over the internet without the need of any platform.

Example: Windows Azure, Force.com, Magento Commerce Cloud, OpenShift.

iii. Infrastructure as a Service (IaaS) – It is also known as **cloud infrastructure services**. It is responsible for managing applications data, middleware, and runtime environments.

Example: Amazon Web Services (AWS) EC2, Google Compute Engine (GCE), Cisco Metapod.

4. Runtime Cloud

Runtime Cloud provides the **execution and runtime environment** to the virtual machines.

5. Storage

Storage is one of the most important components of cloud computing. It provides a huge amount of storage capacity in the cloud to store and manage data.

6. Infrastructure

It provides services on the **host level**, **application level**, and **network level**. Cloud infrastructure includes hardware and software components such as servers, storage, network devices, virtualization software, and other storage resources that are needed to support the cloud computing model.

7. Management

Management is used to manage components such as application, service, runtime cloud, storage, infrastructure, and other security issues in the backend and establish coordination between them.

8. Security

Security is an in-built back end component of cloud computing. It implements a security mechanism in the back end.

9. Internet

The Internet is medium through which front end and back end can interact and communicate with each other.

Cloud Storage or Infrastructure

- Cloud infrastructure means the hardware and software components.
- These components are server, storage, networking and virtualization software.
- These components are required to support the computing requirements of a cloud computing model.

Components of Cloud infrastructure

1) Hypervisor

- Hypervisor is a firmware or low-level program. It acts as a Virtual Machine Manager.
- It enables to share a physical instance of cloud resources between several customers.

2) Management Software

- Management software assists to maintain and configure the infrastructure.

3) Deployment Software

- Deployment software assists to deploy and integrate the application on the cloud.

4) Network

- Network is the key component of the cloud infrastructure.
- It enables to connect cloud services over the Internet.
- The customer can customize the network route and protocol i.e possible to deliver network as a utility over the Internet.

5) Server

- The server assists to compute the resource sharing and offers other services like resource allocation and deallocation, monitoring the resources, provide the security etc.

6) Storage

- Cloud keeps many copies of storage. Using these copies of resources, it extracts another resource if any one of the resources fails.

Example of Cloud Storage

- **Google's data servers.** Users can edit files using a Google application. Users can also publish documents so that other people can read them or even make edits, which means Google Docs is also an example of cloud computing.
- **Web e-mail** providers like Gmail, Hotmail and Yahoo! Mail store e-mail messages on their own servers. Users can access their e-mail from computers and other devices connected to the Internet.

- Sites like Flickr and Picasa host millions of digital photographs. Their users create online photo albums by uploading pictures directly to the services' servers.
- **YouTube** hosts millions of user-uploaded video files.
- **Social networking** sites like Facebook and MySpace allow members to post pictures and other content. All of that content is stored on the respective site's servers.

Cloud Storage Providers

Types of Cloud Storage

1. Personal Cloud Storage

Also known as mobile cloud storage, personal cloud storage is a subset of public cloud storage that applies to storing an individual's data in the cloud and providing the individual with access to the data from anywhere. It also provides data syncing and sharing capabilities across multiple devices. Apple's iCloud is an example of personal cloud storage.

2. Public Cloud Storage

Public cloud storage is where the enterprise and storage service provider are separate and there aren't any cloud resources stored in the enterprise's data center. The cloud storage provider fully manages the enterprise's public cloud storage.

3. Private Cloud Storage

A form of cloud storage where the enterprise and cloud storage provider are integrated in the enterprise's data center. In private cloud storage, the storage provider has infrastructure in the enterprise's data center that is typically managed

by the storage provider. Private cloud storage helps resolve the potential for security and performance concerns while still offering the advantages of cloud storage.

4. Hybrid Cloud Storage

Hybrid cloud storage is a combination of public and private cloud storage where some critical data resides in the enterprise's private cloud while other data is stored and accessible from a public cloud storage provider.

Infrastructural Constraints

Following are the basic constraints:

- Transparency
- Scalability
- Intelligent Monitoring
- Security

Transparency

- In a Cloud environment, virtualization is the key to share the resources.
- It is impossible to satisfy the demand with single resource or server. Hence, there should be transparency in resources, load balancing and application, so that we can scale the resources on demand.

Scalability

- Scaling up an application is easy but scaling up an application delivery solution is not easy, because it includes configuration overhead or even rearchitecting the network.
- Hence, application delivery is required to be scalable that need the virtual infrastructure so that, resources are provisioned and de-provisioned easily.

Intelligent Monitoring

- Application solution delivery is required to be capable of intelligent monitoring to achieve transparency and scalability.

Security

- In security, the mega data center in the cloud must be securely architected.
- The control node, an entry point in mega data center also requires to be secure.

Advantages of Cloud Storage

- **File Accessibility** – The files can be accessed at any time from any place so long as you have Internet access.
- **Offsite Backup** – Cloud Storage provides organizations with offsite (remote) backups of data which in turn reduces costs.
- **Effective Use of Bandwidth** – Cloud storage uses the bandwidth effectively i.e. instead of sending files to recipients, a web link can be sent through email.
- **Security of Data** – Helps in protecting the data against ransomware or malware as it is secured and needs proper authentication to access the stored data.

Disadvantages of Cloud Storage

- **Dependency on Internet Speed** – If the Internet connection is slow or unstable, we might have problems accessing or sharing the files.
- **Dependency on a Third Party** – A third party service provider (company) is responsible for the data stored and hence it becomes an important pre-requisite in selecting a vendor and to examine the security standards prior investing.
- **High Cost for Huge Data** – Organizations that require a large amount of storage may also find costs increase significantly even after the first few gigabytes of data stored.
- **No/ Minimal Control over Data Storage Framework** – Since the cloud storage framework is entirely managed and monitored by the service provider, the customer has minimal control over it .

Why Cloud Computing Matters

Small as well as large IT companies, follow the traditional methods to provide the IT infrastructure. That means **for any IT company, we need a Server Room that is the basic need of IT companies**.

In that server room, there should be a database server, mail server, networking, firewalls, routers, modem, switches, QPS (Query Per Second means how much queries or load will be handled by the server), configurable system, high net speed, and the maintenance engineers.

To establish such IT infrastructure, we need to spend lots of money. To overcome all these problems and to reduce the IT infrastructure cost, Cloud Computing comes into existence

The characteristics of cloud computing are given below:

1) Agility

The cloud **works in a distributed computing environment**. It shares resources among users and works very fast.

2) High availability and reliability

The availability of servers is high and more reliable because the **chances of infrastructure failure are minimum**.

3) High Scalability

Cloud offers "**on-demand**" **provisioning of resources on a large scale**, without having engineers for peak loads.

4) Multi-Sharing

With the help of cloud computing, **multiple users and applications can work more efficiently** with cost reductions by sharing common infrastructure.

5) Device and Location Independence

Cloud computing enables the users to access systems using a web browser regardless of their location or what device they use e.g. PC, mobile phone, etc. **As infrastructure is off-site** (typically provided by a third-party) **and accessed via the Internet, users can connect from anywhere**.

6) Maintenance

Maintenance of cloud computing applications is easier, since they **do not need to be installed on each user's computer and can be accessed from different places**. So, it reduces the cost also.

7) Low Cost

By using cloud computing, the cost will be reduced because to take the services of cloud computing, **IT company need not to set its own infrastructure** and pay-as-per usage of resources.

8) Services in the pay-per-use mode

Application Programming Interfaces **(APIs) are provided to the users so that they can access services on the cloud** by using these APIs **and pay the charges as per the usage of services**.

Advantages CC

1) Back-up and restore data

Once the data is stored in the cloud, it is easier to get back-up and restore that data using the cloud.

2) Improved collaboration

Cloud applications improve collaboration by allowing groups of people to quickly and easily share information in the cloud via shared storage.

3) Excellent accessibility

Cloud allows us to quickly and easily access store information anywhere, anytime in the whole world, using an internet connection. An internet cloud infrastructure increases organization productivity and efficiency by ensuring that our data is always accessible.

4) Low maintenance cost

Cloud computing reduces both hardware and software maintenance costs for organizations.

5) Mobility

Cloud computing allows us to easily access all cloud data via mobile.

6) IServices in the pay-per-use model

Cloud computing offers Application Programming Interfaces (APIs) to the users for access services on the cloud and pays the charges as per the usage of service.

7) Unlimited storage capacity

Cloud offers us a huge amount of storing capacity for storing our important data such as documents, images, audio, video, etc. in one place.

8) Data security

Data security is one of the biggest advantages of cloud computing. Cloud offers many advanced features related to security and ensures that data is securely stored and handled.

Disadvantages CC

1) Internet Connectivity

As you know, in cloud computing, every data (image, audio, video, etc.) is stored on the cloud, and we access these data through the cloud by using the internet connection. If you do not have good internet connectivity, you cannot access these data. However, we have no any other way to access data from the cloud.

2) Vendor lock-in

Vendor lock-in is the biggest disadvantage of cloud computing. Organizations may face problems when transferring their services from one vendor to another. As different vendors provide different platforms, that can cause difficulty moving from one cloud to another.

3) Limited Control

As we know, cloud infrastructure is completely owned, managed, and monitored by the service provider, so the cloud users have less control over the function and execution of services within a cloud infrastructure.

4) Security

Although cloud service providers implement the best security standards to store important information. But, before adopting cloud technology, you should be aware that you will be sending all your organization's sensitive information to a third party, i.e., a cloud computing service provider. While sending the data on the cloud, there may be a chance that your organization's information is hacked by Hackers.

Companies in the Cloud Today

Many companies are delivering services from the cloud. Some notable examples include the following:

• **Google** — Has a private cloud that it uses for delivering Google Docs and many other services to its users, including email access, document applications, text translations, maps, web analytics, and much more.

• **Microsoft** — Has Microsoft® Office 365® online service that allows for content and business intelligence tools to be moved into the cloud, and Microsoft currently makes its office applications available in a cloud.

• **Salesforce.com** — Runs its application set for its customers in a cloud, and its Force.com and Vmforce.com products provide developers with platforms to build customized cloud services.

Cloud Services

A **cloud service** is any **service** made available to users on demand via the Internet from a **cloud computing** provider's servers as opposed to being provided from a company's own on-premises servers

Simply put, **cloud computing** is the delivery of **computing services** including

- servers
- storage
- databases
- networking
- software
- analytics
- and intelligence

Over the Internet ("the **cloud**") to offer faster innovation, flexible resources, and economies of scale.

- **Virtual IT (information technology)**: Configure and deploy remote external servers as extensions to a company's local IT network.
- **Software**: Use commercial software applications, or develop and remotely host custom-built applications,
- **Network storage**: Network storage archives data across the internet to a provider without needing to know the physical location of storage.

Cloud computing systems all generally are designed to support large numbers of customers and surges in demand.

Software-As-A-Service Models
Cloud services using a software-as-a-service model, or SaaS, provide fully functional programs to end users even though the programs may not be resident on their local computers. Email providers like Gmail and Outlook.com are SaaS applications, as well as just about any computer program that runs inside a browser. As such, SaaS is most familiar to home consumers.

Platform-as-a-Service Models
A SaaS solution sits atop a platform. Vendors that offer platform-as-a-service portfolios generally face corporate clients. PaaS products include virtual servers, operating environments, database environments, and any other middleware component that sits between the hardware and the consumer-facing application.

Infrastructure-as-a-Service Models
Platforms, in turn, sit upon infrastructure. Infrastructure-as-a-service solutions generally get to the level of 'bare metal' — the physical servers, networking components, and device storage necessary to make platforms (and, hence, services) functional. IaaS is popular with corporate clients, with tradeoffs between speed, cost, and privacy that each vendor balances in different ways.

Examples of Cloud Computing Services
Many different vendors offer various types of cloud-computing services:

Amazon EC2 — Virtual IT
Google App Engine — Application hosting
Google Apps and Microsoft Office Online — SaaS
Apple iCloud — Network storage
DigitalOcean — Servers (Iaas/PaaS)

Examples of Cloud Storage (BRIEF)
Ex: Dropbox, Gmail, Facebook

The number of cloud storage providers online seems to grow every day. Each competing over the amount of storage they can provide to clients.

Right now, **Dropbox** is the clear leader in streamlined cloud storage allowing users to access files on any device through its application or website with up to 1 terabyte of free storage.

Google's email service provider Gmail, on the other hand, provides unlimited storage on the cloud. Gmail has revolutionized the way we send emails and largely responsible for the increased usage of email worldwide.

Facebook is a mix of the two, in that it can store an infinite amount of information, images, and videos on your profile. They can then be easily accessed on multiple devices. Facebook goes a step further with their Messenger app, which allows for profiles to exchange data.

Marketing Cloud Platforms
Ex: Maropost for Marketing, Hubspot, Adobe Marketing Cloud

A marketing cloud is an end-to-end digital marketing platform for clients to manage contacts and target leads. Maropost Marketing Cloud combines easy-to-use marketing automation and hyper-targeting of leads. At the same time, ensuring emails actually arrive in the inbox, thanks to its advanced email deliverability capabilities.

In general, marketing clouds fulfill a need for personalization. This is important in a market that demands messaging be "more human." That's why communicating that your brand is here to help, will make all the difference in closing.

Cloud Computing in Education
Ex: SlideRocket, Ratatype, Amazon Web Services

Education is increasingly adopting advanced technology because students already are. So, in an effort to modernize classrooms, educators have introduced e-learning software like SlideRocket.

SlideRocket is a platform that students can use to build presentations and submit them. Students can even present them through web conferencing all on the cloud. Another tool teachers use is Ratatype, which helps students learn to type faster and offers online typing tests to track their progress.

For school administration, Amazon's AWS Cloud for K12 and Primary Education features a virtual desktop infrastructure (VDI) solution. Through the cloud, allows instructors and students to access teaching and learning software on multiple devices.

Examples of Cloud Computing in Healthcare
Ex: ClearDATA, Dell's Secure Healthcare Cloud, IBM Cloud

Cloud computing lets nurses, physicians, and administrators share information quickly from anywhere. It also saves on costs by allowing large data files to be shared instantly for maximum convenience. This is a major boost for efficiency.

Ultimately, cloud technology ensures patients receive the best possible care without unnecessary delay. The patient's condition can also be updated in seconds through remote conferencing.

However, many modern hospitals have yet to implement cloud computing but are forecasted to do so in the near future.

Examples of Cloud Computing for Government
Uses: IT consolidation, shared services, citizen services

The U.S. government and military were early adopters of cloud computing. The U.S. Federal Cloud Computing Strategy, introduced under the Obama administration, was instituted to accelerate cloud adoption in all departments.

According to the strategy: "focus will shift from the technology itself to the core competencies and mission of the agency."

The U.S. government's cloud incorporates social, mobile and analytics technologies. However, they must adhere to strict compliance and security measures (FIPS, FISMA, and FedRAMP). This is to protect against cyber threats both domestic and abroad.

Cloud computing is the answer for any business struggling to stay organized, increase ROI, or grow their email lists. Maropost has the digital marketing solutions you need to transform your business.

Web-Based Application

Cloud services, in the form of centralized web-based applications, also appeal to the IT professional. One instance of an application hosted in the cloud is cheaper and easier to manage than individual copies of similar software installed on each user's desktop PC. Upgrading a cloud app only has to be done one time, where upgrading traditional software has to be done for each PC on which that software is installed. Then, of course, we have the promise of cloud-enabled collaboration, which just can't be done with traditional desktop apps.

How Websites Work

Let's start with the most obvious way of using the internet: You visit a website like **www.biher.ac.in**.

The computer user initiates the request for a web page . The browser then sends this request to the **DNS Sever** .

The DNS Server locates the Web Server where the website is hosted and translates the URL into IP address of the **Web Server**.

The Web Sever then responds with the requested web page and send this to the web browser that has initiated the request . The web browser then displays the web page to the user .

The **web server** also needs to have the operating system , **DBMS** (Database Management System) to manage the database stored on the server and server side programming language (scripting language) .

The server software enables the web server to offer various services and respond to the http requests sent by the client machines .

The web server operating system supports the client server architecture and provides a secured hosting environment to the websites hosted on the web server .

Web Server Solution Stack : LAMP , WAMP , XAMPP

A web server software package (Solution Stack) consist of four components bundled together in addition to the Sever hardware (Computer system) .

These four server components include the operating system, Http web server, **DBMS** (Database) and a server side scripting language. All these four components together provide server side run-time environment.

Some of the most popular web server solution stack (bundled together web server software package) include LAMP , XAMPP and WAMP .

The LAMP web server package include combinations of Linux, Apache, MySQL (RDBMS), and PHP, abbreviated and named in the same order.

Another commonly used web server is WAMP for Windows based platforms . The WAMP includes Windows, Apache, MySQL and PHP.

Web based application refers to any program that is accessed over a network using HTTP (Hyper Text Transfer Protocol). **Web based application** often run inside a **web browser**. We can often say that **web based application** are user **based** as we can download some part of the **application** to view it on the desktop

Components of Web application

All **web**-based database **applications** have three primary **components**: A **web** browser (or client), a **web application** server, and a database server.

How a web application works

Web applications are usually coded in browser-supported language such as JavaScript and HTML as these languages rely on the browser to render the program executable. Some of the applications are dynamic, requiring server-side processing. Others are completely static with no processing required at the server.

The web application requires a web server to manage requests from the client, an application server to perform the tasks requested, and, sometimes, a database to store the information. Application server technology ranges from ASP.NET, ASP and ColdFusion, to PHP and JSP.

Here's what a typical web application flow looks like:

1. **User** triggers a request to the **web server** over the **Internet**, either through a web browser or the application's user interface
2. **Web server** forwards this request to the appropriate **web application server**

3. **Web application server** performs the requested task – such as querying the **database** or processing the data – then generates the results of the requested data
4. **Web application server** sends results to the **web server** with the requested information or processed data
5. **Web server** responds back to the client with the requested information that then appears on the user's display

Example

Web applications include online forms, shopping carts, word processors, spreadsheets, video and photo editing, file conversion, file scanning, and email programs such as Gmail, Yahoo and AOL. Popular **applications** include Google Apps and Microsoft 365

Advantages

Advantages of **Web Applications**. **Web Applications** deliver many business benefits compared to office **based** solutions. Reduce business costs - less time spent talking to customers over the phone; eliminate printed materials; allow users to update their own details. Centralised data is secure and easy to backup.

UNIT-II
CLOUD ARCHITECTURE AND MODEL

- Technologies for Network Based system
- System Models for Distributed and Cloud Computing
- NIST Cloud Computing Reference Architecture
- **Cloud models** :
- Characteristics - Cloud Models (IaaS, PaaS, SaaS)
- Public vs. Private Cloud
- Cloud Solutions
- Cloud ecosystem
- Service Management
- Computing on demand.

Introduction

FIGURE 1.1

Evolutionary trend toward parallel, distributed, and cloud computing with clusters, MPPs, P2P networks, grids, clouds, web services, and the Internet of Things.

Technologies for Network Based system

With the concept of scalable computing under our belt, it's time to explore hardware, software, and network technologies for distributed computing system design and applications. In particular, we will focus on viable approaches to building distributed operating systems for handling massive parallelism in a distributed environment.

- Multicore CPUs and Multithreading Technologies
- GPU Computing to Exascale and Beyond
- Memory, Storage, and Wide-Area Networking
- Virtual Machines and Virtualization Middleware
- Data Center Virtualization for Cloud Computing

1.Multicore CPUs and Multithreading Technologies

1.a.Advance in CPU Processor

- Processor speed is measured in millions of instructions per second (MIPS) and network bandwidth is measured in megabits per second (Mbps) or gigabits per second (Gbps). The unit GE refers to 1 Gbps Ethernet bandwidth.

- Advances in CPU Processors advanced CPUs or microprocessor chips assume a multicore architecture with dual, quad, six, or more processing cores. These processors exploit parallelism at ILP and TLP levels.

- ILP mechanisms include multiple-issue superscalar architecture, dynamic branch prediction, and speculative execution, among others. These ILP techniques demand hardware and compiler support.

FIGURE 1.4
Improvement in processor and network technologies over 33 years.

- In addition, DLP and TLP are highly explored in graphics processing units (GPUs) that adopt a many-core architecture with hundreds to thousands of simple cores.

- Both multi-core CPU and many-core GPU processors can handle multiple instruction threads at different magnitudes today.

FIGURE 1.5

Schematic of a modern multicore CPU chip using a hierarchy of caches, where L1 cache is private to each core, on-chip L2 cache is shared and L3 cache or DRAM Is off the chip.

- Figure 1.5 shows the architecture of a typical multicore processor.
- Each core is essentially a processor with its own private cache (L1 cache).
- Multiple cores are housed in the same chip with an L2 cache that is shared by all cores.
- In the future, multiple CMPs could be built on the same CPU chip with even the L3 cache on the chip.
- Multicore and multithreaded CPUs are equipped with many high-end processors, including the Intel i7, Xeon, AMD Opteron, Sun Niagara, IBM Power 6, and X cell processors. Each core could be also multithreaded.

1.b. Multicore CPU and Many-Core GPU Architectures

- Multicore CPUs may increase from the tens of cores to hundreds or more in the future.
- But the CPU has reached its limit in terms of exploiting massive DLP due to the aforementioned memory wall problem. This has triggered the development of many-core GPUs with hundreds or more thin cores.

- Both IA-32 and IA-64 instruction set architectures are built into commercial CPUs. Now, x-86 processors have been extended to serve HPC and HTC systems in some high-end server processors.
- Many RISC processors have been replaced with multicore x-86 processors and many- core GPUs in the Top 500 systems. This trend indicates that x-86 upgrades will dominate in data centers and supercomputers.
- The GPU also has been applied in large clusters to build supercomputers in MPPs.

F

FIGURE 1.6

Five micro-architectures in modern CPU processors, that exploit ILP and TLP supported by multicore and multithreading technologies.

1.c. Multithreading Technology

- Consider in Figure 1.6 the dispatch of five independent threads of instructions to four pipelined data paths (functional units) in each of the following five processor categories,
- from left to right: a four-issue superscalar processor, a fine-grain multithreaded processor, a coarse-grain multithreaded processor, a two-core CMP, and a simultaneous multithreaded (SMT) processor.
- The superscalar processor is single-threaded with four functional units.
- Each of the three multithreaded processors is four-way multithreaded over four functional data paths.
- In the dual-core processor, assume two processing cores, each a single-threaded two-way superscalar processor.

- Instructions from different threads are distinguished by specific shading patterns for instructions from five independent threads.
- Typical instruction scheduling patterns are shown here. Only instructions from the same thread are executed in a superscalar processor.
- Fine-grain multithreading switches the execution of instructions from different threads per cycle.
- Course-grain multithreading executes many instructions from the same thread for quite a few cycles before switching to another thread.
- The multicore CMP executes instructions from different threads completely.
- The SMT allows simultaneous scheduling of instructions from different threads in the same cycle.
- These execution patterns closely mimic an ordinary program.
- The blank squares correspond to no. of available instructions for an instruction data path at a particular processor cycle.
- More blank cells imply lower scheduling efficiency. The maximum ILP or maximum TLP is difficult to achieve at each processor cycle.

2. GPU Computing to Exascale and Beyond

FIGURE 1.7
The use of a GPU along with a CPU for massively parallel execution in hundreds or thousands of processing cores.

- Figure 1.7 shows the interaction between a CPU and GPU in performing parallel execution of floating-point operations concurrently.
- The CPU is the conventional multicore processor with limited parallelism to exploit.
- The GPU has a many-core architecture that has hundreds of simple processing cores organized as multiprocessors.
- Each core can have one or more threads.
- Essentially, the CPU's floating-point kernel computation role is largely offloaded to the many- core GPU.
- The CPU instructs the GPU to perform massive data processing.

- The bandwidth must be matched between the on-board main memory and the on-chip GPU memory.
- This process is carried out in NVIDIA's CUDA programming using the GeForce 8800 or Tesla and Fermi GPUs.

FIGURE 1.8

NVIDIA Fermi GPU built with 16 streaming multiprocessors (SMs) of 32 CUDA cores each; only one SM Is shown. More details can be found also in [49].

The Fermi GPU is a newer generation of GPU, first appearing in 2011. The Tesla or Fermi GPU can be used in desktop workstations to accelerate floating-point calculations or for building large-scale data cen-ters. The architecture shown is based on a 2009 white paper by NVIDIA. There are 32 CUDA cores per SM. Only one SM is shown in Figure 1.8.

Each CUDA core has a simple pipelined integer ALU and an FPU that can be used in parallel. Each SM has 16 load/store units allowing source and destination addresses to be calculated for 16 threads per clock. There are four special function units (SFUs) for executing transcendental instructions.

How GPUs Work

- Today, the NVIDIA GPU has been upgraded to 128 cores on a single chip. Furthermore, each core on a GPU can handle eight threads of instructions.
- This translates to having up to 1,024 threads executed concurrently on a single GPU.
- This is true massive parallelism, compared to only a few threads that can be handled by a conventional CPU.
- The CPU is optimized for latency caches, while the GPU is optimized to deliver much higher throughput with explicit management of on-chip memory.
- GPU offloads the CPU from all data-intensive calculations, not just those that are related to video processing.
- Conventional GPUs are widely used in mobile phones, game consoles, embedded systems, PCs, and servers.
- The NVIDIA CUDA Tesla or Fermi is used in GPU clusters or in HPC systems for parallel processing of massive floating-pointing data.

GPU Programming Model

- Four challenges are identified for exascale computing:
- (1) energy and power,
- (2) memory and storage,
- (3) concurrency and locality
- (4) system resiliency (system elasticity)

Power Efficiency of the GPU

- Power constrains what we can put in a CPU or GPU chip.
- Dally has estimated that the CPU chip consumes about 2 nJ/instruction, while the GPU chip requires 200 pJ/instruction, which is 1/10 less than that of the CPU.
- The CPU is optimized for latency in caches and memory, while the GPU is optimized for throughput with explicit management of on-chip memory.

3. Memory, Storage, and Wide-Area Networking

3.a. Memory Technology

- The capacity increase of disk arrays will be even greater in the years to come.
- Faster processor speed and larger memory capacity result in a wider gap between processors and memory.
- The memory wall may become even worse a problem limiting the CPU performance in the future.

3.b.Disks and Storage Technology

- Eventually, power consumption, cooling, and packaging will limit large system development.
- Power increases linearly with respect to clock frequency and quadratic ally with respect to voltage applied on chips.
- Clock rate cannot be increased indefinitely.
- Lowered voltage supplies are very much in demand.

3.c.System-Area Interconnects

- The nodes in small clusters are mostly interconnected by an Ethernet switch or a local area network (LAN).
- As Figure 1.11 shows, a LAN typically is used to connect client hosts to big servers.
- A storage area network (SAN) connects servers to network storage such as disk arrays.
- Network attached storage (NAS) connects client hosts directly to the disk arrays.
- All three types of networks often appear in a large cluster built with commercial network components.
- If no large distributed storage is shared, a small cluster could be built with a multiport Gigabit Ethernet switch plus copper cables to link the end machines.
- All three types of networks are commercially available.

FIGURE 1.11

Three interconnection networks for connecting servers, client hosts, and storage devices; the LAN connects client hosts and servers, the SAN connects servers with disk arrays, and the NAS connects clients with large storage systems in the network environment.

3.d. Wide-Area Networking

- The lower curve in Figure 1.10 plots the rapid growth of Ethernet bandwidth from 10 Mbps in 1979 to 1 Gbps in 1999, and 40 ~ 100 GE in 2011.
- It has been speculated that 1 Tbps network links will become available by 2013.
- Network links with 1,000, 1,000, 100, 10, and 1 Gbps bandwidths were reported, respectively, for international, national, organization, optical desktop, and copper desktop connections in 2006.
- An increase factor of two per year on network performance was reported, which is faster than Moore's law on CPU speed doubling every 18 months.
- The implication is that more computers will be used concurrently in the future.
- High-bandwidth networking increases the capability of building massively distributed systems.
- Most data centers are using Gigabit Ethernet as the interconnect in their server clusters.

FIGURE 1.10

Improvement in memory and disk technologies over 33 years. The Seagate Barracuda XT disk has a capacity of 3 TB in 2011.

4.Virtual Machines and Virtualization Middleware

- A conventional computer has a single OS image. This offers a rigid architecture that tightly couples application software to a specific hardware platform.

- Some software running well on one machine may not be executable on another platform with a different instruction set under a fixed OS.

- Virtual machines (VMs) offer novel solutions to underutilized resources, application inflexibility, software manageability, and security concerns in existing physical machines.

- Today, to build large clusters, grids, and clouds, we need to access large amounts of computing, storage, and networking resources in a virtualized manner.

- We need to aggregate those resources, and hopefully, offer a single system image.

- In particular, a cloud of provisioned resources must rely on virtualization of processors, memory, and I/O facilities dynamically.

FIGURE 1.12

Three VM architectures in (b), (c), and (d), compared with the traditional physical machine shown in (a).

(Courtesy of M. Abde-Majeed and S. Kulkarni, 2009 USC)

4.a.Virtual Machines

- Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are multiplexed in the same hardware machine.
- Virtual Machine (VM) is not a real machine, but it can be created from another physical machine.

- More than one VM may share the resources (CPU, Memory, Hard Disk & Network Resources) of the same physical machine.
- One physical machine may have more than one VM.
- VM used to create different execution environment which may entirely different from physical host execution environment.

4.b.VM Primitive Operations

- First, the VMs can be **multiplexed between hardware machines**, as shown in Figure 1.13(a).
- Second, a VM can be **suspended and stored in stable storage**, as shown in Figure 1.13(b).
- Third, a **suspended VM can be resumed or provisioned to a new hardware platform**, as shown in Figure 1.13(c).
- Finally, a VM can be migrated from one hardware platform to another, as shown in Figure 1.13(d).
- VM approach will significantly **enhance the utilization of server resources**.
- Multiple server functions can be consolidated on the same hardware platform to achieve higher system efficiency.

4.c. Virtual Infrastructures

- A virtual infrastructure is a software-based IT infrastructure being hosted on another physical infrastructure and meant to be distributed as a service as in cloud computing's infrastructure as a service (IaaS) delivery model.
- Virtual infrastructure is what connects resources to distributed applications.
- It is a dynamic mapping of system resources to specific applications.
- The result is decreased costs and increased efficiency and responsiveness.

5.Data Center Virtualization for Cloud Computing

Data Center Growth and Cost Breakdown

- A large data center may be built with thousands of servers.
- Smaller data centers are typically built with hundreds of servers.
- The cost to build and maintain data center servers has increased over the years.

Low-Cost Design Philosophy

- High-end switches or routers may be too cost-prohibitive for building data centers.
- Thus, using high-bandwidth networks may not fit the economics of cloud computing.

Convergence of Technologies

- Essentially, cloud computing is enabled by the convergence of technologies in four areas:-
  (1) Hardware virtualization and multi-core chips,
  (2) Utility and grid computing,
  (3) SOA, Web 2.0, and WS Mashups, and
  (4) Autonomic computing and data center automation.

System Models for Distributed and Cloud Computing

- *Clusters of Cooperative Computers*
- *Grid Computing Infrastructures*
- *Peer-to-Peer Network Families*
- *Cloud Computing over the Internet*

- Distributed and cloud computing systems are built over a large number of autonomous computer nodes.
- These node machines are interconnected by SANs, LANs, or WANs in a hierarchical manner.

A.Clusters of Cooperative Computers

- A computing cluster consists of interconnected stand-alone computers which work cooperatively as a single integrated computing resource

Cluster Architecture

FIGURE 1.15

A cluster of servers interconnected by a high-bandwidth SAN or LAN with shared I/O devices and disk arrays; the cluster acts as a single computer attached to the Internet.

Single-System Image

The architecture of a typical server cluster built around a low-latency, high bandwidth interconnection network. This network can be as simple as a SAN (e.g., Myrinet) or a LAN (e.g., Ethernet)

- The cluster is connected to the Internet via a virtual private network (VPN) gateway. **The gateway IP address locates the cluster.**
- An ideal cluster should merge multiple system images into a **S**ingle **S**ystem **I**mage (SSI).
- SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource.
- SSI makes the cluster appear like a single machine to the user.

Table 1.3 Critical Cluster Design Issues and Feasible Implementations

Features	Functional Characterization	Feasible Implementations
Availability and Support	Hardware and software support for sustained HA in cluster	Failover, failback, check pointing, rollback recovery, nonstop OS, etc.
Hardware Fault Tolerance	Automated failure management to eliminate all single points of failure	Component redundancy, hot swapping, RAID, multiple power supplies, etc.
Single System Image (SSI)	Achieving SSI at functional level with hardware and software support, middleware, or OS extensions	Hardware mechanisms or middleware support to achieve DSM at coherent cache level
Efficient Communications	To reduce message-passing system overhead and hide latencies	Fast message passing, active messages, enhanced MPI library, etc.
Cluster-wide Job Management	Using a global job management system with better scheduling and monitoring	Application of single-job management systems such as LSF, Codine, etc.
Dynamic Load Balancing	Balancing the workload of all processing nodes along with failure recovery	Workload monitoring, process migration, job replication and gang scheduling, etc.
Scalability and Programmability	Adding more servers to a cluster or adding more clusters to a grid as the workload or data set increases	Use of scalable interconnect, performance monitoring, distributed execution environment, and better software tools

Hardware, Software, and Middleware Support

- Special cluster middleware supports are needed to create SSI or **H**igh **A**vailability (HA).
- Both sequential and parallel applications can run on the cluster
- For example, distributed memory has multiple images. Users may want all distributed memory to be shared by all servers by forming **D**istributed **S**hared **M**emory (DSM).

Major Cluster Design Issues

- Middleware or OS extensions were developed at the user space to achieve SSI at selected functional levels.
- The cluster benefits come from scalable performance, efficient message passing, high system availability, seamless fault tolerance, and cluster-wide job management.

B.Grid Computing Infrastructures

- Internet services such as the Telnet command enables a local computer to connect to a remote computer.
- A web service such as HTTP enables remote access of remote web pages.
- Grid computing is envisioned to allow close interaction among applications running on distant computers simultaneously.

Computational Grids

- A computing grid offers an infrastructure that couples computers, software/middleware, special instruments, and people and sensors together.
- The grid is often constructed across LAN, WAN, or Internet backbone networks at a regional, national, or global scale.
- The computers used in a grid are primarily workstations, servers, clusters, and supercomputers.
- Personal computers, laptops, and PDAs can be used as access devices to a grid system.
- At the server end, the grid is a network. At the client end, we see wired or wireless terminal devices.
- The grid integrates the computing, communication, contents, and transactions as rented services.

FIGURE 1.16
Computational grid or data grid providing computing utility, data, and information services through resource sharing and cooperation among participating organizations.

- Figure 1.16 shows an example computational grid built over multiple resource sites owned by different organizations.

- o The resource sites offer complementary computing resources, including workstations, large servers, a mesh of processors, and Linux clusters to satisfy a chain of computational needs.

- The grid is built across various IP broadband networks including LANs and WANs already used by enterprises or organizations over the Internet.

- The grid is presented to users as an integrated resource pool as shown in the upper half of the figure.

- At the server end, the grid is a network. At the client end, we see wired or wireless terminal devices. The grid integrates t h e computing , communication , contents, and transactions

- Enterprises and consumers form the user base, which then defines the usage trends and service characteristics.

- The NSF TeraGrid in US, EGEE in Europe, and ChinaGrid in China for various distributed scientific grid applications.

Grid Families

Grid systems are classified in essentially two categories: -

- Computational or Data Grids and
- P2P Grids.

Computing or data grids are built primarily at the national level.

Table 1.4 Two Grid Computing Infrastructures and Representative Systems

Design Issues	Computational and Data Grids	P2P Grids
Grid Applications Reported	Distributed supercomputing, National Grid initiatives, etc.	Open grid with P2P flexibility, all resources from client machines
Representative Systems	TeraGrid built in US, ChinaGrid in China, and the e-Science grid built in UK	JXTA, FightAid@home, SETI@home
Development Lessons Learned	Restricted user groups, middleware bugs, protocols to acquire resources	Unreliable user-contributed resources, limited to a few apps

C.Cloud Computing over the Internet

- A cloud is a pool of virtualized computer resources. A cloud can host a variety of different workloads, including batch-style backend jobs and interactive and user facing applications

- The cloud **supports** redundant, self-recovering, highly scalable programming models that allow workloads to recover from many unavoidable hardware/software failures.

Internet Clouds

- Cloud computing applies a virtualized platform with elastic resources on demand by provisioning hardware, software, and data sets dynamically.

- The idea is to move desktop computing to a service-oriented platform using server clusters and huge databases at data centers.

The Cloud Landscape

- Traditional systems have encountered several performance bottlenecks: constant system maintenance, poor utilization, and increasing costs associated with hardware/software upgrades.
- Cloud computing as an on-demand computing paradigm resolves or relieves us from these problems.

1. Infrastructure as a Service (IaaS)

- This model puts together infrastructures demanded by users—namely **servers, storage, networks, and the data center fabric.**

- The user can deploy and run on multiple VMs running guest Operating Systems on specific applications.

- The user does not manage or control the underlying cloud infrastructure, but can specify when to request and release the needed resources.

2. Platform as a Service (PaaS)

- This model enables the user to deploy user-built applications onto a virtualized cloud platform.

- PaaS includes **middleware, databases, development tools, and some runtime support such as Web 2.0 and Java.**

3. Software as a Service (SaaS)

- This refers to browser-initiated application software over thousands of paid cloud customers.

- The SaaS model applies to **business processes, industry applications, consumer relationship management (CRM), enterprise resources planning (ERP), human resources (HR), and collaborative applications.**

- Internet clouds offer **four deployment models: -**
    1) Private,
    2) Public,
    3) Community, and
    4) Hybrid

Reasons to adapt the cloud for upgraded Internet applications and web services:-

1. Desired location in areas with protected space and higher energy efficiency.

2. Sharing of peak-load capacity among a large pool of users, improving overall utilization.

3. Separation of infrastructure maintenance duties from domain-specific application development.

4. Significant reduction in cloud computing cost, compared with traditional computing paradigms.

5. Cloud computing programming and application development.

6. Service and data discovery and content/service distribution.

7. Privacy, security, copyright, and reliability issues.

8. Service agreements, business models, and pricing policies.

UNIT-III CLOUD INFRASTRUCTURE

4.3 ARCHITECTURAL DESIGN OF COMPUTE AND STORAGE CLOUDS

This section presents basic cloud design principles. We start with basic cloud architecture to process massive amounts of data with a high degree of parallelism. Then we study virtualization support, resource provisioning, infrastructure management, and performance modeling.

4.3.1 A Generic Cloud Architecture Design

An Internet cloud is envisioned as a public cluster of servers provisioned on demand to perform collective web services or distributed applications using data-center resources. In this section, we will discuss cloud design objectives and then present a basic cloud architecture design.

4.3.1.1 Cloud Platform Design Goals

Scalability, virtualization, efficiency, and reliability are four major design goals of a cloud computing platform. Clouds support Web 2.0 applications. Cloud management receives the user request, finds the correct resources, and then calls the provisioning services which invoke the resources in the cloud. The cloud management software needs to support both physical and virtual machines. Security in shared resources and shared access of data centers also pose another design challenge.

The platform needs to establish a very large-scale HPC infrastructure. The hardware and software systems are combined to make it easy and efficient to operate. System scalability can benefit from cluster architecture. If one service takes a lot of processing power, storage capacity, or network traffic, it is simple to add more servers and bandwidth. System reliability can benefit from this architecture. Data can be put into multiple locations. For example, user e-mail can be put in three disks which expand to different geographically separate data centers. In such a situation, even if one of the data centers crashes, the user data is still accessible. The scale of the cloud architecture can be easily expanded by adding more servers and enlarging the network connectivity accordingly.

4.3.1.2 Enabling Technologies for Clouds

The key driving forces behind cloud computing are the ubiquity of broadband and wireless networking, falling storage costs, and progressive improvements in Internet computing software. Cloud users are able to demand more capacity at peak demand, reduce costs, experiment with new services, and remove unneeded capacity, whereas service providers can increase system utilization via multiplexing, virtualization, and dynamic resource provisioning. Clouds are enabled by the progress in hardware, software, and networking technologies summarized in Table 4.3.

Scanned with CamScanner

Table 4.3 Cloud-Enabling Technologies in Hardware, Software, and Networking

Technology	Requirements and Benefits
Fast platform deployment	Fast, efficient, and flexible deployment of cloud resources to provide dynamic computing environment to users
Virtual clusters on demand	Virtualized cluster of VMs provisioned to satisfy user demand and virtual cluster reconfigured as workload changes
Multitenant techniques	SaaS for distributing software to a large number of users for their simultaneous use and resource sharing if so desired
Massive data processing	Internet search and web services which often require massive data processing, especially to support personalized services
Web-scale communication	Support for e-commerce, distance education, telemedicine, social networking, digital government, and digital entertainment applications
Distributed storage	Large-scale storage of personal records and public archive information which demands distributed storage over the clouds
Licensing and billing services	License management and billing services which greatly benefit all types of cloud services in utility computing

These technologies play instrumental roles in making cloud computing a reality. Most of these technologies are mature today to meet increasing demand. In the hardware area, the rapid progress in multicore CPUs, memory chips, and disk arrays has made it possible to build faster data centers with huge amounts of storage space. Resource virtualization enables rapid cloud deployment and disaster recovery. *Service-oriented architecture* (SOA) also plays a vital role.

Progress in providing SaaS, Web 2.0 standards, and Internet performance have all contributed to the emergence of cloud services. Today's clouds are designed to serve a large number of tenants over massive volumes of data. The availability of large-scale, distributed storage systems is the foundation of today's data centers. Of course, cloud computing is greatly benefitted by the progress made in license management and automatic billing techniques in recent years.

4.3.1.3 A Generic Cloud Architecture

Figure 4.14 shows a security-aware cloud architecture. The Internet cloud is envisioned as a massive cluster of servers. These servers are provisioned on demand to perform collective web services or distributed applications using data-center resources. The cloud platform is formed dynamically by provisioning or deprovisioning servers, software, and database resources. Servers in the cloud can be physical machines or VMs. User interfaces are applied to request services. The provisioning tool carves out the cloud system to deliver the requested service.

In addition to building the server cluster, the cloud platform demands distributed storage and accompanying services. The cloud computing resources are built into the data centers, which are typically owned and operated by a third-party provider. Consumers do not need to know the underlying technologies. In a cloud, software becomes a service. The cloud demands a high degree of trust of massive amounts of data retrieved from large data centers. We need to build a framework to process large-scale data stored in the storage system. This demands a distributed file system over the database system. Other cloud resources are added into a cloud platform, including storage area networks (SANs), database systems, firewalls, and security devices. Web service providers offer

Scanned with CamScanner

FIGURE 4.14

A security-aware cloud platform built with a virtual cluster of VMs, storage, and networking resources over the data-center servers operated by providers.

(Courtesy of K. Hwang and D. Li, 2010 [36])

special APIs that enable developers to exploit Internet clouds. Monitoring and metering units are used to track the usage and performance of provisioned resources.

The software infrastructure of a cloud platform must handle all resource management and do most of the maintenance automatically. Software must detect the status of each node server joining and leaving, and perform relevant tasks accordingly. Cloud computing providers, such as Google and Microsoft, have built a large number of data centers all over the world. Each data center may have thousands of servers. The location of the data center is chosen to reduce power and cooling costs. Thus, the data centers are often built around hydroelectric power. The cloud physical platform builder is more concerned about the performance/price ratio and reliability issues than shear speed performance.

In general, private clouds are easier to manage, and public clouds are easier to access. The trends in cloud development are that more and more clouds will be hybrid. This is because many cloud applications must go beyond the boundary of an intranet. One must learn how to create a private cloud and how to interact with public clouds in the open Internet. Security becomes a critical issue in safeguarding the operation of all cloud types. We will study cloud security and privacy issues at the end of this chapter.

Scanned with CamScanner

4.3.2 Layered Cloud Architectural Development

The architecture of a cloud is developed at three layers: infrastructure, platform, and application, as demonstrated in Figure 4.15. These three development layers are implemented with virtualization and standardization of hardware and software resources provisioned in the cloud. The services to public, private, and hybrid clouds are conveyed to users through networking support over the Internet and intranets involved. It is clear that the infrastructure layer is deployed first to support IaaS services. This infrastructure layer serves as the foundation for building the platform layer of the cloud for supporting PaaS services. In turn, the platform layer is a foundation for implementing the application layer for SaaS applications. Different types of cloud services demand application of these resources separately.

The infrastructure layer is built with virtualized compute, storage, and network resources. The abstraction of these hardware resources is meant to provide the flexibility demanded by users. Internally, virtualization realizes automated provisioning of resources and optimizes the infrastructure management process. The platform layer is for general-purpose and repeated usage of the collection of software resources. This layer provides users with an environment to develop their applications, to test operation flows, and to monitor execution results and performance. The platform should be able to assure users that they have scalability, dependability, and security protection. In a way, the virtualized cloud platform serves as a "system middleware" between the infrastructure and application layers of the cloud.

FIGURE 4.15

Layered architectural development of the cloud platform for IaaS, PaaS, and SaaS applications over the Internet.

Scanned with CamScanner

(1) Market - oriented cloud Architecture :-

⇒ The cloud is basically built with the following entities :-

(i) Users or brokers - acting on user's behalf submit service requests from anywhere in the world to the data center and cloud to be processed.

(ii) SLA Resource allocator - The SLA resource allocator acts as the interface between the data center / cloud service provider & external users/ brokers.

Scanned with CamScanner

(iii) Request Examiner - The request examiner ensures that there is no overloading of resources whereby many service requests cannot be fulfilled successfully due to limited resources.

(iv) Service Request monitor - In order to make resource allocation decisions effectively. Then it assigns requests to VMs and determines resource entitlements for allocated VMs. Keeps track of the execution progress of service requests.

(V) Pricing : Serves as a basis for managing the supply & demand of computing resources within the data center & facilitates in prioritizing resource allocations effectively.

(vi) Accounting mechanism : maintains the actual usage of resources by requests so that the final cost can be computed & charged to users.

(vii) VM monitor mechanism : keeps track of the availability of VMs and their resource entitlements.

(viii) Dispatcher mechanism starts the execution of accepted service requests on allocated VM's.

Scanned with CamScanner

FIGURE 4.16

Market-oriented cloud architecture to expand/shrink leasing of resources with variation in QoS/demand from users.

(Courtesy of Rai Buyya, et al. [11])

Scanned with CamScanner

4.3.4 Architectural Design Challenges

In this section, we will identify six open challenges in cloud architecture development. Armbrust, et al. [4] have observed some of these topics as both obstacles and opportunities. Plausible solutions to meet these challenges are discussed shortly.

4.3.4.1 Challenge 1—Service Availability and Data Lock-in Problem

The management of a cloud service by a single company is often the source of single points of failure. To achieve HA, one can consider using multiple cloud providers. Even if a company has multiple data centers located in different geographic regions, it may have common software infrastructure and accounting systems. Therefore, using multiple cloud providers may provide more protection from failures. Another availability obstacle is distributed *denial of service* (DDoS) attacks. Criminals threaten to cut off the incomes of SaaS providers by making their services unavailable. Some utility computing services offer SaaS providers the opportunity to defend against DDoS attacks by using quick scale-ups.

Software stacks have improved interoperability among different cloud platforms, but the APIs itself are still proprietary. Thus, customers cannot easily extract their data and programs from one site to run on another. The obvious solution is to standardize the APIs so that a SaaS developer can deploy services and data across multiple cloud providers. This will rescue the loss of all data due to the failure of a single company. In addition to mitigating data lock-in concerns, standardization of APIs enables a new usage model in which the same software infrastructure can be used in both public and private clouds. Such an option could enable "surge computing," in which the public cloud is used to capture the extra tasks that cannot be easily run in the data center of a private cloud.

SDU

4.3.4.2 Challenge 2—Data Privacy and Security Concerns

Current cloud offerings are essentially public (rather than private) networks, exposing the system to more attacks. Many obstacles can be overcome immediately with well-understood technologies such as encrypted storage, virtual LANs, and network middleboxes (e.g., firewalls, packet filters). For example, you could encrypt your data before placing it in a cloud. Many nations have laws requiring SaaS providers to keep customer data and copyrighted material within national boundaries.

Traditional network attacks include buffer overflows, DoS attacks, spyware, malware, rootkits, Trojan horses, and worms. In a cloud environment, newer attacks may result from hypervisor malware, guest hopping and hijacking, or VM rootkits. Another type of attack is the man-in-the-middle

Scanned with CamScanner

attack for VM migrations. In general, passive attacks steal sensitive data or passwords. Active attacks may manipulate kernel data structures which will cause major damage to cloud servers. We will study all of these security and privacy problems on clouds in Section 4.5.

4.3.4.3 Challenge 3—Unpredictable Performance and Bottlenecks

Multiple VMs can share CPUs and main memory in cloud computing, but I/O sharing is problematic. For example, to run 75 EC2 instances with the STREAM benchmark requires a mean bandwidth of 1,355 MB/second. However, for each of the 75 EC2 instances to write 1 GB files to the local disk requires a mean disk write bandwidth of only 55 MB/second. This demonstrates the problem of I/O interference between VMs. One solution is to improve I/O architectures and operating systems to efficiently virtualize interrupts and I/O channels.

Internet applications continue to become more data-intensive. If we assume applications to be "pulled apart" across the boundaries of clouds, this may complicate data placement and transport. Cloud users and providers have to think about the implications of placement and traffic at every level of the system, if they want to minimize costs. This kind of reasoning can be seen in Amazon's development of its new CloudFront service. Therefore, data transfer bottlenecks must be removed, bottleneck links must be widened, and weak servers should be removed. We will study performance issues in Chapter 8.

4.3.4.4 Challenge 4—Distributed Storage and Widespread Software Bugs

The database is always growing in cloud applications. The opportunity is to create a storage system that will not only meet this growth, but also combine it with the cloud advantage of scaling arbitrarily up and down on demand. This demands the design of efficient distributed SANs. Data centers must meet programmers' expectations in terms of scalability, data durability, and HA. Data consistence checking in SAN-connected data centers is a major challenge in cloud computing.

Large-scale distributed bugs cannot be reproduced, so the debugging must occur at a scale in the production data centers. No data center will provide such a convenience. One solution may be a reliance on using VMs in cloud computing. The level of virtualization may make it possible to capture valuable information in ways that are impossible without using VMs. Debugging over simulators is another approach to attacking the problem, if the simulator is well designed.

4.3.4.5 Challenge 5—Cloud Scalability, Interoperability, and Standardization

The pay-as-you-go model applies to storage and network bandwidth; both are counted in terms of the number of bytes used. Computation is different depending on virtualization level. GAE automatically scales in response to load increases and decreases; users are charged by the cycles used. AWS charges by the hour for the number of VM instances used, even if the machine is idle. The opportunity here is to scale quickly up and down in response to load variation, in order to save money, but without violating SLAs.

Open Virtualization Format (OVF) describes an open, secure, portable, efficient, and extensible format for the packaging and distribution of VMs. It also defines a format for distributing software to be deployed in VMs. This VM format does not rely on the use of a specific host platform, virtualization platform, or guest operating system. The approach is to address virtual platform-agnostic packaging with certification and integrity of packaged software. The package supports virtual appliances to span more than one VM.

SDUDCS

Scanned with CamScanner

OVF also defines a transport mechanism for VM templates, and can apply to different virtualization platforms with different levels of virtualization. In terms of cloud standardization, we suggest the ability for virtual appliances to run on any virtual platform. We also need to enable VMs to run on heterogeneous hardware platform hypervisors. This requires hypervisor-agnostic VMs. We also need to realize cross-platform live migration between x86 Intel and AMD technologies and support legacy hardware for load balancing. All these issue are wide open for further research.

4.3.4.6 Challenge 6—Software Licensing and Reputation Sharing

Many cloud computing providers originally relied on open source software because the licensing model for commercial software is not ideal for utility computing. The primary opportunity is either for open source to remain popular or simply for commercial software companies to change their licensing structure to better fit cloud computing. One can consider using both pay-for-use and bulk-use licensing schemes to widen the business coverage.

One customer's bad behavior can affect the reputation of the entire cloud. For instance, blacklisting of EC2 IP addresses by spam-prevention services may limit smooth VM installation. An opportunity would be to create reputation-guarding services similar to the "trusted e-mail" services currently offered (for a fee) to services hosted on smaller ISPs. Another legal issue concerns the transfer of legal liability. Cloud providers want legal liability to remain with the customer, and vice versa. This problem must be solved at the SLA level. We will study reputation systems for protecting data centers in the next section.

Scanned with CamScanner

4.5 INTER-CLOUD RESOURCE MANAGEMENT

This section characterizes the various cloud service models and their extensions. The cloud service trends are outlined. Cloud resource management and intercloud resource exchange schemes are reviewed. We will discuss the defense of cloud resources against network threats in Section 4.6.

Scanned with CamScanner

4.5.2 Resource Provisioning and Platform Deployment

The emergence of computing clouds suggests fundamental changes in software and hardware architecture. Cloud architecture puts more emphasis on the number of processor cores or VM instances. Parallelism is exploited at the cluster node level. In this section, we will discuss techniques to provision computer resources or VMs. Then we will talk about storage allocation schemes to interconnect distributed computing infrastructures by harnessing the VMs dynamically.

4.5.2.1 Provisioning of Compute Resources (VMs)

Providers supply cloud services by signing SLAs with end users. The SLAs must commit sufficient resources such as CPU, memory, and bandwidth that the user can use for a preset period. Underprovisioning of resources will lead to broken SLAs and penalties. Overprovisioning of resources will lead to resource underutilization, and consequently, a decrease in revenue for the provider. Deploying an autonomous system to efficiently provision resources to users is a challenging problem. The difficulty comes from the unpredictability of consumer demand, software and hardware failures, heterogeneity of services, power management, and conflicts in signed SLAs between consumers and service providers.

Efficient VM provisioning depends on the cloud architecture and management of cloud infrastructures. Resource provisioning schemes also demand fast discovery of services and data in cloud computing infrastructures. In a virtualized cluster of servers, this demands efficient installation of VMs, live VM migration, and fast recovery from failures. To deploy VMs, users treat them as physical hosts with customized operating systems for specific applications. For example, Amazon's EC2 uses Xen as the virtual machine monitor (VMM). The same VMM is used in IBM's Blue Cloud.

Scanned with CamScanner

In the EC2 platform, some predefined VM templates are also provided. Users can choose different kinds of VMs from the templates. IBM's Blue Cloud does not provide any VM templates. In general, any type of VM can run on top of Xen. Microsoft also applies virtualization in its Azure cloud platform. The provider should offer resource-economic services. Power-efficient schemes for caching, query processing, and thermal management are mandatory due to increasing energy waste by heat dissipation from data centers. Public or private clouds promise to streamline the on-demand provisioning of software, hardware, and data as a service, achieving economies of scale in IT deployment and operation.

4.5.2.2 Resource Provisioning Methods

Figure 4.24 shows three cases of static cloud resource provisioning policies. In case (a), overprovisioning with the peak load causes heavy resource waste (shaded area). In case (b), underprovisioning (along the capacity line) of resources results in losses by both user and provider in that paid demand by the users (the shaded area above the capacity) is not served and wasted resources still exist for those demanded areas below the provisioned capacity. In case (c), the constant provisioning of resources with fixed capacity to a declining user demand could result in even worse resource waste. The user may give up the service by canceling the demand, resulting in reduced revenue for the provider. Both the user and provider may be losers in resource provisioning without elasticity.

Three resource-provisioning methods are presented in the following sections. The *demand-driven method* provides static resources and has been used in grid computing for many years. The *event-driven method* is based on predicted workload by time. The *popularity-driven method* is based

FIGURE 4.24

Three cases of cloud resource provisioning without elasticity: (a) heavy waste due to overprovisioning, (b) underprovisioning and (c) under- and then overprovisioning.

(Courtesy of Armbrust, et al., UC Berkeley, 2009 [4])

Scanned with CamScanner

on Internet traffic monitored. We characterize these resource provisioning methods as follows (see Figure 4.25).

4.5.2.3 Demand-Driven Resource Provisioning

This method adds or removes computing instances based on the current utilization level of the allocated resources. The demand-driven method automatically allocates two Xeon processors for the user application, when the user was using one Xeon processor more than 60 percent of the time for an extended period. In general, when a resource has surpassed a threshold for a certain amount of time, the scheme increases that resource based on demand. When a resource is below a threshold for a certain amount of time, that resource could be decreased accordingly. Amazon implements such an auto-scale feature in its EC2 platform. This method is easy to implement. The scheme does not work out right if the workload changes abruptly.

The x-axis in Figure 4.25 is the time scale in milliseconds. In the beginning, heavy fluctuations of CPU load are encountered. All three methods have demanded a few VM instances initially. Gradually, the utilization rate becomes more stabilized with a maximum of 20 VMs (100 percent utilization) provided for demand-driven provisioning in Figure 4.25(a). However, the event-driven method reaches a stable peak of 17 VMs toward the end of the event and drops quickly in Figure 4.25(b). The popularity provisioning shown in Figure 4.25(c) leads to a similar fluctuation with peak VM utilization in the middle of the plot.

4.5.2.4 Event-Driven Resource Provisioning

This scheme adds or removes machine instances based on a specific time event. The scheme works better for seasonal or predicted events such as Christmastime in the West and the Lunar New Year in the East. During these events, the number of users grows before the event period and then decreases during the event period. This scheme anticipates peak traffic before it happens. The method results in a minimal loss of QoS, if the event is predicted correctly. Otherwise, wasted resources are even greater due to events that do not follow a fixed pattern.

4.5.2.5 Popularity-Driven Resource Provisioning

In this method, the Internet searches for popularity of certain applications and creates the instances by popularity demand. The scheme anticipates increased traffic with popularity. Again, the scheme has a minimal loss of QoS, if the predicted popularity is correct. Resources may be wasted if traffic does not occur as expected. In Figure 4.25(c), EC2 performance by CPU utilization rate (the dark curve with the percentage scale shown on the left) is plotted against the number of VMs provisioned (the light curves with scale shown on the right, with a maximum of 20 VMs provisioned).

4.5.2.6 Dynamic Resource Deployment

The cloud uses VMs as building blocks to create an execution environment across multiple resource sites. The InterGrid-managed infrastructure was developed by a Melbourne University group [19]. Dynamic resource deployment can be implemented to achieve scalability in performance. The InterGrid is a Java-implemented software system that lets users create execution cloud environments on top of all participating grid resources. Peering arrangements established between gateways enable the allocation of resources from multiple grids to establish the execution environment. In Figure 4.26, a scenario is illustrated by which an *intergrid gateway (IGG)* allocates resources from a local cluster to

Scanned with CamScanner

FIGURE 4.26

Cloud resource deployment using an IGG (intergrid gateway) to allocate the VMs from a Local cluster to interact with the IGG of a public cloud provider.

(Courtesy of Constanzo, et al. [21])

deploy applications in three steps: (1) requesting the VMs, (2) enacting the leases, and (3) deploying the VMs as requested. Under peak demand, this IGG interacts with another IGG that can allocate resources from a cloud computing provider.

A grid has predefined peering arrangements with other grids, which the IGG manages. Through multiple IGGs, the system coordinates the use of InterGrid resources. An IGG is aware of the peering terms with other grids, selects suitable grids that can provide the required resources, and replies to requests from other IGGs. Request redirection policies determine which peering grid InterGrid selects to process a request and a price for which that grid will perform the task. An IGG can also allocate resources from a cloud provider. The cloud system creates a virtual environment to help users deploy their applications. These applications use the distributed grid resources.

The InterGrid allocates and provides a *distributed virtual environment* (DVE). This is a virtual cluster of VMs that runs isolated from other virtual clusters. A component called the *DVE manager* performs resource allocation and management on behalf of specific user applications. The core component of the IGG is a scheduler for implementing provisioning policies and peering with other gateways. The communication component provides an asynchronous message-passing mechanism. Received messages are handled in parallel by a thread pool.

Scanned with CamScanner

P R D EPD P

4.5.2.7 Provisioning of Storage Resources

The data storage layer is built on top of the physical or virtual servers. As the cloud computing applications often provide service to users, it is unavoidable that the data is stored in the clusters of the cloud provider. The service can be accessed anywhere in the world. One example is e-mail systems. A typical large e-mail system might have millions of users and each user can have thousands of e-mails and consume multiple gigabytes of disk space. Another example is a web searching application. In storage technologies, hard disk drives may be augmented with solid-state drives in the future. This will provide reliable and high-performance data storage. The biggest barriers to adopting flash memory in data centers have been price, capacity, and, to some extent, a lack of sophisticated query-processing techniques. However, this is about to change as the I/O bandwidth of solid-state drives becomes too impressive to ignore.

A distributed file system is very important for storing large-scale data. However, other forms of data storage also exist. Some data does not need the namespace of a tree structure file system, and instead, databases are built with stored data files. In cloud computing, another form of data storage is (Key, Value) pairs. Amazon S3 service uses SOAP to access the objects stored in the cloud. Table 4.8 outlines three cloud storage services provided by Google, Hadoop, and Amazon.

Many cloud computing companies have developed large-scale data storage systems to keep huge amount of data collected every day. For example, Google's GFS stores web data and some other data, such as geographic data for Google Earth. A similar system from the open source community is the Hadoop Distributed File System (HDFS) for Apache. Hadoop is the open source implementation of Google's cloud computing infrastructure. Similar systems include Microsoft's Cosmos file system for the cloud.

Despite the fact that the storage service or distributed file system can be accessed directly, similar to traditional databases, cloud computing does provide some forms of structure or semistructure database processing capability. For example, applications might want to process the information contained in a web page. Web pages are an example of semistructural data in HTML format. If some forms of database capability can be used, application developers will construct their application logic more easily. Another reason to build a database-like service in cloud computing is that it will be quite convenient for traditional application developers to code for the cloud platform. Databases are quite common as the underlying storage device for many applications.

Thus, such developers can think in the same way they do for traditional software development. Hence, in cloud computing, it is necessary to build databases like large-scale systems based on data

Table 4.8 Storage Services in Three Cloud Computing Systems

Storage System	Features
GFS: Google File System	Very large sustainable reading and writing bandwidth, mostly continuous accessing instead of random accessing. The programming interface is similar to that of the POSIX file system accessing interface.
HDFS: Hadoop Distributed File System	The open source clone of GFS. Written in Java. The programming interfaces are similar to POSIX but not identical.
Amazon S3 and EBS	S3 is used for retrieving and storing data from/to remote servers. EBS is built on top of S3 for using virtual disks in running EC2 instances.

Scanned with CamScanner

4) Global Exchange of Cloud Resources :-

⇒ To support a large number of application service consumers from around the world, cloud Infrastructure providers (i.e) (Iaas providers) have established data centers in multiple geographical locations to provide redundancy and ensure reliability in case of site failures.

⇒ for eg :- Amazon data centers, however currently Amazon expects it cloud customers (i.e) Saas providers) to express a preference regarding where they want their application services to be hosted.

⇒ First, it is difficult for cloud customers to determine in advance the best location for hosting their services as they may not know the origin of consumers of their services.

Scanned with CamScanner

FIGURE 4.30

Inter-cloud exchange of cloud resources through brokering.

(Courtesy of R. Buyya, et al., University of Melbourne [12])

service providers; operate as part of a market-driven resource leasing federation, where application service providers such as Salesforce.com host their services based on negotiated SLA contracts driven by competitive market prices; and deliver on-demand, reliable, cost-effective, and QoS-aware services based on virtualization technologies while ensuring high QoS standards and minimizing service costs. They need to be able to utilize market-based utility models as the basis for provisioning of virtualized software services and federated hardware infrastructure among users with heterogeneous applications.

They consist of client brokering and coordinator services that support utility-driven federation of clouds: application scheduling, resource allocation, and migration of workloads. The architecture cohesively couples the administratively and topologically distributed storage and compute capabilities of clouds as part of a single resource leasing abstraction. The system will ease the cross-domain capability integration for on-demand, flexible, energy-efficient, and reliable access to the infrastructure based on virtualization technology [6,75].

The *Cloud Exchange (CEx)* acts as a market maker for bringing together service producers and consumers. It aggregates the infrastructure demands from application brokers and evaluates them against the available supply currently published by the cloud coordinators. It supports trading of cloud services based on competitive economic models such as commodity markets and auctions. CEx allows participants to locate providers and consumers with fitting offers. Such markets enable

Scanned with CamScanner

6.2.1 Parallel Computing and Programming Paradigms

Consider a distributed computing system consisting of a set of networked nodes or workers. The system issues for running a typical parallel program in either a parallel or a distributed manner would include the following [36–39]:

- 1 **Partitioning** This is applicable to both computation and data as follows:
- 2 **Computation partitioning** This splits a given job or a program into smaller tasks. Partitioning greatly depends on correctly identifying portions of the job or program that can be performed concurrently. In other words, upon identifying parallelism in the structure of the program, it can be divided into parts to be run on different workers. Different parts may process different data or a copy of the same data.
- 3 **Data partitioning** This splits the input or intermediate data into smaller pieces. Similarly, upon identification of parallelism in the input data, it can also be divided into pieces to be processed on different workers. Data pieces may be processed by different parts of a program or a copy of the same program.
- 4 **Mapping** This assigns the either smaller parts of a program or the smaller pieces of data to underlying resources. This process aims to appropriately assign such parts or pieces to be run simultaneously on different workers and is usually handled by resource allocators in the system.
- 5 **Synchronization** Because different workers may perform different tasks, synchronization and coordination among workers is necessary so that race conditions are prevented and data dependency among different workers is properly managed. Multiple accesses to a shared resource by different workers may raise race conditions, whereas data dependency happens when a worker needs the processed data of other workers.
- 6 **Communication** Because data dependency is one of the main reasons for communication among workers, communication is always triggered when the intermediate data is sent to workers.
- 7 **Scheduling** For a job or program, when the number of computation parts (tasks) or data pieces is more than the number of available workers, a scheduler selects a sequence of tasks or data pieces to be assigned to the workers. It is worth noting that the resource allocator performs the actual mapping of the computation or data pieces to workers, while the scheduler only picks the next part from the queue of unassigned tasks based on a set of rules called the scheduling policy. For multiple jobs or programs, a scheduler selects a sequence of jobs or programs to be run on the distributed computing system. In this case, scheduling is also necessary when system resources are not sufficient to simultaneously run multiple jobs or programs.

Scanned with CamScanner

Motivations :

⇒ Motivations behind parallel & distributed programming models are :

(i) To improve productivity of programmers

(ii) To decrease programs time to market

(iii) to leverage underlying resources more efficiently

(iv) to increase system throughput

(v) to support higher levels of abstraction.

⇒ eg :- Mapreduce, Hadoop, Dryad.

Scanned with CamScanner

6.2.2 MapReduce, Twister, and Iterative MapReduce

MapReduce, as introduced in Section 6.1.4, is a software framework which supports parallel and distributed computing on large data sets [27,37,45,46]. This software framework abstracts the data flow of running a parallel program on a distributed computing system by providing users with two interfaces in the form of two functions: *Map* and *Reduce*. Users can override these two functions to interact with and manipulate the data flow of running their programs. Figure 6.1 illustrates the logical data flow from the *Map* to the *Reduce* function in MapReduce frameworks. In this framework,

FIGURE 6.1

MapReduce framework: Input data flows through the Map and Reduce functions to generate the output result under the control flow using MapReduce software library. Special user interfaces are used to access the Map and Reduce resources.

Scanned with CamScanner

FIGURE 6.4

Use of MapReduce *partitioning* function to link the Map and Reduce workers.

FIGURE 6.5

Data flow implementation of many functions in the Map workers and in the Reduce workers through multiple sequences of partitioning, combining, synchronization and communication, sorting and grouping, and reduce operations.

Scanned with CamScanner

6.2.2.7 Twister and Iterative MapReduce

It is important to understand the performance of different runtimes and, in particular, to compare MPI and MapReduce [43,44,55,56]. The two major sources of parallel overhead are load imbalance and communication (which is equivalent to synchronization overhead as communication synchronizes parallel units [threads or processes] in Categories 2 and 6 of Table 6.10). The communication overhead in MapReduce can be quite high, for two reasons:

- MapReduce reads and writes via files, whereas MPI transfers information directly between nodes over the network.

Scanned with CamScanner

FIGURE 6.6

Control flow implementation of the MapReduce functionalities in Map workers and Reduce workers (running user programs) from input files to the output files under the control of the master user program.

(*Courtesy of Yahoo! Pig Tutorial [54]*)

- MPI does not transfer all data from node to node, but just the amount needed to update information. We can call the MPI flow *δ flow* and the MapReduce flow *full data flow*.

The same phenomenon is seen in all "classic parallel" loosely synchronous applications which typically exhibit an iteration structure over compute phases followed by communication phases. We can address the performance issues with two important changes:

1. Stream information between steps without writing intermediate steps to disk.
2. Use long-running threads or processors to communicate the δ (between iterations) flow.

These changes will lead to major performance increases at the cost of poorer fault tolerance and ease to support dynamic changes such as the number of available nodes. This concept [42] has been investigated in several projects [34,57–59] while the direct idea of using MPI for MapReduce applications is investigated in [44]. The Twister programming paradigm and its implementation architecture at run time are illustrated in Figure 6.7(a, b). In Example 6.1, we summarize Twister [60] whose performance results for K means are shown in Figure 6.8 [55,56], where Twister is much faster than traditional MapReduce. Twister distinguishes the static data which is never reloaded from the dynamic δ flow that is communicated.

Scanned with CamScanner

FIGURE 6.7

Twister: An iterative MapReduce programming paradigm for repeated MapReduce executions.

Example 6.1 Performance of K Means Clustering in MPI, Twister, Hadoop, and DryadLINQ

The MapReduce approach leads to fault tolerance and flexible scheduling, but for some applications the performance degradation compared to MPI is serious, as illustrated in Figure 6.8 for a simple parallel K means clustering algorithm. Hadoop and DryadLINQ are more than a factor of 10 slower than MPI for the largest data set, and perform even more poorly for smaller data sets. One could use many communication mechanisms in Iterative MapReduce, but Twister chose a publish-subscribe network using a distributed set of brokers, as described in Section 5.2 with similar performance achieved with ActiveMQ and NaradaBrokering.

The Map-Reduce pair is iteratively executed in long-running threads. We compare in Figure 6.9. the different thread and process structures of 4 parallel programming paradigms: namely Hadoop, Dryad, Twister (also called MapReduce++), and MPI. Note that Dryad can use pipes and avoids costly disk writing according to the original papers [26,27].

Scanned with CamScanner

FIGURE 6.9

Thread and process structure of four parallel programming paradigms at runtimes.

Scanned with CamScanner

6.2.3 Hadoop Library from Apache

Hadoop is an open source implementation of MapReduce coded and released in Java (rather than C) by Apache. The Hadoop implementation of MapReduce uses the *Hadoop Distributed File System (HDFS)* as its underlying layer rather than GFS. The Hadoop core is divided into two fundamental layers: the MapReduce engine and HDFS. The MapReduce engine is the computation engine running on top of HDFS as its data storage manager. The following two sections cover the details of these two fundamental layers.

HDFS: HDFS is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.

HDFS Architecture: HDFS has a master/slave architecture containing a single NameNode as the master and a number of DataNodes as workers (slaves). To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes). The mapping of blocks to DataNodes is determined by the NameNode. The NameNode (master) also manages the file system's metadata and namespace. In such systems, the namespace is the area maintaining the metadata, and metadata refers to all the information stored by a file system that is needed for overall management of all files. For example, NameNode in the metadata stores all information regarding the location of input splits/blocks in

Scanned with CamScanner

all DataNodes. Each DataNode, usually one per node in a cluster, manages the storage attached to the node. Each DataNode is responsible for storing and retrieving its file blocks [61].

HDFS Features: Distributed file systems have special requirements, such as performance, scalability, concurrency control, fault tolerance, and security requirements [62], to operate efficiently. However, because HDFS is not a general-purpose file system, as it only executes specific types of applications, it does not need all the requirements of a general distributed file system. For example, security has never been supported for HDFS systems. The following discussion highlights two important characteristics of HDFS to distinguish it from other generic distributed file systems [63].

HDFS Fault Tolerance: One of the main aspects of HDFS is its fault tolerance characteristic. Since Hadoop is designed to be deployed on low-cost hardware by default, a hardware failure in this system is considered to be common rather than an exception. Therefore, Hadoop considers the following issues to fulfill reliability requirements of the file system [64]:

- **Block replication** To reliably store data in HDFS, file blocks are replicated in this system. In other words, HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster. The replication factor is set by the user and is three by default.
- **Replica placement** The placement of replicas is another factor to fulfill the desired fault tolerance in HDFS. Although storing replicas on different nodes (DataNodes) located in different racks across the whole cluster provides more reliability, it is sometimes ignored as the cost of communication between two nodes in different racks is relatively high in comparison with that of different nodes located in the same rack. Therefore, sometimes HDFS compromises its reliability to achieve lower communication costs. For example, for the default replication factor of three, HDFS stores one replica in the same node the original data is stored, one replica on a different node but in the same rack, and one replica on a different node in a different rack to provide three copies of the data [65].
- **Heartbeat and Blockreport messages** Heartbeats and Blockreports are periodic messages sent to the NameNode by each DataNode in a cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly, while each Blockreport contains a list of all blocks on a DataNode [65]. The NameNode receives such messages because it is the sole decision maker of all replicas in the system.

HDFS High-Throughput Access to Large Data Sets (Files): Because HDFS is primarily designed for batch processing rather than interactive processing, data access throughput in HDFS is more important than latency. Also, because applications run on HDFS typically have large data sets, individual files are broken into large blocks (e.g., 64 MB) to allow HDFS to decrease the amount of metadata storage required per file. This provides two advantages: The list of blocks per file will shrink as the size of individual blocks increases, and by keeping large amounts of data sequentially within a block, HDFS provides fast streaming reads of data.

HDFS Operation: The control flow of HDFS operations such as write and read can properly highlight roles of the NameNode and DataNodes in the managing operations. In this section, the control flow of the main operations of HDFS on files is further described to manifest the interaction between the user, the NameNode, and the DataNodes in such systems [63].

Scanned with CamScanner

- **Reading a file** To read a file in HDFS, a user sends an "open" request to the NameNode to get the location of file blocks. For each file block, the NameNode returns the address of a set of DataNodes containing replica information for the requested file. The number of addresses depends on the number of block replicas. Upon receiving such information, the user calls the *read* function to connect to the closest DataNode containing the first block of the file. After the first block is streamed from the respective DataNode to the user, the established connection is terminated and the same process is repeated for all blocks of the requested file until the whole file is streamed to the user.
- **Writing to a file** To write a file in HDFS, a user sends a "create" request to the NameNode to create a new file in the file system namespace. If the file does not exist, the NameNode notifies the user and allows him to start writing data to the file by calling the *write* function. The first block of the file is written to an internal queue termed the data queue while a data streamer monitors its writing into a DataNode. Since each file block needs to be replicated by a predefined factor, the data streamer first sends a request to the NameNode to get a list of suitable DataNodes to store replicas of the first block.

The steamer then stores the block in the first allocated DataNode. Afterward, the block is forwarded to the second DataNode by the first DataNode. The process continues until all allocated DataNodes receive a replica of the first block from the previous DataNode. Once this replication process is finalized, the same process starts for the second block and continues until all blocks of the file are stored and replicated on the file system.

Scanned with CamScanner

FIGURE 6.12

Data flow in running a MapReduce job at various task trackers using the Hadoop library.

Scanned with CamScanner

6.2.6 Mapping Applications to Parallel and Distributed Systems

In the past, Fox has discussed mapping applications to different hardware and software in terms of five application architectures [71]. These initial five categories are listed in Table 6.10, followed by a sixth emerging category to describe data-intensive computing [72,73]. The original classifications largely described simulations and were not aimed directly at data analysis. It is instructive to briefly summarize them and then explain the new category. Category 1 was popular 20 years ago, but is no longer significant. It describes applications that can be parallelized with lock-step operations controlled by hardware.

Such a configuration would run on SIMD (*single-instruction and multiple-data*) machines, whereas category 2 is now much more important and corresponds to a SPMD (*single-program and multiple-data*) model running on MIMD (*multiple instruction multiple data*) machines. Here, each decomposed

Scanned with CamScanner

Table 6.10 Application Classification for Parallel and Distributed Systems

Category	Class	Description	Machine Architecture
1	Synchronous	The problem class can be implemented with instruction-level lockstep operation as in SIMD architectures.	SIMD
2	Loosely synchronous (BSP or bulk synchronous processing)	These problems exhibit iterative compute-communication stages with independent compute (map) operations for each CPU that are synchronized with a communication step. This problem class covers many successful MPI applications including partial differential equation solutions and particle dynamics applications.	MIMD on MPP (massively parallel processor)
3	Asynchronous	Illustrated by Compute Chess and Integer Programming; combinatorial search is often supported by dynamic threads. This is rarely important in scientific computing, but it is at the heart of operating systems and concurrency in consumer applications such as Microsoft Word.	Shared memory
4	Pleasingly parallel	Each component is independent. In 1988, Fox estimated this at 20 percent of the total number of applications, but that percentage has grown with the use of grids and data analysis applications including, for example, the Large Hadron Collider analysis for particle physics.	Grids moving to clouds
5	Metaproblems	These are coarse-grained (asynchronous or data flow) combinations of categories 1-4 and 6. This area has also grown in importance and is well supported by grids and described by workflow in Section 3.5.	Grids of clusters
6	MapReduce++ (Twister)	This describes file (database) to file (database) operations which have three subcategories (see also Table 6.11): 6a) Pleasingly Parallel Map Only (similar to category 4) 6b) Map followed by reductions 6c) Iterative "Map followed by reductions" (extension of current technologies that supports linear algebra and data mining)	Data-intensive clouds a) Master-worker or MapReduce b) MapReduce c) Twister

unit executes the same program, but at any given time, there is no requirement that the same instruction be executed. Category 1 corresponds to regular problems, whereas category 2 includes dynamic irregular cases with complex geometries for solving partial differential equations or particle dynamics. Note that synchronous problems are still around, but they are run on MIMD machines with the SPMD model. Also note that category 2 consists of compute–communicate phases and the computations are synchronized by communication. No additional synchronization is needed.

Category 3 consists of asynchronously interacting objects and is often considered the people's view of a typical parallel problem. It probably does describe the concurrent threads in a modern operating system, as well as some important applications, such as event-driven simulations and areas such

Scanned with CamScanner

as search in computer games and graph algorithms. Shared memory is natural due to the low latency often needed to perform dynamic synchronization. It wasn't clear in the past, but now it appears that this category is not very common in large-scale parallel problems of importance.

Category 4 is the simplest algorithmically, with disconnected parallel components. However, the importance of this category has probably grown since the original 1988 analysis when it was estimated to account for 20 percent of all parallel computing. Both grids and clouds are very natural for this class, which does not need high-performance communication between different nodes.

Category 5 refers to the coarse-grained linkage of different "atomic" problems, and this was fully covered in Section 5.5. This area is clearly common and is expected to grow in importance. Remember the critical observation in Section 5.5 that we use a two-level programming model with the metaproblem (workflow) linkage specified in one fashion and the component problems with approaches such as those in this chapter. Grids or clouds are suitable for metaproblems as coarse-grained decomposition does not usually require stringent performance.

As noted earlier, we added a sixth category to cover data-intensive applications motivated by the clear importance of MapReduce as a new programming model. We call this category MapReduce++, and it has three subcategories: "map only" applications similar to pleasingly parallel category 4 [41,43,74,75]; the classic MapReduce with file-to-file operations consisting of parallel maps followed by parallel reduce operations; and a subcategory that captures the extended MapReduce version covered in Section 6.2.2. Note that category 6 is a subset of categories 2 and 4 with additional reading and writing of data and a specialized, loosely synchronous structure compared to that used in data analysis. This comparison is made clearer in Table 6.11.

6.3 PROGRAMMING SUPPORT OF GOOGLE APP ENGINE

Section 4.4.2 introduced the Google App Engine infrastructure in Figures 4.20. In this section, we describe the programming model supported by GAE. The access links to the GAE platform are provided in Chapter 4.

6.3.1 Programming the Google App Engine

Several web resources (e.g., http://code.google.com/appengine/) and specific books and articles (e.g., www.byteonic.com/2009/overview-of-java-support-in-google-app-engine/) discuss how to program GAE. Figure 6.17 summarizes some key features of GAE programming model for two supported languages: Java and Python. A client environment that includes an *Eclipse* plug-in for Java allows you to debug your GAE on your local machine. Also, the GWT Google Web Toolkit is available for Java web application developers. Developers can use this, or any other language using a JVM-based interpreter or compiler, such as JavaScript or Ruby. Python is often used with frameworks such as Django and CherryPy, but Google also supplies a built in *webapp* Python environment.

There are several powerful constructs for storing and accessing data. The data store is a NOSQL data management system for entities that can be, at most, 1 MB in size and are labeled by a set of schema-less properties. Queries can retrieve entities of a given kind filtered and sorted by the values of the properties. Java offers Java Data Object (JDO) and Java Persistence API (JPA) interfaces implemented by the open source Data Nucleus Access platform, while Python has a SQL-like query language called GQL. The data store is strongly consistent and uses optimistic concurrency control.

Scanned with CamScanner

Table 6.11 Comparison of MapReduce++ Subcategories along with the Loosely Synchronous Category Used in MPI

Map-Only	Classic MapReduce	Iterative MapReduce	Loosely Synchronous

• Document conversion (e.g., PDF->HTML)
• Brute force searches in cryptography
• Parametric sweeps
• Gene assembly
• PolarGrid Matlab data analysis (www.polargrid.org)
	• High-energy physics (HEP) histograms
	• Distributed search
	• Distributed sort
	• Information retrieval
	• Calculation of pairwise distances for sequences (BLAST)
		• Expectation maximization algorithms
		• Linear algebra
		• Data mining including
		  • Clustering
		  • K-means
		  • Deterministic annealing clustering
		  • Multidimensional scaling (MDS)
			• Many MPI scientific applications utilizing a wide variety of communication constructs including local interactions
			• Solving differential equations and particle dynamics with short-range forces

← **Domain of MapReduce and Iterative Extensions** → **MPI**

Scanned with CamScanner

FIGURE 6.17

Programming environment for Google AppEngine.

An update of an entity occurs in a transaction that is retried a fixed number of times if other processes are trying to update the same entity simultaneously. Your application can execute multiple data store operations in a single transaction which either all succeed or all fail together. The data store implements transactions across its distributed network using "entity groups." A transaction manipulates entities within a single group. Entities of the same group are stored together for efficient execution of transactions. Your GAE application can assign entities to groups when the entities are created. The performance of the data store can be enhanced by in-memory caching using the *memcache*, which can also be used independently of the data store.

Recently, Google added the *blobstore* which is suitable for large files as its size limit is 2 GB. There are several mechanisms for incorporating external resources. The *Google SDC Secure Data Connection* can tunnel through the Internet and link your intranet to an external GAE application. The *URL Fetch* operation provides the ability for applications to fetch resources and communicate with other hosts over the Internet using HTTP and HTTPS requests. There is a specialized mail mechanism to send e-mail from your GAE application.

Applications can access resources on the Internet, such as web services or other data, using GAE's URL fetch service. The URL fetch service retrieves web resources using the same high-speed Google infrastructure that retrieves web pages for many other Google products. There are dozens of Google "corporate" facilities including maps, sites, groups, calendar, docs, and YouTube, among others. These support the *Google Data API* which can be used inside GAE.

Scanned with CamScanner

6.4 PROGRAMMING ON AMAZON AWS AND MICROSOFT AZURE

In this section, we will consider the programming support in the AWS platform. First we will review the AWS platform and its updated service offerings. Then we will study the EC2, S3, and Simple DB services with programming examples. Returning to the programming environment features in Figures 4.22 and 4.23, Amazon (like Azure) offers a Relational Database Service (RDS) with a messaging interface sketched in Section 6.1.3. The Elastic MapReduce capability is equivalent to Hadoop running on the basic EC2 offering. Amazon has NOSQL support in SimpleDB introduced in Section 6.1.3 and discussed in Section 6.4.5. However, Amazon does not directly support BigTable as described in Section 6.3.4.

Now we will highlight a few more capabilities that are listed in Table 4.6. Amazon offers the Simple Queue Service (SQS) and Simple Notification Service (SNS), which are the cloud implementations of services discussed in Sections 5.2 and 5.4.5. Note that brokering systems run very efficiently in clouds and offer a striking model for controlling sensors and giving back-office support

Scanned with CamScanner

for a growing number of smartphones and tablets [77]. We further note the auto-scaling and elastic load balancing services which support related capabilities. Auto-scaling enables you to automatically scale your Amazon EC2 capacity up or down according to conditions that you define. With auto-scaling, you can ensure that the number of Amazon EC2 instances you're using scales up seamlessly during demand spikes to maintain performance, and scales down automatically during demand lulls to minimize cost.

Elastic load balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances and allows you to avoid nonoperating nodes and to equalize load on functioning images. Both auto-scaling and elastic load balancing are enabled by CloudWatch which monitors running instances. CloudWatch is a web service that provides monitoring for AWS cloud resources, starting with Amazon EC2. It provides customers with visibility into resource utilization, operational performance, and overall demand patterns—including metrics such as CPU utilization, disk reads and writes, and network traffic.

6.4.1 Programming on Amazon EC2

Amazon was the first company to introduce VMs in application hosting. Customers can rent VMs instead of physical machines to run their own applications. By using VMs, customers can load any software of their choice. The elastic feature of such a service is that a customer can create, launch, and terminate server instances as needed, paying by the hour for active servers. Amazon provides several types of preinstalled VMs. Instances are often called *Amazon Machine Images (AMIs)* which are preconfigured with operating systems based on Linux or Windows, and additional software.

Table 6.12 defines three types of AMI. Figure 6.24 shows an execution environment. AMIs are the templates for instances, which are running VMs. The workflow to create a VM is

$$\text{Create an AMI} \rightarrow \text{Create Key Pair} \rightarrow \text{Configure Firewall} \rightarrow \text{Launch} \tag{6.3}$$

This sequence is supported by public, private, and paid AMIs shown in Figure 6.24. The AMIs are formed from the virtualized compute, storage, and server resources shown at the bottom of Figure 6.23.

Table 6.12 Three Types of AMI

Image Type	AMI Definition
Private AMI	Images created by you, which are private by default. You can grant access to other users to launch your private images.
Public AMI	Images created by users and released to the AWS community, so anyone can launch instances based on them and use them any way they like. AWS lists all public images at http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171.
Paid QAMI	You can create images providing specific functions that can be launched by anyone willing to pay you per each hour of usage on top of Amazon's charges.

Scanned with CamScanner

FIGURE 6.23

Amazon EC2 execution environment.

Table 6.13 Instance Types Available on Amazon EC2 (October 6, 2010)

Compute Instance	Memory GB	ECU or EC2 Units	Virtual Cores	Storage GB	32/64 Bit
Standard: small	1.7	1	1	160	32
Standard: large	7.5	4	2	850	64
Standard: extra large	15	8	4	1690	64
Micro	0.613	Up to 2		Only EBS	32 or 64
High-memory	17.1	6.5	2	420	64
High-memory: double	34.2	13	4	850	64
High-memory: quadruple	68.4	26	8	1690	64
High-CPU: medium	1.7	5	2	350	32
High-CPU: extra large	7	20	8	1690	64
Cluster compute	23	33.5	8	1690	64

Example 6.9 Use of EC2 Services in the AWS Platform

Table 6.13 defines the IaaS instances available in October 2010 in five broad classes:

1. **Standard instances** are well suited for most applications.
2. **Micro instances** provide a small number of consistent CPU resources and allow you to burst CPU capacity when additional cycles are available. They are well suited for lower throughput applications and web sites that consume significant compute cycles periodically.
3. **High-memory instances** offer large memory sizes for high-throughput applications, including database and memory caching applications.

Scanned with CamScanner

Table 6.14 Cost of Amazon On-Demand VM Instant Types (October 6, 2010)

VM Instance Type	Size	Linux/UNIX Usage	Windows Usage
	Small (default)	$0.085 per hour	$0.12 per hour
Standard instances	Large	$0.34 per hour	$0.48 per hour
	Extra large	$0.68 per hour	$0.96 per hour
Micro instances	Micro	$0.02 per hour	$0.03 per hour
High-memory instances	Extra large	$0.50 per hour	$0.62 per hour
	Double extra large	$1.00 per hour	$1.24 per hour
	Quadruple extra large	$2.00 per hour	$2.48 per hour
Cluster compute instances	Quadruple extra large	$1.60 per hour	Not available

4. **High-CPU instances** have proportionally more CPU resources than memory (RAM) and are well suited for compute-intensive applications.
5. **Cluster compute instances** provide proportionally high CPU resources with increased network performance and are well suited for high-performance computing (HPC) applications and other demanding network-bound applications. They use 10 Gigabit Ethernet interconnections.

The cost in the third column is expressed in terms of EC2 Compute Units (ECUs) where one ECU provides the CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor. This leads to the cost per hour for CPUs shown in Table 6.14. Note that a real-world use of EC2 must pay for use of many different resources; the CPU charge in Table 6.14 is just one component and all charges (which naturally change often and so the reader should get the latest values online) are available on the AWS web site.

■

6.4.2 Amazon Simple Storage Service (S3)

Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. S3 provides the object-oriented storage service for users. Users can access their objects through *Simple Object Access Protocol* (SOAP) with either browsers or other client programs which support SOAP. SQS is responsible for ensuring a reliable message service between two processes, even if the receiver processes are not running. Figure 6.24 shows the S3 execution environment.

The fundamental operation unit of S3 is called an *object*. Each object is stored in a *bucket* and retrieved via a unique, developer-assigned key. In other words, the bucket is the container of the object. Besides unique key attributes, the object has other attributes such as values, metadata, and access control information. From the programmer's perspective, the storage provided by S3 can be viewed as a very coarse-grained key-value pair. Through the key-value programming interface, users can write, read, and delete objects containing from 1 byte to 5 gigabytes of data each. There are two types of web service interface for the user to access the data stored in Amazon clouds. One is a REST (web 2.0) interface, and the other is a SOAP interface. Here are some key features of S3:

- Redundant through geographic dispersion.
- Designed to provide 99.999999999 percent durability and 99.99 percent availability of objects over a given year with cheaper reduced redundancy storage (RRS).

Scanned with CamScanner

FIGURE 6.24

Amazon S3 execution environment.

- Authentication mechanisms to ensure that data is kept secure from unauthorized access. Objects can be made private or public, and rights can be granted to specific users.
- Per-object URLs and ACLs (access control lists).
- Default download protocol of HTTP. A BitTorrent protocol interface is provided to lower costs for high-scale distribution.
- $0.055 (more than 5,000 TB) to 0.15 per GB per month storage (depending on total amount).
- First 1 GB per month input or output free and then $.08 to $0.15 per GB for transfers outside an S3 region.
- There is no data transfer charge for data transferred between Amazon EC2 and Amazon S3 within the same region or for data transferred between the Amazon EC2 Northern Virginia region and the Amazon S3 U.S. Standard region (as of October 6, 2010).

Scanned with CamScanner

6.5 EMERGING CLOUD SOFTWARE ENVIRONMENTS

In this section, we will assess popular cloud operating systems and emerging software environments. We cover the open source Eucalyptus and Nimbus, then examine OpenNebula, Sector/Sphere, and Open Stack. These environments were introduced in Chapter 3 from a virtualization perspective. Here, we provide details regarding programming requirements. We will also cover the Aneka cloud programming tools recently developed at the University of Melbourne.

6.5.1 Open Source Eucalyptus and Nimbus

Eucalyptus is a product from Eucalyptus Systems (www.eucalyptus.com) that was developed out of a research project at the University of California, Santa Barbara. Eucalyptus was initially aimed at bringing the cloud computing paradigm to academic supercomputers and clusters. Eucalyptus provides an AWS-compliant EC2-based web service interface for interacting with the cloud service. Additionally, Eucalyptus provides services, such as the AWS-compliant Walrus, and a user interface for managing users and images.

6.5.1.1 Eucalyptus Architecture

The Eucalyptus system is an open software environment. The architecture was presented in a Eucalyptus white paper [79,80]. Figure 3.31 introduced Eucalyptus from a virtual clustering point of view. Figure 6.26 shows the architecture based on the need to manage VM images. The system supports cloud programmers in VM image management as follows. Essentially, the system has been extended to support the development of both the computer cloud and storage cloud.

6.5.1.2 VM Image Management

Eucalyptus takes many design queues from Amazon's EC2, and its image management system is no different. Eucalyptus stores images in Walrus, the block storage system that is analogous to the Amazon S3 service. As such, any user can bundle her own root file system, and upload and then register this image and link it with a particular kernel and ramdisk image. This image is uploaded into a user-defined bucket within Walrus, and can be retrieved anytime from any availability zone. This allows users to create specialty virtual appliances (http://en.wikipedia.org/wiki/Virtual_appliance) and deploy them within Eucalyptus with ease. The Eucalyptus system is available in a commercial proprietary version, as well as the open source version we just described.

6.5.1.3 Nimbus

Nimbus [81,82] is a set of open source tools that together provide an IaaS cloud computing solution. Figure 6.27 shows the architecture of Nimbus, which allows a client to lease remote resources by deploying VMs on those resources and configuring them to represent the environment desired

Scanned with CamScanner

FIGURE 6.26

The Eucalyptus architecture for VM image management.

(Courtesy of Eucalyptus LLC [81])

FIGURE 6.27

Nimbus cloud infrastructure.

(Courtesy of Nimbus Project [82])

Scanned with CamScanner

by the user. To this end, Nimbus provides a special web interface known as Nimbus Web [83]. Its aim is to provide administrative and user functions in a friendly interface. Nimbus Web is centered around a Python Django [84] web application that is intended to be deployable completely separate from the Nimbus service.

As shown in Figure 6.27, a storage cloud implementation called Cumulus [83] has been tightly integrated with the other central services, although it can also be used stand-alone. Cumulus is compatible with the Amazon S3 REST API [85], but extends its capabilities by including features such as quota management. Therefore, clients such as boto [86] and s2cmd [87], that work against the S3 REST API, work with Cumulus. On the other hand, the Nimbus cloud client uses the Java Jets3t library [88] to interact with Cumulus.

Nimbus supports two resource management strategies. The first is the default "resource pool" mode. In this mode, the service has direct control of a pool of VM manager nodes and it assumes it can start VMs. The other supported mode is called "pilot." Here, the service makes requests to a cluster's Local Resource Management System (LRMS) to get a VM manager available to deploy VMs. Nimbus also provides an implementation of Amazon's EC2 interface [89] that allows users to use clients developed for the real EC2 system against Nimbus-based clouds.

Scanned with CamScanner

6.5.2.2 OpenStack

OpenStack [103] was been introduced by Rackspace and NASA in July 2010. The project is building an open source community spanning technologists, developers, researchers, and industry to share resources and technologies with the goal of creating a massively scalable and secure cloud infrastructure. In the tradition of other open source projects, the software is open source and limited to just open source APIs such as Amazon.

Scanned with CamScanner

Currently, OpenStack focuses on the development of two aspects of cloud computing to address compute and storage aspects with the OpenStack Compute and OpenStack Storage solutions. "OpenStack Compute is the internal fabric of the cloud creating and managing large groups of virtual private servers" and "OpenStack Object Storage is software for creating redundant, scalable object storage using clusters of commodity servers to store terabytes or even petabytes of data." Recently, an image repository was prototyped. The image repository contains an image registration and discovery service and an image delivery service. Together they deliver images to the compute service while obtaining them from the storage service. This development gives an indication that the project is striving to integrate more services into its portfolio.

6.5.2.3 OpenStack Compute

As part of its computing support efforts, OpenStack [103] is developing a cloud computing fabric controller, a component of an IaaS system, known as Nova. The architecture for Nova is built on the concepts of shared-nothing and messaging-based information exchange. Hence, most communication in Nova is facilitated by message queues. To prevent blocking components while waiting for a response from others, deferred objects are introduced. Such objects include callbacks that get triggered when a response is received. This is very similar to established concepts from parallel computing, such as "futures," which have been used in the grid community by projects such as the CoG Kit.

To achieve the shared-nothing paradigm, the overall system state is kept in a distributed data system. State updates are made consistent through atomic transactions. Nova it implemented in Python while utilizing a number of externally supported libraries and components. This includes boto, an Amazon API provided in Python, and Tornado, a fast HTTP server used to implement the S3 capabilities in OpenStack. Figure 6.30 shows the main architecture of Open Stack Compute. In this architecture, the API Server receives HTTP requests from boto, converts the commands to and from the API format, and forwards the requests to the cloud controller.

The cloud controller maintains the global state of the system, ensures authorization while interacting with the User Manager via Lightweight Directory Access Protocol (LDAP), interacts with

FIGURE 6.30

OpenStack Nova system architecture. The AMQP (Advanced Messaging Queuing Protocol) was described in Section 5.2.

Scanned with CamScanner

the S3 service, and manages nodes, as well as storage workers through a queue. Additionally, Nova integrates networking components to manage private networks, public IP addressing, virtual private network (VPN) connectivity, and firewall rules. It includes the following types:

- *NetworkController* manages address and virtual LAN (VLAN) allocations
- *RoutingNode* governs the NAT (*network address translation*) conversion of public IPs to private IPs, and enforces firewall rules
- *AddressingNode* runs Dynamic Host Configuration Protocol (DHCP) services for private networks
- *TunnelingNode* provides VPN connectivity

The network state (managed in the distributed object store) consists of the following:

- VLAN assignment to a project
- Private subnet assignment to a security group in a VLAN
- Private IP assignments to running instances
- Public IP allocations to a project
- Public IP associations to a private IP/running instance

6.5.2.4 OpenStack Storage

The OpenStack storage solution is built around a number of interacting components and concepts, including a proxy server .a ring, an object server, a container server, an account server, replication, updaters, and auditors. The role of the proxy server is to enable lookups to the accounts, containers, or objects in OpenStack storage rings and route the requests. Thus, any object is streamed to or from an object server directly through the proxy server to or from the user. A ring represents a mapping between the names of entities stored on disk and their physical locations.

Separate rings for accounts, containers, and objects exist. A ring includes the concept of using zones, devices, partitions, and replicas. Hence, it allows the system to deal with failures, and isolation of zones representing a drive, a server, a cabinet, a switch, or even a data center. Weights can be used to balance the distribution of partitions on drives across the cluster, allowing users to support heterogeneous storage resources. According to the documentation, "the Object Server is a very simple blob storage server that can store, retrieve and delete objects stored on local devices."

Objects are stored as binary files with metadata stored in the file's extended attributes. This requires that the underlying file system is built around object servers, which is often not the case for standard Linux installations. To list objects, a container server can be utilized. Listing of containers is handled by the account server. The first release of OpenStack "Austin" Compute and Object Storage was October 22, 2010. This system has a strong developer community.

6.5.3 Manjrasoft Aneka Cloud and Appliances

Aneka (www.manjrasoft.com/) is a cloud application platform developed by Manjrasoft, based in Melbourne, Australia. It is designed to support rapid development and deployment of parallel and distributed applications on private or public clouds. It provides a rich set of APIs for transparently exploiting distributed resources and expressing the business logic of applications by using preferred programming abstractions. System administrators can leverage a collection of tools to monitor and control the deployed infrastructure. It can be deployed on a public cloud such as Amazon EC2

Scanned with CamScanner

accessible through the Internet to its subscribers, or a private cloud constituted by a set of nodes with restricted access as shown in Figure 6.31.

Aneka acts as a workload distribution and management platform for accelerating applications in both Linux and Microsoft .NET framework environments. Some of the key advantages of Aneka over other workload distribution solutions include:

- Support of multiple programming and application environments
- Simultaneous support of multiple runtime environments
- Rapid deployment tools and framework
- Ability to harness multiple virtual and/or physical machines for accelerating application provisioning based on users' Quality of Service/service-level agreement (QoS/SLA) requirements
- Built on top of the Microsoft .NET framework, with support for Linux environments through Mono

FIGURE 6.31
Architecture and components of Aneka.

Scanned with CamScanner

Aneka offers three types of capabilities which are essential for building, accelerating, and managing clouds and their applications:

1. **Build** Aneka includes a new SDK which combines APIs and tools to enable users to rapidly develop applications. Aneka also allows users to build different runtime environments such as enterprise/private cloud by harnessing compute resources in network or enterprise data centers, Amazon EC2, and hybrid clouds by combining enterprise private clouds managed by Aneka with resources from Amazon EC2 or other enterprise clouds built and managed using XenServer.
2. **Accelerate** Aneka supports rapid development and deployment of applications in multiple runtime environments running different operating systems such as Windows or Linux/UNIX. Aneka uses physical machines as much as possible to achieve maximum utilization in local environments. Whenever users set QoS parameters such as deadlines, and if the enterprise resources are insufficient to meet the deadline, Aneka supports dynamic leasing of extra capabilities from public clouds such as EC2 to complete the task within the deadline (see Figure 6.32).

FIGURE 6.32

Aneka using private cloud resources along with dynamically leased public cloud resources.

(Courtesy of Raj Buyya, Manjrasoft, www.manjrasoft.com)

Scanned with CamScanner

3. **Manage** Management tools and capabilities supported by Aneka include a GUI and APIs to set up, monitor, manage, and maintain remote and global Aneka compute clouds. Aneka also has an accounting mechanism and manages priorities and scalability based on SLA/QoS which enables dynamic provisioning.

Here are three important programming models supported by Aneka for both cloud and traditional parallel applications:

1. Thread programming model, best solution to adopt for leveraging the computing capabilities of multicore nodes in a cloud of computers
2. Task programming model, which allows for quickly prototyping and implementing an independent bag of task applications
3. MapReduce programming model,as discussed in Section 6.2.2

6.5.3.1 Aneka Architecture

Aneka as a cloud application platform features a homogeneous distributed runtime environment for applications. This environment is built by aggregating together physical and virtual nodes hosting the Aneka container. The container is a lightweight layer that interfaces with the hosting environment and manages the services deployed on a node. The interaction with the hosting platform is mediated through the *Platform Abstraction Layer (PAL)*, which hides in its implementation all the heterogeneity of the different operating systems.

By means of the PAL it is possible to perform all the infrastructure-related tasks, such as performance and system monitoring. These activities are vital to ensure the desired QoS for applications. The PAL, together with the container, represents the hosting environment of services which implement the core capabilities of the middleware and make it a dynamically composable and extensible system. The available services can be aggregated into three major categories:

Fabric Services: Fabric services implement the fundamental operations of the infrastructure of the cloud. These services include HA and failover for improved reliability, node membership and directory, resource provisioning, performance monitoring, and hardware profiling.

Foundation Services: Foundation services constitute the core functionalities of the Aneka middleware. They provide a basic set of capabilities that enhance application execution in the cloud. These services provide added value to the infrastructure and are of use to system administrators and developers. Within this category we can list storage management, resource reservation, reporting, accounting, billing, services monitoring, and licensing. Services in this level operate across the range of supported application models.

Application Services: Application services deal directly with the execution of applications and are in charge of providing the appropriate runtime environment for each application model. They leverage foundation and fabric services for several tasks of application execution such as elastic scalability, data transfer, and performance monitoring, accounting, and billing. At this level, Aneka expresses its true potential in supporting different application models and distributed programming patterns.

Scanned with CamScanner

UNIT-V
CLOUD SECURITY

Security Overview-Cloud Security Challenges and Risks-Software-as-a-Service-Security Security Governance-Risk Management-Security Monitoring-Security Architecture Design-Data Security-Application Security-Virtual Machine Security-Identity Management and Access Control-Autonomic Security.

Security Overview

It is a set of control-based technologies & policies adapted to stick to regulatory compliances, rules & protect data application and cloud technology infrastructure. Because of cloud's nature of sharing resources, cloud security gives particular concern to identity management, privacy & access control. So the data in the cloud should have to be stored in an encrypted form. With the increase in the number of organizations using cloud technology for a data operation, proper security and other potentially vulnerable areas became a priority for organizations contracting with cloud providers. Cloud computing security processes the security control in cloud & provides customer data security, privacy & compliance with necessary regulations.

Cloud Security Challenges and Risks

These are:

- Analyze the sensitivity to risks of user's resources.
- The cloud service models require the customer to be responsible for security at various levels of service.
- Understand the data storage and transfer mechanism provided by the cloud service provider.
- Consider proper cloud type to be used.

This emergent cloud technology is facing many technological challenges in different aspects of data & information handling & storage.

Some of the challenges are as follows:

- Availability & reliability
- Security & Privacy
- Interoperability
- Performance
- Portability

Figure - Challenges Of Cloud:

The challenges as mentioned above are the most important and concerned points that should be processed for the betterment.

Security Planning

Before deploying a particular resource to cloud, one should need to analyze several aspects of the resource such as:

- Select resource that needs to move to the cloud and analyze its sensitivity to risk.
- Consider cloud service models such as **IaaS, PaaS,** and **SaaS.** These models require customer to be responsible for security at different levels of service.
- Consider the cloud type to be used such as **public, private, community** or **hybrid.**
- Understand the cloud service provider's system about data storage and its transfer into and out of the cloud.

The risk in cloud deployment mainly depends upon the service models and cloud types.

Understanding Security of Cloud

Security Boundaries

A particular service model defines the boundary between the responsibilities of service provider and customer. **Cloud Security Alliance (CSA)** stack model defines the boundaries between each service model and shows how different functional units relate to each other. The following diagram shows the **CSA stack model:**

Key Points to CSA Model

- IaaS is the most basic level of service with PaaS and SaaS next two above levels of services.
- Moving upwards, each of the service inherits capabilities and security concerns of the model beneath.
- IaaS provides the infrastructure, PaaS provides platform development environment, and SaaS provides operating environment.
- IaaS has the least level of integrated functionalities and integrated security while SaaS has the most.
- This model describes the security boundaries at which cloud service provider's responsibilities end and the customer's responsibilities begin.
- Any security mechanism below the security boundary must be built into the system and should be maintained by the customer.

Although each service model has security mechanism, the security needs also depend upon where these services are located, in private, public, hybrid or community cloud.

The main security risks of cloud computing are:

- Compliance violations
- Identity theft
- Malware infections and data breaches
- Diminished customer trust and potential revenue loss

While the concern is understandable, today's reality is that—when implemented correctly—cloud computing security is just as reliable as traditional on-premise IT.

Software-as-a-Service-Security

Software-as-a-service (SaaS) is an on-demand, cloud-based software delivery model that enables organizations to subscribe to the applications they need without hosting them in house. SaaS is one of several categories of cloud subscription services, including platform-as-a-service and infrastructure-as-a-service. SaaS has become increasingly popular because it saves organizations from needing to purchase servers and other infrastructure or maintain an in-house support staff. Instead, a SaaS provider hosts and provides SaaS security and maintenance to their software. Some well-known SaaS applications include Microsoft Office 365, Salesforce.com, Cisco Webex, Box, and Adobe Creative Cloud. Most enterprise software vendors also offer cloud versions of their applications, such as Oracle Financials Cloud.

SaaS security

SaaS providers handle much of the security for a cloud application. The SaaS provider is responsible for securing the platform, network, applications, operating system, and physical infrastructure. However, providers are not responsible for securing customer data or user access to it. Some providers offer a bare minimum of security, while others offer a wide range of SaaS security options.

- **Apply identity and access management (IAM).** A role-based identity and access management solution can ensure that end users do not gain access to more resources than they require for their jobs. IAM solutions use processes and user access policies to determine what files and applications a particular user can access. An organization can apply role-based permissions to data so that end users will see only the data they're authorized to view.
- **Encrypt cloud data.** Data encryption protects both data at rest (in storage) and data in transit between the end user and the cloud or between cloud applications. Government regulations usually require encryption of sensitive data. Sensitive data includes financial information, healthcare data, and personally identifiable information (PII). While a SaaS vendor may provide some type of encryption, an organization can enhance data security by applying its own encryption, such as by implementing a cloud access security broker (CASB).
- **Enforce data loss prevention (DLP).** DLP software monitors for sensitive data within SaaS applications or outgoing transmissions of sensitive data and blocks the transmission. DLP software detects and prevents sensitive data from being downloaded to personal devices and blocks malware or hackers from attempting to access and download data.
- **Monitor collaborative sharing of data.** Collaboration controls can detect granular permissions on files that are shared with other users, including users outside the organization who access the file through a web link. Employees may inadvertently or intentionally share confidential documents through email, team spaces, and cloud storage sites such as Dropbox.

SaaS security solutions

- **Data loss prevention (DLP)**) safeguards intellectual property and protects sensitive data in cloud applications, as well as at endpoints such as laptops. Organizations can define data access policies that DLP enforces.
- **Compliance solutions** provide controls and reporting capabilities to ensure compliance with government and industry regulations.
- **Advanced malware prevention** includes technologies such as behavioral analytics and real-time threat intelligence that can help detect and block zero-day attacks and malicious files that may be spread through cloud email and file sharing applications.
- **Cloud access security brokers (CASBs)** protect enterprise data and users across all cloud services, including SaaS, PaaS, and IaaS. According to Gartner's Magic Quadrant for Cloud Access Security Brokers, CASBs detect threats and provide IT departments with greater visibility into data usage and user behavior for cloud services, end users, and devices. CASBs also act immediately to remediate security threats by eliminating security misconfigurations and correcting high-risk user activities applications. CASBs provide a variety of security services, including:
    - Monitoring for unauthorized cloud services
    - Enforcing data security policies including encryption
    - Collecting details about users who access data in cloud services from any device or location
    - Restricting access to cloud services based on the user, device, and application
    - Providing compliance reporting

Security Governance

Cloud security governance refers to the management model that facilitates effective and efficient security management and operations in the cloud environment so that an enterprise's business targets are achieved. This model incorporates a hierarchy of executive mandates, performance expectations, operational practices, structures, and metrics that, when implemented, result in the optimization of business value for an enterprise. Cloud security governance helps answer leadership questions such as:

- Are our security investments yielding the desired returns?
- Do we know our security risks and their business impact?
- Are we progressively reducing security risks to acceptable levels?
- Have we established a security-conscious culture within the enterprise?

Strategic alignment, value delivery, risk mitigation, effective use of resources, and performance measurement are key objectives of any IT-related governance model, security included. To successfully pursue and achieve these objectives, it is important to understand the operational culture and business and customer profiles of an enterprise, so that an effective security

Key Objectives for Cloud Security Governance

The following represents key objectives to pursue in establishing a governance model for security in the cloud. These objectives assume that appropriate security standards and a governance framework have been chosen based on the enterprise's business targets, customer profile, and obligations for protecting data and other information assets in the cloud environment.

1. **Strategic Alignment**
   Enterprises should mandate that security investments, services, and projects in the cloud are executed to achieve established business goals (e.g., market competitiveness, financial, or operational performance).

2. **Value Delivery**
   Enterprises should define, operationalize, and maintain an appropriate security function/organization with appropriate strategic and tactical representation, and charged with the responsibility to maximize the business value (Key Goal Indicators, ROI) from the pursuit of security initiatives in the cloud.

3. **Risk Mitigation**
   Security initiatives in the cloud should be subject to measurements that gauge effectiveness in mitigating risk to the enterprise (Key Risk Indicators). These initiatives should also yield results that progressively demonstrate a reduction in these risks over time.

4. **Effective Use of Resources**
   It is important for enterprises to establish a practical operating model for managing and performing security operations in the cloud, including the proper definition and operationalization of due processes, the institution of appropriate roles and responsibilities, and use of relevant tools for overall efficiency and effectiveness.

5. **Sustained Performance**
   Security initiatives in the cloud should be measurable in terms of performance, value and risk to the enterprise (Key Performance Indicators, Key Risk Indicators), and yield results that demonstrate attainment of desired targets (Key Goal Indicators) over time.

Risk Management

It is a

- Set of activities and methods to control risks.
- Architecture to manage risks.
- Identifying the security risks
- Occurrence for these risks
- Safeguards against these risks
- Controls for reducing or eliminating those risks

Risk assessment steps

- Threat Identification
- Vulnerability Identification(list)
- Risk Determination(level)
- Control Recommendation

To Evaluate

- Data Location
- Recovery
- Viability
- Support in Reducing Risk

Data Location

- Every customer need to know where his data are hosted, in which country the data is stored

Recovery

- How cloud offerings will recover from total disaster?
- Know what will happen if one of the offered sites went down?
- Can it completely restore everything?
- How much time does it need to complete restoration?

Viability

- What would happen to your service if the provider goes broke?
- How would I get my data back?
- Can I use the data in a replacement application?

Support in Reducing Risk

- How to use the product safely?
- To whom the instructions for setting and monitoring policies provided to ?
- How to avoid phishing or malware attacks?

Security Monitoring

Monitoring is a critical component of cloud security and management. Typically relying on automated solutions, cloud security monitoring supervises virtual and physical servers to continuously assess and measure data, application, or infrastructure behaviors for potential security threats. This assures that the cloud infrastructure and platform function optimally while minimizing the risk of costly data breaches.

BENEFITS OF CLOUD SECURITY MONITORING

Cloud monitoring provides an easier way to identify patterns and pinpoint potential security vulnerabilities in cloud infrastructure. As there's a general perception of a loss of control when valuable data is stored in the cloud, effective cloud monitoring can put companies more at ease with making use of the cloud for transferring and storing data.

When customer data is stored in the cloud, cloud monitoring can prevent loss of business and frustrations for customers by ensuring that their personal data is safe. The use of web services can increase security risks, yet cloud computing offers many benefits for businesses, from accessibility to a better customer experience. Cloud monitoring is one initiative that enables companies to find the balance between the ability to mitigate risks and taking advantage of the benefits of the cloud – and it should do so without hindering business processes.

HOW CLOUD SECURITY MONITORING WORKS

There are several approaches to cloud security monitoring. Cloud monitoring can be done in the cloud platform itself, on premises using an enterprise's existing security management tools, or via a third party service provider. Some of the key capabilities of cloud security monitoring software include:

- **Scalability:** tools must be able to monitor large volumes of data across many distributed locations
- **Visibility:** the more visibility into application, user, and file behavior that a cloud monitoring solution provides, the better it can identify potential attacks or compromises
- **Timeliness:** the best cloud security monitoring solutions will provide constant monitoring, ensuring that new or modified files are scanned in real time
- **Integration:** monitoring tools must integrate with a wide range of cloud storage providers to ensure full monitoring of an organization's cloud usage
- **Auditing and Reporting:** cloud monitoring software should provide auditing and reporting capabilities to manage compliance requirements for cloud security

Security Architecture Design

As a first step, architects need to understand what security capabilities are offered by cloud platforms (PaaS, IaaS). The figure below illustrates the architecture for building security into cloud services.

The following are cloud security best practices to mitigate risks to cloud services:

- **Architect for security-as-a-service** – Application deployments in the cloud involve orchestration of multiple services including automation of DNS, load balancer, network QoS, etc. Security automation falls in the same category which includes automation of firewall policies between cloud security zones, provisioning of certificates (for SSL), virtual machine system configuration, privileged accounts and log configuration. Application deployment processes that depend on security processes such as firewall policy creation, certificate provisioning, key distribution and application pen testing should be migrated to a self-service model. This approach will eliminate human touch points and will enable a security as a service scenario. Ultimately this will mitigate threats due to human errors, improve operational efficiency and embed security controls into the cloud applications.

- **Implement sound identity, access management architecture and practice** – Scalable cloud bursting and elastic architecture will rely less on network based access controls and warrant strong user access management architecture. Cloud access control architecture should address all aspects of user and access management lifecycles for both end users and privileged users – user provisioning & deprovisioning, authentication, federation, authorization and auditing. A sound architecture will enable reusability of identity and access services for all use cases in

public, private and hybrid cloud models. It is good practice to employ secure token services along with proper user and entitlement provisioning with audit trails. Federation architecture is the first step to extending enterprise SSO to cloud services. Refer to cloud security alliance, Domain 12 for detailed guidance here.

- **Leverage APIs to automate safeguards** – Any new security services should be deployed with an API (REST/SOAP) to enable automation. APIs can help automate firewall policies, configuration hardening, and access control at the time of application deployment. This can be implemented using open source tools such as puppet in conjunction with the API supplied by cloud service provider.

- **Always encrypt or mask sensitive data** – Today's private cloud applications are candidates for tomorrow's public cloud deployment. Hence architect applications to encrypt all sensitive data irrespective of the future operational model.

- **Do not rely on an IP address for authentication services** – IP addresses in clouds are ephemeral in nature so you cannot solely rely on them for enforcing network access control. Employ certificates (self-signed or from a trusted CA) to enable SSL between services deployed on cloud.

- **Log, Log, Log** – Applications should centrally log all security events that will help create an end-to-end transaction view with non-repudiation characteristics. In the event of a security incident, logs and audit trails are the only reliable data leveraged by forensic engineers to investigate and understand how an application was exploited. Clouds are elastic and logs are ephemeral hence it is critical to periodically migrate log files to a different cloud or to the enterprise data center.

- **Continuously monitor cloud services** – Monitoring is an important function given that prevention controls may not meet all the enterprise standards. Security monitoring should leverage logs produced by cloud services, APIs and hosted cloud applications to perform security event correlation. Cloud audit (cloudaudit.org) from CSA can be leveraged towards this mission

In general, patterns should highlight the following attributes (but not limited to) for each of the security services consumed by the cloud application:

- Logical location – Native to cloud service, in-house, third party cloud. The location may have an implication on the performance, availability, firewall policy as well as governance of the service.
- Protocol – What protocol(s) are used to invoke the service? For example REST with X.509 certificates for service requests.
- Service function – What is the function of the service? For example encryption of the artifact, logging, authentication and machine finger printing.
- Input/Output – What are the inputs, including methods to the controls, and outputs from the security service? For example, Input = XML doc and Output =XML doc with encrypted attributes.

- Control description – What security control does the security service offer? For example, protection of information confidentiality at rest, authentication of user and authentication of application.
- Actor – Who are the users of this service? For example, End point, End user, Enterprise administrator, IT auditor and Architect.

Data Security

Data security has been incumbent on the cloud service providers, and they have risen to the occasion. No matter which platform you select in the debate between AWS vs. Azure vs. Google, all sport various compliances to standards like HIPAA, ISO, PCI DSS, and SOC.

Privacy Protection

Your data should be protected from unauthorized access regardless of your cloud decisions, which includes data encryption and controlling who sees and can access what. There may also situations where you want to make data available to certain personnel under certain circumstances. For example, developers need live data for testing apps but they don't necessarily need to see the data, so you would use a redaction solution. Oracle, for example, has a Data Redact tool for its databases.
The first step is something you should have done already: identify the sensitive data types and define them. Discover where the sensitive data resides, classify and define the data types, and create policies based on where the data is and which data types can go into the cloud and which cannot. Too many early adopters of the cloud rushed to move all their data there, only to realize it needed to be kept on-premises in a private cloud.

There are automated tools to help discover and identify an organization's sensitive data and where it resides. Amazon Web Services has Macie while Microsoft Azure has Azure Information Protection (AIP) to classify data by applying labels. Third party tools include Tableau, Fivetran, Logikcull, and Looker.

Preserve Data Integrity

Data integrity can be defined as protecting data from unauthorized modification or deletion. This is easy in a single database, because there is only one way in or out of the database, which you can control. But in the cloud, especially a multicloud environment, it gets tricky.

Because of the large number of data sources and means to access, authorization becomes crucial in assuring that only authorized entities can interact with data. This means stricter means of access, like two-factor authorization, and logging to see who accessed what. Another potential means of security is a trusted platform module (TPM) for remote data checks.

Data Availability

Downtime is a fact of life and all you can do is minimize the impact. That's of considerable importance with cloud storage providers because your data is on

someone else's servers. This is where the service-level agreement (SLA) is vital and paying a close eye to details really matters.

For example, Microsoft offers 99.9% availability for major Azure storage options but AWS offers 99.99% availability for stored objects. This difference is not trivial. Also, make sure your SLA allows you to specify where the data is stored. Some providers, like AWS, allow you to dictate in what region data is stored. This can be important for issues of compliance and response time/latency.

Every cloud storage service has a particular strength: Amazon's Glacier is ideal for mass cold storage of rarely accessed data, Microsoft's Azure blob storage is ideal for most unstructured data, while Google Cloud's SQL is tuned for MySQL databases.

Data Privacy

A huge raft of privacy laws, national and international, have forced more than a few companies to say no to the cloud because they can't make heads or tails of the law or it's too burdensome. And it's not hard to see why.

All of these protect the interest of the data owner, so it is in your best interest to know them and know how well your provider complies with them.

Encryption

Encryption is the means for which data privacy is protected and insured, and encryption technologies are fairly mature. Encryption is done via key-based algorithms and the keys are stored by the cloud provider, although some business-related apps, like Salesforce and Dynamix, use tokenization instead of keys. This involves substituting specific token fields for anonymous data tokens.

Virtually every cloud storage provider encrypts the data while it is in transfer. Most do it through browser interfaces, although there are some cloud storage providers like Mega and SpiderOak that use a dedicated client to perform the encryption. This should all be spelled out in the SLA.

Many cloud services offer key management solutions that allow you to control access because the encryption keys are in your hands. This may prove to be a better or at least more reassuring risk because you are in control over who has the keys. Again, this should be spelled out in the SLA.

Threats

If you are online you are under threat of attack, that is a fact of life. The old style of attacks, like DDoS attacks, SQL injection, and cross-site scripting, have faded into new fears. Cloud service providers have a variety of security tools and policies in place but problems still happen, usually originating in human error.

- **Data breaches:** This can happen any number of ways, from the usual means – a hacked account or a lost password/laptop – to means unique to the cloud. For example, it is possible for a user on one virtual machine to listen for the signal that an encryption key has arrived on another VM on the same host. It's called the "side channel timing exposure," and it means the victim's security credentials in the hands of someone else.

- **Data loss:** While the chance of data loss is minimal short of someone logging in and erasing everything, it is possible. You can mitigate this by insuring your applications and data are distributed across several zones and you backup your data using off-site storage.
- **Hijacked accounts:** All it takes is one lost notebook for someone to get into your cloud provider. Secure, tough passwords and two-factor authentication can prevent this. It also helps to have policies that look for and alert to unusual activity, like copying mass amounts of data or deleting it.
- **Cryptojacking:** Cryptojacking is the act of surreptitiously taking over a computer to farm cryptocurrency, which is a very compute-intensive process. Cryptojacking spiked in 2017 and 2018 and the cloud was a popular target because there is more compute resources available. Monitoring for unusual compute activity is the key way to stop this.

Application Security

Cloud application security is a series of defined policies, processes, controls, and technology governing all information exchanges that happen in collaborative cloud environments like Microsoft Office 365, Google G Suite, Slack, and Box (to name a few).
So, if you or your employees frequently store and share data in cloud applications like the ones listed above (or any of the tens of thousands available), it is absolutely necessary to add a cloud application "safety net" to your zero trust security infrastructure.

Top Cloud Application Security Threats
It is no secret that there are security issues in cloud computing that IT teams must be aware of. According to the 2018 Cybersecurity Insider Report, the four most common cloud application security threats that IT teams are facing include:

1. **Misconfiguration of application setup** is the single biggest threat to cloud security because data breaches tend to happen when services are accidentally exposed to the public internet.
2. **Unauthorized access** to a website, server, service, or other system is also an area for great concern because once they're in, there's no telling what unauthorized users will do to create chaos.
3. **Insecure APIs and interfaces** present easy opportunities for attackers to breach systems because they are the only asset(s) outside of the organizational boundary with a public IP address.
4. **Account hijacking** is feared because so much sensitive data and resources is stored and accessed on devices shared by many different users—and because keeping tabs on rogue employees is difficult.

Cloud providers, however, have no visibility into what happens at the application layer and have no way to help the application owners in this area. The application security layer is the responsibility of application owners.

Before we can evaluate a solution for application security, we need to understand the following challenges:

- Security monitoring – there are numerous questions about the solution's capability, but monitoring the security should not be one of them. Security monitoring is imperative; it's a must-have
- Application vulnerabilities – these are susceptible to attackers looking to exploit and attack an application, either to gain complete control over it, deform it or steal data. OWASP analyses such vulnerabilities and exploits, and regularly publishes a list of its top 10 identified vulnerabilities
- Malware and ransomware – another well-known security problem that impacts a lot of users and should be addressed prior to deploying an application in the cloud
- Bots – approximately 30 percent of traffic comes from non-useful bots (i.e. bad bots). While some people don't consider them a security issue, yet, bad bots can waste 30 percent in server resources, resulting in a huge loss of productivity
- Application layer DDoS attacks (volumetric or protocol exploits) – are also a concern as DDoS attacks evolve in size, scope and sophistication. DDoS protection is a serious consideration for both application owners and cloud infrastructure providers

Solving these challenges

- Fortunately, there are solutions available to overcome the security challenges associated with cloud applications.
- Web Application Firewalls (WAFs), for example, can handle the common vulnerabilities listed by OWASP. And IP reputation and other signature databases have been created to combat malware and bad BOTs.
- Many Application Delivery Controllers (ADCs) bundle application security solutions with load balancing and other key application services. Having a complete set of application delivery tools along with security and visibility in a DDoS resilient architecture can create a complicated deployment architecture.

Virtual Machine Security

Bringing VMs (virtual machines) onto the cloud. Recent years have seen great advancements in both cloud computing and virtualization On one hand there is the ability to pool various resources to provide software-as-a-service, infrastructure-as-a-service and platform-as-a-service. At its most basic, this is what describes cloud computing. On the other hand, we have virtual machines that provide agility, flexibility, and scalability to the cloud resources by allowing the vendors to copy, move, and manipulate their VMs at will. The term *virtual machine* essentially describes sharing the resources of one single physical computer into various computers within itself. *VMware* and *virtual box* are very commonly used virtual systems on desktops. Cloud computing effectively stands for many computers

pretending to be one computing environment. Obviously, cloud computing would have many virtualized systems to maximize resources.
One issue is that the cloud-computing scenario is not as transparent as it claims to be. The service user has no clue about how his information is processed and stored. In addition, the service user cannot directly control the flow of data/information storage and processing. The service provider usually is not aware of the details of the service running on his or her environment. Thus, possible attacks on the cloud-computing environment can be classified in to:

1. **Resource attacks:**

   These kinds of attacks include manipulating the available resources into mounting a large-scale botnet attack. These kinds of attacks target either cloud providers or service providers.

2. Data attacks: These kinds of attacks include unauthorized modification of sensitive data at nodes, or performing configuration changes to enable a sniffing attack via a specific device etc. These attacks are focused on cloud providers, service providers, and also on service users.

3. Denial of Service attacks: The creation of a new virtual machine is not a difficult task, and thus, creating rogue VMs and allocating huge spaces for them can lead to a Denial of Service attack for service providers when they opt to create a new VM on the cloud. This kind of attack is generally called virtual machine sprawling.

4. Backdoor: Another threat on a virtual environment empowered by cloud computing is the use of backdoor VMs that leak sensitive information and can destroy data privacy.

5. Having virtual machines would indirectly allow anyone with access to the host disk files of the VM to take a snapshot or illegal copy of the whole System. This can lead to corporate espionage and piracy of legitimate products.

In addition to the challenges introduced by cloud, there are challenges introduced by the dynamism of virtualization in cloud:

- Dynamic relocation of virtual machines (VMs): Hypervisors today move workloads based on the service level agreement (SLA), energy policy, resiliency policy, and a host of other reasons. IT administrators of today can no longer be sure where the workload resides in the data center.
- Increased infrastructure layers to manage and protect: Depending on the type of cloud model in use, there are a large number of additional infrastructure layers such as gateways, firewalls, access routers, and others, that need to be managed and protected, at the same time allowing access to the authorized users to perform their tasks.

- Multiple operating systems and applications per server: On virtualized commodity hardware, multiple workloads on a physical server run concurrently, with multiple operating systems and even with same operating systems but at different patch levels.
- Elimination of physical boundaries between systems: As virtualization adoption increases, workloads are co-located sharing the same physical infrastructure.
- Tracking software and configuration of VMs: As IT infrastructure becomes virtualized, it is increasingly complex to manage software configuration including patch levels, security patches, security audits, and others, not just for guest operating systems but also for other virtualized infrastructure such as virtual distributed switches.

  The figure shows before and after virtualization.

Challenges with existing security products

Traditional security products encounter new challenges in the virtualized world:

- Intrusiveness of existing solutions
- Reconfiguration of virtual network: Some existing solutions might require reconfiguration of the virtual network to allow for packet sniffing and protocol examination.
- Presence in the guest OS: For monitoring purposes, agents are required to be installed on the guest OS.
- Visibility and control gaps
- Virtual servers not connected to the physical network are invisible and unprotected.
- Lacks automation and transparency.

- Static security controls are too rigid. As VMs are moved around by the hypervisor, static controls need to be reapplied.
- No ability to deal with workload mobility exists.
- Resource overhead.
- Network traffic analysis in each guest OS is redundant, consuming more CPU cycles.

Virtual Server Protection for VMware is shown in the next figure.

Virtual Server Protection for VMware provides the following benefits:

- Dynamic protection of every layer of infrastructure, mitigating the risks introduced by virtualization.
- Meets regulatory and compliance requirements.
- Increases ROI of virtual infrastructure because it is easy to deploy and maintain security.

Integrated security benefits of Virtual Server Protection for VMware are as follows:

- **Transparency**
  - No reconfiguration of virtual network required
  - No presence in guest OS
- **Security consolidation**
  - Only *one* Security Virtual Machine (SVM) required per physical server
  - 1:many protection ratio
  - **Automation**
  - Privileged presence gives SVM holistic view of the virtual network
  - Protection applied automatically as each new VM comes online
- **Efficiency**
  - Eliminates redundant processing tasks
- **Protection for any guest OS**

An example of Virtual Server Protection for VMware architecture is shown in the next figure:

In the figure example, three ESX clusters are in three separate network zones (WEB, Transact, and Black), separated physically. Virtual Server Protection for VMware is deployed on each ESX host in the cluster (HS22V blades in this example), which monitors all VMs as they are brought online. All policy events data is forwarded by the SVM to the SiteProtector appliance.

Identity Management and Access Control

IAM products provide IT managers with tools and technologies for controlling user access to critical information within an organization.

Identity and access management (IAM) is a framework of business processes, policies and technologies that facilitates the management of electronic or digital identities. With an IAM framework in place, information technology (IT) managers can control user access to critical information within their organizations. Identity and access management products offer role-based access control, which lets system administrators regulate access to systems or networks based on the roles of individual users within the enterprise.

In this context, access is the ability of an individual user to perform a specific task, such as view, create or modify a file. Roles are defined according to job competency, authority and responsibility within the enterprise.

Systems used for identity and access management include single sign-on systems, multi-factor authentication and privileged access management (PAM).

These technologies also provide the ability to securely store identity and profile data as well as data governance functions to ensure that only data that is necessary and relevant is shared. IAM systems can be deployed on premises, provided by a third-party vendor through a cloud-based subscription model or deployed in a hybrid cloud.

Basic components of IAM

On a fundamental level, IAM encompasses the following components:

- How individuals are identified in a system.
- How roles are identified in a system and how they are assigned to individuals.
- Adding, removing and updating individuals and their roles in a system.
- Assigning levels of access to individuals or groups of individuals.
- Protecting the sensitive data within the system and securing the system itself.

What IAM systems should include

Identity access management systems should consist of all the necessary controls and tools to capture and record user login information, manage the enterprise database of user identities and orchestrate the assignment and removal of access privileges. That means that systems used for IAM should provide a centralized directory service with oversight as well as visibility into all aspects of the company user base.

Technologies for identity access and management should simplify the user provisioning and account setup process. These systems should reduce the time it takes to complete these processes with a controlled workflow that decreases errors as well as the potential for abuse while allowing automated account fulfillment. An identity and access management system should also allow administrators to instantly view and change access rights.

These systems also need to balance the speed and automation of their processes with the control that administrators need to monitor and modify access rights. Consequently, to manage access requests, the central directory needs an access rights system that automatically matches employee job titles, business unit identifiers and locations to their relevant privilege levels.

Multiple review levels can be included as workflows to enable the proper checking of individual requests. This simplifies setting up appropriate review processes for higher-level access as well as easing reviews of existing rights to prevent privilege creep, the gradual accumulation of access rights beyond what users need to do their jobs.

IAM systems should be used to provide flexibility to establish groups with specific privileges for specific roles so that access rights based on employee job functions can be uniformly assigned. The system should also provide request and approval

processes for modifying privileges because employees with the same title and job location may need customized, or slightly different, access.

Benefits of identity and access management

IAM technologies can be used to initiate, capture, record and manage user identities and their related access permissions in an automated manner. This brings an organization the following benefits:

- Access privileges are granted according to one interpretation of policy and all individuals and services are properly authenticated, authorized and audited.
- Companies that properly manage identities have greater control of user access, reducing the risk of internal and external data breaches.
- Automating IAM systems allows businesses to operate more efficiently by decreasing the effort, time and money that would be required to manage access to their networks manually.
- In terms of security, the use of an IAM framework can make it easier to enforce policies around user authentication, validation and privileges and address issues regarding privilege creep.
- IAM systems help companies better comply with government regulations by allowing them to show that corporate information is not being misused. Companies can also demonstrate that any data needed for auditing can be made available on-demand.

Additionally, by implementing identity access management tools and following related best practices, a company can gain a competitive edge. For example, IAM technologies allow the business to give users outside the organization, like customers, partners, contractors and suppliers, access to its network across mobile applications, on-premises apps and software-as-a-service apps without compromising security. This enables better collaboration, enhanced productivity, increased efficiency and reduced operating costs.

IAM in the enterprise

It can be challenging to get funding for IAM projects because they do not directly increase an organization's profitability or functionality. However, a lack of effective identity and access management poses significant risks not only to compliance, but also overall security. These mismanagement issues increase the risk of greater damages from both external and internal threats.

Keeping the required flow of business data going while simultaneously managing its access has always required administrative attention. The business IT environment is ever evolving and the difficulties have only become greater with recent disruptive trends like bring your own device, cloud computing, mobile apps and an increasingly mobile workforce. There are more devices and services to be managed than ever before, with diverse requirements for associated access privileges.

Risks associated with IAM

Implementing proper identity and access management tools or platforms means storing all authorizations and credentials in one, unified place. When not secured

correctly, this can be a huge risk because if an attacker gains access to the system, all digital identities can be compromised. Similarly, if a specific employee that is authorized to the system does not follow security or password best practices, all of the information could be easily leaked.

Another concern for adopting IAM are challenges in implementation. Legacy systems will typically already have an identity management functionality in place, therefore, converting resources to a new system could be challenging, expensive and time-consuming. However, solutions for minimizing the need of technical support, such as cloud services, are becoming more viable.

IAM vendors and tools

Rather than developing internal tools, most companies decide to purchase or subscribe to third-party IAM tools. These products can take on multiple forms, such as an identity as a service (IDaaS) cloud model, a hybrid cloud model, a traditional on-premise model or a microservices model. IAM microservices may cover only one aspect of IAM like privileged account management, account compliance management or user authorization management.

Vendors with products in the IAM space include:

- Microsoft Azure Active Directory.
- IBM Security Identity and Access Assurance.
- Oracle Identity Cloud Service.
- Okta.
- Centrify.
- RSA SecurID Access.

Autonomic Security.

Autonomic computing is a computing model in which the system is self-healing, self-configured, self-protected and self-managed. Designed to mimic the human body's nervous system-in that the autonomic nervous system acts and reacts to stimuli independent of the individual's conscious input-an autonomic computing environment functions with a high level of artificial intelligence while remaining invisible to the users.

The autonomic computing initiative (ACI), which was developed by IBM, demonstrates and advocates networking **computer systems that do not involve a lot of human intervention other than defining input rules.** The ACI is derived from the autonomic nervous system of the human body.

IBM has defined the four areas of automatic computing:

- Self-Configuration.
- Self-Healing (error correction).
- Self-Optimization (automatic resource control for optimal functioning).

Self-Protection (identification and AC was **designed to mimic the human body's nervous system**-in that the autonomic nervous system acts and reacts to stimuli independent of the individual's conscious input-an autonomic computing

environment functions with a high level of **artificial intelligence** while remaining invisible to the users. Just as the human body acts and responds without the individual controlling functions (e.g., internal temperature rises and falls, breathing rate fluctuates, glands secrete hormones in response to stimulus), the autonomic computing environment operates organically in response to the input it collects.

IBM has set forth eight conditions that define an autonomic system:

1. The system must know itself in terms of what resources it has access to, what its capabilities and limitations are and how and why it is connected to other systems.
2. The system must be able to automatically configure and reconfigure itself depending on the changing computing environment.
3. The system must be able to optimize its performance to ensure the most efficient computing process.
4. The system must be able to work around encountered problems by either repairing itself or routing functions away from the trouble.
5. The system must detect, identify and protect itself against various types of attacks to maintain overall system security and integrity.
6. The system must be able to adapt to its environment as it changes, interacting with neighboring systems and establishing communication protocols.
7. The system must rely on open standards and cannot exist in a proprietary environment.
8. The system must anticipate the demand on its resources while keeping transparent to users.

Autonomic computing is one of the building blocks of ***pervasive computing***, an anticipated future computing model in which tiny – even invisible – computers will be all around us, communicating through increasingly interconnected networks leading to the concept of The Internet of Everything (IoE). Many industry leaders are researching various components of autonomic computing.

UNIT I INTRODUCTION TO DBMS

Syllabus:

Introduction - Database System Applications-Purpose of Database Systems- View of Data- Data Abstraction- Instances and Schemas- Relational Databases - Database Design- Data model- Integrity constraints -The Entity - Relationship Model - Transaction Management - Database Architecture - Data Storage and Querying - Database Users and Administrators.

DBMS stands for **D**ata**b**ase **M**anagement **S**ystem. We can break it like this DBMS = Database + Management System. Database is a collection of data and Management System is a set of programs to store and retrieve those data.

Definition:

DBMS is a collection of inter-related data and set of programs to store & access those data in an easy and effective manner.

What is the need of DBMS?

Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization:

1. **Storage of data.**
2. **Retrieval of data**.

Storage:

- According to the principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage.
- Let's take a layman example to understand this:
  In a banking system, suppose a customer is having two accounts, one is saving account and another is salary account.
- Let's say bank stores saving account data at one place (these places are called tables we will learn them later) and salary account data at another place, in that case if the customer information such as customer name, address etc. are stored at both places then this is just a wastage of storage (redundancy/ duplication of data), to organize the data in a better way the

information should be stored at one place and both the accounts should be linked to that information somehow. The same thing we achieve in DBMS.

Fast Retrieval of data:

- Along with storing the data in an optimized and systematic manner, it is also important that we retrieve the data quickly when needed.
- Database systems ensure that the data is retrieved as quickly as possible.

Applications of Database Management Systems:

- **Telecom**: There is a database to keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
- **Industry**: Where it is a manufacturing unit, warehouse or distribution Centre, each one needs a database to keep the records of ins and outs. For example distribution Centre should keep a track of the product units that supplied into the Centre as well as the products that got delivered out from the distribution Centre on each day; this is where DBMS comes into picture.
- **Banking System**: For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.
- **Education sector**: Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a hell lot amount of inter-related data that needs to be stored and retrieved in an efficient manner.
- **Online shopping**: You must be aware of the online shopping websites such as Amazon, Flipkart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system.

Drawbacks of File system:

- **Data Isolation**: Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

- **Duplication of data** – Redundant data
- **Dependency on application programs** – Changing files would lead to change in application programs.

Advantage of DBMS over File System:

There are several advantages of Database management system over file system. Few of them are as follows:

1. No redundant data – Redundancy removed by data normalization
2. Data Consistency and Integrity – data normalization takes care of it too
3. Secure – Each user has a different set of access
4. Privacy – Limited access
5. Easy access to data
6. Easy recovery
7. Flexible

Disadvantages of DBMS:

1. DBMS implementation cost is high compared to the file system
2. Complexity: Database systems are complex to understand
3. Performance: Database systems are generic, making them suitable for various applications. However this feature affect their performance for some applications

DATA MODELS IN DBMS

Data model - A collection of concepts that can be used to describe the structure of a database - provides the necessary means to achieve the abstraction.

A **Data Model** is a logical structure of Database. It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.

It is a tool for data abstraction. A collection of tools for describing

- data
- data relationships
- data semantics
- data constraints

A data model is described by the schema, which is held in the data dictionary.

- Student(Stu_Id, Stu_Name, Stu_Add)
- Course(Course_Id, lecturer)

Categories of Data Models:

Many data models have been proposed, which can categories according to the types of concepts they use to describe the database structure.

1. High level or Conceptual data models.
   - Entity Relationship (E-R) model.
   - Object model
2. Low level or Physical data model.
3. Representational or Implementation data model.
   - Relational model.
   - Network model.
   - Hierarchical model.

High-level or **conceptual data models:**

A conceptual data model identifies the highest-level relationships between the different entities. Features of conceptual data model include:

- Includes the important entities and the relationships among them.
- No attribute is specified.
- No primary key is specified.

Physical (low-level) data models:

Physical data model represents how the model will be built in the database.

A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables.

Features of a physical data model include:

- Specification all tables and columns.
- Foreign keys are used to identify relationships between tables.
- Denormalization may occur based on user requirements.
- Physical considerations may cause the physical data model to be quite different from the logical data model.
- Physical data model will be different for different RDBMS. For example, data type for a column may be different between MySQL and SQL Server.

Implementation (Representational) data models:

A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. Features of a logical data model include:

- Includes all entities and relationships among them.
- All attributes for each entity are specified.
- The primary key for each entity is specified.
- Foreign keys (keys identifying the relationship between different entities) are specified.
- Normalization occurs at this level.

The steps for designing the logical data model are as follows:

1. Specify primary keys for all entities.
2. Find the relationships between different entities.
3. Find all attributes for each entity.
4. Resolve many-to-many relationships.
5. Normalization.

Conceptual, Logical, and Physical Data Models

Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

INSTANCES AND SCHEMAS:

Schema: A logical structure of a database is called the schema. Schema is of three types.

1. Physical schema or internal schema
2. Logical schema
3. View schema or external schema

- The design of a database at physical level is called **physical schema**, how the data stored in blocks of storage is described at this level.
- Design of database at logical level is called **logical schema**, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).
- Design of database at view level is called **view schema**. This generally describes end user interaction with database systems.

Instance: The data stored in database at a particular moment of time is called instance of database.

Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

VIEWS OF DATA:

Abstraction is one of the main features of database systems. Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient **user-database** interaction.

To understand the view of data, you must have a basic knowledge of data abstraction and instance & schema.

Data Abstraction in DBMS

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.

Three levels of abstraction:

Physical level: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

View level: Highest level of data abstraction. This level describes the user interaction with database system.

Example: Let's say we are storing customer information in a customer table. At **physical level** these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are often hidden from the programmers.

At the **logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

At **view level**, user just interact with system with the help of GUI and enter the details at the screen, they are not aware of how the data is stored and what data is stored; such details are hidden from them.

Data Independence:

Change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. **Logical data independence -** is the capacity to change the conceptual schema without having to change external schemas or application programs.
2. **Physical data independence -** is the capacity to change the internal schema without having to change conceptual schemas.

3. Applications depend on the logical schema. In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others

DBMS LANGUAGES:

Database languages are used for read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language).

Types of DBMS languages:

1. Data Definition Language (DDL): DDL is used for specifying the database schema. Let's take SQL for instance to categorize the statements that comes under DDL.

- To create the database instance – CREATE
- To alter the structure of database – **ALTER**
- To drop database instances – DROP
- To delete tables in a database instance – **TRUNCATE**
- To rename database instances – **RENAME**

All these commands specify or update the database schema that's why they come under Data Definition language.

2. Data Manipulation Language (DML): DML is used for accessing and manipulating data in a database.

- To read records from table(s) – SELECT
- To insert record(s) into the table(s) – **INSERT**
- Update the data in table(s) – UPDATE
- Delete all the records from the table – DELETE

3. Data Control language (DCL): DCL is used for granting and revoking user access on a database –

- To grant access to user – GRANT
- To revoke access from user – REVOKE

In practical data definition language, data manipulation language and data control languages are not separate language; rather they are the parts of a single database language such as SQL.

4. **Transaction Control Language (TCL):** Transaction Control Language statements are used to control the transactions in the database.
    - SAVE POINT
    - ROLLBACK
    - COMMITE

DATABASE SYSTEM ARCHITECTURE

Components of DB Architecture

The DBA staff: Works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands.

Casual users: Those who work with interactive interfaces to formulate queries.

Application programmers: Those who create programs using some host programming languages

Parametric users: Who does data entry work by supplying parameters to predefined transactions?

The DDL compiler: Processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.

Query Compiler: Interactive Queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a query compiler that compiles them into an internal form.

The Pre-Compiler: Application programmers write programs in host languages such as Java, C, or C++ that are submitted to a PreCompiler. It extracts DML commands from an application program written in a host programming language.

DML Compiler: DML commands are sent to the DML compiler for compilation into object code for database access

Host Language Compiler: The rest of the program which is extracted from precompiler is sent to the host language compiler

The query optimizer:

- It is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of correct algorithms and indexes during execution.
- It consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the runtime processor.

Runtime Database Processor: The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor.

The System Catalog/Data Dictionary:

- The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints.
- In addition, the catalog stores many other types of information that are needed by the DBMS modules, which can then look up the catalog information as needed.

Stored data manager: Which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory?

Concurrency control / backup / recovery systems: They are integrated into the working of the runtime database processor for purposes of transaction management.

Stored Database: It is storing database in disk.

DATABASE USERS AND ADMINISTRATOR:

Database Users

Users are differentiated by the way they expect to interact with the system.

- **Application programmers**: interact with system through DML calls.
- **Sophisticated users** form requests in a database query language
- **Specialized users** – write specialized database applications that do not fit into the traditional data processing framework
- **Naive users** – invoke one of the permanent application programs that have been written previously

Database Administrator:

- Coordinates all the activities of the database system.
- The database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties (Roles) include:
  - Schema definition
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Specifying integrity constraints
  - Acting as liaison with users
  - Monitoring performance and responding to changes in requirements

E-R MODEL

An **Entity–Relationship Model (ER model)** is a systematic way of describing and defining a business process. An ER model is typically implemented as a database. The main components of E-R model are: entity set and relationship set.

Here are the geometric shapes and their meaning in an E-R Diagram –

Rectangle: Represents Entity sets.

Ellipses: Attributes

Diamonds: Relationship Set

Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

Double Ellipses: Multivalued Attributes

Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets

Double Lines: Total participation of an entity in a relationship set

A sample E-R Diagram:

Sample E-R Diagram

An **Entity** is represented by a set of attributes, which are descriptive properties possessed by all members of an entity set.

<u>Attribute types:</u>

- *Simple* and *composite* attributes.
- *Single-valued* attributes.
- *Multi-valued* attributes.

An attribute that can hold multiple values is known as multivalued attribute. We represent it with double ellipses in an E-R Diagram.

E.g. A person can have more than one phone numbers so the phone number attribute is multivalued.

- *Derived* attributes:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by dashed ellipses in an E-R Diagram.

E.g. Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

E-R diagram with multivalued and derived attributes:

Total Participation of an Entity set:

E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

E-R Diagram with Composite, Multivalued, and Derived Attributes

Cardinality Constraints:

We express ***cardinality constraints*** by drawing either a directed line (→), signifies "**one**" An undirected line (—), signifies "**many**" between the relationship set and the entity set.

One-to-one relationship

- A customer is associated with at most one loan via the relationship *borrower*
- A loan is associated with at most one customer via *borrower*

One-To-Many Relationship:

In the one-to-many relationship a loan is associated with at most one customer via *borrower*; a customer is associated with several loans via *borrower*.

Many-To-One Relationships

Customers via *borrower*, a customer is associated with at most one loan via *borrower*

Many-To-Many Relationship

- A customer is associated with several loans via borrower
- A loan is associated with several customers via borrower

Specialization

- Top-down design process: We designate subgroupings within an entity set that are distinctive from other entities in the set.
- These sub groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (E.g. *customer* "is a" *person*).
- Attribute inheritance – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked

Generalization

- A bottom-up design process – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

Relational model in DBMS

In relational model, the data and relationships are represented by collection of inter-related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records.

Sample relationship Model: Student table with 3 columns and four records.

Stu_Id	**Stu_Name**	**Stu_Age**
BTU10869	Bumesh	20
BTU10870	Pavan	20
BTU10864	Tharun	19
BTU10879	Arun	19

Course table: Course table

Stu_Id	**Course_Id**	**Course_Name**
BTU10869	Co1	Cloud Computing
BTU10870	Co2	OS
BTU10864	C22	DBMS
BTU10879	C39	Computer Networks

Here Stu_Id, Stu_Name & Stu_Age are attributes of table Student and Stu_Id, Course_Id & Course_Name are attributes of table Course. The rows with values are the records (commonly known as tuples).

RDBMS Concepts

RDBMS stands for relational database management system. A relational database has following major components: Table, Record / Tuple, Field & Column /Attribute.

Table:

A table is a collection of data represented in rows and columns. For e.g. following table stores the information of students.

Stu_Id	**Stu_Name**	**Stu_Add**	**Stu_Age**
BTU101	Lokeshwari	Dhayal bagh, Agra	20
BTU102	Bhavana	Delhi	20
BTU103	Saya	Gurgaon	20
BTU104	Bharathi	Chennai	20

Records / Tuple:

Each row of a table is known as record or it is also known as tuple. For e.g. the below row is a record.

BTU102	Bhavana	Delhi	20

Field:

The above table has four fields: Stu_Id, Stu_Name, Stu_Addr & Stu_Age.

Column / Attribute:

Each attribute and its values are known as attributes in a database. For e.g. Set of values of Stud_Id field is one of the four columns of the Student table.

Stu_Id
BTU101
BTU102
BTU103
BTU104

Hierarchical model in DBMS

In **hierarchical model**, data is organized into a tree like structure with each record is having one parent record and many children. The main drawback of this model is that, it can have only one to many relationships between nodes.

Sample Hierarchical Model Diagram:

Example of hierarchical data represented as relational tables: The above hierarchical model can be represented as relational tables like this:

Stu_Id	Stu_Name	Stu_Age
BTU123	Mani Deep	19
BTU367	Sivasaikirann	20
BTU234	Rohith	20

Course Table:

Course_Id	Course_Name	Stu_Id
C01	Cobol	BTU123
C02	Java	BTU367
C21	Perl	BTU367
C33	JQuery	BTU234

UNIT 2: RELATIONAL DATABASES

Syllabus:

Introduction to Relational model - Structure of Relational Databases - Relational database design - keys - Database Schema and Schema Diagrams - Relational Query Languages - Relational algebra and Relational Operations - Relational calculus -Introduction to SQL - Overview of the SQL Query Language - Basic Structure of SQL Queries - Functional dependency -Normal Forms.

RELATIONALDATAMODEL

- The relational model was introduced by **Dr. E.F Codd**in **1970**.
- The relation at model represents data in the form of two-dimensional tables. Each table represents some real-world entity or thing.

Characteristics of Relational Model

- This model data in the database as simple **row/column**.
- Each table is an **independent entity** and there is no physical relationship between tables.
- Relational model of data management is based on **set theory**.
- The user interface used in relational models in non-procedural because only **what needs to be done** is specified and **not how it has to be done**.

E.F. CODD'S LAWS

1. **Information representation**:

   All information stored in relational database is represented only by data item values, which are stored in tables.

2. **Logical accessibility**:

   Every data item value stored in a relational database is accessible by stating the name of the table it is stored in, the name of the column stored and the primary key that defines the row in which the data stored

3. **Representation of null values**:

   The DBMS has a consistent method for representing null values.

4. **Catalog Facilities**:

The logical description of RDBMS is represented as same manner of ordinary data, this is done so that facilities of RDBMS itself can be used to maintain database description.

5. Data Language:

A RDBMS may support many types of languages for describing data and accessing the database.

6. View Updatability:

Any view can be defined using combination of base tables that are theoretically updatable, is capable of being updated by RDBMS.

7. Insert, update and delete:

Any operand that describes the result of a single retrieval operation is capable of being applied to an insert, update and delete.

8. Physical data independence:

Changes made to physical storage or access methods do not require changes in application program.

9. Logical data independence:

Changes made to tables or do not require changes in application program.

10. Integrity constraints:

Constraints that apply to entity integrity and referential integrity are specifiable by the data language implemented by the DBMS and not by the statements coded into the application program.

11. Database Distribution:

The database language implemented by the RDBMS supports the ability to distribute the database without requiring the changes to be made to the application program.

12. Non-Subversion:

If the RDBMS supports facilities that allow application programs to operate on tables a row at a time, an application program using this type of database access is prevented from bypassing entity integrity or referential integrity constraints.

PRINCIPLE COMPONENTS OF RELATIONAL MODEL

1. Data Structure
2. Data integrity
3. Data Manipulation

1. Relational Data Structure

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Stu_Rollno, Stu_Name,etc.
2. **Tables** – In the Relational data model, the relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. ***Rows represent records*** and ***columns represent attributes***.
3. **Tuple** – It is nothing but a ***single row of a table***, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system.
8. Pool of value is **Domain**

Formal relational term	Informal equivalence
relation	table
tuple	row or record
cardinality	number of rows
attribute	column or field
degree	number of columns
primary key	unique identifier
domain	pool of legal values

2. Date Integrity:

Integrity constraints provide a means of ensuring that changes made to the database by authorized users do not result in a loss of data consistency.

Following are types of integrity constraints -

1. Domain Constraints

2. Referential integrity
3. Nulls
4. Entity Integrity
5. Enterprise constraints

- **Domain Constraints:** Domain constraints specify the set of values that can be associated with an attribute. They are tested easily by the system whenever a new data item is entered into the database
- **Referential Integrity:** A value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. This is called as referential integrity.
- **Nulls:** Null represents a value for an attribute that is currently unknown or is not applicable for this tuple.
- **Entity Integrity:** In base relation no attribute of primary key can be null. A primary key is used to identify the tuple uniquely.
- **Enterprise Constraints:** These are additional rules specified by the users or administrators.

3. Data Manipulation:

The manipulative part of relational model consists of a set of operators known collectively as the ***relational algebra*** together with ***relational calculus***.

Advantages of relational model

- **Simplicity:** A relational data model is simpler than the hierarchical and network model.
- **Structural Independence:** The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- **Easy to use:** The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
- **Query capability:** It makes possible for a high-level query language like SQL to avoid complex database navigation.
- **Data independence:** The structure of a database can be changed without having to change any application.
- **Scalable:** Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages of relational model

- Few relational databases have ***limits on field lengths*** which can't be exceeded.
- Relational databases can ***sometimes become complex*** as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems ***may lead to isolated databases*** where the information cannot be shared from one system to another.

KEYS:

What are Keys?

- A DBMS key is an attribute or set of an attribute which helps you to **identify a row (tuple*)*** in a relation (table).
- They allow you to **find the relation between two tables**.
- Keys help you **uniquely identify a row** in a table by a combination of one or more columns in that table.

Example:

STUD_ID	STUD_NAME	EMAIL_ID
U18CN167	Manideep	u18cn167@biher.edu.in
U18CN187	Vishal	u18cn187@biher.edu.in
U18CN189	Lokeshwari	u18cn189@biher.edu.in
U18CN200	Pavan Satya	u18cn200@biher.edu.in

In the above-given example, **STUD_ID is a primary key** because it uniquely identifies a student record. In this table, no other student can have the same STUD_ID.

Why we need a Key?

- Keys help you to identify any row of data in a table.
- Keys allows you to establish a relationship between and identify the relation between tables
- Keys help you to enforce identity and integrity in the relationship.

Various Keys in Database Management System

DBMS has flowing seven types of Keys each have their different functionality:

1. Super Key
2. Primary Key

3. Candidate Key
4. Alternate Key
5. Foreign Key
6. Compound Key
7. Composite Key
8. Surrogate Key

1) Primary Key:

- A column or group of columns in a table which helps us to ***uniquely identify every row*** in the table is called a primary key.
- The same value ***can't appear more than once*** in the table.
- ***Rules*** for defining Primary key:
  1. The primary key field ***cannot be null***.
  2. The value in a primary key column ***can never be modified or updated*** if any foreign key refers to that primary key.

STUD_ID	STUD _NAME	EMAIL_ID
U18CN167	Manideep	u18cn167@biher.edu.in
U18CN187	Vishal	u18cn187@biher.edu.in
U18CN189	Lokeshwari	u18cn189@biher.edu.in
U18CN200	Pavan Satya	u18cn200@biher.edu.in

- From the above example, STUD_ID is a primary key attribute.

2) Candidate Key:

- ***"A candidate key is an attribute or set of an attribute which can uniquely identify a tuple***."
- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

Example:

- In the STUDENT table, STUD_ID is best suited for the primary key.
- Rest of the attributes like REG_NO, and EMAIL_ID are considered as candidate keys which help us to uniquely identify the student record in the table.

STUD_ID	REG_NO	STUD _NAME	EMAIL_ID
U18CN167	1XXX101	Manideep	u18cn167@biher.edu.in
U18CN187	1XXX102	Vishal	u18cn187@biher.edu.in
U18CN189	1XXX103	Lokeshwari	u18cn189@biher.edu.in
U18CN200	1XXX112	Pavan Satya	u18cn200@biher.edu.in

3) Super Key:

- ***"Super key is a set of an attribute which can uniquely identify a tuple***."
- Super key is a superset of a candidate key.

Example: In the above STUDENT table, for (**STUD_ID, STUD_NAME**) the name of two students can be the same, but their **STUD_ID** can't be the same.

Hence, this combination can also be a key.

- The super key would be **STUD_ID**, (**STUD_ID, STUD _NAME**), etc.

4) Alternate Keys:

- ***"Alternate Keys is a column or group of columns in a table that uniquely identify every row in that table."***
- A table can have multiple choices for a primary key but only one can be set as the primary key. All the ***keys which are not primary key are called an Alternate Key***.

Example:

- In this table, STUD_ID, REG_NO, EMAIL_ID are qualified to become a primary key. But since STUD_ID is the primary key, ***REG_NO, EMAIL_ID becomes the alternative key***.

STUD_ID	REG_NO	STUD_NAME	EMAIL_ID
U18CN167	1XXX101	Manideep	u18cn167@biher.edu.in
U18CN187	1XXX102	Vishal	u18cn187@biher.edu.in
U18CN189	1XXX103	Lokeshwari	u18cn189@biher.edu.in
U18CN200	1XXX112	Pavan Satya	u18cn200@biher.edu.in

5) Foreign Key:

- ***"Foreign Key is a column that creates a relationship between two tables."***
- The purpose of Foreign Keys is to ***maintain data integrity*** and allow ***navigation between two different instances*** of an entity.

- It acts as a ***cross-reference between two tables*** as it references the primary key of another table.

Table Name: Faculty

Faculty_Id	Fac_Name	Dept_Code	Email_Id
BU2702	Nithiyanandam	BTUCS51	nithiyanandam@biher.edu.in
BU2703	Janagiraman	BTUCS51	janagiraman@biher.edu.in
BU2543	Elankavi	BTUCE56	elankavi@biher.edu.in
BU2427	Sivaranjani	BTUCS51	ranjibala@gmail.com
BU10270	Arivuselvi	BYUEC55	Arivuselvi270@yahoo.com

Table Name: Department

Dept_Code	Dept_Name
BTUCS51	CSE
BTUCE56	CIVIL
BTUEC55	ECE
BTUEE58	EEE
BTUME60	MECH

6) Compound Key:

- ***"Compound Key has two or more attributes that allow you to uniquely recognize a specific record. "***
- It is possible that each column may not be unique by itself within the database.
- However, when combined with the other column or columns the ***combination of composite keys become unique***.
- The purpose of ***compound key is to uniquely identify each record*** in the table.

RELATIONAL ALGEBRA

- Relational algebra presents the ***basic set of operations*** for relational model.
- It is a ***procedural language***, which describes the procedure to obtain the result.
- Relational algebra is prescriptive because it describes the **order of operations** in the query that specifies ***how*** to retrieve the result of a query.

Relational Algebra Operation:

a) **Selection**
b) **Projection**
c) **Rename**
d) **Union**
e) **Intersection**
f) **Set difference**
g) **Cartesian product**
h) **Join**
i) **division**
j) **Aggregate function**
k) **Outer Join**

a) <u>Select Operation</u>:

- The select operation selects the tuple that satisfy a given predicate (condition).
- The select operation is represented as follows

  $\sigma_{<Condition>}$ **(Relation)**

  σ - Denote the select operator

  Condition – Denotes **expression** specified on the attributes of the **Relation** (Table).

Table Name: **BOOK_DETAIL**

B_ID	TITTLE	AUTHOR	PUBLISHER	YEAR	PRICE
B0001	DBMS	Ramez Elmarsi	McGraw-Hill	2000	250
Book2	Java Programming	K Somasundaram	Jaico Publishing	2004	350
B00k3	Computer Network	S Tanenbaum	Pearson	2003	450
Book4	Concrete Maths	Donald Knuth		2000	500

Select Condition:

- Operators: $<, \leq, \geq, >, =, \neq$
- Simple selection condition:

  <attribute>operator<constant>

  <attribute>operator<attribute>

  <condition> AND <condition>

  <condition> OR <condition>

  NOT <condition>

Example:

1) Display the book detail published in the year 2000

$\sigma_{year=2000}$ (Book_Detail)

B_ID	TITTLE	AUTHOR	PUBLISHER	YEAR	PRICE
B0001	DBMS	Ramez Elmarsi	McGraw-Hill	2000	250
Book4	Concrete Maths	Donald Knuth		2000	500

2) Display the books detail having price greater than 300.

$\sigma_{price>300}$ (Book_Detail)

B_ID	TITTLE	AUTHOR	PUBLISHER	YEAR	PRICE
Book2	Java Programming	K Somasundaram	Jaico Publishing	2004	350
B00k3	Computer Network	S Tanenbaum	Pearson	2003	450
Book4	Concrete Maths	Donald Knuth		2000	500

3) Display the books details whose publishing year is 2000 and the price is greater than 300.

$\sigma_{year=200 \text{ AND } price>300}$ (Book_Detail)

B_ID	TITTLE	AUTHOR	PUBLISHER	YEAR	PRICE
Book4	Concrete Maths	Donald Knuth		2000	500

b) <u>Project Operation</u>:

The projection operation ***selects certain attributes*** from the relation while discarding others.

The projection operation is represented as follows

$\Pi_{attribute\ list}(Relation)$

Table Name: BOOK_DETAIL

B_ID	TITLE	AUTHOR	PUBLISHER	YEAR	PRICE
B0001	DBMS	RamezElmarsi	McGraw-Hill	2000	250
Book2	Java Programming	K Somasundaram	Jaico Publishing	2004	350
B00k3	Computer Network	S Tanenbaum	Pearson	2003	450
Book4	Concrete Maths	Donald knuth		2000	500

Examples:

1) Display all book titles with author name

$\Pi_{title,\ author}$ (Book_detail)

TITLE	AUTHOR
DBMS	Ramez Elmarsi
Java Programming	K Somasundaram
Computer Network	S Tanenbaum
Concrete Maths	Donald Knuth

Composition of Select and Project Operation:

The relational operations select and project can be combined to form a complicated query.

Example:

a) Display the Book titles having price greater than 300.

$\Pi_{title,\ price}(\sigma_{price>300}$ (Book_Detail)

TITLE	PRICE
Java Programming	350
Computer Network	450
Concrete Maths	500

c) <u>Rename Operation</u>:

- In the relational algebra, you can rename either the *relation* or *the attributes* or *both*.
- The symbol $\boldsymbol{\rho}$ (rho) is used to denote the rename operator.
  1. $\boldsymbol{\rho}$ S (new attribute names) (R)
  2. $\boldsymbol{\rho}$ S (R)
  3. $\boldsymbol{\rho}$ (new attribute names) (R)

'**S**' is the new relation and '**R**' is original relation.

Table Name: BOOK_DETAIL

B_ID	TITLE	AUTHOR	PUBLISHER	YEAR	PRICE
B0001	DBMS	Ramez Elmarsi	McGraw-Hill	2000	250
Book2	Java Programming	K Somasundaram	Jaico Publishing	2004	350

B00k3	Computer Network	S Tanenbaum	Pearson	2003	450
Book4	Concrete Maths	Donald Knuth		2000	500

The Rename operator used on Book_Detail relation as follows

1) $\rho_{\text{Book(bk_no, bk_title, bk_author, bk_publisher, pub_year, b_price)}}$ (Book_Detail)

   - Here Relation name and attribute names are renamed.

2) ρ_{Book} (Book_Detail)

   - Here Relation name is renamed.

3) $\rho_{\text{(bk_no, bk_title, bk_author, bk_publisher, pub_year, b_price)}}$(Book_Detail)

   - Here attribute names are renamed.

SET OPERATION:

Union, intersection and **difference -** These operations require that the tables involved be ***union compatible***. Two relations are said to be union compatible if the following conditions are satisfied.

1. The two relations must contain the ***same number of columns***
2. Each column of the first relation must be ***either same data type*** or ***convertible to the same data type*** as corresponding column of the second relation

d) <u>Union Operation</u>:

- The operation is denoted by ***Depositor ∪ Borrower*** that includes all the tuples are either in depositor or borrower.
- Duplicates are eliminated.

Table Name: DEPOSITOR

CUST_ID	CUST_NAME	CITY
5432331001	SAI BHARATH	AVADI
5432331002	K.BALACHANDAR	AVADI
5432331003	MADINA SUSHMA	T NAGAR
5432331003	PRATHYUSHA	T NAGAR
5432331005	NITISH	AMBATTUR
5432331006	SAI KRISHNA	GUINDY
5432331007	P.DIVYA KRISHNA	PAADI
5432331008	G.REETHIKA	AMBATTUR
5432331009	PADMINI	PORUR
5432331010	SAI MAHESH	GUINDY

Table Name: BORROWER

BR_ID	CUST_NAME	CITY
5432331001	SAI BHARATH	AVADI
5432331014	VINAY	PORUR
5432331003	MADINA SUSHMA	T NAGAR
5432331078	ARIJITH	ANNA NAGAR
5432331089	DILEEP	ANNA NAGAR
5432331008	G.REETHIKA	AMBATTUR
5432331010	SAI MAHESH	GUINDY

Table Name: DEPOSITOR

CUST_ID	CUST_NAME	CITY
5432331001	SAI BHARATH	AVADI
5432331002	K. BALACHANDAR	AVADI
5432331003	MADINA SUSHMA	T NAGAR
5432331003	PRATHYUSHA	T NAGAR
5432331005	NITISH	AMBATTUR
5432331006	SAI KRISHNA	GUINDY
5432331007	P. DIVYA KRISHNA	PAADI
5432331008	G. REETHIKA	AMBATTUR
5432331009	PADMINI	PORUR
5432331010	SAI MAHESH	GUINDY

Table Name: BORROWER

BR_ID	CUST_NAME	CITY
5432331001	SAI BHARATH	AVADI
5432331014	VINAY	PORUR
5432331003	MADINA SUSHMA	T NAGAR
5432331078	ARIJITH	ANNA NAGAR
5432331089	DILEEP	ANNA NAGAR
5432331008	G. REETHIKA	AMBATTUR
5432331010	SAI MAHESH	GUINDY

DEPOSITOR U BORROWER

CUST_NAME	CITY
REETHIKA	AMBATHUR
ARIJITH	ANNA NAGAR
DILEEP	ANNA NAGAR
K. BALACHANDAR	AVADI
MADINA SUSHMA	T NAGAR
NITISH	AMBATHUR
P. DIVYA KRISHNA	PAADI
PADHMINI	PORUR
PRATHYUSHA	T NAGAR
REETHIKA	AMBATHUR
SAI BHARATH	AVADI
SAI BHARATH	AVADI
SAI KRISHNA	GUINDY
SAI MAHESH	GUINDY
VINAY	PORUR

e) <u>Intersection Operation</u>:

- It includes all the tuples that are present in both depositor and borrower relations.
- The operation is denoted by ***Depositor*** ∩ ***Borrower.***

DEPOSITOR ∩ BORROWER

CUST_NAME	CITY
HARISH	GUINDY
NIKITHA P	AVADI
PADMINI	AMBATTUR
VISHNU TEJA	T NAGAR

f) <u>Difference Operation</u>:

- The difference operation is denoted by ***Depositor – Borrower***.
- It results the relation that contains all tuples in depositor but not in borrower.

DEPOSITOR – BORROWER

CUST_NAME	CITY
CHANDANA PRIYA	AVADI
GOPALAM KESAVA	PAADI
MOHITH	AMBATTUR
NACHIKETA KUMAR	PORUR
YASHWANTH	T NAGAR
YUVRAJ	GUINDY

BORROWER – DEPOSITOR

CUST_NAME	CITY
ARYAN	ANNA NAGAR
KUSHAL	PORUR
VASUJIT	ANNA NAGAR

g) <u>Cartesian Product</u>:

- The Cartesian product is also known as ***Cross Product*** or ***Cross Joins***, it is denoted by '**X**'.
- The Cartesian product of two relation A and B is denoted as **A X B**.

- When a ***join condition is omitted*** when getting result from two tables then that kind of query gives us Cartesian product, in which all combination of rows is displayed.
- All rows in the first table is joined to all rows of second table.
  - ✓ The result of Cartesian product of two relation which have **m** and **n columns** is a relation that has **m + n columns**.
  - ✓ Hence if the relation will have **x and y** tuples respectively, then the Cartesian product will have **n * m tuples**.

Table Name: **AUTHOR**

AUTHOR_ID	AUTHOR_NAME
125001	Ramez Elmarsi
125010	K Somasundaram
125017	S Tanenbaum
125020	Donald Knuth

Table Name:**BOOK_DETAIL**

BOOK_ID	TITLE
2578	DATABASE SYSTEMS
2568	JAVA PROGRAMMING
2588	COMPUTER NETWORKS
2548	CONCRETE MATHEMATICS

AUTHOR x BOOK_DETAIL

AUTHOR_ID	AUTHOR_NAME	BOOK_ID	BOOK_NAME
125001	Ramez Elmarsi	2578	database systems
125001	Ramez Elmarsi	2568	Java programming
125001	Ramez Elmarsi	2588	Computer networks
125001	Ramez Elmarsi	2548	Concrete mathematics
125010	k somasundaram	2578	database systems
125010	k somasundaram	2568	Java programming
125010	k somasundaram	2588	Computer networks
125010	k somasundaram	2548	Concrete mathematics
125017	S Tanenbaum	2578	database systems
125017	S Tanenbaum	2568	Java programming
125017	S Tanenbaum	2588	Computer networks
125017	S Tanenbaum	2548	Concrete mathematics
125020	Donald knuth	2578	database systems
125020	Donald knuth	2568	Java programming
125020	Donald knuth	2588	Computer networks
125020	Donald knuth	2548	Concrete mathematics

Activate Win
Go to Settings to

h) Natural Join:

- The natural join is a ***binary operation*** that allow us to ***combine certain selections and a Cartesian product*** into one operation.
- It is denoted by the join symbol ⋈.
- ***Duplicate columns eliminated*** from the result.

Table Name: **EMPLOYEE**

EMP_ID	EMP_NAME
E6501	BIKRAM SHAH
E6502	SOURAV KUMAR
E6503	RASHEED ALI
E6504	AJIT KUMAR

Table Name: **SALARY**

EMP_ID	SALRY
E6501	45000
E6502	43000
E6503	45000
E6504	44000

Example: Display the names of all employees with salary, we need to combine the employee and salary relation using ***Cartesian product without duplicated tuples***

$$\Pi_{emp_name,\ salry}(\sigma_{employee.emp_id\ =\ salary.emp_id}(\text{Employee X Salary}))$$

The above query is rewritten using natural join as follows

$$\Pi_{emp_name,\ salry}(\text{Employee} \bowtie \text{Salary})$$

EMP_NAME	SALARY
BIKRAM SHAH	45000
SOURAV KUMAR	43000
RASHEED ALI	45000
AJIT KUMAR	44000

i) Division Operation:

The division operation is denoted by **'÷'.** It is suited to queries that include the phrase for all.

Table Name: **ACCOUNT**

ACCT_NO	B_NAME	BALANCE
A-101	AVADI	25000
A-102	AMBATTUR	22000
A-203	GANDHI ROAD	15000
A-208	NEHRU PARK	42000
A-107	MAHINDRA CITY	53000
A-105	MARKET ROAD	15000
A-109	MARKET ROAD	26000

Table Name: **CUSTOMER**

ACCT_NO	CUST_NAME
A-101	SAI KRISHNA
A-102	CHALAMAIAH
A-203	SHATADRU
A-207	SANJAY DAS
A-107	MANISH KUMAR
A-105	BEJAWADA RAJA

Table Name: **BRANCH**

B_NAME	B_CITY	ASSETS
AVADI	CHENNAI	71000
AMBATTUR	CHENNAI	25000
GANDHI ROAD	KANCHIPURAM	34500
NEHRU PARK	THIRUVALLUR	27800
MAHINDRA CITY	CHENGLEPET	64000
MARKET	VELLORE	29500

Step:1 We can obtain all branches in Chennai by the expression

$R1 = \Pi_{b_name}(\sigma_{b_city = chennai}$ (Branch))

B_NAME
AVADI
AMBATTUR

Step: 2 We can find all (Cust_name, B_name) pairs for which the customer has an account at a branch by as follows

$R2 = \Pi_{cust_name,\ b_name}$(Customer ⋈ Account)

CUST_NAME	B_NAME
SAI KRISHNA	AVADI
CHALAMAIAH	AMBATTUR
SHATADRU	GANDHI ROAD
MANISH KUMAR	MAHINDRA CITY
BEJAWADA RAJA	MARKET ROAD

Step: 3 Now we need to find customers who appear in R2 with every branch name in R1.

$$\Pi_{cust_name,\ b_name}(\text{Customer} \bowtie \text{Account}) \div \Pi_{b_name}(\sigma_{b_city = chennai}\ (\text{Branch}))$$

(or)

$$R2 \div R1$$

CUST_NAME
SAI KRISHNA
CHALAMAIAH

j) <u>Aggregate Functions</u>:

Aggregation function takes a collection of values and returns a single value as a result.

- **avg**: Average value
- **min**: Minimum value
- **max**: Maximum value
- **sum**: Sum of values
- **count**: Number of values

Table Name: **EMPLOYEE**

EMP_ID	EMP_NAME	DEPARTMENT	SALARY
1500101	SHARATH KUMAR	ACCOUNTS	45000
1500103	MANTHRI KARTHIK	SALES	43000
1500104	PALLI SWATHI	SALES	45000
1500105	DHANUSH KUMAR	IT	44000
1500108	HARINATH	IT	36000

Example:1 Calculate Total Salary paid to Employee

$G_{sum(salary)}$**(Employee)**

G – Signifies that aggregate function is to be applied, and its subscript specifies the aggregate operation to be applied.

SUM(SALARY)
213000

Example: 2 Display average salary of employees

$G_{avg(salary)}$**(Employee)**

AVG(SALARY)
42600

Example: 3 Count number of departments in Employee relation

$G_{count\text{-}distinct(department)}$**(Employee)**

DEPARTMENT
3

Example: 4 Display total salary of employees as department wise

Department$G_{sum(salary)}$(Employee)

DEPARTMENT	SUM(SALARY)
ACCOUNTS	45000
SALES	88000
IT	80000

k) <u>Outer Join</u>:

- The outer join operation is an extension of the **JOIN** operation to deal with missing information.

Table Name: **EMPLOYEE**

EMP_NAME	CITY
VIJAYA BHASKAR	PUNE
THUNGA PRATAP	MUMBAI
MARUTHI KUMAR	NASHIK
ROSHAN KUMAR	SOLAPUR

Table Name: **SALARY**

EMP_NAME	DEPARTMENT	SALARY
VIJAYA BHASKAR	PLANNING	65000
THUNGA PRATAP	ANALYST	72000
MANOHARA CHARY	PLANNING	64000
ROSHAN KUMAR	ANALYST	65000

EMPLOYEE⋈ SALARY

EMP_NAME	CITY	DEPARTMENT	SALARY
VIJAYA BHASKAR	PUNE	PLANNING	65000
THUNGA PRATAP	MUMBAI	ANALYST	72000
ROSHAN KUMAR	SOLAPUR	ANALYST	65000

- ✓ ***Maruthi Kumar Data is lost***
- ✓ ***Manohara Chary Data is lost.***

To avoid this loss of information, we can use outer join

Types of outer join:

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

1. **<u>Left Outer Join</u>**:
   - Denoted by ⟕
   - It takes ***all tuples in the left relation*** that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation and adds them to the result of the natural join.

EMPLOYEE ⟕ SALARY

EMP_NAME	CITY	DEPARTMENT	SALARY
VIJAYA BHASKAR	PUNE	PLANNING	65000
THUNGA PRATAP	MUMBAI	ANALYST	72000
MARUTHI KUMAR	NASHIK		
ROSHAN KUMAR	SOLAPUR	ANALYST	65000

2. **<u>Right Outer Join</u>:**
   - The Right outer join is denoted by ⟖
   - It pads ***tuples from right relation*** that did not match with the tuple from the left relation with nulls and adds them to the result of the natural join.

EMPLOYEE ⟖ SALARY

EMP_NAME	CITY	DEPARTMENT	SALARY
VIJAYA BHASKAR	PUNE	PLANNING	65000
THUNGA PRATAP	MUMBAI	ANALYST	72000
MANOHARA CHARY		PLANNING	64000
ROSHAN KUMAR	SOLAPUR	ANALYST	65000

3. **<u>Full Outer Join</u>:**
   - Full outer join denoted by⟗
   - It pads ***tuples from left relation*** that did not match with right relation and as well as ***tuples from right relation*** that did not match with any tuple from left relation and adds them to the resultant relation.

EMPLOYEE ⋈ SALARY

EMP_NAME	CITY	DEPARTMENT	SALARY
VIJAYA BHASKAR	PUNE	PLANNING	65000
THUNGA PRATAP	MUMBAI	ANALYST	72000
MARUTHI KUMAR	NASHIK		
ROSHAN KUMAR	SOLAPUR	ANALYST	65000
MANOHARA CHARY		PLANNING	64000

RELATIONAL CALCULAS:

- Relational Calculus is a higher-level ***Declarative language***.
- Relational calculus defines ***what result is to be obtained***.
- Relational Calculus ***does not specify the sequence of operations*** in which query will be evaluated.

Relational Calculus exists in two forms:

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

(a). Tuple Relational Calculus (TRC)

- The Tuple Relational Calculus **list the tuples** to be selected from a relation, based on a certain **condition** provided.
- It is formally denoted as:
- ***{ t | P(t) }***
- Where **t** is the set of tuples from which the condition **P** is true.

(b). Domain Relational Calculus (DRC)

- Domain Relational Calculus **list the attributes** to be selected from a relation, based on certain **condition**.
- The formal definition of Domain Relational Calculus is as follow:

  {<X1, X2, X3, . . .Xn> | P(X1, X2, X3, . . . Xn)}

  Where **X1, X2, X3, . . .Xn** are the attributes and **P** is the certain condition.

Difference between Relational Algebra and Relational Calculus

BASIS	RELATIONAL ALGEBRA	RELATIONAL CALCULUS
Basic	Relational Algebra is a Procedural language.	Relational Calculus is Declarative language.
State	Relational Algebra states how to obtain the result.	Relational Calculus states what result we have to obtain.
Order	Relational Algebra describes the order in which operations have to be performed.	Relational Calculus does not specify the order of operations.
Domain	Relational Algebra is not domain dependent.	Relation Calculus can be domain dependent.

STRUCTURE QUERY LANGUAGE (SQL):

SQL (Structured Query Language) is a computer language aimed to store, manipulate, and query data stored in relational databases.

TYPES OF SQL COMMANDS

SQL is used to control all of the functions that a DBMS provides for its users, including:

- ***Data definition***, SQL lets a user define the structure and organization of the stored data and relationship among the stored data items.
- ***Data retrieval***, SQL allows a user or an application program to retrieve stored data from the database and use it.
- ***Data manipulation***, SQL allows a user or an application program to update the database by adding new data, removing old data, and modifying previously stored data.
- ***Access control***, SQL can be used to restrict a user's ability to retrieve, add, and modify data, protecting stored data against unauthorized access.
- ***Data sharing***, SQL is used to coordinate data sharing by concurrent users, ensuring that they do not interface with one another.
- ***Data integrity***, SQL defines integrity constraints in the database, protecting it from corruption due to inconsistent updates or system failures.

DATA DEFINITION LANGUAGES (DDL)

A Data Definition Language (DDL) statement are used to define the database structure or schema.

DDL commands:

- ✓ CREATE
- ✓ ALTER
- ✓ DROP
- ✓ RENAME
- ✓ TRUNCATE

Create Table:

To make a new database, table, index, or stored query. A create statement in SQL creates an object inside of a relational database management system (RDBMS).

Syntax:

```
CREATE TABLE table_name
(
        Column_name1 data_type,
        Column_name2 data_type …
        Column_name_ndata_type
);
```

Example:

```
create table employee
(
empid number(10) primary key,
empname varchar2(20) not null,
dob date,
doj date,
dept char(10),
salary number(7)
);
```

Data Types :-

1. **String Data -** char, varchar, varchar2
2. **Numeric Data -** number, integer, float
3. **Temporal Data** – date, time, date & time combination
4. **Large Objects -** These are used for storing data objects like files and images:

   There are two types:

   - Character Large Objects (clobs)
   - Binary Large Objects (blobs)

Alter a Table:

To modify an existing database object. Alter the structure of the database.

a. **To add a column in a table**

   Syntax: ALTER TABLE table_name ADD column_name datatype;

   Example: SQL> alter table employee **add** exp number(2);

b. **To delete a column in a table**

   Syntax: ALTER TABLE table_name DROP column column_nmae;

   Example: SQL> alter table employee **drop column** exp;

c. **To change the data type of a column in a table**

   Syntax: ALTER TABLE table_name MODIFY column_name datatype;

   Example: SQL> alter table employee **modify** empname varchar2(30);

d. **To rename a column in a table**

   Syntax: ALTER TABLE table_name RENAME COLUMN old_column TO new_column;

   Example: SQL> alter table employee **rename column** empname to ename;

   Table altered.

Rename Table:

Purpose: Rename an object

Syntax: RENAME old_table_name to new_table_name;

Example: rename employee to emp;

Describe Table:

Purpose: Describes the structure of the table

Syntax: DESC table_name;

Example: SQL> desc employee;

Name	Null?	Type
EMPID	NOT NULL	NUMBER(10)
ENAME	NOT NULL	VARCHAR2(20)
DOB		DATE
DOJ		DATE
DEPT		CHAR(10)
SALARY		NUMBER(7)

Drop Table:

Delete Objects from the Database

Syntax: DROP TABLE table_name;

Example: drop table employee;

Truncate Table:

Remove all records from a table, including all spaces allocated for the records are removed.

Syntax: TRUNCATE TABLE table_name;

Example: **truncate** table employee;

DATA MANIPULATION LANGUAGE (DML)

- Data manipulation language allows the users to query and manipulate data in existing schema in object.
- It allows following data to ***insert, delete, update and recovery*** data in schema object.

DML COMMANDS:

1. INSERT
2. UPDATE
3. DELETE
4. SELECT

QUERY:

- Query is a statement in the DML that request the ***retrieval of data from database***.
- The portion of the DML used in a Query is called Query language.
- The SELECT statement is used to query a database

INSERT Query:

- Values can be inserted into table using insert commands.
- There are two types of insert commands.

  (i). Multiple row inserts commands (using '&' symbol)

  (ii). Single row insert command (without using '&'symbol)

1) **To insert one row at a time.**

   Syntax: INSERT INTO table_name VALUES (value1, value2, value3,...);

   Ex: insert into employee values (10144, 'Kavitha', '12-APR-2000', '25-APR-2019', 'Develop', 61000);

2) **To insert many rows at a time**

   Syntax: INSERT INTO table_name values (&column1, &column2,….);

   Ex: insert into employee values (&empid, '&empname', '&dob', '&doj', '&dept', '&salary');

3) **Inserting Data's in specified columns:**

   Syntax: INSERT INTO table_name (col1, col2, ..,col-n) VALUES (val1, val2 ,…,val n);

   Example: insert into employee (empid, empname) values (10110, 'Sambasiva');

SELECT Query:

Retrieve (read) data from a database.

1. To select particular column(s) from a table.
2. To select all records from a table
3. To select particular record using WHERE clause.

1) **To select *particular column*(s) from a table.**

Syntax: SELECT column_name(s) FROM table_name;

Example: SQL> select empid, empname from employee;

```
     EMPID EMPNAME
---------- --------------------
     10159 Hima Shekar
     10166 Sriram
     10193 Harini
     10101 Harinath
     10144 Kavitha
     10240 Balakrishna
     10109 Krishna
     10167 Manideep
     10171 SivaSai
     10207 Bharathi
     10607 Harinath
     10236 Balachandar
     10244 Bhumesh
     10225 Sayadeevi
     10208 Arunkumar
     10216 Tharun T
     10235 Anilkumar
17 rows selected.
```

2) **To select *all records* from a table**

Syntax: SELECT * FROM table_name;

Example: SQL> select * from employee;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10159	Hima Shekar	12-NOV-99	30-MAR-15	Plan	65000
10166	Sriram	23-FEB-99	04-APR-18	Develop	60000
10193	Harini	10-JUL-00	31-DEC-16	HR	55000
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10144	Kavitha	12-APR-00	25-APR-19	Develop	61000
10240	Balakrishna	08-JUN-01	16-MAY-17	Analyst	65000
10109	Krishna	28-MAY-00	12-APR-16	Sales	60000

10167	Manideep	12-APR-98	23-MAR-19	Develop	65000
10171	SivaSai	30-NOV-98	05-APR-19	HR	72000
10207	Bharathi	20-MAR-00	31-JAN-18	HR	67000
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600
10236	Balachandar	29-JUN-98	31-DEC-19	Analyst	56000
10244	Bhumesh	20-MAR-98	30-NOV-17	Develop	60000
10225	Sayadeevi	27-JUL-00	15-JAN-19	HR	65000
10208	Arunkumar	03-JAN-99	20-SEP-18	Develop	67000
10216	Tharun T	28-MAY-01	14-JAN-15	Develop	65000
10235	Anilkumar	22-JUL-01	18-DEC-18	Plan	56000

17 rows

3) **To select *particular record* using WHERE clause**.

Syntax: SELECT * FROM table_name WHERE column = value;

Example: SQL> select * from employee where **empid=10167**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
---	---	---	---	---	---
10167	Manideep	12-APR-98	23-MAR-19	Develop	65000

4) **Select using Distinct:**

- It is helpful when there is need of ***avoiding the duplicate values*** present in any specific columns/table.
- **Syntax:** SELECT DISTIN T column name(s) FROM table_name;
- **Example:** SQL> select **distinct dept** from employee;

  DEPT
  Develop
  HR
  Analyst
  Plan
  Sales

UPDATE Query:

- This command is used for updating or modifying the values of columns in a table (relation).

Syntax: UPDATE table_name SET column_name = new_value [WHERE condition];

Example: update employee set empid = ‘10159’ where empname = ‘hima sekar’;

DELETE Query:

- This command is used for removing one or more records from a table (relation).

Syntax: DELETE FROM table_name [WHERE condition];

Example: delete from employee where empid = '10091';

DATA CONTROL LANGUAGE (DCL)

- DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.
  - ✓ CREATE – Create user area in the database.
  - ✓ GRANT - Gives user's access privileges (Rights) to database.
  - ✓ REVOKE - Withdraw user's access privileges given by using the GRANT command.

Create User:

- **Syntax:** CREATE USER user_name IDENTIFIED BY password;
- **Example:** SQL> create user biher identified by cse123;

  User created.

Grant Command:

- **Syntax:** GRANT ALL PRIVILEGES TO User_name;
- **Example:** SQL> grant all privileges to biher;

  Grant succeeded.

  SQL> connect

  Enter user-name: biher

  Enter password:

  Connected.

Revoke Command:

- **Syntax:** REVOKE ALL PRIVILEGES FROM User_Name;
- **Example:** SQL> revoke all privileges from biher;

  Revoke succeeded.

  SQL> connect

Enter user-name: biher

Enter password:

ERROR:

ORA-01045: user BIHER lacks CREATE SESSION privilege; logon denied

Warning: You are no longer connected to ORACLE.

TRANSACTION CONTROL LANGUAGE (TCL): -

- Transaction Control Language (TCL) commands are used to ***manage transactions*** in the database.
- These are used to manage the changes made to the data in a table by DML statements.
- **TCL commands**:
  - ✓ **COMMIT**– Commits a Transaction.
  - ✓ **ROLLBACK**– Rollbacks a transaction in case of any error occurs.
  - ✓ **SAVEPOINT** – Sets a savepoint within a transaction.

Commit:

- COMMIT command is used to ***permanently save any transaction*** into the database.
- When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.
- To avoid that, we use the COMMIT command to mark the changes as permanent.
- **Syntax:** COMMIT;

Savepoint:

- SAVEPOINT command is used to temporarily save a transaction so that you can roll back to that point whenever required.
- **Syntax:** SAVEPOINT Savepoint_Name;

Rollback:

- Restores the database to last committed state.
- It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.
- **Syntax:** ROLLBACK;

(or)

ROLLBACK TO Savepoint_Name;

CONSTRAINS:

It is a mechanism used to present invalid data entry into the table.

Types:

(i) Domain integrity constraints.

(ii) Entity integrity constraints

(iii) Referential integrity constraints

(i). Domain Integrity Constraints:

- Each table has certain ***set of columns*** and each column ***allows a same type of data***, based on its data type.
- The column ***does not accept values of any other data type***.

a) **NOT NULL constraint**:

- By default, a column can hold **NULL**.
- If you do not want to allow **NULL** value in a column, you can place the **NOT NULL** constraint on this column.
- The **NOT NULL** constraint specifies that **NULL** is not an allowable value.
- User has to provide a value for the column.
- **Example**:

```
CREATE TABLE student
(
        Regno       Number(10)   not null,
        Stuname     Varchar(25)  not null,
        Age         Number(3),
        Dept        Char(10)
);
```

b) **CHECK constraints**:

- The **CHECK** constraint ensures that ***all values in a column satisfy certain conditions***.
- Once defined, the database will allow the new value, if the new value satisfies the **CHECK** constraint.
- The **CHECK** constraint is used to ensure data quality.
- **Example**:

```
create table boys_hostel
(
        Hostel_id     Number(10)    not null,
        Regno         Number(10)    not null,
        Stuname       Varchar(25)   not null,
        Gender        Char(8)       CHECK(gender='male')
);
```

(ii). Entity Integrity Constraints:

- Entity integrity is concerned with ensuring that each row of a table has a unique and non-null value.
- There are two types of entity integrity constraints,
  1. **UNIQUE -** Unique constraints is used to prevent duplication of values and null values.
  2. **PRIMARY KEY -** This is used for to prevent duplication of values and null values.

a) UNIQUE Constraints:

Unique constraint is used to prevent duplication of values. It does not allow null value for the column.

Example:

```
create table boys_hostel
(       Hostel_id     Number(10)    unique,
        Regno         Number(10)    not null,
        Stuname       Varchar(25)   not null,
        Gender        Char(8)       check(gender='male')
);
```

b) PRIMARY KEY Constraint:

This is used for to prevent duplication of values .It will not allow null values for the column.

Example:

```
create table boys_hostel
(
        Hostel_id     Number(10)    primary key,
        Regno         Number(10)    not null,
        Stuname       Varchar(25)   not null,
        Gender        Char(8) check(gender='male')
);
```

(iii). Referential Integrity Constraint:

- To establish a ***parent child relationship*** between two tables having a common column we can use referential integrity constraints.
- To implement this, we should define the column in the ***parent table as primary key*** and the ***same column in the child table as a foreign key*** referring to the corresponding parent entry.

Example:

<u>Parent Table</u>:

```
create table student
(
        Regno         Number(10)    primary key,
        Stuname       Varchar(25)   not null,
        Dept          Char(10),
        Age           Number(3),
        Gender        Char
);
```

<u>Child table</u>:

```
create table boys_hostel
(       Hostel_id     Number(10)    primary key,
        Regno         Number(10)    references student (Regno),
        Stuname       Varchar(25)   not null,
        Gender        Char(8)       check(gender='male'));
```

(or**)**

```
create table boys_hostel
(
        Hostel_id     Number(10)    primary key,
        Vtuno         Number(10),
        Stuname       Varchar(25)   not null,
        Gender        Char(8) check(gender='male')
        foreign key(Regno) references student (Regno),
);
```

VIEWS:

- A View is a ***virtual table*** that does not necessarily exist in the database but can be produced upon request by a particular user, at the time of request.
- If data is changing in the original table, the same change is reflected in the view.
- A view can be built on top of a ***single table or multiple tables***.
- It can also be ***built on top of another view***.

Advantages of Views:

1. **Ease of use**: A view hides the complexity of the database tables from end users.
2. **Space savings**: Views takes very little space to store, since they do not store actual data.
3. **Additional data security**: Views can include only certain columns in the table so that only the non-sensitive columns are included and exposed to the end-user.

a) Creating a View:

Syntax: CREATE VIEW view_name AS SELECT column_1, ….column_n FROM table_name;

Examples:

1. create view emp_view as select empid, empname, dept from employee;
2. create view emp_view as select * from employee;
3. create view emp_view as select e.empname, e.dept, *b.branch* from employee e, *branch b* where e.empid=b.empid;

b) Removing a View:

Syntax: DROP VIEW View_Name;

c) Updatable View:

For a view to be updatable, the DBMS must be able to trace any row or column back to its row or column in the source table.

Syntax: UPDATE view_table_name SET column="value" WHERE <CONDITION>;

Examples: SQL>update emp_view set empname='Kumara Swamy' where empid= 10190;

d) Select View:

select * from emp_view;

e) Insert Data into View:

SQL> insert into emp_view values (&empid, '&name', '& dept');

INDEX TABLE:

- I*ndexes* help us ***retrieve data*** from tables ***quicker***.
- ***Without an index***, the database system ***reads through the entire table*** to locate the desired information.
- ***With index***, the database system ***first go through the index table*** to find out where to retrieve the data, and then go to these locations ***directly to get the needed data***. *This is much faster*.

Syntax: CREATE INDEX index_name ON table_name (column_name(s));

Example:

create index emp_index on employee(empid);

create index emp_index on employee (empname, dob);

JOIN QUERIES:

- Join is a query in which data is returned from *two or more tables*.
- The purpose of a join is to combine the data spread across tables.
- **Syntax:** Select column_1, column_2, …..column_n from table1, table2 [where <condition>];
- **Types of Joins**:
  1. Natural Join
  2. Inner Join
  3. Left Outer Join
  4. Right Outer Join
  5. Full Join

Table Name: **Branch**

BNAME	SALES	PRODUCT	TXN_DATE
AVADI	35000	P1	20-JUL-2018
GUINDY	21000	P2	20-JUL-2018
AVADI	15000	P	21-JUL-2018
AMBATTURE	20000	P1	20-JUL-2018
T-NAGAR	32000	P1	21-JUL-2018
TAMBARAM	33000	P2	20-JUL-2018

Table Name**: Region**

Rname	Bname
WEST	AMBATTUR
WEST	T-NAGAR
EAST	AVADI
EAST	GUINDY
NORTH	ANNA NAGAR

1. Natural Join:

It returns the matching rows from the tables that are being joined.

Example: SQL> select b.bname from **branch b, region** where b.bname=region.bname;

BNAME

AVADI
GUINDY
AVADI
AMBATTUR
T-NAGAR

2. Inner Join:

- An inner join in SQL returns rows where there is *at least one match on both tables*.

Syntax: SQL> SELECT column_1, column_2,… column_n from table_1 INNER JOIN table_2 ON table_1.column=table_2.column;

Ex: SQL> select b.b_name, rname from branch b **inner join** region r on b.bname = r.bname;

BNAME	RNAME
AVADI	EAST
GUINDY	EAST
AVADI	EAST
AMBATTUR	WEST
T-NAGAR	WEST

3. Left Outer Join:

- In left outer join, all rows from the first table mentioned in the SQL query is selected, regardless whether there is a matching row on the second table mentioned in the SQL query.

Syntax: SQL> SELECT column_1, column_2, …., column_n FROM table_1 LEFT OUTER JOIN table_2 ON table_1.column=table_2.column;

Ex: SQL> select b.name, rname from branch b **left outer join** region r on b.bname=r.bname;

BNAME	REGNAME
AMBATTUR	WEST
T-NAGAR	WEST
AVADI	EAST
AVADI	EAST
GUINDY	EAST
TAMBARAM	

4. Right Outer Join:

- In right outer join, all rows from the second table mentioned in the SQL query is selected, regardless whether there is a matching row on the first table mentioned in the SQL query.

Syntax: SQL> SELECT column_1, column_2,…..column_n FROM table_1 RIGHT OUTER JOIN table_2 ON table_1.column=table_2.column;

Ex: SQL>select b.bname, rname from branch b **right outer join** region r on b.bname=r.bname;

BNAME	RNAME
AVADI	EAST
GUINDY	EAST
AVADI	EAST
AMBATTUR	WEST
T-NAGAR	WEST
	NORTH

5. Full Join:

- It is the combination of both left outer and right outer join.

Syntax: SQL> SELECT column_1, column_2,…..column_n FROM table_1 FULL JOIN table_2 ON table_1.column=table_2.column;

Ex: SQL> select b.bname, rname from branch b **full join** region r on b.bname=r.bname;

BNAME	RNAME
AMBATTUR	WEST
T-NAGAR	WEST
AVADI	EAST
AVADI	EAST
GUINDY	EAST
TAMBARAM	
	NORTH

<u>SUB QUERIES or NESTED SUB-QUERY</u>:

- Queries, one within the other is termed as a **sub query**.
- **Sub-queries** are used to retrieve data from tables that depend on the values in the table itself.
- Sub-queries can reside in the **WHERE** clause, the **FROM** clause, or the **SELECT** clause.

Table Name: **EMPLOYEE**

EMPID	EMPNAME	BRNO	DEPTNO	SALARY	EXP
1101	GOUSE BASHA	111	1001	25000	5
1102	BALAJEE	111	1001	15000	1
1103	BHARATHI	222	4001	25000	5
1104	KAMATCHI	222	4001	20000	3
1105	MURTHI	333	5001	15000	1
1106	KANNAN	333	5001	20000	3
1107	MALADHI	444	5001	20000	3
1108	ARUN KUMAR	444	3001	15000	1
1109	EDWARD	111	2001	20000	4
1110	SUMANTH	333	2001	25000	6

DEPARTMENT

DNO	DNAME
1001	IT
2001	ACCOUNTS
3001	SALES
4001	SERVICE
5001	PLANNING

BRANCH

BNO	BNAME
111	CHENNAI
222	BANGALURE
333	HYDERABAD
444	MUMBAI

(a). <u>Insert with Sub-Queries</u>:

- INSERT statement can be used with subqueries.

<u>Syntax</u>: INSERT INTO table_name [(column1 [, column2])] SELECT *|column1 [, column2] FROM table1 [, table2] [WHERE VALUE OPERATOR];

<u>Ex</u>: SQL>insert intochennai_branch select * from employee where brno = (select bno from branch where bname='chennai');

```
SQL> select * from chennai_branch;

     EMPID EMPNAME                    BRNO     DEPTNO     SALARY        EXP
---------- -------------------- ---------- ---------- ---------- ----------
      1101 GOUSE BASHA                 111        301      25000          5
      1102 BALAJEE                     111        301      15000          1
```

(b). Update with Sub-Queries:

- In an UPDATE statement, we can set new column value equal to the result returned by a single row subquery.

Syntax: UPDATE table_name SET column_name = new_value WHERE OPERATOR (SELECT column_name FROM table_name) [WHERE <Condition>];

Example: SQL>UPDATE employee SET salary = salary+2000 WHERE deptno = (SELECT dno FROM department WHERE dname='sales');

EMPID	EMPNAME	BRNO	DEPTNO	SALARY	EXP
1101	GOUSE BASHA	111	1001	25000	5
1102	BALAJEE	111	1001	15000	1
1103	BHARATHI	222	4001	25000	5
1104	KAMATCHI	222	4001	20000	3
1105	MURTHI	333	5001	15000	1
1106	KANNAN	333	5001	20000	3
1107	MALADHI	444	5001	25000	5
1108	ARUN KUMAR	444	3001	17000	1
1109	EDWARD	111	2001	20000	4
1110	SUNANTHA	333	2001	25000	6

(c). <u>Delete with Sub-Queries</u>

- DELETE statement can be used with subqueries.

Syntax: DELETE FROM table_name WHERE OPERATOR (SELECT Column_name FROM table_name) [WHERE) <condition>];

Example: SQL>delete from employee where deptno=(select dno from department where dname='it');

```
SQL> SELECT * FROM EMPLOYEE;

     EMPID EMPNAME                    BRNO     DEPTNO     SALARY        EXP
---------- -------------------- ---------- ---------- ---------- ----------
      1103 BHARATHI                    222       4001      25000          5
      1104 KAMATCHI                    222       4001      20000          3
      1105 MURTHI                      333       5001      15000          1
      1106 KANNAN                      333       5001      20000          3
      1107 MALADHI                     444       5001      25000          5
      1108 ARUN KUMAR                  444       3001      17000          1
      1109 EDWARD                      111       2001      20000          4
      1110 SUNANTHA                    333       2001      25000          6

8 rows selected.
```

(d). <u>SQL IN with Sub-Queries</u>

- The IN operator allows you to specify multiple values in a WHERE clause.

Syntax*:* SELECT column_name(s) FROM table_name WHERE column_name IN (SELECT Column_name FROM table_name WHERE condition);

Example: SQL> select empname, bname from employee join branch on brno=bno and brno in (select bno from branch where bnamein('chennai','mumbai‘));

```
EMPNAME                  BRNO BNAME                           BNO
-------------------- ---------- ------------------------------ ----------
MALADHI                     444 MUMBAI                         444
ARUN KUMAR                  444 MUMBAI                         444
EDWARD                      111 CHENNAI                        111
```

(e). Sql NOT IN with Sub-Queries:

- NOT IN ( ) makes sure that the expression proceeded does not have any of the values present in the arguments

<u>Syntax</u>: SELECT column_name(s) FROM table_name WHERE column_name NOT IN (SELECT Column_name FROM table_name WHERE condition);

Example: SQL> SELECT empname, brno, bname, bno FROMemployee JOIN branch ON brno=bno AND brno NOT IN (SELECT bno FROM branch WHERE bname IN ('chennai', 'mumbai'));

```
EMPNAME                      BRNO BNAME                          BNO
-------------------- ---------- ---------- ---------- ---------- ----------
BHARATHI                      222 BANGALURE                     222
KAMATCHI                      222 BANGALURE                     222
MURTHI                        333 HYDERABAD                     333
KANNAN                        333 HYDERABAD                     333
SUNANTHA                      333 HYDERABAD                     333
```

(f). <u>Sql EXIST with Sub-Queries</u>:

- The EXISTS operator is used to test for the existence of any record in a subquery. The EXISTS operator returns true if the subquery returns one or more records.

<u>Syntax</u>: SELECT column_name(s) FROM table_name WHERE EXISTS (SELECT column_name FROM table_name WHERE condition);

Example:

```
SQL> SELECT * FROM EMPLOYEE WHERE EXISTS(SELECT DNO FROM DEPARTMENT WHERE DNAME='SPARES');

no rows selected
```

```
SQL> SELECT * FROM EMPLOYEE WHERE EXISTS(SELECT DNO FROM DEPARTMENT WHERE DNAME='SALES');

     EMPID EMPNAME                    BRNO     DEPTNO     SALARY        EXP
---------- -------------------- ---------- ---------- ---------- ----------
      1103 BHARATHI                    222       4001      25000          5
      1104 KAMATCHI                    222       4001      20000          3
      1105 MURTHI                      333       5001      15000          1
      1106 KANNAN                      333       5001      20000          3
      1107 MALADHI                     444       5001      25000          5
      1108 ARUN KUMAR                  444       3001      17000          1
      1109 EDWARD                      111       2001      20000          4
      1110 SUNANTHA                    333       2001      25000          6

8 rows selected.
```

(g). NOT EXIST with Sub-Queries:

- The NOT condition can be combined with the EXISTS condition to create a NOT EXISTS condition.

Syntax: SELECT column_name(s) FROM table_name WHERE NOT EXISTS (SELECT Column_name FROM table_name WHERE condition);

Example:

```
SQL> SELECT * FROM EMPLOYEE WHERE NOT EXISTS(SELECT DNO FROM DEPARTMENT WHERE DNAME='SALES');

no rows selected
```

```
SQL> SELECT * FROM EMPLOYEE WHERE NOT EXISTS(SELECT DNO FROM DEPARTMENT WHERE DNAME='TAX');

     EMPID EMPNAME                   BRNO     DEPTNO     SALARY        EXP
---------- -------------------- ---------- ---------- ---------- ----------
      1103 BHARATHI                    222       4001      25000          5
      1104 KAMATCHI                    222       4001      20000          3
      1105 MURTHI                      333       5001      15000          1
      1106 KANNAN                      333       5001      20000          3
      1107 MALADHI                     444       5001      25000          5
      1108 ARUN KUMAR                  444       3001      17000          1
      1109 EDWARD                      111       2001      20000          4
      1110 SUNANTHA                    333       2001      25000          6

8 rows selected.
```

AGGREGATE FUNCTIONS:

i) Count(): To count the no of records, based on given attribute.

Syntax: SELECT COUNT(column_name) FROM table_name;

ii) max(): To identify the maximum value

Syntax: SELECT MAX(column_name) FROM table_name;

iii) min(): To identify the minimum value

Syntax: SELECT MIN(column_name) FROM table_name;

iv) sum(): To calculate summation of a particular column

Syntax: SELECT SUM(column_name) FROM table_name;

v) avg(): To calculate average of a particular column

Syntax: SELECT AVG(column_name) FROM table_name;

SET OPERATIONS

1. Union:

Purpose: Used to combine all values present in two relations but the duplicates are removed

Command: UNION

Syntax:

SELECT column_name1,….column_name_n FROM table_name_1 UNION select column_name1….column_name_n FROM table_name_2;

2. <u>Union All</u>:

Purpose: Used to combine all values present in two relations.

Command: UNION

Syntax:

SELECT column_name1,….column_name_nFROM table_name_1 UNION ALL SELECT column_name1….column_name_n FROM table_name_2;

3. <u>Intersect</u>:

Purpose: Combine the common values in two relation

Command : INTERSECT

Syntax:

SELECT column_name1,….column_name_n FROM table1_name_1
INTERSECT SELECTcolumn_name1….column_name_n FROM table_name_2;

4. <u>Minus</u>

Purpose: Includes the tuples from first_table but that should not in second_table

Command: MINUS

Syntax: SELECT column_name1,….column_name_n FROM table1_name_1
MINUS SELECTcolumn_name1….column_name_n FROM table_name_2;

NORMALIZATION:

- Normalization is the ***process of organizing the data*** in the database.
- Normalization is used to ***minimize the redundancy*** from a relation or set of relations. It is also used to ***eliminate*** the undesirable characteristics like ***Insertion, Update and Deletion Anomalies***.
- Normalization divides the ***larger table into the smaller*** table and links them using relationship.
- The ***normal form*** is used to reduce redundancy from the database table.

ANOMALIES IN DBMS

- There are ***three types of anomalies*** that occur when the database is not normalized.
- These are – Insertion, update and deletion anomaly.
- **Example**: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: ***emp_id*** for storing employee's id, ***emp_name*** for storing employee's name, ***emp_address*** for storing employee's address and ***emp_dept*** for storing the department details in which the employee works. At some point of time the table looks like this:

EMP_ID	EMP_NME	EMP_ADDRESS	EMP_DEPT
10196	RAKESH	CHENNAI	D0011
10196	RAKESH	CHENNAI	D002
10198	NITYANAND	HYDERABAD	D890
10203	PAVAN KUMAR	GUNTUR	D900
10203	PAVAN KUMAR	GUNTUR	D004

- The above table is ***not normalized***.

1) **Update anomaly**:
   - In the above employee table, we have ***two rows for employee Rakesh*** as he belongs to two departments of the company.
   - If we ***want to update the address of Rakesh*** then we ***have to update the same in two rows*** or the data will become inconsistent.

2) **Insert anomaly**:
   - Suppose a ***new employee joins the company***, who is under training and currently ***not assigned to any department*** then we would ***not be able to insert the data*** into the table if emp_dept field doesn't allow nulls.

3) **Delete anomaly**:

- ❖ Suppose, if at a point of time the ***company closes the department D890*** then deleting the rows that are having emp_dept as D890 would ***also delete the information of employee Nityanand*** since she is assigned only to this department.
- ❖ To overcome these anomalies, we need to normalize the data.

Functional Dependency in DBMS:

- "The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table".
- **For example:** Suppose we have a ***student table*** with attributes: ***Stu_Id***, ***Stu_Name***, ***Stu_Age***. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as **functional dependency** and can be written as ***Stu_Id->Stu_Name*** or in words we can say ***Stu_Name is functionally dependent on Stu_Id.***

Types of Functional Dependencies

1. Trivial functional dependency
2. non-trivial functional dependency
3. Multivalued dependency
4. Transitive dependency

1) Trivial Functional Dependency:

- The dependency of an attribute on a set of attributes is known as ***trivial functional dependency*** if the set of attributes includes that attribute.

Example: Consider a table with two columns **Student_Id** and **Student_Name**.

- **{Student_Id, Student_Name} -> Student_Id** is a trivial functional dependency as **Student_Id is a subset of {Student_Id, Student_Name}**.
- Also, **Student_Id -> Student_Id** & **Student_Name -> Student_Name** are trivial dependencies too.

2) Non-Trivial Functional Dependency:

- If a functional dependency X->Y holds true where Y is not a subset of X then this dependency is called non trivial Functional dependency.

For example:

- An employee table with three attributes: **emp_id, emp_name,** and **emp_address**.
- The following functional dependencies ***are non-trivial***:
  - ✓ emp_id -> emp_name (emp_name is not a subset of emp_id)
  - ✓ emp_id -> emp_address (emp_address is not a subset of emp_id)
- **On the other hand, the following dependencies are trivial:**
  - ✓ {emp_id, emp_name} -> emp_name [emp_name is a subset of {emp_id, emp_name}]
- **Completely non trivial FD**:
  - ✓ If a FD X->Y holds true where X intersection Y is null then this dependency is said to be completely non trivial function dependency.

3) Multivalued dependency:

- Multivalued dependency occurs when there are more than one **independent** multivalued attributes in a table.
- **For example**: Consider a bike manufacture company, which produces two colors (Black and Red) in each model every year.
- Here columns **manuf_year and color are independent** of each other and **dependent on bike_model**.
- In this case these two columns are said to be multivalued dependent on bike_model. These dependencies can be represented like this:
  - ✓ **bike_model ->> manuf_year**
  - ✓ **bike_model ->> color**

BIKE_MODEL	MANUF_YEAR	COLOR
M1001	2007	Black
M1001	2007	Red
M2012	2008	Black
M2012	2008	Red
M2222	2009	Black
M2222	2009	Red

4) Transitive Dependency:

- A functional dependency is said to be transitive if it is ***indirectly formed by two functional dependencies***. For e.g.
- X -> Z is a transitive dependency if the following three functional dependencies hold true:
  - ✓ X->Y
  - ✓ Y does not ->X
  - ✓ Y->Z
- **Note:** A transitive dependency can only occur in a relation of three of more attributes. This dependency helps us normalizing the database in 3NF (3rd Normal Form).

Example:

BOOK	AUTHOR	AUT_AGE
Game of Thrones	George R.R Martin	66
Harry Potter	J. K. Rowling	49
Dying of the Light	George R.R Martin	66

- ✓ {Book} ->{Author} (if we know the book, we know the author name)
- ✓ {Author} does not ->{Book}
- ✓ {Author} -> {Aut_age}

- Therefore, as per the rule of **transitive dependency**: {Book} -> {Aut_age} should hold, that makes sense because if we know the book name we can know the author's age.

Normalization Types:

Here are the most commonly used normal forms:

1) First normal form(1NF)
2) Second normal form(2NF)
3) Third normal form(3NF)
4) Boyce & Codd normal form (BCNF)

First Normal Form (1NF)

- As per the rule of first normal form, an attribute (column) of a table ***cannot hold multiple values.*** It should hold ***only atomic values***.
- **Example:** Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_MOBILE
10211	Ramesh	Delhi	8912312390
			8812121212
10098	Bharani	Kanpur	9900012222
10127	Abhiram	Chennai	7778881212
			9990000123
10093	Harini	Bangalore	8123450987

- Two employees (**Ramesh & Abhiram) are having two mobile numbers** so the company stored them in the same field as you can see in the table above.
- This **table is not in 1NF** as the rule says "***each attribute of a table must have atomic*** (single) ***values***", the emp_mobile values for employees **Ramesh & Abhiram** violates that rule.
- To make the table complies with 1NF we should have the data like this:

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_MOBILE
10211	Ramesh	Delhi	8912312390
10211	Ramesh	Delhi	8812121212
10098	Bharani	Kanpur	9900012222
10127	Abhiram	Chennai	7778881212

10127	Abhiram	Chennai	9990000123
10093	Harini	Bangalore	8123450987

Second Normal Form (2NF)

- A table is said to be in 2NF if both the following conditions hold:
  1) Table is in 1NF (First normal form)
  2) No non_key attribute is dependent on the any candidate key of table.
- **Example:** Suppose a school wants to store the data of ***teachers*** and ***the subjects*** they teach. They create a table that looks like this:
- Since ***a teacher can teach more than one subjects***, the table can have multiple rows for a same teacher.

Faculty_ID	Subject	Faculty_Age
111	DBMS	38
111	CMC	38
222	PCD	34
222	TOC	34
333	OS	36
333	DSC	36

Candidate Keys: {Faculty_ID, Subject}

Non_key attribute: Faculty_Age

- The ***table is in 1 NF*** because each attribute has atomic values.
- However, ***it is not in 2NF*** because ***non_key attribute Faculty_Age is dependent on Faculty_ID alone*** which is a proper subset of candidate key.
- This violates the rule for 2NF.
- To make the table complies with 2NF ***we can break it in two tables*** like this: **Faculty_Details** table and **Faculty_Subject** table**:**

Faculty_ID	Faculty_Age
111	38
222	34

333	36

Faculty_ID	Subject
111	DBMS
111	CMC
222	PCD
222	TOC
333	OS
333	DSC

Note: Now the tables are in Second normal form (2NF).

Third Normal Form (3NF)

- A table design is said to be in 3NF if both the following conditions hold:
  1. Table must be in 2NF
  2. Transitive functional dependency of non-prime attribute on any super key should be removed.
- An attribute that is not part of any candidate key is known as non-prime attribute.
- In other words, 3NF can be explained like this:

Example: Suppose a company wants to store the complete address of each employee, they create a table named **employee_details** that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
10118	Tharuni	504001	TS	Adilabad	Adilabad
10128	Vinay	600045	TN	Thambaram	Chennai
10146	Kiran Sai	533001	AP	Kakinda	East Godhavari
10248	Vishnuvardhan	523001	AP	Ongole	Prakasham
10235	Anil Kumar	521001	AP	Machilipatnam	Krishna

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip} and so on. **Candidate Keys**: {emp_id}

Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
10118	Tharuni	504001
10128	Vinay	600045
10146	Kiran Sai	533001
10248	Vishnuvardhan	523001
10235	Anil Kumar	521001

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
504001	TS	Adilabad	Adilabad
600045	TN	Thambaram	Chennai
533001	AP	Kakinda	East Godhavari
523001	AP	Ongole	Prakasham
521001	AP	Machilipatnam	Krishna

Boyce Codd normal form (BCNF)

- It is an ***advance version of 3NF*** that's why it is also referred as 3.5NF.
- BCNF is stricter than 3NF.
- A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in more than one department. They store the data like this:

emp_id	emp_city	emp_dept	dept_no	dept_no_of_emp
1001	Chennai	Production and planning	D001	200
1001	Chennai	stores	D001	250
1002	Hyderabad	design and technical support	D134	100
1002	Hyderabad	Purchasing department	D134	600

- **Functional dependencies in the table above:**
  - ✓ emp_id -> emp_nationality
  - ✓ emp_dept -> {dept_no, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:

emp_city:

emp_id	emp_city
1001	Chennai
1002	Hyderabad

emp_dept:

emp_dept	dept_no	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:

- ✓ emp_id -> emp_nationality
- ✓ emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:

- ✓ For first table: emp_id
- ✓ For second table: emp_dept
- ✓ For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

***********BEYOND THE SYLLABUS********

SQL – WHERE Clause: -

- WHERE keyword is used for ***fetching (Read) filtered*** data in a result set.
- It is used to fetch data according to a ***particular criterion***.
- **Syntax:**

  SELECT column1, column2 FROM table_name WHERE column_name OPERATOR value;

List of operators that can be used with where clause:

OPERATOR	DESCRIPTION
>	Greater Than
>=	Greater than or Equal to
<	Less Than
<=	Less than or Equal to
=	Equal to
<>	Not Equal to
BETWEEN	In an inclusive Range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Ex: 1

SQL> select * from employee where **empid=10167**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10167	Manideep	12-APR-98	23-MAR-19	Develop	65000

Ex: 2

SQL> select * from employee where **dob>='01-Jan-2000'**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10193	Harini	10-JUL-00	31-DEC-16	HR	55000
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10144	Kavitha	12-APR-00	25-APR-19	Develop	61000
10240	Balakrishna	08-JUN-01	16-MAY-17	Analyst	65000
10109	Krishna	28-MAY-00	12-APR-16	Sales	60000

10207	Bharathi	20-MAR-00	31-JAN-18	HR	67000
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600
10225	Sayadeevi	27-JUL-00	15-JAN-19	HR	65000
10216	Tharun T	28-MAY-01	14-JAN-15	Develop	65000
10235	Anilkumar	22-JUL-01	18-DEC-18	Plan	56000

10 rows selected.

WHERE clause with - *AND* Operator: -

- The WHERE clause when used together with the AND logical operator, is only executed if ALL filter criteria specified are met.
- **Syntax:** SELECT * FROM table_name WHERE condition1 AND condition2;
- **Example:** SQL> select * from employee where **empname = 'Harinath' and dept = 'Sales'**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
---	---	---	---	---	---
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600

WHERE Clause - BETWEEN Operator:-

- BETWEEN can be used to get those items that fall within a range.
- **Syntax:** SELECT column1 [,columns] FROM table_name WHERE column_name BETWEEN value1 AND value2;
- **Ex:** SQL> select * from employee where **salary between 50000 and 60000**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
---	---	---	---	---	---
10166	Sriram	23-FEB-99	04-APR-18	Develop	60000
10193	Harini	10-JUL-00	31-DEC-16	HR	55000
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10109	Krishna	28-MAY-00	12-APR-16	Sales	60000
10236	Balachandar	29-JUN-98	31-DEC-19	Analyst	56000
10244	Bhumesh	20-MAR-98	30-NOV-17	Develop	60000
10235	Anilkumar	22-JUL-01	18-DEC-18	Plan	56000

7 rows selected.

WHERE Clause - LIKE Operator: -

- LIKE clause compares data with an expression using *wildcard operators* to match pattern given in the condition.
- Wildcard Operators: There are 2 wildcard operators that are used in LIKE clause.
  - ✓ Percent sign" **%**": Represents zero, one or more than one character.

- ✓ Underscore sign" _": Represents only a single character.

Example:

SQL> select * from employee where **empname like '_a%'**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10193	Harini	10-JUL-00	31-DEC-16	HR	55000
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10144	Kavitha	12-APR-00	25-APR-19	Develop	61000
10240	Balakrishna	08-JUN-01	16-MAY-17	Analyst	65000
10167	Manideep	12-APR-98	23-MAR-19	Develop	65000
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600
10236	Balachandar	29-JUN-98	31-DEC-19	Analyst	56000
10225	Sayadeevi	27-JUL-00	15-JAN-19	HR	65000

8 rows selected.

SQL> select * from employee where **salary like '_7%'**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10207	Bharathi	20-MAR-00	31-JAN-18	HR	67000
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600
10208	Arunkumar	03-JAN-99	20-SEP-18	Develop	67000

<u>SQL – ORDER BY Clause</u>: -

- Order by clause is used with SELECT statement for **arranging retrieved data in sorted order**.
- The Order by clause **by default sorts the retrieved data in ascending order**. To sort the data in descending order DESC keyword is used with Order by clause.
- **Syntax:** SELECT column-list|* FROM table-name ORDER BY ASC | DESC;
- **Example:** SQL> select * from employee **order by empid**;

EMPID	**EMPNAME**	**DOB**	**DOJ**	**DEPT**	**SALARY**
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10109	Krishna	28-MAY-00	12-APR-16	Sales	60000
10144	Kavitha	12-APR-00	25-APR-19	Develop	61000
10159	Hima Shekar	12-NOV-99	30-MAR-15	Plan	65000

10166	Sriram	23-FEB-99	04-APR-18	Develop	60000
10167	Manideep	12-APR-98	23-MAR-19	Develop	65000
10171	SivaSai	30-NOV-98	05-APR-19	HR	72000
10193	Harini	10-JUL-00	31-DEC-16	HR	55000
10207	Bharathi	20-MAR-00	31-JAN-18	HR	67000
10208	Arunkumar	03-JAN-99	20-SEP-18	Develop	67000
10216	Tharun T	28-MAY-01	14-JAN-15	Develop	65000
10225	Sayadeevi	27-JUL-00	15-JAN-19	HR	65000
10235	Anilkumar	22-JUL-01	18-DEC-18	Plan	56000
10236	Balachandar	29-JUN-98	31-DEC-19	Analyst	56000
10240	Balakrishna	08-JUN-01	16-MAY-17	Analyst	65000
10244	Bhumesh	20-MAR-98	30-NOV-17	Develop	60000
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600

17 rows selected.

SQL – GROUP BY Clause: -

- Group by clause is used to group the results of a SELECT query based on one or more columns.
- It is also used with SQL functions to group the result from one or more tables.
- group by is used to group different row of data together based on any one column.

Example:

SQL> select dept, sum(salary) from employee **group by dept**;

DEPT	**SUM(SALARY)**
---	---
Develop	435000
HR	259000
Analyst	121000
Plan	121000
Sales	107600

SQL – HAVING Clause: -

- Having clause is used with SQL Queries to **give more precise condition** for a statement.
- It is used to mention a **condition in Group by based SQL queries**, just like WHERE clause is used with SELECT query.

- **Syntax:** SELECT column_name, function(column_name) FROM table_name GROUP BY column_name HAVING condition;
- **Example:** SQL> select dept, sum(salary) from employee group by dept having sum(salary)<200000;

DEPT	SUM(SALARY)
Analyst	121000
Plan	121000
Sales	107600

SELECT INTO statement:

The SELECT INTO statement is most often used to create backup copies of tables or for archiving records.

Syntax: SELECT Column_name(s) INTO variable_name(s) FROM table_name WHERE condition;

❖ **To Select NULL values**:

We can use the SELECT statement to select the 'null' values also. For retrieving roes where some of the columns have been defined as NULLs there is a special comparison operator of the form IS[NOT]NULL.

Syntax: SELECT column name FROM table_name WHERE Column name IS NULL;

TRIGGERS:

- A **trigger** is a special type of stored procedure that automatically executes when an event occurs.
- Triggers are, written to be executed in response to any of the following events −
- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
- Triggers can be defined on the table, view, schema, or database with which the event is associated.

BENEFITS OF TRIGGERS:

- To generate data automatically.
- To enforce complex integrity constraints. (E.g. checking with sysdate, checking with data in another table).
- To customize complex security authorizations.
- To maintain replicate tables.
- To audit data modifications.

CREATING TRIGGERS:

Syntax:

CREATE OR REPLACE TRIGGER <**trigger_name**>
[BEFORE/AFTER] [INSERT/UPDATE/DELETE] ON <**table_name**>
[FOR EACH STATEMENT/FOR EACH ROW]
[WHEN <**condition**>]
PL/SQL BLOCK;

PARTS OF A TRIGGER

A database trigger has three parts, namely, a trigger statement, a trigger body and a trigger restriction.

a) **Trigger Statement:**
A trigger statement specifies the DML statements like update, delete and insert and it fires the trigger body. It also specifies the table to which the trigger is associated.

b) **Trigger Body:**
Trigger body is a PL/SQL block that is executed when a triggering statement is issued.

c) **Trigger Restriction:**
Restrictions on a triggers can be achieved using the WHEN clause as shown in the syntax for creating triggers. They can be included in the definition of a row trigger, where in, the condition in the WHEN clause is evaluated for each row that is affected by the trigger.

TYPES OF TRIGGER:

Triggers are categorized into the following types based on when they are fired:

- Before
- After
- For each row
- For each statement (default)

Examples:

Q1: Write a PL/SQL program to create a trigger before the user inserts the data into the table.

```
SQL>CREATE or REPLACE TRIGGER INS1
        BEFORE INSERT ON employee
            BEGIN
                RAISE_APPLICATION_ERROR (-20001,'you can't insert a row');
            END;
```

OUTPUT:

```
SQL>insert into employee values(&emp_id, '&emp_name', '&dob', '&addr', '&sex', '&desig',
&deptno, '&maritsta', &salary);
ERROR at line 1:
ORA-20001: you cant insert a row
ORA-06512: at "CSE382.ins1", line 2
ORA-04088: error during execution of trigger 'CSE382.INS1'
```

Q2: Write a PL/SQL program to create a trigger before the user deletes the data from the table.

```
SQL>CREATE or REPLACE TRIGGER DEL1
        BEFORE INSERT ON employee
            BEGIN
                RAISE_APPLICATION_ERROR (-20001,'you can't  delete');
            END;
```

OUTPUT:

```
SQL>delete from employee where emp_id=4444;
delete from employee where emp_id=4444;
```

*

ORA-20001: **you can't delete**

ORA-06512: at "CSE382.DEL1", line 2

ORA-04088: error during execution of trigger '**CSE382.DEL1**'

Q3: Write a PL/SQL program to create a trigger before the user changes the value of the salary of the employee.

```
CREATE TRIGGER UPD1 BEFORE UPDATE ON employee
FOR EACH ROW
BEGIN
        IF :new.sal< 1000 THEN
                        RAISE_APPLICATION_ERROR(-20001,'SALARY CAN'T BE LOW
THAN THIS');
        END IF;
 END;
  /
TRIGGER CREATED.
```

OUTPUT:

SQL> UPDATE **employee** SET **sal=500** WHERE **dno=2**;

UPDATE **employee** SET **sal=500** WHERE **dno=2**

*

ERROR at line 1:

ORA-20001: **salary can't be low than this**

ORA-06512: at "**CSE382.UPD1**", line 3

ORA-04088: error during execution of trigger '**CSE382.UPD1**'

EMBEDDED SQL

1. SQL provides a powerful declarative query language. However, access to a database from a general-purpose programming language is required because,

   - SQL is not as powerful as a general-purpose programming language. There are queries that cannot be expressed in SQL, but can be programmed in C, FORTRAN, Pascal, COBOL, etc.
   - Non-declarative actions -- such as printing a report, interacting with a user, or sending the result to a GUI -- cannot be done from within SQL.

1. Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language (host language).

2. The mixture of SQL and general purpose programming languages is called embedded SQL.

Example:

```
#include<stdio.h>

#include<conio.h>

#include<sqlca.h>

void main()

{

    EXEC SQL

    EXEC SQL BEGIN DECLARE SECTION;

intOrderID;        /* Employee ID (from user)        */

intCustID;          /* Retrieved customer ID        */

charSalesPerson[10];   /* Retrieved salesperson name      */
```

```
char Status[6];        /* Retrieved order status        */
    EXEC SQL END DECLARE SECTION;
    /* Prompt the user for order number */
    printf("Enter order number: ");
    scanf("%d", &OrderID);
    /* Execute the SQL query */
    EXEC SQL SELECT CustID, SalesPerson, Status FROM Orders
    WHERE OrderID= :OrderID
INTO :CustID, :SalesPerson, :Status;


    /* Display the results */
    printf("Customer number:%d\n",:CustID);
    printf ("Salesperson: %s\n",:SalesPerson);
    printf ("Status: %s\n", :Status);
    exit();
}
```

UNIT-III DATA STORAGE AND QUERYING 9

Storage and File Structure - File Organization - Indexing and Hashing- Ordered Indices- Static Hashing- Dynamic Hashing- Query Processing- Overview- Measures of Query Cost-Selection- Sorting- Join Operation- Evaluation of Expressions- Query Optimization- Overview- Transformation of Relational Expressions- Estimating Statistics of Expression Results.

STORAGE SYSTEM:

» Databases are stored in file formats, which contain records.

» At physical level, the actual data is stored in electromagnetic format on some device.

» These storage devices can be broadly categorized into three types –

- **Primary Storage** − The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.
- **Secondary Storage** − Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.
- **Tertiary Storage** − Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

Memory Hierarchy:

- A computer system has a well-defined hierarchy of memory. A CPU has direct access to it main memory as well as its inbuilt registers.
- The access time of the main memory is obviously less than the CPU speed.
- To minimize this speed mismatch, cache memory is introduced. Cache memory provides the fastest access time and it contains data that is most frequently accessed by the CPU.
- The memory with the fastest access is the costliest one.
- Larger storage devices offer slow speed and they are less expensive; however, they can store huge volumes of data as compared to CPU registers or cache memory.

Magnetic Disks:

- Hard disk drives are the most common secondary storage devices in present computer systems.
- These are called magnetic disks because they use the concept of magnetization to store information.
- Hard disks consist of metal disks coated with magnetizable material. These disks are placed vertically on a spindle.
- A read/write head moves in between the disks and is used to magnetize or de-magnetize the spot under it. A magnetized spot can be recognized as 0 (zero) or 1 (one).
- Hard disks are formatted in a well-defined order to store data efficiently.
- A hard disk plate has many concentric circles on it, called **tracks**. Every track is further divided into **sectors**. A sector on a hard disk typically stores 512 bytes of data.

Several types of data storage exist in most computer systems.

They vary in ***speed of access***, ***cost per unit of data***, and ***reliability***. Following are the typical available storage media:

1. Cache Memory.
2. Main Memory.
3. Flash Memory.
4. Magnetic-Disk Storage.
5. Optical Storage.
6. Tape Storage.

1. **Cache Memory:**
   - Most costly and fastest form of storage.
   - Usually very small, and managed by the operating system.

2. **Main Memory:**
   - **Primary storage**, also known as **main memory.**
   - Main Memory **for quick access** by the computer's processor.
   - Copy data from **Storage device into main memory**.
   - Main memory is closely connected to the processor, so moving instructions and data into and out of the **processor is very fast**.
   - The contents of Main Memory are **usually lost** if a power failure or system crash occurs.

3. **Flash Memory:**
   - Flash memory is a **non-volatile memory chip**. It is also known as Erasable Programmable Read-Only Memory (**EEPROM**)
   - Transferring data between a personal computer (PC) and digital devices.
   - It has the ability to be electronically **reprogrammed and erased**.
   - It is often found in USB **flash drives**, **MP3 players**, **digital cameras** and **solid-state drives**.
   - This technology also is useful for computer basic input/output systems (**BIOS**), (**PCMCIA**) cards, modems and video **game cards**.

4. **Magnetic-Disk Storage:**
   - The primary medium for the **long-term on-line storage** of data is the magnetic disk.
   - Usually, the **entire database** is stored on magnetic disk (HDD).
   - The system must move the data from **disk to main memory** so that they can be accessed.
   - The size of the magnetic disks currently ranges from a few **GB to 512 TB** (Available in Market).
   - Disk Storage **survives power failures** and system crashes.

5. **Optical Storage Device:**
   - Optical storage, electronic storage medium that uses **low-power laser beams** to record and retrieve digital (binary) data.

- The most popular forms of optical storage are **Compact Disk** (CD -750MB), **Digital Video Disk** (DVD 4.7 or 8.5 GB).
- **Write-once** versions of CD and DVD are called CD-R and DVD-R respectively.
- **Multiple-write** versions of CD and DVD are called CD-RW and DVD-RW respectively.

6. Tape Storage Device:

- Tape storage is used primarily for **Backup** and **Archival data**.
- **Cheaper**, but much **slower access**, since tape must be read **sequentially** from the beginning.
- Tape storage is referred as sequential-access storage
- Used as protection from disk failures.

FILE ORGANIZATION:

» File Organization defines how file records are mapped onto disk blocks. We have four types of File Organization to organize file records –

1) Heap File Organization:

» When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details.

» File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records.

» Heap File does not support any ordering, sequencing, or indexing on its own.

2) Sequential File Organization

» Every file record contains a data field (attribute) to uniquely identify that record.

» In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key.

» Practically, it is not possible to store all the records sequentially in physical form.

3) Hash File Organization

» Hash File Organization uses Hash function computation on some fields of the records.

» The output of the hash function determines the location of disk block where the records are to be placed.

4) Clustered File Organization

» Clustered file organization is not considered good for large databases.

» In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

File Operations:

Operations on database files can be broadly classified into two categories −

- ✓ **Update Operations**
- ✓ **Retrieval Operations**

» **Update operations** change the data values by insertion, deletion, or update.

» **Retrieval operations**, on the other hand, do not alter the data but retrieve them after optional conditional filtering.

» In both types of operations, **selection** plays a significant role.

» Other than creation and deletion of a file, there could be **several operations**, which can be done on files.

1) **Open** − A file can be opened in one of the two modes, **read mode** or **write mode**. In read mode, the operating system does not allow anyone to alter data. In other words, data is read only. Files opened in read mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.
2) **Locate** − Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.
3) **Read** − By default, when files are opened in read mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.
4) **Write** − User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows to do so.
5) **Close** − This is the most important operation from the operating system's point of view. When a request to close a file is generated, the operating system
    1) removes all the locks (if in shared mode),

2) saves the data (if altered) to the secondary storage media, and
3) releases all the buffers and file handlers associated with the file.

» The organization of data inside a file plays a major role here. The process to locate the file pointer to a desired record inside a file various based on whether the records are arranged sequentially or clustered.

RAID (Redundant Array of Independent Disk)

- ❖ RAID is a way of storing the **same data in different places on multiple hard disks** to protect data in the case of a drive failure.
- ❖ RAID has an arrangement of several independent disks that are organized to improve ***reliability*** and at the same time increase ***performance***.

1) **Improved Performance:**
   - ✓ The Data is segmented into **equal-size partitions** which are transparently distributed across **multiple disks.**
   - ✓ The **Data Stripping** improves overall **I/O performance** and also **balances the load** among disks.
2) **Improved Reliability:**
   - ✓ Storing **redundant information** across the disks using a *parity scheme* or an *error-correcting scheme* to **improve the reliability**.
   - ✓ ***Parity Scheme*: -** Each byte may have a parity bit associated with that record.

- ❖ RAID consists of an array of disks in which multiple disks are connected together to achieve different goals.
- ❖ RAID levels define the use of disk arrays.

1. **RAID-0 (Non-redundant)**
2. **RAID-1 (Mirrored)**
3. **RAID-2 (Memory-Style Error-Correcting Code)**
4. **RAID-3 (Bit Interleaved Parity)**
5. **RAID-4 (Block Interleaved Parity)**
6. **RAID-5 (Bit Interleaved – Distributed Parity)**
7. **RAID-6 (P+Q Redundancy)**

RAID 0 (Non-redundant):

- A **striped array of disks** is implemented.
- The **data is broken** into blocks and the blocks are **distributed among disks**.
- Each disk receives a block of data to **write/read in parallel**.
- It enhances the **speed** and **performanc**e of the storage device.
- There is **no parity** and **no backup** in Level 0.
- It offers the best performance, but **no-fault tolerance**.

RAID 1 (Mirrored):

- RAID 1 uses **mirroring techniques**. When data is sent to a RAID controller, it sends a **copy of data to all the disks** in the array.
- RAID -1 provides **100% redundancy** in case of a failure.
  - ✓ **Read performance is improved** since either disk can be read at the same time.
  - ✓ **Write performance is the same** as for single disk storage.

RAID 2 (Memory-Style Error-Correcting Code):

- RAID 2 generate **Error Correction Code** using Hamming distance for its data, **striped on different disks**.
- The **data is broken** into blocks and the blocks are **distributed among disks** and **ECC codes** of the data are stored on **different set disks**.
- Due to its **complex structure** and **high cost**, RAID 2 is no longer used.

RAID 3 (Bit Interleaved Parity):

- RAID-3 **divide** the data into equal smaller blocks and stored onto **multiple disks**.
- The **parity bit** is generated for data and stored on a **different disk**.
- This technique makes it to overcome **single disk failures**.
- This parity information can be used to **recover the data on other disks.**
- This level uses **less storage space** than RAID 1 but the **parity disk** can become a **bottleneck**.

RAID 4 – Block Interleaved Parity:

- RAID-4 uses **large stripes,** which means you can read records from any single drive.
- An **entire block of data** is written onto data disks and then the parity is generated and stored on a different disk.
- Note that level 3 uses **byte-level striping**, whereas level 4 uses **block-level striping.**
- Both level 3 and level 4 require **at least three disks** to implement RAID.

RAID 5 – Block-Interleaved Distributed Parity:

- RAID-5 writes whole **data blocks onto different disks**.
- The **parity** information is **striped across each drive**, allowing the array to function even if one drive were to fail.

RAID 6 – P+Q Redundancy:

- RAID-6 includes a **second parity scheme** that is distributed across the drives in the array.
- The use of **additional parity** allows the array to continue to function even if **<u>two disks fail</u>** simultaneously.
- This level requires at least **four disk drives** to implement RAID.

INDEXING:

- The data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely.
- Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Index structure:

Search key	Data Reference

- The search key that contains a copy of the **primary key or candidate key** of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The data reference, which contains a **set of pointers holding the address of the disk block** where the value of the particular key can be found.
- Indexing is defined based on its indexing attributes. Indexing can be of the following types –

1) Ordered indices:

- The indices are **usually sorted** to make searching faster. The indices which are sorted are known as ordered indices.

Example: Suppose we have an employee table with thousands of records and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

- ✓ In the case of a database **with no index**, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading **543*10=5430 bytes.**
- ✓ In the case of an **index**, we will search using indexes and the DBMS will read the record after reading **542*2= 1084** bytes which are very less compared to the previous case.

2) Primary Index:

- » If the index is created on the **basis of the primary key** of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
- » As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- » The primary index can be classified into two types: **Dense index** and **Sparse index**.

a) Dense index

- » The dense index contains an **index record for every search key value** in the data file. It makes searching faster.
- » In this, the number of records in the index table is same as the number of records in the main table.
- » It **needs more space** to store index record itself. The index records have the search key and a pointer to the actual record on the disk.

b) Sparse index

» In the data file, index record appears only for a few items. Each item points to a block.

» In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.

3) Clustering Index:

» A clustered index can be defined as an **ordered data file**. Sometimes the index is created on non-primary key columns which may not be unique for each record.

» In this case, to identify the record faster, we will **group two or more columns to get the unique value** and create index out of them. This method is called a clustering index.

» The records which have **similar characteristics are grouped**, and indexes are created for these group. **Example**:

» A company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_ID is a non-unique key.

» The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.

Secondary Index:

» In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster.

» Then the secondary memory searches the actual data based on the address got from mapping.

» If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

- In secondary indexing, to **reduce the size of mapping**, another level of indexing is introduced.
- In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges.
- The mapping of the first level is stored in the primary memory, so that address fetch is faster.
- The mapping of the second level and actual data are stored in the secondary memory (hard disk).

B+ TREE:

- The B+ tree is a **balanced binary search tree**. It follows a **multi-level index format**.
- In the B+ tree, **leaf nodes denote actual data pointers**. B+ tree ensures that all leaf nodes remain at the **same height**.

» In the B+ tree, the **leaf nodes are linked using a link list**. Therefore, a B+ tree can **support random access** as well as **sequential access**.

Structure of B+ Tree:

» In the B+ tree, **every leaf node is at equal distance** from the root node. The B+ tree is of the **order n** where **n is fixed for every B+ tree**.

» It contains an internal node and leaf node.

Internal node:

» An internal node of the B+ tree can contain at least n/2 record pointers except the root node.

» At most, an internal node of the tree contains n pointers.

Leaf node:

» The leaf node of the B+ tree can contain at least n/2 record pointers and n/2 key values.

» At most, a leaf node contains n record pointer and n key values.

» Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

Searching a record in B+ Tree:

» Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.

» So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.

B+ Tree Insertion:

» Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

» In this case, we have to split the leaf node, so that it can be inserted into tree without affecting the fill factor, balance and order.

» The 3rd leaf node has the values (50, 55, 60, 65, 70) and its current root node is 50. We will split the leaf node of the tree in the middle so that its balance is not altered. So, we can group (50, 55) and (60, 65, 70) into 2 leaf nodes.

» If these two has to be leaf nodes, the intermediate node cannot branch from 50. It should have 60 added to it, and then we can have pointers to a new leaf node.

» This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

B+ Tree Deletion:

» Suppose we want to delete 60 from the above example. In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too. If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree. So we need to modify it to have a balanced tree.

» After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:

HASHING:

» In a huge database structure, it is very inefficient to search all the index values and reach the desired data.

» Hashing is an effective technique to **calculate the direct location of a data record** on the disk without using index structure.

» Hashing uses **hash functions** with search keys as parameters to **generate the address of a data record**.

Hash Organization:

- **Bucket** − A hash file stores **data in bucket format**. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.

- **Hash Function** − A hash function, **h,** is a **mapping function** that maps all the set of search-keys **K** to the address where actual records are placed. It is a function from search keys to bucket addresses.

Types of Hashing:

Static Hashing:

» In static hashing, the data bucket address will always be the same. That means if we generate an address for EMP_ID =103 using the **hash function mod (5)** then it will always result in same **bucket address 3**. Here, there will be no change in the bucket address.

» Hence in this static hashing, the number of data buckets in memory **remains constant** throughout.

» In this example, we will have **five data buckets** in the memory used to store the data.

Operations of Static Hashing:

1) **Searching a record**

   When a record needs to be searched, then the same hash function retrieves the address of the bucket where the data is stored.

2) **Insert a Record**

   When a new record is inserted into the table, then we will generate an address for a new record based on the hash key and record is stored in that location.

3) **Delete a Record**

   To delete a record, we will first fetch the record which is supposed to be deleted. Then we will delete the records for that address in memory.

4) **Update a Record**

   To update a record, we will first search it using a hash function, and then the data record is updated.

- If we want to **insert some new record into the file** but the address of a **data bucket** generated by the hash function **is not empty**, or **data already exists** in that address.
- This situation in the static hashing is known as **bucket overflow**. This is a critical situation in this method.
- To overcome this situation, there are various methods. Some commonly used methods are as follows:

1) Open Hashing:

- When a hash function generates an address at which data is already stored, then the next bucket will be allocated to it. This mechanism is called as **Linear Probing**.
- **For example:** suppose R3 is a new address which needs to be inserted, the hash function generates address as 112 for R3. But the generated address is already full. So the system searches next available data bucket, 113 and assigns R3 to it.

2) Close Hashing:

» When buckets are full, then a new data bucket is allocated for the same hash result and is linked after the previous one. This mechanism is known as **Overflow chaining**.

» **For example:** Suppose R3 is a new address which needs to be inserted into the table, the hash function generates address as 110 for it. But this bucket is full to store the new data. In this case, a new bucket is inserted at the end of 110 buckets and is linked to it.

Dynamic Hashing:

» The dynamic hashing method is used to **overcome the problems of static hashing** like bucket overflow.

» In this method, **data buckets grow or shrink** as the records increases or decreases. This method is also known as **Extendable hashing method**.

» This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

Search a key:

1. First, calculate the hash address of the key.
2. Check how many bits are used in the directory, and these bits are called as i.
3. Take the least significant i bits of the hash address. This gives an index of the directory.
4. Now using the index, go to the directory and find bucket address where the record might be.

Insert a new record:

1. Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
2. If there is still space in that bucket, then place the record in it.
3. If the bucket is full, then we will split the bucket and redistribute the records.

For example: Consider the following grouping of keys into buckets, depending on the prefix of their hash address:

Key	Hash address
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111

» The last two bits of 2 and 4 are 00. So, it will go into bucket B0.

» The last two bits of 5 and 6 are 01, so it will go into bucket B1.

» The last two bits of 1 and 3 are 10, so it will go into bucket B2.

» The last two bits of 7 are 11, so it will go into B3.

Insert key 9 with hash address 10001 into the above structure:

- Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
- The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
- Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
- Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
- Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.

Advantages of dynamic hashing:

1. In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
2. In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
3. This method is good for the dynamic database where data grows and shrinks frequently.

Disadvantages of dynamic hashing:

1. In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
2. In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

QUERY PROCESSING:

Query processing refers to the range of activities involved in extracting data from a database. The activities include translation of queries in high-level database language into expressions that can be used at physical level of the file system, a variety of query-optimizing and actual evaluation queries.

Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation

Parsing and translation:

1. Translate the query into its internal form. This is then translated into Relational Algebra.
2. Parser checks syntax, verifies relations

   A relational algebra expression may have many equivalent expressions

   E.g., $\sigma_{balance<2500}(\Pi_{balance}(account))$ is equivalent to $\Pi_{balance}(\sigma_{balance<2500}(account))$

Evaluation:

1. The **Query-Execution Engine** takes a query-evaluation plan, executes that plan, and returns the answers to the query.
2. Each Relational Algebra operation can be evaluated using one of several different algorithms correspondingly, a relational-algebra expression can be evaluated in many ways.
3. Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**.

   Example: can use an index on *balance* to find accounts with balance < 2500,

   (or)

   can perform complete relation scan and discard accounts with balance ≥ 2500

$$\Pi\ \textbf{balance}$$
$$\sigma_{balance<2500}$$
$$\textbf{account}$$

Query Optimization:

- Amongst all equivalent evaluation plans choose the one with lowest cost.
- Cost is estimated using statistical information from the database catalog

Example: Number of tuples in each relation, size of tuples, etc.

Measures of Query Cost:

- Cost is generally measured as total elapsed time for answering query
- Many factors contribute to time cost –
  - *disk accesses, CPU,* or even network *communication*
- Typically, *disk access* is the predominant cost, and is also relatively easy to estimate. Measured by taking into account
  1. Number of seeks * average-seek-cost
  2. Number of blocks read * average-block-read-cost
  3. Number of blocks written * average-block-write-cost

Note:

- ❖ Cost to write a block is greater than cost to read a block.
- ❖ Data is read back after being written to ensure that the write was successful

- For simplicity we just use the *number of block transfers from disk and the number of seeks* as the cost measures
  1. t_T – time to transfer one block
  2. t_S – time for one seek
  3. Cost for b block transfers plus S seeks
     $b * t_T + S * t_S$
- We ignore CPU costs for simplicity
  1. Real systems do take CPU cost into account
- We do not include cost to writing output to disk in our cost formulae
- Several algorithms can reduce disk IO by using extra buffer space
  1. Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution
     - We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Required data may be buffer resident already, avoiding disk I/O
  1. But hard to take into account for cost estimation

Selection Operation:

- **File scan** – search algorithms that locate and retrieve records that fulfill a selection condition.

Algorithm A1 (*linear search*): Scan each file block and test all records to see whether they satisfy the selection condition.

2. Cost estimate = b_r block transfers + 1 seek
   - b_r denotes number of blocks containing records from relation r
3. If selection is on a key attribute, can stop on finding record
   - cost = (b_r /2) block transfers + 1 seek
4. Linear search can be applied regardless of
   - selection condition or
   - ordering of records in the file, or
   - availability of indices

Algorithm A2 (*binary search*): Applicable if selection is an equality comparison on the attribute on which file is ordered.

5. Assume that the blocks of a relation are stored contiguously
6. Cost estimate (number of disk blocks to be scanned):
   - cost of locating the first tuple by a binary search on the blocks
     - $\lceil \log_2(b_r) \rceil * (t_T + t_S)$
   - If there are multiple records satisfying selection
     - *Add transfer cost of the* number of blocks containing records that satisfy selection condition

QUERY OPTIMIZATION:

- **Query optimization** is the part of the **query** process in which the database system compares different **query** strategies and chooses the one with the least expected cost.
- The **query optimizer**, which carries out this function, is a key part of the relational database and determines the most efficient way to access data.
- Query optimization involves three steps, namely query tree generation, plan generation, and query plan code generation.

Step 1 − Query Tree Generation

A query tree is a tree data structure representing a relational algebra expression. The tables of the query are represented as leaf nodes. The relational algebra operations are represented as the internal nodes. The root represents the query as a whole.

During execution, an internal node is executed whenever its operand tables are available. The node is then replaced by the result table. This process continues for all internal nodes until the root node is executed and replaced by the result table.

For example, let us consider the following schemas −

Table Name: EMPLOYEE

EmpID	EName	Salary	DeptNo	DateOfJoining

Table Name: DEPARTMENT

DNo	DName	Location

Example:1

Let us consider the query as the following.

$\Pi_{EmpID} (\sigma_{(EName = ArunKumar)}(EMPLOYEE)$

The corresponding query tree will be −

Example: 2

Let us consider another query involving a join.

$\Pi_{(EName, Salary)} (\sigma_{(DName="Marketing")}(DEPARTMENT)) \bowtie (\sigma_{(DNo=DeptNo)})(EMPLOYEE)$

Following is the query tree for the above query.

Step 2 − Query Plan Generation

- After the query tree is generated, a **query plan** is made. A query plan is an extended query tree that includes **access paths for all operations** in the query tree.
- Access paths specify **how the relational operations in the tree should be performed**. For example, a selection operation can have an access path that gives details about the use of B+ tree index for selection.
- Besides, a query plan also states how the intermediate tables should be passed from one operator to the next, how temporary tables should be used and how operations should be pipelined/combined.

Step 3− Code Generation

- Code generation is the **final step in query optimization**. It is the **executable form of the query**; whose form depends upon the type of the underlying operating system.
- Once the query code is generated, the Execution Manager runs it and produces the results.

Approaches to Query Optimization:

Among the approaches for query optimization, exhaustive search and heuristics-based algorithms are mostly used.

1) Exhaustive Search Optimization:

In these techniques, for a query, all possible query plans are initially generated and then the best plan is selected. Though these techniques provide the best solution, it has an exponential time and

space complexity owing to the large solution space. For example, dynamic programming technique.

2) Heuristic Based Optimization:

Heuristic based optimization uses rule-based optimization approaches for query optimization. These algorithms have polynomial time and space complexity, which is lower than the exponential complexity of exhaustive search-based algorithms. However, these algorithms do not necessarily produce the best query plan.

UNIT IV TRANSACTION AND CONCURRENCY

UNIT-IV TRANSACTIONS 9

Transaction Concept-A Simple Transaction Model-Storage Structure-Transaction Atomicity and Durability-Transaction Isolation- Transaction Isolation Levels- Serializability- Transactions as SQL Statements-Concurrency Control- Lock-Based Protocols- Time Stamp-Based Protocols- Validation Based Protocols- Database recovery - Deadlock Handling.

1. TRANSACTION:

Definition:

"A transaction can be defined as an action or series of actions that is carried out by a single user or application program to perform operations for accessing the contents of the database."

Or

"Transaction is a set of operations which are all logically related."

Or

"Transaction is a single logical unit of work formed by a set of operations."

1.1 Operations in Transaction:

The main operations in a transaction are-

- Read Operation
- Write Operation

1. Read Operation: Read operation reads the data from the database and then stores it in the buffer in main memory.

For example: - **Read(A)** instruction will read the value of A from the database and will store it in the buffer in main memory.

2. Write Operation: Write operation writes the updated data value back to the database from the buffer.

For example: - **Write(A)** will write the updated value of A from the buffer to the database.

1.2 Transaction States:

A transaction goes through many different states throughout its life cycle.

These states are called as **transaction states**.

Transaction states are as follows-

1. Active state
2. Partially committed state
3. Committed state
4. Failed state
5. Aborted state
6. Terminated state

Transaction States in DBMS

1.2.1. Active State:

- This is the first state in the life cycle of a transaction.
- A transaction is called in an **active state** as long as its instructions are getting executed.
- All the changes made by the transaction now are stored in the buffer in main memory.

1.2.2. Partially Committed State:

- After the last instruction of transaction has executed, it enters into a partially committed state.
- After entering this state, the transaction is considered to be partially committed.

- It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

1.2.3. Committed State:

- After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.
- Now, the transaction is considered to be fully committed.

Key Notes:

- After a transaction has entered the committed state, it is not possible to roll back the transaction.
- In other words, it is not possible to undo the changes that have been made by the transaction.
- This is because the system is updated into a new consistent state.
- The only way to undo the changes is by carrying out another transaction called as compensating transaction that performs the reverse operations.

1.2.4. Failed State:

- When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a failed state.

1.2.**5. Aborted State:**

- After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
- To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
- After the transaction has rolled back completely, it enters into an aborted state.

1.2.**6. Terminated State:**

- This is the last state in the life cycle of a transaction.
- After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end.

1.3 ACID Properties:

- It is important to ensure that the database remains consistent before and after the transaction.
- To ensure the consistency of database, certain properties are followed by all the transactions occurring in the system.
- These properties are called as ACID Properties of a transaction.

A = Atomicity

C = Consistency

I = Isolation

D = Durability

1.3.1. Atomicity:

- This property ensures that either the transaction occurs completely or it does not occur at all.
- In other words, it ensures that no transaction occurs partially.
- That is why, it is also referred to as "**All or nothing rule**".
- It is the responsibility of Transaction Control Manager to ensure atomicity of the transactions.

1.3.2. Consistency:

- This property ensures that integrity constraints are maintained.
- In other words, it ensures that the database remains consistent before and after the transaction.
- It is the responsibility of DBMS and application programmer to ensure consistency of the database.

1.3.3. Isolation:

- This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
- During execution, each transaction feels as if it is getting executed alone in the system.
- A transaction does not realize that there are other transactions as well getting executed parallely.

- Changes made by a transaction become visible to other transactions only after they are written in the memory.
- The resultant state of the system after executing all the transactions is same as the state that would be achieved if the transactions were executed serially one after the other.
- It is the responsibility of concurrency control manager to ensure isolation for all the transactions.

1.3.4. Durability:

- This property ensures that all the changes made by a transaction after their successful executions are written successfully to the disk.
- It also ensures that these changes exist permanently and are never lost even if there any failure of any kind.
- It is the responsibility of recovery manager to ensure durability in the database.

1.4 Concurrency Problems

- When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems.
- Such problems are called as **concurrency problems**.

The concurrency problems are: -

1.4.1. Dirty Read Problem: -

"Reading the data written by an uncommitted transaction is called as dirty read."

This read is called as dirty read because-

- There is always a chance that the uncommitted transaction might roll back later.

- Thus, uncommitted transaction might make other transactions read a value that does not even exist.
- This leads to inconsistency of the database.

Transaction T1	Transaction T2
R (A)	
W (A)	
⋮	R (A) // Dirty Read
	W (A)
	Commit
Failure	

Here,

1. T1 reads the value of A.
2. T1 updates the value of A in the buffer.
3. T2 reads the value of A from the buffer.
4. T2 writes the updated the value of A.
5. T2 commits.
6. T1 fails in later stages and rolls back.

In this example,

1. T2 reads the dirty value of A written by the uncommitted transaction T1.
2. T1 fails in later stages and roll backs.
3. Thus, the value that T2 read now stands to be incorrect.
4. Therefore, database becomes inconsistent.

1.4.2. Unrepeatable Read Problem: -

This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.

Transaction T1	Transaction T2
R (X)	
	R (X)
W (X)	
	R (X) // Unrepeated Read

Here,

1. T1 reads the value of X (= 10 say).
2. T2 reads the value of X (= 10).
3. T1 updates the value of X (from 10 to 15 say) in the buffer.
4. T2 again reads the value of X (but = 15).

In this example,

1. T2 gets to read a different value of X in its second reading.
2. T2 wonders how the value of X got changed because according to it, it is running in isolation.

1.4.3. Lost Update Problem: -

This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Transaction T1	Transaction T2
R (A)	
W (A)	
⋮	W (A)
	Commit
Commit	

Here,

1. T1 reads the value of A (= 10 say).
2. T2 updates the value to A (= 15 say) in the buffer.

3. T2 does blind write A = 25 (write without read) in the buffer.
4. T2 commits.
5. When T1 commits, it writes A = 25 in the database.

In this example,

1. T1 writes the over written value of X in the database.
2. Thus, update from T1 gets lost.

1.4.4. Phantom Read Problem:

This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later; it finds that the variable does not exist.

Transaction T1	Transaction T2
R (X)	
	R (X)
Delete (X)	
	Read (X)

Here,

1. T1 reads X.
2. T2 reads X.
3. T1 deletes X.
4. T2 tries reading X but does not find it.

In this example,

1. T2 finds that there does not exist any variable X when it tries reading X again.
2. T2 wonders who deleted the variable X because according to it, it is running in

Avoiding Concurrency Problems: -

- To ensure consistency of the database, it is very important to prevent the occurrence of above problems.
- **Concurrency Control Protocols** help to prevent the occurrence of above problems and maintain the consistency of the database.

2. SCHEDULES: -

Definition: *"The order in which the operations of multiple transactions appear for execution is called as a schedule."*

2.1 Types of Schedules:

In DBMS, schedules may be classified as-

2.2 Serial Schedules: -

In serial schedules,

- All the transactions execute serially one after the other.

- When one transaction executes, no other transaction is allowed to execute.

Characteristics-

Serial schedules are always-

1. Consistent
2. Recoverable
3. Cascadeless
4. Strict

Example:

Transaction T1	Transaction T2
R (A)	
W (A)	
R (B)	
W (B)	
Commit	
	R (A)
	W (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing serially one after the other.
- Transaction T1 executes first.
- After T1 completes its execution, transaction T2 executes.
- So, this schedule is an example of a Serial Schedule.

2.3 Non-Serial Schedules:

In non-serial schedules,

- Multiple transactions execute concurrently.
- Operations of all the transactions are interleaved or mixed with each other.

Characteristics-

Non-serial schedules are **NOT** always-

1. Consistent
2. Recoverable

3. Cascadeless
4. Strict

Example-1:

Transaction T1	Transaction T2
R (A)	
W (B)	
	R (A)
R (B)	
W (B)	
Commit	
	R (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.
- So, this schedule is an example of a Non-Serial Schedule.

Example-2:

Transaction T1	Transaction T2
	R (A)
R (A)	
W (B)	
	R (B)
	Commit
R (B)	
W (B)	
Commit	

In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.
- So, this schedule is an example of a Non-Serial Schedule.

SERIALIZABILITY: -

- Some non-serial schedules may lead to inconsistency of the database.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

2.3.1 SERIALIZABLE SCHEDULES

- If a given non-serial schedule of 'n' transactions is equivalent to some serial schedule of 'n' transactions, then it is called as a ***serializable schedule***.

Characteristics-

Serializable schedules behave exactly same as serial schedules.

Thus, serializable schedules are always-

- Consistent
- Recoverable
- Cascadeless
- Strict

Serial Schedules Vs Serializable Schedules

Serial Schedules	Serializable Schedules
No concurrency is allowed. Thus, all the transactions necessarily execute serially one after the other.	Concurrency is allowed. Thus, multiple transactions can execute concurrently.
Serial schedules lead to less resource utilization and CPU throughput.	Serializable schedules improve both resource utilization and CPU throughput.
Serial Schedules are less efficient as compared to serializable schedules. (due to above reason)	Serializable Schedules are always better than serial schedules. (due to above reason)

Types of Serializability: -

Serializability is mainly of two types-

1. Conflict Serializability
2. View Serializability

2.3.1.1 CONFLICT SERIALIZABILITY: -

- If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a conflict serializable schedule.

Conflicting Operations-

- Two operations are called as conflicting operations if all the following conditions hold true for them-
  1. Both the operations belong to different transactions
  2. Both the operations are on the same data item
  3. At least one of the two operations is a write operation

Example-

Consider the following schedule-

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.
- This is because all the above conditions hold true for them.

Checking Whether a Schedule is Conflict Serializable Or Not-

Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

Step-01: Find and list all the conflicting operations.

Step-02: Start creating a precedence graph by drawing one node for each transaction.

Step-03: # Draw an edge for each conflict pair such that if Xi (V) and Y_j (V) forms a conflict pair then draw an edge from Ti to Tj.

This ensures that Ti gets executed before Tj.

Step-04: Check if there is any cycle formed in the graph.

If there is no cycle found, then the schedule is conflict serializable otherwise not.

Practice Problems Based On Conflict Serializability-

Problem-01: Check whether the given schedule S is conflict serializable or not-

S : R1(A) , R2(A) , R1(B) , R2(B) , R3(B) , W1(A) , W2(B)

Solution-

Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$

Step-02:

Draw the precedence graph-

- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Problem-02: Check whether the given schedule S is conflict serializable and recoverable or not-

T1	T2	T3	T4
	R(X)		
		W(X)	
		Commit	
W(X)			
Commit			
	W(Y)		
	R(Z)		
	Commit		
			R(X)
			R(Y)
			Commit

Solution-

Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- R2(X) , W3(X) (T2 → T3)
- R2(X) , W1(X) (T2 → T1)
- W3(X) , W1(X) (T3 → T1)
- W3(X) , R4(X) (T3 → T4)
- W1(X) , R4(X) (T1 → T4)
- W2(Y) , R4(Y) (T2 → T4)

Step-02:

Draw the precedence graph-

- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.

2.3.1.2 VIEW SERIALIZABILITY: -

Definition: "*If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule."*

View Equivalent Schedules-

- Consider two schedules S1 and S2 each consisting of two transactions T1 and T2.
- Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them-

Condition-01:

- For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.

Condition-02:

- If transaction T_i reads a data item that has been updated by the transaction T_j in schedule S1, then in schedule S2 also, transaction T_i must read the same data item that has been updated by the transaction T_j.

Condition-03:

- For each data item X, if X has been updated at last by transaction T_i in schedule S1, then in schedule S2 also, X must be updated at last by transaction T_i.

Checking Whether a Schedule is View Serializable or Not-

Method-01:

- Check whether the given schedule is conflict serializable or not.
- If the given schedule is conflict serializable, then it is surely view serializable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be view serializable.

Key Points:

- All conflict serializable schedules are view serializable.
- All view serializable schedules may or may not be conflict serializable.

Method-02:

- Check if there exists any blind write operation.
  *(Writing without reading is called as a **blind write**).*
- If there does not exist any blind write, then the schedule is surely not view serializable. Stop and report your answer.
- If there any blind write, then the schedule may or may not be view serializable.

Key Points:

- *No blind write means not a view serializable schedule.*

Method-03:

1. In this method, try finding a view equivalent serial schedule.
2. Then, draw a graph using those dependencies.
3. If there exists no cycle in the graph, then the schedule is view serializable otherwise not.

Checking Whether a Schedule is View Serializable Or Not-

Problem-01:

T1	T2	T3	T4
R (A)			
	R (A)		
		R (A)	
			R (A)
W (B)			
	W (B)		
		W (B)	
			W (B)

- If a schedule is conflict serializable, then it is surely view serializable.
- List all the conflicting operations and determine the dependency between the transactions-
  - W1(B) , W2(B) (T1 → T2)
  - W1(B) , W3(B) (T1 → T3)
  - W1(B) , W4(B) (T1 → T4)

- W2(B) , W3(B) (T2 → T3)
- W2(B) , W4(B) (T2 → T4)
- W3(B) , W4(B) (T3 → T4)

Draw the precedence graph-

- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.
- Thus, we conclude that the given schedule is also view serializable.

2.3.2 NON SERILIAZABLE SCHEDULE:

Definition:

"A non-serial schedule which is not serializable is called as a non-serializable schedule."

Characteristics-

- May or may not be consistent
- May or may not be recoverable

Types of Non-Serializable Schedule:

2.3.2.1 RECOVERABLE SCHEDULE:

If in a schedule,

1. A transaction performs a dirty read operation from an uncommitted transaction.
2. And its commit operation is delayed till the uncommitted transaction commits or roll backs, and then such a schedule is known as a ***Recoverable Schedule***.

Here,

- T2 performs a dirty read operation.
- The commit operation of T2 is delayed till T1 commits or roll backs.
- T1 commits later.
- T2 is now allowed to commit.
- In case, T1 would have failed, T2 has a chance to recover by rolling back.

2.3.2.2 NON -RECOVERABLE SCHEDULE:

If in a schedule,

1. A transaction performs a dirty read operation from an uncommitted transaction
2. And commits its operations before the transaction from which it has read the value then such a schedule is known as an ***Non - Recoverable Schedule.***

Here,

- T2 performs a dirty read operation.
- T2 commits before T1.
- T1 fails later and roll backs.
- The value that T2 read now stands to be incorrect.
- T2 can not recover since it has already committed.

Checking Whether a Schedule is Recoverable or Irrecoverable:

Method-01:

- Check whether the given schedule is conflict serializable or not.
- If the given schedule is conflict serializable, then it is surely recoverable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be recoverable. Go and check using other methods.

Key Points:

- All conflict serializable schedules are recoverable.
- All recoverable schedules may or may not be conflict serializable.

Method-02:

- Check if there exists any dirty read operation.
  *(Reading from an uncommitted transaction is called as a **dirty read**)*
- If there does not exist any dirty read operation, then the schedule is surely recoverable. Stop and report your answer.
- If there any dirty read operation, then the schedule may or may not be recoverable.

Checking Whether a Schedule is Recoverable or Irrecoverable:

Problem: 01

SCHEDULE B

T1	T2
Read(X)	
Write(X)	
	Read(X)
Read(Y)	
	Write(X)
	Commit
Abort	

- The Schedule B is Not Recoverable.
- T2 reads item X from T1.
- T2 commits before T1 commits.
- T1 Aborts after T2 Commits.
- The value of X that T2 read is no longer valid.
- T2 must be Aborted after its Committed.
- Leading to schedule that is not recoverable.

Key Points:

- For the schedule to be recoverable, the T2 Commit operation in Schedule B must be postponed until T1 Commits.

Problem: 02

SCHEDULE C

T1	T2
Read(X)	
Write(X)	
	Read(X)
Read(Y)	
	Write(X)
Write(Y)	
Commit	
	Commit

- The Schedule C is Recoverable.
- T2 reads item X from T1.
- T2 commits after T1 commits.

Problem: 03

SCHEDULE D

T1	T2
Read(X)	
Write(X)	
	Read(X)
Read(Y)	
	Write(X)
Write(Y)	
Abort	
	Abort

- The Schedule D is Recoverable.
- T2 reads item X from T1.
- T1 Aborts instead of Committing, Then T2 Should also Abort.
- Because the value of X it read is no longer valid

2.3.1.1 TYPES OF RECOVERABLE SCHEDULE:

I. Cascading Schedule:

If in a schedule,

1. Failure of one transaction causes several other dependent transactions to rollback or abort, and then such a schedule is called as a ***Cascading Schedule*** or ***Cascading Rollback*** or ***Cascading Abort***.
2. It simply leads to the wastage of CPU time.

Example:

T1	T2	T3	T4
R (A)			
W (A)			
	R (A)		
	W (A)		
		R (A)	
		W (A)	
			R (A)
			W (A)
Failure			

Cascading Recoverable Schedule

Hère,

- Transaction T2 dépends on transaction T1.
- Transaction T3 dépends on transaction T2.
- Transaction T4 dépends on transaction T3.

In this schedule,

- The failure of transaction T1 causes the transaction T2 to rollback.
- The rollback of transaction T2 causes the transaction T3 to rollback.
- The rollback of transaction T3 causes the transaction T4 to rollback.

Such a rollback is called as a **Cascading Rollback**.

II. Cascadeless Schedule-

- If in a schedule,
- "A transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called As a Cascadeless Schedule."
- In other words, ***"Cascadeless schedule allows only committed read operations."***
- Therefore, it avoids cascading roll back and thus saves CPU time.

T1	T2	T3
R (A)		
W (A)		
Commit		
	R (A)	
	W (A)	
	Commit	
		R (A)
		W (A)
		Commit

Cascadeless Schedule

- Cascadeless schedule allows only committed read operations.
- However, it allows uncommitted write operations.

III. Strict Schedule-

If in a schedule,

- "A transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, and then such a schedule is called as a **Strict Schedule**."

In other words,

- "Strict schedule allows **only committed read and write** operations."
- Strict schedule implements more restrictions than Cascadeless schedule.

T1	T2
W (A)	
Commit / Rollback	
	R (A) / W (A)

Strict Schedule

Remember-

- Strict schedules are more strict than Cascadeless schedules.
- All strict schedules are Cascadeless schedules.
- All Cascadeless schedules are not strict schedules.

3. CONCURRENCY CONTROL:

- Concurrency control techniques that are used to ensure the non-interference or Isolation property of concurrently executing transactions.
- There are **three methods for concurrency control**. They are as Follows:

1. Locking Methods
2. Time-stamp Methods
3. Optimistic Methods

1. Locking Methods of Concurrency Control:

- "A lock is a variable, associated with the data item, which controls the access of that data item."
- Locking is the most widely used form of the concurrency control mechanism.
- Locks are further divided into three fields:

a) Lock Granularity
b) Lock Types
c) Deadlocks

a) Lock Granularity:

- A database is a collection of named data items.
- The size of the data item chosen as the unit of protection by a concurrency control program is called **GRANULARITY**.

- Locking can take place at the following level :
  1. Database level locking.
  2. Table level locking.
  3. Page level locking.
  4. Row (Tuple) level locking.
  5. Attributes (fields) level locking.

b) Lock Types:

The DBMS mainly uses following types of locking techniques.

1) Binary Locking
2) Shared / Exclusive Locking
3) Two - Phase Locking (2PL)

1) **BINARY LOCKS**:

- A binary lock can have two states or values:
  1) Locked state - (1)
  2) Unlocked state - (0)
- If the value of the lock on X is 1, then the item X cannot be accessed by a other operation.
- If the value of the lock on X is 0, the item can be accessed by other operation.
- The current state of the lock associated with item X is **LOCK(X).**
- Two operations, ***lock_item*** and ***unlock_item***, are used with binary locking.

Lock_item(X):

- A transaction requests an item.
- If LOCK(X) = 1, then transaction has to **wait**.
- If LOCK(X) = 0, it is set to 1 and the transaction is allowed to access item X.

Unlock_item (X):

- Sets LOCK(X) to 0 (unlocks) so that X may be accessed by other transactions. (LOCK(X)←0)
- A binary lock enforces mutual exclusion on the data item ; i.e., at a time only one transaction can hold a lock.

Binary Lock Rules:

1. A transaction T must issue the operation ***lock_item(X)*** before any ***read_item(X)*** or ***write_item(X)*** operations are performed in T.
2. A transaction T must issue the operation ***unlock_item(X)*** after all ***read_item(X)*** and write_item(X) operations are completed in T.
3. A transaction T will not issue a ***lock_item(X)*** operation if it already holds the lock on item X.
4. A transaction T will not issue an ***unlock_item(X)*** operation unless it already holds the lock on item X.

2) **SHARED / EXCLUSIVE LOCK:**

- Binary locking scheme allows at most one transaction can hold a lock on a given time.
- We should allow several transactions to access the same data item X if they all access X for reading purposes only.
- If a transaction is to write a data item X, it must have exclusive access to X.
- For this purpose, a different type of lock called multiple-mode lock is used.
- This scheme is called shared/exclusive lock.
- There are three locking operations:

  1. Read_lock(X) ***2. Write_lock(X)*** ***3.Unlock(X)***
- A lock associated with an data item X, **LOCK(X)**
- Has three states – 1) read-locked state
  2) write-locked state
  3) unlocked state.
- A read-locked item is also called share-locked because other transactions are allowed to read the item.
- A write-locked item is called exclusive –locked because a single transaction exclusively holds the lock on the item.

Shared Lock:

- These locks are referred as read locks, and denoted by **'S'.**
- If a transaction T has obtained Shared-lock on data item X, then T can read X, but cannot write X.
- Multiple Shared lock can be placed simultaneously on a data item.

Exclusive lock:

- These Locks are referred as Write locks, and denoted by **'X'.**
- If a transaction T has obtained Exclusive lock on data item X, then **T can be read as well as write X**.
- Only one Exclusive lock can be placed on a data item at a time.
- This means multiple transactions does not modify the same data simultaneously.

Shared / Exclusive Rules:

1. A transaction T must issue the operation ***read_lock(X)*** or ***write_lock(X)*** before any ***read_item(X)*** operation is performed in T.
2. A transaction T must issue the operation write_lock(X) before any ***write_item(X)*** operation is performed in T.
3. Transaction T must issue the operation unlock(X) after all ***read_item(X)*** and ***write_item(X)*** operations are completed in T.
4. A transaction will not issue a ***read_lock(X)*** operation if it already holds a read (shared) lock or a write (exclusive) lock on item X.
5. A transaction will not issue a ***write_lock(X)*** operation if it already holds a read (shared) lock or a write (exclusive) lock on item X.
6. A transaction will not issue a ***unlock(X)*** operation unless it already holds a ***read_lock(shared)*** or a ***write_lock (exclusive)*** on item X.

Example:

T1:

Lock-X(B)
Read(B)
B:=B-50
Write(B)
Unlock(B)
Lock-X(A)
Read(A)
A:=A+50
Write(A)
Unlock(A)

T2:

Lock-S(A)
Read(A)
Unlock(A)
Lock-S(B)
Read(B)
Unlock(B)
Display(A+B)

3) **TWO-PHASE LOCKING (2PL):**

- Two-phase locking (**2PL**) is a method of controlling concurrent processing in which all locking operations precede the first unlocking operation.

- Thus, a transaction is said to follow the two-phase locking protocol if all locking operations (read_Lock, write_Lock) precede the first unlock operation in the transaction.
- Two-phase locking is the standard protocol used to defines how transactions acquire and hand over locks.
- This protocol requires that each transaction issue lock and unlock requests in two phases.
  1. **Growing Phase**: In this phase, a transaction may obtain locks, but may not release any lock.
  2. **Shrinking Phase**: In this phase, a transaction may release locks, but may not obtain any new locks.
- Initially, a transaction is in the ***growing phase***. The transaction acquires locks as needed.
- Once the transaction releases a lock, it enters in the ***shrinking phase***, and it cannot issue more lock request.

Example: 01

T3	**Lock-X(B)**
	Read(B)
	B:=B-50
	Write(B)
	Lock-X(A)
	Read(A)
	A:=A+50
	Write(A)
	Unlock(B)
	Unlock(A)

Example: 02

T1	T2
Raed_lock(Y)	Read_lock(X)
Read_item(Y)	Read_item(X)
Unlock(Y)	**Unlock(X)**
Write_lock(X)	**Write_lock(Y)**
Read_item(X)	Read_item(Y)
X:X+Y	Y:=X+Y
Write_item(X)	Write_item(Y)
Unlock(X)	Unlock(Y)

Not obey two phase locking

Example: 03

T1	T2
Raed_lock(Y)	Read_lock(X)
Read_item(Y)	Read_item(X)
Write_lock(X)	**Write_lock(Y)**
Unlock(Y)	**Unlock(X)**
Read_item(X)	Read_item(Y)
X:X+Y	Y:=X+Y
Write_item(X)	Write_item(Y)
Unlock(X)	Unlock(Y)

Obey two phase locking

Advantages:

1. The two-phase locking ensures conflict serializability.

Disadvantages:

1. Two-phase locking does not ensure freedom from deadlock.
2. Cascading Roll-backing may occur under two-phase locking.

STRICT TWO-PHASE LOCKING (2PL):

- Strict two-phase locking method used in concurrent systems.
- Strict Two-Phase is most used in real-world implementation.

- **Rules of Strict 2PL**:

1. If a transaction T wants to read/write an object, it must request a shared/exclusive lock on the object.
2. All exclusive locks held by transaction T are released when T commits.
3. A Transaction T does not release any of its Exclusive Lock until it commits or Aborts.
4. Hence, no other transaction can read or write an item that is written by T unless T has committed.

T1	T2
Lock-S(A)	
Read(A)	
	Lock-S(A)
	Read(A)
	Loc-X(B)
	Read(B)
	Write(B)
	Commit
Lock-X(C)	
Read(C)	
Write(C)	
Commit	

- Strict Two-phase locking prevents transaction from reading uncommitted data, overwriting uncommitted data and unrepeatable reads.
- Strict 2PL prevents Cascading Rollbacks.
- Strict 2PL does not guarantee a Deadlock Free Schedule.

RIGOROUS TWO-PHASE LOCKING (2PL):

- This protocol that all locks be held until the transactions commits.

C. DEADLOCKS :

Definition:

"A System is in deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set."

"There exists a set of transactions {T_0, T_1, T_2,….T_n} such that T_0 is waiting for a data item that T_2 holds and T_2 waiting for a data item that T_3 holds…. T_n is waiting for data item that T_0 holds. In such situation, none of the transaction can make progress"

T1	T2
Lock-X(B)	
Read(B)	
B:=B-50	
Write(B)	
	Lock-X(A)
	Read(A)
	Lock-S(B)
Lock-X(A)	

- Here T1 is holding an Exclusive lock on B and t2 is requesting a Shared lock on B, T2 is waiting for T1 to unlock B.
- Similarly, T2 is holding Shared lock on A, T1 is requesting an Exclusive lock on A, T1 is waiting for T2 to unlock A.
- In this situation neither of transaction can proceed with normal execution.
- This situation is called DEADLOCK.

There are two principle methods for dealing with deadlock problem

1. **Deadlock Prevention**: This approach ensures that the system will never enter in deadlock state.
2. **Deadlock Detection and Recovery**: This approach tries to recover from Deadlock if system enters in deadlock state.

Deadlock Prevention:

- Deadlock prevention technique avoids the conditions that lead to deadlocking.
- It requires that every transaction lock all data items it needs in advance.
- In other words, a transaction requesting a new lock is aborted if there is the possibility that a deadlock can occur.

There are two approaches for deadlock prevention:

Approach: 1

- This ensures that no cyclic wait can occur by ordering the request for locks.
- It requires that every transaction lock all data items it needs in advance.
- It is required that, either **all data items should be locked** in one step, or **none should be locked.**

Disadvantages of this approach-1:

1. It is **hard to predict** before the transaction begins, What data items need to be locked.
2. Data item **utilization may be very low**, because many of the data items may be locked but unused for a long time.

Approach: 2

- Deadlock prevention is to use **preemption** and **transaction rollbacks**.
- In preemption when a transaction T2 request a lock that transaction T1 holds.
- The lock granted to T1 may be preempted by rolling back T1, and granting of lock to T2.
- To control preemption, a unique timestamp is assigned to each transaction.
- The system uses timestamp to decide whether a transaction wait or roll back.
- Two different deadlock prevention schemes using timestamp are:
  - Wait Die.
  - Wound Wait.

1. **<u>Wait Die</u>**:

- The Wait-Die scheme is non-preemption technique.
- When Transaction T_i request a data item held by T_j, T_i is allowed to wait only if it has a timestamp smaller than T_j.
- Otherwise, T_i is rolled back (Die)

Example:

- Consider three transactions T1, T2, T3 with timestamps 5, 10, 15 respectively.
- If T1 request a data item held by T2, Then T1 will wait.
- If T3 request data item held by T2, then T2 will be Rolled Back.

2. **Wound Wait**:

- The Wound Wait scheme is preemption technique.
- When Transaction T_i request a data item held by T_j, T_i is allowed to wait only if it has a timestamp greater than T_j.
- Otherwise, T_i is rolled back (Die)

Example:

- Consider three transactions T1, T2, T3 with timestamps 5, 10, 15 respectively.
- If T1 request a data item held by T2, Then T2 will be Rolled Back.
- If T3 request data item held by T2, then T3 will wait.

Deadlock detection:

- This technique allows deadlock to occur, but then, it detects it and solves it.
- Here, a database is periodically checked for deadlocks.
- If a deadlock is detected, one of the transactions, involved in deadlock cycle, is aborted. other transaction continue their execution.

An aborted transaction is rolled back and restarted.

4. **DATABASE RECOVERY TECHNIQUES**

Database Failure:

- We cannot guarantee a failure-free transaction every time. There are several types of failures that affect database processing.
- Some failures may affect the disk storage, while some may only affect the data items of the database residing in the main memory.

Types Failure

- Failures are generally classified as Transaction, System, and Media Failures.
- There are several possible reasons for a transaction to fail in the middle of execution.

1. **A Computer Failure** (System Crash):
   - A hardware, software, or network error occurs in the computer system during transaction execution.
   - Hardware crashes are usually Media Failure

- Example: Main Memory Failure.

2. **A Transaction or System error**:
   - Some operation in the transaction may cause it to fail, such as **integer overflow** or **divide by zero**.
   - Transaction Failure may also occur because of **erroneous parameter values** or because of a **logical programming error**.

3. **Disk Failure**:
   - Some disk blocks may lose their data because of a read or write malfunction or because of a read/write head crash.
   - This may happen during a read or a write operation of the transaction.

4. **Physical Problem and Catastrophes**:
   - This refers to an endless list of problems that includes power failure or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

- Failure of types 1 and 2 are more common.
- Whenever a failure of type 1 and 2 occurs, the system must keep sufficient information to recover from the failure.
- Disk Failure and Catastrophic Failure are do not happen frequently.
- If they occur, recovery is a major task.

Database Recovery Techniques

- Recovery from transaction failure means that the database is restored to the most recent consistent state just before the time of failure.
- Wide portion of the database due to catastrophic failure, the recovery method **restores past copy** of the database and **reconstructs** a more current state by **redoing** the operations **up to the time of failure**.
- When the database is not physically damaged, the recovery system is to reverse any changes that caused the inconsistency by undoing some operations.

- It may also redo some operation in order to restore a consistent state of the database.

There are many different approaches to recover a database: Manual reprocessing, a variety of automated recovery techniques.

1. Manual Reprocessing

- The database is periodically backed up and all transactions applied since the last save are recorded.
- If the system crashes, the latest database save is restored and all of the transactions are re-applied (by users) to bring the database back up to the point just before the crash.

Limitations: 1) **Time required** reapplying transactions.
2) Transactions **might have other** (physical) potential **failures**.
3) Reapplying concurrent transactions is **not straight forward**.

2. **Automated Recovery:**

There are several types of automated recovery techniques including: deferred update and immediate update, shadow paging, etc.

A transaction can be in one of the following states:

1. **Active** - When the transaction just begins
2. **Partially Committed** - After the last operation has completed (but before the commit point is reached)
3. **Failed** - Normal operation is prevented.
4. **Aborted** - Transaction is rolled back.
5. **Committed** - Transaction completes all operations and moved the database to the next consistent state.

Each transaction writes the following information to the log:

6. Start(T) - the fact that transaction T has started.
7. Write(T, X, old_value, new_value) - The Transaction T has written to item X with the new_value. old_value is also maintained.
8. Read(T, X) - The Transaction T has read data item X
9. Either
10. Commit(T) - Transaction T committed, or Abort(T) - transaction T was aborted

Checkpoint:

- A recovery point in the logs where all current transactions have terminated and all updates have been written to disk. *Consists of 4 steps*:
  1. Cease accepting new transactions
  2. Allow all unfinished transactions to complete (commit or abort)
  3. Write all pending changes to disk and to logs
  4. Resume accepting new transactions
- In many environments, it is possible to take checkpoints each 15 minutes or half hour, etc.
- Recovery must then only be done from the time of the last checkpoint.

1. DEFERRED UPDATE RECOVERY:

- Also called NO-UNDO/REDO
- During a transaction, only record the changes to data items in the log.
- When the transaction commits, update the data items on disk.
- Two main rules:
  1. A transaction cannot change any items in the database until it commits.
  2. A transaction may not commit until the entire write operations are successfully recorded in the log.

     This means that we must check the log is actually written to disk.

```
T1:    Ra Rd Wd C
T2:    Rb Wb Rd Wd C
T3:    Ra Wa Rc Wc C
T4:    Rb Wb Ra Wa C

Log file:
Start(T1)
Write(T1, d, old, new)
Commit(T1)
checkpoint
Start(T4)
Write(T4, b, old, new)
Write(T4, a, old. new)
Commit(T4)
Start(T2)
Write(T2, b, old, new)
Start(T3)
Write(T3, a, old, new)
    s y s t e m  c r a s h
```

- Since T1 and T4 committed, their changes were written to disk.
- However, T2 and T3 did not commit, hence their changes were not written to disk.
- T2 and T3 are ignored because they did not reach their commit points.
- T4 is redone because its commit point is after the last system checkpoint.

Advantages:

1. Recovery is made easier:

   Any transaction that reached the commit point (from the log) has its writes applied to the database. All other transactions are ignored.

2. Cascading rollback does not occur because no other transaction sees the work of another until it is committed.

Disadvantages:

1. Concurrency is limited: Must employ Strict 2PL which limits concurrency.

2. **IMMEDIATE UPDATE RECOVERY**

- A Transaction issues an update command; the database can be updated immediately, without any need to wait for the transaction to reach its commit point.
- An update operation must still be *recorded in the log (on disk*) before it is applied to the database – using the ***write ahead logging protocol***.
- So that we can recover in case of failure.
- Immediate Update allows the write operations to the database as the transaction is executing.
- Writes are still saved in the log before being applied to the database - a ***Write-Ahead Log*** (WAL)
- Maintain two logs:
- **REDO log**: A record of each new data item in the database.

  UNDO log: A record of each updated data item (old values).

Two rules:

1. Transaction T may not update the database until all UNDO entries have been written to the UNDO log.
2. Transaction T is not allowed to commit until all REDO and UNDO log entries are written (*forced-written* to disk).

To Recover:

1. Begin at the end of the log and read backwards to the last checkpoint Create two lists:
   - ✓ **C** - transactions that have committed (Committed Transaction)
   - ✓ **NC** - transactions that did not commit (Active Transaction)

- ✓ Undo all the write operations of the Active transactions from the log, using UNDO procedure.
- ✓ Redo the write operations of the Committed Transactions from the log using REDO procedure.

```
T1:   Ra Rd Wd C
T2:   Rb Wb Rd Wd C
T3:   Ra Wa Rc Wc C
T4:   Rb Wb Ra Wa C

Log file:
Start(T1)
Write(T1, d, old, new)
Commit(T1)
checkpoint
Start(T4)
Write(T4, b, old, new)
Write(T4, a, old. new)
Commit(T4)
Start(T2)
Write(T2, b, old, new)
Start(T3)
Write(T3, a, old, new)
    s y s t e m  c r a s h
```

- ❖ Since T1, T2, T3 and T4 changes were written to disk.
- ❖ However, T1 and T4 are committed.
- ❖ T2 and T3 are not committed.
- ❖ T2 and T3 are in Active Transaction List and T4 is in Committed Transaction.
- ❖ T2 and T3 are Undone from the log using UNDO procedure.
- ❖ T4 is redone because its commit point is after the last system checkpoint using REDO procedure.

Advantages:

Immediate update **allows higher concurrency** because transactions write continuously to the database rather than waiting until the commit point.

Disadvantages:

Can lead to cascading rollbacks - time consuming and may be problematic.

3. **<u>SHADOW PAGING</u>:**

- ❖ The AFIM (After Image) does not overwrite its BFIM (Before Image) but recorded at another place on the disk.
- ❖ Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.

- To recover from a failure, it is sufficient to free the modified pages and discard the current directory.
- The state of the database before transaction execution is available through the shadow directory.
- Database can be returned to its previous state that was executing when the crash occurred.
- We can categorize as a **NO-UNDO/NO-REDO** technique for recovery Logs and checkpoints must be incorporated into the shadow paging technique.
- **Disadvantages:**

Complex storage management strategies, the overhead of writing shadow directories to disk, garbage collection overhead (old pages referenced by the shadow directory).

Unit – 5

Database Security and Object – Based Databases

Unit – 5

Database Security and Object – Based Databases

Syllabus:

Database Security-Authentication, Authorization and access control - DAC, MAC and RBAC models –SQL Injection-Object based databases- Object oriented and object relational databases- Persistent Programming Languages-Object-Relational Mapping-Object-Oriented versus Object-Relational Databases – Logical and web databases-Overview of Data warehousing - Data mining techniques.

Security Risk to Database Includes

- Unauthorized database users.
- Unauthorized Database Administrator
- Unauthorized access to Database
- Unauthorized alternation to available data
- Lack of access to Database services

Sensitive data includes

- ✓ Bank/Demat accounts
- ✓ Credit card, Salary,
- ✓ Income tax data\
- ✓ University admissions, marks/grades
- ✓ Land records, licenses

Database Security

Database Security is defined as the process by which "Confidentiality, Integrity and Availability" of the database can be protected.

Database Security Concepts

1. Confidentiality
2. Integrity
3. Availability

Database Security Concepts

(1) Confidentiality

- Enforced by encrypting the data in the stored database.
- Encryption is a technique or a process by which the data is encoded in such a way that only that authorized users are able to read the data.
- Encryption is rendering sensitive data unreadable to unauthorized users.

Database Security Concepts

(2)Integrity

- Enforced by defining which user has to be given permission to access the data in the database.
- **Example:** Data related to employee may have permission for viewing records and altering only the part of information like his contact details, where as the person like Human resource manager will have more privileges.

Database Security Concepts

(3)Availability

- Database must have not unplanned downtime.
- To ensure this ,following steps should be taken –

1. Restrict the amount of the storage space given to each user in the database.
2. Limit the number of concurrent sessions made available to each database user.
3. Back up the data at periodic intervals to ensure data recovery in case of application users.

Threats to Database

- ✓ SQL Injection.
- ✓ Unauthorized access.
- ✓ Password Cracking.
- ✓ Network EavesDropping.

Threats to Database

(1)SQL Injection

A form of attack on a database-driven Web site in which the attacker executes unauthorized SQL commands by taking advantage of insecure code on a system connected to the Internet, bypassing the firewall.

Vulnerabilities:

- ✓ Poor Input validation to web application.
- ✓ Unsafe ,dynamically constructed SQL commands.
- ✓ Weak permissions that fail to restrict the application to Database

Threats to Database

(1)SQL Injection

Countermeasures:

- ✓ Your application should constrain and sanitize input data before using it in SQL queries.
- ✓ Use type safe SQL parameters for data access. These can be used with stored procedures or dynamically constructed SQL command strings. Using SQL parameters ensures that input data is subject to type and length checks.
- ✓ Use a SQL Server login that has restricted permissions in the database. Ideally, you should grant execute permissions only to selected stored procedures in the database and provide no direct table access.

Threats to Database

(2)Unauthorized Access

Direct access to your database server should be restricted to specific client computers to prevent unauthorized server access.

Vulnerabilities:

- ✓ Failure to block the SQL Server port at the perimeter firewall
- ✓ Lack of IPSec or TCP/IP filtering policies

Threats to Database

(2)Unauthorized Access

Countermeasures:

- ✓ Make sure that SQL Server ports are not visible from outside of the perimeter network.
- ✓ Within the perimeter, restrict direct access by unauthorized hosts, for example, by using IPSec or TCP/IP filters.

Threats to Database

(3) Password cracking

- ✓ A common first line of attack is to try to crack the passwords of well known account names, such as SA (the SQL Server administrator account).

Vulnerabilities:	Countermeasures:
✓ Weak or blank passwords ✓ Passwords that contain everyday words	✓ Create passwords for SQL Server login accounts that meet complexity requirements. ✓ Avoid passwords that contain common words found in the dictionary.

Threats to Database

(4) Network Eavesdropping

- ✓ Eavesdropping refers to unauthorized access of reading messages.
- ✓ The deployment architecture of most applications includes a physical separation of the data access code from the database server.
- ✓ As a result, sensitive data, such as application-specific data or database login credentials, must be protected from network eavesdroppers.

Threats to Database

(4) Network Eavesdropping

Vulnerabilities:

- ✓ Insecure communication channels
- ✓ Passing credentials in clear text to the database; for example:
  - Using SQL authentication instead of Windows authentication
  - Using SQL authentication without a server certificate

Methods of securing the database

- ✓ Authorization - privileges, views.
- ✓ Authentication – passwords.
- ✓ Encryption - public key / private key, secure sockets.
- ✓ Logical - firewalls, net proxies.

(1) Authorization

- Authorization is a process by which a server determines if the client has permission to use a resource or access a file.
- Authorization is usually coupled with authentication so that the server has some concept of who the client is that is requesting access.
- The type of authentication required for authorization may vary; passwords may be required in some cases but not in others.
- In some cases, there is no authorization; any user may be use a resource or access a file simply by asking for it.

(2) Authentication

- Authentication is used by a server when the server needs to know exactly who is accessing their information or site.
- In Authentication, the user or computer has to prove its identity to the server or client.
- Usually, Authentication by a server entails the use of a user name and password. Other ways to authenticate can be through cards, retina scans, voice recognition, and fingerprints.
- Authentication by a client usually involves the server giving a certificate to the client in which a trusted third party.

Authentication Vs Authorization

Authentication Vs Authorization

S.NO	AUTHENTICATION	AUTHORIZATION
1.	In authentication process, the **identity of users are checked** for providing the access to the system.	While in authorization process**, person's or user's authorities are checked** for accessing the resources.
2.	In authentication process, users or persons are **verified.**	While in this process, users or persons are **validated.**
3.	It is **done before the authorization** process.	While this process is **done after the authentication** process.
4.	It needs usually **user's login details**.	While it needs **user's privilege or security levels**.

(3) Security of the database Through Abstraction

- Data encryption enables to encrypt sensitive data, such as credit card numbers, stored in table columns.
- Encrypted data is decrypted for a database user who has access to the data.
- Data encryption helps protect data stored on media in the event that the storage media or data file gets stolen.

(3) Security of the database Through Abstraction

Encryption:

- Data encryption is a key-based access control system. Even if the encrypted data is retrieved, it cannot be understood until authorized decryption occurs.
- When a table contains encrypted columns, a single key is used regardless of the number of encrypted columns. This key is called the column encryption key.
- The column encryption keys for all tables, containing encrypted columns, are encrypted with the database server master encryption key and stored in a dictionary table in the database.
- The master encryption key is stored in an external security module that is outside the database and accessible only to the security administrator.

(3) Security of the database Through Abstraction

Encryp

(4) Security of the database through FIREWALLS

- A FIREWALL is dedicated software on another computer which inspects network traffic passing through it.
- It denies (or) permits passage based on set of rules.
- Database Firewalls are a type of Web Application Firewalls that monitor databases to identify and protect against database specific attacks that mostly seek to access sensitive information stored in the databases.

(4) Security of the database through FIREWALLS

ACCESS CONTROL

ACCESS CONTROL

What is Access Control in Database Security?

- Database access control is a method of allowing access to company's sensitive data only to those people (database users) who are allowed to access such data and to restrict access to unauthorized persons.
- It includes two main components: Authentication and Authorization.
- Note that Authentication & Authorization are not enough to protect data.
- An additional layer of security is required.
- Without authentication and authorization, there is no data security.

TYPES OF ACCESS CONTROL

Access Control Models include –

1. Discretionary Access Control (DAC)
2. Mandatory Access Control (MAC).
3. Role Based Access Control (RBAC) - is the most common method today.
4. Attribute Based Access Control (ABAC).

TYPES OF ACCESS CONTROL

(1) Discretionary Access Control (DAC):

- DAC models, the Data Owner Allows Access.
- DAC is a means of assigning access rights based on user-specified rules.
- DAC is the least restrictive compared to the other systems

TYPES OF ACCESS CONTROL

(2) Mandatory Access Control (MAC)

- MAC was developed using a nondiscretionary model, in which people are granted access based on an information clearance.
- MAC is a policy in which access rights are assigned based on central authority regulations.

TYPES OF ACCESS CONTROL

(3) Role Based Access Control (RBAC)

- RBAC grants access based on a user's role and implements key security principles such as "least privilege" and "separation of privilege".
- Thus, someone attempting to access information can only access data necessary for their role.

TYPES OF ACCESS CONTROL

(4) Attribute Based Access Control (ABAC)

- In ABAC, each resource and user are assigned a series of attributes.
- In this dynamic method, a comparative assessment of the user's attributes, including time of day, position and location, are used to make a decision on access to a resource.

SQL INJECTION

SQL INJECTION

- SQL Injection (SQLi) is a type of an attack that makes it possible to execute malicious SQL statements.
- These SQL statements control a database server behind a web application.
- Attackers can use SQL Injection vulnerabilities to bypass application security measures.
- They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database.
- They can also use SQL Injection to add, modify, and delete records in the database.

SQL INJECTION – ATTACK INTENSION

1. Determining database schema
2. Extracting data
3. Adding or modifying data
4. Bypassing authentication

SQL INJECTION - INTENSION

HOW SQL INJECTION ATTACK WORKS?

1. The types of attacks that can be performed using SQL injection vary depending on the type of database engine.
2. The attack works on dynamic SQL statements.
3. A dynamic statement is a statement that is generated at run time using parameters password from a web form or URI query string.

HOW SQL INJECTION ATTACK WORKS?

Simple Web Application with LOGIN FORM:

```
<form action=‘index.php’ method="post">

<input type="email" name="email" required="required"/>

<input type="password" name="password"/>

 <input type="checkbox" name="remember_me" value="Remember me"/>

<input type="submit" value="Submit"/> </form>
```

HOW SQL INJECTION ATTACK WORKS?

Simple Web Application with LOGIN FORM:

HOW SQL INJECTION ATTACK WORKS?

The statement at the backend for checking user ID is as follows

```
SELECT * FROM users WHERE email = $_POST['email'] AND password = md5($_POST['password']);
```

HERE,

1. The above statement uses the values of the $_POST[] array directly without sanitizing them.
2. The password is encrypted using MD5 algorithm.

HOW SQL INJECTION ATTACK WORKS?

```
CREATE TABLE `users` (
 `id` INT NOT NULL AUTO_INCREMENT,
 `email` VARCHAR(45) NULL,
 `password` VARCHAR(45) NULL,
 PRIMARY KEY (`id`));


insert into users (email,password) values ('m@m.com',md5('abc'));
```

STEP 1

STEP 2

Build Schema | Edit Fullscreen | Browser | [;]

```
select * from users
```

STEP 3

STEP 4

Run SQL | Edit Fullscreen | Format Code | [;]

ID	EMAIL	PASSWORD
1	m@m.com	900150983cd24fb0d6963f7d28e17f72

HOW SQL INJECTION ATTACK WORKS?

Suppose user supplies admin@admin.sys and 1234 as the password. The statement to be executed against the database would be

```
SELECT * FROM users WHERE email = 'admin@admin.sys' AND password = md5('1234’);
```

An Attacker generated dynamic statement will be as follows.

```
SELECT * FROM users WHERE email = 'xxx@xxx.xxx' OR 1 = 1 LIMIT 1 -- ' ] AND password = md5('1234’);
```

HERE,

- xxx@xxx.xxx ends with a single quote which completes the string quote.
- **“OR 1 = 1 LIMIT 1”** is a condition that will always be true and limits the returned results to only one record.
- **“-- ' AND”** … is a SQL comment that eliminates the password part.

HOW SQL INJECTION ATTACK WORKS?

```
1 SELECT * FROM users WHERE email = 'xxx@xxx.xxx'
2 OR 1 = 1 LIMIT 1 -- ' ] AND password = md5('1234');
```

The text in brown color means it is a comment

Run SQL ▶ | Edit Fullscreen | Format Code | [;]

Our statement returned a record

ID	EMAIL	PASSWORD
1	m@m.com	900150983cd24fb0d6963f7d28e17f72

TYPES OF SQL INJECTION ATTACK

1. In-band SQLi
   a) Error-based SQLi
   b) Union-based SQLi
2. Blind SQLi
   a) Boolean
   b) Time-based
3. Out-of-band SQLi

How to Prevent SQL Injections

Step 1: Train and maintain awareness

Step 2: Don't trust any user input

Step 3: Use whitelists, not blacklists

Step 4: Adopt the latest technologies

Step 5: Employ verified mechanisms

Step 6: Scan regularly

OODB
Object Oriented Databases

DATA MODELS

- Hierarchical model
- Network Model
- Relational Model
- Object Model

OODB

- Object Oriented Database Systems are alternative to Relational Database.
- In Object Oriented Database, information is represented in the form of Objects.
- If we can combine the features of Relational Model and Object Oriented Databases, the resultant model is called as Object Oriented Database Model.

OODB

Inheritance
Polymorphism
Encapsulation
Object-Oriented Programming
+
Relational Databases
Integrity
Query Processing
Concurrency
(Object-Oriented database is product of OOP and RDB)
Object-Oriented database

OODB

- An Object-Oriented Database System should satisfy two criteria:
  - ✓ It should be a DBMS, and it should be an Object-Oriented System.
- OODB implements OO concepts such as Object Identity, Polymorphism, Encapsulation and Inheritance.
- Object Oriented Databases are designed to work well with Object Oriented Programming languages such as Python, Java, Objective-C.

OODB Architecture

OODB architecture can be divided into two categories:

1. **Standalone OODB** where all of the data available is stored in object data model. Thus there is no overhead in mapping objects to application objects.
2. **Object Relational DBMS:** OODB acts as a staging layer for existing data in relational database. The data in relational database are mapped to object models and stored in object data database.

Object Definition Language (ODL)

Object Definition Language (ODL)

- ODL is the standardized language for defining the structure of database with respect to the object data model.
- ODL creates a layer of abstraction making data language and database independent to allow applications to move between databases or different language implementations.
- ODL defines three components of the object oriented data model:
    1. Abstraction
    2. Inheritance
    3. Encapsulation

Object Definition Language (ODL)

(1) Data Abstraction

- In OODB, Abstract Data Model is implemented as a GRAPH.
  - ✓ Vertices representing the Objects
  - ✓ Edges representing Relations.
- **Object Instance**: An Entity in an object model is called an Object Instance.
- **Object Identification:** Every Object Instance has a Unique Identity. This id used to reference object instances.

Object Definition Language (ODL)

(1) Data Abstraction

- **Object Class:** Similar Object Instances are grouped together into a class.
- **Object Reference Or Relationships:** The Object Model directly supports references. Object Instances (Entity) refer other using Object Identities.

Object Definition Language (ODL)

(2) Data Encapsulation:

- Encapsulation in the Object Model allows the Object Instances defined by the class.
- This allows Code and Data to be Packaged Together.
- This includes an Interface and an Implementation for each object.

Object Definition Language (ODL)

(3) Data Inheritance:

- Derived directly from Object Oriented Programming Languages.
- Object Data Models also allow for Inheritance, Polymorphism and Overriding.

Object Query Language (OQL)

- Object Query Language allows SQL-like queries to be performed on a OODB.
- Like-SQL, it is a Declarative Language.
- Query Structures look very similar in SQL and OQL but the results returned are different.

Object Query Language (OQL)

- Example: OQL query to obtain Voter Names who are from the state of kerala.
  - Select distinct Voter_Name from voters where State = “AP”

Voter_ID	Voter_Name	State
445620551	Siva Sai Kiran	TS
415320529	Mani Deep	AP
545620515	Abhiram N	AP
414420531	Ajay Ram	TN

Result from SQL Query

Voter_Name
Mani Deep
Abhiram N

Result from OQL Query

String
Mani Deep

String
Abhiram N

Object Vs Relational Model

Object Vs Relational Model

Key	RDBMS	OODBMS
Definition	RDBMS stands for Relational Database Management System.	OODBMS stands for Object Oriented Database Management System.
Data Management	Data is stored as entities defined in tabular format.	Data is stored as objects.
Data Complexity	RDBMS handles simple data.	OODBMS handles large and complex data.
Term	An entity refers to collection of similar items having same definition.	An class refers to group of objects having common relationships, behaviors and properties.
Data Handling	RDBMS handles only data.	OODBMS handles both data and functions operating on that data.
Objective	To keep data independent from application program.	To implement data encapsulation.
Key	A primary key identifies in object in a table uniquely.	Object Id, OID represents an object uniquely in group of objects.

Advantages of OODB

1. Can handle large collections of complex data including user defined data types.
2. Expressive data relationships
3. Version control for evolving classes and projects
4. Efficiently handles many-to-many relationships

Persistent Programming Language

Persistent Programming Language

- Programming languages that allow objects to continue existing after the program has been closed down are called persistent programming languages.
- A persistent programming language is extended with constructs to handle persistent data.
- In a persistent programming language:
  - ✓ *The query language is fully integrated with the host language and both share the same type system.*
  - ✓ *Any format changes required between the host language and the database are carried out transparently.*

Persistent Programming Language

- PPL allow an object to be created and stored in a database.
- PPL allows data to be manipulated directly from the programming language.
- No need to go through SQL.
- No need for explicit format changes.
- PPL allow objects to be manipulated in memory.
- No need to explicitly load from or store to the database.

Persistent Programming Language

Drawbacks:

1. It is easy to make programming errors that damage the database.
2. It is harder to do automatic high-level optimization.
3. They do not support declarative querying well.

Data Warehousing

Data Warehousing

- A Data Warehousing (DW) is the process for collecting and managing data from various sources to provide meaningful business insights.
- A Data warehouse is typically used to connect and analyze business data from heterogeneous sources.

How Datawarehouse Works?

- A Data Warehouse works as a central repository where information arrives from one or more data sources.
- Data may be:
  1. Structured
  2. Semi-structured
  3. Unstructured data
- The data is processed, transformed, and ingested so that users can access the processed data in the Data Warehouse through Business Intelligence tools, SQL clients, and spreadsheets.

Types of Datawarehouse

Three main types of Data Warehouses are:

1) Enterprise Data Warehouse:

2) Operational Data Store

3) Data Mart

Types of Datawarehouse

1) Enterprise Data Warehouse:

- Enterprise Data Warehouse is a Centralized Warehouse, which provides Decision Support Service across the enterprise.
- It offers a Unified Approach to Organizing and Representing Data.
- It also provides the ability to Classify Data according to the subject and Give Access according to those divisions.

Types of Datawarehouse

2) Operational Data Store:

- Operational Data Store (ODS) required when neither Data warehouse nor OLTP systems support organizations reporting needs.
- In ODS, Data warehouse is refreshed in real time.
- Hence, it is widely preferred for routine activities like storing records of the Employees.

Types of Datawarehouse

(3) Data Mart:

- A Data Mart is a subset of the data warehouse.
- It Specially Designed for a particular line of business, such as Sales, Finance, Sales or Finance.
- In an Independent Data Mart, data can collect directly from sources.

Datawarehouse Architecture

- Data Warehouse architecture is based on a Relational Database Management System Server that functions as the central repository for informational data.
- In the data warehouse architecture, Operational Data and Processing are separate from data warehouse processing.
- A Data Warehouse adopts a Three-tier Architecture.
  1. Bottom tier
  2. Middle tier
  3. Top tier

Datawarehouse Architecture

Three-Tier Data Warehouse Architecture

Datawarehouse Architecture

- **Bottom Tier: R**epresents the Data Warehouse Database Server. Back-end tools and utilities perform the Extract, Clean, Load, and Refresh functions.
- **Middle Tier: L**ies the OLAP Server. The ROLAP maps the operations on multidimensional data to standard relational OLAP (MOLAP) model, which directly implements the multidimensional data and operations.
- **Top-Tier:** Represents the Front-end Client Layer. This layer holds the Query Tools and Reporting Tools, Analysis Tools and Data Mining Tools.

Datawarehouse Architecture

Advantages:

1. Data warehouse allows business users to quickly access critical data all in one place.
2. Data warehouse provides consistent information.
3. It is also supporting ad-hoc reporting and query.
4. Data Warehouse helps to integrate many sources of data to reduce stress on the production system.
5. Data warehouse helps to reduce total turnaround time for analysis and reporting.
6. Restructuring and Integration make it easier for the user to use for reporting and analysis.

Datawarehouse Architecture

Disadvantages:

1. Creation and Implementation of Data Warehouse is surely time confusing affair.
2. Difficult to make changes in data types, ranges, data source schema, indexes, and queries.
3. It is too complex for the average users.
4. Organizations need to spend lots of their resources for training and Implementation purpose.

DATA MINING

DATA MINING

- There is a huge amount of data available in the Information Industry. This data is of no use until it is converted into useful information.
- To analyze this huge amount of data and extract useful information from it.
- Extraction of information is not only the single process, it involves Data Cleaning, Data Integration, Data Transformation, Data Mining, Pattern Evaluation and Data Presentation.
- Once all these processes are over, we would be able to use this information in many applications such as Fraud detection, Market analysis, Science exploration, etc.

WHAT IS DATA MINING?

- "Data Mining" refers to the extraction of useful information from a bulk of data or data warehouses.

WHAT IS DATA MINING?

- Data Mining, the result of extraction process is not data!! Instead, the result of data mining is the patterns and knowledge that we gain at the end of the extraction process.
- In that sense, Data Mining is also known as Knowledge Discovery in Databases (KDD) or Knowledge Extraction.

WHO USES DATA MINING?

- Data Mining is used in almost all the places where a large amount of data is stored and processed.
- Example, Banks typically use ‘data mining’ to find out their prospective customers who could be interested in Credit Cards, Personal Loans or Insurances as well.
- Since banks have the Transaction Details and Detailed Profiles of their customers, they analyze all this data and try to find out patterns which help them predict that certain customers could be interested in personal loans etc.

PURPOSE OF DATA MINING?

- The Data Mining helps to Predict Hidden Patterns, Future Trends and Behaviors and allowing businesses to Take Decisions.
- Technically, data mining is the computational process of Analyzing Data from different perspective.
- Data Mining can be applied to any type of data e.g. Data Warehouses, Transactional Databases, Relational Databases, Multimedia Databases, Spatial Databases, Time-series Databases, World Wide Web.

Data Mining as a Whole Process

The whole process of Data Mining comprises of Three Main Phases:

1. Data Pre-processing – *Data Cleaning, Integration, Selection and Transformation.*
2. Data Extraction – *Occurrence of exact data mining*
3. Data Evaluation and Presentation – *Analyzing and presenting results*

Data Mining as a Whole Process

The whole process of Data Mining comprises of Three Main Phases:

(1) Data Pre-Processing

- Data Pre-Processing is a data mining technique which is used to transform the raw data in a useful and efficient format.
- Steps involved in Data Pre-Processing,
  1. Data Cleaning.
  2. Data Transformation
  3. Data Reduction.

(1) Data Pre-Processing

1. **Data Cleaning:**

The data can have many irrelevant and missing parts. To handle this part, data cleaning is done.

2. **Data Transformation:**

To transform the data in appropriate forms suitable for mining process.

3. **Data Reduction:**

While working with huge volume of data, analysis became harder.

In order to get rid of this, we uses data reduction technique.

It aims to increase the storage efficiency and reduce data storage and analysis costs.

(2) Data Extraction

- In this Phase, Mathematical Models are used to determine Data Patterns.
- Based on the Business Objectives, suitable Modeling Techniques should be selected for the Prepared Dataset.
- Create a scenario to test check the Quality and Validity of the model.
- Run the model on the prepared dataset.
- To make sure that model can Meet Data Mining Objectives.

(3) Data Evaluation & Presentation

Evaluation:

- In this phase, Patterns identified are Evaluated against the business objectives.
- Gaining Business Understanding is an iterative process.
- A Go or No-Go Decision is taken to move the model in the deployment phase.

(3) Data Evaluation & Presentation

Presentation:

- Ship your data mining discoveries to everyday business operations.
- The knowledge or information discovered during data mining process should be made easy to understand for non-technical stakeholders.
- A detailed deployment plan, for shipping, maintenance, and monitoring of data mining discoveries is created.
- This helps to improve the organization's business policy.

Applications of Data Mining

1) Communications.
2) Insurance.
3) Education.
4) Manufacturing.
5) Banking.
6) Retail.
7) Service Providers.
8) E-Commerce.
9) Super Markets.
10) Crime Investigation.
11) Bioinformatics'

Benefits of Data Mining

1) Helps companies to get Knowledge-Based Information.
2) Helps organizations to make the profitable adjustments in operation and production.
3) The data mining is a cost-effective and efficient solution.
4) Data mining helps with the decision-making process.
5) Facilitates automated prediction of trends and behaviors.

Disadvantages of Data Mining

1) There are chances of companies may sell useful information of their customers to other companies for money. For example, American Express has sold credit card purchases of their customers to the other companies.

2) Many data mining analytics software is difficult to operate and requires advance training to work on.

3) The selection of correct data mining tool is a very difficult task.

Operating systems-PCCS404
UNIT-1

UNIT – I INTRODUCTION

Introduction: Concept of Operating Systems, Generations of Operating systems, Types of Operating Systems, OS Services, System Calls, Structure of an OS - Layered, Monolithic, Microkernel Operating Systems, Concept of Virtual Machine. Case study on UNIX and WINDOWS Operating System.

Operating System Definition and Function

In the Computer System (comprises of Hardware and software), Hardware can only understand machine code (in the form of 0 and 1) which doesn't make any sense to a naive user.

We need a system which can act as an intermediary and manage all the processes and resources present in the system.

An **Operating System** can be defined as an **interface between user and hardware**. It is responsible for the execution of all the processes, Resource Allocation, CPU management, File Management and many other tasks.

The purpose of an operating system is to provide an environment in which a user can execute programs in convenient and efficient manner.

Structure of a Computer System

A Computer System consists of:

- Users (people who are using the computer)
- Application Programs (Compilers, Databases, Games, Video player, Browsers, etc.)
- System Programs (Shells, Editors, Compilers, etc.)
- Operating System (A special program which acts as an interface between user and hardware)
- Hardware (CPU, Disks, Memory, etc)

Scanned with CamScanner

What does an Operating system do?

1. Process Management
2. Process Synchronization
3. Memory Management
4. CPU Scheduling
5. File Management
6. Security

OS GENERATION:

❖ The operating systems are designed to run on any of a class of machines at a variety of sites with a variety of peripheral configurations. ❖ The Computer system must then be configured or generated for each specific computer site, a process sometimes known as **system generation SYSGEN**.

43

❖ The operating system is normally distributed on disk, on CD-ROM or DVD-ROM, or as an "ISO" image, which is a file in the format of a CD-ROM or DVD-ROM.

❖ To generate a system, the special program called SYSGEN program reads from a given file, or asks the operator of the system for information concerning the specific configuration of the hardware system.

❖ The following kinds of information must be determined.

o What CPU is to be used? o How will the boot disk be formatted? o How much memory is available? o What devices are available? o What operating-system options are desired, or what parameter values

Scanned with CamScanner

are to be used? ❖ A system administrator can use this information to modify a copy of the source code of the operating system. The operating system then is completely compiled.

- ❖ The system description can lead to the creation of tables and the selection of modules from a precompiled library. These modules are linked together to form the generated operating system
- ❖ It is also possible to construct a system that is completely table driven. All the code is always part of the system, and selection occurs at execution time, rather than at compile or link time.

Types of OS

There are many types of operating system exists in the current scenario:

Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows –

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows –

- Problem of reliability.
- Question of security and integrity of user programs and data.

Scanned with CamScanner

- Problem of data communication.

Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows –

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows –

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows –

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

Scanned with CamScanner

Operating systems-PCCS404
UNIT-1

Batch Operating System

In the era of 1970s, the Batch processing was very popular. The Jobs were executed in batches. People were used to have a single computer which was called mainframe.

In Batch operating system, access is given to more than one person; they submit their respective jobs to the system for the execution.

The system put all of the jobs in a queue on the basis of first come first serve and then executes the jobs one by one. The users collect their respective output when all the jobs get executed.

Disadvantages of Batch OS

1. Starvation

Batch processing suffers from starvation. If there are five jobs J1, J2, J3, J4, J4 and J5 present in the batch. If the execution time of J1 is very high then other four jobs will never be going to get executed or they will have to wait for a very high time. Hence the other processes get starved.

2. Not Interactive

Batch Processing is not suitable for the jobs which are dependent on the user's input. If a job requires the input of two numbers from the console then it will never be going to get it in the batch processing scenario since the user is not present at the time of execution.

Multiprogramming Operating System

Scanned with CamScanner

Multiprogramming is an extension to the batch processing where the CPU is kept always busy. Each process needs two types of system time: CPU time and IO time.

In multiprogramming environment, for the time a process does its I/O, The CPU can start the execution of other processes. Therefore, multiprogramming improves the efficiency of the system.

Multiprocessing Operating System

In Multiprocessing, Parallel computing is achieved. There are more than one processors present in the system which can execute more than one process at the same time. This will increase the throughput of the system.

Multi Processing

Real Time Operating System

Scanned with CamScanner

In Real Time systems, each job carries a certain deadline within which the Job is supposed to be completed, otherwise the huge loss will be there or even if the result is produced then it will be completely useless.

The Application of a Real Time system exists in the case of military applications, if you want to drop a missile then the missile is supposed to be dropped with certain precision.

Operating System - Services

An Operating System provides services to both the users and to the programs.

- It provides programs an environment to execute.
- It provides users the services to execute the programs in a convenient manner.

Following are a few common services provided by an operating system −

- Program execution
- I/O operations
- File System manipulation
- Communication
- Error Detection
- Resource Allocation
- Protection

Program execution

Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management −

- Loads a program into memory.
- Executes the program.
- Handles program's execution.
- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

I/O Operation

An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

An Operating System manages the communication between user and device drivers.

Scanned with CamScanner

- I/O operation means read or write operation with any file or any specific I/O device.
- Operating system provides the access to the required I/O device when required.

File system manipulation

A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long-term storage purpose. Examples of storage media include magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management −

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

Communication

In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, the operating system manages communications between all the processes. Multiple processes communicate with one another through communication lines in the network.

The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication −

- Two processes often require data to be transferred between them
- Both the processes can be on one computer or on different computers, but are connected through a computer network.
- Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

Error handling

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

- The OS constantly checks for possible errors.
- The OS takes an appropriate action to ensure correct and consistent computing.

Resource Management

Scanned with CamScanner

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

Protection

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities.

Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

- The OS ensures that all access to system resources is controlled.
- The OS ensures that external I/O devices are protected from invalid access attempts.
- The OS provides authentication features for each user by means of passwords.

SYSTEM CALLS:

❖ The system call provides an interface to the operating system services.

❖ Application developers often do not have direct access to the system calls, but can access them through an application programming interface (API). The functions that are included in the API invoke the actual system calls.

❖ Systems execute thousands of system calls per second. Application developers design programs according to an **application programming interface (API)**.

❖ For most programming languages, the Application Program Interface provides a **system call interface** that serves as the link to system calls made available by the operating system. ❖ The system-call interface intercepts function calls in the API and invokes the necessary system calls within the Operating system.

❖ **Example: System calls for writing a simple program to read data from one file and copy them to another file**

❖ The caller of the system call need know nothing about how the system call is implemented or what it does during execution.

❖ The caller need only obey the API and understand what the operating system will do as a result of The execution of that system call.

❖ Three general methods are used to pass parameters to the operating

Scanned with CamScanner

system o pass the parameters in registers o parameters are generally stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register o Parameters also can be placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system.

Types of System Calls:

❖ System calls can be grouped roughly into six major categories

1.Processcontrol,
2.File manipulation,
3.Device manipulation,
4.Information maintenance,
5.Communications,
6.Protection

1.Process Control

This system calls perform the task of process creation, process termination, etc.

Functions:

Scanned with CamScanner

- End and Abort
- Load and Execute
- Create Process and Terminate Process
- Wait and Signed Event
- Allocate and free memory

2.File Management or manipulation

File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.

Functions:

- Create a file
- Delete file
- Open and close file
- Read, write, and reposition
- Get and set file attributes

3.Device Management or manipulation

Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.

Functions

- Request and release device
- Logically attach/ detach devices
- Get and Set device attributes

4.Information Maintenance

It handles information and its transfer between the OS and the user program.

Functions:

- Get or set time and date
- Get process and device attributes

5.Communication:

These types of system calls are specially used for interprocess communications.

Scanned with CamScanner

Functions:

- Create, delete communications connections
- Send, receive message
- Help OS to transfer status information
- Attach or detach remote devices

OPERATING SYSTEM STRUCTURE:

❖ The operating systems are large and complex. A common approach is to partition the task into small components, or modules, rather than have one **monolithic** system.

❖ The structure of an operating system can be defined the following structures.

← Simple structure ← Layered approach ←
Microkernels ← Modules ← Hybrid systems

Simple structure:

❖ The Simple structured operating systems do not have a well defined structure. These systems will be simple, small and limited systems.

Example: MS-DOS.

❖ In MS-DOS, the interfaces and levels of functionality are not well separated. ❖ In MS-DOS application programs are able to access the basic I/O routines. This causes the entire systems to be crashed when user programs fail.

Example: Traditional UNIX OS

❖ It consists of two separable parts: the kernel and the system programs. ❖ The kernel is further separated into a series of interfaces and device driver

❖ The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.

❖ This monolithic structure was difficult to implement and maintain.

Layered approach:

❖ A system can be made modular in many ways. One method is the **layered approach**, in which the operating system is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.

❖ An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data.

❖ The main advantage of the layered approach is simplicity of construction and debugging. The layers are selected so that each uses functions (operations) and services of only lower-level layers.

❖ Each layer is implemented only with operations provided by lower-level layers. A layer does not need to know how these operations are implemented; it needs to know only what these operations do.

❖ The major difficulty with the layered approach involves appropriately defining the various layers

Scanned with CamScanner

because a layer can use only lower-level layers.

- A problem with layered implementations is that they tend to be less efficient than other types.

Microkernels:

- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the **microkernel** approach
- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs.
- Microkernel provide minimal process and memory management, in addition to a communication facility.
- The main function of the microkernel is to provide communication between the client program and the various services that are also running in user space.
- The client program and service never interact directly. Rather, they communicate indirectly by exchanging messages with the microkernel.
- One benefit of the microkernel approach is that it makes extending the operating system easier. All new services are added to user space and consequently do not require modification of the kernel.
- The performance of microkernel can suffer due to increased system-function overhead.

Modules:

- The best current methodology for operating-system design involves using **loadable kernel Modules**
- The kernel has a set of core components and links in additional services via modules, either at boot time or during run time.
- The kernel provides core services while other services are implemented dynamically, as the kernel is running
- Linking services dynamically is more comfortable than adding new features directly to the kernel, which would require recompiling the kernel every time a change was made.

Example: Solaris OS

- The Solaris operating system structure is organized around a core kernel with seven types of loadable kernel modules:

← Scheduling classes ← File systems ← Loadable system calls ← Executable formats ← STREAMS modules ← Miscellaneous ← Device and bus drivers

HYBRID SYSTEMS

- The Operating System combines different structures, resulting in hybrid systems that address

Scanned with CamScanner

performance, security, and usability issues.

- They are monolithic, because having the operating system in a single address space provides very efficient performance
- However, they are also modular, so that new functionality can be dynamically added to the kernel.
- Example: Linux and Solaris are monolithic (simple) and also modular, IOS.
- Apple IOS Structure

TOPICS LET:
1.Concept of virtual machine
2.case study on unix and windows os

2 MARK QUESTIONS

1.What is operating system?

A. An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is a vital component of the system software in a computer system

2.What is time sharing operating system?

A. Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

3.What is a system call?

A. A **system call** is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.

4.what is real-time operating system?

A. Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

5.List the services provided by the operating system?

A. Following are a few common services provided by an operating system −

- Program execution
- I/O operations
- File System manipulation
- Communication

Scanned with CamScanner

- Error Detection
- Resource Allocation
- Protection

6.what is virtual machine?

A. Virtual Machine abstracts the hardware of our personal computer such as CPU, disk drives, memory, NIC (Network Interface Card) etc, into many different execution environments as per our requirements, hence giving us a feel that each execution environment is a single computer. For example: VirtualBox.

7.what are the advantages of multi processing operating system?

- The biggest advantage of a multiprocessor system is that it helps you to get more work done in a shorter period.
- The code is usually straightforward.
- Takes advantage of multiple CPU & cores
- Helps you to avoid GIL limitations for CPython
- Remove synchronization primitives unless if you use shared memory.
- Child processes are mostly interruptible/killable
- It helps you to get work done in a shorter period.
- These types of systems should be used when very high speed is required to process a large volume of data.
- Multiprocessing systems save money compared to single processor systems as processors can share peripherals and power supplies.

Scanned with CamScanner

* Virtual Machines:

The fundamental idea behind a virtual machine is to abstract the hardware of a single computer into several different execution environments thereby creating the illusion that each ~~spe~~ seperate execution environment is running its own private computer.

By using CPU scheduling and virtual Memory techniques an, operating system can create the illusion that a process has its own processor with its own (virtual) Mfy.

Scanned with CamScanner

The virtual Machine approach does not provide any such additional functionality but rather provides an interface that is identical to the bare hardware. Each process is provided with a (virtual) copy of the computer.

Non virtual Machine

Virtual Machine.

There are several reasons for creating a virtual Machine, all of which are fundamentally related to being able to share the same h/w yet run several different execution environment.

Minidisks:-

The physical Machine cannot allocate a disk drive to each virtual machine because the virtual machine software itself will need substantial disk space to provide virtual m/y and spooling. Hence we need virtual disk which is known as minidisks. The sum of the sizes of all the minidisks must be smaller

Scanned with CamScanner

than the size of the physical disk space available. Users are given their own Virtual machines.

Benefits:

* Complete protection of the various s/m resources.
* Sharing of disk.
* Communication.

Examples:

* VMware
* The Java virtual Machine.

Scanned with CamScanner

* System Calls:-

System calls provides an interface to the services made available by an operating system. These are generally available as routines written in C & C++.

How system calls are used:-

To read data from one file and copy it in another file.

Source File → Destination File.

System Call Sequence

* Acquire i/p file name
  → write Prompt to screen
  → Accept i/p
* Acquire o/p file name
  → write prompt to screen
  → Accept i/p
* Open the i/p file
  → If file doesn't exist, abort
* Create output file
  if file exist; abort

Loop
  Read from i/p file
  Write to output file
Until read fails
Close the output file

* Write completion msg to screen
* Terminate Normally.

Scanned with CamScanner

The run-time support system for most programming languages provides a system-call interface that servers as the link to system calls made available by the operating system.

The system call interface intercepts function calls in the API and invokes the necessary system call within the operating system.

A number is associated with each system calls and the system call interface maintains a table indexed according to these numbers.

The system call interface invokes the intended s/m call in the OS kernel & returns the status of the s/m call and any return values.

fig:- Handling of a user application invoking the open() system call.

Scanned with CamScanner

System calls occur in different ways depending on the computer in use. Often more information is required than simply the identity of desired s/m call.

Passing parameters to the operating system:.

Three general Methods are there to pass parameters to the operating system.

1. To pass parameters in registers.

2. Parameters are stored in a block, table, memory and the address of the block is passed as a parameter in a register.

2. Parameters also can be placed or pushed onto the stack by the program and popped off the stack by the OS.

3. Some OS prefer the block / stack because those approaches do not limit the no or length of parameters being passed.

Types of system calls:.

* Process control
  - End, abort
  - Load, execute
  - Create process, terminate process
  - Get process attributes, set process attributes
  - Wait for time

Scanned with CamScanner

- wait event, Signal event
- Allocate and free memory.

* File Management.
  - Create file, delete file
  - Open, close
  - Read, write, reposition.
  - Get file attributes, set file attributes.

* Device Management
  - Request device, Release device.
  - read, write, reposition
  - Get device attributes, set device attributes
  - logically attach or detach devices.

* Information Maintenance
  - Get time or date, set time or date
  - Get sim data, set sim data.
  - get process, file or device attributes
  - set. " " "

* Communications.
  - Create, delete communication connection
  - send, receive msg
  - Transfer status info
  - Attach or detach remote devices.

Scanned with CamScanner

UNIT – II PROCESSES

Processes: Definition, Process Relationship, Different states of a Process, Process State transitions, Process Control Block (PCB), Context switching

Thread: Definition, Various states, Benefits of threads, Types of threads, Concept of multithreads,

Process Scheduling: Foundation and Scheduling objectives, Types of Schedulers, Scheduling criteria: CPU utilization, Throughput, Turnaround Time, Waiting Time, Response Time; Scheduling algorithms: Pre-emptive and Non pre-emptive, FCFS, SJ F, RR; Multiprocessor scheduling: Real Time scheduling: RM and EDF.

PROCESSES:

A Process is defined as a program in execution. A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a process. In order to accomplish its task, process needs the computer resources.

There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.

Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

The operating system is responsible for the following activities in connection with Process Management:

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.

Process States:

❖ As a process executes, it changes **state**. The state of a process is defined in part by the current activity of that process

A process may be in one of the following states:

New. The process is being created.

Running. Instructions are being executed.

Waiting. The process is waiting for some event to occur

Ready. The process is waiting to be assigned to a processor.

Terminated. The process has finished execution.

State Diagram

The process, from its creation to completion, passes through various states. The minimum number of states is five.

The names of the states are not standardized although the process may be in one of the following states during execution.

1. New

A program which is going to be picked up by the OS into the main memory is called a new process.

2. Ready

Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

3. Running

One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.

4. Block or wait

From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

5. Completion or termination

When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

6. Suspend ready

A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.

If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. The suspend ready processes remain in the secondary memory until the main memory gets available.

7. Suspend wait

Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory. Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and make room for the higher priority process. These processes complete their execution once the main memory gets available and their wait is finished.

PROCESSES
OPERATING SYSTEMS -UNIT II

Process Control Block:

❖ Each process is represented in the operating system by a **process control block (PCB)**—also called a **task control block**.
❖ It contains many pieces of information associated with a specific process

❖ **Process state**. The state may be new, ready, running, waiting, halted, and so on.
❖ **Program counter**. The counter indicates the address of the next instruction to be executed for this process.
❖ **CPU registers**. The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers

- **CPU-scheduling information.** This information includes a process priority, pointers to scheduling queues
- **Memory-management information.** This information may include such items as the value of the base and limit registers and the page tables
- **Accounting information.** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers,
- **I/O status information.** This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

Various Times related to the Process

1. Arrival Time

The time at which the process enters into the ready queue is called the arrival time.

2. Burst Time

The total amount of time required by the CPU to execute the whole process is called the Burst Time. This does not include the waiting time. It is confusing to calculate the execution time for a process even before executing it hence the scheduling problems based on the burst time cannot be implemented in reality.

3. Completion Time

The Time at which the process enters into the completion state or the time at which the process completes its execution, is called completion time.

4. Turnaround time

The total amount of time spent by the process from its arrival to its completion, is called Turnaround time.

5. Waiting Time

The Total amount of time for which the process waits for the CPU to be assigned is called waiting time.

6. Response Time

The difference between the arrival time and the time at which the process first gets the CPU is called Response Time.

CONTEXT SWITCHING

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.

Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use.

- Program Counter
- Scheduling information
- Base and limit register value

- Currently used register
- Changed State
- I/O State information
- Accounting information

THREADS

What is Thread?

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.

Single Process P with single thread

Single Process P with three threads

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread

Threads are implemented in following two ways –

- **User Level Threads** – User managed threads.
- **Kernel Level Threads** – Operating System managed threads acting on kernel, an operating system core.

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between

threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can

be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Multithreading Models

Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

Many to Many Model

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.

One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.

Operating System - Process Scheduling

Definition

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.

The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Two-State Process Model

Two-state process model refers to running and non-running states which are described below −

S.N.	State & Description
1	**Running** When a new process is created, it enters into the system as in the running state.
2	**Not Running** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary

storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

CPU Scheduling Criteria

A CPU scheduling algorithm tries to maximize and minimize the following:

Maximize:

CPU utilization: CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible. It can range from 0 to 100 percent. However, for the RTOS, it can be range from 40 percent for low-level and 90 percent for the high-level system.

Throughput: The number of processes that finish their execution per unit time is known Throughput. So, when the CPU is busy executing the process, at that time, work is being done, and the work completed per unit time is called Throughput.

Minimize:

Waiting time: Waiting time is an amount that specific process needs to wait in the ready queue.

Response time: It is an amount to time in which the request was submitted until the first response is produced.

Turnaround Time: Turnaround time is an amount of time to execute a specific process. It is the calculation of the total time spent waiting to get into the memory, waiting in the queue and, executing on the CPU. The period between the time of process submission to the completion time is the turnaround time.

Interval Timer

Timer interruption is a method that is closely related to preemption. When a certain process gets the CPU allocation, a timer may be set to a specified interval. Both timer interruption and preemption force a process to return the CPU before its CPU burst is complete.

Most of the multi-programmed operating system uses some form of a timer to prevent a process from tying up the system forever.

What is Dispatcher?

It is a module that provides control of the CPU to the process. The Dispatcher should be fast so that it can run on every context switch. Dispatch latency is the amount of time needed by the CPU scheduler to stop one process and start another.

Functions performed by Dispatcher:

- Context Switching
- Switching to user mode
- Moving to the correct location in the newly loaded program.

Types of CPU Scheduling

Here are two kinds of Scheduling methods:

Preemptive Scheduling

In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

Non-Preemptive Scheduling

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That's because it doesn't need special hardware (for example, a timer) like preemptive scheduling.

When scheduling is Preemptive or Non-Preemptive?

To determine if scheduling is preemptive or non-preemptive, consider these four parameters:

1. A process switches from the running to the waiting state.
2. Specific process switches from the running state to the ready state.
3. Specific process switches from the waiting state to the ready state.
4. Process finished its execution and terminated.

Only conditions 1 and 4 apply, the scheduling is called non- preemptive.

All other scheduling are preemptive.

Important CPU scheduling Terminologies

- **Burst Time/Execution Time:** It is a time required by the process to complete execution. It is also called running time.
- **Arrival Time:** when a process enters in a ready state
- **Finish Time:** when process complete and exit from a system
- **Multiprogramming:** A number of programs which can be present in memory at the same time.
- **Jobs:** It is a type of program without any kind of user interaction.
- **User:** It is a kind of program having user interaction.
- **Process:** It is the reference that is used for both job and user.
- **CPU/IO burst cycle:** Characterizes process execution, which alternates between CPU and I/O activity. CPU times are usually shorter than the time of I/O.

Types of CPU scheduling Algorithm

There are mainly six types of process scheduling algorithms

1. First Come First Serve (FCFS)
2. Round Robin Scheduling
3. Shortest-Job-First (SJF) Scheduling
4. Shortest Remaining Time
5. Priority Scheduling
6. Multilevel Queue Scheduling

Scheduling Algorithms

here are various algorithms which are used by the Operating System to schedule the processes on the processor in an efficient way.

There are the following algorithms which can be used to schedule the jobs.

1. First Come First Serve

It is the simplest algorithm to implement. The process with the minimal arrival time will get the CPU first. The lesser the arrival time, the sooner will the process gets the CPU. It is the non-preemptive type of scheduling.

2. Round Robin

In the Round Robin scheduling algorithm, the OS defines a time quantum (slice). All the processes will get executed in the cyclic way. Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a preemptive type of scheduling.

3. Shortest Job First

The job with the shortest burst time will get the CPU first. The lesser the burst time, the sooner will the process get the CPU. It is the non-preemptive type of scheduling.

4. Shortest remaining time first

It is the preemptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution.

5. Priority based scheduling

In this algorithm, the priority will be assigned to each of the processes. The higher the priority, the sooner will the process get the CPU. If the priority of the two processes is same then they will be scheduled according to their arrival time.

6. Highest Response Ratio Next or multi level queue scheduling

In this scheduling Algorithm, the process with highest response ratio will be scheduled next. This reduces the starvation in the system.

FCFS Scheduling

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

- Simple
- Easy
- First come, First serv

Disadvantages of FCFS

1. The scheduling method is non preemptive, the process will run to the completion.
2. Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.

3. Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

Example

Let's take an example of The FCFS scheduling algorithm. In the Following schedule, there are 5 processes with process ID **P0, P1, P2, P3 and P4**. P0 arrives at time 0, P1 at time 1, P2 at time 2, P3 arrives at time 3 and Process P4 arrives at time 4 in the ready queue. The processes and their respective Arrival and Burst time are given in the following table.

The Turnaround time and the waiting time are calculated by using the following formula.

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turnaround time - Burst Time

The average waiting Time is determined by summing the respective waiting time of all the processes and divided the sum by the total number of processes.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Wai
0	0	2	2	2	0
1	1	6	8	7	1
2	2	4	12	8	4
3	3	9	21	18	9
4	4	12	33	29	17

Avg Waiting Time=31/5

(Gantt chart)

Shortest Job First (SJF) Scheduling

Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, SJF scheduling algorithm, schedules the processes according to their burst time.

In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Advantages of SJF

1. Maximum throughput
2. Minimum average waiting and turnaround time

Disadvantages of SJF

1. May suffer with the problem of starvation
2. It is not implementable because the exact Burst time for a process can't be known in advance.

There are different techniques available by which, the CPU burst time of the process can be determined. We will discuss them later in detail.

Example

In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

PID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting T
1	1	7	8	7	0
2	3	3	13	10	7
3	6	2	10	4	2
4	7	10	31	24	14
5	9	8	21	12	4

Since, No Process arrives at time 0 hence; there will be an empty slot in the **Gantt chart** from time 0 to 1 (the time at which the first process arrives).

According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue.

Till now, we have only one process in the ready queue hence the scheduler will schedule this to the processor no matter what is its burst time.

This will be executed till 8 units of time. Till then we have three more processes arrived in the ready queue hence the scheduler will choose the process with the lowest burst time.

Among the processes given in the table, P3 will be executed next since it is having the lowest burst time among all the available processes.

So that's how the procedure will go on in **shortest job first (SJF)** scheduling algorithm.

Avg Waiting Time = 27/5

Round Robin Scheduling Algorithm

Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems. This is the **preemptive version** of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a **cyclic way**. A certain time slice is defined in the system which is called time **quantum**. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution

Advantages

1. It can be actually implementable in the system because it is not depending on the burst time.
2. It doesn't suffer from the problem of starvation or convoy effect.
3. All the jobs get a fare allocation of CPU.

. Disadvantages

1. The higher the time quantum, the higher the response time in the system.
2. The lower the time quantum, the higher the context switching overhead in the system.
3. Deciding a perfect time quantum is really a very difficult task in the system.

RR Scheduling Example

In the following example, there are six processes named as P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table. The time quantum of the system is 4 units.

Process ID	Arrival Time	Burst Time
1	0	5
2	1	6
3	2	3
4	3	1
5	4	5
6	6	4

According to the algorithm, we have to maintain the ready queue and the Gantt chart. The structure of both the data structures will be changed after every scheduling.

Ready Queue:

Initially, at time 0, process P1 arrives which will be scheduled for the time slice 4 units. Hence in the ready queue, there will be only one process P1 at starting with CPU burst time 5 units.

P1
5

GANTT chart

The P1 will be executed for 4 units first.

Ready Queue

Meanwhile the execution of P1, four more processes P2, P3, P4 and P5 arrives in the ready queue. P1 has not completed yet, it needs another 1 unit of time hence it will also be added back to the ready queue.

P2	P3	P4	P5	P1
6	3	1	5	1

GANTT chart

After P1, P2 will be executed for 4 units of time which is shown in the Gantt chart.

Ready Queue

During the execution of P2, one more process P6 is arrived in the ready queue. Since P2 has not completed yet hence, P2 will also be added back to the ready queue with the remaining burst time 2 units.

P3	P4	P5	P1	P6	P2
3	1	5	1	4	2

GANTT chart

After P1 and P2, P3 will get executed for 3 units of time since its CPU burst time is only 3 seconds.

Ready Queue

Since P3 has been completed, hence it will be terminated and not be added to the ready queue. The next process will be executed is P4.

P4	P5	P1	P6	P2
1	5	1	4	2

GANTT chart

After, P1, P2 and P3, P4 will get executed. Its burst time is only 1 unit which is lesser then the time quantum hence it will be completed.

Ready Queue

The next process in the ready queue is P5 with 5 units of burst time. Since P4 is completed hence it will not be added back to the queue.

P5	P1	P6	P2
5	1	4	2

GANTT chart

P5 will be executed for the whole time slice because it requires 5 units of burst time which is higher than the time slice.

Ready Queue

P5 has not been completed yet; it will be added back to the queue with the remaining burst time of 1 unit.

P1	P6	P2	P5
1	4	2	1

GANTT Chart

The process P1 will be given the next turn to complete its execution. Since it only requires 1 unit of burst time hence it will be completed.

Ready Queue

P1 is completed and will not be added back to the ready queue. The next process P6 requires only 4 units of burst time and it will be executed next.

P6	P2	P5
4	2	1

GANTT chart

P6 will be executed for 4 units of time till completion.

Ready Queue

Since P6 is completed, hence it will not be added again to the queue. There are only two processes present in the ready queue. The Next process P2 requires only 2 units of time.

P2	P5
2	1

GANTT Chart

P2 will get executed again, since it only requires only 2 units of time hence this will be completed.

P1	P2	P3	P4	P5	P1	P6	P2

0 4 8 11 12 16 17 21 23

Ready Queue

Now, the only available process in the queue is P5 which requires 1 unit of burst time. Since the time slice is of 4 units hence it will be completed in the next burst.

P5
1

GANTT chart

P5 will get executed till completion.

P1	P2	P3	P4	P5	P1	P6	P2	P5

0 4 8 11 12 16 17 21 23 24

The completion time, Turnaround time and waiting time will be calculated as shown in the table below.

As, we know,

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

rocess ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time

1	0	5	17	17	12
2	1	6	23	22	16
3	2	3	11	9	6
4	3	1	12	9	8
5	4	5	24	20	15
6	6	4	21	15	11

Avg Waiting Time = (12+16+6+8+15+11)/6 = 76/6 units

Highest Response Ratio Next (HRRN) Scheduling

Highest Response Ratio Next (HRNN) is one of the most optimal scheduling algorithms. This is a non-preemptive algorithm in which, the scheduling is done on the basis of an extra parameter called Response Ratio. A Response Ratio is calculated for each of the available jobs and the Job with the highest response ratio is given priority over the others.

Response Ratio is calculated by the given formula.

Response Ratio = (W+S)/S

Where,

W → Waiting Time
S → Service Time or Burst Time

If we look at the formula, we will notice that the job with the shorter burst time will be given priority but it is also including an extra factor called waiting time. Since,

HRNN W
HRNN (1/S)

Hence,

This algorithm not only favors shorter job but it also concern the waiting time of the longer jobs.

Its mode is non preemptive hence context switching is minimal in this algorithm.

HRNN Example

In the following example, there are 5 processes given. Their arrival time and Burst Time are given in the table.

Process ID	Arrival Time	Burst Time
0	0	3
1	2	5
2	4	4
3	6	1
4	8	2

At time 0, The Process P0 arrives with the CPU burst time of 3 units. Since it is the only process arrived till now hence this will get sche
immediately.

P0 is executed for 3 units, meanwhile, only one process P1 arrives at time 3. This will get scheduled immediately since the OS doesn't h

P1 is executed for 5 units. Meanwhile, all the processes get available. We have to calculate the Response Ratio for all the remaining jobs.

RR (P2) = ((8-4) +4)/4 = 2
RR (P3) = (2+1)/1 = 3
RR (P4) = (0+2)/2 = 1

Since, the Response ratio of P3 is higher hence P3 will be scheduled first.

P3 is scheduled for 1 unit. The next available processes are P2 and P4. Let's calculate their Response ratio.

RR (P2) = (5+4)/4 = 2.25
RR (P4) = (1+2)/2 = 1.5

The response ratio of P2 is higher hence P2 will be scheduled.

Now, the only available process is P4 with the burst time of 2 units, since there is no other process available hence this will be scheduled.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting T
0	0	3	3	3	0
1	2	5	8	6	1
2	4	4	13	9	5
3	6	1	9	3	2

4	8	2	15	7	5

Average Waiting Time = 13/5

TOPICS LEFT IN UNIT-II

1. Real Time scheduling:
2. RM and
3. EDF.

ALL THE BEST

OPERATING SYSTEMS -U18PCCS404

UNIT-III

UNIT – III IPC AND DEADLOCK **9**

Inter-process Communication: Critical Section, Race Conditions, Mutual Exclusion, Hardware Solution, Strict Alternation, Peterson's Solution, The Producer Consumer Problem, Semaphores, Event Counters, Monitors, Message Passing, Classical IPC Problems: Reader's & Writer Problem, Dinning Philosopher Problem etc.

Deadlocks: Definition, Necessary and sufficient conditions for Deadlock, Deadlock Prevention, Deadlock Avoidance: Banker's algorithm, Deadlock detection and Recovery.

What is Inter Process Communication?

Inter process communication (IPC) is used for exchanging data between multiple threads in one or more processes or programs. The Processes may be running on single or multiple computers connected by a network. The full form of IPC is Inter-process communication.

It is a set of programming interface which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system. This allows a specific program to handle many user requests at the same time.

Since every single user request may result in multiple processes running in the operating system, the process may require to communicate with each other. Each IPC protocol approach has its own advantage and limitation, so it is not unusual for a single program to use all of the IPC methods

The Critical Section Problem

Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device.

The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.

Requirements of Synchronization mechanisms

Primary

1. **Mutual Exclusion**

   Our solution must provide mutual exclusion. By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section.

2. **Progress**

   Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

Secondary

1. **Bounded Waiting**

   We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical section.

2. **Architectural Neutrality**

   Our mechanism must be architectural natural. It means that if our solution is working fine on one architecture then it should also run on the other ones as well.

Strict Alternation Approach

Turn Variable or Strict Alternation Approach is the software mechanism implemented at user mode. It is a busy waiting solution which can be implemented only for two processes. In this approach, A turn variable is used which is actually a lock.

This approach can only be used for only two processes. In general, let the two processes be Pi and Pj. They share a variable called turn variable. The pseudo code of the program can be given as following.

For Process Pi

```
Non - CS
while (turn ! = i);
Critical Section
turn = j;
Non - CS
```

For Process Pj

```
Non - CS
while (turn ! = j);
Critical Section
turn = i ;
Non - CS
```

The actual problem of the lock variable approach was the fact that the process was entering in the critical section only when the lock variable is 1. More than one process could see the lock variable as 1 at the same time hence the mutual exclusion was not guaranteed there.

This problem is addressed in the turn variable approach. Now, A process can enter in the critical section only in the case when the value of the turn variable equal to the PID of the process.

There are only two values possible for turn variable, i or j. if its value is not i then it will definitely be j or vice versa.

In the entry section, in general, the process Pi will not enter in the critical section until its value is j or the process Pj will not enter in the critical section until its value is i.

Initially, two processes Pi and Pj are available and want to execute into critical section.

The turn variable is equal to i hence Pi will get the chance to enter into the critical section. The value of Pi remains I until Pi finishes critical section.

Pi finishes its critical section and assigns j to turn variable. Pj will get the chance to enter into the critical section. The value of turn remains j until Pj finishes its critical section.

Analysis of Strict Alternation approach

Let's analyze Strict Alternation approach on the basis of four requirements.

Mutual Exclusion

The strict alternation approach provides mutual exclusion in every case. This procedure works only for two processes. The pseudo code is different for both of the processes. The process will only enter when it sees that the turn variable is equal to its Process ID otherwise not Hence No process can enter in the critical section regardless of its turn.

Progress

Progress is not guaranteed in this mechanism. If Pi doesn't want to get enter into the critical section on its turn then Pj got blocked for infinite time. Pj has to wait for so long for its turn since the turn variable will remain 0 until Pi assigns it to j.

Portability

The solution provides portability. It is a pure software mechanism implemented at user mode and doesn't need any special instruction from the Operating System.

Mutual Exclusion	✓
Progress	✗
Bounded Waiting	✓
Portability	✓

Paterson Solution

This is a software mechanism implemented at user mode. It is a busy waiting solution can be implemented for only two processes. It uses two variables that are turn variable and interested variable.

The Code of the solution is given below

```
# define N 2
# define TRUE 1
# define FALSE 0
int interested[N] = FALSE;
int turn;
voidEntry_Section (int process)
{
   int other;
   other = 1-process;
   interested[process] = TRUE;
   turn = process;
   while (interested [other] =True && TURN=process);
}
voidExit_Section (int process)
{
   interested [process] = FALSE;
}
```

Till now, each of our solution is affected by one or the other problem. However, the Peterson solution provides you all the necessary requirements such as Mutual Exclusion, Progress, Bounded Waiting and Portability.

Analysis of Peterson Solution

```
voidEntry_Section (int process)
{
  1. int other;
  2. other = 1-process;
  3. interested[process] = TRUE;
  4. turn = process;
  5. while (interested [other] =True && TURN=process);
}

Critical Section

voidExit_Section (int process)
{
  6. interested [process] = FALSE;
}
```

This is a two process solution. Let us consider two cooperative processes P1 and P2. The entry section and exit section are shown below. Initially, the value of interested variables and turn variable is 0.

Initially process P1 arrives and wants to enter into the critical section. It sets its interested variable to True (instruction line 3) and also sets turn to 1 (line number 4). Since the condition given in line number 5 is completely satisfied by P1 therefore it will enter in the critical section.

P1 → 1 2 3 4 5 CS

Meanwhile, Process P1 got preempted and process P2 got scheduled. P2 also wants to enter in the critical section and executes instructions 1, 2, 3 and 4 of entry section. On instruction 5, it got stuck since it doesn't satisfy the condition (value of other interested variable is still true). Therefore it gets into the busy waiting.

P2 → 1 2 3 4 5

P1 again got scheduled and finish the critical section by executing the instruction no. 6 (setting interested variable to false). Now if P2 checks then it are going to satisfy the condition since other process's interested variable becomes false. P2 will also get enter the critical section.

P1 → 6
P2 → 5 CS

Any of the process may enter in the critical section for multiple numbers of times. Hence the procedure occurs in the cyclic order.

Mutual Exclusion

The method provides mutual exclusion for sure. In entry section, the while condition involves the criteria for two variables therefore a process cannot enter in the critical section until the other process is interested and the process is the last one to update turn variable.

Progress

An uninterested process will never stop the other interested process from entering in the critical section. If the other process is also interested then the process will wait.

Bounded waiting

The interested variable mechanism failed because it was not providing bounded waiting. However, in Peterson solution, A deadlock can never happen because the process which first sets the turn variable will enter in the critical section for sure. Therefore, if a process is preempted after executing line number 4 of the entry section then it will definitely get into the critical section in its next chance.

Portability

This is the complete software solution and therefore it is portable on every hardware.

Mutual Exclusion	✓
Progress	✓
Bounded Waiting	✓
Portability	✓

Mutual exclusion

A **mutual exclusion** (mutex) is a program object that prevents simultaneous access to a shared resource. This concept is used in concurrent programming with a critical section, a piece of code in which processes or threads access a shared resource.

Race Condition

A race condition is a situation that may occur inside a critical section. This happens when the result of multiple thread execution in critical section differs according to the order in which the threads execute.

Race conditions in critical sections can be avoided if the critical section is treated as an atomic instruction. Also, proper thread synchronization using locks or atomic variables can prevent race conditions.

Producer Consumer problem

Let's examine the basic model that is sleep and wake. Assume that we have two system calls as **sleep** and **wake**. The process which calls sleep will get blocked while the process which calls will get waked up.

There is a popular example called **producer consumer problem** which is the most popular problem simulating **sleep and wake** mechanism.

The concept of sleep and wake is very simple. If the critical section is not empty then the process will go and sleep. It will be waked up by the other process which is currently executing inside the critical section so that the process can get inside the critical section.

In producer consumer problem, let us say there are two processes, one process writes something while the other process reads that. The process which is writing something is called **producer** while the process which is reading is called **consumer**.

In order to read and write, both of them are usinga buffer. The code that simulates the sleep and wake mechanism in terms of providing the solution to producer consumer problem is shown below.

```
#define N 100 //maximum slots in buffer
#define count=0 //items in the buffer
void producer (void)
{
   int item;
   while(True)
   {
      item = produce_item(); //producer produces an item
      if(count == N) //if the buffer is full then the producer will sleep
         Sleep();
      insert_item (item); //the item is inserted into buffer
      countcount=count+1;
      if(count==1) //The producer will wake up the
      //consumer if there is at least 1 item in the buffer
      wake-up(consumer);
   }
}

void consumer (void)
{
   int item;
   while(True)
   {
      {
         if(count == 0) //The consumer will sleep if the buffer is empty.
         sleep();
         item = remove_item();
         countcount = count - 1;
         if(count == N-1) //if there is at least one slot available in the buffer
         //then the consumer will wake up producer
         wake-up(producer);
         consume_item(item); //the item is read by consumer.
      }
   }
}
```

The producer produces the item and inserts it into the buffer. The value of the global variable count got increased at each insertion. If the buffer is filled completely and no slot is available then the producer will sleep, otherwise it keep inserting.

On the consumer's end, the value of count got decreased by 1 at each consumption. If the buffer is empty at any point of time then the consumer will sleep otherwise, it keeps consuming the items and decreasing the value of count by 1.

The consumer will be waked up by the producer if there is at least 1 item available in the buffer which is to be consumed. The producer will be waked up by the consumer if there is at least one slot available in the buffer so that the producer can write that.

Well, the problem arises in the case when the consumer got preempted just before it was about to sleep. Now the consumer is neither sleeping nor consuming. Since the producer is not aware of the fact that consumer is not actually sleeping therefore it keep waking the consumer while the consumer is not responding since it is not sleeping.

This leads to the wastage of system calls. When the consumer get scheduled again, it will sleep because it was about to sleep when it was preempted.

The producer keep writing in the buffer and it got filled after some time. The producer will also sleep at that time keeping in the mind that the consumer will wake him up when there is a slot available in the buffer.

The consumer is also sleeping and not aware with the fact that the producer will wake him up.

This is a kind of deadlock where neither producer nor consumer is active and waiting for each other to wake them up. This is a serious problem which needs to be addressed.

Using a flag bit to get rid of this problem

A flag bit can be used in order to get rid of this problem. The producer can set the bit when it calls wake-up on the first time. When the consumer got scheduled, it checks the bit.

The consumer will now get to know that the producer tried to wake him and therefore it will not sleep and get into the ready state to consume whatever produced by the producer.

This solution works for only one pair of producer and consumer, what if there are n producers and n consumers. In that case, there is a need to maintain an integer which can record how many wake-up calls have been made and how many consumers need not sleep**. This integer variable is called semaphore.**

We will discuss more about semaphore later in detail.

Introduction to semaphore

To get rid of the problem of wasting the wake-up signals, Dijkstra proposed an approach which involves storing all the wake-up calls. Dijkstra states that, instead of giving the wake-up calls directly to the consumer, producer can store the wake-up call in a variable. Any of the consumers can read it whenever it needs to do so.

Semaphore is the variables which stores the entire wake up calls that are being transferred from producer to consumer. It is a variable on which read, modify and update happens automatically in kernel mode.

Semaphore cannot be implemented in the user mode because race condition may always arise when two or more processes try to access the variable simultaneously. It always needs support from the operating system to be implemented.

According to the demand of the situation, Semaphore can be divided into two categories.

1. Counting Semaphore
2. Binary Semaphore or Mutex

We will discuss each one in detail.

Counting Semaphore

There are the scenarios in which more than one processes need to execute in critical section simultaneously. However, counting semaphore can be used when we need to have more than one process in the critical section at the same time.

The programming code of semaphore implementation is shown below which includes the structure of semaphore and the logic using which the entry and the exit can be performed in the critical section.

In this mechanism, the entry and exit in the critical section are performed on the basis of the value of counting semaphore. The value of counting semaphore at any point of time indicates the maximum number of processes that can enter in the critical section at the same time.

A process which wants to enter in the critical section first decrease the semaphore value by 1 and then check whether it gets negative or not. If it gets negative then the process is pushed in the list of blocked processes (i.e. q) otherwise it gets enter in the critical section.

When a process exits from the critical section, it increases the counting semaphore by 1 and then checks whether it is negative or zero. If it is negative then that means that at least one process is waiting in the blocked state hence, to ensure bounded waiting, the first process among the list of blocked processes will wake up and gets enter in the critical section.

The processes in the blocked list will get waked in the order in which they slept. If the value of counting semaphore is negative then it states the number of processes in the blocked state while if it is positive then it states the number of slots available in the critical section.

```
struct Semaphore
{
   int value; // processes that can enter in the critical section simultaneously.
   queue type L; // L contains set of processes which get blocked
}
Down (Semaphore S)
{
   SS.value = S.value - 1; //semaphore's value will get decreased when a new
   //process enter in the critical section
   if (S.value< 0)
   {
      put_process(PCB) in L; //if the value is negative then
      //the process will get into the blocked state.
      Sleep();
   }
   else
      return;
}
up (Semaphore s)
{
   SS.value = S.value+1; //semaphore value will get increased when
   //it makes an exit from the critical section.
   if(S.value<=0)
   {
      select a process from L; //if the value of semaphore is positive
      //then wake one of the processes in the blocked queue.
      wake-up();
   }
   }
}
```

Problem on Counting Semaphore

.Generally the questions are very simple that contains only subtraction and addition.

Wait → Decre → Down → P
Signal → Inc → Up → V

A Counting Semaphore was initialized to 12. then 10P (wait) and 4V (Signal) operations were computed on this semaphore. What is the result?

S = 12 (initial)
10 p (wait) :
SS = S -10 = 12 - 10 = 2
then 4 V :
SS = S + 4 =2 + 4 = 6

Hence, the final value of counting semaphore is 6.

Binary Semaphore or Mutex

In counting semaphore, Mutual exclusion was not provided because we has the set of processes which required to execute in the critical section simultaneously.

However, Binary Semaphore strictly provides mutual exclusion. Here, instead of having more than 1 slots available in the critical section, we can only have at most 1 process in the critical section. The semaphore can have only two values, 0 or 1.

Let's see the programming implementation of Binary Semaphore.

```
StructBsemaphore

{

   enum Value(0,1); //value is enumerated data type which can only have two values 0 or 1.

   Queue type L;

}

/* L contains all PCBs corresponding to process

Blocked while processing down operation unsuccessfully.

*/

Down (Bsemaphore S)

{

   if (s.value == 1) // if a slot is available in the

   //critical section then let the process enter in the queue.
```

```
    {
        S.value = 0; // initialize the value to 0 so that no other process can read it as 1.
    }
    else
    {
        put the process (PCB) in S.L; //if no slot is available
        //then let the process wait in the blocked queue.
        sleep();
    }
}
Up (Bsemaphore S)
{
    if (S.L is empty) //an empty blocked processes list implies that no process
    //has ever tried to get enter in the critical section.
    {
        S.Value =1;
    }
    else
    {
        Select a process from S.L;
        Wakeup(); // if it is not empty then wake the first process of the blocked queue.
    }
}
```

Message Passing:

It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.

IPC mechanism provides two operations:

- Send (message)- message size fixed or variable
- Received (message)

What is Readers Writer Problem?

Readers writer problem is another example of a classic synchronization problem. There are many variants of this problem, one of which is examined below.

The Problem Statement

There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context. They are **reader** and **writer**. Any number of **readers** can read from the shared resource simultaneously, but only one **writer** can write to the shared resource. When a **writer** is writing data to the resource, no other process can access the resource. A **writer** cannot write to the resource if there are non zero number of readers accessing the resource at that time.

The Solution

From the above problem statement, it is evident that readers have higher priority than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource.

Here, we use one **mutex** `m` and a **semaphore** `w`. An integer variable `read_count` is used to maintain the number of readers currently accessing the resource. The variable `read_count` is initialized to `0`. A value of `1` is given initially to `m` and `w`.

Instead of having the process to acquire lock on the shared resource, we use the mutex `m` to make the process to acquire and release lock whenever it is updating the `read_count` variable.

The code for the **writer** process looks like this:

```
while(TRUE)
{
    wait(w);


   /* perform the write operation */
```

```
   signal(w);
}
```

And, the code for the **reader** process looks like this:

```
while(TRUE)
{
    //acquire lock
    wait(m);
    read_count++;
    if(read_count == 1)
        wait(w);

    //release lock
    signal(m);

    /* perform the reading operation */

    // acquire lock
    wait(m);
    read_count--;
    if(read_count == 0)
        signal(w);

    // release lock
    signal(m);
}
```

Here is the Code uncoded(explained)

- As seen above in the code for the writer, the writer just waits on the **w** semaphore until it gets a chance to write to the resource.
- After performing the write operation, it increments **w** so that the next writer can access the resource.

- On the other hand, in the code for the reader, the lock is acquired whenever the **read_count** is updated by a process.
- When a reader wants to access the resource, first it increments the **read_count** value, then accesses the resource and then decrements the **read_count** value.
- The semaphore **w** is used by the first reader which enters the critical section and the last reader which exits the critical section.
- The reason for this is, when the first readers enters the critical section, the writer is blocked from the resource. Only new readers can access the resource now.
- Similarly, when the last reader exits the critical section, it signals the writer using the **w** semaphore because there are zero readers now and a writer can have the chance to access the resource.

Dining Philosophers Problem

The dining philosophers problem is another classic synchronization problem which is used to evaluate situations where there is a need of allocating multiple resources to multiple processes.

What is the Problem Statement?

Consider there are five philosophers sitting around a circular dining table. The dining table has five chopsticks and a bowl of rice in the middle as shown in the below figure.

Dining Philosophers Problem

At any instant, a philosopher is either eating or thinking. When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

Here's the Solution

From the problem statement, it is clear that a philosopher can think for an indefinite amount of time. But when a philosopher starts eating, he has to stop at some point of time. The philosopher is in an endless cycle of thinking and eating.

An array of five semaphores, `stick[5]`, for each of the five chopsticks.

The code for each philosopher looks like:

```
while(TRUE)
{
    wait(stick[i]);
    /*
        mod is used because if i=5, next
        chopstick is 1 (dining table is circular)
    */
    wait(stick[(i+1) % 5]);

    /* eat */
    signal(stick[i]);

    signal(stick[(i+1) % 5]);
    /* think */
}
```

When a philosopher wants to eat the rice, he will wait for the chopstick at his left and picks up that chopstick. Then he waits for the right chopstick to be available, and then picks it too. After eating, he puts both the chopsticks down.

But if all five philosophers are hungry simultaneously, and each of them pickup one chopstick, then a deadlock situation occurs because they will be waiting for another chopstick forever. The possible solutions for this are:

- A philosopher must be allowed to pick up the chopsticks only if both the left and right chopsticks are available.

- Allow only four philosophers to sit at the table. That way, if all the four philosophers pick up four chopsticks, there will be one chopstick left on the table. So, one philosopher can start eating and eventually, two chopsticks will be available. In this way, deadlocks can be avoided.

Deadlock?

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.

Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R1 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution

because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.

Necessary and sufficient conditions for Deadlock (or)

Deadlock prevention

Deadlock happens only when Mutual Exclusion, hold and wait, No preemption and circular wait holds simultaneously. If it is possible to violate one of the four conditions at any time then the deadlock can never occur in the system.

The idea behind the approach is very simple that we have to fail one of the four conditions but there can be a big argument on its physical implementation in the system.

If we simulate deadlock with a table which is standing on its four legs then we can also simulate four legs with the four conditions which when occurs simultaneously, cause the deadlock.

However, if we break one of the legs of the table then the table will fall definitely. The same happens with deadlock, if we can be able to violate one of the four necessary conditions and don't let them occur together then we can prevent the deadlock.

Let's see how we can prevent each of the conditions.

1. Mutual Exclusion

Mutual section from the resource point of view is the fact that a resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock. If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.

However, if we can be able to violate resources behaving in the mutually exclusive manner then the deadlock can be prevented.

Spooling

For a device like printer, spooling can work. There is a memory associated with the printer which stores jobs from each of the process into it. Later, Printer collects all the jobs and print each one of them according to FCFS. By using this mechanism, the process doesn't have to wait for the printer and it can continue whatever it was doing. Later, it collects the output when it is produced.

Although, Spooling can be an effective approach to violate mutual exclusion but it suffers from two kinds of problems.

1. This cannot be applied to every resource.
2. After some point of time, there may arise a race condition between the processes to get space in that spool.

We cannot force a resource to be used by more than one process at the same time since it will not be fair enough and some serious problems may arise in the performance. Therefore, we cannot violate mutual exclusion for a process practically.

2. Hold and Wait

Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

However, we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait. That means, a process must be assigned all the necessary resources before the execution starts. A process must not wait for any resource once the execution has been started.

!(Hold and wait) = !hold or !wait (negation of hold and wait is, either you don't hold or you don't wait)

This can be implemented practically if a process declares all the resources initially. However, this sounds very practical but can't be done in the computer system because a process can't determine necessary resources initially.

Process is the set of instructions which are executed by the CPU. Each of the instruction may demand multiple resources at the multiple times. The need cannot be fixed by the OS.

The problem with the approach is:

1. Practically not possible.
2. Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.

3. No Preemption

Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.

4. Circular Wait

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	✗
Hold and Wait	Request for all the resources initially	✗
No Preemption	Snatch all the resources	✗
Circular Wait	Assign priority to each resources and order resources numerically	✓

Among all the methods, violating Circular wait is the only approach that can be implemented practically.

Deadlock avoidance

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

Safe and Unsafe States

The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.

A state of a system recorded at some random time is shown below.

Resources Assigned

Process	Type 1	Type 2	Type 3	Type 4
A	3	0	2	2
B	0	0	1	1
C	1	1	1	0
D	2	1	4	0

Resources still needed

Process	Type 1	Type 2	Type 3	Type 4
A	1	1	0	0
B	0	1	1	2
C	1	2	1	0
D	2	1	1	2

E = (7 6 8 4)
P = (6 2 8 3)
A = (1 4 0 1)

Above tables and vector E, P and A describes the resource allocation state of a system. There are 4 processes and 4 types of the resources in a system. Table 1 shows the instances of each resource assigned to each process.

Table 2 shows the instances of the resources, each process still needs. Vector E is the representation of total instances of each resource in the system.

Vector P represents the instances of resources that have been assigned to processes. Vector A represents the number of resources that are not in use.

A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

If the system cannot fulfill the request of all processes then the state of the system is called unsafe.

The key of Deadlock avoidance approach is when the request is made for resources then the request must only be approved in the case if the resulting state is also a safe state.

Bankers algorithm:

Banker's Algorithm is a resource allocation and deadlock avoidance algorithm. This algorithm test for safety simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

In simple terms, it checks if allocation of any resource will lead to deadlock or not, OR is it safe to allocate a resource to a process and if not then resource is not allocated to that process. Determining a safe sequence(even if there is only 1) will assure that system will not go into deadlock.

Banker's algorithm is generally used to find if a safe sequence exist or not. But here we will determine the total number of safe sequences and print all safe sequences.

Following **Data structures** are used to implement the Banker's Algorithm:

- Available vector
- Max Matrix
- Allocation Matrix
- Need Matrix

Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$

Max :

- It is a 2-d array of size '**n*m'** that defines the maximum demand of each process in a system.
- Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$

Need :

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process $\mathbf{P_i}$ currently need **'k'** instances of resource type $\mathbf{R_j}$

for its execution.

- Need [i, j] = Max [i, j] – Allocation [i, j]

Example:

Input:

Total Resources	R1	R2	R3
	10	5	7

Process	Allocation			Max		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3
P2	2	0	0	3	2	2
P3	3	0	2	9	0	2
P4	2	1	1	2	2	2

```
Output: Safe sequences are:
P2--> P4--> P1--> P3
P2--> P4--> P3--> P1
P4--> P2--> P1--> P3
P4--> P2--> P3--> P1

There are total 4 safe-sequences
```

Explanation:

Total resources are R1 = 10, R2 = 5, R3 = 7 and allocated resources are R1 = (0+2+3+2 =) 7, R2 = (1+0+0+1 =) 2, R3 = (0+0+2+1 =) 3. Therefore, remaining resources are R1 = (10 – 7 =) 3, R2 = (5 – 2 =) 3, R3 = (7 – 3 =) 4.

Remaining available = Total resources – allocated resources

and

Remaining need = max – allocated

Process	Max			Allocation			Available			Needed		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	7	5	3	0	1	0	3	3	4	7	4	3
P2	3	2	2	2	0	0				1	2	2
P3	9	0	2	3	0	2				6	0	0
P4	2	2	2	2	1	1				0	1	1
				7	2	3						

So, we can start from either P2 or P4. We can not satisfy remaining need from available resources of either P1 or P3 in first or second attempt step of Banker's algorithm. There are only four possible safe sequences. These are :

P2–> P4–> P1–> P3
P2–> P4–> P3–> P1
P4–> P2–> P1–> P3
P4–> P2–> P3–> P1

Banker's algorithm consists of Safety algorithm and Resource request algorithm

Safety Algorithm:

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.
Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both
a) Finish*[i] = false*
b) Need$_i$ <= Work
if no such i exists goto step (4)
3) Work = Work + Allocation[i]
Finish[i] = true
goto step (2)

4) if Finish [i] = true for all i
then the system is in a safe state

Resource-Request Algorithm:

Let Request$_i$ be the request array for process P_i. Request$_i$ [j] = k means process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

1) If Request$_i$ <= Need$_i$
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
2) If Request$_i$ <= Available
Goto step (3); otherwise, P_i must wait, since the resources are not available.
3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as
follows:
Available = Available – Requesti
Allocation$_i$ = Allocation$_i$ + Request$_i$
Need$_i$ = Need$_i$– Request$_i$

Deadlock detection and recovery:

When a Deadlock Detection Algorithm determines that a deadlock has occurred in the system, the system must recover from that deadlock…..(or)

This approach let the processes fall in deadlock and then periodically check whether deadlock occur in the system or not. If it occurs then it applies some of the recovery methods to the system to get rid of deadlock.

There are two approaches of breaking a Deadlock:

1. Process Termination:

To eliminate the deadlock, we can simply kill one or more processes. For this, we use two methods:

- **(a). Abort all the Deadlocked Processes:**
  Aborting all the processes will certainly break the deadlock, but with a great expenses. The deadlocked processes may have computed for a long time and the result of those partial computations must be discarded and there is a probability to recalculate them later.

- **(b). Abort one process at a time untill deadlock is eliminated:**
  Abort one deadlocked process at a time, untill deadlock cycle is eliminated from the system. Due to this method, there may be considerable overhead, because after aborting each process, we have to run deadlock detection algorithm to check whether any processes are still deadlocked.

2. Resource Preemption:

To eliminate deadlocks using resource preemption, we preepmt some resources from processes and give those resources to other processes. This method will raise three issues –

- **(a). Selecting a victim:**
  We must determine which resources and which processes are to be preempted and also the order to minimize the cost.
- **(b). Rollback:**
  We must determine what should be done with the process from which resources are preempted. One simple idea is total rollback. That means abort the process and restart it.
- **(c). Starvation:**
  In a system, it may happen that same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called **Starvation** and must be avoided. One solution is that a process must be picked as a victim only a finite number of times.

All the best

UNIT-IV MEMORY MANAGEMENT

UNIT – IV MEMORY MANAGEMENT **9**

Memory Management: Basic concept, Logical and Physical address map, Memory allocation – Contiguous Memory allocation – Fixed and variable partition – Internal and External fragmentation and Compaction, Paging – Principle of operation – Page allocation – Hardware support for paging, Protection and Sharing, Disadvantages of Paging.

Virtual Memory: Basics of Virtual Memory – Hardware and control structures – Locality of reference, Page fault , Working Set , Dirty page/Dirty bit – Demand paging, Page Replacement algorithms: Optimal, First in First Out (FIFO), Second Chance (SC), Not recently used (NRU) and Least Recently used (LRU

MEMORY:

Computer memory can be defined as a collection of some data represented in the binary format. On the basis of various functions, memory can be classified into various categories.

A computer device that is capable to store any information or data temporally or permanently, is called storage device.

MEMORY MANAGEMENT:

- Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.
- Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free.
- It checks how much memory is to be allocated to processes.
- It decides which process will get memory at what time.
- It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Memory Management Unit

The purpose of Memory Management Unit (MMU) is to convert the logical address into the physical address. The logical address is the address generated by the CPU for every page while the physical address is the actual address of the frame where each page will be stored.

When a page is to be accessed by the CPU by using the logical address, the operating system needs to obtain the physical address to access that page physically.

The logical address has two parts.

1. Page Number
2. Offset

Memory management unit of OS needs to convert the page number to the frame number.

Example:

Considering the above image, let's say that the CPU demands 10th word of 4th page of process P3. Since the page number 4 of process P1 gets stored at frame number 9 therefore the 10th word of 9th frame will be returned as the physical address.

How Data is being stored in a computer system?

In order to understand memory management, we have to make everything clear about how data is being stored in a computer system.

Machine understands only binary language that is 0 or 1. Computer converts every data into binary language first and then stores it into the memory.

That means if we have a program line written as **int a = 10** then the computer converts it into the binary language and then store it into the memory blocks.

The representation of **inti = 10** is shown below.

The binary representation of 10 is 1010. Here, we are considering 32 bit system therefore, the size of int is 2 bytes i.e. 16 bit. 1 memory block stores 1 bit. If we are using signed integer then the most significant bit in the memory array is always a signed bit.

Signed bit value 0 represents positive integer while 1 represents negative integer. Here, the range of values that can be stored using the memory array is -32768 to +32767.

well, we can enlarge this range by using unsigned int. in that case, the bit which is now storing the sign will also store the bit value and therefore the range will be 0 to 65,535.

Physical and Logical Address Space

Physical Address Space

Physical address space in a system can be defined as the size of the main memory. It is really important to compare the process size with the physical address space. The process size must be less than the physical address space.

Physical Address Space = Size of the Main Memory

If, physical address space = 64 KB = 2 ^ 6 KB = 2 ^ 6 X 2 ^ 10 Bytes = 2 ^ 16 bytes

Let us consider,
word size = 8 Bytes = 2 ^ 3 Bytes

Hence,
Physical address space (in words) = (2 ^ 16) / (2 ^ 3) = 2 ^ 13 Words

Therefore,
Physical Address = 13 bits

In General,
If, Physical Address Space = N Words

then, Physical Address = $\log_2 N$

Logical Address Space

Logical address space can be defined as the size of the process. The size of the process should be less enough so that it can reside in the main memory.

Let's say,

Logical Address Space = 128 MB = (2 ^ 7 X 2 ^ 20) Bytes = 2 ^ 27 Bytes
Word size = 4 Bytes = 2 ^ 2 Bytes

Logical Address Space (in words) = (2 ^ 27) / (2 ^ 2) = 2 ^ 25 Words
Logical Address = 25 Bits

In general,
If, logical address space = L words
Then, Logical Address = Log_2L bits

What is a Word?

The Word is the smallest unit of the memory. It is the collection of bytes. Every operating system defines different word sizes after analyzing the n-bit address that is inputted to the decoder and the 2 ^ n memory locations that are produced from the decoder.

MEMORY ALLOCATION

Contiguous memory allocation.

The earliest and one of the simplest technique which can be used to load more than one processes into the main memory is Fixed partitioning or Contiguous memory allocation.

In this technique, the main memory is divided into partitions of equal or different sizes. The operating system always resides in the first partition while the other partitions can be used to store user processes. The memory is assigned to the processes in contiguous way.

IN contiguous memory allocation,

1. The partitions cannot overlap.
2. A process must be contiguously present in a partition for the execution.

There are various cons of using this technique.

1. Internal Fragmentation

If the size of the process is lesser then the total size of the partition then some size of the partition get wasted and remain unused. This is wastage of the memory and called internal fragmentation.

As shown in the image below, the 4 MB partition is used to load only 3 MB process and the remaining 1 MB got wasted.

2. External Fragmentation :The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.

As shown in the image below, the remaining 1 MB space of each partition cannot be used as a unit to store a 4 MB process. Despite of the fact that the sufficient space is available to load the process, process will not be loaded.

3. Limitation on the size of the process

If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

4. Degree of multiprogramming is less

By Degree of multi programming, we simply mean the maximum number of processes that can be loaded into the memory at the same time. In fixed partitioning, the degree of multiprogramming is fixed and very less due to the fact that the size of the partition cannot be varied according to the size of processes.

BRIEF EXPLANATION ABOUT FRAGMENTATION

Fragmentation:

Fragmentation is a phenomenon in which storage space is used inefficiently, reducing capacity or performance and often both.

The exact consequences of fragmentation depend on the specific system of storage allocation in use and the particular form of fragmentation.

In many cases, fragmentation leads to storage space being "wasted", and in that case the term also refers to the wasted space itself. For other systems (e.g. the FAT file system) the space used to store given data (e.g. files) is the same regardless of the degree of fragmentation (from none to extreme).

Whenever a process is loaded or removed from the physical memory block, it creates a small hole in memory space which is called fragment. Due to fragmentation, the system fails in allocating the contiguous memory space to a process even though it have the requested amount of memory but, in a non-contiguous manner.

The fragmentation is further classified into two categories Internal and External Fragmentation.

Internal fragmentation: Internal fragmentation is the wasted space within each allocated block because of rounding up from the actual requested allocation to the allocation granularity.

External fragmentation: External fragmentation is the various free spaced holes that are generated in either your memory or disk space.

Both the internal and external classification affects data accessing speed of the system. They have a basic difference between them i.e. Internal fragmentation occurs when fixed sized memory blocks are allocated to the process without concerning about the size of the process, and External fragmentation occurs when the processes are allocated memory dynamically.

Basis for comparision	**Internal Fragmentation**	**External Fragmentation**
Basic	It occurs when fixed sized memory blocks are allocated to the processes.	It occurs when variable size memory space are allocated to the processes dynamically.
Occurence	When the memory assigned to the process is slightly larger than the memory requested by the process this creates free space in the allocated block causing internal fragmentation.	When the process is removed from the memory, it creates the free space in the memory causing external fragmentation.
Solution	The memory must be partitioned into variable sized blocks and assign the best fit block to the process.	Compaction, paging and segmentation.

Fixed Partitioning

(Contiguous memory allocation)

Fixed and variable partition

Dynamic Partitioning OR Fixed and variable partition

Dynamic partitioning tries to overcome the problems caused by fixed partitioning. In this technique, the partition size is not declared initially. It is declared at the time of process loading.

The first partition is reserved for the operating system. The remaining space is divided into parts. The size of each partition will be equal to the size of the process. The partition size varies according to the need of the process so that the internal fragmentation can be avoided.

Advantages of Dynamic Partitioning over fixed partitioning

1. No Internal Fragmentation:

Given the fact that the partitions in dynamic partitioning are created according to the need of the process, It is clear that there will not be any internal fragmentation because there will not be any unused remaining space in the partition.

2. No Limitation on the size of the process

In Fixed partitioning, the process with the size greater than the size of the largest partition could not be executed due to the lack of sufficient contiguous memory. Here, In Dynamic partitioning, the process size can't be restricted since the partition size is decided according to the process size.

3. Degree of multiprogramming is dynamic

Due to the absence of internal fragmentation, there will not be any unused space in the partition hence more processes can be loaded in the memory at the same time.

Disadvantages of dynamic partitioning

External Fragmentation

Absence of internal fragmentation doesn't mean that there will not be external fragmentation.

Let's consider three processes P1 (1 MB) and P2 (3 MB) and P3 (1 MB) are being loaded in the respective partitions of the main memory.

After some time P1 and P3 got completed and their assigned space is freed. Now there are two unused partitions (1 MB and 1 MB) available in the main memory but they cannot be used to load a 2 MB process in the memory since they are not contiguously located.

The rule says that the process must be contiguously present in the main memory to get executed. We need to change this rule to avoid external fragmentation

Complex Memory Allocation:

In Fixed partitioning, the list of partitions is made once and will never change but in dynamic partitioning, the allocation and deallocation is very complex since the partition size will be varied every time when it is assigned to a new process. OS has to keep track of all the partitions.

Due to the fact that the allocation and deallocation are done very frequently in dynamic memory allocation and the partition size will be changed at each time, it is going to be very difficult for OS to manage everything.

Compaction

We got to know that the dynamic partitioning suffers from external fragmentation. However, this can cause some serious problems.

To avoid compaction, we need to change the rule which says that the process can't be stored in the different places in the memory.

We can also use compaction to minimize the probability of external fragmentation. In compaction, all the free partitions are made contiguous and all the loaded partitions are brought together.

By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called defragmentation.

Compaction

As shown in the image above, the process P5, which could not be loaded into the memory due to the lack of contiguous space, can be loaded now in the memory since the free partitions are made contiguous.

Problem with Compaction

The efficiency of the system is decreased in the case of compaction due to the fact that all the free spaces will be transferred from several places to a single place.

Huge amount of time is invested for this procedure and the CPU will remain idle for all this time. Despite of the fact that the compaction avoids external fragmentation, it makes system inefficient.

Let us consider that OS needs 6 NS to copy 1 byte from one place to another.

1 B transfer needs 6 NS
256 MB transfer needs 256 X 2^20 X 6 X 10 ^ -9 secs

hence, it is proved to some extent that the larger size memory transfer needs some huge amount of time that is in seconds.

Paging :

In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.

The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.

One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.

Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.

Example:

Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each.

There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.

Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way.

Frames, pages and the mapping between the two is shown in the image below.

Paging

Let us consider that, P2 and P4 are moved to waiting state after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty place. The process P5 of size 8 KB (8 pages) is waiting inside the ready queue.

Given the fact that, we have 8 non contiguous frames available in the memory and paging provides the flexibility of storing the process at the different places. Therefore, we can load the pages of process P5 in the place of P2 and P4.

16 KB
1 Frame → 1 KB
Frame Size = Page Size
Process P1
Process P3
Process P5
P1
P1
P1
P1
P5
P5
P5
P5
P3
P3
P3
P3
P5
P5
P5
P5
Main Memory
(Collection of Frames)

Paging

What is Paging Protection?

The paging process should be protected by using the concept of insertion of an additional bit called Valid/Invalid bit. Paging Memory protection in paging is achieved by associating protection bits with each page. These bits are associated with each page table entry and specify protection on the corresponding page.

Advantages of Paging

Here, are advantages of using Paging method:

- Easy to use memory management algorithm
- No need for external Fragmentation
- Swapping is easy between equal-sized pages and page frames.

Disadvantages of Paging

Here, are drawback/ cons of Paging:

- May cause Internal fragmentation
- Complex memory management algorithm
- Page tables consume additonal memory.
- Multi-level paging may lead to memory reference overhead.

Virtual Memory

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.

Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.

By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.

How Virtual Memory Works?

In modern word, virtual memory has become quite common these days. In this scheme, whenever some pages needs to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory.

Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited

Virtual memory management system:

Let us assume 2 processes, P1 and P2, contains 4 pages each. Each page size is 1 KB. The main memory contains 8 frame of 1 KB each. The OS resides in the first two partitions. In the third partition, 1st page of P1 is stored and the other frames are also shown as filled with the different pages of processes in the main memory.

The page tables of both the pages are 1 KB size each and therefore they can be fit in one frame each. The page tables of both the processes contain various information that is also shown in the image.

The CPU contains a register which contains the base address of page table that is 5 in the case of P1 and 7 in the case of P2. This page table base address will be added to the page number of the Logical address when it comes to accessing the actual corresponding entry.

Advantages of Virtual Memory:

1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory:

1. The system becomes slower since swapping takes time.
2. It takes more time in switching between applications.
3. The user will have the lesser hard disk space for its use.

Hardware and Control Structures

• Memory references are dynamically translated into physical addresses at run time – A process may be swapped in and out of main memory such that it occupies different regions

• A process may be broken up into pieces that do not need to be located contiguously in main memory

• All pieces of a process do not need to be loaded in main memory during execution 3 Execution of a Program

Execution of a program

• Operating system brings into main memory a few pieces of the program

• Resident set - portion of process that is in main memory

• An interrupt is generated when an address is needed that is not in main memory

• Operating system places the process in a blocking state 4 Execution of a Program

Execution of a Program

-Piece of process that contains the logical address is brought into main memory

– Operating system issues a disk I/O Read request

– Another process is dispatched to run while the disk I/O takes place

– An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the Ready state.

LOCALITY OF REFERENCE:

Locality of reference, also known as the **principle of locality**, is the tendency of a processor to access the same set of memory locations repetitively over a short period of time.

There are two basic types of reference locality –

Temporal and spatial locality.

1.**Temporal locality** refers to the reuse of specific data, and/or resources, within a relatively small time duration.

2.Spatial locality (also termed *data locality*) refers to the use of data elements within relatively close storage locations. Sequential locality, a special case of spatial locality, occurs when data elements are arranged and accessed linearly, such as, traversing the elements in a one-dimensional array.

Locality is a type of predictable behavior that occurs in computer systems. Systems that exhibit strong *locality of reference* are great candidates for performance optimization

through the use of techniques such as the caching, prefetching for memory and advanced branch predictors at the pipelining stage of a processor core.

PAGE FAULT:

A page fault occurs when a program attempts to access a block of memory that is not stored in the physical memory, or RAM. The fault notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM.

What is a Page Fault?

If the referred page is not present in the main memory then there will be a miss and the concept is called Page miss or page fault.

The CPU has to access the missed page from the secondary memory. If the number of page fault is very high then the effective access time of the system will become very high.

WORKING SET

- **Working set** is a concept in computer science which defines the amount of memory that a process requires in a given time interval.

DEFINITION:

Peter Denning (1968) defines "the working set of information of a process at time to be the collection of information referenced by the process during the process time interval

Typically the units of information in question are considered to be memory pages. This is suggested to be an approximation of the set of pages that the process will access in the future (say during the next time units), and more specifically is suggested to be an indication of what pages ought to be kept in main memory to allow most progress to be made in the execution of that process.

DIRTY PAGE /DIRTY BIT:

When a bit is modified by the CPU and not written back to the storage, it is called as a dirty bit. This bit is present in the memory cac`he or the virtual storage space.

What is dirty bit?

A dirty bit is a flag that indicates whether an attribute needs to be updated. Such situations usually occur when a bit in a memory cache or virtual memory page that has been changed by a processor but has not been updated in the storage.

Demand Paging

According to the concept of Virtual Memory, in order to execute some process, only a part of the process needs to be present in the main memory which means that only a few pages will only be present in the main memory at any time.

However, deciding, which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult because we cannot say in advance that a process will require a particular page at particular time.

Therefore, to overcome this problem, there is a concept called Demand Paging is introduced. It suggests keeping all pages of the frames in the secondary memory until they are required. In other words, it says that do not load any page in the main memory until it is required.

Whenever any page is referred for the first time in the main memory, then that page will be found in the secondary memory.

After that, it may or may not be present in the main memory depending upon the page replacement algorithm which will be covered later in this tutorial.

Advantages:

Following are the advantages of Demand Paging –

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages:

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Page Replacement Algorithms

The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).

There are two main aspects of virtual memory, Frame allocation and Page Replacement. It is very important to have the optimal frame allocation and page replacement algorithm. Frame allocation is all about how many frames are to be allocated to the process while the page replacement is all about determining the page number which needs to be replaced in order to make space for the requested page.

What If the algorithm is not optimal?

1. If the number of frames which are allocated to a process is not sufficient or accurate then there can be a problem of thrashing. Due to the lack of frames, most of the pages will be residing in the main memory and therefore more page faults will occur.

However, if OS allocates more frames to the process then there can be internal fragmentation.

2. If the page replacement algorithm is not optimal then there will also be the problem of thrashing. If the number of pages that are replaced by the requested pages will be referred in the near future then there will be more number of swap-in and swap-out and therefore the OS has to perform more replacements then usual which causes performance deficiency.

Therefore, the task of an optimal page replacement algorithm is to choose the page which can limit the thrashing.

Types of Page Replacement Algorithms

There are various page replacement algorithms. Each algorithm has a different method by which the pages can be replaced.

1. **Optimal Page Replacement algorithm** → this algorithms replaces the page which will not be referred for so long in future. Although it can not be practically implementable but

it can be used as a benchmark. Other algorithms are compared to this in terms of optimality.

2. **Least recent used (LRU) page replacement algorithm** → this algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future.
3. **FIFO** → in this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rare end of the queue will be replaced on the every page fault.

IN OUR SYLLABUS WE ARE HAVING (BIHER R-2018)

1.OPTIMAL PAGE REPLACEMENT ALGORITHM

2.FIRST IN FIRST OUT(FIFO)

3.REPLACEMENT SECOND CHANCE(SC)

4.NOT RECENTLY USED(NRU)

5.LEAST RECENTLY USED(LRU)

Page Replacement Algorithms :

1.OPTIMAL PAGE REPLACEMENT

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already there so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
0 is already there so —> **0 Page fault.**.
4 will takes place of 1 —> **1 Page Fault.**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.
Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

2.First In First Out (FIFO) –
This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example: Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames.Find number of page faults.

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**
when 3 comes, it is already in memory so —> **0 Page Faults.**
Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**
6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**
Finally when 3 come it is not avilable so it replaces 0 **1 page fault**

3.REPLACEMENT SECOND CHANCE(SC):

Apart from LRU, OPT and FIFO page replacement policies, we also have the second chance/clock page replacement policy. In the Second Chance page replacement policy, the candidate pages for removal are considered in a round robin matter, and a page that has been accessed between consecutive considerations will not be replaced. The page replaced is the one that, when considered in a round robin matter, has not been accessed since its last consideration.

It can be implemented by adding a "second chance" bit to each memory frame-every time the frame is considered (due to a reference made to the page inside it), this bit is set to 1, which gives the page a second chance, as when we consider the candidate page for replacement, we replace the first one with this bit set to 0 (while zeroing out bits of the other pages we see in the process). Thus, a page with the "second chance" bit set to 1 is never replaced during the first consideration and will only be replaced if all the other pages deserve a second chance too!

Example –
Let's say the reference string is **0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4** and we have **3** frames. Let's see how the algorithm proceeds by tracking the second chance bit and the pointer.

SOLUTON:

- Initially, all frames are empty so after first 3 passes they will be filled with {0, 4, 1} and the second chance array will be {0, 0, 0} as none has been referenced yet. Also, the pointer will cycle back to 0.
- **Pass-4:** Frame={0, 4, 1}, second_chance = {0, 1, 0} [4 will get a second chance], pointer = 0 (No page needed to be updated so the candidate is still page in frame 0), pf = 3 (No increase in page fault number).
- **Pass-5:** Frame={2, 4, 1}, second_chance= {0, 1, 0} [0 replaced; it's second chance bit was 0, so it didn't get a second chance], pointer=1 (updated), pf=4
- **Pass-6:** Frame={2, 4, 1}, second_chance={0, 1, 0}, pointer=1, pf=4 (No change)
- **Pass-7:** Frame={2, 4, 3}, second_chance= {0, 0, 0} [4 survived but it's second chance bit became 0], pointer=0 (as element at index 2 was finally replaced), pf=5
- **Pass-8:** Frame={2, 4, 3}, second_chance= {0, 1, 0} [4 referenced again], pointer=0, pf=5
- **Pass-9:** Frame={2, 4, 3}, second_chance= {1, 1, 0} [2 referenced again], pointer=0, pf=5
- **Pass-10:** Frame={2, 4, 3}, second_chance= {1, 1, 0}, pointer=0, pf=5 (no change)
- **Pass-11:** Frame={2, 4, 0}, second_chance= {0, 0, 0}, pointer=0, pf=6 (2 and 4 got second chances)
- **Pass-12:** Frame={2, 4, 0}, second_chance= {0, 1, 0}, pointer=0, pf=6 (4 will again get a second chance)
- **Pass-13:** Frame={1, 4, 0}, second_chance= {0, 1, 0}, pointer=1, pf=7 (pointer updated, pf updated)
- **Page-14:** Frame={1, 4, 0}, second_chance= {0, 1, 0}, pointer=1, pf=7 (No change)
- **Page-15:** Frame={1, 4, 2}, second_chance= {0, 0, 0}, pointer=0, pf=8 (4 survived again due to 2nd chance!)
- **Page-16:** Frame={1, 4, 2}, second_chance= {0, 1, 0}, pointer=0, pf=8 (2nd chance updated)
- **Page-17:** Frame={3, 4, 2}, second_chance= {0, 1, 0}, pointer=1, pf=9 (pointer, pf updated)
- **Page-18:** Frame={3, 4, 2}, second_chance= {0, 1, 0}, pointer=1, pf=9 (No change)

In this example, second chance algorithm does as well as the LRU method, which is much more expensive to implement in hardware.

4.NOT RECENTLY USED (NRU):

The not recently used (NRU) page replacement algorithm is an algorithm that favours keeping pages in memory that have been recently used. This algorithm works on the following principle: when a page is referenced, a referenced bit is set for that page, marking it as referenced. Similarly, when a page is modified (written to), a modified bit is set. The setting of the bits is usually done by the hardware, although it is possible to do so on the software level as well.

At a certain fixed time interval, a timer interrupt triggers and clears the referenced bit of all the pages, so only pages referenced within the current timer interval are marked with a referenced bit. When a page needs to be replaced, the operating system divides the pages into four classes:

3. referenced, modified
2. referenced, not modified
1. not referenced, modified
0. not referenced, not modified

Although it does not seem possible for a page to be modified yet not referenced, this happens when a class 3 page has its referenced bit cleared by the timer interrupt. The NRU algorithm picks a random page from the lowest category for removal. So out of the above four page categories, the NRU algorithm will replace a not-referenced, not-modified page if such a page exists. Note that this algorithm implies that a modified but not-referenced (within the last timer interval) page is less important than a not-modified page that is intensely referenced.

5.LEAST RECENTLY USED (LRU):

In this algorithm page will be replaced which is least recently used.
Example: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames.Find number of page faults.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already their so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
0 is already in memory so —> **0 Page fault**.
4 will takes place of 1 —> **1 Page Fault**
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

7,0,1,2,0,3,0,4,2,3,0,3,2,3 No. of Page frame - 4

1	2	0	3	0	4	2	3	0	3	2	3
	2	2	2	2	2	2	2	2	2	2	2
1	1	1	1	1	4	4	4	4	4	4	4
0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

----------ALL THE BEST WISHES----------

U18PCCS405 - FORMAL LANGUAGES & AUTOMATA THEORY

UNIT I FUNDAMENTALS

Strings, Alphabet, Language, Operations, Finite state machine, definitions, finite automaton model, acceptance of strings, and languages, deterministic finite automaton and non- deterministic finite automaton, transition diagrams and Language recognizers.

Theory of Automata

Theory of automata is a theoretical branch of computer science and mathematical. It is the study of abstract machines and the computation problems that can be solved using these machines. The abstract machine is called the automata. The main motivation behind developing the automata theory was to develop methods to describe and analyse the dynamic behaviour of discrete systems.

This automaton consists of states and transitions. The **State** is represented by **circles**, and the **Transitions** is represented by **arrows**.

Automata is the kind of machine which takes some string as input and this input goes through a finite number of states and may enter in the final state.

There are the basic terminologies that are important and frequently used in automata:

Symbols:

Symbols are an entity or individual objects, which can be any letter, alphabet or any picture.

Example:

1, a, b, #

Alphabets:

Alphabets are a finite set of symbols. It is denoted by $\sum$.

Examples:

$\sum = \{a, b\}$

$\sum = \{A, B, C, D\}$

$\sum = \{0, 1, 2\}$

$\sum = \{0, 1, \ldots\ldots, 5]$

$\Sigma = \{\#, \beta, \Delta\}$

String:

It is a finite collection of symbols from the alphabet. The string is denoted by w.

Example 1:

If $\Sigma = \{a, b\}$, various string that can be generated from Σ are {ab, aa, aaa, bb, bbb, ba, aba.....}.

- A string with zero occurrences of symbols is known as an empty string. It is represented by ε.
- The number of symbols in a string w is called the length of a string. It is denoted by |w|.

Example 2:

w = 010

Number of Sting |w| = 3

Language:

A language is a collection of appropriate string. A language which is formed over Σ can be **Finite** or **Infinite**.

Example: 1

L1 = {Set of string of length 2}

= {aa, bb, ba, bb} **Finite Language**

Example: 2

L2 = {Set of all strings starts with 'a'}

= {a, aa, aaa, abb, abbb, ababb} **Infinite Language**

- **Power of an Alphabet:**
  - If $\sum$ is an alphabet, we can express the set of all strings of certain length from that alphabet by using an exponential notation. It is denoted by $\sum^k$ is the set of strings of length k, each of whose symbols is in $\sum$.

Example :

$\sum = \{0,1\}$ has 2 symbols

i) $\sum^1 = \{0,1\}$ $(\therefore 2^1 = 2)$

ii) $\sum^2 = \{00, 01, 10, 11\}$ $(\therefore 2^2 = 4)$

iii) $\sum^3 = \{000,001,010,011,100,101,110,111\}$ $(\therefore 2^3 = 8)$

- The set of strings over an alphabet $\sum$ is usually denoted by $\sum^*$.
  For instance, $\sum^* = \{0,1\}^* = \{\epsilon,0,1,00,01,10,11\}$
  $(\therefore \sum^* = \sum^0 \cup \sum^1 \cup \sum^2 \ldots\ldots)$ - with ϵ symbol.

- The set of strings over an alphabet $\sum$ excluding ϵ is usually denoted by $\sum^+$.
  For instance, $\sum^+ = \{0,1\}^+ = \{0,1,00,01,10,11\}$
  $(\therefore \sum^+ = \sum^* - \{\epsilon\}$ or $\sum^1 \cup \sum^2 \cup \sum^3 \ldots\ldots)$
  - without ϵ symbol.

Parts of the string:

Parts of a string

Prefix of S

A string obtained by removing zero or more trailing symbols of string S.

Example

ban is a prefix of banana.

Suffix of S

A string formed by deleting zero or more of the leading symbols of S.

Example

nana is a suffix of banana.

Substring of S

A string obtained by deleting a prefix and a suffix from S.

Example

nan is a substring of banana.

Every prefix and every suffix of S is a substring of S, but not every substring of S is a prefix or a suffix of S.

For every string S, both S and $\in$ are prefixes, suffixes, and substrings of S.

Proper prefix

Any nonempty string X that is a prefix of S such that $S \neq X$.

Proper suffix

Any nonempty string X that is a suffix of S such that $S \neq X$.

Proper substring

Any nonempty string X that is a substring of S such that $S \neq X$.

Subsequence of S

Any string formed by deleting zero or more not necessarily contiguous symbols from S.

Example

baaa is a subsequence of banana.

- ✓ **Operations on Languages**
  - ✓ **Complementation**
    Let L be a language over an alphabet Σ. The complementation of L, denoted by $\bar{L}$, is $\Sigma^* - L$
  - ✓ **Union**
    Let L_1 and L_2 be languages over an alphabet Σ. The union of L_1 and L_2, denoted by $L_1 \cup L_2$, is $\{x \mid x \text{ is in } L_1 \text{ or } L_2\}$.
  - ✓ **Intersection**
    Let L_1 and L_2 be languages over an alphabet Σ. The intersection of L_1 and L_2, denoted by $L_1 \cap L_2$, is $\{x \mid x \text{ is in } L_1 \text{ and } L_2\}$.

- ✓ **Concatenation**
  Let L_1 and L_2 be languages over an alphabet Σ. The concatenation of L_1 and L_2, denoted by $L_1 \cdot L_2$, is $\{w_1 \cdot w_2 \mid w_1 \text{ is in } L_1 \text{ and } w_2 \text{ is in } L_2\}$.
- ✓ **Reversal**
  Let L be a language over an alphabet Σ. The reversal of L, denoted by L^r, is $\{w^r \mid w \text{ is in } L\}$.
- ✓ **Kleene's closure**
  Let L be a language over an alphabet Σ. The Kleene's closure of L, denoted by L^*, is $\{x \mid \text{for an integer } n \geq 0\ x = x_1 x_2 \ldots x_n \text{ and } x_1, x_2, \ldots, x_n \text{ are in } L\}$.

LectureNotes.in

$$L^* = \bigcup_{i=0}^{\infty} L^i \qquad (\text{e.g. } a^* = \{\varepsilon, a, aa, aaa, \ldots\ldots\})$$

- ✓ **Positive Closure**
  Let L be a language over an alphabet Σ. The closure of L, denoted by L+, is $\{x \mid \text{for an integer } n \geq 1,\ x = x_1 x_2 \ldots x_n \text{ and } x_1, x2, \ldots, x_n \text{ are in } L\}$

$$L^+ = \bigcup_{i=1}^{\infty} L^i \qquad (\text{e.g. } a^* = \{a, aa, aaa, \ldots\ldots\})$$

2.2 Finite State Machine

The finite state system represents **a mathematical model** of a system with certain input. The model finally gives certain output. The input when is given to the machine it is processed by various states, these states are called as **intermediate states.**

The very good example of finite state system is a control mechanism of elevator. This mechanism only remembers the current floor number pressed, it does not remember all the previously pressed numbers.

The finite state system is a very good design tool for the programs such as text editors and lexical analyzers (which is used in compilers). The lexical analyzer is a program which scans your program character by character and recognizes those words as tokens.

2.2.1 Definition

A finite automata is a collection of 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where,

Q is a finite **set of states**, which is non empty.

Σ is **input** alphabet, indicates input set.

q_0 is an **initial state** and q_0 is in Q i.e. $q_0 \in Q$.

F is a set of **final states.**

δ is a **transition function** or a mapping function. Using this function the next state can be determined.

2.2.2 Finite Automata Model

The finite automata can be represented as :

Fig. 2.1 Model for finite automata

The finite automata can be represented using

i) Input tape - It is a linear tape having some number of cells. Each input symbol is placed in each cell.

ii) Finite control - The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right and at a time only one input symbol is read.

For example, suppose current state is q, and suppose reader is reading the symbol 'a' then it is finite control which decides what will be the next state at input 'a'. The transition from current state q with input w to next state q′ producing w′ will be represented as,

$$(q, w) \vdash (q', w')$$

If w is a string and M is a finite automata, then w is accepted by the FA.

iff $(w, s) \vdash^{*} (q, \varepsilon)$

with q as final state.

The set of strings accepted by a FA given by M then M is accepted by language L. The acceptance of M by some language L is denoted by L (M).

A machine M accepts a language L iff,

$$L = L(M).$$

Transition Diagram:

A transition diagram or state transition diagram is a directed graph which can be constructed as follows:

- There is a node for each state in Q, which is represented by the circle.
- There is a directed edge from node q to node p labeled a if $\delta(q, a) = p$.
- In the start state, there is an arrow with no source.
- Accepting states or final states are indicating by a double circle.

Some Notations that are used in the transition diagram:

✓ **Transition Diagram (Transition graph)**

It is a directed graph associated with the vertices of the graph corresponds to the states of the finite automata. (or) It is a 5-tuple graph used state and edges represent the transitions from one state to other state.

Example:

✓ **Transition Table.**

It is the tabular representation of the DFA. For a transition table the transition function is used.

Example:

States	Input	
	0	1
→{q0}	{q1}	{q0}
{q1}	-	{q2}
*{q2}	-	-

Lecturenotes.in

✓ **Transition Function.**

- The mapping function or transition function denoted by δ.
- Two parameters are passed to this transition function: (i) current state and (ii) input symbol.
- The transition function returns a state which can be called as next state.

δ (current_state, current_input_symbol) = next_state

Example:

δ (q0, a) = q1

✓ **Computation of a Finite Automaton**

- The automaton receives the input symbols one by one from left to right.
- After reading each symbol, M1 moves from one state to another along the transition that has that symbol as its label.
- When M1 reads the last symbol of the input it produces the output: accept if M1 is in an accept state, or reject if M1 is not in an accept state.

- ✓ **Applications**
  - It plays an important role in complier design.
  - In switching theory and design and analysis of digital circuits automata theory is applied.
  - Design and analysis of complex software and hardware systems.
  - To prove the correctness of the program automata theory is used.
  - To design finite state machines such as Moore and mealy machines.
  - It is base for the formal languages and these formal languages are useful of the programming languages.

- ✓ **Types of Finite Automata**
  - Finite Automata without output
    - Deterministic Finite Automata (DFA)
    - Non-Deterministic Finite Automata (NFA or NDFA)
    - Non-Deterministic Finite Automata with ε move (ε-NFA or ε-NDFA)
  - Finite Automata with output
    - Moore Machine
    - Mealy Machine

Deterministic Finite Automata (DFA)

Deterministic Finite Automaton is a FA in which there is **only one path for a specific input from current state to next state.** There is a unique transition on each input symbol.

- ✓ Formal Definition of Deterministic Finite Automata

A finite automaton is a 5-tuples; they are $M=(Q, \Sigma, \delta, q_0, F)$
where

Q is a finite set called the states
Σ is a finite set called the alphabet
$\delta : Q \times \Sigma \rightarrow Q$ is the transition function
$q_0 \in Q$ is the start state also called initial state
$F \subseteq Q$ is the set of accept states, also called the final states

Non-Deterministic Finite Automata (NDFA or NFA)

Non-Deterministic Finite Automaton is a FA in which there **many paths for a specific input from current state to next state.**

- ✓ Formal Definition of Non-Deterministic Finite Automata

  A finite automaton is a 5-tuples; they are $M=(Q, \Sigma, \delta, q_0, F)$

  where

  Q is a finite set called the states

  Σ is a finite set called the alphabet

  $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function

  $q_0 \in Q$ is the start state also called initial state

  $F \subseteq Q$ is the set of accept states, also called the final states

Finite Automaton with ε- moves

The finite automata is called NFA when there exists **many paths for a specific input or ε from current state to next state.** The **ε is a character** used to indicate null string.

- ✓ Formal Definition of Non-Deterministic Finite Automata

  A finite automaton is a 5-tuples; they are $M=(Q, \Sigma, \delta, q_0, F)$

  where

  Q is a finite set called the states

  Σ is a finite set called the alphabet

  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$ is the transition function

  $q_0 \in Q$ is the start state also called initial state

  $F \subseteq Q$ is the set of accept states, also called the final states

Differentiate DFA and NFA

Sl. No	DFA	NFA
1.	DFA is Deterministic Finite Automata	NFA is Non-Deterministic Finite Automata
2.	For given state, on a given input we reach to deterministic and unique state.	For given state, on a given input we reach to more than one state.
3.	DFA is a subset of NFA	Need to convert NFA to DFA in the design of complier.
4.	$\delta : Q \times \Sigma \rightarrow Q$ Example: $\delta(q_0, a) = \{q_1\}$	$\delta : Q \times \Sigma \rightarrow 2^Q$ Example : $\delta(q_0, a) = \{q_1, q_2\}$

Some important points about DFA and NFA:

DFA

DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

NFA

NFA stands for non-deterministic finite automata. It is used to transmit any number of states for a particular input. It can accept the null move.

1. Every DFA is NFA, but NFA is not DFA.
2. There can be multiple final states in both NFA and DFA.
3. DFA is used in Lexical Analysis in Compiler.
4. NFA is more of a theoretical concept.

Example 2.1 : *Design a FA which accepts the only input 101 over the input set* $Z = \{0, 1\}$.

Solution :

Fig. 2.5

Note that in the problem statement it is mentioned as only input 101 will be accepted. Hence in the solution we have simply shown the transitions, for input 101. There is no other path shown for other input.

Example 2.2 : *Design a FA which checks whether the given binary number is even.*

Solution : The binary number is made up of 0's and 1's when any binary number ends with 0 it is always even and when a binary number ends with 1 it is always odd. For example,

0100 is a even number, it is equal to 4.

0011 is a odd number, it is equal to 3.

And so while designing FA we will assume one start state, one state ending in 0 and other state for ending with 1. Since we want to check whether given binary number is even or not, we will make the state for 0, the final state.

Fig. 2.6

The FA indicates clearly S_1 is a state which handles all the 1's and S_2 is a state which handles all the 0's. Let us take some input.

$$01000 \Rightarrow 0\underline{S_2}1000$$

$$01\underline{S_1}000$$

$$010\underline{S_2}00$$

$$0100\underline{S_2}0$$

$$01000S_2$$

Now at the end of input we are in final or in accept state so it is a even number. Similarly let us take another input.

$$1011 \Rightarrow 1\underline{S_1}011$$

$$10\underline{S_2}11$$

$$101\underline{S_1}1$$

$$1011S_1$$

Now we are at the state S_1 which is a non final state.

Another idea to represent FA with the help of transition table.

States \ Input	0	1
→ S_0	S_2	S_1
S_1	S_2	S_1
S_2	S_2	S_1

Transition table

Thus the table indicates in the first column all the current states. And under the column 0 and 1 the next states are shown.

The first row of transition table can be read as : When current state is S_0, on input 0 the next state will be S_2 and on input 1 the next state will be S_1. The arrow marked to S_0 indicates that it is a start state and circle marked to S_2 indicates that it is a final state.

Example 2.3 : *Design FA which accepts only those strings which start with 1 and ends with 0.*

Solution : The FA will have a start state A from which only the edge with input 1 will go to next state.

Fig. 2.6 For Ex. 2.3

In state B if we read 1 we will be in B state but if we read 0 at state B we will reach to state C which is a final state. In state C if we read either 0 or 1 we will go to state C or B respectively. Note that the special care is taken for 0, if the input ends with 0 it will be in final state.

Example 2.4 : *Design FA which accepts odd number of 1's and any number of 0's.*

Solution : In the problem statement, it is indicated that there will be a state which is meant for odd number of 1's and that will be the final state. There is no condition on number of 0's.

Fig. 2.7 For Ex. 2.4

At the start if we read input 1 then we will go to state S_1 which is a final state as we have read odd number of 1's. There can be any number of zeros at any state and therefore the self loop is applied to state S_2 as well as to state S_1. For example if the input is 10101101, in this string there are any number of zeros but odd number of ones. The machine will derive this input as follows -

Fig. 2.8 Ladder diagram of processing the input

Example 2.5 : *Design FA which checks whether the given unary number is divisible by 3.*

Solution :

Fig. 2.9 For Ex. 2.5

The unary number is made up of ones. The number 3 can be written in unary form as 111, number 5 can be written as 11111 and so on. The unary number which is divisible by 3 can be 111 or 111111 or 111111111 and so on. The transition table is as follows

Input	1
→ q_0	q_1
q_1	q_2
q_2	q_3
q_3	q_1

Consider a number 111111 which is equal to 6 i.e. divisible by 3. So after complete scan of this number we reach to final state q_3.

Start 111111

State q_0

$1\,\underline{q_1}\,11111$

$11\,\underline{q_2}\,1111$

$111\,q_3\,111$

$1111\,\underline{q_1}\,11$

$11111\,\underline{q_2}\,1$

$111111\,q_3$ → Now we are in final state.

Example 2.6 : *Design FA to check whether given decimal number is divisible by three.*

Solution : To determine whether the given decimal number is divisible by three, we need to take the input number digit by digit. Also, while considering its divisibility by three, we have to consider that the possible remainders could be 1, 2 or 0. The remainder 0 means, it is divisible by 3. Hence from input set {0, 1, 2, ..., 9} (since decimal number is a input), we will get either remainder 0 or 1 or 2 while testing its divisibility by 3. So we need to group these digits according to their remainders. The groups are as given below -

remainder 0 group : S_0 : (0, 3, 6, 9)

remainder 1 group : S_1 : (1, 4, 7)

remainder 2 group : S_2 : (2, 5, 8)

We have named out these states as S_0, S_1 and S_2. The state S_0 will be the final state as it is remainder 0 state.

Fig. 2.11

Let us test the above FA, if the number is 36 then it will proceed by reading the number digit by digit.

Fig. 2.12

Hence the number is divisible 3, isn't it ? Similarly if number is 121 which is not divisible by 3, it will be processed as

S 121

$1\underline{S_1}\,21$

$12\underline{S_0}\,1$

$121\,S_1$ which is remainder 1 state.

Example 2.7 : *Design FA which checks whether a given binary number is divisible by three.*

Solution : As we have discussed in previous example, the same logic is applicable to this problem even ! The input number is a binary number. Hence the input set will be $\Sigma = \{0, 1\}$. The start state is denoted by S, the remainder 0 is by S_0, remainder 1 by S_1 and remainder 2 by S_2.

Fig. 2.13

In the above transition graph we have handled the case as if the input starts with 1 then it will be in q_3 state which is a dead state and never lead to final state. Thus this machine strictly handles the strings starting with 0 and ending with 1.

Example 2.11 : *Design FA to accept L, where L = {Strings in which a always appears trippled} over the set $\Sigma = \{a, b\}$.*

Solution : For this particular language the valid strings are *aaa*b, b*aaaaaa*, bb*aaa*b and so on. The *a* always appears in a clump of 3. The TG (transition graph) will look like this -

Fig. 2.18

Note that the sequence of tripple *a* is maintained to reach to the final state.

Note the difference between previous example and this, here there is no condition as *a* should be in clump but total number of *a*'s in a string are divisible by 3. Hence b's are allowed in between.

Example 2.12 : *Design FA to accept L where all the strings in L are such that total number of a's in them are divisible by 3.*

Solution : As we have seen earlier, while testing divisibility by 3, we group the input as remainder 0, remainder 1 and remainder 2.

Hence,

S_0 : State of remainder 0.

S_1 : State of remainder 1.

S_2 : State of remainder 2.

Fig. 2.19

Example 2.13 : *Design a FA that reads strings made up of letters in the word. CHARIOT and recognize those strings that contain the word 'CAT' as a substring.*

Solution : To design this FA we will first consider the input set which will be $\Sigma = \{C, H, A, R, I, O, T\}$. From this input set we have to recognize the word 'CAT'. We can do that as follows -

Step 1 :

Fig. 2.20

Step 2 :

Fig. 2.21 Required finite automata

From Fig. 2.21 it is clear that on recognizing C initially from $\Sigma = \{C, H, A, R, I, O, T\}$ we are moving to state q_1, other than C all the characters are remaining in state q_0 only. From q_1 the character A will be identified and from q_2 character T will be identifier. The transition table can be drawn as follows :

State \ Input	C	H	A	R	I	O	T
$\rightarrow q_0$	q_1	q_0	q_0	q_0	q_0	q_0	q_0
q_1	q_1	q_0	q_2	q_0	q_0	q_0	q_0
q_2	q_1	q_0	q_0	q_0	q_0	q_0	q_3
q_3	q_3	q_3	q_3	q_3	q_3	q_3	q_3

The substring may appear anywhere in the input string for any number of times. For example, 'HACCATHA'. From this 'CAT' can be recognized and we should reach to q_3 state finally.

Example 2.18 : *Design a DFA which accepts strings with even number of 0's followed by single 1 over* $\Sigma = \{0, 1\}$.

Solution : The DFA can be shown by a transition diagram as

Here state S_1 will accept all the odd number of 0's and S_0 will accept all even number of 0's. It should then be followed by single 1. Hence S_2 is a final state. Consider the input

	0	0	0	1	0	1
0	S_1	0	0	1	0	1
0	0	S_0	0	1	0	1
0	0	0	S_1	1	0	1
0	0	0	1	S_1	0	1
0	0	0	1	0	S_0	1
0	0	0	1	0	1	S_2

We now will reach to final state S_2, and the input ends here. Hence 000101 is a valid input string. The transition table for above drawn DFA can be represented as

States \ Input	0	1
S_0	S_1	S_2
S_1	S_0	S_1
S_2	-	-

Transition table

Example 2.19 : *Design DFA which accepts all the strings not having more than two a's over $\Sigma = \{a, b\}$.*

Solution : In this DFA maximum two a's are accepted. If we try to accept third a then it should not lead us to final state. Such a DFA can be as shown below.

The transition table can be as shown -

States \ Input	a	b
q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_3	q_3

Transition table

Example 2:

DFA with Σ = {0, 1} accepts all starting with 0.

Solution:

Explanation:

- In the above diagram, we can see that on given 0 as input to DFA in state q0 the DFA changes state to q1 and always go to final state q1 on starting input 0. It can accept 00, 01, 000, 001....etc. It can't accept any string which starts with 1, because it will never go to final state on a string starting with 1.

Example 3:

DFA with Σ = {0, 1} accepts all ending with 0.

Solution:

Explanation:

In the above diagram, we can see that on given 0 as input to DFA in state q0, the DFA changes state to q1. It can accept any string which ends with 0 like 00, 10, 110, 100....etc. It can't accept any string which ends with 1, because it will never go to the final state q1 on 1 input, so the string ending with 1, will not be accepted or will be rejected.

Example 1:

Design a FA with Σ = {0, 1} accepts those string which starts with 1 and ends with 0.

Solution:

The FA will have a start state q0 from which only the edge with input 1 will go to the next state.

In state q1, if we read 1, we will be in state q1, but if we read 0 at state q1, we will reach to state q2 which is the final state. In state q2, if we read either 0 or 1, we will go to q2 state or q1 state respectively. Note that if the input ends with 0, it will be in the final state.

Example 3:

Design FA with Σ = {0, 1} accepts even number of 0's and even number of 1's.

Solution:

This FA will consider four different stages for input 0 and input 1. The stages could be:

Here q0 is a start state and the final state also. Note carefully that a symmetry of 0's and 1's is maintained. We can associate meanings to each state as:

q0: state of even number of 0's and even number of 1's.
q1: state of odd number of 0's and even number of 1's.
q2: state of odd number of 0's and odd number of 1's.
q3: state of even number of 0's and odd number of 1's.

Example 4:

Design FA with Σ = {0, 1} accepts the set of all strings with three consecutive 0's.

Solution:

The strings that will be generated for this particular languages are 000, 0001, 1000, 10001, ... in which 0 always appears in a clump of 3. The transition graph is as follows:

Example 5:

Design a DFA L(M) = {w | w ε {0, 1}*} and W is a string that does not contain consecutive 1's.

Solution:

When three consecutive 1's occur the DFA will be:

Here two consecutive 1's or single 1 is acceptable, hence

The stages q0, q1, q2 are the final states. The DFA will generate the strings that do not contain consecutive 1's like 10, 110, 101,..... etc.

Example 6:

Design a FA with Σ = {0, 1} accepts the strings with an even number of 0's followed by single 1.

Solution:

The DFA can be shown by a transition diagram as:

Acceptance of strings, and languages:

Example 1:

Consider the transition diagram. In our formal notation this FA is denoted $M = (Q, \Sigma, \delta, q_0, F)$,

where $Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

$F = \{q_0\}$

δ is

Transition Table:

	States	Inputs 0	Inputs 1
→ *	q_0	q_2	q_1
	q_1	q_3	q_0
	q_2	q_0	q_3
	q_3	q_1	q_2

Transition diagram

Transition function

$\delta(q_0, 0) = q_2$

$\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_3$

$\delta(q_1, 1) = q_0$

$\delta(q_2, 0) = q_0$

$\delta(q_2, 1) = q_3$

$\delta(q_3, 0) = q_1$

$\delta(q_3, 1) = q_2$

suppose 110101 is input to M.

(i) to show the string 110101 is accepted or not by DFA.

Method 1: 110101

$\delta(q_0, 1) = q_1$

$\delta(q_1, 1) = q_0$

$\delta(q_0, 0) = q_2$

$\delta(q_2, 1) = q_3$

$\delta(q_3, 0) = q_1$

$\delta(q_1, 1) = q_0$

Since the q_0 is the final state, the given input string is accepted.

Method 2:

$$\delta(q_0, 110101)$$
$$= \delta(\delta(q_0, 1), 10101)$$
$$= \delta(q_1, 10101)$$
$$= \delta(\delta(q_1, 1), 0101)$$
$$= \delta(q_0, 0101)$$
$$= \delta(\delta(q_0, 0), 101)$$
$$= \delta(q_2, 101)$$
$$= \delta(\delta(q_2, 1), 01) = \delta(q_3, 01)$$
$$= \delta(\delta(q_3, 0), 1) = \delta(q_1, 1)$$
$$= q_0$$

Here q_0 is the final state.

$\therefore$ It is accepted.

Method 3: 110101

① $\delta(q_0, 1) = q_1$ and $\delta(q_1, 1) = q_0$

$$\delta(q_0, 11) = \delta(\delta(q_0, 1), 1) = \delta(q_1, 1) = q_0$$

② $\delta(q_0, 0) = q_2$

$$\delta(q_2, 110) = \delta(\delta(q_0, 11), 0)$$
$$= \delta(q_0, 0)$$
$$\delta(q_0, 110) = q_2$$

③ $\delta(q_0, 1101) = \delta(\delta(q_0, 110), 1)$

$= \delta(q_2, 1)$

$= q_3$

④ $\delta(q_0, 11010) = \delta(\delta(q_0, 1101), 0)$

$= \delta(q_3, 0)$

$= q_1$

⑤ $\delta(q_0, 110101) = \delta(\delta(q_0, 11010), 1)$

$= \delta(q_1, 1)$

$= q_0$

the entire sequence of states is

$$q_0 \overset{1}{} q_1 \overset{1}{} q_0 \overset{0}{} q_2 \overset{1}{} q_3 \overset{0}{} q_1 \overset{1}{} q_0$$

$\therefore$ thus 110101 is in $L(M)$.

Language accepted by DFA:

> A string x is said to be <u>accepted</u> by a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ if

$\delta(q_0, x) = p$ for some p in F.

> The <u>language accepted</u> by M, designated $L(M)$, is the set $\{x \mid \delta(q_0, x) \text{ is in } F\}$.

> A language is a <u>regular set</u> if it is the set accepted by some finite automaton.

① To show the string 100 is accepted or not by DFA:

100 : input string

$\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_3$

$\delta(q_3, 0) = q_1$

$\therefore$ since q_1 is not a final state, the string 100 is not accepted.

② To show the string 1110 is accepted or not by DFA

1110 : input string

$\delta(q_0, 1) = q_1$

$\delta(q_1, 1) = q_0$

$\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_3$

since q_3 is not a final state, the string 1110 is not accepted.

Example 2:

Consider the DFA, $M = (Q, \Sigma, \delta, q_0, F)$, where δ is given by,

States	Input symbol	
	0	1
→ q_0	q_0	q_1
q_1	q_0	q_2
* q_2	q_2	q_1

Show that the string 011010 is accepted or not.

Let $Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

$S = q_0$

$F = q_2$

Transition diagram:

the string accepted by this DFA = {0110, 010110, 110, 11011, 0110}

Transition function

$\delta(q_0, 0) = q_0$

$\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_0$

$\delta(q_1, 1) = q_2$

$\delta(q_2, 0) = q_2$

$\delta(q_2, 1) = q_1$

To show that the string is accepted or not.

011010 : i/p string

$= \delta(q_0, 011010)$

$= \delta(\delta(q_0, 0), 11010)$

$= \delta(q_0, 11010)$

$= \delta(\delta(q_0, 1), 1010)$

$= \delta(q_1, 1010)$

$= \delta(\delta(q_1, 1), 010)$

$= \delta(q_2, 010)$

$= \delta(\delta(q_2, 0), 10)$

$= \delta(q_2, 10)$

$= \delta(\delta(q_2, 1), 0)$

$= \delta(q_1, 0)$

$= q_0 \notin F$

the string is not accepted.

③ show that the language is accepted by DFA is given as

the DFA is given by $M = (Q, \Sigma, \delta, S, F)$ here

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

$S = q_0 \in Q$

$F = \{q_2\} \subseteq Q$

Transition Table:

	States	Input symbol	
		0	1
→	q_0	q_0	q_1
	q_1	q_0	q_2
*	q_2	q_0	q_2

∴ the language accepted by DFA M is L(M).

$L(M) = \{0111, 0110111, 010111, 111, 011011\}$. set of strings accepted by DFA is L(M).

properties of Transition Function (δ)

(i) $\delta(q, \epsilon) = q$.

this means that the state of the system can be changed only by an input symbol, remains in the original state.

Eg: $\delta(q_0, \epsilon) = q_0$

$\delta(q_1, \epsilon) = q_1$

No input supplied, it will be in the same state.

(ii) For all the strings 'w' and an input symbol 'a'.

$$\delta(q, aw) = \delta(\delta(q, a), w)$$

eg: $\delta(q_0, aw) = \delta(\delta(q_0, a), w)$

$\delta(q_0, 1101) = \delta(\delta(q_0, 1), 101)$

(iii) the transition function δ can be extended to $\bar{\delta}$ to operate on strings.

$$\bar{\delta}(q, \epsilon) = q$$

$$\bar{\delta}(q, xa) = \delta(\bar{\delta}(q, x), a)$$

NON-DETERMINISTIC FINITE AUTOMATA (NFA)

the finite automaton model to allow zero, one or more transitions from a state on the same input symbol. this new model is called a non-deterministic finite Automaton (NFA).

* Any set accepted by a NFA can also be accepted by a DFA.
* NFA is a useful concept in proving theorems.

* Non-determinism plays a central role in both the theory of languages and the theory of computation.

the NFA is defined by 5 tuples $(Q, \Sigma, \delta, S, F)$, where

Q - finite non-empty set of states

Σ - a finite set of input symbols

S - start states

F - set of final states.

the transition function δ is defined by

δ :- transition function that takes state and an input symbol as arguments and returns one (or) more states as output.

$\therefore \delta = Q \times \Sigma^* \text{ to } 2^Q = \hat{\delta}$

<u>Extended transition function ($\bar{\delta}$)</u>

(i) $\bar{\delta}(q, \epsilon) = q$

(ii) $\bar{\delta}(q, xa) = \cup \delta(p, a)$

where $p - \bar{\delta}(q, x)$

<u>Language acceptance by NFA</u>:

the language accepted by NFA 'A' is $L(A)$.

$L(A) = \{ w \mid \delta(q, x) = p, \text{ where } p \text{ is in } F \}$

① Consider the given NFA to check $w = 01001$ accepted or not.

the NFA, M is defined by $N(Q, \Sigma, \delta, S, F)$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$

$S = \{q_0\}$

$F = \{q_2, q_4\}$

δ is defined by the transition table.

δ is defined by the transition table.

	States	Input symbol	
		0	1
→	q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
	q_1	ϕ	$\{q_2\}$
*	q_2	$\{q_2\}$	$\{q_2\}$
	q_3	$\{q_4\}$	ϕ
*	q_4	$\{q_4\}$	$\{q_4\}$

To check whether the string $w = 01001$ is accepted by NFA.

$w = 01001$ be input

$\delta(q_0, 01001)$

(i) $\delta(q_0, 0) = \{q_0, q_3\}$ {from the transition table}

(ii) $\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$

$= \delta(\{q_0, q_3\}, 1)$

$= \delta(q_0, 1) \cup \delta(q_3, 1)$

$= \{q_0, q_1\} \cup \phi$

$\delta(q_0, 01) = \{q_0, q_1\}$

Similarly, we compute

(iii) $\delta(q_0, 010) = \delta(\delta(q_0, 01), 0)$

$= \delta(\{q_0, q_1\}, 0)$

$= \delta(q_0, 0) \cup \delta(q_1, 0)$

$= \{q_0, q_3\} \cup \phi$

$\delta(q_0, 010) = \{q_0, q_3\}$

(iv) $\delta(q_0, 0100) = \delta(\delta(q_0, 010), 0)$

$= \delta(\{q_0, q_3\}, 0)$

$= \delta(q_0, 0) \cup \delta(q_3, 0)$

$= \{q_0, q_3\} \cup \{q_4\}$

$\therefore \delta(q_0, 0100) = \{q_0, q_3, q_4\}$

(v) $\delta(q_0, 01001) = \delta(\delta(q_0, 0100), 1)$

$= \delta(\{q_0, q_3, q_4\}, 1)$

$= \delta(q_0, 1) \cup \delta(q_3, 1) \cup \delta(q_4, 1)$

$= \{q_0, q_1\} \cup \phi \cup \{q_4\}$

$\delta(q_0, 01001) = \{q_0, q_1, q_4\}$

Since q_4 is a final state, the string is accepted.
(ie) the final state q_4 is present in the solution.

$\therefore$ the string 01001 is accepted by NFA.

(2) For the NFA, shown check whether the input string 0100 is accepted (or) not.

the NFA, M is defined by $(Q, \Sigma, \delta, S, F)$

where $Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

$S = \{q_0\} \subseteq Q$

$F = \{q_0\} \in Q$

the transition table is given by

States	Input symbol	
	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$
q_2	$\{q_0, q_2\}$	$\{q_1\}$

To check whether the string $w = 0100$ is accepted by NFA we use

$$\delta(q_0, 0100)$$

(i) let $\delta(q_0, 0) = \{q_0, q_1\}$

(ii) $\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$

$= \delta(\{q_0, q_1\}, 1)$

$= \delta(q_0, 1) \cup \delta(q_1, 1)$

$= \{q_2\} \cup \{q_1, q_2\}$

$\therefore \delta(q_0, 01) = \{q_1, q_2\}$

(iii) $\delta(q_0, 010) = \delta(\delta(q_0, 01), 0)$

$= \delta(\{q_1, q_2\}, 0)$

$= \delta(q_1, 0) \cup \delta(q_2, 0)$

$= \phi \cup \{q_0, q_2\}$

$\therefore \delta(q_0, 010) = \{q_0, q_2\}$

(iv) $\delta(q_0, 0100) = \delta(\delta(q_0, 010), 0)$

$= \delta(\{q_0, q_2\}, 0)$

$= \delta(q_0, 0) \cup \delta(q_2, 0)$

$= \{q_0, q_1\} \cup \{q_0, q_2\}$

$\therefore \delta(q_0, 0100) = \{q_0, q_1, q_2\}$

the final state q_0 is present in the solutions.

thus the string is accepted by NFA.

Example 2.24 : *Construct a NFA for the language*

L_1 = *{consisting a substring 0 1 0 1}*

$L_2 = \{a^n \cup b^n\}$.

Solution : We will consider L_1 first to design NFA. There can be any combination of 0 and 1 in the language but a substring 0101 must be present. We will get such a substring then it leads to a final state or accept state.

This is a NFA as for 0 input we have two different paths one going to q_0 and other is going to q_1.

The string 00010101 is acceptable by above given NFA and it is as shown below.

00010101 $\quad 0q_0 \vdash 010101$

$00q_0 \vdash 010101$

$000q_1 \vdash 10101$

$0001q_2 \vdash 0101$

$00010q_3 \vdash 101$

$000101q_4 \vdash 01$

$0001010\ q_4 \vdash 1$

$00010101q_4 \vdash$

00010101 is accepted as q_4 is final state.

Now we will build a NFA for L_2. The language L_2 is a language in which there be any number of a's or any number of b's. It accepts {a, b, aa, bb, aaa, bbb, ...}.

Hence the NFA will be

The NFA shows two different states q_1 and q_2 for the input ε from q_0 state.

Here ε (pronounced as epsilon) is basically a null move i.e. a move carrying no symbol from input set Σ. But a state change occurs from one state to other.

Example 2.25 : *Design the NFA transition diagram for the transition table as given below -*

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_3\}$	
q_2	$\{q_2, q_3\}$	$\{q_3\}$
q_3	$\{q_3\}$	$\{q_3\}$

Here the NFA is $M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_3\})$.

Solution : The transition diagram can be drawn by using the mapping function as given in table.

$$\delta(q_0, 0) = \{q_0, q_1\}$$

$$\delta(q_0, 1) = \{q_0, q_2\}$$

Then, $\delta(q_1, 0) = \{q_3\}$

Then, $\delta(q_2, 0) = \{q_2, q_3\}$

$\delta(q_2, 1) = \{q_3\}$

Then, $\delta(q_3, 0) = \{q_3\}$

$\delta(q_3, 1) = \{q_3\}$

Example 2.26 : *Construct NFA for the language*

$$L = \left\{0101^n \cup 0100 \mid n \geq 0\right\}$$

Over $\Sigma = \{0, 1\}$.

Solution : Here in language L first three symbols are common i.e. 010. Hence the NFA can be drawn as

The states q_3 and q_5 are final states accepting 0101^n and 0100 respectively. The NFA then can be denoted by,

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \delta, q_0, \{q_3, q_5\})$$

Example 2.27 : *Construct a transition diagram for the NDFA* $M = (\{q_1, q_2, q_3\}, \delta, q_1, \{q_3\})$ *where* δ *is given by,*

$\delta(q_1, 0) = \{q_2, q_3\}$, $\delta(q_1, 1) = \{q_1\}$

$\delta(q_2, 0) = \{q_1, q_2\}$, $\delta(q_2, 1) = \phi$

$\delta(q_3, 0) = \{q_2\}$, $\delta(q_3, 1) = \{q_1, q_2\}$

Solution : Firstly we will design a transition table using the given mapping function δ.

States \ Input	0	1
$\rightarrow q_1$	$\{q_2, q_3\}$	q_1
q_2	$\{q_1, q_2\}$	ϕ
q_3	q_2	$\{q_1, q_2\}$

The NDFA will be

Example 2.28 : *Construct a NFA for a language L which accepts all the strings in which the third symbol from right end is always a. Over $\Sigma = \{a, b\}$.*

Solution : The strings in such a language are of the form

Thus we get third symbol from right end as 'a' always. The NFA then can be

The above figure is a NFA because in state q_0 with input 'a' we can go to either state q_0 or state q_1.

Example 2.29 : *Design NFA accepting all strings ending with 01. Over $\Sigma = \{0, 1\}$*

Solution :

Hence, NFA would be

Example 2.30 : *Design NFA to accept strings with a's and b's such that the string end with 'aa'.*

Solution : The simple FA which accepts a string with 'aa' is -

Now there can be a situation where in

Hence we can design a required NFA as

It can be denoted by,

$$M = (\{q_0, q_1, q_2\}, \delta, \{q_0\}, \{q_2\})$$

We can test some strings for above drawn NFA.

Consider

$$\delta(q_0, aaa) \vdash \delta(q_0, aa)$$
$$\vdash \delta(q_1, a)$$
$$\vdash \delta(q_2, \varepsilon)$$

i.e. we reach to accept state.

$$\delta(q_0, aaa) \vdash \delta(q_0, aa)$$
$$\vdash \delta(q_0, a)$$
$$\vdash \delta(q_1, \varepsilon)$$

i.e. we are not in final state.

Thus there are two possibilities by which we move with string 'aaa' in above given NFA.

Example 2.31 : *Construct a NFA in which double '1' is followed by double '0'. Over* $\Sigma = \{0, 1\}$.

Solution : The FA with double 1 is as drawn below.

It should be immediately followed by double 0.

Then,

Now before double 1 there can be any string of 0 and 1. Similarly after double 0 there can be any string of 0 and 1.

Hence, the NFA becomes

The transition table for above transition diagram can be as given below -

States \ Input	1	0
q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	-
q_2	-	q_3
q_3	-	q_4
q_4	q_4	q_4

Consider string 11100,

$$\delta(q_0, 111000) \vdash \delta(q_0, 1100)$$
$$\vdash \delta(q_0, 100)$$
$$\vdash \delta(q_1, 00)$$

Got stuck! As there is no path from q_1 for input symbol 0. We can process string 11100 in another way.

$$\begin{aligned}\delta(q_0, 11100) &\vdash \delta(q_0, 1100)\\ &\vdash \delta(q_1, 100)\\ &\vdash \delta(q_2, 00)\\ &\vdash \delta(q_3, 0)\\ &\vdash \delta(q_4, \varepsilon)\end{aligned}$$

As q_4 is accept state we get, the complete string scanned and reaching to final state.

Example 2.32 : *Design NFA which accepts the string containing either '01' or '10'. Over* $\Sigma = \{0, 1\}$.

Solution : The NFA can be drawn in stages as follows.

Consider a FA with '01' or '10'.

Now before '01' or '10' there can be any number or 0 and 1. Hence in next stage the FA can be

Now after '01' or '10' there can be any combination of 0 and 1. Hence the NFA in its final stage will be -

We can design a transition table as

δ	Input / States	0	1	Input Σ
	q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$	
→	q_1	–	q_2	
	q_2	q_2	q_2	
	q_3	q_4	-	
	q_4	q_4	q_4	

This section focuses on the fundamental aspects of automata theory. We have learnt basic concepts of DFA and NFA. Now let us discuss interconversion between them.

(13) $L = (ab \cup aba)^*$ $\Sigma = \{a, b\}$

Note

ε-transition

$a^n b^m$ $n > 0$ $b > 0$

(14) L = NFA that accepts all strings ending in '101' $\Sigma = \{0, 1\}$

i/p string 1101

q_3 - final state

- the i/p string accepted

(15) NFA $a^* + (ab)^*$

(16) NFA for any binary string $\Sigma = \{0, 1\}$ that contains 00 (or) 11 as substring

(17) Construct a NFA that accepts sets of all strings over {0,1} of length 2.

$\Sigma = \{0,1\}$

$L = \{00, 01, 10, 11\}$

(18) Construct a NFA that accepts sets of all strings that ends with '1'.

$L = \{1, 01, 11, 011, 101 - - -\}$

(19) Construct a NFA that accepts set of all strings that contain '0'

$L = \{0, 01, 10, 000, 110, 101 - - - -\}$

(20) Construct a NFA that accepts set of all strings that starts with '10'.

L = { 10, 100, 101, 1011 - - - }

(21) Construct a NFA that accepts sets of all strings that contain '01'

L = { 01, 101, 001, 011, 010 - - - }

(22) Construct a NFA that accepts sets of all strings that ends with '11'.

L = { 11, 011, 111, - - - }

(23) Construct a NFA that accepts sets of all strings over {0,1} that have atleast two consecutive 0's (or) 1's.

L = { 00, 11 001, 000, 111, 110 - - - }

DFA vs NFA

NFA refers to Nondeterministic Finite Automaton. DFA refers to Deterministic Finite Automaton. DFA can be best described and understood as one machine. NFA is like multiple small machines that are performing computational activities at the same time. All DFAs are derived from NFAs. The main difference between **DFA and NFA**, the two classes handling the transition functions of finite automata/ finite automaton theory, impact their behaviour in many ways.

Basis of Difference	DFA	NFA
Reaction to symbols	For each symbolic representation of the alphabet, only a singular state transition can be attained in DFA.	No specifications are needed from the user with respect to how certain symbols impact the NFA.
Empty string transition	DFA is not capable of using an Empty String transition.	NFA can use an empty String transition.
Structure	DFA can be best described and understood as one machine.	NFA is like multiple small machines that are performing computational activities at the same time.
Rejection of string	DFA rejects the string in case it terminates in a state that is different from the accepting state.	NFA rejects the string in the event of all branches dying or refusing the string.
Backtracking	It is possible to use backtracking in DFA.	It is not possible to use backtracking at all times in the case of NFA.
Ease of construction	Given its complex nature, it is tougher to construct DFA.	NFA is more easily constructed in comparison to DFA.
Supremacy	All DFAs are derived from NFAs.	All NFAs are not DFAs.
Transition functions	The number related to the next state is one.	The number related to the next state/ states is either zero or one.
Complexities of time	The total time required for running any input string in DFA is less than what it is in NFA.	The total time required for running any input string in NFA is larger than that in comparison to DFA.
Full form	The full form of DFA is Deterministic Finite Automata.	The full form of NFA is Nondeterministic Finite Automata (NFA).

Basis of Difference	DFA	NFA
Space requirement	More space allocation needed.	Less space needed.
The setting of the next possible set	The next possible state is clearly set in DFA.	In NFA, every single pair of input symbols and states may contain many possible next states.

TWO MARKS

1.

What is a formal language?

Language is a set of valid strings from some alphabet. The set may be empty, finite or infinite. L(M) is the language defined by machine M and L(G) is the language defined by context free grammar. The two notations for specifying formal languages are:

- Grammar or regular expression (Generative approach)
- Automaton (Recognition approach)

2.

What is Automata Theory?

- Automata theory is the study of abstract machines (or abstract 'mathematical' machines or systems) and the computational problems that can be solved using these machines. These abstract machines are called automata.
- Automata which means that something is doing something by itself. Automata theory is also closely related to formal language theory, as the automata are often classified by the class of formal languages they are able to recognize. An automaton can be a finite representation of a formal language that may be an infinite set.

3.

Why are switching circuits called as finite state systems?

A switching circuit consists of a finite number of gates, each of which can be in any one of the two conditions 0 or 1. Although the voltages assume infinite set of values, the electronic circuitry is designed so that the voltages corresponding to 0 or 1 are stable and all others adjust to these values. Thus control unit of a computer is a finite state system.

4.

What are the Operations on strings?

- Length of string
- Empty or null string
- Concatenation of string
- Reverse of a string
- Powers of an alphabet
- Substring
- Reversal
- Kleene Closure

5.

Define alphabets in automata?

An alphabet is a finite, non-empty set no of symbols. It is denoted by Σ.

Example:

- ✓ $\Sigma = \{a, b\}$, an alphabet of 2 symbols *a* and *b*.
- ✓ $\Sigma = \{0, 1, 2\}$ an alphabet of 3 symbols *0, 1*, and *2*.

6.

Define strings with examples?

A String or Word is a finite sequence of symbols from Σ. The group of characters also referred as a String.

Example:

- ✓ *a, b,* and *c* are symbols and *abcb* is a string.
- ✓ If $\Sigma = \{a,b\}$ then *abab,aabbb,aaabb,.....* are all strings over the alphabet $\Sigma = \{a,b\}$.
- ✓ *0, 1, 11, 00,* and *01101* are strings over {0, 1}.

7.

What is substring with example?

Let *x* and *y* be strings over an alphabet Σ.

The string *x* is a substring of *y* if there exist strings *w* and *z* over Σ such that *y* = *w x z*.

- ε is a substring of every string.
- For every string *x*, *x* is a substring of *x* itself.

Example:

- ε, comput and computation are substrings of computation.

8.

Explain the following a. Length of string b. Empty or null string?

a. Length of string

Let w is a string. Length of the string is |w|=> number of symbols composing the string.

Example:

1) If w=abcd then |w|=4
2) If x=01010110 the |x|=8
3) If y= 0101 the |y|=4
4) If |Є| = 0

b. Empty or null string

The string consisting of zero symbols. It is denoted by Є.

Example:

{Є}=0.

9.

What is meant by Powers of an alphabet?

Let Σ be the alphabet

Σ^* is the set of all strings over alphabet Σ.

Σ^m denotes the set of all strings over alphabet Σ of length m.

Example:

$\Sigma = \{0, 1\}$

$\Sigma^0 = \{\epsilon\}$ empty string of length 0.

$\Sigma^1 = \{0, 1\}$ is the set of all strings of length one over $\Sigma = \{0, 1\}$

$\Sigma^2 = \{00, 01, 10, 11\}$ is the set of all strings of length two over $\Sigma = \{0, 1\}$.

10.

Define Kleen Closure?

Let Σ be the alphabet

" Kleen Closure"-> Σ^* denotes the set of all strings (Σ) over the alphabet

Example:

If $\Sigma = \{a\}$ the $\Sigma^* = \{\epsilon, a, aa, aaa, \ldots\ldots\ldots\}$

If $\Sigma = \{0,1\}$ the $\Sigma^* = \{\epsilon, 0, 1, 00, 10, 10, \ldots\ldots\ldots\ldots\ldots\ldots\}$

$\Sigma^0 = \{\epsilon\}$

$\Sigma^1 = \{a\}$

$\Sigma^2 = \{aa\}$

Therefore $\Sigma^* = \Sigma^0 U\ \Sigma^1 U\ \Sigma^2, \ldots\ldots\ldots\ldots\ldots$

11.

Define a language.

An alphabet is a finite set of symbols. A language is a set of strings of symbols from someone alphabet.

e.g. If $\Sigma = \{0,1\}$,then $\Sigma^* = \{\epsilon, 0, 1, 01, 11, 10, 111, \ldots\ldots\ldots\}$

12.

List out the applications of automata theory?

- String Processing
- Analysis of algorithms
- Complexity Theory
- Cryptography
- Compiler Design
- Interpreters
- Parser
- Circuit design
- Robotics
- Artificial Intelligence
- Knowledge Engineering
- Natural language Processing

13.

What are Finite Automata? Give an example.

- A finite automaton (FA) consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from an alphabet Σ.
- Each input symbol there is exactly one transition out of each state (possibly back to the state itself).
- One State, usually q_0, is the initial state, in which the automaton starts.
- Some states are designated as final or accepting states.

Example:

Let a deterministic finite automaton be

1. Q = {a, b, c},
2. ∑ = {0, 1},
3. q_0={a},
4. F={c}, and
5. Transition function δ as shown by the following table

Present State	Next State for Input 0	Next State for Input 1
a	a	b
b	c	a
c	b	c

Its graphical representation would be as follows –

14.

How does an FA work?

- At the beginning,
  - ✓ an FA is in the start state (initial state)
  - ✓ its tape head points at the first cell
- For each move, FA
  - ✓ reads the symbol under its tape head
  - ✓ changes its state (according to the transition function) to the next state determined by the symbol read from the tape and its current state
  - ✓ move its tape head to the right one cell

15.

. Give the purpose of transition diagram?

A directed graph, called a transition diagram, is associated with an FA as follows:

- The vertices of the graph correspond to the states of the FA.
- If there is a transition from state q to state p on input a, then there is an arc labeled a from state q to state p in the transition diagram.
- The FA accepts a string x if the sequence of transitions corresponding to the symbols of x leads from the start state to an accepting state.

16.

What is Transition Table?

The transition table is the tabular representation of the transition function "delta (∂)" with the rows denoting states and columns denoting input symbol.

17.

What are the applications of Finite Automata (FA)?

- Software for designing and checking the behavior of digital circuits.
- Lexical analyzer of a typical compiler.
- Software for scanning large bodies of text (e.g., web pages) for pattern finding.
- Software for verifying systems of all types that have a finite number of states (e.g., stock market transaction, communication/network protocol).

18.

Write the DFA Specifications?

Deterministic Finite Automata (DFA) is generally specified by a 5 tuples **($Q, \Sigma, \delta, q_0, F$),**

where

1. **Q** is a finite set of states
2. **Σ** is a finite input alphabet
3. **q_0** in Q is the start state or initial state
4. **$F \subseteq Q$** is the set of final states
5. **δ** is the transition function mapping **$\delta: Q \times \Sigma \rightarrow Q$**.(i.e.) δ(q,a) is a state for each state *q* and input symbol *a*.

19.

Explain working of DFA?

1. **Initialisation**
   - Reader (read head) should be over the leftmost symbol.
   - Finite Control is in start state.
2. **Single Step**
   - Reader moves to the next symbol to the right.
   - Control enters a new state that (deterministically) depends on current state and current symbol. There may be no desired next state, Machine will stop.
   - The machine repeats this action.
3. **No current symbol**
   - All Symbols have been read.
   - If control is in final state, the input string is accepted.
   - Otherwise, the input string is not accepted.

20.

What are the Applications of DFA?

- Compiler Design
- Design of Lexical Analyser
- Text Editor
- Pattern Matching
- File searching Program
- Text Processing

21.
. **Define NFA.**

Non-Deterministic Finite Automata (NFA) is generally specified by a 5 tuples **(Q, Σ, δ, q_0, F)**

Where,

1. **Q** is a finite set of states
2. **Σ** is a finite input alphabet
3. **q_0** in Q is the start state or initial state
4. **F⊆Q** is the set of final states
5. **δ** is the transition function mapping δ → Q x Σ* to 2^Q.

On receiving same inputs it goes to many states

- the machine can move without consuming any symbols and sometimes there are possible moves and sometimes there are move then one possible moves.
- the state is only partially determined by the current state and i/p symbol.

22.

What are the difference between NFA and DFA?

DFA	NFA
i. For each and every state and for each and every input symbol there exists at most one transition. **ii.** The transition mapping for DFA is **Q×Σ→Q.** **iii.** The language accepted by DFA is denoted as L(M). **iv.** Epsilon transition is not possible.	**i.** For each and every state and for each and every input symbol there exists more than one transition. **ii.** The transition mapping for NFA is **Q × (Σ U {ε}) →2^Q.** **iii.** The language accepted by NFA is denoted by L(M'). **iv.** Epsilon transition is possible.

23.

What are Transition Function and give their properties?

Changing from one state to another state on applying the inputs

<u>Properties of Transition function:</u>

1. $\delta(q, \varepsilon)=q$

   This means the state of the system can be changed only by an input symbol else remains in original state.

2. For all strings w and input symbol a $\delta(q,aw)= \delta(\delta(q,a),w)$

   Similarly $\delta(q,wa)= \delta(\delta(q,w),a)$

3. The transition function δ can be extended to δ or δ that operates on states and strings.

   Basis: $\delta(q, \varepsilon)=q$

   Induction: $\delta(q,xa)= \delta(\delta(q,x),a)$

UNIT-II

FINITE AUTOMATA

NFA with ε-transitions - Significance, acceptance of languages. Conversions and Equivalence : Equivalence between NFA with and without ε transitions, NFA to DFA conversion, minimization of FSM, equivalence between two FSM's, Finite Automata with output-Moore and Melay machines.

2.1 Introduction

In earlier chapter we have learnt the concept of finite automata. We have also discussed the two types of finite automata: DFA and NFA. These two types of automata are equivalent. That means we can convert any NFA to DFA. These automata are designed to accept some language. Basically there are two types of languages: **regular languages** and **non regular languages**. The finite automata is a mathematical model which always accepts regular languages.

In this chapter, we will discuss the equivalence between NFA and DFA. And then we will discuss the logical unit of representing regular language i.e. **Regular expressions**. We will also discuss the relationship between regular expression and finite state machines.

2.2 NFA with ε - Transitions

The ε-transitions in NFA are given in order to move from one state to another without having any symbol from input set Σ.

Fig. 2.1 NFA with ε - transitions

Consider the NFA with ε as :

In this NFA with ε, q_0 is a start state with input 0 one can be either in state q_0 or in state q_1. If we get at the start a symbol 1 then with ε - move we can change state from q_0 to q_1 and then with input we can be in state q_1. On the other hand, from start state q_0, with input 1 we can reach to state q_2. Thus it is not definite that on input 1 whether we will be in state q_1 or q_2. Hence it is called nondeterministic finite automata (NFA) and since there are some ε moves by which we can simply change the states from one state to other. Hence it is called NFA with ε.

2.2.1 Significance of NFA with ε

As construction of DFA is very difficult for certain input set, we construct NFA. This NFA then can be converted to DFA. ε is an empty string. The ε - transitions are used simply to change one state to other. Sometimes to reach to final state we do not get proper state from start state. In such a case we simply want to reach to certain state which leads to final state. Such a transition to that specific state should be without any input symbol. Hence we require some ε - moves by which a proper state can be obtained for reaching to final state. Thus ε - moves play an important role in NFA.

2.2.2 Acceptance of Language

The language L accepted by NFA with ε, denoted by $M = (Q, \Sigma, \delta, q_0, F)$ can be defined as :

Let, $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA with ε.

Where

Q is a finite set of states.

Σ is input set.

δ is a transition or a mapping function for transitions from $Q \times \{\Sigma \cup \varepsilon\}$ to 2^Q.

q_0 is a start state.

F is a set of final states such that $F \in Q$.

The string w in L accepted by NFA can be represented as

$L(M) = \{w \mid w \in \Sigma^* \text{ and } \delta \text{ transition for } w \text{ from } q_0 \text{ reaches to } F\}$

Example 2.1 : *Construct NFA with ε which accepts a language consisting the strings of any number of a's followed by any number of b's. Followed by any number of c's.*

Solution : Here any number of a's or b's or c's means zero or more in number. That means there can be zero or more a's followed by zero or more b's followed by zero or more c's. Hence NFA with ε can be -

Fig. 2.2

Normally ε's are not shown in the input string. The transition table can be -

δ	Σ			
State \ Input	a	b	c	ε
q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ

We can parse the string aabbcc as follows -

$\delta(q_0, aabbcc) \vdash \delta(q_0, abbcc)$
$\vdash \delta(q_0, bbcc)$
$\vdash \delta(q_0, \varepsilon bbcc)$
$\vdash \delta(q_1, bbcc)$
$\vdash \delta(q_1, bcc)$
$\vdash \delta(q_1, cc)$
$\vdash \delta(q_1, \varepsilon cc)$
$\vdash \delta(q_2, cc)$
$\vdash \delta(q_2, c)$
$\vdash \delta(q_2, \varepsilon)$

Thus we reach to accept state, after scanning the complete input string.

Definition of ε - closure

The ε - closure (p) is a set of all states which are reachable from state p on ε - transitions such that :

i) ε - closure (p) = p where $p \in Q$.

ii) If there exists ε - closure (p) = {q} and $\delta(q, \varepsilon) = r$ then
ε - closure (p) = {q, r}

Example 2.2 : *Find ε - closure for the following NFA with ε*

Fig. 2.3

Solution :

ε - closure $(q_0) = \{q_0, q_1, q_2\}$ means self state + ε - reachable states.

ε - closure $(q_1) = \{q_1, q_2\}$ means q_1 is a self state and q_2 is a state obtained from q_1 with ε input.

ε - closure $(q_2) = \{q_2\}$

2.3 Conversions and Equivalence

The NFA with ε can be converted to NFA without ε. And this NFA without ε can be converted to DFA. In this section we will discuss these conversions and will find them equivalent to each other.

Fig. 2.4 Moves to DFA

2.3.1 Conversion from NFA with ε to NFA without ε

In this method we try to remove all the ε transitions from given NFA. The method will be

1. Find out all the ε transitions from each state from Q. That will be called as ε-closure $\{q_i\}$ where $q_i \in Q$.
2. Then δ' transitions can be obtained. The δ' transitions means an ε-closure on δ moves.
3. Step-2 is repeated for each input symbol and for each state of given NFA.
4. Using the resultant states the transition table for equivalent NFA without ε can be built.

Theorem : If L is accepted by NFA with ε transitions, then there exist L which is accepted by NFA without ε transitions.

Theorem : If L is accepted by NFA with ε transitions, then there exist L which is accepted by NFA without ε transitions.

Proof :

Let, $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA with ε transitions.

Construct $M' = (Q, \Sigma, \delta', q_0, F')$ where

$$F' = \begin{cases} F \cup \{q_0\} & \text{if } \varepsilon \text{ - closure contains state off} \\ F & \text{otherwise} \end{cases}$$

M' is a NFA without ε moves. The δ' function can be denoted by δ^* with some input. For example, $\delta'(q, a) = \delta^*(q, a)$ for some q in Q and a from Σ. We will apply the method of induction with input x. The x will not be ε because

$\delta'(q_0, \varepsilon) = \{q_0\}$.

$\delta^*(q_0, \varepsilon) = \varepsilon$-closure (q_0). Therefore we will assume length of string to be 1.

Basis : $|x| = 1$. Then x is a symbol a.

$$\delta'(q_0, a) = \delta^*(q_0, a)$$

Induction : $|x| > 1$ Let $x = wa$

$$\delta'(q_0, wa) = \delta'(\delta'(q_0, w), a)$$

By inductive hypothesis,

$$\delta'(q_0, w) = \delta^*(q_0, w) = p$$

Now we will show that $\delta'(p, a) = \delta(q_0, wa)$

But $\delta'(p, a) = \bigcup_{q \text{ in } p} \delta'(q, a) = \bigcup_{q \text{ in } p} \delta^*(q, a)$

As $p = \delta^*(q_0, w)$

We have, $\bigcup_{q \text{ in } p} \delta^*(q, a) = \delta^*(q_0, wa)$

Thus by definition δ^*

$$\delta'(q_0, wa) = \delta^*(q_0, wa)$$

Rule for conversion

$$\boxed{\begin{array}{l} \delta'(q, a) = \varepsilon \text{ - closure}\left(\delta\left(\hat{\delta}(q, \varepsilon), a\right)\right) \\ \text{where } \hat{\delta}(q, \varepsilon) = \varepsilon \text{ - closure}(q) \end{array}}$$

Before solving some examples based on conversion of NFA with ε to NFA without ε above rule should be remembered.

Example 2.3 :*Convert the given NFA with ε to NFA without ε.*

Fig. 2.5

Solution : We will first obtain ε - closure of each state i.e. we will find out ε - reachable states from current state.

Hence

$$\varepsilon \text{ - closure } (q_0) = \{q_0, q_1, q_2\}$$

ε - closure $(q_1) = \{q_1, q_2\}$

ε - closure $(q_2) = \{q_2\}$

As ε - closure (q_0) means with null input (no input symbol) we can reach to q_0 , q_1 or q_2. In a similar manner for q_1 and q_2 ε - closures are obtained. Now we will obtain δ' transitions for each state on each input symbol.

$$\begin{aligned}
\delta'(q_0, 0) &= \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_0, \varepsilon), 0\right)\right) \\
&= \varepsilon\text{ - closure}\left(\delta\left(\varepsilon\text{ - closure}(q_0), 0\right)\right) \\
&= \varepsilon\text{ - closure}\left(\delta(q_0, q_1, q_2), 0\right) \\
&= \varepsilon\text{ - closure}\left(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)\right) \\
&= \varepsilon\text{ - closure}\left(q_0 \cup \phi \cup \phi\right) \\
&= \varepsilon\text{ - closure}(q_0) = \{q_0, q_1, q_2\} \\
\delta'(q_0, 1) &= \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_0, \varepsilon), 1\right)\right) \\
&= \varepsilon\text{ - closure}\left(\delta(q_0, q_1, q_2), 1\right) \\
&= \varepsilon\text{ - closure}\left(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)\right) \\
&= \varepsilon\text{ - closure}\left(\phi \cup q_1 \cup \phi\right) \\
&= \varepsilon\text{ - closure}(q_1)
\end{aligned}$$

$$\delta'(q_0, 1) = \{q_1, q_2\}$$

$$\begin{aligned}
\delta'(q_1, 0) &= \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_1, \varepsilon), 0\right)\right) \\
&= \varepsilon\text{ - closure}\left(\delta\left(\varepsilon\text{ - closure}(q_1), 0\right)\right) \\
&= \varepsilon\text{ - closure}\left(\delta(q_1, q_2), 0\right) \\
&= \varepsilon\text{ - closure}\left(\delta(q_1, 0) \cup \delta(q_2, 0)\right) \\
&= \varepsilon\text{ - closure}\left(\phi \cup \phi\right) \\
&= \varepsilon\text{ - closure}(\phi) \\
&= \phi \\
\delta'(q_1, 1) &= \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_1, \varepsilon), 1\right)\right) \\
&= \varepsilon\text{ - closure}\left(\delta\left(\varepsilon\text{ - closure}(q_1), 1\right)\right) \\
&= \varepsilon\text{ - closure}\left(\delta(q_1, q_2), 1\right)
\end{aligned}$$

$$= \varepsilon\text{ - closure } (\delta(q_1, 1) \cup \delta(q_2, 1))$$

$$= \varepsilon\text{ - closure } (q_1 \cup \phi)$$

$$= \varepsilon\text{ - closure } (q_1)$$

$$= \{q_1, q_2\}$$

$$\delta^*(q_2, 0) = \varepsilon\text{ - closure } \left(\delta\left(\hat{\delta}(q_2, \varepsilon), 0\right)\right)$$

$$= \varepsilon\text{ - closure } \left(\delta\left(\varepsilon\text{ - closure}(q_2), 0\right)\right)$$

$$= \varepsilon\text{ - closure } (\delta(q_2, 0))$$

$$= \varepsilon\text{ - closure } (\phi)$$

$$= \phi$$

$$\delta^*(q_2, 1) = \varepsilon\text{ - closure } \left(\delta\left(\hat{\delta}(q_2, \varepsilon), 1\right)\right)$$

$$= \varepsilon\text{ - closure } \left(\delta\ \left(\varepsilon\text{ - closure}(q_2), 1\right)\right)$$

$$= \varepsilon\text{ - closure } (\delta(q_2, 1))$$

$$= \varepsilon\text{ - closure } (\phi)$$

$$\delta^*(q_2, 1) = \phi$$

$$\delta^*(q_0, 2) = \varepsilon\text{ - closure } \left(\delta\left(\hat{\delta}(q_0, \varepsilon), 2\right)\right)$$

$$= \varepsilon\text{ - closure } \left(\delta\left(\varepsilon\text{ - closure}(q_0), 2\right)\right)$$

$$= \varepsilon\text{ - closure } (\delta(q_0, q_1, q_2), 2)$$

$$= \varepsilon\text{ - closure } (\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2))$$

$$= \varepsilon\text{ - closure } (\phi \cup \phi \cup q_2)$$

$$= \varepsilon\text{ - closure } (q_2)$$

$$= \{q_2\}$$

$$\delta^*(q_1, 2) = \varepsilon\text{ - closure } \left(\delta\left(\hat{\delta}(q_1, \varepsilon), 2\right)\right)$$

$$= \varepsilon\text{ - closure } \left(\delta\left(\varepsilon\text{ - closure } (q_1), 2\right)\right)$$

$$= \varepsilon\text{ - closure } (\delta(q_1, q_2), 2)$$

$$= \varepsilon\text{ - closure } (\delta(q_1, 2) \cup \delta(q_2, 2))$$

$$= \varepsilon\text{ - closure } (\phi \cup q_2)$$

$$= \{q_2\}$$

$$\delta'(q_2, 2) = \varepsilon\text{ - closure } \left(\delta\left(\hat{\delta}(q_2, \varepsilon), 2\right)\right)$$

$$= \varepsilon\text{ - closure } \left(\delta\left(\varepsilon\text{ - closure}(q_2), 2\right)\right)$$

$$= \varepsilon\text{ - closure } \left(\delta\left(q_2, 2\right)\right)$$

$$= \varepsilon\text{ - closure } (q_2)$$

$$= \{q_2\}$$

Now we will summarize all the computed δ' transitions -

$\delta'(q_0, 0) = \{q_0, q_1, q_2\}$ $\delta'(q_0, 1) = \{q_1, q_2\}$ $\delta'(q_0, 2) = \{q_2\}$

$\delta'(q_1, 0) = \phi$ $\delta'(q_1, 1) = \{q_1, q_2\}$ $\delta'(q_1, 2) = \{q_2\}$

$\delta'(q_2, 0) = \phi$ $\delta'(q_2, 1) = \phi$ $\delta'(q_2, 2) = \{q_2\}$

From this we can write the transition table as -

State \ Input	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

The NFA will be -

Fig. 2.6

Here q_0, q_1 and q_2 is a final state because ε - closure (q_0), ε - closure (q_1) and ε - closure (q_2) contains final state q_2.

Example 2.4 : *Convert the following NFA with ε to NFA without ε.*

Fig. 2.7

Solution : We will first obtain ε - closure of every state. The ε - closure is basically an ε - transition from one state to other. Hence

ε - closure (0) = {0}

ε - closure (1) = {1}

ε - closure (2) = {2, 3, 4, 6, 9}

ε - closure (3) = {3, 4, 6}

ε - closure (4) = {4}

ε - closure (5) = {5, 8, 3, 4, 6, 9}

= {3, 4, 5, 6, 8, 9} sorted it!

ε - closure (6) = {6}

ε - closure (7) = {7, 8, 3, 4, 6, 9}

= {3, 4, 6, 7, 8, 9}

ε - closure (8) = {8, 3, 4, 6, 9}

= {3, 4, 6, 8, 9}

ε - closure (9) = {9}

Now we will obtain δ' transitions for each state and for each input symbol.

$$\delta'(0, a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(0, \varepsilon), a\right)\right)$$

$$= \varepsilon\text{-closure}\big(\delta(\varepsilon\text{-closure}(0), a)\big)$$

$$= \varepsilon\text{-closure}\,(\delta(0, a))$$

$$= \varepsilon\text{-closure}(1)$$

$$= \{1\}$$

$$\delta'(0, b) = \varepsilon\text{-closure}\Big(\delta\big(\hat{\delta}(0,\varepsilon), b\big)\Big)$$

$$= \varepsilon\text{-closure}\big(\delta(\varepsilon\text{-closure}(0), b)\big)$$

$$= \varepsilon\text{-closure}\,(\delta(0, b))$$

$$= \varepsilon\text{-closure}(\phi)$$

$$= \phi$$

$$\delta'(1, a) = \varepsilon\text{-closure}\Big(\delta\,\big(\hat{\delta}(1,\varepsilon), a\big)\Big)$$

$$= \varepsilon\text{-closure}\big(\delta(\varepsilon\text{-closure}(1), a)\big)$$

$$= \varepsilon\text{-closure}\,(\delta(1, a))$$

$$= \varepsilon\text{-closure}(\phi)$$

$$= \phi$$

$$\delta'(1, b) = \varepsilon\text{-closure}\Big(\delta\,\big(\hat{\delta}(1,\varepsilon), b\big)\Big)$$

$$= \varepsilon\text{-closure}\big(\delta(\varepsilon\text{-closure}(1), b)\big)$$

$$= \varepsilon\text{-closure}\,(\delta(1, b))$$

$$= \varepsilon\text{-closure}(2)$$

$$= \{2, 3, 4, 6, 9\}$$

$$\delta'(2, a) = \varepsilon\text{-closure}\Big(\delta\,\big(\hat{\delta}(2,\varepsilon), a\big)\Big)$$

$$= \varepsilon\text{-closure}\big(\delta(\varepsilon\text{-closure}(2), a)\big)$$

$$= \varepsilon\text{-closure}\,(\delta(2, 3, 4, 6, 9), a)$$

$$= \varepsilon\text{-closure}\big(\delta(2, a) \cup \delta(3, a) \cup \delta(4, a) \cup \delta(6, a) \cup \delta(9, a)\big)$$

$$= \varepsilon\text{-closure}(\phi \cup \phi \cup 5 \cup \phi \cup \phi)$$

$$= \varepsilon\text{-closure}(5)$$

$$= \{3, 4, 5, 6, 8, 9\}$$

$$\delta(2, b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(2, \varepsilon), b\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(2), b)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(2, 3, 4, 6, 9), b\right)$$

$$= \varepsilon\text{-closure}\left(\delta(2, b) \cup \delta(3, b) \cup \delta(4, b) \cup \delta(6, b) \cup \delta(9, b)\right)$$

$$= \varepsilon\text{-closure}(\phi \cup \phi \cup \phi \cup 7 \cup \phi)$$

$$= \varepsilon\text{-closure}(7)$$

$$= \{3, 4, 6, 7, 8, 9\}$$

$$\delta(3, a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(3, \varepsilon), a\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(3), a)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, 4, 6), a\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, a) \cup \delta(4, a) \cup \delta(6, a)\right)$$

$$= \varepsilon\text{-closure}(\phi \cup 5 \cup \phi)$$

$$= \varepsilon\text{-closure}(5)$$

$$= \{3, 4, 5, 6, 8, 9\}$$

$$\delta(3, b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(3, \varepsilon), b\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(3), b)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, 4, 6), b\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, b) \cup \delta(4, b) \cup \delta(6, b)\right)$$

$$= \varepsilon\text{-closure}(\phi \cup \phi \cup 7)$$

$$= \varepsilon\text{-closure}(7)$$

$$= \{3, 4, 6, 7, 8, 9\}$$

$$\delta(4, a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(4, \varepsilon), a\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(4), a)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(4, a)\right)$$

$$= \varepsilon\text{-closure}(5)$$

$$= \{3, 4, 5, 6, 8, 9\}$$

$$\delta(2, b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(2,\varepsilon), b\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(2), b)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(2, 3, 4, 6, 9), b\right)$$

$$= \varepsilon\text{-closure}\left(\delta(2, b) \cup \delta(3, b) \cup \delta(4, b) \cup \delta(6, b) \cup \delta(9, b)\right)$$

$$= \varepsilon\text{-closure}(\phi \cup \phi \cup \phi \cup 7 \cup \phi)$$

$$= \varepsilon\text{-closure}(7)$$

$$= \{3, 4, 6, 7, 8, 9\}$$

$$\delta(3, a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(3,\varepsilon), a\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(3), a)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, 4, 6), a\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, a) \cup \delta(4, a) \cup \delta(6, a)\right)$$

$$= \varepsilon\text{-closure}(\phi \cup 5 \cup \phi)$$

$$= \varepsilon\text{-closure}(5)$$

$$= \{3, 4, 5, 6, 8, 9\}$$

$$\delta(3, b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(3,\varepsilon), b\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(3), b)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, 4, 6), b\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, b) \cup \delta(4, b) \cup \delta(6, b)\right)$$

$$= \varepsilon\text{-closure}(\phi \cup \phi \cup 7)$$

$$= \varepsilon\text{-closure}(7)$$

$$= \{3, 4, 6, 7, 8, 9\}$$

$$\delta(4, a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(4,\varepsilon), a\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(4), a)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(4, a)\right)$$

$$= \varepsilon\text{-closure}(5)$$

$$= \{3, 4, 5, 6, 8, 9\}$$

$$\delta'(4, b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(4, \varepsilon), b\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(4), b)\right)$$

$$= \varepsilon\text{-closure}(\delta(4, b))$$

$$= \varepsilon\text{-closure}(\phi)$$

$$= \phi$$

$$\delta'(5, a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(5, \varepsilon), a\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(5), a)\right)$$

$$= \varepsilon\text{-closure}(\delta(3, 4, 5, 6, 8, 9), a)$$

$$= \varepsilon\text{-closure}(\delta(3, a) \cup \delta(4, a) \cup \delta(5, a) \cup \delta(6, a) \cup \delta(8, a) \cup \delta(9, a))$$

$$= \varepsilon\text{-closure}(\phi \cup 5 \cup \phi \cup \phi \cup \phi \cup \phi)$$

$$= \varepsilon\text{-closure}(5)$$

$$= \{3, 4, 5, 6, 8, 9\}$$

$$\delta'(5, b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(5, \varepsilon), b\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(5), b)\right)$$

$$= \varepsilon\text{-closure}(\delta(3, 4, 5, 6, 8, 9), b)$$

$$= \varepsilon\text{-closure}(\delta(3, b) \cup \delta(4, b) \cup \delta(5, b) \cup \delta(6, b) \cup \delta(8, b) \cup \delta(9, b))$$

$$= \varepsilon\text{-closure}(\phi \cup \phi \cup \phi \cup 7 \cup \phi \cup \phi)$$

$$= \varepsilon\text{-closure}(7)$$

$$= \{3, 4, 6, 7, 8, 9\}$$

$$\delta'(6, a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(6, \varepsilon), a\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(6), a)\right)$$

$$= \varepsilon\text{-closure}(\delta(6, a))$$

$$= \varepsilon\text{-closure}(\phi)$$

$$= \phi$$

$$\delta(6, b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(6, \varepsilon), b\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(6), b)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(6, b)\right)$$

$$= \varepsilon\text{-closure}(7)$$

$$= \{3, 4, 6, 7, 8, 9\}$$

$$\delta(7, a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(7, \varepsilon), a\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(7), a)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, 4, 6, 7, 8, 9), a\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, a) \cup \delta(4, a) \cup \delta(6, a) \cup \delta(7, a) \cup \delta(8, a) \cup \delta(9, a)\right)$$

$$= \varepsilon\text{-closure}(\phi \cup 5 \cup \phi \cup \phi \cup \phi)$$

$$= \varepsilon\text{-closure}(5)$$

$$= \{3, 4, 5, 6, 8, 9\}$$

$$\delta(7, b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(7, \varepsilon), b\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(7), b)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, 4, 6, 7, 8, 9), b\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, b) \cup \delta(4, b) \cup \delta(6, b) \cup \delta(7, b) \cup \delta(8, b) \cup \delta(9, b)\right)$$

$$= \varepsilon\text{-closure}(\phi \cup \phi \cup 7 \cup \phi \cup \phi \cup \phi)$$

$$= \varepsilon\text{-closure}(7)$$

$$= \{3, 4, 6, 7, 8, 9\}$$

$$\delta(8, a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(8, \varepsilon), a\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(8), a)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, 4, 6, 8, 9), a\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, a) \cup \delta(4, a) \cup \delta(6, a) \cup \delta(8, a) \cup \delta(9, a)\right)$$

$$= \varepsilon\text{-closure}(\phi \cup 5 \cup \phi \cup \phi \cup \phi)$$

$$= \varepsilon\text{-closure}(5)$$

$$= \{3, 4, 5, 6, 8, 9\}$$

$$\delta(8, b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(8, \varepsilon), b\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(8), b)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, 4, 6, 8, 9), b\right)$$

$$= \varepsilon\text{-closure}\left(\delta(3, b) \cup \delta(4, b) \cup \delta(6, b) \cup \delta(8, b) \cup \delta(9, b)\right)$$

$$= \varepsilon\text{-closure}(\phi \cup \phi \cup 7 \cup \phi \cup \phi)$$

$$= \varepsilon\text{-closure}(7)$$

$$= \{3, 4, 6, 7, 8, 9\}$$

$$\delta(9, a) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(9, \varepsilon), a\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(9), a)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(9, a)\right)$$

$$= \varepsilon\text{-closure}(\phi)$$

$$= \phi$$

$$\delta(9, b) = \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(9, \varepsilon), b\right)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(\varepsilon\text{-closure}(9), b)\right)$$

$$= \varepsilon\text{-closure}\left(\delta(9, b)\right)$$

$$= \varepsilon\text{-closure}(\phi)$$

$$= \phi$$

Now we will build the transition table using above calculated δ transitions.

State \ Input	a	b
{0}	{1}	ϕ
{1}	ϕ	{2, 3, 4, 6, 9}
{2}	{3, 4, 5, 6, 8, 9}	{3, 4, 6, 7, 8, 9}
{3}	{3, 4, 5, 6, 8, 9}	{3, 4, 6, 7, 8, 9}

{4}	{3, 4, 5, 6, 8, 9}	ϕ
(5)	{3, 4, 5, 6, 8, 9}	{3, 4, 6, 7, 8, 9}
{6}	ϕ	{3, 4, 6, 7, 8, 9}
(7)	{3, 4, 5, 6, 8, 9}	{3, 4, 6, 7, 8, 9}
(8)	{3, 4, 5, 6, 8, 9}	{3, 4, 6, 7, 8, 9}
(9)	ϕ	ϕ

State {2} is a final state since ε - closure (2) contains 9 which is actually a final state in given NFA. Similarly state 5, 7, 8 and 9 are final states.

Example 2.5 : *Convert the following NFA with ε to NFA without ε.*

Fig. 2.8

Solution : We will first obtain ε - closures of q_0, q_1 and q_2 as follows.

$$\varepsilon\text{ - closure }(q_0) = \{q_0\}$$

$$\varepsilon\text{ - closure }(q_1) = \{q_1, q_2\}$$

$$\varepsilon\text{ - closure }(q_2) = \{q_2\}$$

Now the δ' transitions on each input symbol is obtained as

$$\begin{aligned}
\delta'(q_0, a) &= \varepsilon\text{ - closure}\big(\delta(\hat{\delta}(q_0, \varepsilon), a)\big) \\
&= \varepsilon\text{ - closure}\big(\delta(\varepsilon\text{ - closure}(q_0), a)\big) \\
&= \varepsilon\text{ - closure}\big(\delta(q_0, a)\big) \\
&= \varepsilon\text{ - closure}(q_1) \\
&= \{q_1, q_2\} \\
\delta'(q_0, b) &= \varepsilon\text{ - closure}\big(\delta(\hat{\delta}(q_0, \varepsilon), b)\big) \\
&= \varepsilon\text{ - closure}\big(\delta(\varepsilon\text{ - closure}(q_0), b)\big) \\
&= \varepsilon\text{ - closure}\big(\delta(q_0, b)\big) \\
&= \phi
\end{aligned}$$

$$
\begin{aligned}
\delta'(q_1, a) &= \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(q_1, \varepsilon), a\right)\right) \\
&= \varepsilon\text{-closure}\left(\delta\left(\varepsilon\text{-closure}(q_1), a\right)\right) \\
&= \varepsilon\text{-closure}\left(\delta(q_1, q_2), a\right) \\
&= \varepsilon\text{-closure}\left(\delta(q_1, a) \cup \delta(q_2, a)\right) \\
&= \varepsilon\text{-closure}(\phi \cup \phi) \\
&= \phi \\
\delta'(q_1, b) &= \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(q_1, \varepsilon), b\right)\right) \\
&= \varepsilon\text{-closure}\left(\delta\left(\varepsilon\text{-closure}(q_1), b\right)\right) \\
&= \varepsilon\text{-closure}\left(\delta(q_1, q_2), b\right) \\
&= \varepsilon\text{-closure}\left(\delta(q_1, b) \cup \delta(q_2, b)\right) \\
&= \varepsilon\text{-closure}(\phi \cup q_2) \\
&= \varepsilon\text{-closure}(q_2) \\
&= \{q_2\} \\
\delta'(q_2, a) &= \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(q_2, \varepsilon), a\right)\right) \\
&= \varepsilon\text{-closure}\left(\delta\left(\varepsilon\text{-closure}(q_2), a\right)\right) \\
&= \varepsilon\text{-closure}\left(\delta(q_2, a)\right) \\
&= \varepsilon\text{-closure}(\phi) \\
&= \phi \\
\delta'(q_2, b) &= \varepsilon\text{-closure}\left(\delta\left(\hat{\delta}(q_2, \varepsilon), b\right)\right) \\
&= \varepsilon\text{-closure}\left(\delta\left(\varepsilon\text{-closure}(q_2), b\right)\right) \\
&= \varepsilon\text{-closure}\left(\delta(q_2, b)\right) \\
&= \varepsilon\text{-closure}(q_2) \\
&= \{q_2\}
\end{aligned}
$$

The transition table can be -

State \ Input	a	b
→ q_0	$\{q_1, q_2\}$	ϕ
q_1	ϕ	$\{q_2\}$
q_2	ϕ	$\{q_2\}$

States q_1 and q_2 becomes the final as ε - closures of q_1 and q_2 contains the final state q_2. The NFA can be shown by following transition diagram -

Fig. 2.9

Example 2.6 : *Convert the given NFA with ε to ordinary NFA.*

Solution :

Fig. 2.10

We will first obtain ε - closure of each state.

$$\varepsilon \text{ - closure } (q_0) = \{q_0, q_1, q_2\}$$

$$\varepsilon \text{ - closure } (q_1) = \{q_1, q_2\}$$

$$\varepsilon \text{ - closure } (q_2) = \{q_2\}$$

Now δ' transitions for each state on each input.

$$\delta'(q_0, a) = \varepsilon \text{ - closure } \left(\delta(\delta(q_0, \varepsilon), a)\right)$$

$$= \varepsilon \text{ - closure } (\delta(\varepsilon \text{ - closure } (q_0), a))$$

$$= \varepsilon \text{ - closure } \left(\delta\left((q_0, q_1, q_2), a\right)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)\right)$$

$$= \varepsilon \text{ - closure } (\phi \cup q_1 \cup q_1)$$

$$= \varepsilon \text{ - closure } (q_1)$$

$$= \{q_1, q_2\}$$

$$\delta^*(q_0, b) = \varepsilon \text{ - closure } \left(\delta(\hat{\delta}(q_0, \varepsilon), b)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta(\varepsilon \text{ - closure } (q_0), b)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta\left((q_0, q_1, q_2), b\right)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)\right)$$

$$= \varepsilon \text{ - closure } (q_0 \cup \phi \cup q_0)$$

$$= \varepsilon \text{ - closure } (q_0)$$

$$= \{q_0, q_1, q_2\}$$

$$\delta^*(q_1, a) = \varepsilon \text{ - closure } \left(\delta(\hat{\delta}(q_1, \varepsilon), a)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta(\varepsilon \text{ - closure } (q_1), a)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta(\{q_1, q_2\}, a)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta(q_1, a) \cup \delta(q_2, a)\right)$$

$$= \varepsilon \text{ - closure } (q_1 \cup q_1)$$

$$= \varepsilon \text{ - closure } (q_1)$$

$$= \{q_1, q_2\}$$

$$\delta^*(q_1, b) = \varepsilon \text{ - closure } \left(\delta(\hat{\delta}(q_1, \varepsilon), b)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta(\varepsilon \text{ - closure } (q_1), b)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta\left((q_1, q_2), b\right)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta(q_1, b) \cup \delta(q_2, b)\right)$$

$$= \varepsilon \text{ - closure } (\phi \cup q_0)$$

$$= \{q_0, q_1, q_2\}$$

$$\delta^*(q_2, a) = \varepsilon \text{ - closure } \left(\delta(\hat{\delta}(q_2, \varepsilon), a)\right)$$

$$= \varepsilon \text{ - closure } \left(\delta(\varepsilon \text{ - closure } (q_2), a)\right)$$

$$= \varepsilon \text{ - closure } (\delta(q_2, a))$$

$$= \varepsilon \text{ - closure } (q_1)$$

$$= \{q_1, q_2\}$$

$$\delta'(q_2, b) = \varepsilon \text{ - closure } \left(\delta(\hat{\delta}(q_2, \varepsilon), b)\right)$$

$$= \varepsilon \text{ - closure } (\delta(\varepsilon \text{ - closure } (q_2), b))$$

$$= \varepsilon \text{ - closure } (\delta(q_2, b))$$

$$= \varepsilon \text{ - closure } (q_0)$$

$$= \{q_0, q_1, q_2\}$$

The transition table can be -

		a	b
→	q_0	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
	q_1	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
	q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

2.4 NFA to DFA Conversion

As we have discussed, the finite automata can either be DFA or NFA. You might be thinking now, who is better NFA or DFA. Which has more power ? Here is a theorem which tell you that any NFA can be converted to its equivalent DFA. That is any language L acceptable by NFA can be acceptable by its equivalent DFA. The basic idea in this theorem is that, DFA keeps track of all the states, that NFA could be in reading the same input as the DFA has read.

Let us see the theorem.

Theorem : Let L be a set accepted by non-deterministic finite automation. Then there exists a deterministic finite automation that accepts L.

Proof : Let

$M = (Q, \Sigma, \delta, q_0, F)$ be an NFA for language L. Then define DFA M' such that

$$M' = (Q', \Sigma, \delta', q_0', F')$$

The states of M' are all the subset of M'. The $Q' = 2^Q$.

F' be the set of all the final states in M.

The elements in Q' will be denoted by $[q_1, q_2, q_3, \ldots q_i]$ and the elements in Q are denoted by $\{q_0, q_1, q_2, \ldots\}$. The $[q_1, q_2, \ldots q_i]$ will be assumed as one state in Q' if in the NFA q_0 is a initial state it is denoted in DFA as $q_0' = [q_0]$. We define,

$$\delta'([q_1, q_2, q_3, \ldots q_i], a) = [p_1, p_2, p_3, \ldots p_j]$$

if and only if,

$$\delta(\{q_1, q_2, q_3, \ldots q_i\}, a) = \{p_1, p_2, p_3, \ldots p_j\}$$

This means that whenever in NFA, at the current states $\{q_1, q_2, q_3, \ldots q_i\}$ if we get input a and it goes to the next states $\{p_1, p_2, \ldots p_j\}$ then while constructing DFA for it the current state is assumed to be $[q_1, q_2, q_3, \ldots q_i]$. At this state, the input is a and the next is assumed to be $[p_1, p_2, \ldots p_j]$. On applying δ function on each of the states $q_1, q_2, q_3, \ldots q_i$ the new states may be any of the states from $[p_1, p_2, \ldots p_j]$. The theorem can be proved with the induction method by assuming length of input string x

$$\delta'(q_0', x) = [q_1, q_2, \ldots q_i]$$

if and only if,

$$\delta(q_0, x) = \{q_1, q_2, q_3, \ldots q_i\}$$

Basis : If length of input string is 0

i.e. $|x| = 0$, that means x is ε then $q_0' = [q_0]$

Induction : If we assume that the hypothesis is true for the input string of length m or less than m. Then if x a is a string of length m+1. Then the function δ' could be written as

$$\delta'(q_0', xa) = \delta'(\delta'(q_0, x), a)$$

By the induction hypothesis,

$$\delta'(q_0', x) = [p_1, p_2, \ldots p_j]$$

if and only if,

$$\delta(q_0, x) = \{p_1, p_2, p_3, \ldots p_j\}$$

By definition of δ'

$$\delta'([p_1, p_2, \ldots p_j], a) = [r_1, r_2, \ldots r_k]$$

if and only if,

$$\delta(\{p_1, p_2, \ldots p_j\}, a) = \{r_1, r_2, \ldots r_k\}$$

Thus

$$\delta'(q_0', xa) = [r_1, r_2, \ldots r_k]$$

if and only if

$$\delta(q_0, x\ a) = \{r_1, r_2, \ldots r_k\}$$

is shown by inductive hypothesis.

Thus $\quad L(M) = L(M')$

With the help of this theorem, let us try to solve few examples.

Conversion from NFA to DFA

We will discuss the method of converting NFA to its equivalent DFA.

Let, $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA which accepts the language L(M). There should be equivalent DFA denoted by $M' = (Q', \Sigma', \delta', q_0', F')$ such that $L(M) = L(M')$.

The conversion method will follow following steps -

1) The start state of NFA M will be the start for DFA M'. Hence add q_0 of NFA (start state) to Q'. Then find the transitions from this start state.
2) For each state $[q_1, q_2, q_3, \ldots\ldots q_i]$ in Q' the transitions for each input symbol Σ can be obtained as,
   i) $\delta'([q_1, q_2, \ldots\ldots q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \ldots\ldots \delta(q_i, a)$
   $= [q_1, q_2, \ldots\ldots q_k]$ may be some state.
   ii) Add the state $[q_1, q_2, \ldots\ldots q_k]$ to DFA if it is not already added in Q'.
   iii) Then find the transitions for every input symbol from Σ for state $[q_1, q_2, \ldots\ldots q_k]$. If we get some state $[q_1, q_2, \ldots\ldots q_n]$ which is not in Q' of DFA then add this state to Q'.
   iv) If there is no new state generating then stop the process after finding all the transitions.
3) For the state $[q_1, q_2, \ldots\ldots q_n] \in Q'$ of DFA if any one state q_i is a final state of NFA then $[q_1, q_2, \ldots\ldots q_n]$ becomes a final state. Thus the set of all the final states $\in F'$ of DFA.

Example 2.7 : *Let* $M = (\{q_0, q_1\}, \{0,1\}, \delta, q_0, \{q_1\})$

be NFA where $\delta(q_0, 0) = \{q_0, q_1\}, \delta(q_0, 1) = \{q_1\}$

$\delta(q_1, 0) = \phi, \delta(q_1, 1) = \{q_0, q_1\}$

Construct its equivalent DFA.

Solution : Let the DFA $M' = (Q', \Sigma, \delta', q_0', F')$

Now, the δ' function will be computed as follows -

As $\delta(q_0, 0) = \{q_0, q_1\}$ $\delta'([q_0], 0) = [q_0, q_1]$

As in NFA the initial state is q_0, the DFA will also contain the initial state $[q_0]$.

Let us draw the transition table for δ function for a given NFA.

		State \ Input	0	1		
$\delta(q_0, 0)$	$\Rightarrow$	$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$	$\Leftarrow$	$\delta(q_0, 1)$
$\delta(q_1, 0)$	$\Rightarrow$	q_1	ϕ	$\{q_0, q_1\}$	$\Leftarrow$	$\delta(q_1, 1)$

δ Function for NFA

From the transition table we can compute that there are $[q_0]$, $[q_1]$, $[q_0, q_1]$ states for its equivalent DFA. We need to compute the transition from state $[q_0, q_1]$.

$$\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$$

$$= \{q_0, q_1\} \cup \phi$$

$$= \{q_0, q_1\}$$

So, $\delta'([q_0, q_1], 0) = [q_0, q_1]$

Similarly,

$$\delta(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1)$$

$$= \{q_1\} \cup \{q_0, q_1\}$$

$$= \{q_0, q_1\}$$

So, $\delta'([q_0, q_1], 1) = [q_0, q_1]$

As in the given NFA q_1 is a final state, then in DFA wherever q_1 exists that state becomes a final state. Hence in the DFA final states are $[q_1]$ and $[q_0, q_1]$. Therefore set of final states $F = \{[q_1], [q_0, q_1]\}$.

The equivalent DFA is -

State \ Input	0	1
$\rightarrow q_0$	$[q_0, q_1]$	$[q_1]$
$[q_1]$	ϕ	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

Transition table for equivalent DFA

Fig. 2.11

Even we can change the names of the states of DFA.

$$A = [q_0]$$

$$B = [q_1]$$

$$C = [q_0, q_1]$$

With these new names the DFA will be as follows -

Fig. 2.12 An equivalent DFA

Example 2.8 : *Convert the given NFA to DFA.*

State \ Input	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	q_2	q_1
q_2	q_3	q_3
q_3	ϕ	q_2

Solution : As we know, first we will compute δ' function.

$$\delta(\{q_0\}, 0) = \{q_0, q_1\}$$

Hence $\delta'([q_0], 0) = [q_0, q_1]$

Similarly,

$$\delta(\{q_0\}, 1) = \{q_0\}$$

Hence $\delta'([q_0], 1) = [q_0]$

Thus we have got a new state $[q_0, q_1]$.

Let us check how it behaves on input 0 and 1.

So,

$$\begin{aligned}\delta'([q_0, q_1], 0) &= \delta([q_0], 0) \cup \delta([q_1], 0)\\ &= \{q_0, q_1\} \cup \{q_2\}\\ &= \{q_0, q_1, q_2\}\end{aligned}$$

Hence a new state is generated i.e. $[q_0, q_1, q_2]$

Similarly

$$\begin{aligned}\delta'([q_0, q_1], 1) &= \delta([q_0], 1) \cup \delta([q_1], 1)\\ &= \{q_0\} \cup \{q_1\}\\ &= \{q_0, q_1\}\end{aligned}$$

No new state is generated here.

Again δ' function will be computed for $[q_0, q_1, q_2]$, the new state being generated.

State	0	1
$[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$

As, you have observed in the above table for a new state $[q_0, q_1, q_2]$ the input 0 will give a new state $[q_0, q_1, q_2, q_3]$ and input 1 will give a new state $[q_0, q_1, q_3]$, because

$$\begin{aligned}\delta([q_0, q_1, q_2], 0) &= \delta([q_0], 0) \cup \delta([q_1], 0) \cup \delta([q_2], 0) \\ &= \{q_0, q_1\} \cup \{q_2\} \cup \{q_3\} \\ &= \{q_0, q_1, q_2, q_3\} \\ &= [q_0, q_1, q_2, q_3]\end{aligned}$$

Same procedure for input 1. Thus the final DFA is as given below.

State	0	1
$\rightarrow [q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_1]$	$[q_2]$	$[q_1]$
$[q_2]$	$[q_3]$	$[q_3]$
$[q_3]$	ϕ	$[q_2]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$

DFA for example 2.8

Example 2.9 :*Construct DFA equivalent to the given NFA.*

State \ Input	0	1
→ p	{p, q}	p
q	r	r
r	s	–
s	s	s

Solution : The NFA M = {{p, q, r, s}, {0, 1}, δ {p}, {s}}

The equivalent DFA will be constructed.

State \ Input	0	1
→ [p]	[p, q]	[p]
[q]	[r]	[r]
[r]	[s]	–
[s]	[s]	[s]
[p, q]	[p, q, r]	[p, r]

Continuing with the generated new states.

State \ Input	0	1
→ [p]	[p, q]	[p]
[q]	[r]	[r]
[r]	[s]	–
[s]	[s]	[s]
[p, q]	[p, q, r]	[p, r]
[p, q, r]	[p, q, r, s]	[p, r]
[p, r]	[p, q, s]	[p]
[p, q, r, s]	[p, q, r, s]	[p, r, s]
[p, q, s]	[p, q, r, s]	[p, r, s]
[p, r, s]	[p, q, s]	[p, s]
[p, s]	[p, q, s]	[p, s]

Fig. 2.14

Example 2.11 : *Convert the following NFA into DFA.*

Fig. 2.17

Solution : We will first design a transition table from given transition diagram.

State \ Input	0	1
q_0	$\{q_1, q_2\}$	ϕ
q_1	$\{q_1, q_2\}$	ϕ
q_2	ϕ	$\{q_1, q_3\}$
q_3	$\{q_1, q_2\}$	ϕ

Now, $\delta(q_0, 0) = \{q_1, q_2\}$

We can write this as

$$\delta([q_0], 0) = [q_1, q_2]$$

Here $[q_1, q_2]$ becomes one newly generated state when 0 input is received in state $[q_0]$

Similarly

$$\delta([q_0], 1) = \phi \quad \text{No state getting generated.}$$

$$\delta(q_1, 0) = \{q_1, q_2\} \text{ we can write it as}$$

$$\delta([q_1], 0) = [q_1, q_2]$$

$$\delta([q_1], 1) = \phi$$

$$\delta([q_2], 0) = \phi$$

$$\delta(q_2, 1) = \{q_1, q_3\}$$

i.e. $\delta([q_2], 1) = [q_1, q_3]$ again a new state $[q_1, q_3]$ gets generated.

Similarly,

$$\delta([q_3], 0) = [q_1, q_2]$$

$$\delta([q_3], 1) = \phi$$

Now we will obtain δ transitions on newly generated states $[q_1, q_2]$ and $[q_1, q_3]$.

$$\begin{aligned}\delta([q_1, q_2], 0) &= \delta([q_1], 0) \cup \delta([q_2], 0) \\ &= \{q_1, q_2\} \cup \phi \\ &= \{q_1, q_2\}\end{aligned}$$

$$\therefore \quad \delta([q_1, q_2], 0) = [q_1, q_2]$$

Similarly,

$$\begin{aligned}\delta([q_1, q_2], 1) &= \delta([q_1], 1) \cup \delta([q_2], 1) \\ &= \phi \cup \{q_1, q_3\} \\ &= \{q_1, q_3\}\end{aligned}$$

$$\therefore \quad \delta([q_1, q_2], 1) = [q_1, q_3]$$

$$\begin{aligned}\text{Now, } \delta([q_1, q_3], 0) &= \delta([q_1], 0) \cup \delta([q_3], 0) \\ &= \{q_1, q_2\} \cup \{q_1, q_2\} \\ &= \{q_1, q_2\}\end{aligned}$$

$$\therefore \quad \delta([q_1, q_3], 0) = [q_1, q_2]$$

Similarly,

$$\delta([q_1, q_3], 1) = \delta([q_1], 1) \cup \delta([q_3], 1)$$
$$= \phi \cup \phi$$
$$= \phi$$

$$\therefore \quad \delta([q_1, q_3], 1) = \phi$$

As now no new states are getting generated.

The transition table for DFA using above δ transitions i

State \ Input	0	1
→ $[q_0]$	$[q_1, q_2]$	ϕ
$[q_1]$	$[q_1, q_2]$	ϕ
$[q_2]$	ϕ	$[q_1, q_3]$
$[q_3]$	$[q_1, q_2]$	ϕ
$[q_1, q_2]$	$[q_1, q_2]$	$[q_1, q_3]$
$[q_1, q_3]$	$[q_1, q_2]$	ϕ

The transition diagram can be -

Fig. 2.18

As we can see the marked part of DFA consisting of q_1, q_2, q_3 is of no use since there is no path from start state to these states. Hence we will never reach to these states. So we can eliminate these states. Hence new DFA becomes.

Fig. 2.19

As states q_0 and q_1 are final states original NFA states containing q_1 i.e. $[q_1, q_3]$ and $[q_1, q_2]$ are also final states.

Example 2.12 : *Convert the following NFA into its equivalent DFA. Also recognize the language generated by this DFA.*

Fig. 2.20

Solution : We will first desing the transition table from given transition diagram.

State \ Input	0	1
→ q_0	$\{q_0, q_2\}$	q_1
q_1	q_3	q_4
q_2	ϕ	q_4
q_3	q_3	ϕ
q_4	q_3	q_3

$\delta([q_0], 0) = [q_0, q_2]$ new state generated

$\delta([q_0], 1) = [q_1]$

$\delta([q_1], 0) = [q_3]$

$\delta([q_1], 1) = [q_4]$

$\delta([q_2], 0) = \phi$

$\delta([q_2], 1) = [q_4]$

$\delta([q_3], 0) = [q_3]$

$\delta([q_3], 1) = \phi$

$\delta([q_4], 0) = [q_3]$

$\delta([q_4], 1) = [q_3]$

Now we will obtain input transitions on $[q_0, q_2]$.

$$\begin{aligned}\delta([q_0, q_2], 0) &= \delta([q_0], 0) \cup \delta([q_2], 0)\\ &= \{q_0, q_2\} \cup \phi\\ &= \{q_0, q_2\}\end{aligned}$$

i.e. $\delta([q_0, q_2], 0) = [q_0, q_2]$

Similarly,

$$\begin{aligned}\delta([q_0, q_2], 1) &= \delta([q_0], 1) \cup \delta([q_2], 1)\\ &= [q_1, q_4] \text{ new state generated.}\end{aligned}$$

We will obtain δ transitions for new state $[q_1, q_4]$.

$$\begin{aligned}\delta([q_1, q_4], 0) &= \delta([q_1], 0) \cup \delta([q_4], 0)\\ &= \{q_3\} \cup \{q_3\}\\ &= \{q_3\}\end{aligned}$$

$\therefore \quad \delta([q_1, q_4], 0) = [q_3]$

Now $\delta([q_1, q_4], 1) = \delta([q_1], 1) \cup \delta([q_4], 1)$

$$\begin{aligned}&= \{q_4\} \cup \{q_3\}\\ &= \{q_4, q_3\}\end{aligned}$$

$\therefore \quad \delta([q_1, q_4], 1) = [q_4, q_3]$ new state generated.

We will obtain δ transitions for new state $[q_4, q_3]$

$$\delta([q_4, q_3], 0) = \delta([q_4], 0) \cup \delta([q_3], 0)$$
$$= \{q_3\} \cup \{q_3\}$$
$$= \{q_3\}$$
$$\therefore \quad \delta([q_4, q_3], 0) = [q_3]$$

Similarly,

$$\delta([q_4, q_3], 1) = \delta([q_4], 1) \cup \delta([q_3], 1)$$
$$= \{q_3\} \cup \phi$$
$$= \{q_3\}.$$
$$\therefore \quad \delta([q_4, q_3], 1) = [q_3]$$

Now since no new state is being generated.

We will now design the transition diagrams from the above δ transitions.

State \ Input	0	1
→ $[q_0]$	$[q_0, q_2]$	$[q_1]$
$[q_1]$	$[q_3]$	$[q_4]$
$[q_2]$	ϕ	$[q_4]$
$[q_3]$	$[q_3]$	ϕ
$[q_4]$	$[q_3]$	$[q_3]$
$[q_0, q_2]$	$[q_0, q_2]$	$[q_1, q_4]$
$[q_1, q_4]$	$[q_3]$	$[q_4, q_3]$
$[q_4, q_3]$	$[q_3]$	$[q_3]$

The states q_0 and q_4 are basically final states. Hence newly generated states containing q_4 are final states. Therefore $[q_1, q_4]$ and $[q_4, q_3]$ are final states. Now the transition diagram for DFA are -

Fig. 2.21

The DFA will then be -

Fig. 2.22

This is a language accepting '11' or 0^n 11 where $n \geq 0$. That means it is a language of any number of zeros followed by 11 or 1.

Example 2.13 : *For the following state transition table draw the state transition diagram. Find its equivalent machine. For the string abbaaab test whether both give same result or not. q_0 is initial state and q_3 is a final state.*

q/Σ	0	1
q_0	q_1	q_2
q_1	q_1	q_1, q_3
q_2	ϕ	ϕ
q_3	q_0, q_3	q_3

Solution : The transition diagram for the given transition table can be -

Fig. 2.23

This is clearly a non deterministic Finite Automata.

Now we will convert it to an equivalent DFA.

$$\delta(q_0, 0) = \{q_1\}$$

$$\therefore \quad \delta([q_0], 0) = [q_1]$$

Similarly, $\delta([q_0], 1) = [q_2]$

$$\delta([q_1], 0) = \{q_1\}$$

$$= [q_1]$$

$$\delta([q_1], 1) = \{q_1, q_3\}$$

$$= [q_1, q_3] \leftarrow \text{new state generated.}$$

$$\delta([q_2], 0) = \phi$$

$$\delta([q_2], 1) = \phi$$

$$\delta([q_3], 0) = [q_0, q_3] \leftarrow \text{new state generated.}$$

$$\delta([q_3], 1) = [q_3]$$

Now we will obtain δ' transitions on the new states generated.

$$\begin{aligned}
\delta'([q_1, q_3], 0) &= \delta([q_1], 0) \cup \delta([q_3], 0) \\
&= \{q_1\} \cup \{q_0, q_3\} \\
&= \{q_0, q_1, q_3\} \\
&= [q_0, q_1, q_3] \leftarrow \text{new state generated.} \\
\delta'([q_1, q_3], 1) &= \delta([q_1], 1) \cup \delta([q_3], 1) \\
&= \{q_1, q_3\} \cup \{q_3\} \\
&= \{q_1, q_3\} \\
&= [q_1, q_3] \\
\delta'([q_0, q_3], 0) &= \delta([q_0], 0) \cup \delta([q_3], 0) \\
&= \{q_1\} \cup \{q_0, q_3\} \\
&= \{q_0, q_1, q_3\} \\
&= [q_0, q_1, q_3] \\
\delta'([q_0, q_3], 1) &= \delta([q_0], 1) \cup \delta([q_3], 1) \\
&= \{q_2\} \cup \{q_3\} \quad \leftarrow \text{new state generated.} \\
&= [q_2, q_3]
\end{aligned}$$

Again we get two more new states $[q_0, q_1, q_3]$ and $[q_2, q_3]$. Hence we will obtain δ' transitions on them

$$\begin{aligned}
\delta'([q_0, q_1, q_3], 0) &= \delta([q_0], 0) \cup \delta([q_1], 0) \cup \delta([q_3], 0) \\
&= \{q_1\} \cup \{q_1\} \cup \{q_0, q_3\} \\
&= \{q_0, q_1, q_3\} \\
&= [q_0, q_1, q_3] \\
\delta'([q_0, q_1, q_3], 1) &= \delta([q_0], 1) \cup \delta([q_1], 1) \cup \delta([q_3], 1) \\
&= \{q_2\} \cup \{q_1\} \cup \{q_3\} \\
&= [q_1, q_2, q_3] \leftarrow \text{new state generated} \\
\delta'([q_2, q_3], 0) &= \delta([q_2], 0) \cup \delta([q_3], 0) \\
&= \phi \cup \{q_0, q_3\} \\
&= [q_0, q_3] \\
\delta'([q_2, q_3], 1) &= \delta([q_2], 1) \cup \delta([q_3], 1)
\end{aligned}$$

$$= \phi \cup \{q_3\}$$
$$= \{q_3\}$$
$$= [q_3]$$

The δ' transition on new state $[q_1, q_2, q_3]$ is -

$$\delta'([q_1, q_2, q_3], 0) = \delta([q_1], 0) \cup \delta([q_2], 0) \cup \delta([q_3], 0)$$
$$= \{q_1\} \cup \phi \cup \{q_0, q_3\}$$
$$= \{q_0, q_1, q_3\}$$
$$= [q_0, q_1, q_3]$$
$$\delta'([q_1, q_2, q_3], 1) = \delta([q_1], 1) \cup \delta([q_2], 1) \cup \delta([q_3], 1)$$
$$= \{q_1, q_3\} \cup \phi \cup \{q_3\}$$
$$= \{q_1, q_3\}$$
$$= [q_1, q_3]$$

From the above computed δ' transitions we can build the transition table as -

	q/Σ	0	1
→	$[q_0]$	$[q_1]$	$[q_2]$
	$[q_1]$	$[q_1]$	$[q_1, q_3]$
	$[q_2]$	ϕ	ϕ
*	$[q_3]$	$[q_0, q_3]$	$[q_3]$
*	$[q_1, q_3]$	$[q_0, q_1, q_3]$	$[q_1, q_3]$
*	$[q_0, q_3]$	$[q_0, q_1, q_3]$	$[q_2, q_3]$
*	$[q_2, q_3]$	$[q_0, q_3]$	$[q_3]$
*	$[q_0, q_1, q_3]$	$[q_0, q_1, q_3]$	$[q_1, q_2, q_3]$
*	$[q_1, q_2, q_3]$	$[q_0, q_1, q_3]$	$[q_1, q_3]$

The DFA can be represented using following transition diagram.

Fig. 2.24

To simulate string abbaaab we use above DFA. Assume input 0 = a and input 1 = b. Then

$$\delta(q_0, \text{abbaaab}) \vdash a\, q_1\, \text{bbaaab}$$
$$\vdash \text{ab}\,(q_1, q_3)\,\text{bbaaab}$$
$$\vdash \text{abb}(q_1, q_3)\,\text{baaab}$$
$$\vdash \text{abbb}(q_1, q_3)\text{aaab}$$
$$\vdash \text{abbba}(q_0, q_1, q_3)\,\text{aab}$$
$$\vdash \text{abbbaa}\,(q_0, q_1, q_3)\,\text{ab}$$
$$\vdash \text{abbbaaa}\,(q_0, q_1, q_3)\,\text{b}$$
$$\vdash \text{abbbaaab}(q_1, q_2, q_3)$$

is in final state.

Thus the given input can be accepted by a DFA.

Example 2.14 : *Convert the given NFA to DFA.*

Fig. 2.25

Solution : Conversion of NFA to DFA -

For the given transition diagram in Fig. 2.25 we will first construct the transition table.

	State / Σ	0	1
→	q_0	q_0	q_1
	q_1	$\{q_1, q_2\}$	q_1
	q_2	$\{q_2\}$	$\{q_1, q_2\}$

Now we will obtain δ transitions for each state.

$$\delta([q_0], 0) = \{q_0\}$$
$$= [q_0]$$
$$\delta([q_0], 1) = \{q_1\}$$
$$= [q_1]$$
$$\delta([q_1], 0) = \{q_1, q_2\}$$
$$= [q_1, q_2] \rightarrow \text{new state generated.}$$
$$\delta([q_1], 1) = \{q_1\} = [q_1]$$

$$\delta([q_2], 0) = \{q_2\}$$
$$= [q_2]$$
$$\delta([q_2], 1) = \{q_1, q_2\}$$
$$= [q_1, q_2]$$

Now we will obtain δ transitions on $[q_1, q_2]$

$$\delta([q_1, q_2], 0) = \delta([q_1], 0) \cup \delta([q_2], 0)$$
$$= \{q_1, q_2\} \cup \{q_2\}$$
$$= \{q_1, q_2\}$$
$$= [q_1, q_2]$$
$$\delta([q_1, q_2], 1) = \delta([q_1], 1) \cup \delta([q_2], 1)$$
$$= \{q_1\} \cup \{q_1, q_2\}$$
$$= [q_1, q_2]$$

The state $[q_1, q_2]$ is final state as well because it contains a final state q_2.The transition table for the constructed DFA will be -

	State / Σ	0	1
→	$[q_0]$	$[q_0]$	$[q_1]$
	$[q_1]$	$[q_1, q_2]$	$[q_1]$
	$[q_2]$	$[q_2]$	$[q_1, q_2]$
	$[q_1, q_2]$	$[q_1, q_2]$	$[q_1, q_2]$

The transition diagram will be -

Fig. 2.26

Consider a string 0010 we will get -

Using NFA

$$q_0\ 0010 \vdash 0q_0 0110$$
$$\vdash 00q_0 110$$
$$\vdash 001q_1 10$$
$$\vdash 0011q_1 0$$
$$\vdash 00110q_2 \text{ or } \vdash 00110q_1$$

Accept non-accept.

Using DFA

$$q_0 00110 \vdash 0q_0 0110$$
$$\vdash 00q_0 110$$
$$\vdash 001q_1 10$$
$$\vdash 0011q_1 0$$

$\vdash$ 00110 $[q_1, q_2]$

Accept state.

Thus both the NFA and DFA accept the same string 00110.

2.4.1 Conversion of NFA with ε to DFA

We have already seen the method of converting NFA with ε to NFA without ε. Now we will learn how to convert NFA with ε to its equivalent DFA. In this method we first convert NFA with ε to NFA without ε. Then the NFA without ε can be converted to its equivalent DFA.

Method for converting NFA with ε to DFA

Step 1 : Consider $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA with ε. We have to convert this NFA with ε to equivalent DFA denoted by

$$M_D = (Q_D, \Sigma, \delta_D, q_0, F_D)$$

Then obtain,

ε - closure$(q_0) = \{p_1, p_2, p_3, \ldots p_n\}$ then $[p_1, p_2, p_3, \ldots p_n]$ becomes a start state of DFA.

Now $[p_1, p_2, p_3, \ldots p_n] \in Q_D$

Step 2 : We will obtain δ transitions on $[p_1, p_2, p_3, \ldots p_n]$ for each input.

$$\delta_D([P_1, P_2, \ldots P_n], a) = \varepsilon\text{ - closure}\big(\delta(P_1, a) \cup \delta(P_2, a) \cup \ldots \delta(P_n, a)\big)$$

$$= \bigcup_{i=1}^{n} \varepsilon\text{ - closure}\,\delta(p_i, a)$$

where a is input $\in \Sigma$.

Step 3 : The states obtained $[p_1, p_2, p_3, \ldots p_n] \in Q_D$. The states containing final state in p_i is a final state in DFA.

Now let us see some examples of conversion based on this procedure.

Example 2.15 : *Convert the following NFA with ε to equivalent DFA.*

Fig. 2.27

Solution : To convert this NFA we will first find ε-closures.

$$\varepsilon\text{-closures } \{q_0\} = \{q_0, q_1, q_2\}$$

$$\varepsilon\text{-closures } \{q_1\} = \{q_1, q_2\}$$

$$\varepsilon\text{-closures } \{q_2\} = \{q_2\}$$

Let us start from ε-closure of start state

$$\varepsilon\text{-closure } \{q_0\} = \{q_0, q_1, q_2\} \text{ we will call this state as A.}$$

Now let us find transitions on A with every input symbol.

$$\begin{aligned}
\delta'(A, a) &= \varepsilon\text{-closure } (\delta(A, a)) \\
&= \varepsilon\text{-closure } (\delta(q_0, q_1, q_2), a) \\
&= \varepsilon\text{-closure } \{\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)\} \\
&= \varepsilon\text{-closure } \{q_1\} \\
&= \{q_1, q_2\}. \text{ Let us call it as state B.}
\end{aligned}$$

$$\begin{aligned}
\delta'(A, b) &= \varepsilon\text{-closure } (\delta(A, b)) \\
&= \varepsilon\text{-closure } (\delta(q_0, q_1, q_2), b) \\
&= \varepsilon\text{-closure } \{\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)\} \\
&= \varepsilon\text{-closure } \{q_0\} \\
&= \{q_0, q_1, q_2\} \text{ i.e. A.}
\end{aligned}$$

Hence we can state that

$$\delta'(A, a) = B$$

$$\delta'(A, b) = A$$

Now let us find transitions for state $B = \{q_1, q_2\}$.

$$\begin{aligned}
\delta'(B, a) &= \varepsilon\text{-closure } (\delta(q_1, q_2), a) \\
&= \varepsilon\text{-closure } \{q_1\} \\
&= \{q_1, q_2\} \text{ i.e. B} \\
\delta'(B, b) &= \varepsilon\text{-closure } (\delta(q_1, q_2), b) \\
&= \varepsilon\text{-closure } \{\delta(q_1, b) \cup \delta(q_2, b)\} \\
&= \varepsilon\text{-closure } \{q_0\} \\
&= \{q_0, q_1, q_2\} \text{ i.e. A.}
\end{aligned}$$

Hence the generated DFA is

	a	b
→ (A)	B	A
(B)	B	A

Fig. 2.28

Example 2.16 : *Convert the given NFA into its equivalent DFA -*

Fig. 2.29

Solution : Let us obtain ε-closure of each state.

$$\varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$$

$$\varepsilon\text{-closure}(q_1) = \{q_1, q_2\}$$

$$\varepsilon\text{-closure}(q_2) = \{q_2\}$$

Now we will obtain δ' transition. Let ε-closure $(q_0) = \{q_0, q_1, q_2\}$ call it as state A.

$$\begin{aligned}\delta'(A,0) &= \varepsilon\text{-closure}\{\delta((q_0, q_1, q_2), 0)\}\\ &= \varepsilon\text{-closure}\{\delta(q_0,0) \cup \delta(q_1,0) \cup \delta(q_2,0)\}\\ &= \varepsilon\text{-closure}\{q_0\}\\ &= \{q_0, q_1, q_2\} \qquad \text{i.e. state A}\end{aligned}$$

$$\begin{aligned}\delta'(A,1) &= \varepsilon\text{-closure}\{\delta((q_0, q_1, q_2), 1)\}\\ &= \varepsilon\text{-closure}\{\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2,1)\}\\ &= \varepsilon\text{-closure}\{q_1\}\\ &= \{q_1, q_2\} \qquad \text{Call it as state B}\\ \delta'(A,2) &= \varepsilon\text{-closure}\{\delta((q_0, q_1, q_2), 2)\}\end{aligned}$$

$$= \varepsilon\text{-closure}\{\delta(q_0,2) \cup \delta(q_1,2) \cup \delta(q_2,2)\}$$
$$= \varepsilon\text{-closure}\{q_2\}$$
$$= \{q_2\} \qquad \text{Call it as state C.}$$

Thus we have obtained

$$\delta'(A,0) = A$$
$$\delta'(A,1) = B$$
$$\delta'(A,2) = C$$

i.e.

Now we will find transitions on states B and C for each input.

Hence

$$\delta'(B,0) = \varepsilon\text{-closure}\{\delta(q_1,q_2),0\}$$
$$= \varepsilon\text{-closure}\{\delta(q_1,0) \cup \delta(q_2,0)\}$$
$$= \varepsilon\text{-closure}\{\phi\}$$
$$= \phi$$
$$\delta'(B,1) = \varepsilon\text{-closure}\{\delta(q_1,q_2),1\}$$
$$= \varepsilon\text{-closure}\{\delta(q_1,1) \cup \delta(q_2,1)\}$$
$$= \varepsilon\text{-closure}\{q_1\}$$
$$= \{q_1,q_2\} \text{ i.e state B itself.}$$
$$\delta'(B,2) = \varepsilon\text{-closure}\{\delta(q_1,q_2),2\}$$
$$= \varepsilon\text{-closure}\{\delta(q_1,2) \cup \delta(q_2,2)\}$$
$$= \varepsilon\text{-closure}\{q_2\}$$
$$= \{q_2\} \text{ i.e state C.}$$

Hence

$$\delta'(B, 0) = \phi$$

$$\delta'(B, 1) = B$$

$$\delta'(B, 2) = C$$

The partial transition diagram will be

Fig. 2.31

Now we will obtain transitions for C :

$$\delta'(C, 0) = \varepsilon\text{-closure}\{\delta(q_2, 0)\}$$

$$= \varepsilon\text{-closure}\{\phi\}$$

$$= \phi$$

$$\delta'(C, 1) = \varepsilon\text{-closure}\{\delta(q_2, 1)\}$$

$$= \varepsilon\text{-closure}\{\phi\}$$

$$= \phi$$

$$\delta'(C, 2) = \varepsilon\text{-closure}\{\delta(q_2, 2)\}$$

$$= q_2$$

Hence the DFA is

Fig. 2.32

As $A = \{q_0, q_1, q_2\}$ in which final state q_2 lies hence A is final state in $B = \{q_1, q_2\}$ the state q_2 lies hence B is also final state in $C = \{q_2\}$, the state q_2 lies hence C is also a final state.

Example 2.17 : *Convert the given NFA with ε to its equivalent DFA.*

Fig. 2.33

Solution :

$$\varepsilon\text{-closure}\{q_0\} = \{q_0, q_1, q_2\}$$
$$\varepsilon\text{-closure}\{q_1\} = \{q_1\}$$
$$\varepsilon\text{-closure}\{q_2\} = \{q_2\}$$
$$\varepsilon\text{-closure}\{q_3\} = \{q_3\}$$
$$\varepsilon\text{-closure}\{q_4\} = \{q_4\}$$

Now, Let ε-closure $\{q_0\} = \{q_0, q_1, q_2\}$ be state A.

Hence $\delta'(A, 0) = \varepsilon\text{-closure}\{\delta((q_0, q_1, q_2), 0)\}$

$$= \varepsilon\text{-closure}\{\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)\}$$
$$= \varepsilon\text{-closure}\{q_3\}$$
$$= \{q_3\} \text{ call it as state B.}$$
$$\delta'(A, 1) = \varepsilon\text{-closure}\{\delta((q_0, q_1, q_2), 1)\}$$
$$= \varepsilon\text{-closure}\{\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)\}$$
$$= \varepsilon\text{-closure}\{q_3\}$$
$$= q_3 = B.$$

The partial DFA will be

Now,

Fig. 2.34

$$\delta'(B,0) = \varepsilon\text{-closure}\{\delta(q_3,0)\}$$
$$= \phi$$
$$\delta'(B,1) = \varepsilon\text{-closure}\{\delta(q_3,1)\}$$
$$= \varepsilon\text{-closure}\{q_4\}$$
$$= \{q_4\} \text{ i.e. state C}$$
$$\delta'(C,0) = \varepsilon\text{-closure}\{\delta(q_4,0)\}$$
$$= \phi$$
$$\delta'(C,1) = \varepsilon\text{-closure}\{\delta(q_4,1)\}$$
$$= \phi$$

The DFA will be,

Fig. 2.35

Example 2.18 : *Consider following NFA with ε.*

State \ Input	a	b	c	ε
→ p	{p}	{q}	{r}	ϕ
q	{q}	{r}	ϕ	{p}
r	{r}	ϕ	{p}	{q}

Convert it to its equivalent DFA.

Solution : We will first compute ε - closure for start state p.

Fig. 2.36

ε - closure (p) = {p} call it as state A.

Now we will obtain δ transitions on state A.

State A

$$\begin{aligned}
\delta(A, a) &= \varepsilon\text{ - closure } (\delta(A, a)) \\
&= \varepsilon\text{ - closure}(\delta(p, a)) \\
&= \varepsilon\text{ - closure } (p) \\
&= \{p\} \text{ i.e. state A only.} \\
\delta(A, b) &= \varepsilon\text{ - closure } (\delta(A, b)) \\
&= \varepsilon\text{ - closure}(\delta(p, b)) \\
&= \varepsilon\text{ - closure } (q) \\
&= \{q, p\} \text{ i.e. } \{p, q\} \text{ Let us call it state B.} \\
\delta(A, b) &= B \\
\delta(A, c) &= \varepsilon\text{ - closure } (\delta(A, c)) \\
&= \varepsilon\text{ - closure } (\delta(p, c)) \\
&= \varepsilon\text{ - closure } (r) = \{q, r\}. \text{ Call it as state C.}
\end{aligned}$$

State B = {p, q}

$$\begin{aligned}
\delta(B, a) &= \varepsilon\text{ - closure } (\delta(B, a)) \\
&= \varepsilon\text{ - closure}(\delta(p, q), a) \\
&= \varepsilon\text{ - closure } (\delta(p, a) \cup \delta(q, a)) \\
&= \varepsilon\text{ - closure } (p \cup q) \\
&= \varepsilon\text{ - closure } (p, q) \\
&= \varepsilon\text{ - closure}(p) \cup \varepsilon\text{ - closure}(q) \\
&= \{p\} \cup \{q\} = \{p, q\} \text{ i.e. state B only.} \\
\delta(B, a) &= B. \\
\delta(B, b) &= \varepsilon\text{ - closure } (\delta(B, b)) \\
&= \varepsilon\text{ - closure}(\delta(p, q), b) \\
&= \varepsilon\text{ - closure } (\delta(p, b) \cup \delta(q, b)) \\
&= \varepsilon\text{ - closure } (q \cup r) \\
&= \varepsilon\text{ - closure } (q, r) \\
&= \varepsilon\text{ - closure}(q) \cup \varepsilon\text{ - closure}(r) \\
&= \{p, q\} \cup \{q, r\} = \{p, q, r\} \text{ i.e. state D.}
\end{aligned}$$

$\therefore \quad \delta(B, b) = D.$

$$\begin{aligned}
\delta(B, c) &= \varepsilon\text{ - closure } (\delta(B, c)) \\
&= \varepsilon\text{ - closure}(\delta(p, q), c) \\
&= \varepsilon\text{ - closure } (\delta(p, c) \cup \delta(q, c)) \\
&= \varepsilon\text{ - closure } (r \cup \phi) \\
&= \varepsilon\text{ - closure } (r) \\
&= \{q, r\} \\
\delta(B, c) &= C
\end{aligned}$$

State C = {q, r}

$$\begin{aligned}
\delta(C, a) &= \varepsilon\text{ - closure } (\delta(C, a)) \\
&= \varepsilon\text{ - closure}(\delta(q, r), a) \\
&= \varepsilon\text{ - closure } (\delta(q, a) \cup \delta(r, a)) \\
&= \varepsilon\text{ - closure } (q \cup r) \\
&= \varepsilon\text{ - closure } (q) \cup \varepsilon\text{ - closure}(r) \\
&= \{p, q\} \cup \{q, r\} \\
&= \{p, q, r\} \text{ i.e. state D.}
\end{aligned}$$

$\therefore \quad \delta(C, a) = D$

$$\begin{aligned}
\delta(C, b) &= \varepsilon\text{ - closure } (\delta(C, b)) \\
&= \varepsilon\text{ - closure}(\delta(q, r), b) \\
&= \varepsilon\text{ - closure } (\delta(q, b) \cup \delta(r, b)) \\
&= \varepsilon\text{ - closure } (r \cup \phi) \\
&= \varepsilon\text{ - closure } (r) \\
&= \{q, r\} \text{ i.e. state C.}
\end{aligned}$$

$\therefore \quad \delta(C, b) = C$

$$\begin{aligned}
\delta(C, c) &= \varepsilon\text{ - closure } (\delta(C, c)) \\
&= \varepsilon\text{ - closure}(\delta(q, r), c) \\
&= \varepsilon\text{ - closure } (\delta(q, c) \cup \delta(r, c)) \\
&= \varepsilon\text{ - closure } (\phi \cup p)
\end{aligned}$$

$$= \varepsilon\text{ - closure }(p)$$

$$= \{p\} \text{ i.e. state A.}$$

$$\therefore \quad \delta(C, c) = A$$

State D = {p, q, r}

$$\delta(D, a) = \varepsilon\text{ - closure }(\delta(D, a))$$

$$= \varepsilon\text{ - closure}(\delta(p, q, r), a)$$

$$= \varepsilon\text{ - closure }(\delta(p, a) \cup \delta(q, a) \cup \delta(r, a))$$

$$= \varepsilon\text{ - closure }(p \cup q \cup r)$$

$$= \varepsilon\text{ - closure }(p) \cup \varepsilon\text{ - closure}(q) \cup \varepsilon\text{ - closure}(r)$$

$$= \{p, q, r\} \text{ i.e. state D.}$$

$$\therefore \quad \delta(D, a) = D$$

$$\delta(D, b) = \varepsilon\text{ - closure }(\delta(D, b))$$

$$= \varepsilon\text{ - closure}(\delta(p, q, r), b)$$

$$= \varepsilon\text{ - closure }(\delta(p, b) \cup \delta(q, b) \cup \delta(r, b))$$

$$= \varepsilon\text{ - closure }(q \cup r \cup \phi)$$

$$= \varepsilon\text{ - closure}(q, r)$$

$$= \varepsilon\text{ - closure }(q) \cup \varepsilon\text{ - closure}(r)$$

$$= \{p, q, r\} \text{ i.e. state D.}$$

$$\therefore \quad \delta(D, b) = D$$

$$\delta(D, c) = \varepsilon\text{ - closure }(\delta(D, c))$$

$$= \varepsilon\text{ - closure}(\delta(p, q, r), c)$$

$$= \varepsilon\text{ - closure }(\delta(p, c) \cup \delta(q, c) \cup \delta(r, c))$$

$$= \varepsilon\text{ - closure }(r \cup \phi \cup p)$$

$$= \varepsilon\text{ - closure }(r) \cup \varepsilon\text{ - closure}(p)$$

$$= \{q, r\} \cup \{p\}$$

$$= \{p, q, r\} \text{ i.e. state D.}$$

$$\therefore \quad \delta(D, c) = D$$

The transition table from above calculations can be obtained as

State \ Input	a	b	c
→ A	A	B	C
B	B	D	C
(C)	D	C	A
(D)	D	D	D

As state A = {p} it is a start state and states C and D contain final state r, hence these are final states. The transition diagram for the DFA is

Fig. 2.37

Example 2.19 : *Construct DFA for the NFA - ε Fig. 2.38 shown below.*

Fig. 2.38

Solution : We will first write ε - closure of start state q_0.

ε - closure (q_0) = $\{q_0\}$ call it as state A.

Now obtain δ transitions for input a and b for state A.

$$\begin{aligned}
\delta(A, a) &= \varepsilon \text{ - closure } (\delta(A, a)) \\
&= \varepsilon \text{ - closure } (\delta(q_0, a)) \\
&= \varepsilon \text{ - closure } (q_0, q_1) \\
&= \varepsilon \text{ - closure } (q_0) \cup \varepsilon \text{ - closure } (q_1) \\
&= \{q_0\} \cup \{q_1, q_2\} \\
&= \{q_0, q_1, q_2\} \rightarrow \text{call it as state B.} \\
\delta(A, b) &= \varepsilon \text{ - closure } (\delta(A, b)) \\
&= \varepsilon \text{ - closure } (\delta(q_0, b)) \\
&= \varepsilon \text{ - closure } (\phi) \\
&= \phi
\end{aligned}$$

Now consider state B = $\{q_0, q_1, q_2\}$

$$\begin{aligned}
\delta(B, a) &= \varepsilon \text{ - closure } \{\delta(B, a)\} \\
&= \varepsilon \text{ - closure } \{\delta\{q_0, q_1, q_2\}, a\} \\
&= \varepsilon \text{ - closure } \{\delta\{q_0, a\} \cup \delta\{q_1, a\} \cup \delta\{q_2, a\}\} \\
&= \varepsilon \text{ - closure } \{\{q_0, q_1\} \cup \phi \cup \phi\} \\
&= \{q_0, q_1, q_2\} \text{ i.e. state B.} \\
&= B \\
\delta(B, b) &= \varepsilon \text{ - closure } \{\delta(B, b)\} \\
&= \varepsilon \text{ - closure } \{\delta(\{q_0, q_1, q_2\}, b)\} \\
&= \varepsilon \text{ - closure } \{\delta\{q_0, b\} \cup \delta\{q_1, b\} \cup \delta\{q_2, b\}\} \\
&= \varepsilon \text{ - closure } \{\phi \cup \{q_3\} \cup \{q_2\}\} \\
&= \varepsilon \text{ - closure } \{\{q_2, q_3\}\} \\
&= \varepsilon \text{ - closure } \{q_2\} \cup \varepsilon \text{ - closure } \{q_3\} \\
&= \{q_2\} \cup \{q_3\} \\
&= \{q_2, q_3\} \rightarrow \text{call it as state C.}
\end{aligned}$$

The δ transitions on C are obtained as -

$$\delta(C, a) = \varepsilon \text{ - closure } \{\delta(C, a)\}$$

$$= \varepsilon \text{ - closure } \{ \delta((q_2, q_3), a)\}$$

$$= \varepsilon \text{ - closure } \{\delta(q_2, a) \cup \delta(q_3, a)\}$$

$$= \varepsilon \text{ - closure } \{\phi \cup \{q_0\}\}$$

$$= \varepsilon \text{ - closure } \{q_0\}$$

$$= \{q_0\} \text{ i.e. A}$$

$$\delta(C, b) = \varepsilon \text{ - closure } \{\delta(C, b)\}$$

$$= \varepsilon \text{ - closure } \{\delta\{q_2, q_3\}, b\}$$

$$= \varepsilon \text{ - closure } \{\delta(q_2, b) \cup \delta(q_3, b)\}$$

$$= \varepsilon \text{ - closure } \{q_2 \cup \phi\}$$

$$= \varepsilon \text{ - closure } \{q_2\}$$

$$= \{q_1, q_2\} \rightarrow \text{call it as state D.}$$

$$\delta(D, a) = \varepsilon \text{ - closure } \{\delta(D, a)\}$$

$$= \varepsilon \text{ - closure } \{ \delta(\{q_1, q_2\}, a)\}$$

$$= \varepsilon \text{ - closure } \{\delta(q_1, a) \cup \delta(q_2, a)\}$$

$$= \varepsilon \text{ - closure } \{\phi \cup \phi\}$$

$$= \phi$$

$$\delta(D, b) = \varepsilon \text{ - closure } \{ \delta(\{q_1, q_2\}, b)\}$$

$$= \varepsilon \text{ - closure } \{\delta(q_1, b) \cup \delta(q_2, b)\}$$

$$= \varepsilon \text{ - closure } \{q_3 \cup q_2\}$$

$$= \varepsilon \text{ - closure } \{q_2, q_3\}$$

$$= \varepsilon \text{ - closure } \{q_2\} \cup \varepsilon \text{ - closure } \{q_3\}$$

$$= \{q_2, q_3\} \text{ i.e. C.}$$

$$= C$$

Now no new state is generated. Hence we will write transition table for above computations.

State \ Input	a	b
→ A	B	ϕ
B	B	C
C	A	D
D	ϕ	C

Fig. 2.39

Example 2.20 : *Develop a deterministic Finite Automata accepting the language given over the alphabet {0, 1}. L = {The set of all strings such that every block of five consecutive contains at least two 0's}.*

Ans. : We consider the five consecutive symbols as

Similarly,

But,

Thus by considering various combinations of 0's and 1's over a block of 5 symbols we can draw DFA as follows -

Thus we avoid 4 or 5 1's in the strings. Not having 4 or 5 1's means allowing at least two 0's in the block of 5 symbols. With this simple logic the DFA can be constructed.

3.5 Minimization of FSM

The minimization of FSM means reducing the number of states from given FA. Thus we get the FSM with redundant states after minimizing the FSM.

While minimizing FSM we first find out which two states are equivalent we can represent those two states by one representative state.

The two states q_1 and q_2 are equivalent if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both of them are nonfinal states for all $x \in \Sigma^*$ (Σ^* indicate any string of any length) we can minimize the given FSM by finding equivalent states.

Method for Construction of Minimum State Automata :

Step 1 : We will create a set Π_0 as $\Pi_0 = \left\{ Q_1^0, Q_2^0 \right\}$ where Q_1^0 is set of all final states and $Q_2^0 = Q - Q_1^0$ where Q is a set of all the states in DFA.

Step 2 : Now we will construct Π_{K+1} from Π_K. Let Q_i^K be any subset in Π_K. If q_1 and q_2 are in Q_i^K they are (K + 1) equivalent provided $\delta(q_1, a)$ and $\delta(q_2, a)$ are K equivalent. Find out whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are residing in the same equivalence class Π_K. Then it is said that q_1 and q_2 are (K + 1) equivalent. Thus Q_i^K is further divided into (K + 1) equivalence classes. Repeat step 2 for every Q_i^K in Π_K and obtain all the elements of Π_{K+1}.

Step 3 : Construct Π_n for n = 1, 2, ... until $\Pi_n = \Pi_{n+1}$.

Step 4 : Then replace all the equivalent states in one equivalence class by representative state. This helps in minimizing the given DFA.

Let us understand this method with the help of some examples.

Example 3.7 : *Construct the minimum state automaton for the following transition diagram.*

Solution : We will first construct a transition table for the given DFA.

We will start constructing equivalence classes.

State \ Input	a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

There is only one final state i.e. q_3. Hence we can partition Q as Q_1^0 and Q_2^0.

$$Q_1^0 = F = \{q_3\} \text{ As } \{q_3\} \text{ cannot be partitioned further.}$$

$$Q_2^0 = Q - Q_1^0$$

$$= \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}$$

Now, we will compare q_0 with q_2.

	a	b
q_0	q_1	q_0
q_2	q_3	q_1

Under a - column of q_0 and q_2 we get q_1 and q_3 respectively. But q_1 and q_3 lie in different states. Hence they are not 1 - equivalent.

Similarly consider q_0 and q_4.

	a	b
q_0	q_1	q_0
q_4	q_3	q_5

Under a - column of q_0 and q_4 we get q_1 and q_3 which lie in different sets. Hence q_0 is not 1 - equivalent to q_4. Similarly q_0 is not 1 - equivalent to q_7 (observe b-column of q_0 and q_7). But q_0 is 1 - equivalent to q_1, q_5 and q_6.

$$Q_1' = \{q_3\}$$

$$Q_2' = \{q_0, q_1, q_5, q_6\}$$

q_2 is 1 - equivalent to q_4. Hence

$$Q_3' = \{q_2, q_4\}$$

Since under a-column we get q_3 only for q_2 and q_4. And under b - column of q_2 and q_4 we get q_1 and q_5. But q_1 and q_5 lie in one set. Thus q_2 and q_4 are 1 - equivalent.

The only element left over in Q_2^0 is q_7.

$$\therefore \quad Q_4' = \{q_7\} \text{ The equivalence class is -}$$

$$\Pi_1 = \{\{q_3\}, \{q_0, q_1, q_5, q_6\}, \{q_2, q_4\}, \{q_7\}\}$$

$$Q_1^2 = \{q_3\}$$

Now q_0 is 2 - equivalent to q_6 because

	a	b
q_0	q_1	q_0
q_6	q_5	q_6

Both lie in same set.

Fig. 3.5 (a)

But q_0 is not 2 - equivalent to q_1 and q_5.

	a	b
q_0	q_1	q_0
q_1	q_0	q_2

	a	b
q_0	q_1	q_0
q_5	q_6	q_4

Do not lie in same set.

Fig. 3.5 (b)

Hence $\quad Q_2^2 = \{q_0, q_6\}$

But q_1 is 2 - equivalent to q_5.

	a	b
q_1	q_0	q_2
q_5	q_6	q_4

Lie in one set. Lie in one set.

Fig. 3.5 (c)

Hence $\quad Q_3^2 = \{q_1, q_5\}$

Similarly q_2 is 2 - equivalent to q_4.

$$Q_4^2 = \{q_2, q_4\}$$

$$Q_5^2 = \{q_7\}$$

Thus, $\boxed{\Pi_2 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}}$

$$Q_1^3 = \{q_3\}$$

As q_0 is 3 - equivalent to q_6,

$$Q_2^3 = \{q_0, q_6\}$$

As q_1 is 3 - equivalent to q_5,

$$Q_3^3 = \{q_1, q_5\}$$

As q_2 is 3 - equivalent to q_4,

$$Q_4^3 = \{q_2, q_4\}$$

and $$Q_5^3 = \{q_7\}$$

$\therefore$ $$\boxed{\Pi_3 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}}$$

As $\Pi_3 = \Pi_2$ we have got equivalence class from Π_2. Hence we can write a transition table as

	State \ Input	a	b
→	$[q_0, q_6]$	$[q_1, q_5]$	$[q_0, q_6]$
	$[q_1, q_5]$	$[q_0, q_6]$	$[q_2, q_4]$
	$[q_2, q_4]$	$[q_3]$	$[q_1, q_5]$
*	$[q_3]$	$[q_3]$	$[q_0, q_6]$
	$[q_7]$	$[q_0, q_6]$	$[q_3]$

The transition diagram with minimized states is -

Fig. 3.6

Example 3.8 : *Construct a minimum state DFA for the following given automata -*

Fig. 3.7

Solution : We will first construct a transition table from given FA.

State \ Input	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Now we will first obtain Q_1^0.

$$Q_1^0 = F = \{q_2\} \rightarrow \text{Final state}$$

$$Q_2^0 = Q - Q_1^0$$

$$= \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}$$

Hence $\Pi_0 = \{\{q_2\}\ \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\}$

We cannot partition $\{q_2\}$ further. Hence $Q_1' = \{q_2\}$.

Now consider another set from Π_0 i.e. $\{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}$. We will now compare input entries of q_0 for all remaining states in this set. Consider

State \ Input	0	1
q_0	q_1	q_5
q_1	q_6	q_2

Here $q_5 \in Q_2^0$ and $q_2 \in Q_1^0$.
Hence they are not 1-equivalent.

Fig. 3.8 (a)

State \ Input	0	1
q_0	q_1	q_5
q_3	q_2	q_6

Here $q_1 \in Q_2^0$ and $q_2 \in Q_1^0$.
Hence they are not 1-equivalent.

Fig. 3.8 (b)

State \ Input	0	1
q_0	q_1	q_5
q_5	q_2	q_6

Here $q_1 \in Q_2^0$ and $q_2 \in Q_1^0$.
Hence they are not 1-equivalent.

Fig. 3.8 (c)

State \ Input	0	1
q_0	q_1	q_5
q_7	q_6	q_2

Here $q_5 \in Q_2^0$ and $q_2 \in Q_1^0$.
Hence they are not 1-equivalent.

Fig. 3.8 (d)

Thus q_0 is not ρ - equivalent to q_3, q_5 and q_7. Now we will obtain the set of equivalent sets. Consider q_0 with q_4, q_6.

State \ Input	0	1
q_0	q_1	q_5
q_4	q_7	q_5

Both $\in Q_2^0$

Same states

State \ Input	0	1
q_0	q_1	q_5
q_6	q_6	q_4

Both $\in Q_2^0$

Fig. 3.8 (e)

Hence $\{q_0, q_4, q_6\}$ is one subset in Π_1.

Therefore $Q'_2 = \{q_0, q_4, q_6\}$.

We will consider a subset $\{q_1, q_3, q_5, q_7\}$. Now we will find the 1 - equivalent subset from this subset. Hence we need to compare q_1 with q_3, q_5 and q_7 respectively.

State \ Input	0	1
q_1	q_6	q_2
q_3	q_2	q_6

$q_6 \in Q_2^0$ and $q_2 \in Q_1^0$.
$\therefore$ Both are not 1-equivalent.

Fig. 3.8 (f)

State \ Input	0	1
q_1	q_6	q_2
q_5	q_2	q_6

$q_6 \in Q_2^0$ and $q_2 \in Q_1^0$.
$\therefore$ Both are not 1-equivalent.

Fig. 3.8 (g)

State \ Input	0	1
q_1	q_6	q_2
q_7	q_6	q_2

$q_6 \in Q_2^0$
Both are 1-equivalent.

$q_2 \in Q_1^0$
Both are 1-equivalent.

Fig. 3.8 (h)

i.e. q_1 is 1 - equivalent to q_7. Hence $\{q_1, q_7\}$ will be one subset. We cannot partition this subset further. Hence $Q'_3 = \{q_1, q_7\}$.

Now we will compare q_3 with q_5.

State \ Input	0	1
q_3	q_2	q_6
q_5	q_2	q_6

Clearly both the states are 1-equivalent.

Fig. 3.8 (i)

$$\therefore \qquad Q'_4 = \{q_3, q_5\}$$

Now,

$$\Pi_2 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

From Π_2 we can consider a subset $\{q_0, q_4, q_6\}$. We will again compare q_0 with q_4 and q_6.

State \ Input	0	1
q_0	q_1	q_5
q_4	q_7	q_5

Belongs to Q'_3 Hence are 2-equivalent. Both are same

State \ Input	0	1
q_0	q_1	q_5
q_6	q_6	q_4

Not 2-equivalent.

Fig. 3.8 (j)

Thus q_0 and q_4 and 2 - equivalent and q_0, q_6 are not 2 - equivalent.

Similarly q_3 and q_5 are 2 - equivalent.

Also q_1 and q_7 are 2 - equivalent.

Thus $\Pi_2 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$

Now, q_0 and q_4 are 3 - equivalent.

q_1 and q_7 are 3 - equivalent.

q_3 and q_5 are 3 - equivalent.

As $\Pi_2 = \Pi_3$. The Π_2 gives equivalence classes. Therefore we can construct a finite automata with minimized state as

$$M' = (Q', \{0, 1\}, \delta', q_0', F')$$

where Q′ is a set of states in FA.

$$\text{i.e.} = \{[q_2], [q_0, q_4], [q_6], [q_1, q_7], [q_3, q_5]\}$$

q_0' is initial state i.e. $[q_0, q_4]$

F′ is a set of final states i.e. $[q_2]$

The δ' can be shown by following transition table.

Input / State	0	1
$[q_0, q_4]$	$[q_1, q_7]$	$[q_3, q_5]$
$[q_1, q_7]$	$[q_6]$	$[q_2]$
$[q_2]$	$[q_0, q_4]$	$[q_2]$
$[q_3, q_5]$	$[q_2]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_0, q_4]$

Here q_0 and q_4, q_1 and q_7, q_3 and q_5 are treated as one state. Now we can draw the transition diagram for minimized machine as -

Fig. 3.9

Example 3.9 : *Construct a minimum state automaton for following DFA.*

Fig. 3.10

Solution : It will be easier if we construct a transition table for given finite automata.

State \ Input	a	b
→ q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
q_4	q_1	q_2

Now we will first obtain Q_1^0.

$\therefore \quad Q_1^0 = F = \{q_4\} \rightarrow$ Final state

$Q_2^0 = Q - Q_1^0.$

$= \{q_0, q_1, q_2, q_3\}$

Hence $\quad \Pi_0 = \{\{q_4\}, \{q_0, q_1, q_2, q_3\}\}$

As we cannot partition $Q_1^0 = \{q_4\}$ further we say $Q_1' = \{q_4\}$.

Now consider subset $\{q_0, q_1, q_2, q_3\}$. We will compare q_0 with q_1, q_2 and q_3.

State \ Input	a	b
q_0	q_1	q_2
q_1	q_1	q_3

Both are 1-equivalent.

(a)

State \ Input	a	b
q_0	q_1	q_2
q_2	q_1	q_2

Clearly q_0 and q_2 are equivalent.

(b)

State \ Input	a	b
q_0	q_1	q_2
q_3	q_1	q_4

$q_2 \in Q_2^0$ and

$q_4 \in Q_1^0$ Hence both are not 1-equivalent.

(c)

Fig. 3.11

$$Q'_2 = \{q_0, q_1, q_2\}$$

$$Q'_3 = \{q_3\}$$

Hence, $\Pi_1 = \{\{q_4\}, \{q_0, q_1, q_2\}, \{q_3\}\}$

Now we will compare q_0 with q_1 and q_2 respectively.

State \ Input	a	b
q_0	q_1	q_2
q_1	q_1	q_3

As $q_2 \in Q'_2$ and $q_3 \in Q'_3$ Both are not 2-equivalent

Fig. 3.11 (d)

State \ Input	a	b
q_0	q_1	q_2
q_2	q_1	q_2

Clearly both the states are equivalent.

Fig. 3.11 (e)

Hence we get, $\Pi_2 = \{\{q_4\}, \{q_0, q_2\}, \{q_1\}, \{q_3\}\}$

We can now state $\{q_0, q_2\}$ are 3 - equivalent. And we cannot partition $\{q_1\}$ and $\{q_3\}$ further. Hence

$$\Pi_3 = \{\{q_4\}, \{q_0, q_2\}, \{q_1\}, \{q_3\}\}$$

As $\Pi_3 = \Pi_2$ and Π_2 gives equivalence classes we can construct minimum state finite automata as -

State \ Input	a	b
→ $[q_0, q_2]$	q_1	$[q_0, q_2]$
q_1	$[q_1]$	$[q_3]$
q_3	$[q_1]$	$[q_4]$
q_4	$[q_1]$	$[q_0, q_2]$

The transition diagram can be -

Fig. 3.12

Minimization of DFA (Alternate Method)

This method helps in finding equivalent states in one stroke. Hence it is an efficient method of minimization of given DFA. Let us understand this method with the help of some example.

Example 3.10 : *Minimize the DFA as given below.*

Fig. 3.13

Solution : We will build the transitions table for given DFA as follows.

	0	1
→ A	B	F
B	G	C
C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

We will now construct a table for each pair of states.

B							
C							
D							
E							
F							
G							
H							
	A	B	C	D	E	F	G

We will mark X for all the final and nonfinal states. Hence

B							
C	X	X					
D			X				
E			X				
F			X				
G			X				
H			X				
	A	B	C	D	E	F	G

Thus we have marked X for (A, C), (B, C), (C, D), (C, E), (C, F), (C, G), (C, H).

Now we will consider every pair form above table. Consider pair (G, H).

δ (G, 0) = G, δ (G, 1) = E

δ (H, 0) = G, δ (H, 1) = C

As for δ (G, 1) = E and δ (H, 1) = C. We can see X in (C, E) pair. Hence pair (G, H) is not equivalent.

Hence we will mark X in pair (G, H). Thus we will find the equivalent pairs.

Consider pair (B, H).

δ (B, 0) = G, δ (B, 1) = C

δ (H, 0) = G, δ (H, 1) = C

Thus the pair (B, H) is equivalent. Thus we obtain,

B	X						
C	X	X					
D	X	X	X				
E		X	X	X			
F	X	X	X		X		
G	X	X	X	X	X	X	
H	X		X	X	X	X	X
	A	B	C	D	E	F	G

Thus we get equivalent pairs as (A, E), (B, H), (D, F). Hence the minimized DFA will be,

	0	1
→ A	B	D
B	G	C
(C)	A	C
D	C	G
G	G	A

Example 3.11 : *Minimize the finite automata in Fig. 3.14 below and show both the given and the reduced one are equivalent.*

Fig. 3.14

Solution : We will first construct the transition table for given DFA -

State \ Input	0	1
→ q_0	q_1	q_3
q_1	q_2	q_4
q_2	q_1	q_4
q_3	q_2	q_4
q_4	q_4	q_4

Now we will partition the states Q in Q_1^0 and Q_2^0. The state q_4 is a final state. Hence

$$Q_1^0 = F = \{q_4\}$$
$$Q_2^0 = Q - Q_1^0 = \{q_0, q_1, q_2, q_3\}$$
$$\Pi_0 = \{\{q_4\}, \{q_0, q_1, q_2, q_3\}\}$$

Now we will compare q_0 with q_1. We find that under 0 column we get q_1 and q_2. And q_1 and q_2 lie in the same set i.e. Q_2^0. But under 1-column of q_0 and q_1 we get q_3, q_4 respectively. But $q_3 \in Q_2^0$ and $q_4 \in Q_1^0$. Hence they are not 1-equivalent. Similarly compare q_0 and q_2. We will find that under 0 column of q_0 and q_2 we get q_1 only. But under 1-column of q_0 and q_2 we get q_3 and q_4. The $q_3 \in Q_2^0$ and $q_4 \in Q_1^0$. Hence q_0 and q_2 are not 1-equivalent. Similarly q_0 is not 1-equivalent to q_3 because of 1-column entries.

$$Q_1' = \{q_4\}$$
$$Q_2' = \{q_0\}$$
$$Q_3' = \{q_1, q_2, q_3\}$$

Hence,

$$\Pi_1 = \{\{q_4\}, \{q_0\}, \{q_1, q_2, q_3\}\}$$

Now in set $\{q_1, q_2, q_3\}$ we compare q_1 with q_2 under 0-column we get q_2 and q_1 which are in the same set, and under 1-column we get q_4. And for state q_1 and q_3 we get same entries under 0 - column and under 1-column. Hence we can group them $\{q_2, q_3\}$.

The equivalence class becomes

$$\Pi_2 = \{\{q_4\}, \{q_0\}, \{q_1\}, \{q_2, q_3\}\}$$

Now further we cannot partition any of the set. We get

$$\Pi_3 = \{\{q_0\}, \{q_2\}, \{q_4\}, \{q_1, q_3\}\}$$

Hence minimized DFA is -

State \ Input	0	1
→ q_0	q_1	q_1
q_1	q_2	q_4
q_2	q_1	q_4
q_4	q_4	q_4

The transition diagram will be -

Fig. 3.15

3.4 Equivalence between Two FSMs

The two finite automata are said to be equivalent if both the automata accept the same set of strings, over an input set Σ. When two FAs are equivalent then there is some string x over Σ, on acceptance of that string if one FA reaches to final state other FA also reaches to final state. We can compare whether two FAs are equivalent or not using following method.

Method for Comparing two FAs

Let M and M' be two FAs and Σ is a set of input strings.

1. We will construct a transition table have pairwise entries (q, q') where $q \in M$ and $q' \in M'$ for each input symbol.
2. If we get in a pair as one final state and other nonfinal state then we terminate construction of transition table declaring that two FAs are not equivalent.
3. The construction of transition table gets terminated when there is no new pair appearing in the transition table.

Let us take one example to understand the technique of comparing two FAs.

Example 3.5 :*Consider the DFAs given below. Are they equivalent ?*

Fig. 3.2

Solution : We will first build the transition table for each input c and d. From first machine M on receiving input c in state q_1 we reach to state q_1 only. From second machine M', for state q_4 on receiving c we reach to state q_4. Thus for state (q_1, q_4), for input c we get next state as (q_1, q_4). Similarly for input d in state (q_1, q_4) we get next state as (q_2, q_5).

Both q_1 and q_4 are final states obtained in pair (q_1, q_4). Both q_2 and q_5 are nonfinal states obtained in pair (q_2, q_5) we will obtain transition for (q_2, q_5) for input c and d. The complete table is as given below.

	c	d
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_6)
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

From the above table note that we do not get one final state and other nonfinal state in a pair. Hence we declare that two DFAs are equivalent.

Example 3.6 : *Following are two FAs check whether they are equivalent or not.*

Fig. 3.3

Solution : We will design the transition table for each input symbol c and d as follows.

	c	d
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_7)	(q_1, q_6)

We will terminate the construction of transition table because we get a pair (q_1, q_6) in which q_1 is a final state and q_6 is a nonfinal state. As per equivalence rule final and nonfinal state cannot form a pair. Hence the given FAs are not equivalent.

2.8 Finite Automata with Output

The finite automata is a collection of $(Q, \Sigma, \delta, q_0, F)$ where Q is a set of states including q_0 as a start state. In FA after reading the input string if we get final state then the string is said to be "acceptable". If we do not get final state then it is said that string is "rejected". That means there is no need of output for the finite Automata. The 'accept' or 'reject' acts like 'yes' or 'no' output for the machine. But if there is a need for specifying the output other than yes or no, then in such a case we require finite automata along with output. There are two types of FA with output and those are :

1) Moore machine

2) Mealy machine

1) Moore machine

Moore machine is a finite state machine in which the next state is decided by current state and current input symbol. The output symbol at a given time depends only on the present state of the machine. The formal definition of Moore machine is,

"Moore machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

Q is finite set of states.

Σ is finite set of input symbols.

Δ is an output alphabet.

δ is an transition function such that $Q \times \Sigma \rightarrow Q$. This is also known as state function.

λ is output function $Q \rightarrow \Delta$. This function is also known as machine function.

q_0 is the initial state of machine.

For example :

Consider the Moore machine given below -

Fig. 2.45

The transition table will be -

Current state	Next state (δ)		Output (λ)
	0	1	
q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

In Moore machine output is associated with every state. In the above given Moore machine when machine is in q_0 state the output will be 1. For the Moore machine if the length of input string is n then output string has length n+1.

For the string 0110 then the output will be 11110.

2) Mealy machine

Mealy machine is a machine in which output symbol depends upon the present input symbol and present state of the machine. The Mealy machine can be defined as -

Mealy machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

Q is finite set of states.

Σ is finite set of input symbols.

Δ is an output alphabet.

δ is state transition function such that $Q \times \Sigma \rightarrow Q$.

λ is machine function such that $Q \times \Sigma \rightarrow \Delta$.

q_0 is initial state of machine.

For example :

Fig. 2.46

For the input string 1001 the output will be 0001. In mealy machine the length of input string is equal to length of output string.

Example 2.22 : *Design a Moore machine to generate 1's complement of given binary number.*

Solution : To generate 1's complement of given binary number the simple logic which we will apply is that if input is 0 then output will be 1 and if input is 1 then output will be 0. That means there are three state one-start state, second state is for taking 0's as input and produces output as 1. Then third state is for taking 1's as input and producing output as 0.

Hence the Moore machine will be,

Fig. 2.47

For instance, take one binary number 1011 then

Input :		1	0	1	1
State :	q_0	q_2	q_1	q_2	q_2
Output :	0	0	1	0	0

Thus we get 00100 as 1's complement of 1011, we can neglect the initial 0 and the output which we get is 0100 which is 1's complement of 1011. The transition table can be drawn as below -

Current state	Next state (δ) 0	Next state (δ) 1	Output (λ)
→ q_0	q_1	q_2	0
q_1	q_1	q_2	1
q_2	q_1	q_2	0

Thus Moore machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$; where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0,1\}$, $\Delta = \{0,1\}$.

The transition table shows the δ and λ functions.

Example 2.23 : *Design a Moore and Mealy machine for a binary input sequence such that if it has a substring 101 the machine outputs A if input has substring 110 it outputs B otherwise it outputs C.*

Solution : For designing such a machine we need to take care of two conditions and those are checking 101 and checking 110. If we get 101 the output will be A. If we recognize 110 the output will be B. For other strings the output will be C. We can make a partial design of it as follows.

Fig. 2.48

Now we will insert the possibilities of 1's and 0's for each state. Then the Moore machine becomes

Fig. 2.49

Now the Mealy machine can be

Fig. 2.50

Example 2.24 : *Design a Mealy machine to find 2's complement of a given binary number.*

Solution: For designing 2's complement of a binary number we assume that input is read from LSB to MSB. We will keep the binary number as it is until we read first 1. Keep that 1 as it is then change remaining 1's by 0's and 0's by 1's.

For example :

Let the binary number be

1011

$\longleftarrow$

read from LSB

Keep the first 1 from LSB as it is and toggle the remaining bits we will get

0101

Thus 2's complement of 1011 is 0101. The required Mealy machine will be -

Fig. 2.51

Example 2.25 : *Design a Moore machine which will increment the given binary number by 1.*

Solution : We will read the binary number from LSB one bit at a time. We will replace each 1 by 0 until we get first 0. Once we get first 0 we will replace it by 1 and then keep remaining bits as it is.

For example :

1	0	1	1	
				← Read from LSB bit by bit
1	0	1	0	
			↑	
1	0	0	0	
		↑		
1	1	0	0	
	↑			
1	1	0	0	
↑				

Using this logic we can built a Mealy machine as follows -

Fig. 2.52

Example 2.26 : *Give Moore and Mealy machine for $\Sigma = \{0, 1, 2\}$, print the residue modulo 5 of input treated as a ternary number.*

Solution : The ternary number is made up of 0,1 and 2. The interpretation of ternary number n can be -

i) If we write 0 after n then number becomes 3n.

ii) If we write 1 after n then number becomes 3n+1.

iii) If we write 2 after n then number becomes 3n+2.

For example :

If n = 4 then its value is $4 \times 3^0 = 4$.

If we write 0 after 4 i.e.

$$40 = 4 \times 3^1 + 0 \times 3^0 = 12 \qquad \text{i.e. } (3 \times 4)$$

If we write 1 after 4 then

$$41 = 4 \times 3^1 + 1 \times 3^0 = 13 \qquad \text{i.e. } 3n+1$$

If we write 2 after 4 we get

$$42 = 4 \times 3^1 + 2 \times 3^0 = 14 \qquad \text{i.e. } 3n+2$$

For residue modulo 5 we will get remainder 0, remainder 1, remainder 2, remainder 3 and remainder 4 values. Then we assume various states for these remainders as -

q_0 - remainder 0 state

q_1 - remainder 1 state

q_2 - remainder 2 state

q_3 - remainder 3 state

q_4 - remainder 4 state

Now consider n = 4,

For 4.0 we get decimal value 12 that means 12 % 5= 2 it gives remainder 2.

For 4.1 we get decimal value 13 that means 13 % 5= 3 it gives remainder 3.

Similarly 4.2 gives remainder 4.

n = 4 itself is remainder 4. Hence we can design,

Fig. 2.53

Considering all possible cases we can design Moore machine as,

Fig. 2.54

Similarly the Mealy machine can be -

Fig. 2.55

Example 2.27 : *Design a Moore machine that will read sequences made up of letters A, E, I, O, U and will give as output the same sequences except that in case where an I directly follows an E, it will be changed to u. Design the Mealy machine for the same.*

Solution : We will assume a separate state for each alphabet. The output of that state will be corresponding letter.

For instance : For alphabet A the state will be q_0 and output of q_0 will be A. For alphabet E the state will be q_1 and output of q_1 will be E. Continuing in this fashion we will have q_0, q_1, q_2, q_3 and q_4 states. The start state will be q_0 we will take care of one thing and that is when I comes immediately after E it will lead to the state q_4 because output of state q_4 is u. With this logic the corresponding Moore machine will be -

Fig. 2.56

Now we will draw the Mealy machine for the same problem.

Fig. 2.57

Here the Mealy machine has the output along the edge itself we have got only two state. The transition changes from q_0 to q_1 when we get E. The I which is immediately following E will have the output u. Consider the string AIEOAEI will be given an output.

Example 2.28 : *Construct a Moore machine to determine residue mod 3 for binary number.*

Solution : This Moore machine is also called remainder 3 tester. In this machine we will get remainder 0, remainder 1 and remainder 2. To interpret the given binary number in its decimal value we consider n as a number if 0 is written after n then its value becomes 2n. If 1 is written after n then it's value becomes 2n + 1. For instance, if n = 0 then its decimal value is 0 then,

$$01 = 2n+1 = 1\times 0+1 = 1$$

$$011 = 2n+1 = (1\times 2)+1 = 3$$

Consider n = 1 its decimal value is 1. After 1 if 0 comes then its value will be

$$10 = 2n = 2\times 1 = 2$$

If 1 comes after 10 then its value becomes,

$$101 = 2n+1 = (2\times 2) + 1 = 5$$

With this logic we can construct a Moore machine with 3 states. q_0 is the start state and is considered as remainder 0 state. q_1 is considered to be remainder 1 state and q_2 is considered as remainder 2 state.

Fig. 2.58

2.9 Equivalence of Moore and Mealy Machines

The meaning of word equivalence is the two machines which accept the same language. Hence equivalence of Moore and Mealy machine means both the machines generate the same output string for same input string.

We cannot convert Moore machine to its equivalent Mealy machine directly because length of Moore machine is one longer than Mealy machine for the given input. We will make use of following method to convert Moore machine to Mealy machine.

Method for conversion of Moore machine to Mealy machine

Let $M = (Q, \Sigma, \delta, \lambda, q_0)$ be a Moore machine. The equivalent Mealy machine can be represented by $M' = (Q, \Sigma, \delta, \lambda', q_0)$. The output function λ' can be obtained as -

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Let us understand this procedure with the help of some examples.

Example 2.29 : *The Moore machine to determine residue mod 3 for binary number is given below. Convert it to Mealy equivalent machine.*

Q \ Σ	0	1	Output (λ)
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

Solution : The transition diagram for the given problem can be drawn as -

Fig. 2.59

The output function λ' can be obtained using following rule,

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Hence we will obtain output for every transition corresponding to input symbol.

$$\lambda'(q_0, 0) = \lambda(\delta(q_0, 0))$$

$$= \lambda(q_0) \quad \text{i.e. output of } q_0$$

$$\boxed{\lambda'(q_0, 0) = 0}$$

$$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1))$$

$$= \lambda(q_1) \quad \text{i.e. output of } q_1$$

$$\boxed{\lambda'(q_0, 1) = 1}$$

$$\lambda'(q_1,0) = \lambda(\delta(q_1,0))$$
$$= \lambda(q_2)$$
$$\boxed{\lambda'(q_1,0) = 2}$$
$$\lambda'(q_1,1) = \lambda(\delta(q_1,1))$$
$$= \lambda(q_0)$$
$$\boxed{\lambda'(q_1,1) = 0}$$
$$\lambda'(q_2,0) = \lambda(\delta(q_2,0))$$
$$= \lambda(q_1)$$
$$\boxed{\lambda'(q_2,0) = 1}$$
$$\lambda'(q_2,1) = \lambda(\delta(q_2,1))$$
$$= \lambda(q_2)$$
$$\boxed{\lambda'(q_2,1) = 2}$$

Hence the transition table can be drawn as follows –

Hence the transition table can be drawn as follows –

Q \ Σ	Input 0		Input 1	
	State	O/P	State	O/P
q_0	q_0	0	q_1	1
q_1	q_2	2	q_0	0
q_2	q_1	1	q_2	2

The transition diagram of Mealy machine is,

Fig. 2.60

The input string 10011 then the output for Mealy machine will be

Next state	$q_1 \rightarrow$	$q_2 \rightarrow$	$q_2 \rightarrow$	$q_1 \rightarrow$	q_0
Output	1	2	2	1	0

The output sequence for Moore machine will be

Input		1	0	0	1	1
Next state	$q_0 \rightarrow$	$q_1 \rightarrow$	$q_2 \rightarrow$	$q_2 \rightarrow$	$q_1 \rightarrow$	q_0
Output	0	1	2	2	1	0

The length of output sequence is n+1 in Moore machine and is n in Mealy machine which is desired.

Example 2.30 : *Convert the following Moore machine into equivalent Mealy machine* $M = (\{q_0, q_1\}, \{a, b\}, \{0,1\}\ \delta, \lambda, q_0)$

Solution :

δ	a	b	Output (λ)
q_0	q_0	q_1	0
q_1	q_0	q_1	1

Fig. 2.61

The equivalent Mealy machine can be obtained as follows -

$$\lambda'(q_0, a) = \lambda(\delta(q_0, a))$$
$$= \lambda(q_0)$$
$$= 0$$
$$\lambda'(q_0, b) = \lambda(\delta(q_0, b))$$
$$= \lambda(q_1)$$
$$= 1$$
$$\lambda'(q_1, a) = \lambda(\delta(q_1, a))$$
$$= \lambda(q_0)$$
$$= 0$$

$$\lambda'(q_1, b) = \lambda(\delta(q_1, b))$$
$$= \lambda(q_1)$$
$$= 1$$

Hence the transition table can be drawn as follows -

Q \ Σ	Input a		Input b	
	State	O/P	State	O/P
q_0	q_0	0	q_1	1
q_1	q_0	0	q_1	1

The equivalent Mealy machine will be,

Fig. 2.62

Consider the output sequence for Moore machine for input sequence,

a b b a

Then

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_0$$

0 0 1 1 0

Similarly output sequence for Mealy machine will be

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \longrightarrow q_0$$

0 1 1 1 0

We can note that length of Moore machine is n+1 and that of Mealy machine is n.

Method for conversion of Mealy machine to Moore machine

In Moore machine the output is associated with every state and in Mealy machine the output is given along the edge with input symbol. To convert Moore machine to Mealy machine state output symbols are distributed to input symbol paths. But while converting Mealy machine to Moore machine we will create a separate state for every new output symbol and according incoming and outgoing edges are distributed.

Let $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Mealy machine then there exists a Moore machine M′ equivalent to M.

The M′ can be given as $M' = (Q \times \Delta, \Sigma, \Delta, \delta', \lambda', [q_0, b_0])$ where b_0 is an output symbol selected from Δ. We will calculate δ′ and λ′ as follows -

$$\delta'([q,b], a) = [\delta(q,a), \lambda(q,a)]$$

$$l([q,b]) = b$$

δ' defines the move made by M′ on input a.

M′on input a.

λ' defines the output made for the corresponding state q.

Thus we have to calculate δ' and λ' for $q_0, q_1, q_2, \ldots\ldots q_n$ on input $a_0, a_1, \ldots\ldots a_n$ and should emit the outputs $b_1, b_2, b_3, \ldots\ldots b_n$.

Let us understand this method of conversion with the help of some examples.

Example 2.31 : *Convert the following Mealy machine into equivalent Moore machine.*

Fig. 2.63

Solution : The states for Moore machine will be $[q_0, A]$, $[q_0, B]$, $[q_1, A]$, $[q_1, B]$. Then we will calculate δ' and λ' as follows -

$$\delta'((q_0, A), 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$
$$= [q_0, A]$$
$$\lambda'([q_0, A]) = A$$
$$\delta'([q_1, A], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$
$$= [q_1, B]$$
$$\lambda'([q_1, A]) = A$$
$$\delta'([q_1, A], 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$$
$$= [q_0, A]$$
$$\lambda'([q_1, A]) = A$$

Hence,

	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A
$[q_0, B]$	$[q_0, A]$	$[q_1, B]$	B
$[q_1, A]$	$[q_1, B]$	$[q_0, A]$	A

$$\delta'([q_1, B], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$
$$= [q_1, B]$$
$$\lambda'([q_1, B]) = B$$
$$\delta'((q_1, B), 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$$
$$= [q_0, B]$$
$$\lambda'([q_1, B]) = B$$

Finally the transition table will be ,

$$\delta'((q_0, A), 1) = [\delta(q_0, 1), \lambda(q_0, 1)]$$
$$= [q_1, B]$$
$$\lambda'([q_0, A]) = A$$

Hence we can show a partial transition table as,

	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A

Continuing in this fashion we can calculate δ' and λ' as follows -

$$\delta'([q_0, B], 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$
$$= [q_0, A]$$
$$\lambda'[q_0, B]) = B$$
$$\delta'((q_1, B), 1) = [\delta(q_0, 1), \lambda(q_0, 1)]$$
$$= [q_1, B]$$
$$\lambda'([q_0, B]) = B$$

Hence

	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A
$[q_0, B]$	$[q_0, A]$	$[q_1, B]$	B

State \ I/P	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A
$[q_0, B]$	$[q_0, A]$	$[q_1, B]$	B
$[q_1, A]$	$[q_1, B]$	$[q_0, A]$	A
$[q_1, B]$	$[q_1, B]$	$[q_0, A]$	B

The transition diagram will be,

Fig. 2.64

Example 2.32 : *Convert the following Mealy machine into equivalent Moore machine.*

Fig. 2.65

Solution : We will write the transition table for given transition graph -

	Input 0	Output	Input 1	Output
q_0	q_1	N	q_2	N
q_1	q_1	Y	q_2	N
q_2	q_1	N	q_2	Y

Now we will find out the states and corresponding outputs for Moore machine. The states for Moore machine will be -

$[q_0, N]$, $[q_0, Y]$, $[q_1, N]$, $[q_1, Y]$, $[q_2, N]$, $[q_2, Y]$

Now let us calculate δ' and λ' for all the above given states -

$$\delta'([q_0, N)], 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$
$$= [q_1, N]$$
$$\lambda'([q_0, N]) = N$$

Similarly,

$$\delta'([q_0, N), 1) = [\delta(q_0, 1), \lambda(q_0, 1)]$$
$$= [q_2, N]$$
$$\lambda'([q_0, N]) = N$$

The partial transition table is -

State \ I/P	0	1	Output
$[q_0, N]$	$[q_1, N]$	$[q_2, N]$	N

Now, for remaining states the corresponding transitions and outputs can be obtained as follows -

$$\delta'([q_0, Y], 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$
$$= [q_1, N]$$
$$\lambda'([q_0, Y]) = Y$$
$$\delta'([q_0, Y], 1) = [\delta(q_0, 1), \lambda(q_0, 1)]$$
$$= [q_2, N]$$
$$\lambda'([q_0, Y]) = Y$$
$$\delta'([q_1, N)], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$
$$= [q_1, Y]$$
$$\lambda'([q_1, N]) = N$$
$$\delta'([q_1, N), 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$$
$$= [q_2, N]$$
$$\lambda'([q_1, N]) = N$$

$$\delta'([q_1, Y], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$
$$= [q_1, Y]$$
$$\lambda([q_1, Y]) = Y$$
$$\delta'([q_2, N], 0) = [\delta(q_2, 0), \lambda(q_2, 0)]$$
$$= [q_1, N]$$
$$\lambda'([q_2, N]) = N$$
$$\delta'([q_2, N], 1) = [\delta(q_2, 1), \lambda(q_2, 1)]$$
$$= [q_2, Y]$$
$$\lambda'([q_2, N]) = N$$
$$\delta'([q_1, Y], 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$$
$$= [q_2, N]$$
$$\lambda'([q_1, Y]) = Y$$
$$\delta'([q_2, Y], 0) = [\delta(q_2, 0), \lambda(q_2, 0)]$$
$$= [q_1, N]$$
$$\lambda'([q_2, Y]) = Y$$
$$\delta'([q_2, Y], 1) = [\delta(q_2, 1), \lambda(q_2, 1)]$$
$$= [q_2, Y]$$

$$\lambda'([q_2, Y]) = Y$$

The transition table can be draw as -

State \ Σ	0	1	Output
$[q_0, N]$	$[q_1, N]$	$[q_2, N]$	N
$[q_0, Y]$	$[q_1, N]$	$[q_2, N]$	Y
$[q_1, N]$	$[q_1, Y]$	$[q_2, N]$	N
$[q_1, Y]$	$[q_1, Y]$	$[q_2, N]$	Y
$[q_2, N]$	$[q_1, N]$	$[q_2, Y]$	N
$[q_2, Y]$	$[q_1, N]$	$[q_2, Y]$	Y

Unit - 3

Regular Set :-

Any Set that Represent the Value of the Regular Expression is called as Regular Set

$M = (S, T, F, S_0, F)$

where $S \Rightarrow$ Finite Set of State

$T \Rightarrow$ Input Symbol / Alphabet

$F \Rightarrow$ Mapping Function $S * T$ to S

$S_0 \Rightarrow$ Starting Symbol

$F \Rightarrow$ Final Symbol

Property of Regular Set :-

Property 1 : The union of two Regular Set is Regular

Proof

Let us take two Regular Expression

$RE = a(aa)^*$ & $RE = (aa)^*$

So $L_1 = \{a, aaa, aaaaa \ldots\}$ (String of odd length excluding NULL)

$L_2 = \{\varepsilon, aa, aaaa, aaaaaa \ldots\}$ (String of even length including NULL)

$L_1 \cup L_2 = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$ (String of all possible length including N

$RE(L_1 \cup L_2) = a^*$

Hence its proved.

Scanned with CamScanner

Property 4 : The difference of two Regular Set is Regular

Proof :-

Let us take two Reg Expression

$RE_1 = a(a^*)$ $RE_2 = (aa)^*$

So $L_1 = \{a, aa, aaa, aaaa \ldots\}$ {all possible length excluding NULL)

$L_2 = \{\varepsilon, aa, aaaa, aaaaaa \ldots\}$ (all possible even length including NULL)

$L_1 - L_2 = \{a, aaa, aaaaa, aaaaaaa \ldots\}$ (Strings of all odd length excluding NULL)

$RE(L_1 - L_2) = a(aa)^*$

Hence, proved

Property 5 : The Reversal of a Regular Set is Regular.

Proof :- we have to prove L^R is also regular if L a regular set

$L = \{01, 10, 11, 10\}$

$RE(L) = 01 + 10 + 11 + 10$

$L^R = \{10, 01, 11, 01\}$

$RE(L^R) = 01 + 10 + 11 + 10$

Hence its proved

Scanned with CamScanner

Property 6: The closure of a Regular Set is Regular.

Proof:-

if $L = \{a, aaa, aaaaa \ldots\}$ (String of odd length excluding NULL)

ie $RE(L) = a(aa)^*$

$L^* = \{a, aa, aaa, aaaa, aaaaa \ldots\}$ (String of all length excluding NULL)

$RE(L^*) = a(a^*)$

Hence Proved

Property 7: The concatenation of two Regular Set is Regular

<u>Proof</u>:

Let $RE_1 = (0+1)^* 0$ & $RE_2 = 01(0+1)^*$

Here $L_1 = \{0, 00, 10, 000, 010 \ldots\}$ (Set of Strings ending in 0)

$L_2 = \{01, 010, 011 \ldots\}$ (Set of String begin with 01)

$L_1 L_2 = \{001, 0010, 0011, 0001, 00010, \ldots\}$

$RE = (0+1)^* 001 (0+1)^*$

Hence proved.

Scanned with CamScanner

eg Construct the R.E for the language accepting all the String combination which 'a's' over the set $\Sigma = \{a\}$

Soln

$L = \{\varepsilon, a, aa, aaa \ldots\}$

$R = a^*$

eg 2 Design the R.E for the language accepting all combination of "a's" except null String over $\Sigma = \{a\}$

Soln

$L = \{a, aa, aaa \ldots\}$

$R = a^+$

eg 3 Design the R.E for language containing all the String containing any number of a's and b's

Soln

$R.L = \{\varepsilon, a, aa, ab, b \ldots\}$

$= (a+b)^*$

eg 4: Construct the R.E for the Lang accepting all the String which are ending with 00 over the Set $\Sigma = \{0, 1\}$

$L = \{01, 001, 011\} 00$

$= (0+1)^* 00$

Scanned with CamScanner

Regular Expression:-

Let Σ be an alphabet which is used to denote the input set. The Regular expression over Σ can be defined as follows.

$\varphi \Rightarrow$ Reg. Exp which denotes the empty set

$\varepsilon \Rightarrow$ Reg. Exp & denote the set $\{\varepsilon\}$ and its null String

For each 'a' in Σ is 'a' regular Express and denote by set $\{a\}$

If r & s are Reg Exp denoting the lang L_1 and L_2 respectively

union $r+s$ equivalent $L_1 \cup L_2$ i.e. union

Concatenation

$r.s$ equivalent to $L_1 L_2$

closure

r^* equivalent to L^*

Kleen closure

The r^* is known as Kleen closure which indicate occurrence of r for number of times

Scanned with CamScanner

eg 5, If L = { The Language Starting and ending with a & having any combination of b's between that what is r?

$L = \{aba, abba, abbba \ldots\}$

$r = ab^* a$

eg 6 Describe the Simple English the Language represented by the following Regular Expression

$r = (a+b)^*$

Soln

$L_r = \{a, aba, abab, aab \ldots\}$

eg 7 Write R.E to denote the Language Over Σ, where $\Sigma = \{a, b, c\}$ in which every String will be Such that any number of a's followed by any number of b's is followed by any number of

$L = \{abc, aabcc, aaabbbccc \ldots$

$= a^* b^* c^*$

Scanned with CamScanner

Finite Automata to Regular Expression Conversion

Constrain:

While converting from FA (DFA / NFA) we always write Regular Expression for Final State.

A Stage having more than one edge into a Single Direction

A Stage having Consequtive Edges:-

We can write RE by using multiplication for Consequence edges:-

Loops

There are two types of loop

Self Loop

for Loops we use closure (*) Sign

$\Rightarrow a c^* b$

Scanned with CamScanner

If a State has more than one Self loop

State ~~rep~~ q_1 can repeat any combination of a & b then:- $a\,[c+d]^*\,b$

2) Loop between the two States:

2 ways

1, Resolving to Left Side

$\Rightarrow$

$\Rightarrow (ab)^* a$

2, Resolving to ~~hight~~ Right Side

$\Rightarrow$

$\Rightarrow a(ba)^*$

$(ab^*)^*a = a(ba)^*$

If final State is middle State with the Loop

$a(b + a(a+b)^* b)^*$

Scanned with CamScanner

4, If finite Automata has more than one final State

In this case we have to write Regular Expression for all the state and combine it the result

In other case we can convert them into Single final State using ε Symbol

eg

Soln

[∴ The State q_3 have no does not reach final State So we consider it as Trap / Dead State]

R.E for q_0 [Assume q_1 & q_2 are non final)

$q_0 = [1 + 01]^*$

R.E for q_1 [assume q_0 & q_2 are Non final]

$q_1 = [1 + 01]^* 0$

RE for q_2 [assume q_0 & q_1 are non final]

$q_2 = [1 + 01]^* 00^*$

$RE = [1 + 01]^* + [1 + 01]^* 0 + [1 + 01]^* 00^*$

$= [1 + 01]^* [\varepsilon + 0 + 00^*]$

Scanned with CamScanner

2 Convert FA to Regular Expression :-

Step 1

Combine the Consequtive State by multiplication

RE for q_4 [Consume Remaing NF]

$$q_4 \Rightarrow (aa+bb)(a+b)^* a$$

RE for q_5

$$q_5 = (aa+bb)(a+b)^* b$$

RE for q_6 $q_6 = (aa+bb)(a+b)^* c$

Combain of all the Result

$$R.E = [aa+bb][a+b]^* a + [aa+bb][a+b]^* b + [aa+bb][a+b]^* c$$

$$= [aa+bb][a+b]^* [a+b+c]$$

[Take common Term]

Scanned with CamScanner

Problem for Reverse Loop -

Problem 1

$$[b + ac^*a]^* \, ac^*a$$

2)

[Initial & final State Same]

$$ba^* \, ba^* \, b$$

$$\boxed{\Rightarrow [a + ba^*ba^*b]^*}$$

With Dead / Trap State

Scanned with CamScanner

Problem on Loop

For State $q_1 \rightarrow$

$$(0+11)^* 1 (01^* 0) = x$$

For State $q_2 \Rightarrow$

$$R = (0+11)^* 1 (01^* 0) * 01^*$$

Combine

$$(0+11)^* 1 (01^* 0) + (0+11)^* 1 (01^* 0) * 01^*$$

ARDEN'S Theorem:-

If P & Q are two Regular Expression over Σ and P does not contain ε then the following equation in R given by $R = Q + RP$ has a unique Solution That is $R = QP^*$

$$R = Q + RP \rightarrow (1)$$

Scanned with CamScanner

Sub $R = QP^*$ in ①

$= Q + QP^* P$

$\Rightarrow Q(\varepsilon + P^* P)$ $[\varepsilon + R^* R = R^*]$

$= QP^*$

$\Rightarrow$ Hence its proved

② prove $R = QP^*$ is Unique Solution

$R = Q + RP \longrightarrow$ ②

Gn the $R = QP^* \longrightarrow$ ①

Sub ② in ①

$R = Q + [Q + RP]P$

$= Q + QP + RP^2$. Sub $R = [Q + RP]$

$= Q + QP + [Q + RP]P^2$

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

$\vdots$

$= Q + QP + QP^2 + QP^3 \ldots QP^n + QP^* P^{n+1}$

[Take Q as common]

$= Q[\varepsilon + P + P^2 \ldots P^n + P^* P^{n-1}]$

$= QP^*$

Scanned with CamScanner

NFA to Regular Expression Conversion

1

<u>Soln</u> [check incoming transition]

$q_3 \Rightarrow q_2 a \longrightarrow (1)$

$q_2 = q_1 a + q_2 b + q_3 b \longrightarrow (2)$

$q_1 \Rightarrow q_1 a \quad q_2 b + \varepsilon \longrightarrow (3)$

$(1) \Rightarrow q_3 = q_2 a$ [Sub q_2 value]

$= (q_1 a + q_2 b + q_3 b) a$

$= q_1 aa + q_2 ba + q_3 ba \longrightarrow (4)$

[Sub value of q_3 from (1)]

$(2) \Rightarrow q_2 = q_1 a + q_2 b + q_3 b$

$= q_1 a + q_2 b + (q_2 a) b$

$\Rightarrow q_1 a + q_2 b + q_2 ab$

$q_2 \Rightarrow q_1 a + q_2 (b + ab)$ [$R = Q + RP$]

$q_2 \Rightarrow q_1 a \cdot (b + ab)^* \longrightarrow (5)$ [by aderi Theorem] $R = QP^*$

$(3) \Rightarrow q_1 = \varepsilon + q_1 a + q_2 b$ [Sub q_2 value in q_1]

$q_1 = \varepsilon + \underset{R}{q_1} a + \underbrace{(a(b + ab)^*)}_{P} b$ $R = Q + RP$

$= \varepsilon (a + a(b + ab^*)b)^*$ $R = QP^*$

Scanned with CamScanner

$$q_1 = (a + a(b+ab)^* b)^* \longrightarrow ⑥ \qquad (\epsilon \cdot R = R)$$

Sub all Value for final State

$$q_3 = \underline{\underline{q_2}}\, a \qquad [\text{Sub eq } 5]$$

$$= q_1 a (b+ab)^* a$$

$$q_3 = (a + a(b+ab)^* b)^* a(b+ab)^* a$$

{Sub Value q_1 from ⑥}

= Regular Exp for NFA

DFA to Regular Expression Conversion

<u>Soln</u> $q_1 = \epsilon + q_2 b + q_3 a \longrightarrow ①$

$q_2 = q_1 a \longrightarrow ②$

$q_3 = q_1 b \longrightarrow ③$

$q_4 = q_2 a + q_3 b + q_4 ab$

Take final State

$① \Rightarrow q_1 = \varepsilon + q_2 b + q_3 a$ [Sub q_2 q_3 value]

$= \varepsilon + q_1 ab + q_1 ba$

$= \varepsilon + q_1 [ab + ba]$ [by adern Theorem]

$q_1 = \varepsilon [ab+ba]^*$

Scanned with CamScanner

$\epsilon R = R$

$q_1 = \epsilon\ (ab + ba)^*$

$q_1 = (ab + ba)^*$

Required Regular Expression

[incoming δ]

Soln

$\Rightarrow q_1 = \epsilon + q_1 0 \rightarrow (1)$

$q_2 \Rightarrow q_1 1 + q_2 1 \rightarrow (2)$

$q_3 = q_2 0 + q_3 0 1 \rightarrow (3)$

~~Not~~ final State [By adern Theorem]

$\frac{q_1}{P} = \frac{\epsilon}{Q} + \frac{q_1}{R}\frac{0}{P}$

$= \epsilon 0^*$ [$\epsilon R = R$]

$q_1 = 0^* \rightarrow (4)$

Not final State

$q_2 = q_1 1 + q_2 1$ [Sub q_1 value]

$\frac{q_2}{R} = \frac{0^* 1}{Q} + \frac{q_2}{R}\frac{1}{P}$ [from eq (4)] [By aderis Theorem]

$= (3)$

$q_2 = 0^* 1 (1)^* \rightarrow (5)$

Combain the Result (4) & (5)

$= 0^* + 0^* 1 (1)^*$

$= 0^* (\epsilon + 11^*) = 0^* (1)^* = 0^* 1^*$

Scanned with CamScanner

DFA to RE: by Rajji Theorem:-

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

$i \Rightarrow$ Starting State $\quad j \Rightarrow$ final State $\quad K \Rightarrow$ No of nodes

$R \Rightarrow$ Regular Expression

$$R_{12}^{2} = R_{12}^{1} + R_{12}^{1}\left[R_{22}^{1}\right]^{*} R_{22}$$

Possiblity

$$R_{11}^{0} = 1 + \varepsilon \qquad \varphi \Rightarrow \text{No Path}$$

$$R_{12}^{0} = 0$$

$$R_{21}^{0} = \varphi$$

$$R_{22}^{0} = 0 + 1 + \varepsilon$$

$\underline{\underline{K=1}}$

$$R_{12}^{1} = R_{12}^{0} + R_{11}^{0}\left[R_{11}^{0}\right]^{*} R_{12}$$

$$= 0 + (1+\varepsilon)(1+\omega)^{*}(0)$$

$$= 0 + (1+\varepsilon)\,1^{*}\,0$$

$$= 0 + 1^{*}\,0$$

$$= 0 + 1^{*}\,0$$

$$\boxed{R_{12}^{1} = 1^{*}\,0}$$

Scanned with CamScanner

$$R_{22}^{1} = R_{22}^{0} + R_{21}^{0}\,[R_{11}^{0}]^{*}\,R_{12}^{0}$$

$$= (0+1+\varepsilon) + \varphi\,[1+\varepsilon]^{*}\,0$$

$$= (0+1+\varepsilon)$$

[$\varphi \times$ Something $= \varphi$] anything

$$R_{22}^{1} = (0+1+\varepsilon$$

$$R_{12}^{2} = R_{12}^{1} + R_{12}^{1}\,[R_{22}^{1}]^{*}\,R_{22}^{1}$$

$$= 1^{*}0 + [1^{*}0\,[\varepsilon+0+1]^{*}[\varepsilon+0+1]]$$

$$= 1^{*}0 + [1^{*}0\,[\varepsilon+0+1]^{*}]\,[\varepsilon+0+1]]$$

$$= 1^{*}0 + [1^{*}0\,[0+1]^{*}[\varepsilon+0+1]] \qquad \varepsilon + R = R$$

$$= 1^{*}0 + [1^{*}0\,(0+1)^{*}]$$

$$= 1^{*}0 + 1^{*}(0+1)^{*}$$

$$\boxed{R_{12}^{2} = 1^{*}0\,(0+1)^{*}}$$

Scanned with CamScanner

Regular Expression To Finite Automata

Regular Expression to FA

① $(a+b)$

② $a.b$

③ a^*

Convert the following Reg Expression to their Equivalent Finite Automata

1, ba^*b

$L = \{bb, bab, baab \ldots\}$ Relevant [possible input]

④ $(a+b)c$

$L = \{ac, bc\}$

Scanned with CamScanner

③ $a(bc)^*$

$L = \{ a, abc, abcbc, abcbc, abcbc \ldots \}$

An Example Proof using Indentities of Regular Expression

Prove That $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$ is equal to $0^*1(0+10^*1)^*$

<u>Proof</u>

$LHS = (1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$

(Take the Common Term)

$= (1+00^*1)\left[\varepsilon + (0+10^*1)^*(0+10^*+1)\right]$

$\varepsilon + R^*R = R^*$

$= (1+00^*1)(0+10^*1)^*$

$= (\varepsilon \cdot 1 + 00^*1)(0+10^*1)^*$ $\qquad \varepsilon \cdot R = R$

$= (\varepsilon \cdot 00^*)(\varepsilon 1(0+10^*1)^*$

$= \cancel{(\varepsilon + 00^*)}$ $\qquad \varepsilon + R^*R = R^*$

$= 0^*1(0+10^*1)^*$

$LHS = RHS$

Hence its proved.

Scanned with CamScanner

Regular Expression To Finite Automata
[Thompson's Construction]

① $R = \varphi$

{φ - { φ ⇒ No path}

② $R = \varepsilon$

{ε ⇒ without input moving to next State}

③ $R = a$

④ $R = a + b$

$r_1 = a \qquad r_2 = b$

$r_1 = a$

$r_2 = b$

Combain $r_1 + r_2$

Scanned with CamScanner

⑤ $R = a^*$

$L = \{\varepsilon, a, aa, aaa \ldots\}$

$r_1 = a$

For any Combination of 'a'

Start State $= q_1$ Final State $= q_2$

q_2 to q_1

$L = \{a, aa, aaa \ldots$

$L = \{\varepsilon, a, aa, aaa \ldots\}$

) $R = a.b$

$r_1 = a$

$r_2 = b$

$r = a.b$

Scanned with CamScanner

a

$R = (a+b)(c+d)$

$r = (a+b)$

$r = (c+b)$

Combain the Result

Scanned with CamScanner

Pumping Lemma for Regular Grammar

Theroem : Let L be a RL Then there exist a Constant 'p' such that for every String w in L & - $|w| \geq p$

we can break 'w' into three String $w = xyz$ such that

[w - String
$|w|$ - length of String]

$|y| > 0$

$|xy| \leq p$

For all $k \geq 0$ the String xy^kz is also in L

Application

Pumping Lemma is to be applied to show that certain Languages are not regular. It should never be used to show a lang is regular.

If L is regular, it satisfies pumping Lemma

If L does not satisfy pumping Lemm

its not Regular

To Prove the Lang Not Regular

→ At first we have to assume that L is regular

→ So, the pumping Lemma should hol function L

Scanned with CamScanner

→ It has to have a pumping Length (Say P)

⇒ All String longer than P can be pumped

→ Now find a String w in L such that $|w| \geq P$

→ Divide w into xyz

→ Show the $xy^iz \in L$ function some i

⇒ Then consider all ways that w can be divide into xyz

⇒ Show that none of these can satisfy all 3 pumping condition at the same time

⇒ W cannot be pumped ⇒ contradiction

$L = \{a^n b^n / n \geq 1\}$ is Not Regular using pumping Lemma.

Sol

Assume that L is Regular

Pumping length = P

assume $P = 7$

$S = a^P b^P$

$\therefore S = \{a\,a\,a\,a\,a\,a\,a\,b\,b\,b\,b\,b\,b\,b\}$

Scanned with CamScanner

Case 1 - The Y is in the 'a' part [we are going to Split into 3 there was no constrain for Splitting but for position x - 1st Y - 2 parts Z - last part]

$\underbrace{aa}_{x}\ \underbrace{aaaaa}_{Y}\ \underbrace{bbbbbbb}_{z}$

Case 2: The Y is in 'b' part

$\underbrace{aaaaaaa}_{x}\ \underbrace{bbbbbb}_{Y}\ \underbrace{b}_{z}$

Case 3: The Y is in the 'a' & 'b' part

$\underbrace{aaaa}_{x}\ \underbrace{aaa\ bb}_{Y}\ \underbrace{bbbbb}_{z}$

$x = aa \quad Y = aaaaa$

$z = bbbbbbb$

i = 2 case 1

$xy^i z = xy^2 z$

aaaaaaa aaaaabbbbbbbb

12(a's) $\neq$ 7(b's)

i = 2

$xy^i z$ in Case 2

$x = aaaaaaa$

$Y = bbbbb$

$z = bb$

$xyz = xy^2 z$

aaaaaaabb bbbb bbbb b

7'(a's) $\neq$ 12'(b's)

=2 case 3

$xyz = xy^2 z$

aa aaaaa aabb aabb bbbbb

~~a's~~ ~~a~~ It doesnot follow patte

$|xy| \leq P$

Hence its proved this Lang Not Regular

Scanned with CamScanner

Show that $L = \{0^n 1^{n+1} \mid n > 0\}$ is not Regular ?

Sol:

assume L is Regular Language

Pumping Length $p = 3$

$S = 0^p 1^{p+1}$

$p = 3$

$S = 0^3 1^4$

$S = \{0001111\}$

Case:

$\underbrace{\overbrace{000}^{xy^2z}}_{x\ y} \underbrace{1111}_{z}$

$\Rightarrow x = 0 \quad y = 00 \quad z = 1111$

$\Rightarrow 0\ 00\ 00\ 1111$

$\Rightarrow 5 \neq 4$

$|y| > 0$ ✓

$|xy| \leq p$

$5 \not\leq 3$ ✗

L is Not Regular

Scanned with CamScanner

Kleene Closure:-

If L_1 is regular Lang its Kleen Closure L_1^* will also be regular.

eg $L_1 = (a \cup b)$

$L_2^* = (a \cup b)^*$

Complement:-

If $L(G)$ is regular Lang its complement $L'(G)$ will also be regular.

eg $L(G) = \{a^n \mid n > 3\}$

$L'(G) = \{a^n \mid n <= 3\}$

Difference:

If L and M are regular Languages then so

L - M = String is L but not M

Let A & B be a DFA whose Lang are L & M respectively

Construct C, the product automaton of A & B make the final State of C be pair wher A - State is final but B - State is not

Positive Closure:

R3 is regular Expression whose lan is L, M. R^+ is regular expression who Lang L^+

Scanned with CamScanner

Reverse Operator

Given lang L, L^R is the set of string whose reversal is in L

eg $L = \{0, 01, 100\}$

$L^R = 0, 10, 001\}$

Homomorphism:-

A homomorphism on an alphabet is a function that gives string for each symbol in that alphabet $h(L) = \{h(w) : w \in L$

eg $h(0101) = ababab$

eg 2 $\Sigma = \{a, b\}$, $\Delta = \{0, 1, 2\}$ | $L = a^*b$

$= h(a) = 01$ $h(b) = 112$ | $h(L) = h(a^*b) = h(a^*)h(b)$

$= h(a)^* h(b) = (01)^* 112$

Inverse homomorphism:

Let h be an homomorphism and L a language whose alphabet is the output language of h $h^{-1}(L) = \{x \mid h(x)$ is in $L\}$

eg Let $\Sigma = \{0, 1, 2\}$, $\Delta = (a, b)$

$h(0) = a$ $h(1) = \underline{ab}$ $h(2) = ba$

$L = a(ba)^*$ $h^{-1}(L) = ?$

$L = a\{\epsilon, ba, babc, bababa \ldots\}$

$= \{a, aba, ababa, abababa \ldots\}$

$= a, 1002, 110, \}$ $\Rightarrow 1^*0$, 02^*, 1^*02^*

$= \{0, 10, 110, 022$

$02, 102$

$a^n b^m$ $n \geq 0$ & $m \geq 0$

Closure Properties of Regular Language

Union:- If L_1 and L_2 are two regular Lang and their union $L_1 \cup L_2$ will also be regular.

eg $L_1 = \{a^n \mid n \geq 0\}$ & $L_2 = \{b^n \mid n \geq 0\}$

$L_3 = L_1 \cup L_2$

$= \{a^n \cup b^n \mid n \geq 0\}$

$\Rightarrow$ Its a regular.

Concatenation

If L_1 and L_2 are two regular Lang their concatenation $L_1 . L_2$ will also be regular.

eg $L_1 = \{a^n \mid n \geq 0\}$ & $L_2 = \{b^m \mid m \geq 0$

$L_3 = L_1 . L_2$

$= \{a^n b^m \mid m \geq 0 \text{ \& } n \geq 0\}$

Its regular.

Intersection

If L_1 and L_2 are two regular Lang their intersection $L_1 \cap L_2$ will be regular

$L_1 = \{a^m b^n \mid n \geq$ & $m \geq 0\}$ &

$L_2 = \{a^m b^n \cup b^n a^m \mid n \geq 0$ & $m \geq 0\}$

$L_3 = L_1 \cap L_2 = \{a^m b^n \mid n \geq 0$ & $m \geq 0$ is Regul

3.b 10

Regular Linear Grammar and FA Equivalence

Let $M = (\{q_0 q_1 \ldots q_n\}, \Sigma, \delta, q_0, F)$ be a DFA.

The Equivalent Grammar can be Constructed from given DFA Such that production corresponded to transition

By the transition terminating we can encountered the final State

Let

$G = (\{A_0, A_1 \ldots A_n\}, \Sigma, P, A_0)$

where P is the Set of production Rule

1, $A_i \to aA_j$ is a production Rule if

$\delta(q_i, a) = q_j$ & $q_j \in F$

2, $A_i \to aA_j$ and $A_i \to a$ are production rule if $\delta(q_i, a) = q_j$ where $q_j \in F$

Thus given grammar is accepted by DFA M

Problem on FA to Regular Grammar.

eg 4.3 Construct Regular Grammar for given DFA

Soln

$G = (V, T P S)$ $\quad q_0 = A_0 \quad q_1 = A_1$

$V = \{A_0 \; A_1\}$

$T = \{a, b\}$

$A_0 \rightarrow aA_0 \quad A_1 \rightarrow aA_1 | a$

$A_0 \rightarrow b | A_1 \quad A_1 \rightarrow bA_1 | b$

$A_0 \rightarrow b$

$A_0 \rightarrow aA_0 | bA_1 | b$

$A_1 \rightarrow aA_1 | bA_1 | a | b$

② Construct a Regular Grammar for given FA

$q_1 = A_0$

$q_2 = A_1$

$q_3 = A_2$

$q_4 = A_3$

Soln

$V = (V T P S)$ where

$V = \{A_0, A_1, A_2, A_3\}$

Production

$A_0 \rightarrow bA_1 | aA_2$

$A_1 \rightarrow bA_3 | aA_2 | b$

$A_2 \rightarrow bA_1 | aA_3 | a$

$A_3 \rightarrow aA_3 | bA_3 | a | b$

Scanned with CamScanner

3-b-4/10-b

3) Construct Regular grammar for

Soln

$V = \{A_0, A_1, A_2, A_3\}$

$A_0 \to 0A_1$

$A_1 \to 1A_1 \mid 0A_2$

$A_2 \to 0A_2 \mid 1A_3 \mid 1$

$A_3 \to 0A_2 \mid 1A_1$

Construction of FA from Regular Grammar.

Let $G = (V, \Sigma, P, A_0)$ be Regular grammar

We can Construct DFA M whose

① States correspond to Variable

② Initial State correspond to A_0

③ Transition M correspond to production P

If there is a production from $A_i \to a$ the corresponding transition terminal at a new State (final State)

DFA $\Rightarrow$ $M = (\{q_0, q_1 \cdots q_n q_f\}, \Sigma, \delta, q_0, \{q_f\})$

Scanned with CamScanner

Thus δ defined

① Each production $A_i \to a A_j$ (trans from q_i to q_j with label a)

② Each production $A_k \to a$ (δ from q_k to q_f with label a)

Construct a FA recognizing $L(G)$ where G is the grammar

$$S \to aS \mid bA \mid b$$
$$A \to aA \mid bS \mid a$$

Soln

$$M = (\{q_0\ q_1\ q_f\}, \{a, b\}, \delta, q_0, \{q_f\})$$

Scanned with CamScanner

Context Free Grammar :-

⇒ Context Free grammar is a formal Grammar which is used to generate all possible String in a given formal Language

⇒ CFG Can be define by four tuples as

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

$V/N \rightarrow$ Non terminal

$T \rightarrow$ Terminal

$P \rightarrow$ Production Rule

$S \rightarrow$ Start Symbol

⇒ If production Rule

$A \rightarrow \alpha$

$\alpha \in (V+T)^*$

eg $A \rightarrow \varepsilon$

$A \rightarrow BCD$

Context

① $aAb \rightarrow bb$

$a \rightarrow$ left context of A

$b \rightarrow$ Right Context of A

② $\varepsilon A \varepsilon$

No Context

Scanned with CamScanner

Capabilities of CFG

⇒ CFG is used to describe most of the programming Languages

⇒ If the grammar is properly designed then an efficient parser can be Constructed automatically

⇒ CFG is capable of describing nested Structures like balanced parentheses, matching begin-end, Corresponding if then-else & so on.

Properties of CFG :-

① Based on number of String it generates

If CFG is generating finite number of String then CFG is Non-Recursive.

If CFG can generate infinite number of String then the grammar is said to be Recursive grammar.

② Based on number of derivation tree

If there is only 1 derivation tree then the CFG is unambiguous

If there are more than 1 derivation tree then CFG is ambiguous.

Scanned with CamScanner

eg Recursive Grammar:-

① $S \to SaS$

$S \to b$

$L = \{b, bab, babab\}$

② $S \to Aa$

$A \to Ab | c$

$L = \{ca, cba, cbba \ldots\}$

eg Non Recursive Grammar:-

$S \to Aa$

$A \to b | c$

$L = \{ba, ca$

Context Free Language:-

In formal Language theory a Context Free language is Langugе generated by some CFG

The Set of all CFL is identical to the Set of Languages accepted by pushdown automata.

Conversion of Language to Grammar.

$L - \{a^n | n \geq 0\}$

Soln

$L = \{\varepsilon, a, aa \ldots\}$

$S \to aS | \varepsilon$

Scanned with CamScanner

3.b 1

ip = a

ip = aa

ip aaa

② L = { any Combination of a's & b's }

<u>Soln</u>

$L = (a+b)^*$

$L = \{ \varepsilon, a, b, ab, ba, aa, abb \ldots \}$

$S \rightarrow aS \mid bS \mid \varepsilon$

ip = a

ip = ab

ip = ba

abab

Scanned with CamScanner

Generating grammers from Language

1) $L = \{a^n b^m \mid n \geq 1, m \geq 0\}$

$n = 1$ no

$a^n \Rightarrow L = \{a, aa\}$

$L = aA / a$

$b^m \rightarrow b^0 \Rightarrow 1 = \varepsilon$

$L = \{\varepsilon, b, bb, bbb \ldots\}$

$B \rightarrow bB / \varepsilon$

$S \rightarrow AB$

$A \rightarrow aA / a$

$B \rightarrow bB / \varepsilon$

---

2) $L = \{a^n b^m \mid (n+m) \text{ is even}\}$

$(aa)^*(bb)^* + a(aa)^* b(bb)^*$

$\boxed{\text{even} + \text{even} = \text{even}}$ $\boxed{\text{odd} + \text{odd} = \text{even}}$

$S \rightarrow AB + aAbB$

$A \rightarrow aaA / \varepsilon$

$B \rightarrow bbB / \varepsilon$

NOTE

$L = (a)^* = \{\varepsilon, a, aa, aaa \ldots\}$ $A \rightarrow aaA / \varepsilon$

$(aa)^* = \{\varepsilon, aa, aaaa, \ldots\}$

$a(aa)^* = \{a, aaa, aaaaa \ldots\}$

Scanned with CamScanner

3) $L = \{a^n b^m \mid (n+m) \text{ is odd}$

$(aa)^* b (bb)^* + a(aa)^* (bb)^*$

$(E + o = o)$ $(o + E = o)$

$S \to AbB \mid aAB$

$A \to aaA \mid \varepsilon$

$B \to bbB \mid \varepsilon$

---

(4) $L = \{a^n b^m \mid n \geq 3 \quad m \geq 2\}$

$n = 3 \quad m = 2$

$L = \{aaabb, aaaabbb \ldots\}$

$S \to aaaAbbB$

$A \to aA \mid \varepsilon$

$B \to bB \mid \varepsilon$

---

5. $L = \{a^n b^n \mid n \geq 0\}$

$n = 1$

$L = \{\varepsilon, ab, aabb \ldots\}$

$S \to aSb \mid \varepsilon$

Scanned with CamScanner

5 $L = \{a^n b^m \mid n \geq m\}$

Soln

$L(G) = \{ab, aab, aabb, aaabb \ldots\}$

$S \to aAb \mid aSb$

$A \to aA \mid \varepsilon$

[NOTE $\Rightarrow A \to aA \mid \varepsilon$

$A \Rightarrow aA \Rightarrow aaA \Rightarrow aaA$

$L = a^*$

$n = m$

$n > m$

$n > m$

4 Find the grammar that generate all String of a's & b's where each String is Starting and ending with same Symbol

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

$[(a+b)^* = aA$

Soln

The grammar is

$S \to aAa \mid bAb \mid a \mid b$

any combination of b

$A \to aA \mid bA \mid \varepsilon$

Scanned with CamScanner

3. b (1 4)

5) Starting and Ending with diff Symbol

R.E =

6) Length of String is odd (a & b input)

RE = $[(a+b)(a+b)]^*(a+b)$

Soln

$(a+b) \Rightarrow A \rightarrow a/b$

$(a+b)(a+b) = B \rightarrow AA$

$[(a+b)(a+b)]^* \Rightarrow C \rightarrow BC/\varepsilon$

$A = a+b$

$C = [(a+b)(a+b)]^*$

the grammar is

$S \rightarrow AC$

$C \rightarrow BC/\varepsilon$

$B \rightarrow AA$

$A \rightarrow a/b$

7) String Starting with zero and total length of the String is even or if String Starting with 1 then total length of the String is odd (≤ 0, 1)

Soln Regular Expression

$\{0 + odd\} = even$

+

$\{1 + even\} = odd$

$RE = 0[(0+1)((0+1)(0+1))^*] +$

$1[((0+1)(0+1))^*]$

$(0+1) \Rightarrow A \rightarrow 0/1$

$(0+1)(0+1) \Rightarrow B \rightarrow AA$

Scanned with CamScanner

3.b 13

$[(0+1)(0+1)]^* = C \to BC \mid \varepsilon$

The grammar is

$S \to 0AC \mid 1C$

$C \to BC \mid \varepsilon$

$B \to AA$

$A \to 0 \mid 1$

II Converter of Language to Grammar

① Develop a grammar for the Language

$L_1 = \{ wcw^R$, w is in $(a,b)\}$

where w = word, c = Single character c w^R = word

Soln

$L = \{abcba, aabcbaa \ldots\}$

The grammar

$S \to aSa \mid bSb \mid c$.

input String abcba

abcba

Scanned with CamScanner

② Develop a grammar for the following Language:

$L = \{ a^n b^m c^m d^n$ where $m, n \geq 1 \}$

Sol:

$S \rightarrow aSd \mid aAd$

$A \rightarrow bAc \mid bc$

$L = \{ abbccd, aaabcddd \ldots$

aaabbcccddd

③ $L = \{ a^n b^n c^m d^m$ where $m, n \geq 1 \}$

$m \& n \geq = 0$

$G = \{ (S, A, B)(a\ b\ c\ d) \}$

$n = 0 = a^0 b^0 c^1 d^1$ not appli

$m = 1 = cd$

$S \rightarrow AB$

$A \rightarrow aAb \mid ab$

$B \rightarrow cBd \mid cd$

$m = 3 \quad n = 2$

aabbcccddd

Scanned with CamScanner

Derivation

A grammar derives a String by beginning with the Start Symbol & repeatly replacing a Nonterminal by the Right Side of a production for that Non terminal is called as derivation

A replacement according to a production is Known as Derivation

Two type of Derivation

1, Left most Derivation

2, Right most Derivation / Caronical Derivation

① Left most Derivation

In this derivation the leftmost Nonterminal is replaced at every Step

eg Consider the production

$E \rightarrow E+E \mid E*E \mid id$ The input String id+id+id

Soln $E \rightarrow E+E$

$\rightarrow E+E+E$

$\rightarrow id+id+E$

$\rightarrow id+id+id$

② Right most Derivation

In this derivation the ~~Left~~ Right most nonterminal is expanded first

$E \rightarrow E+\underline{E} \rightarrow E+E+\underline{E}$

$E+E+id$

$E+id+id \rightarrow id+id+id$

Scanned with CamScanner

3. bcl7

Derive the string 'aabbabba' for Left Most derivation & Right most derivation using CFG given by

$S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

$B \to b \mid bS \mid aBB$

Soln Left Most Derivation

S

$a\underline{B}$

$aa\underline{B}B$

$aabB$

$aabb\underline{S}$

$aabbaB$

$aabbabbS$

$aabbabbA$

$aabbabba$

$S \to aB$

$B \to aBB$

$B \to b$

$B \to bS$

$S \to aS$

$B \to bS$

$S \to bA$

$S \to a$

Derivation

Scanned with CamScanner

aa b/b S

aabbabS

$S \to aB$

$B \to bS$

Right Most Derivation

S

aB

aaBB

aaBbS

aaBbbA

aaBbba

aabSbba

aabbAbba

aabbabba

$S \to aB$

$B \to aBB$

$B \to bS$

$S \to bA$

$A \to a$

$B \to bS$

$S \to bA$

$A \to a$

② Derive the String 1000111 for left most & Right derivation using CFG where

$V = \{S, T\}$ $T = \{0, 1\}$

$P \to \{ S \to T00T$

$T \to 0T \mid 1T \mid \varepsilon \}$

Left Most

S

T00T

1T00T

10T00T

10ε00T

$S \to T00T$

$T \to 1T$

$T \to 0T$

$T \to \varepsilon$

Scanned with CamScanner

1000T	$T \to 1T$
10001T	$T \to 1T$
100011T	$T \to \varepsilon$
100111	

Right Most Derivation

$S \to$	$S \to T00T$
T00T	$T \to 1T$
T001T	$T \to 1T$
T0011T	$T \to 1T$
T00111T	$T \to \varepsilon$
T00111ε	$T \to 1T$
T00111ε	$T \to 0T$
1T00111ε	$T \to \varepsilon$
10T00111ε	
100111	

Scanned with CamScanner

3-b (19)

Construct the derivation tree for the String aabb abba for CGF given by

$S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

$B \to b \mid bS \mid aBB$

Soln

S	$S \to aB$
aB	$B \to aBB$
$aaBB$	$B \to b$
$aabbS$	$B \to bS$
$aabbaB$	$S \to aB$
$aabbabS$	$B \to bS$
$aabbabbA$	$S \to bA$
$aabbabba$	$A \to a$

aa bb a bb a

Scanned with CamScanner

3.b(20)

Sentential Forms:-

A Sentential form is the Start Symbol 'S' of a grammar (or) any String $(V \cup T)^*$ that can be derived from 'S'

Consider the Grammar $V = \{S, B\}$, $T = \{a, b\}$
$P = \{S \to aS,\ S \to B,\ B \to bB,\ B \to \varepsilon\}$ ~~up~~
input String abb

A derivation using this grammar might look like this

$S \to aS$ $\qquad$ $S \to B$

aB $\qquad$ $B \to bB$

abB $\qquad$ $B \to bB$

$abbB$ $\qquad$ $B \to \varepsilon$

abb

Sentential form
$\{S, aS, abB, abbB, abb\}$

Scanned with CamScanner

Method to find wheather a String belong to a grammar or not

① Start with the Symbol and Choose the closest production that matches to the given String

② Replace the Value with its most appropriate production. Repeat the process until the String is generated / until no other production left

Verify whether its the Grammar

$S \to 0B / 1A$

$A \to 0 / 0S / 1AA / \varepsilon$

$B \to 1 / 1S / 0BB$

Generate the String 00110101

Soln

$S \to 0B$ $\quad$ $S \to 0B$

$\to 0\,0BB$ $\quad$ $B \to 0BB$

$\to 001B$ $\quad$ $B \to 1$

$\to 0011S$ $\quad$ $B \to 1S$

$\to 00110B$ $\quad$ $S \to 0B$

$\to 00110S$ $\quad$ $B \to 1S$

$\to 0011010B$ $\quad$ $S \to 0B$

$\to 00110101$ $\quad$ $B \to 1$

00110101 =

Scanned with CamScanner

3.b(2)

Language:-

Set of String generate by any grammar is known as language.

Identify the Lang generate by grammar.

① $A \to aA \mid a$ find Lang

Soln $A \to a$

$\therefore L = \{a$

$A \to aA$

$L = \{aa, aaa \ldots$

$L = \{a, aa, aaa \ldots$

$L = a^+$

$S \to bS \mid \varepsilon$

Soln $S \to \varepsilon$

$L = \{\varepsilon$

$S \to bS$

$L = \{b, bb, bbb \ldots$

$\therefore L = \{\varepsilon, b, bb, bbb \ldots\}$

$L = a^*$

Scanned with CamScanner

4) $S \to \cancel{bS}\, AB$, $A \to a$, $B \to b$

$S \to AB$

$L = \{ab\}$

5) $S \to aS \mid bS \mid a$

Son

① $S \to aS$

$L = \{aaa\}$

$S \to aS \Rightarrow S \to aS$

$aaS \Rightarrow S \to a$

aaa

② $S \to aS$

$L = \{ab$

$S \to aS$

$ab \}$

aba

$S \to bS$

$S \to \varepsilon$

③ $S \to a$

$L = a$

④ $S \to bS$

$\to baS$

$\to baa$

$L = \{a, aaa, aba, baa \ldots$

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

5) $S \to aS \mid bS \mid \varepsilon$

① $S \to \varepsilon$

$L = \{\varepsilon$

2 $S \to aS$

$\to a \cdot \varepsilon$

$= a$

$L = \{\varepsilon, a$

Scanned with CamScanner

3) $S \to bS$

$\to b \cdot \varepsilon$

$\to b$

$L = \{\varepsilon, a, b$

4, $S \to baS$

$\to abS$

$\to ab\varepsilon$

$\to ab$

5, $S \to baS$

$\to baS$

$\to ba\varepsilon$

$\to ba$

$L = \{\varepsilon, a, b, ab, ba, bb, aa \ldots$

$(a+b)^*$

Scanned with CamScanner

UNIT - I LOGIC

logic is concerned with all kinds of reasoning, whether they be a legal arguments or mathematical proofs or conclusions in a scientific theory based upon set of hypothesis. It is the theoretical basis for many areas of computer science such as digital logic design, construction of computer programs & its verification, artificial intelligence, automata theory etc. It is the foundation on which a comp sys is built in which bits are switched on & off using 1's & 0's, in boolean logic, there are only two states on & off.

There are 2 main components of logic
(i) propositional logic (ii) predicate logic.
propositional logic deals with propositions & the the analysis of propositions.
predicate logic deals with predicate which are propositions containing variables

<u>propositional logic</u>: The study of propositional logic consist of syntax (grammar), sematics (meaning), inference rule & derivation.
propositional logic can be considered as a language of human reasoning. This language is based on symbols which consist of
i) propositional variables denoted by p, q, r, s (which are simple statement)
ii) propositional constants denoted by T, F
iii) connectives or basic logical operators denoted by $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$.

<u>Defn</u>: A proposition or statement is a declarative sentence that is either T or F but not both.

<u>Note</u>: sentences which are exclamatory, interrogative & imperative in nature are not proposition.

<u>Examples</u>: 1. $x^2+9=0$ (statement) $x+2=5$
2. $x^2+2x+1=0$ (statement).
3. chennai is the capital of TN (statement).
4. study Discrete mathematics (Not a statement)
5. open the door (" ")
6. can you speak tamil (" ")
7. wow! what a beautiful place (" ")

<u>primary statements</u> (Atomic statements)
A declarative sentence which cannot be split up into simple sentences are called primary statements.
In other words, primary statements do not contain logical connectives. (i.e, or, and, if, if & only if ...).
Eg: (i) Malini is a dancer (primary statement).
(ii) Malini is a dancer & also good singer (not a primary statement).

Compound statement: (Molecular proposition)

A proposition obtained by combining two or more propositions by means of logical connectives is called a compound statement.

Eg: Sridevi & Swathi went to their native place.

Connectives: Connective is an operation which is used to connect 2 or more than 2 statements. Simply it is called sentential connectives. It is also known as logical connectives or logical operators.

Five Basic connectives

logical connectives	Name	symbols	Type of operator
NOT	Negation	$\neg$ (or) $\sim$	unary
AND	Conjunction	$\wedge$	Binary
OR	Disjunction	$\vee$	Binary
If...then	Conditional (or) Implication	$\rightarrow$	Binary
If & only if	Biconditional	$\Leftrightarrow$ or $\leftrightarrow$	Binary

Truth table: The truth value of a proposition is either true (T) or False (F).

A truth table is a table that shows the truth value of a compound proposition & possible cases.

If the compound statement have 'n' variables then the truth value will have 2^n possibilities

Example: 1) If the compound statement have 2 variables P & Q then the possible truth value are in the table.

Truth table:

P	T	T	F	F
Q	T	F	T	F

2) If the compound statement have 3 variables P, Q, R then the possible truth value are

P	T	T	T	T	F	F	F	F
Q	T	T	F	F	T	T	F	F
R	T	F	T	F	T	F	T	F

1) **Negation ($\neg$ or $\sim$) (not):** If p is a proposition, then the negation of p is written as '$\neg p$' & is read as 'not p'. If p has truth value T then '$\neg p$' has truth value F & vice versa.

Example: 1) p: Today is saturday (T)

$\neg p$: Today is not saturday (F).

2) p : chennai is a city (T)

$\neg p$: chennai is not a city (F).

Truth table:

P	$\neg P$
T	F
F	T

Conjunction ($\wedge$) (and): If p & q are 2 propositions then the conjunction of p & q is the compound proposition p & q and is denoted by $p \wedge q$. If both p & q have the same truth value T, then $p \wedge q$ is T otherwise $p \wedge q$ is F.

Eg: ① p: Today is monday (T)

q: This is the month of july (T)

$p \wedge q$: Today is monday & the month of july (T).

② p: $3<5$ (T)

q: $2+3=6$ (F)

$p \wedge q = 3<5$ & $2+3=6$ (F).

Truth table:

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

Disjunction ($\vee$) (or): The disjunction of 2 statements P & Q is the statement denoted by $P \vee Q$ read as P or Q.

$P \vee Q$ has the truth value T if anyone of P or Q has truth value T & otherwise it is F.

Example: P: There are 10 birds.

Q: There are 5 birds

$P \vee Q$: There are 10 or 5 birds.

Truth table:

P	Q	$P \vee Q$
T	T	T
T	F	T
F	T	T
F	F	F

Conditional statement: ($\rightarrow$) (If...then).

If P & Q are any 2 propositions then the statement $P \rightarrow Q$ which is read as "if P then Q" is called as conditional statement, $P \rightarrow Q$ is F if P is T & Q is F. In other case $P \rightarrow Q$ is T.

Eg: P: It is hot (T)

Q: $5+3=6$ (F).

$P \rightarrow Q$: If it is hot then $5+3=6$ (F).

Truth table:

P	Q	$P \rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

Biconditional: ($\leftrightarrow$ or $\Leftrightarrow$) (iff)

If P & Q be any 2 statement then the

Statement $P \leftrightarrow Q$ which read as p "if & only if" is called biconditional statement. The statement $P \leftrightarrow Q$ is T whenever P & Q have the same truth values & otherwise is F.

Truth value:

P :	T	T	F	F
Q :	T	F	T	F
$P \leftrightarrow Q$:	T	F	F	T

Eg: P : Selvam can write the examinations (T)

Q : Selvam pays the fees (T)

$P \leftrightarrow Q$: Selvam can write the examinations if & only if he pays the fees (T).

Statement formula: A statement formula is an expression which is a string consisting of variables, parenthesis & connective symbols.

Well formed formula: A well formed formula can be generated by the following rules:

i) A statement variable standing alone is a well formed formula.

ii) If A is a well formed formula, then $\neg A$ is a well formed formula.

iii) If A & B are well formed formula, then $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$ & $(A \leftrightarrow B)$ are well formed formula.

iv) A string of symbols containing the statement variables, connectives & parenthesis is a well formed formula, iff it can be obtained by finitely many application of the above 3 steps.

Consider the following statement:

i) $\neg P \wedge Q$ is a wff.

ii) $(P \rightarrow Q) \wedge R$ is a wff.

iii) $P \rightarrow (Q \wedge R)$ is not wff. [$P \rightarrow (Q \wedge R)$ is wff or $(P \rightarrow Q) \wedge R$ is wff].

iv) $(P \rightarrow Q$ is not a wff. since one parenthesis in the right is missing while $(P \rightarrow Q)$ or $P \rightarrow Q$ are wff.

Problems: ① write the following statements in symbolic forms:

If either S. Pavithra takes calculus or sharnika takes sociology, then malathy will take english.

Soln: P : S. Pavithra takes calculus.

Q : sharnika takes sociology.

R : malathy takes English.

$(P \vee Q) \rightarrow R$

② If P: Malathy is rich

Q: Malathy is happy.

write in symbolic form:

a) Malathy is poor but happy.

b) Malathy is rich or unhappy.

c) Malathy is neither rich nor happy.

Soln: Given, P: malathy is rich — ①

$\neg P$: malathy is poor — ②

Given, Q: Malathy is happy — ③

$\neg Q$: Malathy is unhappy — ④

a) combining ② & ③, we get, malathy is poor & happy ie, $\neg P \wedge Q$.

b) Combining ① & ④, malathy is rich or unhappy ie, $P \vee \neg Q$

c) Combining ② & ④, malathy is neither rich nor happy ie, $\neg P \vee \neg Q$.

Construct the truth table:

1) $\neg(P \wedge Q)$

Soln

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

② $(P \wedge (P \to Q)) \to Q$.

P	Q	$P \to Q$	$P \wedge (P \to Q)$	$(P \wedge (P \to Q)) \to Q$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

③ $P \leftrightarrow (Q \vee P)$

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

④ $\neg P \vee (Q \to R)$.

P	Q	R	$\neg P$	$Q \to R$	$\neg P \vee (Q \to R)$
T	T	T	F	T	T
T	T	F	F	F	F
T	F	T	F	T	T
T	F	F	F	T	T
F	T	T	T	T	T
F	T	F	T	F	T
F	F	T	T	T	T
F	F	F	T	T	T

HW ① $\neg(P \vee Q) \wedge (P \vee R)$ Ans: [FFFFFFFT]

② $(P \to Q) \leftrightarrow (\neg P \vee Q)$ Ans [TTTT]

3. $(P \wedge Q) \vee (\neg P \wedge Q) \vee (P \wedge \neg Q)$ Ans: [T, T, T, F]

Tautologies & contradiction

Tautology: A logical expression is said to be a tautology if it is true under all possible assignment.

Eg:

P	$\neg P$	$P \vee \neg P$
T	F	T
F	T	T

$\therefore P \vee \neg P$ is a tautology.

Contradiction: A logical expression is said to be a contradiction if it is false under all possible assignment.

Eg:

P	$\neg P$	$P \wedge \neg P$
T	F	F
F	T	F

$\therefore P \wedge \neg P$ is a contradiction.

Problems: ① Determine which of the following compound propositions are tautologies & which of them are contradictions, using truth tables.

$((P \to q) \wedge (q \to r)) \to (P \to r)$.

P	q	r	$P \to q$	$P \to r$	$q \to r$	$(P \to q) \wedge (q \to r)$	$(P \to q) \wedge (q \to r) \to (P \to r)$
T	T	T	T	T	T	T	T
T	T	F	T	F	F	F	T
T	F	T	F	T	T	F	T
T	F	F	F	F	T	F	T
F	T	T	T	T	T	T	T
F	T	F	F	T	T	F	T
F	F	T	F	T	F	F	T
F	F	F	T	T	T	T	T

Since the truth value of the given statement is T for combinations of truth values of P, q, r.

$\therefore$ It is a tautology.

② $\underbrace{(\neg(q \to r) \wedge r)}_{A} \wedge (P \to q)$ — ①

P	q	r	$P \to q$	$q \to r$	$\neg(q \to r)$	A	①
T	T	T	T	T	F	F	F
T	T	F	T	F	T	F	F
T	F	T	F	T	F	F	F
T	F	F	F	T	F	F	F
F	T	T	T	T	F	F	F
F	T	F	T	F	T	F	F
F	F	T	T	T	F	F	F
F	F	F	T	T	F	F	F

The given statement contains only F as the truth value. ∴ It is a contradiction.

3. $((P \vee Q) \wedge (q \to r)) \wedge (q \to r)) \to r$ Tautology.

4. $P \to (P \vee q)$ Tautology

5. $(q \to P) \wedge (\neg P \wedge q)$ Contradiction.

6. $(\neg q \wedge (P \to q)) \to \neg P$ Tautology.

Equivalence of Formulae:

The two statement formulas A & B are equivalent iff $A \to B$ (or) $A \rightleftarrows B$ is a tautology. It is denoted by the symbol $A \Leftrightarrow B$ which is read as "A is equivalent to B".

Eg: $P \to Q \Leftrightarrow \neg Q \to \neg P$.

P	Q	$P \to Q$	$\neg Q$	$\neg P$	$\neg Q \to \neg P$
T	T	T	F	F	T
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

Tautological Implications:

A statement A is said to tautologically imply to a statement B iff $A \to B$ is a tautology.

we denote it as "$A \Leftrightarrow B$" which is read as "A implies B".

Note that, $A \Leftrightarrow B$ means a 2 way implication

$A \Rightarrow B$ means a 1 way implication.

Example: ① consider the statement $(P \wedge Q) \to P$

P	Q	$P \wedge Q$	$(P \wedge Q) \to P$
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	T

Since $(P \wedge Q) \to P$

$\therefore P \wedge Q \Rightarrow P$

② prove the following equivalence

$\underbrace{(P \to q) \wedge (P \to r)}_{A} \Leftrightarrow \underbrace{P \to (q \wedge r)}_{B}$ — I

P	q	r	$P \to q$	$P \to r$	$q \wedge r$	A	B	I
T	T	T	T	T	T	T	T	T
T	T	F	T	F	F	F	F	T
T	F	T	F	T	F	F	F	T
T	F	F	F	F	F	F	F	T
F	T	T	T	T	T	T	T	T
F	T	F	T	T	F	T	T	T
F	F	T	T	T	F	T	T	T
F	F	F	T	T	F	T	T	T

Since $(P \to q) \wedge (P \to r) \Leftrightarrow P \to (q \wedge r)$ is a tautology.

$(P \to q) \wedge (P \to r) \Leftrightarrow P \to (q \wedge r)$.

3. $p \to (q \to r) \Rightarrow (p \to q) \to (p \to r)$ — (I)

(A) (B)

p	q	r	$q \to r$	(A)	$p \to q$	$p \to r$	(B)	(I)
T	T	T	T	T	T	T	T	T
T	T	F	F	F	T	F	F	T
T	F	T	T	T	F	T	T	T
T	F	F	T	T	F	F	T	T
F	T	T	T	T	T	T	T	T
F	T	F	F	T	T	T	T	T
F	F	T	T	T	T	T	T	T
F	F	F	T	T	T	T	T	T

Since $p \to (q \to r) \to (p \to q) \to (p \to r)$ is a tautology $\therefore p \to (q \to r) \Rightarrow (p \to q) \to (p \to r)$.

4. $p \vee \neg(q \wedge r) \wedge \neg p \Rightarrow (\neg q \vee \neg r)$

5. $\neg(p \leftrightarrow q) \Leftrightarrow \neg p \leftrightarrow q$

6. $\neg(p \vee (\neg p \wedge q)) \Leftrightarrow \neg p \wedge \neg q$

7. $\neg p \Rightarrow p \to q$

8. $\neg(p \to q) \Rightarrow p$

9. $p \wedge (p \to q) \to q$.

Logical Equivalences (or) Equivalence Rules

laws	Rules
1. Idempotent laws	$p \wedge p \Leftrightarrow p$, $p \vee p \Leftrightarrow p$
2. Associative laws	$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$ $(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$
3. commutative laws	$p \wedge q \Leftrightarrow q \wedge p$ $p \vee q \Leftrightarrow q \vee p$
4. De morgan's law	$\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$ $\neg(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$
5. Distributive law	$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$ $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$
6. complement laws	$p \wedge \neg p \Leftrightarrow F$ $p \vee \neg p \Leftrightarrow T$
7. Dominance laws	$p \vee T \Leftrightarrow T$ $p \wedge F \Leftrightarrow F$.
8. Identity laws	$p \wedge T \Leftrightarrow p$ $p \vee F \Leftrightarrow p$
9. Absorption laws	$p \vee (p \wedge q) \Leftrightarrow p$ $p \wedge (p \vee q) \Leftrightarrow p$
10. Double negation law	$\neg(\neg p) \Leftrightarrow p$
11. contra +ve law	$p \to q \Leftrightarrow \neg q \to \neg p$

12. Conditional as disjunction $\quad p \to q \Leftrightarrow \neg p \vee q$

13. Biconditional as conditional. $\quad p \leftrightarrow q \Leftrightarrow (p \to q) \wedge (q \to p)$

① Without using truth table show that $Q \vee (P \wedge \neg Q) \vee (\neg P \wedge \neg Q)$ is a tautology.

Soln:

	Reasons
$Q \vee (P \wedge \neg Q) \vee (\neg P \wedge \neg Q)$	
$\Rightarrow Q \vee ((P \vee \neg P) \wedge \neg Q)$	Distributive law.
$\Rightarrow Q \vee (P \vee \neg P) \wedge (Q \vee \neg Q)$	"
$\Rightarrow (Q \vee T) \wedge T$	$\because P \vee \neg P \Rightarrow T$
$\Rightarrow T \wedge T$	$P \vee T = T$
$\Rightarrow T$	

$\therefore$ Given statement is tautology.

② Determine whether $(\neg Q \wedge (P \to Q)) \to \neg P$ is a tautology.

	Reason
$(\neg Q \wedge (P \to Q)) \to \neg P$	
$\Rightarrow (\neg Q \wedge (\neg P \vee Q)) \to \neg P$	Conditional as disjunction.
$\Rightarrow \neg(\neg Q \wedge (\neg P \vee Q)) \vee \neg P$	"
$\Rightarrow Q \vee (P \wedge \neg Q) \vee \neg P$	Demorgan's law
$\Rightarrow ((Q \vee P) \wedge (Q \vee \neg Q)) \vee \neg P$	distributive law.
$\Rightarrow ((Q \vee P) \wedge T) \vee \neg P$	$(P \vee \neg P = T)$
$\Rightarrow (Q \vee P) \vee \neg P$	$(\because P \wedge T \Leftrightarrow P)$
$\Rightarrow Q \vee (P \vee \neg P)$	Associate law.
$\Rightarrow Q \vee T$	$(P \vee \neg P = T)$.
$\Rightarrow T$	$(P \vee T = T)$

$\therefore$ It is tautology.

③ Using truth table show that P is equivalent to (i) $(P \wedge Q) \vee (P \wedge \neg Q)$ (ii) $(P \vee Q) \wedge (P \vee \neg Q)$

Soln: To prove $P \Leftrightarrow (P \wedge Q) \vee (P \wedge \neg Q)$.

P	Q	$P \wedge Q$	$\neg Q$	$P \wedge \neg Q$	$(P \wedge Q) \vee (P \wedge \neg Q)$
T	T	T	F	F	T
T	F	F	T	T	T
F	T	F	F	F	F
F	F	F	T	F	F

$\therefore P \Leftrightarrow (P \wedge Q) \vee (P \wedge \neg Q)$

ii) To prove $P \Leftrightarrow (P \vee Q) \wedge (P \vee \neg Q)$

P	Q	$P \vee Q$	$P \vee \neg Q$	$(P \vee Q) \wedge (P \vee \neg Q)$
T	T	T	T	T
T	F	T	T	T
F	T	T	F	F
F	F	F	T	F

$\therefore$ the result.

prove the following equivalences

1. $\neg(P \vee Q) \Leftrightarrow \neg P \wedge \neg Q$
2. $\neg(P \rightarrow Q) \Leftrightarrow P \wedge \neg Q$

① Without using truth table S.T $\neg(P \vee (\neg P \wedge Q))$ & $\neg P \wedge \neg Q$ are logically equivalent.

Soln

	Reasons
$\neg(P \vee (\neg P \wedge Q))$	
$\Leftrightarrow \neg P \wedge [\neg(\neg P) \wedge Q]$	Demorgan's law
$\Leftrightarrow \neg P \wedge [\neg(\neg P) \vee \neg Q]$	"
$\Leftrightarrow \neg P \wedge (P \vee \neg Q)$	Double negation
$\Leftrightarrow (\neg P \wedge P) \vee (\neg P \wedge \neg Q)$	Distributive law
$\Leftrightarrow F \vee (\neg P \wedge \neg Q)$	$\neg P \wedge P \Leftrightarrow F$
$\Leftrightarrow (\neg P \wedge \neg Q) \vee F$	commutative law
$\Leftrightarrow \neg P \wedge \neg Q$	Identity law.

② $(P \vee Q) \wedge (\neg P \wedge (\neg P \wedge Q)) \Leftrightarrow \neg P \wedge Q$

Soln

	Reason
$(P \vee Q) \wedge (\neg P \wedge (\neg P \wedge Q))$	
$\Leftrightarrow (P \vee Q) \wedge ((\neg P \wedge \neg P) \wedge Q)$	Associative law
$\Leftrightarrow (P \vee Q) \wedge (\neg P \wedge Q)$	Idempotent law
$\Leftrightarrow (P \vee Q) \wedge (Q \wedge \neg P)$	Commutative law
$\Leftrightarrow ((P \vee Q) \wedge Q) \wedge \neg P$	Associative law.
$\Leftrightarrow Q \wedge \neg P$	Absorption law
$\Leftrightarrow \neg P \wedge Q$	Commutative law.

$\therefore$ It is equivalent.

③ $(P \vee Q) \wedge \neg P \Leftrightarrow \neg P \wedge Q$.

Soln

	Reasons
$(P \vee Q) \wedge \neg P$	
$\Leftrightarrow \neg P \wedge (P \vee Q)$	commutative law
$\Leftrightarrow (\neg P \wedge P) \vee (\neg P \wedge Q)$	Distributive law.
$\Leftrightarrow F \vee (\neg P \wedge Q)$	$P \wedge \neg P = F$
$\Leftrightarrow \neg P \wedge Q$	Identity law.

$\therefore$ It is equivalent.

H.W

1. $P \rightarrow (Q \rightarrow R) \Leftrightarrow (P \wedge Q) \rightarrow R$
2. $(P \rightarrow r) \wedge (Q \rightarrow r) \Leftrightarrow (P \vee Q) \rightarrow r$
3. $(\neg P \wedge (\neg Q \wedge R)) \vee (Q \wedge R) \vee (P \wedge R) \Leftrightarrow R$.

The Theory of Inference

Laws of Implication & Equivalence

1. Detachment law : $(P \to q) \wedge P \Rightarrow q$
2. Contrapositive law : $(P \to q) \wedge \neg q \Rightarrow \neg P$
3. Disjunctive addition: $P \Rightarrow P \vee q$, $q \Rightarrow P \vee q$.
4. Conjunctive simplification: $P \wedge q \Rightarrow P$, $P \wedge q \Rightarrow q$
5. Disjunctive simplification: $(P \vee q) \wedge \neg P \Rightarrow q$

   $(P \vee q) \wedge \neg q \Rightarrow P$.
6. Chain Rule : $(P \to q) \wedge (q \to r) \Rightarrow (P \to r)$
7. Conditional Equivalence:

   $(P \to q) \Leftrightarrow \neg q \to \neg P \Leftrightarrow \neg P \vee q$.
8. Biconditional Equivalence:

   $(P \leftrightarrow q) \Leftrightarrow (P \to q) \wedge (q \to P) \Leftrightarrow (P \wedge q) \vee (\neg P \wedge \neg q)$.

A thm consists of a set of premises & a conclusion. A thm is proved by showing that the conclusion is T whenever all the premises are assumed to be T

The Theory of Inference:

Defn: To check the logical validity of the conclusion, from the given set of premises, by making use of equivalences rule & implication rule. The theory associated with such things is called inference theory.

Note: premises is a statement which is assumed to be T

Rules for Inference theory:

Rule P: A given premise may be introduced at any stage in the derivation.

Rule T: A formula S may be introduced in

in a derivation if s is tautologically implied by one or more of the preceeding formula in the derivation.

Rule CP : If we can derive s from R & a set of grnl. premises, then we can derive $R \to S$ from the set of premises alone.

In such a case R is taken as an additional premise. Rule CP is also called the deduction thm.

The analysis of the validity of the formula from the given set of premises by using derivation is called "Theory of inference".

Remark: whenever the assumed premise is used in the derivation, then the method of derivation is called indirect method of derivation.

Problems: ① Demonstrate that R is a valid inference from the premises $P \to Q$, $Q \to R$ & P.

Soln. Given premises are

1) $P \to Q$

2) $Q \to R$ Conclusion is R.

3) P.

Step	Derivation	Rule
1.	$P \to Q$	P
2.	$Q \to R$	P
3.	$P \to R$	T (1 & 2 chain rule)
4.	P.	P
5.	R	T (from 3 & 4) (Detachment law)

② S.T RVS logically follows from the premises $C \vee D$, $(C \vee D) \to \neg H$, $\neg H \to (A \wedge \neg B)$ & $(A \wedge \neg B) \to R \vee S$.

Soln: Given 1) $C \vee D$

2) $(C \vee D) \to \neg H$

3) $\neg H \to (A \wedge \neg B)$

4) $(A \wedge \neg B) \to R \vee S$.

Step	Derivation	Rule
1.	$C \vee D$	P
2.	$(C \vee D) \to \neg H$	P
3.	$\neg H$.	T (from 1 & 2) Detachment law
4.	$\neg H \to (A \wedge \neg B)$	P.
5.	$A \wedge \neg B$.	(from 3 & 4) Detachment law
6.	$(A \wedge \neg B) \to R \vee S$.	P.
7.	$R \vee S$	T, from 5 & 6, Detachment law

3. S.T $R \to S$ can be derived from the premises $P \to (Q \to S)$, $\neg R \vee P$ & Q.

Soln Gnl. 1. $P \to (Q \to S)$:

2. $\neg R \vee P$

3. Q

Steps	Derivation	laws
1.	R	Assumed premises
2.	$\neg R \vee P$	P
3.	$R \to P$	T, $P \to Q \Leftrightarrow \neg P \vee Q$
4.	P	T, (from 1 & 3) (Detachment)
5.	$P \to (Q \to S)$	P
6.	$Q \to S$	T (from 4 & 5) Detachment
7.	Q	P
8.	S	T (from 6, 7)
9.	$R \to S$	Rule CP (from 1 & 8)

Note: Rule CP is generally used if the conclusion is of the form $R \to S$

Ex 1. S.T $\neg P$ follows logically from the premises. $\neg(P \wedge \neg Q)$, $(\neg Q \vee R)$ & $\neg R$.

2. S.T $R \wedge (P \vee Q)$ is a valid conclusion from the premises $P \vee Q$, $Q \to R$, $P \to M$ & $\neg M$.

Consistent: The gnl. set of premises $P_1, P_2 \ldots P_m$ are said to be consistent iff $P_1 \wedge P_2 \wedge \ldots \wedge P_m = T$.

Inconsistent: The gnl. set of premises $P_1, P_2 \ldots P_m$ are said to be inconsistent iff $P_1 \wedge P_2 \ldots \wedge P_m = F$.

4. S.T the foll. premises are inconsistent.

i) If Jack misses many classes due to illness, then he fails in high school.

ii) If Jack fails in high school, then he is uneducated.

iii) If Jack reads a lot of books then he is not uneducated.

iv) Jack misses many classes due to illness & reads a lot of books.

Soln. let us consider the atomic statement

P: Jack misses many classes due to illness.

Q: Jack fails in high school.

R: Jack reads a lot of books

S: Jack is uneducated.

The gnl. premises are $P \to Q$, $Q \to S$, $R \to \neg S$, $P \wedge R$.

step	Derivation	Rule
1.	$P \to Q$	P
2.	$Q \to S$	P
3.	$P \to S$	T (from 1 & 2) chain rule
4.	$\neg S \to \neg P$	T (from 3) $A \to B \Leftrightarrow \neg B \to \neg A$ Conditional equivalence
5.	$R \to \neg S$	P
6.	$R \to \neg P$ $\neg S \to \neg P$	T (from 5 & 4)
7.	$\neg R \vee \neg P$ $\neg R \to \neg P$	T (from 6) conditional.
8.	$\neg(P \wedge R)$	T (from 7) demorgan law $(\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B)$
9.	$P \wedge R$	P
10.	$\neg(P \wedge R) \wedge (P \wedge R)$	T (from 8 & 9) $(\neg P \wedge P \Leftrightarrow F)$ Complement law
11.	F	

2. S.T the foll statements constitute a valid argument.

If there was rain then the travelling was difficult.

If they had umbrella, then travelling was not difficult. They had umbrella. ∴ there was no rain.

Soln P : There was rain.

Q : Travelling was difficult

R : They had umbrella.

∴ The gn/. premises are 1) $P \to Q$

2) $R \to \neg Q$, 3) R.

Conclusion : $\neg P$.

Step	Derivation	Rule
1.	$R \to \neg Q$	P
2.	R	P
3.	$\neg Q$	T (from 1 & 2) detachment
4.	$P \to Q$	P
5.	$\neg P$.	T $(P \to Q \wedge \neg Q \Rightarrow \neg P)$ Contrapositive law

∴ It is valid conclusion.

Indirect method of proof: In this method we assume the negation of the distsed conclusion as one of the gn/. premises & we arrive at a contradiction.

1. Prove by indisect method $\neg Q, P \to Q, P \vee R \Rightarrow R$.

Sln: Include $\neg R$ as a gn/. premises.

Step	Derivation	Rule
1.	$P \vee R$	P
2.	$\neg R$	Rule CP, Assumed Premises
3.	P	T (from 1 & 2) $(P \vee R) \wedge \neg R \Rightarrow P$ disjunctive simplification
4.	$P \to Q$	P
5.	Q	T (from 3 & 4) (Detachment law)
6.	$\neg Q$	P
7.	$Q \wedge \neg Q$	T (from 5 & 6).

which is nothing but false. ∴ by the method of contradiction

$(P \to Q) \wedge (Q \to R) \wedge (P \vee R) \Rightarrow R$

Hw

ST $\neg(P \wedge Q)$ follows from $\neg P \wedge \neg Q$ using indisect method.

logic

Predicate Calculus: The analysis based on atomic statement is called predicate calculus

→ words which represent quantity, all, some, some

Universal Quantifier: The symbolic representation $(\forall x)$ is read as "for all x" is called universal quantifier. It is equivalent to

1) for all x
2) for every x
3) for each x
4) everything x is such that
5) Each thing x is such that.

Example: Let us consider the statement

All men are rich

To write this statement in symbolic form:

let $M(x)$: x is a man

$R(x)$: x is rich.

then $\forall x\,(M(x) \to R(x))$

universally valid statement: If the statement is valid for all the values, then the st/. is called universally valid statement.

Eg: consider the statement

$P(x) = x + 0 = 0 + x = x$,

clearly $(\forall x) \in Z$, the statement $P(x)$

is valid. $\therefore$ $P(x)$ is UVS.

Existential Quantifier: The symbolic representation $(\exists x)$ read as "for some x" is called existential quantifier. It is equivalent to

1) for some x 2) for x such that 3) There exists an x such that 4) There is an such that 5) There is atleast one x such that.

Example: Some men are clever.

$m(x)$: x is a man

$c(x)$: x is clever

$\exists(x)\,(m(x) \rightarrow c(x))$

Existentially valid statement: If the statement is valid for some of the values, then the statement is called EVS.

Eg: Consider the statement $G(x) = x^2 - 1 = 0$.

The statement $G(x)$ is valid only for $|x| \in z$.

$\therefore$ $G(x)$ is EVS.

Rules in Quantifiers:

Let $P(x)$ be any statement.

Rule US: universal specification.

$(\forall x)\,P(x) \Rightarrow P(y)$

Rule UG: Universal Generalisation.

$P(y) \Rightarrow (\forall x)\,P(x)$.

Rule ES: Existential specification.

$\exists(x)\,P(x) \Rightarrow P(y)$.

Rule EG: Existential Generalisation.

$P(y) \Rightarrow \exists(x)\,P(x)$

Formula in Quantifier:

1. $(\exists x)\,(A(x) \vee B(x)) \Leftrightarrow (\exists x)\,A(x) \vee (\exists x)\,B(x)$
2. $(\exists x)\,(A(x) \wedge B(x)) \Leftrightarrow (\exists x)\,A(x) \wedge (\exists x)\,B(x)$
3. $(\forall x)\,(A(x) \vee B(x)) \Leftrightarrow (\forall x)\,A(x) \vee (\forall x)\,B(x)$
4. $(\forall x)\,(A(x) \wedge B(x)) \Leftrightarrow (\forall x)\,A(x) \wedge (\forall x)\,B(x)$.
5. $\neg(\forall x)\,A(x) \Leftrightarrow (\exists x)\,\neg A(x)$.
6. $\neg(\exists x)\,A(x) \Leftrightarrow (\forall x)\,\neg A(x)$
7. $(\forall x)\,(A(x) \rightarrow B(x)) \Leftrightarrow (\exists x)\,A(x) \rightarrow (\forall x)\,B(x)$.
8. $(\exists x)\,(A(x) \rightarrow B(x)) \Leftrightarrow (\forall x)\,A(x) \rightarrow (\exists x)\,B(x)$
9. $(\forall x)\,(A \wedge B(x)) \Leftrightarrow A \vee (\forall x)\,B(x)$
10. $(\exists x)\,(A \wedge B(x)) \Leftrightarrow A \wedge (\exists x)\,B(x)$.
11. $(\forall x)\,A(x) \rightarrow B \Leftrightarrow (\exists x)\,(A(x) \rightarrow B)$
12. $(\exists x)\,A(x) \rightarrow B \Leftrightarrow (\forall x)\,(A(x) \rightarrow B)$
13. $A \rightarrow (\forall x)\,B(x) \Leftrightarrow (\forall x)\,(A \rightarrow B(x))$
14. $A \rightarrow (\exists x)\,B(x) \Leftrightarrow (\exists x)\,(A \rightarrow B(x))$

problems: 1. S.T $(\forall x)(H(x) \to m(x)) \wedge H(y) \Rightarrow m(y)$.

Soln: The premises are $(\forall x)(H(x) \to m(x)), H(y)$

step	Derivation	Rule
1.	$(\forall x)(H(x) \to m(x))$	P.
2.	$H(y) \to m(y)$	US from ①
3.	$H(y)$	P
4.	$m(y)$	T (from ② & ③) $\because (P \to Q) \wedge P \Rightarrow Q$)

2. S.T $(\exists x) m(x)$ follows logically from the premises $(\forall x)(H(x) \to m(x)) \wedge (\exists x) H(x)$.

Soln The premises are $(\forall x)(H(x) \to m(x)), (\exists x) H(x)$

step	Derivation	Rule.
1.	$(\forall x)(H(x) \to M(x))$	P
2.	$H(y) \to m(y)$	US from ①
3.	$(\exists x) H(x)$	P
4.	$H(y)$	ES from ③
5.	$M(y)$	T from ② & ④
6.	$(\exists x) m(x)$	EG from ⑤

3. P.T $(\exists x)(P(x) \wedge Q(x)) \Rightarrow (\exists x) P(x) \wedge (\exists x) Q(x)$

Soln The premises are $(\exists x)(P(x) \wedge Q(x))$

step	Derivation	Rule.
1.	$(\exists x)(P(x) \wedge Q(x))$	P
2.	$P(y) \wedge Q(y)$	ES
3.	$P(y)$	T (from ② $\because P \wedge Q \Rightarrow P$)
4.	$Q(y)$	T (from ② $\because P \wedge Q \Rightarrow Q$)
5.	$(\exists x) P(x)$	EG from ③
6.	$(\exists x) Q(x)$	EG from ④
7.	$(\exists x) P(x) \wedge (\exists x) Q(x)$	T from ⑤ & ⑥

4. S.T $(\forall x)(P(x) \to Q(x)) \wedge (\forall x)(Q(x) \to R(x)) \Rightarrow (\forall x)(P(x) \to R(x))$

Soln: The premises are $(\forall x)(P(x) \to Q(x)), (\forall x) Q(x) \to R(x)$.

Step	Derivation	Rule
1.	$(\forall x)(P(x) \to Q(x))$	P.
2.	$P(y) \to Q(y)$	US (from ①)
3.	$(\forall x)(Q(x) \to R(x))$	P
4.	$Q(y) \to R(y)$	US (from ③)
5.	$P(y) \to R(y)$	T (from 2 & 4)
6.	$(\forall x)(P(x) \to R(x))$	UG (from 5)

5. S.T $(\forall x)(P(x) \vee Q(x)) \Rightarrow (\forall x)P(x) \vee (\exists x)Q(x)$.

Soln let us assume, $\neg[(\forall x)P(x) \vee (\exists x)Q(x)]$ as an additional premise.

Step	Derivation	Rule.
1.	$\neg[(\forall x)P(x) \vee (\exists x)Q(x)]$	CP
2.	$\neg[(\forall x)P(x) \wedge (\exists x)Q(x)]$	T (from 1)
3.	$\neg(\forall x)P(x)$	T (from 2)
4.	$\neg(\exists x)Q(x)$	T (from 2)
5.	$(\exists x)\neg P(x)$	T (from 3)
6.	$(\forall x)\neg Q(x)$	T (from 4)
7.	$\neg P(a)$	ES (from 5)
8.	$\neg Q(a)$	US (from 6)
9.	$\neg P(a) \wedge \neg Q(a)$	T (from 7 & 8)
10.	$\neg(P(a) \vee Q(a))$	T
11.	$(\forall x)(P(x) \vee Q(x))$	P
12.	$P(a) \vee Q(a)$	US (from 11)
13.	$(P(a) \vee Q(a)) \wedge \neg(P(a) \vee Q(a))$	T (from 10 & 12)

Unit-2

Relation and Function

Relation

Let $A = \{1,2,3\}$, $B = \{a,b\}$

Then $A \times B = \{(1,a)\ (2,a)\ (3,a)\ (1,b)\ (2,b)\ (3,b)\}$

A subset of $A \times B$ is

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

R is relation from A to B.

i.e., $1Ra$, $2Rb$, $3Ra$

Definition

A binary relation R from a set A to set B is defined as a subset of $A \times B$.

A relation R form A to B is represented by means of a set of ordered pairs (a,b) where $a \in A$ & $b \in B$

If $(a,b) \in R$, then a is related to b

i.e., aRb

If $(a,b) \notin R$, then a is not related to b

i.e., $a \not R b$.

Scanned by CamScanner

Scanned with CamScanner

Relation on a set

If A is a non empty finite set then the relation A is defined as a subset of $A \times A$.

Ex:

If $A = \{1,2,3,4\}$ and $R = \{(a,b) / a \text{ divides } b\}$ is a relation on A and R.

i.e.,

$$A \times A = \{(1,1)\ (2,1)\ (3,1)\ (4,1)\ (1,2)\ (2,2)\ (3,2)\ (4,2)\ (1,3)\ (2,3)\ (3,3)\ (4,3)\ (1,4)\ (2,4)\ (3,4)\ (4,4)\}$$

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

Note:

1) If set A has 'n' elements then the cartesian product $A \times A$ has n^2 elements. Since relation of A is a subset of $A \times A$ there are 2^{n^2} relations on a set with 'n' elts.

Scanned by CamScanner

Scanned with CamScanner

2) If a set A has m elements and a set B has n elements then there are 2^{mn} relations defined from A to B.

Domain and Range of a Relation:

If R is a relation from A to B, then

Domain of $R = \{a : (a,b) \in R\}$ &

Range of $R = \{b : (a,b) \in R\}$

Inverse Relation

If R is a relation from A to B then its inverse R^{-1} is a relation from B to A and defined by

$$R^{-1} = \{(b,a) : (a,b) \in R\}$$

Ex: $R = \{(1,2)\ (1,3)\ (2,3)\ (2,4)\ (3,4)\}$ is a relation on $A = \{1,2,3,4\}$ then

$R^{-1} = \{(2,1)\ (3,1)\ (3,2)\ (4,2)\ (4,3)\}$

Scanned by CamScanner

Scanned with CamScanner

Identity Relation

A relation R on set A defined by

$$R = \{(x,y) : (x=y)\} = I_A$$

i.e., $I_A = \{(x,x) : x \in A\}$ is called an identity relation on A.

ex:

If $A = \{1,2,3\}$ then $I_A = \{(1,1)(2,2)(3,3)\}$ is the identity relation on A.

Universal Relation:

A relation R on set A is said to be an universal relation when R coincides with $A \times A$.

ex:

If $A = \{x, y, z\}$

Then universal relation on A is

$$R = \{(x,x),(x,y)(x,z)(y,y)(y,x)(y,z)(z,z)(z,x)(z,y)\}$$

Complementary Relations

If R is relation defined on a set A, then its complementary

Scanned by CamScanner

Scanned with CamScanner

relation $\bar{R}$ is defined by
$\bar{R} = \{(a,b) : (a,b) \notin R\}$

ex:

If $R = \{(a,b) : a \text{ divides } b\}$ then
$\bar{R} = \{(a,b) : a \nmid b\}$

Composition of Relations:

If R is a relation from A to B and S is a relation from B to C, the composition of R and S denoted by $S \circ R$ which consists of set of order pairs (a,c) for which $a \in A$, & $c \in A$ such that $(a,b) \in R$ & $(b,c) \in S$ where $b \in B$.

i.e., $S \circ R = \{(a,c) \mid \exists\, b \in B \text{ with } (a,b) \in R \text{ and } (b,c) \in S\}$ which is a relation from A to C.

ex:

$R = \{(1,3)\ (2,1)\ (3,4)\}$ and
$S = \{(1,2)\ (2,3)\ (3,4)\ (4,3)\}$ are relation on $A = \{1,2,3,4\}$

Then $S \circ R = \{(1,4)\ (2,2)\ (3,3)\}$

$\because$ $(1,3) \in R$ & $(3,4) \in S \Rightarrow (1,4) \in S \circ R$
$(2,1) \in R$ & $(1,2) \in S \Rightarrow (2,2) \in S \circ R$
$(3,4) \in R$ & $(4,3) \in S \Rightarrow (3,3) \in S \circ R$

Scanned by CamScanner

Scanned with CamScanner

Powers of a Relation

If R is a relation defined on a set A then its powers are defined recursively by $R^1 = R$ & $R^{n+1} = R^n \circ R$.

i.e, $R^2 = R \circ R$, $R^3 = R^2 \circ R$ etc,

ex: If $R = \{(1,2)(2,3)(3,1)\}$ then

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

$(R^2 = R \circ R$, $1R2, 2R3 \Rightarrow 1R^2 3$ etc.,)

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

$(R^3 = R^2 \circ R$, $1R^2 3, 3R1 \Rightarrow 1R^3 1$

$2R^2 1, 1R2 \Rightarrow 2R^3 2$

$3R^2 2, 2R3 \Rightarrow 3R^3 3$

Types of Relations and their Properties:

i) Reflexive Relation:

A relation R on a set A is said to be reflexive if $(a,a) \in R$, $\forall a \in A$.

Scanned by CamScanner

Scanned with CamScanner

II) Symmetric Relation.

A relation R defined on A is said to be symmetric if

$aRb \Leftrightarrow bRa$

i.e., if $(a,b) \in R \Rightarrow (b,a) \in R$.

Note:

The relation R is said to be symmetric if $R = R^{-1}$

III) Transitive Relation

A relation R is said to be transitive if $(a,b) \in R$ and $(b,c) \in R \Rightarrow (a,c) \in R$. (or) if aRb and bRc then aRc.

i.e., R is transitive if aRb and $bRc \Rightarrow aRc$.

IV) Antisymmetric Relation.

A relation R on A is said to be antisymmetric if aRb and $bRa \Rightarrow a = b$

i.e., R is antisymmetric if $(a,b \in R$ then $(b,a) \notin R$.

Scanned by CamScanner

Scanned with CamScanner

Note:

(i) The total number of reflexive relations defined on a set A with 'n' elements is $2^{n(n-1)}$.

Relation Matrix And the Graph of a Relation.

A relation between finite set can be represented by means of Boolean Alger matrix (or) zero-one matrix.

Relation Matrix:

Let R be a relation from $A = \{a_1, a_2 \cdots a_m\}$ to $B = \{b_1, b_2, b_3 \cdots b_n\}$ Then the relation R can be represented by a $m \times n$ matrix

$M_R = (m_{ij})$

where $m_{ij} = \begin{cases} 1 & \text{if } a_i R b_j \\ 0 & \text{if } a_i \not R b_j \end{cases}$

Scanned by CamScanner

Scanned with CamScanner

ex:

Consider the relation $R = \{(x,y) / x \le y\}$ on $A = \{1,2,3,4\}$

Soln

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

The matrix of Relation R is

$$M_R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Note:

The Relation matrix reflects some of the properties of a relation on a set.

i) If a relation is reflexive, all the diagonal elements must be 1

ii) If a relation is Symmetric, then the relation matrix is Symmetric

iii) If a relation is antisymmetric then its relation matrix is such that if $m_{ij} = 1$ then $m_{ji} = 0$ for $i \neq j$

Scanned by CamScanner

Scanned with CamScanner

Graph of a Relation:

A relation defined on a finite set can be represented by means of a graph.

The elements of a set are represented by means of dots or small circles called nodes (or) vertex.

i) If x_i is related to x_j $[x_i R x_j]$ then there is an arc joining the nodes x_i and x_j with an indication of arrow mark $[\rightarrow]$

ii) If x_i is not related to x_j $[x_i \not R x_j]$, There is no arc joining the nodes x_i and x_j

iii) If x_i is related to itself $[x_i R x_i]$, then there is an edge joining at node x_i and ends at x_i. Such an edge is called a loop. If $[x_i \not R x_i]$ there is no loop at the node x_i.

Scanned by CamScanner

Scanned with CamScanner

Properties:

1) If a relation is reflexive, then there is a loop at every node.

2) If a relation is Irreflexive, then there is no loop at any node.

3) If the relation is symmetric and one node is connected to another, then there must be a return arc from second node to first.

* For antisymmetric relation, no such direct return path should exist.

Irreflexive Relation:

A relation R on a set A is irreflexive if $\forall x \in A$, $(x, x) \notin R$.

ex:

The relation "$<$" (less than) on the set of real nos., is irreflexive.

Note: A relation R on a set 'A' is

i) Reflexive $\Leftrightarrow I_A \subseteq R$ where $I_A = \{(x, x) : x \in A\}$

ii) Symmetric $\Leftrightarrow R^{-1} = R$

iii) Transitive $\Leftrightarrow R \circ R \subseteq R$.

Scanned by CamScanner

Scanned with CamScanner

Operations on Relations:

If R and S are relations on a set then,

$$R \cup S = \{(x,y) \mid \overset{(x,y)\in R}{xRy} \text{ or } \overset{(x,y)\in S}{xSy}\}$$

$$R \cap S = \{(x,y) \mid (x,y) \in R \text{ and } (x,y) \in S\}$$

$$R - S = \{(x,y) \mid (x,y) \in R \ \& \ (x,y) \notin S\}$$

$$\overline{R} = \{(x,y) \mid x \not R y\}$$

Matrices of the relations:

If M_R and M_S are relation matrices of R and S resp., then

$$M_{R \cup S} = M_R \vee M_S$$

$$M_{R \cap S} = M_R \wedge M_S$$

$$M_{\overline{R}} = \overline{M_R}$$

$$M_{R^{-1}} = (M_R)^T$$

$$M_{S \circ R} = (M_R)(M_S)$$

$$M_{R^n} = M_R M_R \cdots M_R$$

Where 'v' and '∧' are defined according to Boolean arithmetic

i.e., $1 \vee 1 = 1$ $\quad$ $1 \wedge 1 = 1$

$1 \vee 0 = 1$ $\quad$ $1 \wedge 0 = 0$

$0 \vee 1 = 1$ $\quad$ $0 \wedge 1 = 0$

$0 \vee 0 = 0$ $\quad$ $0 \wedge 0 = 0$

Scanned by CamScanner

Scanned with CamScanner

Note:

If in the digraph, the edges (x, y) and (y, z) exists then the edge (x, z) must exist for a transitive relation.

Problems:

1) Given $S = \{1, 2, \cdots 10\}$ and Relation R on S, where $R = \{(x, y) : x + y = 10\}$ What are the properties of R?

Soln

Gn, $R = \{(x, y) : x + y = 10\}$

i.e., $R = \{(1,9)\ (2,8)\ (3,7)\ (4,6)\ (5,5)\ (6,4)\ (7,3)\ (8,2)\ (9,1)\}$

Properties:

i) R is neither reflexive nor irreflexive since $(5,5) \in R$ and $(x, x) \notin R$ for other elements of S.

ii) R is symmetric

iii) R is not transitive

[i.e., $(1,9) \in R$ & $(9,1) \in R \Rightarrow (1,1) \notin R$]

Scanned by CamScanner

Scanned with CamScanner

3) Let $X = \{1, 2, 3, 4\}$ and R be the relation on X defined by
$R = \{(2,1)(3,1)(4,1)(3,2)(4,2)(4,3)\}$.
Draw the graph of R and also give its matrix.

<u>Soln</u>

Graph of R:

<u>Matrix of R:</u>

$$M_R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

4) Let $X = \{1, 2, 3, 4\}$ & $R = \{(1,1)(1,4)(4,1)(4,4)(2,2)(2,3)(3,2)(3,3)\}$. Write

Scanned by CamScanner

Scanned with CamScanner

the matrix of R and draw its graph.

Soln

Matrix of R

$$M_R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Graph of R

3) Consider the set $A = \{1, 2, 3\}$. What are the properties satisfied by the following relations?

i) $R_1 = \{(1,1)(2,2)(3,3)(1,2)(2,1)\}$

ii) $R_2 = \{(1,1)(2,2)(1,2)(2,1)\}$

iii) $R_3 = \{(1,1)(2,2)(3,3)(1,2)\}$

iv) $R_4 = \{(1,1)(2,2)(3,3)(1,2)(2,1)(2,3)(3,2)\}$

v) $R_5 = \{(1,1)(1,2)(2,3)\}$

Soln i) R_1 is reflexive, symmetric and transitive.

ii) R_2 is symmetric, transitive but not symmetric

Scanned by CamScanner

Scanned with CamScanner

iii) R_3 is reflexive, transitive but not symmetric.

iv) R_4 is reflexive, symmetric but not transitive

v) R_5 satisfies none of the properties of reflexive, symmetric & transitive.

Connectivity Relation:

Let R be a relation on a set A. The connectivity relation of R is defined by

$R^\infty = \{(x, y) /$ there is a path from x to y (or) y is reachable from x by means of some path of any length$\}$

Some well known Relations and their Properties:

Set	Relation	Properties.
i) R, set of real nos.,	$>$ (greater than) $<$ (less than)	Irreflexive & transitive.

Scanned by CamScanner

Scanned with CamScanner

ii) R, set of real nos,	= (equality)	Reflexive, Symmetric, transitive
iii) X, the collection of subsets of a set.	i) Set inclusion ($\subseteq$)	Reflexive, Symmetric transitive.
	ii) Proper inclusion ($\subset$)	Irreflexive, antisymmetric, transitive.

4) Let $A = \{a, b, c, d, e\}$ and $R = \{(a,a)(a,b)(b,c)(c,e)(c,d)(d,e)\}$ be a relation on A. Compute R^2 and R^∞.

Soln

i) R^2:

$R^2 = R \circ R$

$= \{(a,a)(a,b)(a,c)(b,e)(b,d)(c,e)\}$.

ii) R^∞

$R^\infty = \{(x,y) / y$ is reachable from x by a path of any length$\}$

Scanned by CamScanner

Scanned with CamScanner

$= \{(a,a)\ (a,b)\ (a,c)\ (a,d)\ (a,e)\ (b,c)$
$(b,d)\ (b,e)\ (c,d)\ (c,e)\ (d,e)\}$ //

5) Let R and S be relations on set $A = \{1,2,3\}$ with relation matrices

$$M_R = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \& \quad M_S = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Find the matrices of the following relations: i) $R \cup S$ ii) $R \cap S$ iii) $\overline{R}$ iv) R^{-1} v) $S \circ R$

Soln From the matrix we get $R = \{(1,1)\ (1,3)\ (2,2)\ (2,3)\}$
$S = \{(1,2)\ (1,3)\ (2,1)\ (2,2)\ (3,2)\}$

i) $M_{R \cup S} = M_R \vee M_S$

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

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

ii) $M_{R \cap S} = M_R \wedge M_S$.

Scanned by CamScanner

Scanned with CamScanner

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

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

iii) $M_{\bar{R}} = \overline{M_R}$

$\bar{R}$ is the complement of R that consists of elements of $A \times A$ that are not in R. i.e., $\bar{R} = \{(1,2)\ (2,1)\ (3,1)\ (3,2)\ (3,3)\}$

$$\therefore \overline{M_R} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

iv) $M_{R^{-1}} = (M_R)^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$

v) $M_{S \circ R} = M_R \odot M_S = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \odot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

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

$[R \circ S = \{(1,2)\ (2,1)\ (2,2)\ (1,3)\}]$

Scanned by CamScanner

Scanned with CamScanner

6) If R is the relation on $A = \{1, 2, 3\}$ such that $(a,b) \in R$, if and only if $a+b$ = even, find the relation matrix M_R. Find also the relational matrices R^{-1}, $\bar{R}$ & R^2.

Soln

Gn., $A = \{1,2,3\}$, $(a,b) \in R \Leftrightarrow a+b =$ even

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

$$\therefore M_R = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{array}$$

i) $M_{R^{-1}} = (M_R)^T$

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

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

$$\therefore M_{\bar{R}} = \overline{M_R} = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 \\ 3 & 0 & 1 & 0 \end{array}$$

(which is the same as the matrix obtained from M_R by changing 0's to 1's & 1's to 0's)

Scanned by CamScanner

Scanned with CamScanner

iii) $M_{R^2} = M_R \cdot M_R$

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

$$= \begin{bmatrix} 1\vee 0\vee 1\cancel{\vee 0} & 0\vee 0\vee 0 & 1\vee 0\vee 1 \\ 0\vee 0\vee 0 & 0\vee 1\vee 0 & 0\vee 0\vee 0 \\ 1\vee 0\vee 1 & 0\vee 0\vee 0 & 1\vee 0\vee 1 \end{bmatrix}$$

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

H.W

7. Let $A = \{1,2,3\}$ & $B = \{1,2,3,4\}$ and R and S be the relations from A to B whose matrices are.

$$M_R = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \& \; M_S = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Compute i) $\bar{S}$ ii) $R \cap S$ iii) $R \cup S$

iv) R^{-1}

Soln From the matrices we get Relations

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

Scanned by CamScanner

Scanned with CamScanner

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

i) $\bar{S} = \{(1,1)\ (1,4)\ (2,2)\ (2,3)\ (3,3)\ (3,4)\}$

$$M_{\bar{S}} = \overline{M_S} = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 0 & 0 & 1 \\ 2 & 0 & 1 & 1 & 0 \\ 3 & 0 & 0 & 1 & 1 \end{array}$$

ii) $M_{R \cap S} = M_R \wedge M_S$

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

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

iii) $M_{R \cup S} = M_R \cup M_S = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \vee \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$

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

iv) $M_{R^{-1}} = (M_R)^T = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$

Scanned by CamScanner

Scanned with CamScanner

8) Let R and S be relations on set A with

$$M_R = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad M_S = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Find the matrices a) $R \cup S$ b) $R \cap S$ c) $R \cdot S$ d) $S \cdot R$.

Partition:

A Partition P of a non-empty set A is a collection of non-empty subsets of A. such that,

i) Each element of A belongs to exactly one of the subsets in P.

ii) If $A_1, A_2, \cdots A_n$ are distinct subsets in P then $A = A_1 \cup A_2 \cup \cdots \cup A_n$.

ex:

List all the partitions of $A = \{1, 2, 3\}$

Soln

$P_1 = \{\{1\}, \{2\}, \{3\}\}$ $\quad P_5 = \{1, 2, 3\}$

$P_2 = \{\{1\}, \{2, 3\}\}$

$P_3 = \{\{2, \}, \{1, 3\}\}$

$P_4 = \{\{3\}, \{1, 2\}\}$

Scanned by CamScanner

Scanned with CamScanner

Equivalence Relation:

A relation on a set A is called an equivalence relation if its reflexive, symmetric & transitive.

Problems in Relations: (Have to date this problem in Relation topic)

1) Determine whether the relation R on the set of all integers is reflexive, symmetric, antisymmetric and/or transitive where aRb if and only if

i) $a \neq b$ ii) $ab > 0$, iii) $ab \geq 1$

iv) a is a multiple of b. v) $|a - b| = 1$

Soln

i) $a \neq a$ is not true, Hence R is not reflexive.

$a \neq b \Rightarrow b \neq a$ $\therefore$ R is symmetric

$a \neq b$ & $b \neq c \not\Rightarrow a \neq c$

$\therefore$ R is not transitive.

ii) $a^2 \geq 0$ $\therefore$ R is reflexive.

$ab \geq 0 \Rightarrow ba \geq 0$ $\therefore$ R is symmetric

Consider,

$(2,0)$ & $(0,-3) \in R$ But $(2,-3) \notin R$

as $2(-3) < 0$ $\therefore$ R is not transitive.

Scanned by CamScanner

Scanned with CamScanner

iii) $a^2 \geq 1$

need not be true. Since a may be zero

$\therefore$ R is not reflexive.

$ab \geq 1 \Rightarrow ba \geq 1$ $\therefore$ R is symmetric

$ab \geq 1$ & $bc \geq 1 \Rightarrow \forall a, b, c > 0$ or < 0

if $a, b, c > 0$

least a = least b = least c = 1

$\therefore ac \geq 1$

if $a, b, c < 0$

greatest a = greatest b = greatest c = -1

$\therefore ac \geq 1$. $\therefore$ R is transitive.

iv) a is a multiple of a

$\therefore$ R is reflexive.

If 'a' is multiple of b, b is not a multiple of 'a' in general.

But if 'a' is a multiple of b & b is a multiple of a then a=b

$\therefore$ R is antisymmetric.

'a' is a multiple of b & b is a mple of c then a is a multiple of c

$\therefore$ R is symmetric.

v) $|a-a| \neq 1$ $\therefore$ R is not reflexive

Scanned by CamScanner

Scanned with CamScanner

$|a-b| = 1 \Rightarrow |b-a| = 1$ $\therefore$ R is symmetric

$|a-b| = 1$ $[a-b = 1$ (or) $-1]$

$|b-c| = 1$ $[b-c = 1$ (or) $-1]$

then $|a-c| = 2$ (or) 0

i.e., $|a-c| \neq 1$ $\therefore$ R is not transitive

Problems in Equivalence Relation:

1) P.T the relation "consquence modulo m" defined by $R = \{(x,y) / x-y$ is divisible by $m\}$ is a equivalence relation over the set of the integers show also that if $x_1 \equiv y_1$ & $x_2 \equiv y_2$ then $(x_1 + x_2) \equiv (y_1 + y_2)$

Soln

i) R is reflexive:

For any $x \equiv x \pmod m$

Since $x - x$ is divisible by m

$\therefore (x,x) \in R \Rightarrow$ R is reflexive.

ii) R is symmetric:

Let $(a,b) \in R \Rightarrow a \equiv b \pmod m$

i.e., $a-b$ is divisible by m.

$b-a = -(a-b)$, which is also divisible by m.

Scanned by CamScanner

Scanned with CamScanner

$\therefore b \equiv a \pmod{m} \Rightarrow (b, a) \in R$

$\therefore$ R is symmetric

iii) R is transitive:

Let $(a, b) \in R$ & $(b, c) \in R$

$\therefore a \equiv b \pmod{m}$ & $b \equiv c \pmod{m}$

i.e., $a-b$ is divisible by m & $b-c$ is divisible by m.

Then, $a - c = (a-b) + (b-c)$

$= k_1 m + k_2 m$

$= (k_1 + k_2) m$

$\Rightarrow a-c$ is divisible by $m \Rightarrow a \equiv c \pmod{m}$

i.e., $(a, c) \in R \Rightarrow R$ is transitive

Hence R is an equivalence relation.

$x_1 \equiv y_1 \Rightarrow x_1 - y_1 = am$

$x_2 \equiv y_2 \Rightarrow x_2 - y_2 = bm$

$\therefore x_1 - y_1 + x_2 - y_2 = (a+b)m$

$(x_1 + x_2) - (y_1 + y_2)$ is divisible by m.

$\therefore (x_1 + x_2) \equiv (y_1 + y_2)$.

2) Let $X = \{1, 2, 3, \cdots 7\}$ and $R = \{x, y \mid x-y$ is divisible by $3\}$. S.T R is a equivalence relation & draw the graph

Scanned by CamScanner

Scanned with CamScanner

Soln

i) For any $a \in X$, $a-a$ is divisible by 3. $\therefore (a,a) \in R \ \forall a \in X$

$\therefore$ R is reflexive.

ii) Let $(a,b) \in R$

$a-b$ is divisible by 3

$\Rightarrow b-a$ is also divisible by 3

$(a,b) \in R \Rightarrow (b,a) \in R$

$\therefore$ R is symmetric

iii) Let $(a,b) \in R$ & $(b,c) \in R$

$a-b$ & $b-c$ are divisible by 3

ie., $a-b = 3k$ & $b-c = 3m$

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

$= 3(k+m)$ is divisible by 3

$\Rightarrow (a,c) \in R \Rightarrow$ R is transitive

$\therefore$ R is a equivalence relation.

Now :

$R = \{ (1,4)\ (4,1)\ (2,5)\ (5,2)\ (3,6)\ (6,3)$
$(4,7)\ (7,4)\ (1,7)\ (7,1)\ (1,1)$
$(2,2)\ (3,3)\ (4,4)\ (5,5)\ (6,6)$
$(7,7) \}$

Graphs of the relation:

Scanned by CamScanner

Scanned with CamScanner

Equivalence Classes:

Let R be an equivalence relation on a set A. Let $a \in A$, then the set of all those elements of A which are related to a is called an equivalence class determined by 'a' and it is denoted by $[a]$.

i.e., $[a] = \{ b \in A \,/\, (a,b) \in R \}$

Theorem:

Let 'R' be an equivalence relation on a set A and $[a]$, the equivalence class determined by the element a. Then

i) $a \in [a]$

ii) $a \in [b] \iff [a] = [b]$

iii) The equivalence classes

Scanned by CamScanner

Scanned with CamScanner

determined by two elements are either disjoint (or) identical.

i.e., $[a] \cap [b] = \phi$ (or) $[a] = [b]$

Proof

i) The relation R is reflexive, we have aRa, $\forall a \in A$,

$\therefore a \in [a]$, $\forall a \in A$

ii) Let $a \in [b]$, so aRb — (1)

Now inorder to S.T $[a] = [b]$, we observe that for an arbitrary element $x \in [a]$, we have

$x \in [a] \iff xRa$

$\iff xRa$ & aRb (2) (gn)

(from (1) & (2)) $\iff xRb$ (transitivity)

$\iff x \in [b]$

Converse, $\therefore [a] = [b]$

Thus $a \in [b] \Rightarrow [a] = [b]$

Again, let $[a] = [b]$

Then from (i), $a \in [a]$, and so $a \in [b]$,

Thus $[a] = [b] \Rightarrow a \in [b]$

Hence $a \in [b] \iff [a] = [b]$

Scanned by CamScanner

Scanned with CamScanner

iii) For any two classes $[a]$ and $[b]$ If $[a] \cap [b] = \phi$, then the result follows.

So, let us consider, $[a] \cap [b] \neq \phi$

Let $c \in [a] \cap [b]$

Then $c \in [a]$ and $c \in [b]$

$\Rightarrow cRa$ & cRb

$\Rightarrow aRc$ & aRb (symmetry)

$\Rightarrow aRb$ (transitivity)

$\Rightarrow a \in [b]$

$\Rightarrow [a] = [b]$.

Equivalently,

If $[a] \neq [b]$ then $[a] \cap [b] = \phi$.

i.e., Two distinct equivalent classes are disjoint.

Quotient Set:

If R is an equivalence relation defined on X, then the set of all equivalence classes generated by elements of X is called the Quotient set of X/R

i.e., $X/R = \{[x] \mid x \in X\}$

Scanned by CamScanner

Scanned with CamScanner

Partial Order relation:

A relation R on a set A. is said to be a partial order relation if R is reflexive, antisymmetric and transitive.

Problems

1) Let $S = \{1,2,3,4,5\}$ and $A = S \times S$. The relation R on A is defined by $(a,b) R (a',b')$ iff $ab' = a'b$. S.T R is an equivalence relation and compute A/R.

Soln

i) $(a,b) R (a,b)$ since $ab = ba$

$\therefore$ R is reflexive.

ii) Let $(a,b) R (a',b')$

Then $ab' = a'b$

$\Rightarrow a'b = ab'$

$\therefore (a',b') R (a,b)$

$\therefore$ R is symmetric

iii) Let $(a,b) R (a',b')$ &

Scanned by CamScanner

Scanned with CamScanner

$(a', b') R (a'', b'')$

$\therefore \quad ab' = a'b \quad \& \quad a'b'' = a''b' \quad \left\{ b'' = \frac{a''b'}{a'} \right\}$

Now, $ab'' = a \cdot \frac{a''b'}{a'}$

$= a \cdot \frac{a''}{a'} \cdot \frac{a'b}{a} \quad \left(\because b' = \frac{a'b}{a} \right)$

$\therefore \quad ab'' = a''b$

$\therefore \quad (a, b) R (a'', b'')$

$\therefore$ R is transitive.

$\therefore$ R is an equivalence relation

To find A/R.

$S = \{1, 2, 3, 4, 5\}$

$A = S \times S$

$\therefore A/R = [\{(1,1), (2,2) (3,3) (4,4) (5,5)\},$
$\{(1,2) (2,4)\}, \{(1,3)\} \{(1,4)\} \{(1,5)\}$
$\{(2,1)\} \{(4,2)\}, \{(2,3)\} \{(2,5)\}$
$\{(3,1)\} \{(3,2)\} \{(3,4)\} \{(3,5)\},$
$\{(4,1)\} \{(4,5)\} \{(5,1)\}]$

2) Let $A = \{1, 2, 3, 4\}$ and R be a relation A defined by the matrix.

$$M_R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Compute A/R.

Scanned by CamScanner

Scanned with CamScanner

Soln The relation.

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

The Equivalence class is

$$[a] = \{b \in A : (a,b) \in R\}$$

$$[1] = \{(1,1)\} = \{1\}$$

$$[2] = \{(2,2)\ (2,3)\ (2,4)\} = \{2,3,4\}$$

$$[3] = \{(3,2)\ (3,3)\ (3,4)\} = \text{''}$$

$$[4] = \{(4,2)\ (4,3)\ (4,4)\} = \text{''}$$

$$\therefore A/R = \{\{1,1\}\ \{2,2\}\ \{2,3\}\ \{2,4\},\ \{3,2\}\ \{3,3\}\ \{3,4\}\ \{4,2\},\ \{4,3\}\ \{4,4\}\}$$

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

3) Prove that the intersection of two equivalence relation.

Soln Consider R_1 & R_2 are the equivalence relation.

1) Let $a \in A$.

Since R_1 is an equivalence relation on A, $(a,a) \in R_1\ \forall\ a \in A$.

Scanned by CamScanner

Scanned with CamScanner

III) $(a,a) \in R_2$ since R_2 is an equivalence relation.

$\therefore (a,a) \in R_1 \cap R_2 \quad \forall a \in A$.

$\Rightarrow R_1 \cap R_2$ is reflexive.

ii) Let $(a,b) \in R_1 \cap R_2$

$(a,b) \in R_1$ & $(a,b) \in R_2$

Since R_1 & R_2 are symmetric

$(a,b) \in R_1 \Rightarrow (b,a) \in R_1$

$(a,b) \in R_2 \Rightarrow (b,a) \in R_2$

$\therefore (b,a) \in R_1 \cap R_2$.

$\therefore R_1 \cap R_2$ is symmetric.

iii) Let $(a,b) \in R_1 \cap R_2$ & $(b,c) \in R_1 \cap R_2$

$\Rightarrow (a,b) \in R_1$ & $(b,c) \in R_1 \Rightarrow (a,c) \in R_1$ (transitive)

$\Rightarrow (a,b) \in R_2$ & $(b,c) \in R_2 \Rightarrow (a,c) \in R_2$

$\Rightarrow (a,c) \in R_1 \cap R_2$

$\therefore R_1 \cap R_2$ is transitive.

$\therefore R_1 \cap R_2$ is an equivalence Relation.

H.W

4) Let $A = R \times R$, where R = set of real nos., The relation S on A is defined by $(a,b) S (c,d)$ iff $a^2+b^2 =$

Scanned by CamScanner

Scanned with CamScanner

c^2+d^2. S.T S is an equivalence relation.

Soln

i) $(a,b)\,S\,(a,b)$ since $a^2+b^2=a^2+b^2$

$\therefore$ S is reflexive.

ii) Let if $(a,b)\,S\,(c,d)$, then

$a^2+b^2=c^2+d^2$.

$\therefore\ c^2+d^2=a^2+b^2$

$\Rightarrow\ (c,d)\,S\,(a,b)$

$\therefore$ S is symmetric.

iii) Let $(a,b)\,S\,(c,d)$ & $(c,d)\,S\,(x,y)$

$\therefore\ a^2+b^2=c^2+d^2$ & $c^2+d^2=x^2+y^2$

$\therefore\ a^2+b^2=x^2+y^2 \Rightarrow (a,b)\,S\,(x,y)$

$\therefore$ S is transitive.

$\therefore$ S is an equivalence Relation

5) Let N be set of all natural nos., Prove that the relation R in N defined by $aRb \Leftrightarrow a$ divides b is a partial order relation.

Soln

i) Reflexive:

aRa, $\forall a \in N$, since every

Scanned by CamScanner

Scanned with CamScanner

natural number divides itself..

ii) Antisymmetric.

Let aRb & bRa

a divides b & b divides a.

This is possible only when $a=b$.

iii) Transitive.

Let aRb & bRc. Then a divides b & b divides c.

$\exists$. natural nos m & n $\ni$: $b = ma$ & $c = nb$

$\therefore$ $c = nb \Rightarrow c = n(mn) = (nm)a$

$\therefore$ a divides c and so aRc.

$\Rightarrow$ R is transitive.

$\therefore$ R is a POR.

HW

6) Let R be the set of all real nos, P.T the relation S defined by "$a \leq b$" is Partial Order relation.

Scanned by CamScanner

Scanned with CamScanner

Unit - III

GROUPS

Introduction:

Let A be any set. A mapping $f: A \times A \times \cdots \times A \to A$ (or) $f: A^n \to A$ is called an n-ary operation and 'n' is called the order of the operation.

for $n=1$, $f: A \to A$ is called an Unary operation

for $n=2$, $f: A \times A \to A$ is called Binary operation

Algebraic System:

A set together with a number of operations. (one or more n-ary operations) on the set is called an algebraic system (or) an Algebra.

Example:

Semigroups $(S, *)$, Monoids $(M, *)$, Groups $(G, *)$ are algebraic system with one binary operation. Rings $(R, +, \otimes)$, Integral domains $(I, +, \cdot)$, fields are algebraic system with two binary operations etc.

Scanned by CamScanner

General Properties of Algebraic Systems:

Let $\{S, *, \oplus\}$ be an algebraic system, where $*$ and $\oplus$ are binary operations on S.

1) Closure Property

for any $a, b \in S$, $a * b \in S$

for example: If $a, b \in Z$, $a+b \in Z$ and $a * b \in Z$ where $+$ and $\times$ are the operations of addition and multiplication.

2) Associative Property:

For any $a, b, c \in S$, $(a * b) * c = a * (b * c)$

for example: If $a, b, c \in Z$

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

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

3) Commutative Property:

for any $a, b \in S$, $a * b = b * a$

for example: If $a, b \in Z$,

$a+b = b+a$ &

$a \times b = b \times a$

Scanned by CamScanner

4) Identity element:

There exists a distinguished element $e \in S$, such that for any $a \in S$, $a * e = e * a = a$. The element $e \in S$ is called the identity element of S with respect to the operation $*$.

for example: 0 and 1 are the identity element of Z with respect to the operations of addition and multiplication resp., since for any $a \in Z$, $a + 0 = 0 + a = a$

$a \times 1 = 1 \times a = a$.

5) Inverse Element.

for each $a \in S$, there exists an element $a^{-1} \in S$ such that $a * a^{-1} = a^{-1} * a = e$. The element $a^{-1} \in S$ is called the inverse of under $*$.

for example: for each $a \in Z$, $-a$ is the inverse of a under the operation of addition.

Since $a + (-a) = -a + a = 0$, where '0' is the identity element of Z under addition.

Scanned by CamScanner

b) Distributive Property :

for any $a, b, c, \in S$

$a * (b \oplus c) = (a * b) \oplus (a * c)$

In this case, the operation $*$ is said to be distributive over the operation $\oplus$.

for example : The usual multiplication is distributive over addition.

since $a \times (b+c) = (a \times b) + (a \times c)$

7) Cancellation Property :

for any $a, b, c, \in S$ and $a \neq 0$

$a * b = a * c \Rightarrow b = c$ (left cancellation law)

$b * c = c * a \Rightarrow b = c$ (Right cancellation law)

ex: Cancellation property holds good for any $a, b, c, \in Z$ under addition and multiplication.

8) Idempotent Element :

An element $a \in S$ is called an idempotent element with respect

to the operation. $\forall$ if $a * a = a$.

ex: $0 \in Z$ is an idempotent element under addition since $0+0=0$ and $0, 1 \in Z$ are idempotent element under multiplication or since $0 \times 0 = 0$ $1 \times 1 = 1$.

9) Homomorphism:

If $\{X, \cdot\}$ and $\{Y, *\}$ are two algebraic systems where '$\cdot$' and '$*$' are binary operations then a mapping $g: X \to Y$ is called a homomorphism or simply morphism from $\{X, \cdot\}$ to $\{Y, *\}$ if for any $x_1, x_2 \in X$.

$$g(x_1 \cdot x_2) = g(x_1) * g(x_2).$$

Semi Groups:

Let S be a non-empty set with the binary operation '$*$'. The algebraic system $(S, *)$ is called a semigroup if the operation '$*$' satisfies the following property.

i) Closure Property: For any $a, b \in S$

$a * b \in S$

Scanned by CamScanner

(ii) Associative Property :

For any $a, b, c \in S$,

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

Ex:

Let $(N, +)$ be an algebraic system with binary operation as $+$, where $N = \{1, 2, 3 \ldots\}$ be the set of all natural nos., and $+$ is usual addition. It is clear that $(N, +)$ satisfies closure property and Associative Property.

$\therefore$ The Algebraic system $(N, +)$ is a semi-group.

Monoid.

Let M be a non-empty set with the binary operation "$*$". The algebraic system $(M, *)$ is called a Monoid if the operation $*$ satisfies the following property.

i) Closure Property : For any $a, b \in M$, $a * b \in M$

ii) Associative Property : For any $a, b, c \in M$

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

iii) Identity Element : There exists an element $e \in M \ni : e * a = a * e = a$ $\forall a \in M$.

Scanned by CamScanner

example:

Let $(Z,+)$ be an algebraic system with binary operation '+', where Z is the set of all Integers

i) $x, y \in Z$, $x+y \in Z$

ii) $x, y, z \in Z$, $(x+y)+z = x+(y+z)$

iii) $\exists$ an elt, $0 \in Z$, $\ni : 0+x = x+0 = x$

Identity elt, is "0".

Groups:

A group $(G,*)$ is an algebraic system in which the binary operation "*" on G satisfies the following properties

i) Closure Property:

$\forall\ a, b \in G$, $a * b \in G$

ii) Associative Property:

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

iii) Identity Element:

For every $e \in G$, $\ni$: for any $a \in G$,

$a * e = e * a = a$

iv) Inverse elt,

$\forall\ a \in G$, $\exists$ an inverse elt, of 'a' denoted by $a^{-1} \in G$ $\ni : a * a^{-1} = a^{-1} * a = e$

Scanned by CamScanner

Ex:

Let 'z' be the set of all integers, then the algebraic system $(z,+)$ is a group, where '+' is the usual addition / multiplication

Abelian Group:

A group $(G,*)$ in which the binary operation $*$ is commutative is called an abelian group.

i.e., Let $(G,*)$ be any Group, for any $a,b \in G$ if $a*b = b*a$ then $(G,*)$ is an abelian group.

ex: The group $(z,+)$ is an abelian group.

Order of a Group:

The no., of elements in a group is denoted by $O(G)$ is called the order of the group G.

Order of an Element

Let G be a group and $a \in G$. If for some positive integer 'n', $a^n = e$ then 'n' is called the order of the element 'a' and is denoted by $O(a)$.

Scanned by CamScanner

ex:

i) Let $G = \{1, 0, -1\}$ is not a group with respect to the operation addition

Since $1+1 = 2 \notin G$

(1st property is not satisfied)

+	1	0	-1
1	2	1	0
0	1	0	-1
-1	0	-1	-2

ii) Let $G = \{1, -1, i, -i\}$ with operation multiplication defined by

*	1	-1	i	-i
1	1	-1	i	-i
-1	-1	1	-i	i
i	i	-i	-1	1
-i	-i	i	1	-1

In the table we can observe that the closure property & associative property are verified.

The identity elt is 1

$\because$ $1 \times 1 = 1$, $-1 \times 1 = -1$, $i \times 1 = i$, $-i \times 1 = -$

Scanned by CamScanner

The inverse element of 1 is 1 ($\because 1 \times 1 = 1$)
The inverse element of -1 is -1
The inverse elt., of i is $-i$ ($i \times -i = 1$)
The inverse element of $-i$ is i

$\therefore$ G is a group.

$O(1) = 1 \quad \because (1)^1 = 1$

$O(-1) = 2 \quad (-1)^2 = 1$

$O(i) = 4 \quad (+i)^4 = 1$

$O(-i) = 4 \quad (-i)^4 = 1$

where 1 is the identity element of G
$O(G) = 4$. G is ~~the~~ an abelian group since it satisfies commutative property also.

Properties of a Group:

Property 1 (Uniqueness of Identity)

The identity element in a group is unique.

Proof:

Let e_1 and e_2 be two identity elements of G

Scanned by CamScanner

If e_1 is the identity elt, & $e_2 \in G$, then

$e_1 * e_2 = e_2 * e_1 = e_2$ (Taking e_1 as identity) —— (1)

If e_2 is the identity elt., & $e_1 \in G$, then

$e_1 * e_2 = e_2 * e_1 = e_1$ (Taking e_2 as identity) —— (2)

(from (1) & (2))

$\therefore e_1 = e_2$ $\therefore$ The identity elt, is unique

Property 2

The inverse of every element is unique.

proof:

Let $(G, *)$ be a group, with identity element e. Let b and c be inverses of an element $a \in G$.

$$a * b = b * a = e \quad \text{—— (1)}$$

$$a * c = c * a = e \quad \text{—— (2)}$$

Let $b = b * e$

$= b * (a * c)$ (by (2))

$= (b * a) * c$

$= e * c$ (by (1))

$$\boxed{b = c}$$

$\therefore$ Inverse of every elt, in a group is unique.

Scanned by CamScanner

Property 8:

In any group $(G, *)$, $(a * b)^{-1} = b^{-1} * a^{-1}$ for any $a, b \in G$.

(or)

i.e, The inverse of the product of two elements is equal to the product of their inverses in reverse order.

Proof: Gn, $(G, *)$ be a group.

To prove: $(a*b)^{-1} = b^{-1} * a^{-1}$

We have to show that $(a*b)*(b^{-1}*a^{-1}) = (b^{-1}*a^{-1})*(a*b) = e$

Let $a, b \in G$ & a^{-1}, b^{-1} be their inverses respectively.

$$\therefore \quad a * a^{-1} = e = a^{-1} * a$$

$$\& \quad b * b^{-1} = e = b^{-1} * b$$

Now $(a * b) * (b^{-1} * a^{-1}) = a * [b * (b^{-1} * a^{-1})]$

$$= a * [(b * b^{-1}) * a^{-1}]$$

$$= a * (e * a^{-1}) \quad (b * b^{-1} = e)$$

$$= a * a^{-1}$$

i.e, $(a*b) * (b^{-1}*a^{-1}) = e$ —— (1)

i.e, we can P.T

Scanned by CamScanner

$$(b^{-1} * a^{-1}) * (a * b) = e \quad \text{--- (2)}$$

from (1) & (2), we get

$$(a * b)^{-1} = b^{-1} * a^{-1}$$

i.e., The inverse of $(a * b)$ is $b^{-1} * a^{-1}$

Property 4:

In a group the cancellation property are true for all $a, b, c \in G$

i) $a * b = a * c \Rightarrow b = c$

ii) $b * a = c * a \Rightarrow b = c$.

Proof:

Gn., $(G, *)$ is a group, let e be the identity element of G.

i) Given $(a * b) = a * c$

Let a^{-1} be the inverse of 'a'

Pre multiplying by a^{-1}, we get

$$a^{-1} * (a * b) = a^{-1} * (a * c)$$

$$(a^{-1} * a) * b = (a^{-1} * a) * c \quad \text{(By Ass. Prop)}$$

$$e * b = e * c \quad \text{(\" Inverse Identity)}$$

$$b = c \quad \text{(By Identity)}$$

Scanned by CamScanner

ii) G_n, $b * a = c * a$

post xplying by a^{-1}.

$\Rightarrow (b * a) * a^{-1} = (c * a) * a^{-1}$

$\Rightarrow b * (a * a^{-1}) = c * (a * a^{-1})$

$\Rightarrow b * e = c * e$

$\Rightarrow \boxed{b = c}$

Hence cancellation property are true in a group.

SubGroup:

Let G be a group with binary operation $*$. Let H be a non-empty subset of G. If H is also a group under the same operation $*$. Then H is called a subgroup of G.

Ex:

1) Every group $(G, *)$ is a subgroup of itself.

2) Let G be a group $G = \{1, -1, i, -i\}$ with the operation xplication

The set $\{1, -1\}$ and $\{1\}$ are subgroups of G.

$\cdot$	1	-1
1	1	-1
-1	[illegible]	[illegible]

$\cdot$	i	$-i$
i	-1	1
$-i$	[illegible]	[illegible]

is not a

Scanned by CamScanner

Theorem: 1

The necessary and sufficient condition that a non-empty subset H of a group G be a subgroup is $a \in H, b \in H \Rightarrow a * b^{-1} \in H$.

Proof:

Necessary Condition:

Assume that H is a subgroup of G. Since H itself is a group. We have, $a, b \in H \Rightarrow a * b \in H$ (closure)

Also, $b \in H \Rightarrow b^{-1} \in H$ (inverse)

$\therefore a, b \in H \Rightarrow a, b^{-1} \in H$

$\Rightarrow a * b^{-1} \in H$.

ie, $a, b \in H \Rightarrow a * b^{-1} \in H$

Sufficient Condition:

Let $a * b^{-1} \in H$ — (1) $\forall a, b \in H$

and $H \subseteq G$.

To prove: H is a subgroup of G.

For,

i) Identity: Let $a, a \in H$, $a \in H$

$\Rightarrow a * a^{-1} \in H$ (by (1))

$= 1 \quad e \in H$

Hence the identity 'e' is the elt. of H

Scanned by CamScanner

ii) Inverse:

Let $e \in H$, $a \in H \Rightarrow e * a^{-1} \in H$

$\Rightarrow a^{-1} \in H$

i.e., Every elt. of H has an inverse which is in H.

iii) Closure:

Let $b \in H \Rightarrow b^{-1} \in H$

$\therefore$ for $a, b \in H \Rightarrow a, b^{-1} \in H$

$\Rightarrow a * (b^{-1})^{-1} \in H$

$= a * b \in H$

iv) Associative: Let $a, b, c \in H \subset G$

$\Rightarrow (a * b) * c = a * (b * c)$

Since all the four properties are true.

$\therefore$ H is a subgroup of G.

Theorem: 2

The intersection of two subgroups of a group is also a subgroup of the group. (or)

If H_1, H_2 are two subgroups of G. then $H_1 \cap H_2$ is also a subgroup of G.

Scanned by CamScanner

Proof:

Let H_1 & H_2 be any two subgroups of G.

Then $H_1 \cap H_2 \neq \phi$ because atleast the identity element is common to both H_1 & H_2.

To prove:

$H_1 \cap H_2$ is a subgroup, it's enough to show that for $a, b \in H_1 \cap H_2$

$\Rightarrow a * b^{-1} \in H_1 \cap H_2$.

Let $a, b \in H_1 \cap H_2 \Rightarrow a, b \in H_1$ &

$a, b \in H_2$

$\Rightarrow a * b^{-1} \in H_1$ & $a * b^{-1} \in H_2$

$\Rightarrow a * b^{-1} \in H_1 \cap H_2$

i.e., $a, b \in H_1 \cap H_2 \Rightarrow a * b^{-1} \in H_1 \cap H_2$

$\therefore H_1 \cap H_2$ is a subgroup of G.

Example:

Determine whether $H_1 = \{0, 5, 10\}$ and $H_2 = \{0, 4, 8, 12\}$ are subgroups of Z_{15} under addition

Scanned by CamScanner

Soln

For H_1

$+_{15}$	0	5	10
0	0	5	10
5	5	10	0
10	10	0	5

For H_2

$+_{15}$	0	4	8	12
0	0	4	8	12
4	4	8	12	1
8	8	12	1	5
12	12	1	5	9

Since all the entries in the addition table for H_1 are the elts. of H_1.

$\therefore$ H_1 a subgroup of Z_{15}

$|||^{ly}$ all the entries in the addition table for H_2 are not the elements of H_2.

i.e., H_2 is not closed under addition

$\therefore$ H_2 is not a subgroup of Z_{15}.

Scanned by CamScanner

Cyclic Group:

A group $(G, *)$ is said to be cyclic group if there exist an element $a \in G$ ⇒: every elt, of the group is of the form ~~a^n~~ ~~some integer~~ ⇒, $x = a^n$ (or) $x = na$ where 'n' is some integer. Here the element 'a' is called the generator of G and is written as $G = (a)$ is read as G is cyclic group generated by 'a'.

ex:

Let $G = \{1, -1, i, -i\}$, then $(G, *)$ is a cyclic group generated by i, since $i^2 = -1$, $i^3 = -i$, $i^4 = 1$, $i^1 = i$

i.e, All the elements of G can be expressed as integral powers of the element i.

$\therefore$ G is a cyclic group generated by i.

Since i is the generator of G, $(i)^{-1}$ is also generator of G.

Hence G is a cyclic group & its

Scanned by CamScanner

generators are i & $-i$,

[(1 & -1) are not generator]

Theorem: 3.

Every subgroup of a cyclic group is cyclic.

Proof:

Let $G = (a)$ is a cyclic group generated by 'a' and H is its subgroup.

The elements of H are integral powers of a.

If $a^s \in H$, then the inverse $a^{-s} \in H$

$\therefore$ H contains elements which are positive as well as negative integral powers of a.

let 'm' be the least +ve integer $\ni$ $a^m \in H$, we have to prove that $H = (a^m)$, i.e., H is cyclic & its generator is a^m.

For, let a^t be any arbitrary elt, of H.

Scanned by CamScanner

By division algorithm, $\exists$ integers q & r

$\Rightarrow t = mq + r, \; 0 \le r < m$

Now, $a^t = a^{mq+r}$,

$= a^{mq} \cdot a^r$

$\Rightarrow a^r = a^t * a^{-mq} = a^{t-mq}$ — (1)

Now, $a^m \in H \Rightarrow (a^m)^q \in H$

$= a^{mq} \in H \Rightarrow a^{-mq} \in H$

Also, $a^t, a^{-mq} \in H$

$\Rightarrow a^t * a^{-mq} \in H \Rightarrow a^{t-mq} \in H$

$\Rightarrow a^r \in H$

(by (1))

Since m is the least +ve integers

$\Rightarrow a^m \in H$ and $0 \le r < m$. $\therefore r = 0$

Hence $t = mq$

$a^t = a^{mq} = (a^m)^q \quad \forall \; a^t \in H$

Thus every elt., $a^t \in H$ is of the form $(a^m)^q$.

$\therefore$ H is cyclic & is generated by a^m. i.e., $H = (a^m)$

Scanned by CamScanner

Theorem-4:

Every cyclic group is abelian

Proof:

Let $(G, *)$ be a cyclic group generated by 'a'.

Then $G = \{a^n / n \in Z\}$

Let $x, y, \in G$ be any two elts, then $x = a^m$, $y = a^n$ for some integers m & n.

$$x * y = a^m * a^n$$
$$= a^{m+n} = a^{n+m} = a^n * a^m$$
$$= y * x$$

$\therefore x * y = y * x, \forall x, y \in G.$

$\therefore$ Every cyclic group is abelian.

Group Homomorphism:

Let $(G, *)$ & (G', Δ) be two groups. A mapping $f: G \to G'$ is said to be a group homomorphism

Scanned by CamScanner

if for any $a, b \in G$.

$$f(a * b) = f(a) \Delta f(b)$$

A homomorphism f is said to be group homomorphism if 'f' is 1-1 & onto.

examples:

1. Let G be the set of all integers with the operation '+'. Then $f: G \to G$ defined by $f(x) = 2x$ is a homomorphism

$$f(x+y) = 2(x+y) = 2x + 2y$$
$$= f(x) + f(y)$$

i.e., f is a homomorphism

2. Let G be the set of all integers with addition as the operation. Define; $f: G \to G'$ $\exists$ $f(x) = (x+2)$

To prove: $f(x+y) = f(x) + f(y)$

L.H.S: $f(x+y) = (x+y) + 2$

R.H.S $f(x) + f(y) = x + 2 + y + 2$
$= x + y + 4$

$\therefore f(x+y) \neq f(x) + f(y)$

i.e., f is not homomorphism.

Scanned by CamScanner

Theorem: 5

If 'f' is a homomorphism of a group G into G', then

i) Group homomorphism preserves identities

i.e., $f(e) = e'$, e & e' are identity elt of G & G' resp.,

ii) Group homomorphism preserves inverses

i.e., $f(a^{-1}) = [f(a)]^{-1}$ $\forall$ $a \in G$.

Proof:

i) Let $a \in G$ & f is a homomorphism from G into G'.

then $f(a) \in G'$

$\therefore f(a) * e' = f(a) = f(a * e)$

$f(a) * e' = f(a) * f(e)$

$\Rightarrow \boxed{e' = f(e)}$ (by L. cancellation law)

ii) Let $a \in G$, then $a^{-1} \in G$ and $a * a^{-1} \in G$, $a * a^{-1} = e = a^{-1} * a$

$\therefore e' = f(e)$

$= f(a * a^{-1})$

$= f(a) * f(a^{-1})$

i.e., $f(a) * f(a^{-1}) = e'$

Scanned by CamScanner

i.e., $f(a^{-1})$ is the inverse of $f(a)$ in G

i.e., $[f(a)]^{-1} = f(a^{-1})$.

Theorem : 6

If $f : G \to G'$ is a group homomorphism, H is a subgroup of G, then $f(H)$ is a subgroup of G'.

proof:

Let $f(H) = \{ f(x) \mid x \in G \}$

Then $f(H)$ is a non-empty subset of G'.

Let $a', b' \in f(H)$

we have $f(a) = a'$, $f(b) = b'$

for $a, b \in H$.

$$\therefore a' * (b')^{-1} = f(a) * [f(b)]^{-1}$$

$$= f(a) * f(b^{-1})$$

$$= f(a * b^{-1})$$

But $a * b^{-1} \in H \Rightarrow f(a * b^{-1}) \in f(H)$

$$\Rightarrow a' * (b')^{-1} \in f(H)$$

i.e., for $a', b' \in f(H) \Rightarrow a' * (b')^{-1} \in f(H)$

Scanned by CamScanner

$\therefore f(H)$ is a subgroup of G'.

Kernel of a Homomorphism:

Let $f: G \to G'$ be a group homomorphism, then the set of element of G, which are mapped onto e', the identity element of G', is called the kernal of f & is denoted by $Ker(f)$

i.e., $Ker(f) = \{x \in G : f(x) = e', e' \text{ is the identity of } G'\}$

Theorem: 7

The kernel of a homomorphism f from a group G to G' is a subgroup of G.

Proof:

If $Ker(f) = \{x \in G \mid f(x) = e'\}$

Since $f(e) = e'$ is true always, atleast $e \in Ker(f)$

i.e., $Ker(f)$ is a non empty subset of G.

Scanned by CamScanner

Let $a, b \in \ker f$

with $f(a) = e'$ & $f(b) = e'$

$\therefore f(a * b^{-1}) = f(a) * f(b^{-1})$

$= f(a) * [f(b)]^{-1}$

$= e' * e' = e'$

$\Rightarrow a * b^{-1} \in \ker f$

i.e., $a, b \in \ker f \Rightarrow a * b^{-1} \in \ker f$

$\therefore \ker f$ is a subgroup of G.

Cosets:

Let $(H, *)$ be a subgroup of $(G, *)$ and $a \in G$. Then $aH = \{a * h \mid h \in H\}$ is called left coset of H in G.

Illy $Ha = \{h * a \mid h \in H\}$ is called right coset of H in G.

Ex:

Let $G = \{1, -1, i, -i\}$ and '$\cdot$' be usual multiplication, $H = \{1, -1\}$ is a subgroup of G. Find all the cosets of H.

Soln

$H(1) = \{1 \cdot 1, -1 \cdot 1\} = \{1, -1\} = H$

$H(-1) = \{-1 \cdot 1, -1 \cdot -1\} = \{-1, 1\} = H$

Scanned by CamScanner

$Hi = \{i, -i\}$

$H(-i) = \{-i, i\} = Hi = H(i)$

Note:

1. H and Hi are the 2 distinct right cosets of H in G.
2. $H \cup Hi = G$ i.e., G is partitioned into 2 distinct right cosets of H in G.
3. Any two right (or left) cosets of H in G are either disjoint (or) identical.

Thm: 8 (Lagrange's theorem).

The order of each subgroup of a finite group is a divisor of the order of the group. i.e., $O(H) / O(G)$

Proof:

Let $(G, *)$ be a finite group & $O(G) = n$. Let $(H, *)$ be a subgroup of $(G, *)$ & $O(H) = m$.

Suppose that, $h_1, h_2 \ldots h_m$ are

Scanned by CamScanner

m members of H. For $a \in G$, the right coset Ha of H in G is defined by

$$Ha = \{h_1 * a, h_2 * a, h_3 * a \ldots h_m * a\}$$

Since there should be a 1-1 correspondence betn H and Ha, the members of Ha are distinct.

Hence each right coset of H in G has m distinct members.

W.K.T any right coset of H in G are either disjoint (or) identical.

Since G is a finite, the no. of distinct right cosets of H in G will be finite say 'k'.

The union of these k distinct right cosets of H in G is equal to G.

Hence, if $Ha_1, Ha_2 \ldots Ha_k$ are 'k' distinct right cosets of H in G, then,

$$G = Ha_1 \cup Ha_2 \cup Ha_3 \cup \ldots \cup Ha_k$$

$$O(G) = O(Ha_1) \cup O(Ha_2) \cup O(Ha_3) \cup \ldots \cup O(Ha_k)$$

$$n = m + m + \ldots + m \quad (k \text{ times})$$

$$n = km \Rightarrow k = n/m$$ i.e., m is a divisor of n.

$\therefore$ O(H) is a divisor of O(G). Hence proved.

Scanned by CamScanner

Thm 10: (Fundamental Theorem on Group Homomorphism).

Every homomorphic Image of a Group G is Isomorphic to some quotient group of G.

proof:

Let G' be the homomorphic image of a group G and $f: G \to G'$ be a homomorphism

We have already proved that

$K = \ker f = \{a \in G \mid f(a) = e'\}$ is a subgroup of $(G, *)$, where e' is the identity element of G'. Then K is a normal subgroup of G.

To prove: $G/K \cong G'$

if $a \in G$, then $Ka \in G/K$ & $f(a) \in G'$

Define $\phi: G/K \to G'$ $\ni$: $\phi(Ka) = f(a)$ $\forall a \in G$.

i) Claim: ϕ is well defined.

Since if $Ka = Kb \Rightarrow a * b^{-1} \in K$

$\Rightarrow f(a * b^{-1}) = e'$

$\Rightarrow f(a) * f(b^{-1}) = e'$

$\Rightarrow f(a) * [f(b)]^{-1} = e'$

Scanned by CamScanner

$\Rightarrow f(a) * [f(b)]^{-1} * f(b) = e' * f(b)$

$\Rightarrow f(a) * e' = e' * f(b)$

$\Rightarrow f(a) = f(b)$

$\Rightarrow \phi(Ka) = \phi(Kb)$

$\therefore \phi$ is well defined.

To prove ϕ is a homomorphism

ii) <u>claim</u>: ϕ is one-one.

If $\phi(Ka) = \phi(Kb)$ then $f(a) = f(b)$

$f(a) * f(b^{-1}) = f(b) * f(b^{-1})$

$= f(b * b^{-1})$

$f(a) * f(b^{-1}) = f(e)$

$f(a * b^{-1}) = e'$

$\Rightarrow a * b^{-1} \in K \Rightarrow Ka = Kb.$

$\therefore \phi$ is 1-1

iii) <u>Claim</u>: ϕ is onto.

Let y be any element of G'.

Then $y = f(a)$ for some $a \in G$.

$\therefore$ for every $a \in G$, $Ka \in G/K$

we get $\phi(Ka) = f(a) \forall f(a) = y \in G'$

$\therefore \phi$ is onto.

Scanned by CamScanner

Finally <u>Claim</u>: ϕ is homomorphism.

Let $Ka, Kb \in G/K \Rightarrow \phi(Ka) = f(a)$

$\phi(Kb) = f(b)$

Now $\phi(Ka * Kb) = \phi(K(a * b))$

$= f(a * b) = f(a) * f(b)$

$= \phi(Ka) * \phi(Kb)$

$\therefore \phi$ is a homomorphism.

$\therefore \phi$ is an isomorphism betn G/K & G'

$\therefore G/K \cong G'$.

<u>Rings</u>:

An algebraic system $(R, +, \cdot)$ is called a ring if the binary operations $+$ & $\cdot$ satisfies the following properties.

i) $a+b \in R, \forall a, b \in R$

ii) $(a+b)+c = a+(b+c) \quad \forall a, b, c \in R$

iii) there is an element $0 \in R, \ni: a+0 = 0+a = a \quad \forall a \in R$

iv) $\forall a \in R, \exists -a \in R \ni: a+(-a) = -a+a = 0$

v) $a+b = b+a$

vi) $a \cdot b \in R \quad \forall a, b \in R$

vii) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$

Scanned by CamScanner

(viii) The operation '.' is distributive over +,

ie, $a \cdot (b+c) = (a \cdot b) + (a \cdot c)$

6) Field:

A commutative division ring is called a field.

5'

25

Q.

Scanned by CamScanner

Graphs

Definition:

A Graph $G = (V, E, \phi)$ consists of non-empty set $V = \{v_1, v_2 \cdots\}$ called the set of vertices of the graph G. $E = \{e_1, e_2, \cdots\}$ is the set of edges of the graph G, and ϕ is a mapping from the set of edges E to set of ordered or unordered pairs of elements of V.

web page can be represented by a directed graph

Note: Vertices are the web pages available at the website & a directed edge from page A to page B exist $\Leftrightarrow$ A contains a link to B.

The vertices are represented by points and each edge is represented by a line segment diagramatically.

Directed Graph:

If in a Graph $G = (V, E)$ each edge ~~eee~~ is associated with an ordered pair of vertices, then G is called a directed graph (or) digraph.

Applications: i) In C.S, G.T provides algorithms for doing all sorts of useful things with graphs - Algorithm, Networks flow.

ii) For security purpose (or) to schematize network topologies

Scanned by CamScanner

Undirected Graph:

If each edge is a associated with an unordered pair of vertices then G is called as undirected Graph.

Example:

Undirected graph

Direct graph.

Note:

In a diagram of digraph, each edge $e=(u,v)$ is represented by means of an arrow or direct curve drawn from the initial point is to the terminal point v.

Consider the following graph G:

where $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

$e_1 = \langle v_1, v_2 \rangle$ or $\langle v_2, v_1 \rangle$

$e_2 = \langle v_2, v_3 \rangle$ or $\langle v_3, v_2 \rangle$

$e_3 = \langle v_2, v_4 \rangle$

$e_4 = \langle v_3, v_5 \rangle$

$e_5 = \langle v_3, v_5 \rangle$, $e_6 = \langle v_4, v_4 \rangle$

$e_7 = \langle v_4, v_5 \rangle$

Adjacent:

The vertices v_i and v_j in the undirected graph are said to be adjacent if there is atleast one Edge between v_i and v_j. In this case, the vertices v_i and v_j are called End points of the edge 'e'. The edge 'e' is called also said to be incident with v_i & v_j.

In the graph G, v_1 and v_2, v_2 & v_3, v_3 & v_5, v_2 & v_4 are adjacent, but v_1 and v_3, v_2 and v_5 are not adjacent.

Loop:

If there is an edge from v_i to v_i then that edge is called self loop, or simply "loop".

In the graph G, $e_6 = \langle v_4, v_4 \rangle$ is called self loop.

Scanned by CamScanner

Parallel Edges:

If two edges have the same end points then the edges are called as parallel edges.

In the graph G, $e_4 = \langle V_3, V_5 \rangle$ and $e_5 = \langle V_3, V_5 \rangle$ are called parallel Edges.

Degree of a vertex

The degree of a vertex in an undirected graph is the number of edges incident with point, with the exception that a loop at a vertex contributes twice to the degree of that vertex, the degree of a vertex V is denoted by deg(V) (or) d(V).

In a Graph G, $d(V_1) = 1$, $d(V_2) = 3$, $d(V_3) = 3$, $d(V_4) = 4$, $d(V_5) = 3$, $d(V_6) = 0$.

Isolated Vertex:

A vertex having no edge incident on it is called an isolated vertex. i.e., $d(V) = 0$, then V is called Isolated vertex. In the graph G, $d(V_6) = 0$, V_6 is called isolated vertex.

Pendant Vertex and Pendant Edge:

Any vertex having degree one is called pendant vertex and the corresponding Edge is called Pendant Edge.

In the graph, G, $d(V_1) = 1$, the vertex V_1 is called Pendant Vertex the Edge e_1 is called Pendant Edge.

Incident

If the vertex V_i is an end vertex of some edge e_k, then e_k is said to be incident with V_i.

Adjacent Edges and vertices

Two edges are said to be adjacent if they are incident on a common vertex. In the Graph G, e_1, e_2 and e_3 are adjacent.

Scanned by CamScanner

The vertices v_i and v_j are said to adjacent if v_i, v_j is an edge of the graph.

Simple Graph:

A graph which has neither self loops nor parallel edges is called a simple graph.

MultiGraph:

A Graph is said to be multi-graph if it has parallel Edges.

Pseudo Graphs:

A graph is said to be Pseudo-graph if it has both parallel edges and selfloops.

Multi Graph

Simple graph.

Trivial Graph:

A Graph with one vertex is said to be a trivial graph. $\bullet v_1$

Null Graph

A Graph with isolated vertices is called Null Graph.

i.e., A graph is said to be a null graph if it has no edges between any pair of vertices.

Degree of vertex in a Digraph

a) out degree b) Indegree.

$deg^-(a) = 3$	$deg^+(a) = 1$
$deg^-(b) = 1$	$deg^+(b) = 2$
$deg^-(c) = 2$	$deg^+(c) = 1$
$deg^-(d) = 1$	$deg^+(d) = 3$.
(Indegree)	(Outdegree)

Scanned by CamScanner

Indegree:

The Number of edges with V as their terminal vertex is called indegree of V and is denoted as $\deg^-(V)$

Out degree

In a directed graph, the number of edges with V as their initial vertex is called outdegree of V and is denoted as $\deg^+(V)$

Source:

A vertex in a digraph with zero indegree is called a source.

Sink:

A vertex in a digraph with zero outdegree is called a sink.

Also Note that,

$$\sum \deg^-(V) = \sum \deg^+(V) = \text{No., of edges.}$$

Note:

A loop at a vertex contributes 1 to both the Indegree and the outdegree of this vertex.

Theorem ①: Handshaking Theorem

If $G(V, E)$ is an undirected graph with 'e' edges then

$\sum \deg(V_i) = 2e$.

[i.e., the sum of the degrees of all the vertices of an undirected graph is twice the no., of edges of the graph and hence even.]

proof:

Since every edge is incident with exactly two vertices, every edge contributes 2 to the sum of the degree of the vertices.

$\therefore$ All of the 'e' edges contributes $(2e)$ to the sum of the degree of the vertices.

$$\therefore \sum_i \deg(V_i) = 2e.$$

Ex:-

$d(V_1) = 2$ $d(V_2) = 4$

$d(V_3) = 2$ $d(V_4) = 2$

No., of edges $= 5$

$\sum d(V_i) = 2+4+2+2 = 10 = 2 \times e = 2 \times 5 = 10$

Scanned by CamScanner

Theorem: 2

In an undirected graph, the number of odd degree vertices are even.

proof:

Let $G = (V, E)$ be the undirected graph. Let V_1 and V_2 be the set of all vertices of even degree and set of all vertices of odd degree respectively.

Then by previous thm, we have

$$2e = \sum_{v_i \in V_1} \deg(v_i) + \sum_{v_j \in V_2} \deg(v_j) \quad (1)$$

Since each $\deg(v_i)$ is even,

$\sum_{v_i \in V} \deg v_i$ is even.

As LHS of Eqn (1) is even, we get

$\sum_{v_j \in V} \deg(v_j)$ is even.

Since each $\deg(v_j)$ is odd, the number of terms contained in $\sum_{v_j \in V_2} \deg(v_j)$ is even.

i.e., The no. of vertices of odd degree is even

Ex:

$d(v_1) = 2$
$d(v_2) = 3$
$d(v_3) = 4$
$d(v_4) = 3$
$d(v_5) = 4$

Theorem: 3

The maximum number of edges in a simple graph with 'n' vertices is $\frac{n(n-1)}{2}$

Proof:

We prove this theorem by the method of Mathematical induction 'n'.

for $n=1$, a simple graph with one vertex has no edges

$\therefore$ The result is true for $n=1$.

for $n=2$, a simple graph with 2 vertices may have atmost one edge

$\therefore \frac{2(2-1)}{2} = 1$

The result is true for $n=2$

Assume that the result is true for $n=k$, i.e., a graph with

Scanned by CamScanner

K, vertices has atmost $\frac{K(K-1)}{2}$ Edges. when $n = K+1$, Let G be a graph having 'n' vertices and G' be the graph obtained from G by deleting one vertex say $v \in V(G)$

Since G' has K vertices then by the hypothesis G' has atmost $\frac{K(K-1)}{2}$ edges. Now add the vertex v to G'. (v may be adjacent to all the K vertices of G'.)

$\therefore$ The total number of edges in G are $\frac{K(K-1)}{2} + K = \frac{K^2 - K + 2K}{2}$

$$= \frac{K^2 + K}{2} = \frac{K(K+1)}{2}$$

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

$\therefore$ The result is true for $n = K+1$

Hence the maximum no. of edges in a simple graph with n vertices is $\frac{n(n-1)}{2}$.

Some Special Types of Graphs:

Regular Graph:

If Every vertex of a simple Graph has the same degree, then the graph is called a regular graph

If every vertex in a regular graph has degree n then the graph is called n-regular graph.

2- Regular Graphs:

3- regular Graphs:

Complete Graph:

A Simple Graph, in which there is exactly one edge between each pair of distinct vertices is called a complete graph. The complete graph on 'n' vertices is denoted by K_n

Scanned by CamScanner

Bipartite Graph:

A Simple graph G is said to be bipartite if its vertex set V can be partitioned into two disjoint non-empty sets V_1 and V_2, $V_1 \cup V_2 = V$, such that every edge of G has one end vertex in V_1 and another end vertex in V_2. (So that no edge in G, connects either two vertices in V_1 or two vertices in V_2).

Complete Bipartite Graph:

A Bipartite graph G, with the bipartition V_1 and V_2 is called complete bipartite graph, if every vertex in V_1, is adjacent to every vertex in V_2. Similarly, Every vertex in V_2 is adjacent to every vertex in V_1.

A complete bipartite graph is denoted by $K_{m,n}$, if V_1 contains m vertices and V_2 contains n vertices, assume $m \le n$

$K_{2,3}$

$K_{3,3}$

Graph Representation:

The diagrammatic representation of a graph is very convenient for visual study, if the number of vertices and edges are vertically small. If they are large, the graph can be represented by a matrix. The matrix representation of a graph may be useful in computer programming. There are

Scanned by CamScanner

mainly two ways of representing a graph by a matrix, namely,

i) Adjacency matrix and.

ii) Incidence matrix

Adjacency Matrix of a Simple Graph:

Let $G = (V, E)$ be a simple graph with 'n' vertices, Let the vertices of G be denoted as $V_1, V_2 \cdots V_n$. The adjacency matrix of G is the $n \times n$ matrix $[a_{ij}]$.

where $a_{ij} = \begin{cases} 1, & \text{if there exist an edge betn } v_i \text{ \& } v_j \\ 0, & \text{otherwise.} \end{cases}$

and is denoted by A (or) A(G).

Example:

G_1:

$$A(G_1) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

G_2:

$$A(G_2) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

The Following Basic Properties of an adjacency matrix are obvious:

1) Since a simple graph has no loops, each diagonal entry of A, $a_{ii} = 0$, for $i = 1, 2 \cdots n$.

2) The adjacency matrix of a simple graph is symmetric $a_{ij} = a_{ji}$. Since both of there entries are 1 when v_i and v_j are adjacent and both are zero otherwise. Conversely, given any symmetric zero one (1) matrix. 'A' which contains only 0's on its diagonal there exists a simple graph G whose adjacency matrix is 'A'.

Scanned by CamScanner

3) $\deg(v_i)$ is equal to the number of 1's in the ith row or j^{th} column.

Adjacency Matrix of a Digraph.

When G is a digraph with 'n' vertices $v_1, v_2 \cdots v_n$ the matrix A (or) $A(G) = [a_{ij}]$ where $a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \\ 0, & \end{cases}$

is a directed edge of G / otherwise } is called Adjacency Matrix of digraph.

G :

$$A(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$d^+(v_1) = 1$, $d^+(v_2) = 1$, $d^+(v_3) = 2$, $d^+(v_4) = 2$

$d^-(v_1) = 1$, $d^-(v_2) = 3$, $d^-(v_3) = 2$, $d^-(v_4) = 0$.

Note that, in the adjacency Matrix of a digraph the number of 1's in the ith row of $A(G)$ is the Number of outdegree of the the vertex v_i (-) and the number of 1's in the jth column of $A(G)$ is the number of Indegree of the vertex v_j.

For a digraph, the adjacency matrix $A(G)$ need not be symmetric

Incidence Matrix of a Simple Graph.

When $G = (V, E)$ is an undirected Simple Graph with 'n' vertices $v_1, v_2 \cdots v_n$ and 'm' edges $e_1, e_2 \cdots e_n$ the matrix B (or) $B(G) = [b_{ij}]$ where $b_{ij} = \begin{cases} 1, & \text{when edge } e_i \text{ is Incident with } v_j \\ 0, & \text{otherwise} \end{cases}$

is called the Incidence Matrix of G.

$$B(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Scanned by CamScanner

The following basic properties of an Incidence matrix are obvious.

1) Each column of B contains exactly 2 unit entries

2) A row with all 0 entries corresponds to an isolated vertex.

3) A row with a single unit entry corresponds to a pendant vertex.

4) $\deg(v_i)$ is equal to the number of 1's in the ith row.

G :

$$B(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Incidence Matrix of a Digraph.

When $G = (V, E)$ is a digraph with 'n' vertices $v_1, v_2 \cdots v_n$ and m edges $e_1, e_2 \cdots, e_m$ the $n \times m$ matrix, B (or) $B(G) = [b_{ij}]$ where

$$b_{ij} = \begin{cases} 1, & \text{if } j^{th} \text{ edge having } i^{th} \text{ vertex as initial vertex} \\ -1, & \text{if } j^{th} \text{ edge having } i^{th} \text{ vertex as terminal vertex} \\ 0, & \text{otherwise.} \end{cases}$$

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

Degree Sequence of a Graph:

We represent the degree of th vertices in the graph G as a non decreasing sequence called degr

Scanned by CamScanner

sequence of the graph.

Let us consider the Graphs.

Degree sequence of G_1 : 0, 1, 2, 2, 3

H : 2, 2, 3, 3, 4, 4.

Adjacency Matrix of a Pseudograph:

A Pseudograph (namely an undirected graph with loops and Parallel edges) can also be represented by an adjacency matrix. In this case, a loop at the vertex v_i is represented by a 1 at the $(i,j)^{th}$ position and the $(i,j)^{th}$ entry equals the number of edges that are incident on v_i and v_j.

The adjacency Matrix of a pseudo-graph is also a symmetric matrix.

$$A(G_1) = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & 2 & 0 & 1 \\ 2 & 0 & 3 & 0 \\ 0 & 3 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

[one to one : $f: A \to B$, $A = \{1,2,3,4,5\}$, $B = \{5,6,7,8,9,10\}$ defined by $f(x) = x+4$.

on-to fn : $f: A \to B$, $A = \{-5,-6,-7,-8\}$, $B = \{5,6,7,8\}$ defined by $f(x) = -x$.

Graph Isomorphism:

The simple graphs $G_1 = (V_1, E_1)$ & $G_2 = (V_2, E_2)$ are isomorphic if there is a one to one and onto fn, 'f' from V_1 to V_2 with the property that x and y are adjacent in G_1, if and only if $f(x)$ and $f(y)$ are adjacent in G_2 for all $x, y \in V_1$ such a funct f is called an isomorphism.

Scanned by CamScanner

In case of isomorphism, we write $G_1 \cong G_2$. We observe that isomorphic graphs have

1) The same number of vertices
2) The same number of edges
3) The degree sequence of the graphs are same.

If any of those conditions in not satisfied in two graphs then we can say the graphs are not isomorphic.

Problems:

① Show that the given graphs are isomorphic.

Soln:

The number of vertices in $G_1 = 5$

The number of vertices in $G_2 = 5$

The number of edges in $G_1 = 6$

The number of edges in $G_2 = 6$

Degree sequence of G_1 : 2, 2, 2, 3, 3

Degree sequence of G_2 : 2, 2, 2, 3, 3

The number of vertices and edges are same. The degree sequence are same.

Since in G_1 we've the vertices v_2 and v_4 of degree 3. They must be mapped to u_2 and u_4 in G_2.

Define a fn., $f : G_1 \to G_2$ as.

$f(v_1) = u_1$; $f(v_2) = u_2$; $f(v_3) = u_3$;

$f(v_4) = u_4$; $f(v_5) = u_5$).

$f(e_1) = x_1$; $f(e_2) = x_2$; $f(e_3) = x_4$;

$f(e_4) = x_3$; $f(e_5) = x_6$; $f(e_6) = x_5$.

$\therefore$ f is a onto, one to one between the vertices and edges.

$\therefore$ Given Graphs G_1 and G_2 are isomorphic.

2) Show that the graphs G_1 and G_2 are not isomorphic.

Scanned by CamScanner

Soln:

In the Graph G_1, we have the degree sequence is 2, 2, 2, 3, 3, but the degree sequence of G_2 is 1, 2, 2, 3, 4, so we are not able to find 1-1 correspondence betn the edges of G_1 & G_2.

$\therefore$ G_1 and G_2 are not isomorphic.

Connectivity:

WALK: A walk is an undirected graph $G = (V, E)$ is a finite non-empty sequence $w = v_0 e_1 v_1 e_2 v_2 e_3 \ldots v_{k-1} e_k v_k$ consisting of vertices and edges alternatively, such that e_i connects v_{i-1} and v_i. The vertices v_0 and v_k are called origin and terminus of the walk w. The other vertices are called internal vertices

Open walk:

A walk is open if its initial (starting) and final (Ending) vertices are different

Closed Walk:

A walk is closed if its initial and final vertices are the same.

Trail:

A trail is an open walk in which no edges is repeated, Vertices

Scanned by CamScanner

can be repeated in a trail

Circuit:

A circuit is a closed trail or a circuit is a closed walk in which no edges is repeated vertices may be repeated.

path:

A path is trail with no repeated vertex (or) a path is an open walk in which neither any Edge nor any vertex is repeated. It is also called an Elementary path.

Cycle:

A cycle is a circuit in which no vertex is repeated except the initial vertex, or a cycle is a closed walk in which neither any edge nor any vertex except the initial vertex is repeated.

Example:

1) Open walk: $v_1 e_5 v_3 e_7 v_5 e_8 v_4 e_6 v_3 e_5 v_1 e_1 v_2$.

2) Closed walk: $v_2 e_4 v_4 e_6 v_3 e_6 v_4 e_3 v_1 e_2 v_2$

3) Trail: $v_1 e_1 v_2 e_2 v_1 e_5 v_3 e_7 v_5$

4) Circuit: $v_1 e_1 v_2 e_2 v_1 e_5 v_3 e_7 v_5 e_8 v_4 e_3 v_1$

5) Path: $v_1 e_5 v_3 e_7 v_5 e_8 v_4 e_4 v_2$

6. Cycle: $v_1 e_3 v_4 e_8 v_5 e_7 v_3 e_5 v_1$

Directed Walk:

A directed walk in a digraph G is a finite non-null sequence $w = v_0 e_1 v_1 e_2 v_2 e_3 \cdots v_{k-1} e_k v_k$ consists of vertices and directed edges alternating in which each directed edge e_i has v_{i-1} as initial vertex, v_i as terminal vertex.

Open Directed Walk:

An open directed walk is a directed walk whose initial and final vertices are distinct.

Scanned by CamScanner

Closed Directed Walk:

A closed directed walk is a directed walk whose initial & End vertices are the same.

Directed Trail:

A directed trial is an open directed walk in which no edge is repeated.

Directed Circuit:

A directed circuit is a closed directed walk in which no directed edge is repeated. It is closed directed trail.

Directed Path:

A directed path is a directed trail with no repeated vertex. It is an open directed walk in which neither any directed edge nor any vertex is repeated.

Directed Cycle:

A directed cycle is a directed circuit in which no vertex is repeated except the initial vertex which is also the end vertex of the sequence. It is a closed walk in with neither any directed edge nor any vertex, except the initial vertex is repeated.

Example:

Open directed Walk:

$v_1 e_1 v_2 e_3 v_3 e_4 v_2 e_3 v_3 e_4 v_2 e_6 v_5$

Closed directed Walk:

$v_2 e_3 v_3 e_4 v_2 e_3 v_3 e_5 v_4 e_2 v_2$

Directed Trail:

$v_1 e_1 v_2 e_3 v_3 e_4 v_2 e_6 v_5$

Directed Circuit:

$v_2 e_3 v_3 e_5 v_4 e_2 v_2$

Scanned by CamScanner

Directed Circuit: $v_2 e_3 v_3 e_5 v_4 e_2 v_2$

Directed Path: $v_1 e_1 v_2 e_6 v_5$

Directed Cycle: $v_2 e_3 v_3 e_4 v_2$.

Connected And DisConnected Graph:

An undirected graph is said to be connected if there exists atleast one path between any pair of vertices of the graph.

A graph, which is not connected is a disconnected graph.

Here G is connected but not H since there is no path between v_1 and v_5. In that case we say H is a disconnected graph.

Components of a Graph:

A Disconnected graph consists of two (or) more connected graphs. Each of these graphs is called a component.

In otherwords, the connected subgraphs of a graph G are called components of the graph G.

The number of components is denoted by $\omega(G)$

Here $\omega(G) = 4$.

Remark:

i) For a connected graph $\omega(G) =$

ii) For a null graph having 'n' vertices $\omega(G) = n$.

Scanned by CamScanner

Defn.

A digraph is said to be connected if there exists atleast one directed path between every pair of vertices.

Connected.

Disconnected.

Theorem; 4:

A simple graph with 'n' vertices and 'k' components can have at most $\frac{(n-k)(n-k+1)}{2}$ edges.

proof:

Let $n_1, n_2 \ldots n_k$ be the number of vertices in each of k components of the graph G.

i.e., $\sum_{i=1}^{k} n_i = n$ —(1)

Consider

$$\sum_{i=1}^{k} (n_i - 1) = (n_1 - 1) + (n_2 - 1) + \ldots + (n_k - 1)$$

$$= \sum_{i=1}^{k} n_i - k = n - k$$

(by (1))

Squaring on both sides we get

$$(n_1-1)^2 + (n_2-1)^2 + \ldots + (n_k-1)^2 \le (n-k)^2$$

$$\sum_{i=1}^{k} n_i^2 - 2n + k \le (n-k)^2$$

$$\sum_{i=1}^{k} n_i^2 \le (n-k)^2 + 2n - k \quad \text{—(2)}$$

Since G is simple, the maximum number of edges of G in its components is $\frac{n_i(n_i-1)}{2}$.

$\therefore$ Maximum number of edges of G.

$$= \sum_{i=1}^{k} \frac{n_i(n_i-1)}{2} = \sum_{i=1}^{k} \left(\frac{n_i^2 - n_i}{2}\right)$$

$$\Rightarrow \frac{1}{2}\left[\sum_{i=1}^{k} n_i^2 - \sum_{i=1}^{k} n_i\right]$$

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

(by (1) & (2))

$$\Rightarrow \le \frac{1}{2}\left[(n-k)^2 + (n-k)\right]$$

$$\Rightarrow \le \frac{1}{2}\left[(n-k)[n-k+1]\right]$$

$\therefore$ Maximum number of edges of G $\le \frac{(n-k)(n-k+1)}{2}$

Scanned by CamScanner

Theorem:

If a graph has exactly two vertices of odd degree, there must be a path joining these two vertices.

Proof:

Let G be a graph having exactly two vertices u and v of odd degree.

If G is connected, then there is a u-v path.

Let G be not connected and let G_1 be the component of G containing the vertex G.

We observe that,

i) G_1 is connected subgraph of G.

ii) $d_{G_1}(v) = d_G(v)$ for every vertex v of G_1

iii) $u \in G_1$ and $d(u)$ = odd integer (degree)

iv) If $v \notin G_1$ and u is the only odd degree vertex in G_1, then it is a contradiction the theorem.

W.K.T In a graph the number of odd degree vertices are even.

v) If $v \in G_1$, then $u, v \in G_1$ and G_1 is connected, therefore there is a u-v path in G_1 and hence in G.

From the case (iv) and (v) we get there is a u-v path in G_1.

Subgraph:

A graph $H = (V(H), E(H))$ is said to be subgraph of a graph $G = (V(G), E(G))$ if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

Spanning Subgraph:

A Subgraph H of G is said to be a spanning subgraph of G if $V(H) = V(G)$.

Example:

Scanned by CamScanner

Theorem: 6b

A simple graph with 'n' vertices must be connected if it has more than $\frac{(n-1)(n-2)}{2}$ edges.

proof:

Let G be a simple graph with 'n' vertices and more than $\frac{(n-1)(n-2)}{2}$ edges.

To prove that G is connected.

Suppose if G is not connected then G must have atleast 2 components. Let it be G_1 and G_2. Let V_1 be the vertex set of G_1, with $|V_1| = m$. If V_2 is the vertex set of G_2. Then $|V_2| = n-m$. Then

i) $1 \le m \le n-1$

ii) There is no edge joining a vertex of V_1 and a vertex of V_2

iii) $|V_2| = n-m \geq 1$

Now, $|E(G)| = |E(G_1 \cup G_2)|$

$= |E(G_1)| + |E(G_2)|$

$\le \frac{m(m-1)}{2} + \frac{(n-m)(n-m-1)}{2}$

(By Thm 5)

$\le \frac{1}{2}\left[m^2 - m + n(n-m-1) - m(n-m-1)\right]$

$\le \frac{1}{2}\left[n(n-1) - nm - m(n-m-1) + m^2 - m\right]$

$\le \frac{1}{2}\left[n(n-1) + 2n - 2 - 2n + 2 - nm - nm + m^2 + m + m^2 - m\right]$

[Adding & Subtracting 2n-2]

$\le \frac{1}{2}\left[n(n-1) - 2(n-1) + 2n - 2 - 2nm + 2m^2\right]$

$\le \frac{1}{2}\left[(n-1)(n-2) + 2n(1-m) + 2(m^2-1)\right]$

$\le \frac{1}{2}\left[(n-1)(n-2) - 2n(m-1) + 2(m-1)(m+1)\right]$

$\le \frac{1}{2}\left[(n-1)(n-2) - 2(m-1)(n-m-1)\right]$

$|E(G)| \le \frac{(n-1)(n-2)}{2}$. Since $(m-1)$ for $1 \le m \le n-1$

$(n-m-1) \geq 0$. which is a contradiction as G has

Scanned by CamScanner

more than $\frac{(n-1)(n-2)}{2}$ Edges.

Hence G is a connected graph.

Euler Graph And Hamilton Graph:

Eulerian Path:

A path of a graph G is called a Eulerian path, if it contains each edge of the graph exactly once.

Eulerian Circuit (or) Eulerian cycle:

A circuit of a graph G is called an Eulerian circuit if it includes each edge of G exactly once.

Eulerian Graph: (or) Euler Graph:

Any graph containing an Eulerian circuit is called an Eulerian graph.

ex:

Eulerian Path. E & D

$E \xrightarrow{e_5} D \xrightarrow{e_4} C \xrightarrow{e_2} B \xrightarrow{e_1} A \xrightarrow{e_6} E \xrightarrow{e_7} B \xrightarrow{e_3} D$

For the above graph, we can not find Eulerian circuit (cycle). Therefore, given graph is non-Eulerian.

Example for Euler graph:

Euler Graph

Non - Eulerian

Konigsberg Bridge Problem:

There are two islands A & B formed by a river. They are connected to each other and to river banks C and D by means of 7 bridges. The problem is to start from any one of the 4 land areas A, B, C, D walk across each bridge exactly once and

Scanned by CamScanner

returning to the starting point.

This problem is a famous Konigsberg bridge problem. When the situation is represented by a graph with vertices representing the land areas and the edges the bridges., we get a multigraph as shown below.

Since, the above graph has more than two vertices of odd degree, we cannot have an Euler trail. ∴ it is not possible to travel all the bridges exactly once from start to end of any town.

[In a town of Konigsberg, Prussia was divided into 4 sections by the branches of the pregor river]

This problem is the same as that of drawing the graph without lifting the pen from the paper and without retracing any line.

In otherwords, the problem is to find whether there is an Eulerian circuit (a simple circuit containing every edge) in a graph. But a connected graph has an Eulerian circuit if and only if each of its vertices is of even degree. In the present case, all the vertices are of odd degree. Hence, Konigsberg bridge problem has no solution.

Theorem : 9.

A connected graph is Euler graph (contains Eulerian circuit) if and only if each of its vertices is of even degree.

Scanned by CamScanner

proof:

Let G be any graph having an Eulerian circuit and let c be an Eulerian circuit of G with origin & terminus vertex as u. Each time a vertex v occurs as an internal vertex of c, then two of the edges incident with v are accounted for degree. We get, for internal vertex $v \in V(G)$

$$d(v) = 2 \times \left\{ \begin{array}{l} \text{no., of times } v \text{ occur inside} \\ \text{the Euler circuit } c. \end{array} \right.$$

$$= \text{Even degree}.$$

and since an Euler circuit c contains every edge of G and c starts and ends at u.

$$\therefore d(u) = 2 + 2 \times \left\{ \begin{array}{l} \text{no., of times } u \text{ occur} \\ \text{inside } c. \end{array} \right.$$

$$= \text{Even degree}$$

$\therefore$ G has all the vertices of even degree.

Conversely,

Assume each of its vertices has an even degree.

Claim:

G has an Eulerian circuit.

Suppose not, i.e., Assume G be connected graph which is not having an Euler circuit, with all vertex of even degree and less number of edges.

i.e., Any graph having less number of edges than G then it has an Eulerian circuit. Since each vertex of u has degree atleast two, therefore G contains closed path. Let c be a closed path of maximum possible length in G. If c itself has all the edges of G, then c itself an Eulerian circuit in G.

By assumption, c is not an Euler circuit of G and $G - E(c)$ has some component G' with $|E(G')| > 0$, c has less number of edges than G, therefore 'c' itself

is an Eulerian, and C has all the vertices of even degree thus the connected graph G' also has all the vertices of even degree.

Since $|E(G')| < |E(G)|$,

$\therefore$ G' has an Euler circuit C'.

Because G is connected, there is a vertex v in both C and C'. Now join C and C' and transverse all the edges of C and C' with common vertex v. we get CC' is a closed path in G and $E(CC') > E(C)$, which is not possible for the choices of C.

$\therefore$ G has an Eulerian circuit

$\therefore$ G is a Euler Graph.

1) Check the given graph is Euler graph or not.

Soln

$d(v_1) = 2$, $d(v_2) = 4$; $d(v_3) = 2$

$d(v_4) = 2$ $d(v_5) = 4$.

Since, all the vertices is of even degree, by the above theorem the given graph is Euler graph.

2) Check whether the given graph is Euler (or) not.

Hamilton path.

A path of a graph G is called a Hamilton path, if it includes each vertex of G exactly one.

Scanned by CamScanner

Hamilton cycle:

A cycle of a graph G is called a Hamilton cycle G, if it includes each vertex of G exactly one, except the starting and ending vertices.

Hamiltonian Graph:

Any graph containing a Hamiton cycle is called a Hamiltonian graph.

example:

Hamilton cycle:

$v_4 - v_3 - v_2 - v_1 - v_5 - v_4$

P_1: $v_1 e_1 v_2 e_5 v_5 e_6 v_4 e_4 v_3$ - Hamilton path.

Hamilton path: $v_1 - v_2 - v_3 - v_4 - v_5$

P_1: $v_1 e_1 \ldots$ Hamilton path.

P_2: $v_5 e_5 v_2 e_2 v_3 e_4 v_4$ is a path but not Hamilton path.

C_1: $v_4 e_4 v_3 e_2 v_2 e_1 v_1 e_7 v_5 e_6 v_4$ is a Hamilton cycle.

C_2: $v_5 e_5 v_2 e_2 v_3 e_4 v_4 e_6 v_5$ is a cycle but not Hamilton cycle

Travelling Salesman Problem:

A salesman starts from his home and plans a trip to visit $n-1$ other places and return back to home. The natural objective is to minimize the total travelling time. If we assign each edge of K_n of a weights equal to the travelling time between the corresponding places then we seek a Hamilton cycle with minimum total weights. This problem is known as Travelling salesman problem.

Scanned by CamScanner

Nearer Neighbour Algorithm:

Let $V_1, V_2 \ldots V_n$ be the vertices of K_n and w_{ij} denote the weight of edge $\langle V_i, V_j \rangle$. from an arbitrary starting vertex, move to a vertex with the least cost incident edge, and continue always visiting an unvisited neighbours of the current vertex until we visit all the vertices and return back to the starting vertex. Consider the weighted graph K_5. Let V_1 be the starting vertex.

i) from V_1 go to V_2 (Since $w(V_1, V_2)$ is minimum)

ii) from V_2 go to V_4

iii) from V_4 go to V_5

iv) from V_5 go to V_3 (V_2 is visited vertex, V_1 is starting vertex)

v) from V_3 go to V_1

The travelling route is $V_1V_2V_4V_5V_3V_1$

$2+2+4+6+3 = 16$

Problem:

1) Give an Example of a graph which is

i) Eulerian but not Hamiltonian

ii) Hamiltonian but not Eulerian.

iii) Both Eulerian and Hamitonian

iv) Non Eulearian and non Hamiltonian

Soln:

1) Eulerian but not Hamiltonian.

G_1 contains the Eulerian cycle. A-B-C-D-B-E-A

$\therefore G_1$ is Euler graph.

We cannot find Hamiltonian cycle. As the vertex B is repeated twice.

2) Hamiltonian but not Eulerian.

Since G_2 contains the Hamiltonian cycle namely A-B-C-D-E-A. G_2 is Hamiltonian graph.

G_2:

Since the degree of the vertex A is 3. deg(A) is not an even no., $\therefore$ is not an Euler graph.

Scanned by CamScanner

3) Both Eulerian and Hamiltonian.

G_3 :

G_3 : A – B – C – D – A

Since the cycle contains all the edges G_3 is Eulerian. Since the cycle contain all the vertices exactly once, G_2 is Hamiltonian.

$\therefore$ G_3 is both Eulerian & Hamiltonian.

4) Non-Eulerian & non-Hamiltonian :

G_4 :

$\deg(B) = \deg(D) = 3$

Since degree of B and D are not an even number G_4 is not a Euler graph. As no cycle passes through each of the vertices exactly once, the given graph G_4 is not Hamiltonian.

$\therefore$ G_4 is neither Euler graph nor Hamiltonian graph.

Scanned by CamScanner

Unit-5

❑ BOOLEAN ALGEBRA ❑

A complemented distributive Lattice is called Boolean Algebra.

i.e., A Boolean algebra is distributive Lattice with '0' element and '1' element in which every element has a complement.

i.e., A Boolean algebra is a non empty set with 2 binary operations $\wedge$ and $\vee$ and is satisfied by the following conditions.

$\forall\ a, b, c \in L$

1. L_1 :	$a \wedge a = a$	$a \vee a = a$
2. L_2 :	$a \wedge b = b \wedge a$	$a \vee b = b \vee a$
3. L_3 :	$a \wedge (b \wedge c) = (a \wedge b) \wedge c$	$a \vee (b \vee c) = (a \vee b) \vee c$
4. L_4 :	$a \wedge (a \vee b) = a$	$a \vee (a \wedge b) = a$

5. $D_1 : a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

6. $D_2 : a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

7. There exist elements '0' and '1' such that

$$a \wedge 0 = 0,\ a \vee 0 = a,\ \ a \wedge 1 = a \text{ and } a \vee 1 = 1 \quad \forall\ a$$

8. $\forall\ a \in L$, there exist corresponding element a' in L such that $a \wedge a' = 0$ and $a \vee a' = 1$.

Note: Boolean Algebra is denoted by $(B, \wedge, \vee, 0, 1)$

Example :

$(\rho(A), \cup, \cap)$ is a Boolean Algebra, where A is any finite set.

Scanned with CamScanner

Here, '0' element is ϕ

'1' element is given set A

Complement of $a = a' = A - a$

In a Boolean Algebra

$$(0)' = 1$$

and $$(1)' = 0$$

Note :

(1) In Lattice we have used the notations $\vee$ for sum and $\wedge$ for product. Boolean Algebra is a special case of Lattice and we use '+' for Boolean sum and '·' for Boolean Product.

(2) Therefore, here after we use the notations + and · instead of $\vee$ and $\wedge$ respectively.

Boolean Sum is defined as

$1 + 1 = 1 \quad 1 + 0 = 1 \quad 0 + 1 = 1 \quad 0 + 0 = 0$

Boolean Product is defined as

$1 \cdot 1 = 1 \quad 1 \cdot 0 = 0 \quad 0 \cdot 1 = 0 \quad 0 \cdot 0 = 0$

Example : 1

Prove that $a + ab = a$

Solution :

$$\begin{aligned} \text{LHS} &= a + ab \\ &= a(1 + b) && \text{(Distributive law)} \\ &= a(1) && (\because 1 + a = 1) \\ \boxed{a + ab = a} & && (a \cdot 1 = a) \end{aligned}$$

Example : 2

Prove that $a + \bar{a}b = a + b$

Solution :

$$\begin{aligned} \text{LHS} &= a + \bar{a}b \\ &= a + ab + \bar{a}b && (\because a = a + ab) \\ &= a + b(a + \bar{a}) && \text{(Distributive law)} \\ &= a + b(1) && (\because a + \bar{a} = 1) \\ \boxed{a + \bar{a}b = a + b} & && (\because a \cdot 1 = a) \end{aligned}$$

Example : 3

Prove that $(a + b)(a + c) = a + bc$

Solution :

$$\begin{aligned} \text{LHS} &= (a + b)(a + c) \\ &= aa + ac + ab + bc && \text{(Distributive law)} \\ &= a + ac + ab + bc && (\because a \cdot a = a) \\ &= a(1 + c) + ab + bc && \text{(Distributive law)} \\ &= a + ab + bc && (\because 1 + a = 1) \\ & && (\because a + ab = a) \end{aligned}$$

$$\boxed{(a + b)(a + c) = a + bc}$$

Example : 4

In any Boolean Algebra, show that $a = b$ iff $a\bar{b} + \bar{a}b = 0$

Solution :

Let (B, ·, +, 0, 1) be any Boolean Algebra.

Scanned with CamScanner

Let $a, b \in B$ and $a = b$ ... (1)

Claim : $a\bar{b} + \bar{a}b = 0$

Now, $a\bar{b} + \bar{a}b = a \cdot \bar{b} + \bar{a} \cdot b$

$= a \cdot \bar{a} + \bar{a} \cdot a$ (Using (1))

$= 0 + 0$ $(\because a \cdot a' = 0)$

$= 0$

$\therefore$ $a\bar{b} + \bar{a}b = 0$

Conversely, Assume

$a\bar{b} + \bar{a}b = 0$

$\Rightarrow$ $a + a\bar{b} + \bar{a}b = a$ (Left Cancelation law)

$\Rightarrow$ $a + a\bar{b} = a$ (Absorption law)

$\Rightarrow$ $(a + \bar{a}) \cdot (a + b) = a$ (Distributive law)

$\Rightarrow$ $1 \cdot (a + b) = a$ $(a + \bar{a} = 1)$

$\Rightarrow$ $a + b = a$ $(a \cdot 1 = a)$ ... (A)

Consider, $a\bar{b} + \bar{a}b = 0$

$\Rightarrow$ $a\bar{b} + \bar{a}b + b = b$ (Right Cancelation law)

$\Rightarrow$ $a\bar{b} + b = b$ (Absorption law)

$\Rightarrow$ $(a + b) \cdot (b + \bar{b}) = b$ (Distributive law)

$\Rightarrow$ $(a + b) \cdot 1 = b$ $(b + \bar{b} = 1)$

$\Rightarrow$ $a + b = b$ $(b \cdot 1 = b)$ ... (B)

From (A) and (B)

$a = a + b = b$

$\Rightarrow$ $a = b$

Example : 5

In any Boolean Algebra, show that

$$(a \,.\, \bar{b}) + (b \,.\, \bar{a}) = (a + b) \,.\, (\bar{a} + \bar{b})$$

Solution :

Let $(B, \cdot, +, 0, 1)$ be any Boolean Algebra.

Let $a, b \in B$

Now $(a \cdot \bar{b}) + (b \cdot \bar{a}) = (a + (b \cdot \bar{a})) \cdot (\bar{b} + (b \cdot \bar{a}))$ (Distributive rule)

$= (a + b) \cdot (a + \bar{a}) \cdot (\bar{b} + b) \cdot (\bar{b} + \bar{a})$ (Distributive rule)

$= (a + b) \cdot (1) \cdot (1) \cdot (\bar{a} + \bar{b})$

$= (a + b) \cdot (\bar{a} + \bar{b})$

Example : 6

Evaluate the expression for $\bar{a} = 0$, $a = 1$, $b = 1$, $c = 1$ and $d = 1$. $X = \bar{a} \,.\, b + c\,(\overline{a \cdot d})$.

Solution :

$X = \bar{a}.b + c\,(\overline{a \cdot d})$

$= 0 \cdot 1 + 1\,(\overline{1 \cdot 1})$

Scanned with CamScanner

$= 0 \cdot 1 + 1(\overline{1})$

$= 0 \cdot 1 + 1\,(0)$

$= 0 + 0 = 0$

Example : 7

Evaluate the expression for $X = a\,[\overline{(b+c)+\overline{d}}]$ for $a = 0$, $b = 0$, $c = 1$ and $d = 1$.

Solution :

$X = a\,[\overline{(b+c)+\overline{d}}]$

$= 0\,\overline{(0+1)+\overline{1}}$

$= 0\,[\overline{1+0}]$

$= 0\,[0+1]$

$= 0\,[1]$

$= 0$

Example : 8

Reduce the expression $a \,.\, \overline{a}b$.

Solution :

$a \,.\, \overline{a}b = 0 \cdot b$ $\quad [\because a\overline{a} = 0]$

$= 0$ $\quad [\because a \cdot 0 = 0]$

Example : 9

Reduce the expression $a\,(a+c) = aa + ac$.

Solution :

$a\,(a+c) = aa + ac$

$= a + ac$ $\quad [\because a \cdot a = a]$

$= a\,(1+c)$ $\quad$ (Distributive law)

$= a$ $\quad [a + 1 = 1]$

Example : 10

Reduce the expression $ab + abc + ab\overline{c} + \overline{a}bc$.

Solution :

$ab + abc + ab\overline{c} + \overline{a}bc$

$= ab\,(1+c) + ab\overline{c} + \overline{a}bc$

$= ab + ab\overline{c} + \overline{a}bc$ $\quad [\because a + 1 = 1]$

$= ab\,(1+\overline{c}) + \overline{a}bc$

$= ab + \overline{a}bc$ $\quad [\because a + 1 = 1]$

$= b\,(a + \overline{a}c)$

$= b\,(a+c)$ $\quad [\because a + \overline{a}b = a + b]$

Example : 11

Reduce the expression $xy + \overline{xz} + x\overline{y}z\,(xy+z)$.

Solution :

$xy + \overline{xz} + x\overline{y}z\,(xy+z)$

$= xy + \overline{xz} + xx\overline{y}yz + x\overline{y}zz$ $\quad$ (Distributive law)

$= xy + \overline{xz} + x\overline{y}zz$ $\quad (\because a \cdot \overline{a} = 0)$

$= xy + \overline{xz} + x\overline{y}z$ $\quad (\because a \cdot a = a)$

Scanned with CamScanner

$= xy + \overline{x} + \overline{z} + x\overline{y}z$ $\quad (\overline{ab} = \overline{a} + \overline{b})$

$= \overline{x} + x\overline{y}z + y + \overline{z}$ $\quad$ (Commutative law)

$= \overline{x} + \overline{y}z + y + \overline{z}$ $\quad (a + \overline{a}b = a + b)$

Here $b = \overline{y}z$

$= \overline{x} + y + \overline{z} + \overline{y}z$ $\quad$ (Commutative law)

$= \overline{x} + y + \overline{z} + \overline{y}$ $\quad (a + \overline{a}b = a + b)$

$= \overline{x} + \overline{z} + 1$ $\quad (a + \overline{a} = 1)$

$= 1$ $\quad (a + 1 = 1)$

Example : 12

Reduce the expression $\overline{a}b\overline{c}d + \overline{a}bcd + abd$.

Solution :

$\overline{a}b\overline{c}d + \overline{a}bcd + abd$

$= \overline{a}bd(\overline{c} + c) + abd$ $\quad$ (Distributive law)

$= \overline{a}bd + abd$ $\quad (\because a + \overline{a} = 1)$

$= bd(\overline{a} + a)$ $\quad$ (Distributive law)

$= bd$ $\quad (a + \overline{a} = 1)$

Example : 13

Simplify the expression $z(y + z)(x + y + z)$

Solution :

$z(y + z)(x + y + z)$

$= (zy + zz)(x + y + z)$

$= (zy + z)(x + y + z)$

$= z(y + 1)(x + y + z)$

$= z(x + y + z)$

$= zx + zy + zz$

$= zx + zy + z$

$= z(x + y + 1)$

$= z$

Example : 14

Simplify the expression $(x + \overline{y} + \overline{z})(x + \overline{y} + z)$.

Solution :

$(x + \overline{y} + \overline{z})(x + \overline{y} + z)$

$= xx + x\overline{y} + xz + \overline{y}x + \overline{y}\,\overline{y} + \overline{y}z + \overline{z}x + \overline{z}\,\overline{y} + \overline{z}z$

$= x + x\overline{y} + xz + \overline{y}x + \overline{y} + \overline{y}z + \overline{z}x + \overline{z}\,\overline{y} + \overline{z}z$

$= x(1 + \overline{y} + z + \overline{y} + \overline{z}) + \overline{y}(1 + z + \overline{z}) + 0$

$= x \cdot 1 + \overline{y} \cdot 1$

$= x + \overline{y}$

Example : 15

Simplify the expression $(a \cdot b) + (b + c)$.

Solution :

$Y = (a \cdot b) + (b + c)$

$= b(a + 1) + c$

$= b \cdot 1 + c$

$= b + c$

Scanned with CamScanner

Example : 16

Prove that, in any Boolean Algebra

$$a\bar{b} + b\bar{c} + c\bar{a} = \bar{a}b + \bar{b}c + \bar{c}a$$

Solution :

Now $a\bar{b} + b\bar{c} + c\bar{a} = a\bar{b} \cdot 1 + b\bar{c} \cdot 1 + c\bar{a} \cdot 1$

$= a\bar{b}(c + \bar{c}) + b\bar{c}(a + \bar{a}) + c\bar{a}(b + \bar{b})$

$= a\bar{b}c + a\bar{b}\bar{c} + b\bar{c}a + b\bar{c}\bar{a} + c\bar{a}b + c\bar{a}\bar{b}$

$(\because a + \bar{a} = 1)$

$= a\bar{b}c + \bar{a}\bar{b}c + ba\bar{c} + \bar{b}ac + c\bar{a}b + \bar{c}\bar{a}b$

(Distributive rule)

$= (a + \bar{a})\bar{b}c + (b + \bar{b})a\bar{c} + (c + \bar{c})\bar{a}b$

$= \bar{b}c(a + \bar{a}) + \bar{c}a(b + \bar{b}) + \bar{a}b(c + \bar{c})$

$= \bar{b}c \cdot 1 + \bar{c}a \cdot 1 + \bar{a}b \cdot 1 \quad (\because a + \bar{a} = 1)$

$= \bar{b}c + \bar{c}a + \bar{a}b$

$= \bar{a}b + \bar{b}c + \bar{c}a$

Example : 17

Show that in any Boolean algebra

$$(a + b)(a' + c) = ac + a'b + bc$$

Solution :

Let (B, ·, +, 0, 1) be any Boolean Algebra.

$$\text{LHS} = (a + b) \cdot (a' + c)$$

$$= (a + b) \cdot a' + (a + b) \cdot c$$

(Distributive rule)

$= a \cdot a' + b \cdot a' + a \cdot c + b \cdot c$

(Distributive rule)

$= 0 + a'b + ac + bc \quad (\because a \cdot a' = 0)$

$= ac + a'b + bc$

$= \text{RHS}$

Example : 18

Prove the following Boolean identities:

1. $a + (a' \cdot b) = a + b$
2. $a \cdot (a' + b) = a \cdot b$
3. $(a \cdot b) + (a \cdot b') = a$

Solution :

(1) $\text{LHS} = a + (a' \cdot b)$

$= (a + a') \cdot (a + b)$ (Distributive rule)

$= 1 \cdot (a + b) \quad (\because a + a' = 1)$

$= a + b \quad (a \cdot 1 = a)$

$= \text{RHS}$

(2) $a \cdot (a' + b) = a \cdot b$

$\text{LHS} = a \cdot (a' + b)$

$= (a \cdot a') + (a \cdot b)$ (Distributive rule)

$= 0 + (a \cdot b) \quad (a \cdot a' = 0)$

$= a \cdot b \quad (a + 0 = a)$

$= \text{RHS}$

(3) $(a \cdot b) + (a \cdot b') = a$

$\text{LHS} = (a + (a \cdot b')) \cdot (b + (a \cdot b'))$

$\Rightarrow ((a + a) \cdot (a + b')) \cdot ((b + a) \cdot (b + b'))$ (Distributive)

$\Rightarrow (a \cdot (a + b')) \cdot ((b + a) \cdot 1)$ (Idempotent)

Scanned with CamScanner

$\Rightarrow (a \cdot (a+b)) \cdot (b+a)$ (Absorption)

$\Rightarrow a \cdot (a+b)$ (Absorption)

$\Rightarrow a$

$\Rightarrow$ RHS

Example : 19

Show that in a Boolean algebra the law of the double complement holds.

(or)

Prove the involution law $(a')' = a$

Solution :

It is enough to show that

$$a' + a = 1 \quad \text{and} \quad a \cdot a' = 0$$

By domination laws of Boolean Algebra, we get

$$a + a' = 1 \quad \text{and} \quad a \cdot a' = 0$$

By commutative laws, we get

$$a' + a = 1 \quad \text{and} \quad a' \cdot a = 0$$

$\therefore$ Complement of a' is a

$$(a')' = a$$

i.e., $$a'' = a$$

Example : 20

Prove that $x \cdot (x+y) = x$

Solution :

$x \cdot (x+y) = (x+0) \cdot (x+y)$ ($\because x + 0 = x$ – Identity law)

$= x + 0 \cdot y$ (Distributive)

$= x + 0$ ($x \cdot 0 = 0$ - Dominance Rule)

$= x$

Example : 21

Simplify : (1) $(a \,.\, b)' + (a+b)'$

(2) $(a' \,.\, b' \,.\, c) + (a \,.\, b' \,.\, c)$

Solution :

(1) $(a \,.\, b)' + (a+b)'$

$= (a' + b') + (a' \cdot b')$ (Demorgan's law)

$= ((a' + b') + a') \cdot ((a' + b') + b')$ (Distributive law)

$= (a' + b') \cdot (a' \cdot b')$ (Idempotent)

$= (a' \cdot (a' \cdot b')) + (b' \cdot (a' \cdot b'))$ (Distributive law)

$= (a' \cdot a' \cdot b') + (b' \cdot b' \cdot a')$ (Associative law)

$= (a' \cdot b') + (a' \cdot b)$ (Idempotent)

$= (a' \cdot b')$ (Idempotent)

(2) $(a' \cdot b' \cdot c) + (a \cdot b' \cdot c)$

$= (a' + a) \cdot (b' \cdot c)$ (Distributive rule)

$= 1 \cdot (b' \cdot c)$ $(a + a' = 1)$

$= b' \cdot c (a \cdot 1 = a)$

Example : 22

Let a, b, c be any 3 elements of Boolean Algebra B, prove that (1) $a \cdot a = a$ *(2)* $a + a = a$

Solution :

Let $a = a \cdot 1$

Scanned with CamScanner

$$= a \cdot (a + a') \qquad (a + a' = 1)$$

$$= a \cdot a + a \cdot a' \qquad \text{(Distributive rule)}$$

$$= (a \cdot a) + 0 \qquad (\because a + a' = 0)$$

$$= a \cdot a \qquad (a + 0 = a)$$

$$\therefore \quad a = a \cdot a$$

By taking dual of $a \cdot a = a$, we have $a + a = a$

Example : 23

Let a, b, c ∈ B. Show that (1) $a \cdot 0 = 0$ (2) $a + 1 = 1$

Solution :

$$a \cdot 0 = (a \cdot 0) + 0 \qquad (a + 0 = a)$$

$$= (a \cdot 0) + (a \cdot a') \qquad (\because a + a' = 0)$$

$$= a \cdot (0 + a') \qquad \text{(Distributive rule)}$$

$$= a \cdot (a' + 0) \qquad \text{(Commutative rule)}$$

$$= a \cdot a' \qquad (a' + 0 = a)$$

$$= 0$$

$$\therefore \quad a \cdot 0 = 0 \qquad (a \cdot a' = 0)$$

By taking dual, of $a \cdot 0 = 0$, we have $a + 1 = 1$

Example : 24

Prove the following Boolean identities:

(1) $a + (a' \cdot b) = a + b$

(2) $(a \cdot b) + (a \cdot b') = a$

(3) $a \cdot (a' + b) = a \cdot b$

Solution :

(1) $$a + (a' \cdot b) = (a + a') \cdot (a + b) \qquad \text{(Distributive rule)}$$

$$= 1 \cdot (a + b) \qquad (\because a + a' = 1)$$

$$= a + b \qquad (a \cdot 1 = a)$$

(2) $$(a \cdot b) + (a \cdot b') = a \cdot (b + b') \qquad \text{(Distributive rule)}$$

$$= a \cdot 1 \qquad (b + b' = 1)$$

$$= a$$

$$\therefore \quad (a \cdot b) + (a \cdot b') = a \qquad (b \cdot 1 = b)$$

(3) $$a \cdot (a' + b) = (a \cdot a') + (a \cdot b) \qquad \text{(Distributive rule)}$$

$$= 0 + (a \cdot b)$$

$$= a \cdot b$$

$$\therefore \quad a \cdot (a' + b) = a \cdot b$$

Scanned with CamScanner

Example : 25

Show that

$$(a_1' \cdot a_2' \cdot a_3' \cdot a_4') + (a_1' \cdot a_2' \cdot a_3' \cdot a_4) + (a_1' \cdot a_2' \cdot a_3 \cdot a_4)$$
$$+ (a_1' \cdot a_2' \cdot a_3 \cdot a_4') = a_1' \cdot a_2'$$

Solution :

$$\left(a_1' \cdot a_2' \cdot a_3' \cdot a_4'\right) + \left(a_1' \cdot a_2' \cdot a_3' \cdot a_4\right) = a_1' \cdot a_2' \cdot a_3'$$

and

$$\left(a_1' \cdot a_2' \cdot a_3 \cdot a_4\right) + \left(a_1' \cdot a_2' \cdot a_3 \cdot a_4'\right) = a_1' \cdot a_2' \cdot a_3$$

Hence the given formula is equal to

$$\left(a_1' \cdot a_2' \cdot a_3'\right) + \left(a_1' \cdot a_2' \cdot a_3\right) = a_1' \cdot a_2'$$

Example : 26

Find the value of $a_1 . a_2 . \left[(a_1 . a_4) + a_2' + \left(a_3 \cdot a_1'\right)\right]$ *for* $a_1 = a$, $a_2 = 1$, $a_3 = b$ *and* $a_4 = 1$ *where a, b, 1 are elements of Boolean algebra.*

Solution :

By substituting the required elements in the expression, we get

$$a \cdot 1 \cdot [(a \cdot 1) + 1' + (b \cdot a')]$$
$$= a \cdot [a + 0 + (b \cdot b)]$$
$$= a \cdot (a + b)$$

$$= a \cdot 1$$
$$= a$$

Alternatively, the given expression can be written in an equivalent form as

$$\left(a_1 \cdot a_2 \cdot a_1 \cdot a_4\right) + \left(a_1 \cdot a_2 \cdot a_2'\right) + \left(a_1 \cdot a_2 \cdot a_3 \cdot a_1'\right)$$
$$= \left(a_1 \cdot a_2 \cdot a_4\right) + 0 + 0$$
$$= a_1 \cdot a_2 \cdot a_4$$

Now replacing variables, we get

$$a \cdot 1 \cdot 1 = a$$

Example : 27

Apply Demorgan's theorem to the following expression

(i) $\overline{(x + \bar{y})(\bar{x} + y)}$; (ii) $\overline{(a + b + c)\, d}$

Solution :

(i) $\overline{(x + \bar{y})(\bar{x} + y)} = \overline{a + \bar{b}} + \overline{\bar{a} + b}$ Demorgan's theorem

$= \bar{a} \cdot \bar{\bar{b}} + \bar{\bar{a}} \cdot b$ Demorgan's theorem

$= \bar{a} \cdot b + a \cdot \bar{b}$ $(\bar{\bar{a}} = a)$

$= a \oplus b$

(ii) $\overline{(a + b + c)\, d} = \overline{a + b + c} + \bar{d}$ Demorgan's theorem

$= \bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{d}$ Demorgan's theorem

Scanned with CamScanner