BYJU'S EXAM PREP

# Computer Science & IT

## Complete Syllabus

## Formulae Book

# IMPORTANT FORMULAS TO REMEMBER

## SUBJECT 1: COMPUTER NETWORKS

**Delay Calculations:**

- Propagation delay $= \dfrac{\text{Distance Travelled by frame or packet}}{\text{Propagation speed}}$
- Transmission delay $= \dfrac{\text{No. of bits to transfer}}{\text{Transmission speed or bandwidth } (\text{in bps})}$

**Channel Utilization:**

1. For Ethernet, channel utilization, $u = \dfrac{1}{1+5a}$, where $a = \dfrac{\text{propagation delay}}{\text{transmission delay}}$

   1.1. For 1- persistent CSMA/CD LAN 10-1000Mbps speed, simplified channel efficiency is,

   [Channel efficiency] $= \dfrac{T_d}{T_d + \dfrac{2\tau}{A}}$, where A = kp $(1 - p)^{k-1}$, k stations each with probability p to transmit in a contention slot. $T_d$ is the time to transmit a frame. $\tau$ is the worst case one way propagation delay.

   1.2. Efficiency of 802.3 Ethernet at 10Mbps with 512 bit connection:

   (Channel Efficiency) $= \dfrac{1}{1+\frac{2BLe}{cF}}$, where, B = network bandwidth. L = Cable Length. C = speed of signal propagation. E = optimal number if contention slots per frame. F = frame size

2. For Token ring (release after transmission or early token release),

   2.1. Channel utilization, $u = \dfrac{1}{1+\dfrac{a}{N}}$, where N is the number of stations in the ring.

3. For Token ring (release after reception or delayed token release),

   3.1. Channel utilization, $u = \dfrac{1}{1+a}$, where N is the number of stations in the ring.

4. For unrestricted simplex protocol: If a frame has d data bits and h overhead bits and channel bandwidth = b bits/sec then,

   4.1. Maximum channel utilization $= \dfrac{\text{Data size}}{\text{Frame size}} = \dfrac{d}{d+h}$

   4.2. Maximum data throughput $= \dfrac{\text{Data size}}{\text{Frame size}} \times \text{Bandwidth} = \dfrac{d}{d+h} \times b$

5. For stop-and-wait,
   5.1. Channel utilization, $u = \frac{1-p}{1+2a}$, where $a = \frac{\text{propagation delay}}{\text{transmission delay}}$ and p is the probability that a frame is in error.
   5.2. Also Maximum channel utilization = $\frac{\text{Time to transmit a frame}}{\text{Round trip time } (R)} \times \frac{d}{d+h} = \frac{d}{b \times R}$
   5.3. Maximum data throughput = $\frac{d}{b \times R} \times b = \frac{d}{R}$
6. For Simplex positive acknowledgement with retransmission (PAR) protocol: -
   6.1. Maximum channel utilization and throughput is similar to stop-and-wait protocol when the effect of errors is ignored.
7. For Sliding Window Protocols with window size of w,
   7.1. Go-Back-N,
      7.1.1. Channel utilization, u

$$= \begin{cases} \frac{1-p}{1+2ap} & \text{, if window fills the pipe i.e., } w \geq 2a+1 \\ \frac{w(1-p)}{(1+2a)(1-p+wp)} & \text{, if window does not fills the pipe i.e., } w < 2a+1 \end{cases}$$

   7.2. Selective reject,
      7.2.1. Channel utilization, $u = \begin{cases} (1-p) & \text{, if window fills the pipe i.e., } w \geq 2a+1 \\ \frac{w(1-p)}{(1+2a)} & \text{, if window does not fills the pipe i.e., } w < 2a+1 \end{cases}$
   7.3. Condition for maximum utilization or throughput is:
      [Time to transmit w frames] ≥ [Round Trip Time]

**Throughput Calculations:**

Throughput = Channel Utilization × Channel Bandwidth

**Signal and Noise Calculations:**

1. Signal to Noise Ratio (in decibels, **dB**) = $10\log_{10}\frac{S}{N}$,
   a. where S = Signal strength and N = noise strength.
2. Signal Attenuation (in decibels, **dB**) = $10\log_{10}\frac{\text{Transmitted Power}}{\text{Received Power}}$

**Data Rate and Channel Capacity Calculations:**

1. Nyquist Theorem: Maximum data rate = $2H \log_2 V$ bits/sec, where H is bandwidth in hertz (Hz) and V is number of levels.
2. Shannon's theorem: Channel capacity = $H \log_2\left(1+\frac{S}{N}\right)$ bits/sec, where H is bandwidth in hertz (Hz).

   (Note: here $\frac{S}{N}$ is **not** in decibels).

**Baud rate:** A baud is the number of changes per second in the signal.

- For Manchester encoding, **baud rate = 2 × bit-rate**

**MAC Sub layer:**

Static channel allocation in LANs and MANs.

If C = channel capacity in bps

λ = arrival rate of frames (frames/sec)

$\frac{1}{\mu}$ = no. of bits per frame, then

Mean time to delay, $T = \frac{1}{\mu C - \lambda}$,

Now, if the channel is divided into N sub channels each with capacity $\frac{C}{N}$ and arrival rate or input rate on each of the N channels is $\frac{\lambda}{N}$ then,

$$T'(fdm) = \frac{1}{\mu\frac{C}{N} - \frac{\lambda}{N}} = \frac{N}{\mu C - \lambda}$$

**Dynamic Channel Allocation:**

$\begin{bmatrix} \text{Probability of a frame} \\ \text{being generated in a period of length } \Delta t \end{bmatrix} = \lambda \Delta t$, Where $\lambda$ is the arrival rate of frames.

**Multiple access protocol:**

**Pure ALLOHA protocol**

- Infinite senders assumed.
- Poisson distribution with mean rate **S** frames per frame time
- Combined frame rate with retransmission **G** frames per frame time.
- '**t**' is the time required to transmit a frame.
- In multiple access protocol, a frame is successful if no other frames are transmitted in the vulnerable period.
- Probability of **k** frames being generated during a frame transmission time:

  $P_k = \frac{G^k e^{-G}}{k!}$
- Hence, probability of zero frames in 2 frame periods is, $P_0 = e^{-2G}$
- Therefore, for pure ALLOHA,

  - Mean rate $S = GP_0 = Ge^{-2G}$ which becomes maximum at G = ½,

    Max(S) = $\frac{1}{2e}$ = 0.184 = 18.4% throughput.
- Vulnerability period in pure ALLOHA: For successful frame transmission, no other frame should be on the channel for vulnerability period equal to twice the time to transmit one frame. That is,

  $\begin{pmatrix} \text{Vulnerability period} \\ \text{in case of PURE ALLOHA} \end{pmatrix}$ = 2t , where t is the time to transmit one frame.

**Slotted ALLOHA protocol**

- Time is divided into discrete frame slots.
- A station is required to wait for the beginning of the next slot to transmit a frame.
- Vulnerability period is halved as opposed to pure ALLOHA protocol. That is,

  $\begin{pmatrix}\text{Vulnerability period}\\ \text{in case of SLOTTED ALLOHA}\end{pmatrix}$ = t, where t is the time to transmit one frame.
- Probability of **k** frames being generated during a frame transmission time:

  $P_k = e^{-G}(1 - e^{-G})^{k-1}$
- Hence, probability of zero frames in 1 frame period is, $P_0 = e^{-G}$

  - Mean rate $S = GP_0 = Ge^{-G}$ which becomes maximum at G = 1,

  Max(S) = $\frac{1}{e}$ = 0.368 = 36.8% throughput.
- Expected number of retransmissions, $E = e^G$.

**PPP**

- In **Point to Point Protocol** (PPP), number of channels grows as square of the number of computers. That is, $\begin{pmatrix}\text{Number of channels or links}\\ \text{for n computers}\end{pmatrix} = \frac{n(n-1)}{2}$

**Binary Exponential Back-off Algorithm:**

- After i collisions wait a random number of slots between 0 and $2^i - 1$ with a maximum of 1023. (**Note:** After 16 collisions, failure is reported to the higher layers.)

**Minimum frame size for IEEE 802.3 (Ethernet) frame = 64 bytes.**

| ROUTING ALGORITHMS | STATE |
|---|---|
| Shortest Path algorithm | Non-Adaptive (or Static) |
| Flooding Algorithm | Non-Adaptive (or Static) |
| Flow Based Routing Algorithm, It uses the formula, Delay time, T = $\frac{1}{\mu C - \lambda}$, where C = channel capacity, 1/μ = mean packet size in bits, and λ = mean number of arrivals in packets/sec | Non-Adaptive (or Static) |
| Distance Vector Routing (DVR)<br>• Based on Bellman-Ford Algorithm and the Ford-Fulkerson Algorithm.<br>• It suffers from the "Count to infinity problem"<br>• Exchange information of the entire network with its neighbours. | Adaptive Algorithm (or non-static) |

| ROUTING ALGORITHMS | STATE |
|---|---|
| Link State Routing Algorithm (LSR)<br>• Discovers its neighbours and construct a packet telling all it has just learned and send this packet to all other routers. | Adaptive algorithm (or non-static) |
| Hierarchical Routing Algorithm<br>• For N router subnet, the total number of levels = ln N<br>• And each router will have e × ln N number of entries in their routing tables. | Adaptive Algorithm (or non-static) |
| Broadcast Routing Algorithm | Adaptive Algorithm (or non-static) |

- Congestions deals with wires and routers while flow deals with hosts.
- Traffic Shaping:
  - Leaky Bucket Algorithm (If the bucket overflows, then packets are discarded).
  - Token Bucket Algorithm (causes a token to be generated periodically).
- Congestion control through Load Shedding may lead to deadlock and unfairness.
- The size of data portion of the datagram is = [Total length] – [size of header]
- Maximum length of the datagram = 65535 bytes.

**Datagram format:**

| | |
|---|---|
| Total length | 16 bits |
| Header length | 4 bits |
| Flag length | 3 bits |
| Type of service | 8 bits |
| Identification bits | 16 bits |
| Fragment offset | 13 bits |
| Time to Live | 8 bits |
| Protocol version | 8 bits |
| Header Checksum | 16 bits |
| Source Address | 32 bits |
| Destination Address | 32 bits |

- In Layer 2 of OSI model (Data link layer), destination field appears before source field where as in layer 3 (Network layer), the ordering is opposite.

**IP class addressing:**

| Class Name | Starts with | Range |
|---|---|---|
| Class A | 0 | 0-127 |
| Class B | 10 | 128-191 |
| Class C | 110 | 192-223 |
| Class D | 1110 | 224-239 |
| Class E | 11110 | 240-255 |

- Internet addresses refer to network connections rather than hosts. (For example, Gateways have two or more network connections and each interface has its own IP address). Thus, Physical (or Ethernet) address is constant (fixed) but Network (or IP address) may change.

**Transport Layer:**

To cope with the widely varying delays, TCP maintains a dynamic estimate of the current RTT (Round Trip Time) calculated this way:

- When sending a segment, the sender starts a timer.
- Upon receipt of an acknowledgement, it stops the timer and record the actual elapsed delay M between sending the segment and receiving its acknowledgement.
- Whenever a new value M for the current RTT is measured, it is averaged into a smoothed RTT depending on the last measured value as follows:
  - **New RTT = a (Old RTT) + (1-a)(Current RTT M),** where **a** is known as the smoothing factor and it determines how much weight the new measurement carries. When **a** is 0, we simply use the new value and when **a** is 1 we ignore the new value. Typically, the value of **a** lies between 0.8 and 0.9
- **TCP** can provide reliable service where UDP cannot as they choose to use. IP in two different modes of service provided by IP either reliable or connectionless.
- A transparent router connects a LAN to a WAN.
- TCP does not have in-built mechanism to distinguish between a timeout owing to error and that owing to congestion.
- During early start phase, the congestion window inside TCP grows Exponentially, whereas after that it grows linearly.
- If two clients try to connect to the same TCP service, then they can
- Private network address enables reuse of IP addresses, thereby aiming to reduce the IP shortage problem.

**CLIENT SERVER MODEL:**

| SERVER | CLIENT |
|---|---|
| Socket | Socket |
| Bind | Connect |
| Listen() | Send() |
| Accept() | Receive() |
| Receive() | Close() |
| Send() | |

- Terms connected with RPCs(Remote Procedural Calls):
  - Marshalling, Shunt
  - Client, Skelton
  - Procedure call, server
- Terms not-connected with RPCs:
  - Socket, connect.
- While trying to access a webpage www.facebook.com, a user gets the "http" error message "could not resolve host name". What could be the cause of this problem?
  - The local DNS is not running.
- In TCP "SYN" flag is used for connection establishment.
- **A**wants to send a group message to a number of people. How can it be ensured that the message actually came from **A** and not from someone else?
  - **A** encrypts via its **own private key**; and the group decrypts via **A's public key**.
- The number of bits used for IPV6 addressing is 128 bits whereas for Ethernet address 48 bits are used.
- If n is the number of bits used to represent the frame sequence number then:

| Name of sliding window protocol | Sender window size | Receiver window size |
|---|---|---|
| Go-back-N | $2^n - 1$ | 1 |
| Selective Repeat | $2^{n-1}$ | $2^{n-1}$ |

- The maximum burst rate at which network can handle the traffic is given by,
  - $S = \frac{C}{M - R}$, where C = Capacity of bucket(in bits); M = Network rate(e.g., token ring with 10 Mbps network LAN); R = arrival rate(e.g., rate of entering into the bucket).

- In any sliding window protocol,
  - The optimal window size = RRT × Channel Bandwidth,
- Residual Error Rate = $\frac{\text{Number of lost or garbled messages}}{\text{Total Sent}}$
- Ping Works on IP Layer / Network layer.

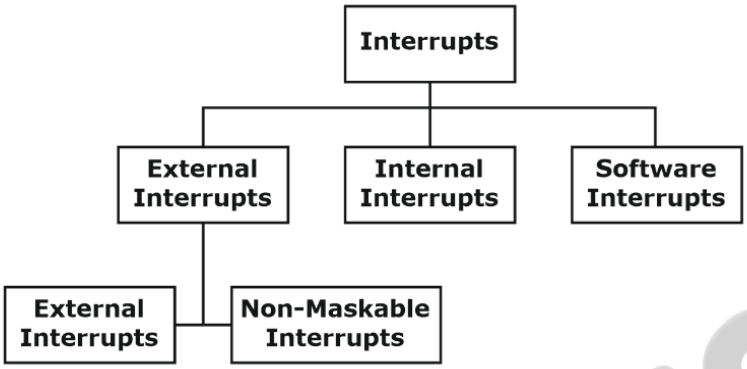
**Devices and their OSI Layer:**

| | |
|---|---|
| GATEWAY | Application Layer, Presentation Layer, Session Layer, Transport Layer |
| Router | Network Layer |
| Bridges and Switches | Data Link Layer |
| Repeaters, Hubs, Amplifiers, Multiplexers | Physical Layer |

- A group of $(2^n - 1)$ routers are interconnected in a centralized binary tree, with a router at each node. Router **I** communicate with router **J** by sending a message to the root of the tree. The root then sends the message back down to J, then

$$\left(\begin{array}{c}\text{The mean number of hops}\\ \text{per message for large n}\\ \text{Assuming that all router pairs}\\ \text{are equally likely}\end{array}\right) = \frac{2\left(2^n \times n - 2\left(2^n - 1\right)\right)}{2^n - 1},$$

## SUBJECT 2 : COMPUTER ORGANIZATION AND ARCHITECTURE




- Maskable interrupts are enabled and disabled by executing instructions such as EI or DI. If the computer's interrupts is disabled, the computer ignores the maskable interrupt.
  - Interrupt is disabled → Interrupt flag bit = 1
  - Interrupt is enabled → Interrupt flag bit = 0
- The Non-Maskable interrupt has higher priority than the maskable interrupt.
- If both maskable and non-maskable interrupts are activated at the same time, the processor will service the non-maskable interrupt first.
- Non- maskable interrupts are typically used in power failure interrupts

**PIPELINE**

- $\begin{pmatrix}\text{Clock Period of}\\ \text{pipeline}\end{pmatrix}$ = MaxDelay (T1, T2, T3,…) + L, where $T_i$ = pipeline segment delay; L = Latch delay.
- (Speed Up of Pipeline) $= \dfrac{\text{Time Taken to perform the Task without Pipeline}}{\text{Time Taken to perform the Task with Pipeline}}$
- Throughput of pipeline $= \dfrac{\text{Total number of tasks to perform}}{\text{Total time to perform the task with pipeline}}$
- Efficiency of pipeline $= \dfrac{\text{Speedup of pipeline}}{\text{Maximum speed up of pipeline}}$
- Maximum speed up of pipeline= Total number of pipe segments in the pipeline.

**Hazards of Pipeline:**

Branch Instruction: For each branch instruction additional (n-1) pipeline clocks are required; where n is the number of segments in the pipeline.

Data dependency

Resource Conflicts.

- $\begin{pmatrix}\text{Total number of clocks required to process}\\ \text{a stream of S instructions}\end{pmatrix}$ = (n + S – 1) + SC(n – 1), where S= no. of instructions; n = no. of pipe segments; C= Probability for an instruction to be a conditional branch.
- Cache works on the principle of locality.
- Each process has its own virtual address space.
- The unit block of data for transferring between disk and memory is called a page. (Typically 4kB)
- TLB is actually a cache for the page table.
- Virtual memory uses write back method because frequent access to disk is very slow.
- A DRAM consists of 1 transistor and 1 capacitor.
- An interrupt in which device supplies the interrupt requests as well as its address is called Vectored Interrupt.
- When INTER is encountered, the processor branches to the memory location which is-
  - Determined by the RST N instruction given by the IO device.
  - Determined by the Call address instruction given by the IO device.
- In 8085 microprocessor,
  - XCHG: Exchange the content of the HL and DE register pair respectively.
  - DAD H: Add HL pair with HL pair and store the result in HL pair.
- Thrashing implies excessive page IO
- In 8085, a stack pointer is a 16-bit register that points to stack.
- (Read Stall Cycles) = Read × Read miss rate × Read miss rate × Read miss penalty,
- (Write Stall Cycles) = writes × write miss rate × write miss penalty + write buffer stall
- (Memory Stall Cycles) = Memory Access × Cache Miss rate × Cache Miss Penalty
- $(\text{CPU Time}) = \text{Ic} \times \left(\text{CPI} + \frac{\text{Memory clock cycle}}{\text{Instruction}}\right) \times \text{Clock cycle time}$, where Ic= number of instructions; CPI=CPU clock cycles;
- PARAM is a parallel computer.
- ALE is a pin of an 8085 microprocessor that is used to the latch the 8 bits of address lines AD7-AD0 into an external latch.
- The magnetic bubble memory has an access time of 30μs.
- The magnetic material that is normally used in bubble memories is GARNET.

**For Relative addressing:**

- Effective address = Program counter + Address part of instruction.
  For example: 2 word instruction, I = JMP 65 stored in 2000 in decimal.
  So EA = PC + Address part; EA = 2002 + 65 =2067 in decimal

- (CPU time on machine 1) $= \frac{\text{CPU clock cycles of the program}}{\text{Clock rate for M1}}$

- $(\text{Speed Up}) = \dfrac{\text{Performance with enhancement}}{\text{Performance without enhancement}} = \dfrac{\text{Execution time}(\text{old})}{\text{Execution time }(\text{new})}$
- Memory Chip notation: (Memory Depth) × (Memory Width) ; e.g. 16M * 8 = 128 Mb
- (Memory Density) = (Memory Depth) × (Memory Width)
- Chip notation: (Memory Depth per bank) × (Memory Width) × (number of banks)

**For memory access:**

- Cycle Time = Latency Time + Transfer Time.

(Utilization Factor) = $= \dfrac{\text{No. of clock cycles in pipeline}}{\text{No. of clock cycles in non}-\text{pipeline}}$

Average Instruction Execution Time = (Average CPI) × (Clock cycles time)

Maximum Forbidden latency: ≤ (n – 1) ; where n = number of columns in the reservation table.

- A SIMD (Single instruction multiple data) is characterized by, C = < N, F, I, M > notation.
- $\begin{pmatrix}\text{Average Memory}\\ \text{Access Time}\end{pmatrix}$ = (Hit time) + (Miss Rate) × (Miss Penalty)
- (Effective Memory Access Time) = (Clock cycle time of pipeline) + (Stall Time) × (Stall Frequency)
- $\begin{pmatrix}\text{Effective Memory}\\ \text{Access Time}\end{pmatrix}$ = (Hit Rate) (Hit time) + (Miss Rate) × (Miss Penalty)
- $\begin{pmatrix}\text{Average Instruction}\\ \text{Execution time in a k–stage pipeline}\end{pmatrix} = \begin{pmatrix}\text{Clock cycle time}\\ \text{of pipeline}\end{pmatrix}$ + (Stall Time) × (Stall Frequency)
- How many bits will be required to store a decimal number containing
  (a) 3 digit (b) 4 digit (c) 96 digits
  - Key formula is No. of possibilities = $(\text{base})^{\text{No. of digits}}$
  - (No. of possibilities in base 2) = (No. of possibilities in base 10) = (No. of possibilities in any base)
  - Let N be the required number of bits and n be the number of decimal digits given, then
  - $2^N = 10^n \rightarrow N = \lfloor \log_2 10^n \rfloor + 1$; so for 96 decimal digit number we would need
  - $N = \lfloor \log_2 10^n \rfloor + 1 = \lfloor \log_2 10^{96} \rfloor + 1 = 318 + 1 = 319$ bits. in binary number system

**For page table:**

- $\begin{pmatrix}\text{No. of comparators}\\ \text{required}\end{pmatrix} = (\text{No. of bits in the line offset or index field of CPU address})$
- $\begin{pmatrix}\text{Bit size of}\\ \text{each comparator}\end{pmatrix} = (\text{No. of bits the TAG field of the CPU address})$
- $\begin{pmatrix}\text{Maximum number of files}\\ \text{in a file system}\end{pmatrix} = \dfrac{\text{Directory size}}{\text{Dictionary Entry Size}}$

**Amdahl's Law:**

- Maximum speed up with N processors = $\frac{1}{(1-P)+\frac{P}{N}}$, P is the proportion of a program that can be made parallel (i.e., benefit from parallelization); (1 − P) is the proportion that cannot be parallelized (remains serial).

## SUBJECT 3: THEORY OF COMPUTATION

- If an NFA contains N no. of States then the corresponding DFA will contain maximum of $2^N$ states.
- The minimum number of states in DFA accepting strings containing number of 0's divisible by P and 1's divisible by Q = P × Q
- The number of possible DFA's with N states and M input symbol = $2^N N^{N \times M}$
- For the strings over some alphabet ∑, the following are primitive regular expressions:
  - X, for each x ∈ ∑
  - λ, the empty string
  - ϕ, indicating number of string.

  Thus if |∑| = n, then there are (n + 2) Primitive regular expressions defined over ∑.

  Every primitive regular expression is a regular expression.
- Identities for regular expressions:
  - ∅ + R = R + ∅ = R
  - ∅.R = R.∅ = ∅
  - ε.R = R.ε = R
  - ε* = ε &∅* = ε
  - R + R = R
  - RR* = R*R = $R^+$
  - ((R*)*) = R*
  - R*R* = R*
  - (P + Q)* = (P*Q*)*; Where P and Q are regular expression.
  - R(P + Q) = RP + RQ & (P + Q)R = PR + QR
  - P(QP)* = (PQ)*P
- **Arden's theorem:**
  - If P & Q are two regular expressions over an alphabet S such that P does not contain ε then the following equation:

  **R = Q + RP** in R has a unique solution(only one solution) i.e., R = QP*

**Push Down Automata:** Pushdown Automata has extra memory called stack which gives more power than Finite automata. It is used to recognize context free languages.

**Deterministic and Non-Deterministic PDA:** In deterministic PDA, there is only one move from every state on every input symbol but in Non-Deterministic PDA, there can be more than one move from one state for an input symbol.

**Note:**

- Power of NPDA is more than DPDA.
- It is not possible to convert every NPDA to corresponding DPDA.
- Language accepted by DPDA is subset of language accepted by NPDA.
- The languages accepted by DPDA are called DCFL (Deterministic Context Free Languages) which are subset of NCFL (Non Deterministic CFL) accepted by NPDA.

**Linear Bound Automata:** Linear Bound Automata has finite amount of memory called tape which can be used to recognize Context Sensitive Languages.

- LBA is more powerful than Push down automata.

**FA < PDA < LBA < TM**

**Turing Machine**: Turing machine has infinite size tape and it is used to accept Recursive Enumerable Languages.

- Turing Machine can move in both directions. Also, it doesn't accept ε .
- If the string inserted in not in language, machine will halt in non-final state.

**Deterministic and Non-Deterministic Turing Machines:** In deterministic turing machine, there is only one move from every state on every input symbol but in Non-Deterministic turing machine, there can be more than one move from one state for an input symbol.

**Note:**

- Language accepted by NTM, multi-tape TM and DTM are same.
- Power of NTM, Multi-Tape TM and DTM is same.
- Every NTM can be converted to corresponding DTM.

**Decidable and Undecidable Problems:**

A language is **Decidable or Recursive** if a Turing machine can be constructed which accepts the strings which are part of language and rejects others. e.g.; A number is prime or not is a decidable problem.

A language is **Semi–Decidable or Recursive Enumerable** if a turing machine can be constructed which accepts the strings which are part of language and it may loop forever for strings which are not part of language.

A problem is **undecidable** if we can't construct an algorithms and Turing machine which can give yes or no answer. e.g.; Whether a CFG is ambiguous or not is undecidable.

| Decidability Table | | | | | | |
|---|---|---|---|---|---|---|
| **Problem** | **RL** | **DCFL** | **CFL** | **CSL** | **RL** | **REL** |
| Membership Problem | D | D | D | D | D | UD |
| Emptiness Problem | D | D | D | UD | UD | UD |
| Completeness Problem | D | UD | UD | UD | UD | UD |
| Equality Problem | D | D | UD | UD | UD | UD |
| Subset Problem | D | UD | UD | UD | UD | UD |
| L1 ∩ L2 = ϕ | D | UD | UD | UD | UD | UD |
| Finiteness | D | D | D | UD | UD | UD |
| Complement is of same type | D | D | UD | D | D | UD |
| Intersection is of same type | D | UD | UD | UD | UD | UD |
| Is L regular | D | D | UD | UD | UD | UD |

**Closure Properties :**

| **Operation** | **REG** | **DCFL** | **CFL** | **CSL** | **RC** | **RE** |
|---|---|---|---|---|---|---|
| Union | Y | N | Y | Y | Y | Y |
| Intersection | Y | N | N | Y | Y | Y |
| Set difference | Y | N | N | Y | Y | N |
| complementation | Y | Y | N | Y | Y | N |
| Intersection with a regular language | Y | Y | Y | Y | Y | Y |
| Union with a regular language | Y | Y | Y | Y | Y | Y |
| Left Difference with a regular language (L-regular) | Y | Y | Y | Y | Y | Y |
| Right Difference with a regular language (Regular-L) | Y | Y | N | Y | Y | N |
| Concatenation | Y | N | Y | Y | Y | Y |
| Kleene star | Y | N | Y | Y | Y | Y |
| Kleene plus | Y | N | Y | Y | Y | Y |
| Reversal | Y | Y | Y | Y | Y | Y |
| Epsilon-free homomorphism | Y | N | Y | Y | Y | Y |
| Homomorphism | Y | N | Y | N | N | Y |
| Inverse homomorphism | Y | Y | Y | Y | Y | Y |

| Operation | REG | DCFL | CFL | CSL | RC | RE |
|---|---|---|---|---|---|---|
| Epsilon-free substitution | Y | N | Y | Y | Y | Y |
| Substitution | Y | N | Y | N | N | Y |
| Right quotient with a regular language | Y | Y | Y | N | Y | Y |
| Left quotient with a regular language | Y | Y | Y | N | Y | Y |
| Subset | N | N | N | N | N | N |

**Countability:**

Set of all strings over any finite alphabet are countable.

Every subset of countable set is either finite or countable.

Set of all Turing Machines are countable.

The set of all languages that are not recursive enumerable is Uncountable.

- Some important results:
  - If L1 is DCFL and L2 is regular, then L1 ∪L2 is also DCFL
  - If L1 is DCFL and L2 is regular, then L1 ∩L2 is also DCFL
  - Every regular language is DCFL
  - The union of DCFL & regular language is DCFL
  - DPDA ≠ NPDA
  - DPDA ⊂ NPDA
  - Power of DPDA < Power of NPDA
  - $\left(\begin{matrix}\text{Primitive Recursive}\\ \text{function}\end{matrix}\right) \subset \left(\begin{matrix}\text{Total Recursive}\\ \text{function}\end{matrix}\right) \subset \left(\begin{matrix}\text{Partial Recursive}\\ \text{function}\end{matrix}\right)$
  - All RE languages are TM enumerable.
  - The power set of an infinite set is not countable
  - Not all languages are recursively enumerable.
  - Recursive languages are closed under complementation but recursively enumerable languages are NOT closed under complementation.
  - The complement of a CFL may or may not be closed under complementation.

# SUBJECT 4 : DISCRETE MATHEMATICS

## Prepositional logic

1. **Implication( →)**: For any two propositions p and q, the statement "if p then q" is called an implication and it is denoted by p → q.

| p | q | p→q |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |

2. **if and only if(↔)**: For any two propositions p and q, the statement "p if and only if(iff) q" is called a biconditional and it is denoted by p ↔ q.

| p | q | p ↔ q |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | T |

**De Morgan's Law** :

- ¬(p ∧ q) ≡ ¬p ∨ ¬q
- ¬(p ∨ q) ≡ ¬p ∧ ¬q

1. Implication :p → q
2. Converse : The converse of the proposition p → q is q → p
3. Contrapositive : The contrapositive of the proposition p → q is ¬q → ¬p
4. Inverse: The inverse of the proposition p → q is ¬p →¬q

**Types of propositions based on Truth values**

1. **Tautology** - A proposition which is always true, is called a tautology.
2. **Contradiction** - A proposition which is always false, is called a contradiction.
3. **Contingency** - A proposition that is neither a tautology nor a contradiction is called a contingency.

**There are two very important equivalences involving quantifiers**

1. ∀x(P(x) ∧ Q(x)) ≡ ∀xP(x) ∧ ∀xQ(x)
2. ∃x(P(x) ∨ (x)) ≡ ∃xP(x) ∨ ∃xQ(x)

**Rules of inference**

| Rule of Inference | Tautology | Name |
|---|---|---|
| $\begin{array}{l} p \\ \underline{p \to q} \\ \therefore q \end{array}$ | $(p \wedge (p \to q)) \to q$ | Modus Ponens |
| $\begin{array}{l} \neg q \\ \underline{p \to q} \\ \therefore \neg q \end{array}$ | $(\neg q \wedge (p \to q)) \to \neg p$ | Modus Tollens |
| $\begin{array}{l} p \to q \\ \underline{q \to r} \\ \therefore p \to q \end{array}$ | $((p \to q) \wedge (q \to r)) \to (p \to r)$ | Hypothetical syllogism |
| $\begin{array}{l} \neg q \\ \underline{q \vee r} \\ \therefore q \end{array}$ | $(\neg p \wedge (p \vee q)) \to q$ | Disjunctive Syllogism |
| $\begin{array}{l} p \\ \hline \therefore (p \vee q) \end{array}$ | $p \to (p \vee q)$ | Addition |
| $\begin{array}{l} (p \wedge q) \to r \\ \hline \therefore p \to (q \to r) \end{array}$ | $((p \wedge q) \to r) \to (p \to (q \to r))$ | Exportation |
| $\begin{array}{l} p \vee q \\ \underline{\neg p \vee r} \\ \therefore q \vee r \end{array}$ | $((p \vee q) \wedge (\neg p \vee r)) \to q \vee r$ | Resolution |

## Set theory

A **Set** is an unordered collection of objects, known as elements or members of the set.

An element 'a' belong to a set A can be written as 'a ∈ A', 'a ∉ A' denotes that a is not an element of the set A.

### Equal sets

Two sets are said to be equal if both have same elements. For example A = {1, 3, 9, 7} and B = {3, 1, 7, 9} are equal sets.

**NOTE: Order of elements of a set doesn't matter.**

### Subset

A set A is said to be **subset** of another set B if and only if every element of set A is also a part of other set B.

Denoted by '⊆'.

'A ⊆ B ' denotes A is a subset of B.

To prove A is the subset of B, we need to simply show that if x belongs to A then x also belongs to B.

To prove A is not a subset of B, we need to find out one element which is part of set A but not belong to set B.

**Size of a Set**

Size of a set can be finite or infinite.

For example

Finite set: Set of natural numbers less than 100.

Infinite set: Set of real numbers.

Size of the set S is known as **Cardinality number**, denoted as |S|.

Note: Cardinality of a null set is 0.

**Power Sets**

The power set is the set all possible subset of the set S. Denoted by P(S).

Example: What is the power set of {0, 1, 2}?

Solution: All possible subsets

{∅}, {0}, {1}, {2}, {0, 1}, {0, 2}, {1, 2}, {0, 1, 2}.

Note: Empty set and set itself is also the member of this set of subsets.

**Cardinality of power set** is $2^n$, where n is the number of elements in a set.

**Cartesian Products**

Let A and B be two sets. Cartesian product of A and B is denoted by A × B, is the set of all ordered pairs (a, b), where a belong to A and b belong to B.

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

**The cardinality of A × B** is N*M, where N is the Cardinality of A and M is the cardinality of B.

**Group-**

A non-empty set G, (G, *) is called a group if it follows the following axiom:

- **Closure:**(a*b) belongs to G for all a, b ∈ G.
- **Associativity:** a*(b*c) = (a*b)*c ∀ a, b, c belongs to G.
- **Identity Element:** There exists e ∈ G such that a*e = e*a = a ∀ a ∈ G
- **Inverses:** ∀ a ∈ G there exists $a^{-1}$ ∈ G such that $a*a^{-1} = a^{-1}*a = e$

**Relations:**

|A| = m and |B| = n, then

1. No. of functions from A to B = $n^m$
2. No. of one to one function = (n, P, m)
3. No. of onto function =$n^m$ – (n, C, 1)*$(n-1)^m$ + (n, C, 2)*$(n-2)^m$ …. +$(-1)^m$*(n, C, n-1), if m >= n; 0 otherwise
4. Necessary condition for bijective function |A| = |B|
5. The no. of bijection function =n!
6. No. of relations =$2^{mn}$

- For a set A of n distinct elements:

7. Number of symmetric and reflexive relations = $2^{\frac{n(n-1)}{2}}$
8. Number of symmetric relations = $2^{\frac{n(n+1)}{2}}$
9. Number of reflexive relations = $2^{n(n-1)}$
10. Number of irreflexive relations = $2^{n(n-1)}$
11. Number of transitive relations = 1,2,13,171,3994 … for 1, 2, 3, 4, 5,.. elements
12. Number of anti-symmetric relations = $2^n 3^{\frac{n(n-1)}{2}}$
13. Number of reflexive and anti-symmetric relations = $3^{\frac{n(n-1)}{2}}$
14. Number of equivalence relations = 1,2,5,15,52,203… for 1,2,3,4,5,6,… elements. (i.e., the BELL number).
15. Number of all possible relations = $2^{n^2}$

- A relation is a partial order if
  - Reflexive
  - Antisymmetric
  - Transitive
- Meet Semi Lattice :
- For all a, b belongs to L $a \wedge b$ exists
- Join Semi Lattice
- For all a, b belongs to L $a \vee b$ exists
- A poset is called Lattice if it is both meet and join semi-lattice

  Complemented Lattice: Every element has complement

  Distributive Lattice : Every Element has zero or 1 complement .

  Boolean Lattice: It should be both complemented and distributive. Every element has exactly one complement.
- A relation is an equivalence if
  - Reflexive
  - symmetric
  - Transitive

**Graph Theory:**

1. No. of edges in a complete graph = n(n-1)/2
2. Bipartite Graph : There is no edges between any two vertices of same partition . In complete bipartite graph no. of edges =m*n
3. Sum of degree of all vertices is equal to twice the number of edges.
4. Maximum no. of connected components in graph with n vertices = n
5. Minimum number of connected components =

   0 (null graph)

   1 (not null graph)

6. Minimum no. of edges to have connected graph with n vertices = n-1
7. To guarantee that a graph with n vertices is connected, minimum no. of edges required = {(n-1)*(n-2)/2 } + 1
8. A graph is euler graph if it there exists atmost 2 vertices of odd – degree
9. Tree
   - Has exactly one path btw any two vertices
   - not contain cycle
   - connected
   - no. of edges = n -1
10. For complete graph the no . of spanning tree possible = $n^{n-2}$
11. For simple connected planar graph
    - A graph is planar if and only if it does not contain a subdivision of $K_5$ and $K_{3,3}$ as a subgraph.
    - Let G be a connected planar graph, and let n, m and f denote, respectively, the numbers of vertices, edges, and faces in a plane drawing of G. Then n – m + f = 2.
    - Let G be a connected planar simple graph with n vertices and m edges, and no triangles. Then m ≤ 2n – 4.
    - Let G be a connected planar simple graph with n vertices, where n? 3 and m edges. Then m ≤ 3n – 6.
12. Every bipartite graph is 2 colourable and vice versa
13. The no. of perfect matchings for a complete graph $(2n)/(2^n n!)$
14. The no. of complete matchings for $K_{n.n}$ = n!

## SUBJECT 5 : DATA STRUCTURE AND ALGORITHMS

C operators in order of precedence (highest to lowest). Their associativity indicates in what order operators of equal precedence in an expression are applied.

| Operator | Description | Associativity |
|---|---|---|
| ()<br>[]<br>.<br>->><br>++ -- | Parentheses(function call)<br>Brackets (array subscript)<br>Member selection via objet name<br>Member selection via pointer<br>Postfix increment / decrement | Left-to-Right |
| ++ --<br>+ -<br>! ~<br>(type)<br>*<br>&<br>Sizeof | Prefix increment / decrement<br>Unary Plus / minus<br>Logical negation/bitwise complement<br>Cast (convert value to temporary value of type)<br>Dereference<br>Address (of operand)<br>Determine size in bytes on this implementation | Right-to-Left |
| * / % | Multiplication / division / modulus | Left-to-Right |
| + - | Addition / Subtraction | Left-to-Right |
| <<>> | Bitwise shift left, Bitwise shift right | Left-to-right |
| <<=<br>>>= | Relational less than/ less than or equal to<br>Relational greater than/ greater than or equal to | Left to right |
| == !== | Relational is equal to/is not equal to | Left to right |
| & | Bitwise AND | Left to right |
| ^ | Bitwise exclusive OR | Left to right |
| \| | Bitwise inclusive OR | Left to right |
| && | Logical AND | Left to right |
| \|\| | Logical OR | Left to right |
| ?: | Ternary conditional | Right to left |
| =<br>+= -=<br>*= /=<br>%= &=<br>^= \|=<br><<= >>= | Assignment<br>Addition/subtraction assignment<br>Multiplication / division assignment<br>Modulus / bitwise AND assignment<br>Bitwise exclusive / inclusive OR assignment<br>Bitwise shift left / right assignment | Right of left |
| , | Comma (separate expression) | Left to right |

- Radix sort is a non-comparative integer sorting algorithm.
  - Worst case time complexity = O(kN); N number of numbers each with k digits.

- Worst case space complexity = O( k + N )

- Insertion sort is stable and inplace sorting.
  - Best case time complexity = O(n) and
  - worst case time complexity = $O(n^2)$
- Binary Insertion sort
  - Best case time complexity = O(nlog n)
  - Worst case time complexity = $O(n^2)$
- For a look up in a B+ tree, we traverse from root node to leaf node. If there are K search keys and number of pointers per node (or degree) is P, then for a look up,
  - $\begin{pmatrix}\text{The path is}\\ \text{no longer than}\end{pmatrix} = \frac{\left\lceil \log \left\lfloor \frac{K+1}{2} \right\rfloor \right\rceil}{\left\lceil \log \left\lceil \frac{P}{2} \right\rceil \right\rceil}$
- Number of articulation points in a tree = number of internal nodes.
- For a full K-ary tree,
  - Total number of nodes = K × (number of internal nodes) + 1
- Formula that computes the address LOC(A[J,K]) of A[J,K]
  - Column-major order:
    LOC(A[J, K]) = BASE(A) + w[M(K – 1) + (J – 1)]
  - Row – major order:
    LOC(A[J, K]) = BASE(A) + w[(K – 1) +N(J – 1)]
- Where Array size = (M,N); M= number of rows; N= number of columns;

| Asymptotic form | Relationship | Definition |
|---|---|---|
| F(n)∈ θ(g(n)) | F(n) ≡ g(n) | $0 < \lim_{n\to\infty} \frac{fn}{gn} < \infty$ |
| F(n)∈ 0(g(n)) | F(n) ≤ g(n) | $0 \le \lim_{n\to\infty} \frac{fn}{gn} < \infty$ |
| F(n)∈ Ω(g(n)) | F(n) ≥ g(n) | $0 \le \lim_{n\to\infty} \frac{fn}{gn}$ |
| F(n)∈o(g(n)) | F(n)<g(n) | $\lim_{n\to\infty} \frac{fn}{gn} = 0$ |
| F(n)∈ ω(g(n)) | F(n)>g(n) | $\lim_{n\to\infty} \frac{fn}{gn} = \infty$ |

- o(g(n)) ∧ω(g(n)) is the empty set. ((a < b) ∩ (a > b))
- For any real constant a and b, where b>0
  - $(n + a)^b = \theta(n^b)$

- **Complexity ordering:**
  - $O(1) < O(\log 2n) < O(n) < O(n^2) < \ldots < O(n^k) < O(2^n)$
- So space requirement for an inplace sort is O(1).
- Bubble sort:
  - We need to make a maximum of (n-1) passes for n-input elements.
  - $\begin{pmatrix} \text{Total number of comparisons} \\ \text{in bubble sort after K} - \text{iteration} \\ \text{of outer loop} \end{pmatrix} = \frac{2kn - k^2 - k}{2},$
  - The average number of iterations of outer loop = O(n)
  - Total number of comparisons required = $\frac{n^2}{2}$
  - Total number of swapping required for arranging n elements in the sorted order by using bubble sort = $\frac{3n^2}{4}$
  - And hence the average time complexity = $O(n^2)$
  - Space complexity of bubble sort = O(1)
  - The number of interchanges an never be greater than the number of comparisons.
  - If the file is already sorted then the time taken by bubble sort is order of O(n), as it will require only one pass of (n-1) comparisons.
- **Quick sort:**
  - Best and average case time complexity = O(n logn)
  - Worst case time complexity = $O(n^2)$
  - Worst case space complexity = O(n)
- **Insertion Sort:**
  - Best case:
    Time complexity = O(n)
  - Average and Worst case:
    Time complexity = $O(n^2)$
  - Total number of comparisons are = $\sum_{j=2}^{n-1} \frac{j+1}{2} = \frac{n^2 + n - 2}{4}$
- **Selection Sort:**
  - Best, average, and worst case time complexity = $O(n^2)$
  - Worst case space complexity = O(n) {total} and O(1) {auxiliary}
  - Selection sort has the minimum number of swaps.
- **SHELL sort:**
  - It is also known as diminishing increment sort.
  - Time complexity = $O(n(\log n)^2)$
  - If n is an exact power of 2, then average running time is $O\left(n^{\frac{3}{2}}\right)$

- **Merge Sort:**
  - Time Complexity = O(n logn)
  - Space complexity of Merge sort = O(log n)
- **Binary Search:**
  - Maximum number of comparisons is approximately $\log_2 n$
- **Interpolation search:**
  - Time complexity = log (log n)
  - Worst case behaves like linear search = O(n)
- Robust interpolation search worst complexity = $(\log n)^2$
- **Hashing:**
  - If n: no. of elements; m: no. of slots, and $\alpha = \frac{n}{m}$

    Resolve by chaining unsuccessful search takes $\theta(1+\alpha)$

    Successful search takes $\theta\left(2+\frac{\alpha}{2}+\frac{\alpha}{2n}\right)$
- **Double hashing:**
  - Here we use two different hash functions as

    h(k,i) = (h1(k) + ih2(k)) mod m

    Initial probe is to position T(h1(k))
  - Here $\theta(m)^2$probe sequences are used.
- In an open addressing scheme, the expected number of probes in an unsuccessful search is at most $\frac{1}{1-\alpha}$ assuming uniform hashing.
  - And expected number of probes in a successful search is at most $\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$
- The total number of nodes in a complete binary tree = $2^{d+1}$ - 1 ; where d is the depth.
  - Number of leaf nodes = $2^d$
  - Number of non-leaf nodes = $2^d - 1$
- Time complexity of matrix chain multiplication = $O(n^3)$

- **Balance tree:**
  - AVL tree
  - B-Tree.
- **Kruskal's Algorithm:**
  - The total time taken by this algorithm to find the minimum spanning tree is $O(E \log_2 E)$
  - But the time complexity, if edges are not sorted is $O(E \log_2 V)$
  - Where E: number of edges; V: number of vertices.
- **Prim's Algorithm:**
  - The time complexity for Prims algorithm is $O(E \log_2 V)$

- The depth of a complete binary tree with n nodes:
  - Depth = $\log_2 n + 1$
- The worst case height of AVL tree with n nodes = 1.44 log n
- Number of trees (not only binary trees) possible with n- nodes = $2^n - n$
- For a complete n-ary tree having either 0 or 1 sons. If k is the number of internal nodes of a complete n-ary tree, then the number of leaves in it is given by = k(n – 1) + 1
- Heapify() call takes O(log n) times.
- **Heap Sort:**
  - The running time is O(n log n)
  - A tighter bound is θ(n)
- Number of different unlabelled binary trees possible with n-nodes = $\frac{1}{n+1}\binom{2n}{n}$
- Number of different labelled binary tree possible with n-nodes = $\frac{1}{n+1}\binom{2n}{n}(n!)$
- The height of a red black tree with n-internal nodes is = $(2 \log_2 n - 1)$
- Suppose we use a hash function h to hash n distinct keys into a array T of length m. Assuming simple uniform hashing:
  - The expected number of collisions = $\frac{n(n+1)}{2m}$
- Number of moves required to solve the Tower of Hanoi Problem = $2^n - 1$ ; where n is the number of disks.
- **Dijkstra's algorithm:**
  - With V vertices and E edges, Time complexity = 0(E + V log V)
- **Bellman Ford Algorithm:**
  - With V vertices and E edges; Time complexity = O(EV)
- Let T(l) and T(r) denote the left and right subtrees of a binary tree T. If T has n nodes, then the
  - Total Path Length, P(T) = P( l )+P( r ) + n-1 ; where P(l) is the path length of left subtree and P(r) is the path length of right subtree
- Different implementation of Hashing and their time complexity:

| Implementation | Insert | Search |
|---|---|---|
| Direct addressing | O(1) | O(1) |
| Ordered addressing | O(N) | O(log N) |
| Ordered List | O(N) | O(N) |
| Unordered array | O(1) | O(N) |
| Unordered List | O(1) | O(N) |

- Chromatic number of a bipartite graph = 2
- Chromatic number of a planar graph ≤ 4
- We can find the colouring of n vertices in $O(2^n)$
- Time complexity of DFS with V vertices and E edges = O(V + E )
- Time complexity of BFS with V vertices and E edges = O(V + E )
- Total number of BST(binary search trees) with n distinct keys = $\frac{1}{n+1}\binom{2n}{n}$
- In complete k-ary tree, every internal node has exactly k children. The number of leaves in such a tree with n internal nodes is = n( k − 1 ) + 1
- For a hash table of size N,
  - Probability of inserting K integers with no collision = $e^{\frac{-K(K-1)}{2N}}$
  - Thus probability of collision for inserting K integers = $1-e^{\frac{-K(K-1)}{2N}} \approx \frac{K(K-1)}{2N} \approx \frac{K^2}{2N}$
- Total number of stack permutations of a n distinct element is = $\frac{1}{n+1}\binom{2n}{n}$
- Total number of invalid permutations = $n!-\frac{1}{n+1}\binom{2n}{n}$
- Time complexity of Bankers algorithm = O ($mn^2$); where m= number of resources and n= number of processes.
- An n-dimensional hypercube ($Q_n$) is simple undirected graph with $2^n$vertices. The vertices of ($Q_n$) are bit-strings of length n. Two vertices of ($Q_n$) are connected by an edge only if u & v differs in exactly one bit position.
  - ($Q_n$) is Regular bipartite graph.
  - Degree of each vertex =2 => is bipartite.
  - Number of edges = $n2^{n-1}$
- Floyd Warshall algorithm has time complexity = $O(V^3)$
- Fibonacci series recurrence relation regardless of any initial condition is
  - $T(n) = T(n-1) + T(n-2) + n = 2^n + n2^n$

  Best case = $\Omega(2^n)$

  Worst cast = $O(n2^n)$
- Fibonacci series recurrence relation regardless of any initial condition is
  - T(n) = T(n − 1) + (n − 2) + K(constant)

  Time complexity in all cases = $O(2^n)$

**MASTER METHOD**

- It is applicable to only limited classes of recurrences which are in the form of,

$T(n) = aT\left(\frac{a}{b}\right) + fn$; and a >= 1 and b > 1 fn is asymptotically positive. {that is, fn> 0 for n >= n0}

compare fn with $n^{\log_b a}$

**case 1:** If $fn = O(n^{\log_b a-\varepsilon})$, for some $\varepsilon = 0$

$Tn = \theta(n^{\log_b a})$

**case 2:** $fn = \theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$

$tn = \theta(n^{\log_b a} \log^{k+1} n)$,

**Case 3:** If $fn = \Omega(n^{\log_b a+\varepsilon})$, for some $\varepsilon > 0$ and $af\frac{n}{b} \leq (1-\varepsilon')fn$

$Tn = \theta(fn)$

• For example, Find the time complexity of the following recurrence relation:

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

$n^{\log_2 2} = n$; so $fn = n \log n = \theta(n\log n)$

Hence by case 2 of master theorem,

$tn = \theta(n^{\log_b a} \log^{k+1} n) = \theta(n\log^{1+1} n) = \theta(n\log^2 n)$

• Finding complexity by Substitution method for the following recurrence relation:

➢ $Tn=T(\sqrt{n})+C1$ (const).

Let $n = 2^m$

$So\, T(2^m) = T\left(2^{\frac{m}{2}}\right) + C1$

Let $T(2^m) = S(m)$

$S(m) = S\left(\frac{m}{2}\right) + C1 = O(\log m) = O(\log\log n)$

# SUBJECT 6 : DATABASES

**Cross Product:**

- If n1 tuples in R and if n2 tuples in S
- Then, in P = R × S, (n1 × n2) tuples will be present

**Database Languages:**

- **Data Definition Language:** Create, Alter, Drop, Truncate, Rename, Comment
- **Data Manipulation Language:** Select, Insert, Update, Delete, Merge, Call, Explain Plan, Lock Table
- **Data Control Language:** Grant, Revoke

**Functional dependency:**

- **Trivial Functional Dependencies:** $A \rightarrow B$ is said to be trivial if and only if $B \subseteq A$
- **Non- Trivial Functional Dependencies**: $A \rightarrow B$ has a non-trivial functional dependency if B is not a subset of A.

**Armstrong Rules over FD's:**

- **Reflexivity :** $x \rightarrow x$ is always true
- **Transitivity:**if $X \rightarrow Y$ & $y \rightarrow z$ then $X \rightarrow Z$
- **Augmentation:** if $X \rightarrow Y$ then $XZ \rightarrow YZ$
- **Split Rule:** if $X \rightarrow YZ$ then $X \rightarrow Y$, $X \rightarrow Z$
- **Merge/Union Rule:**if $X \rightarrow Y$, $X \rightarrow Z$ then $X \rightarrow YZ$

**Pseudo transitive Rule:**

if $wx \rightarrow y$ and $XY \rightarrow Z$ then $wx \rightarrow z$

$wx \rightarrow y \Rightarrow wx \rightarrow xy$. (Augmentation)

$wx \rightarrow xy \quad xy \rightarrow z \Rightarrow wx \rightarrow Z$ (transitive)

- **Candidate Key** = Minimal Super key
- No. of possible Super keys in R with n attributes = $2^{n-1}$ , if each attribute of R is a candidate key.
- If all attributes form one candidate key, then number of super keys of R=1

**NORMALIZATION:**

**Equality of FD set:** Let F and E are two FD sets for a relation R.

- If all FDs of F can be derived from FDs present in E, we can say that $E \supset F$.
- If all FDs of E can be derived from FDs present in F, we can say that $F \supset E$.
- If both the above points are true, F=E.
  - **Lossless Join Decomposition**: if $[R_1 \bowtie R_2 \bowtie \ldots\ldots \bowtie R_n] = r$
  - **Lossy Join Decomposition:** if $[R_1 \bowtie R_2 \bowtie \ldots\ldots \bowtie R_n] \supset r$

**Dependency Preserving Decomposition:**

Relational scheme R with Functional Dependency set F decomposed into sub relation $R_1$, $R_2$, $R_3$ … $R_n$ with sets $F_1$, $F_2$ … $F_n$.

$$F_1 \cup F_2 \cup F_3 \ldots F_n \subseteq F$$

if $[F_1 \cup F_2 \cup F_3 \ldots F_n] = F$ (dependency preserving)

if $[F_1 \cup F_2 \cup F_3 \ldots F_n] \subset F$ (not dependency pressing)

**Normal Forms:**

- 1NF: contains atomic values, cannot hold multiple values, by default every relation is in 1NF
- 2NF: R is in 1 NF, R does not contain any partial dependency

Y→Z is said to be partial dependency iff,

- Y is proper subset of candidate key, or
- Y is a prime attribute
- There is no condition on non-prime →non-prime

**Testing for 2 NF:**

$$\left[\text{Proper Subset of candidate key}\right]^{+} = \{\text{Only Prime}\}$$

- **3NF:** R is in 2 NF
  - X must be a super key of R, or
  - Y must be a prime attribute of R
  - No transitive dependency exists for non-prime attributes.

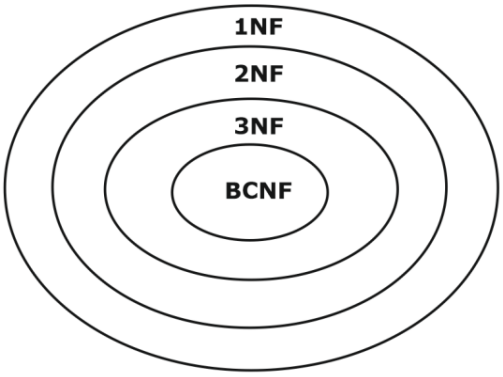
A → B is called a transitive dependency if and only if:

**a.** A is not a super key.

**b.** B is a non-prime attribute.

If any one condition fails, then it is not a transitive dependency

**Boyce Codd Normal Form (BCNF):** A relational schema R is in BCNF iff every non-trivial functional dependency X→Y is in relation R with determinant X must be a candidate key/super key




- Non-prime attributes transitively determined by super key is not allowed in 3 NF.
- Prime attributes transitively determined by super key is allowed in 3 NF but not allowed in BCNF.

- If every candidate key of relation R is simple candidate key, then relation R always in 3 NF.
- If every attribute of relation R is prime attribute, then relation R always in 3 NF but may not be in BCNF.
- If relation R with no non-trivial functional dependency, then R always in BCNF.

**RELATIONAL CALCULUS:**

**Relational Algebra:**

- **Basic Operators:** Selection operator (σ), Projection Operator (∏), Rename(ρ), Cross Product(X), Union (U), Set Difference (-)
- **Derived Operators:**
  - Intersection (∩) {derived using set difference}
  - Join (⋈) {derived using cross product, selection and projection}
  - Division Operator (÷) {derived using selection, set difference and cross product}
- Division Expansion:
  $\Pi_{sidcid}(E) / \Pi_{eid}(C) = \Pi_{sid}(E) - \Pi_{sid}((\Pi_{sid}(E) \times \Pi_{cid}(C) - \Pi_{sidcid}(E))$
- If n1 tuples in R and if n2 tuples in S
  Then, in P= R X S , (n1 X n2 ) tuples will be present

| Operation | Relational Algebra | Tuple Relational Calculas |
|---|---|---|
| Selection | $\sigma_{cond}(R)$ | {T\|R(T) AND Cond (T)} |
| Projection | $\Pi A_1, A_2, \ldots A_k(R)$ | $\{T \mid T.A_1, T.A_2, \ldots, T.A_k \mid R(T)\}$ |
| Cartesian Product | R × S | $\{T \mid \exists T_1 \in R, \exists T_2 \in R(T.A_1 = T_1.A_1, \text{ AND}, \ldots, T.A_n = T_1.A_n \text{ AND } T.B_1 = T_2.B_1 \text{ AND}, \ldots, T.B_m = T_2.B_m)]$ |
| Union | R ∪ S | {T\|R(T) AND S(T)} |
| Set Difference | R – S | {T\|R(T) AND ∀ T1 ∈ S, (T1 <> T)} where T <> $T_1$ is $T.A_1 <> T_1A_1$ OR …. $T.A_n <> T_1.A_n$ |

**SQL Constraints:**

- **NOT NULL**: Not Null constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.
- **UNIQUE Constraint:** UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data.
- **Primary Key Constraint:** Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value.
- **Foreign Key Constraint:** FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables.

- **Check constraint:** The Check constraint is used to limit the value range that can be placed in a column.
- **Default Constraint:** The Default constraint is used to provide a default value for a column.

**INDEXING:**

➢ File organisation:

- **Block Factor:**

$$\text{Block factor}=\left\lfloor \frac{\text{Block size}}{\text{Record size}} \right\rfloor \text{records/block}$$

$$=\left\lfloor \frac{\text{Block size}-\text{Block Header}}{\text{Record size}} \right\rfloor$$

- **Block factor of Index File:**

$$\text{Block factor of index file}=\left\lfloor \frac{\text{Block size}-\text{Block Header}}{\text{Search Key}+\text{Pointer}} \right\rfloor \text{Entries/block}$$

**B & B⁺ TREE:**

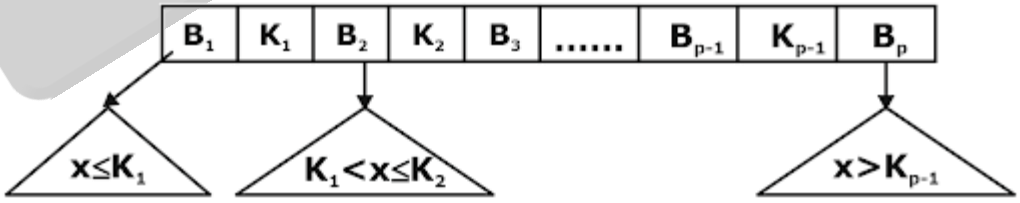
➢ **B⁺ Tree:**

(a) Leaf Node:

It consists of set of (Key, record pointer) pairs and one block pointer which points to next leaf. node. It contains atleast $\left\lceil \frac{p}{2} \right\rceil - 1$ keys and atmost P-1 keys


(b) Internal Node:

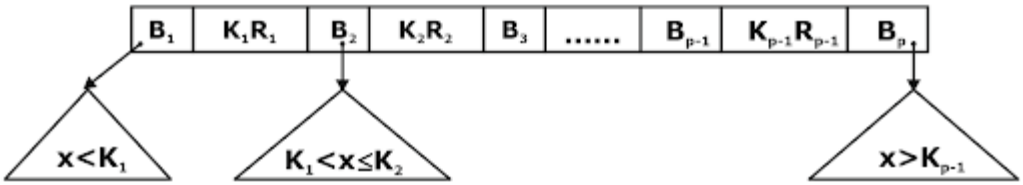
It contains atleast $\left\lceil \frac{p}{2} \right\rceil$ block pointers & $\left\lceil \frac{p}{2} \right\rceil - 1$ keys and at most p pointers and (p – 1) keys.




**Block size=** p(size of block pointer) +(p-1)*(size of key field + size of record pointer)

➢ **B Tree:**

- Each node has at most p tree pointers.
- Each node, except the root leaf nodes, has at least ⌈p/2⌉ tree pointers. The root node has at least two tree pointers unless it is the only node in the tree.
- A node with q tree pointers, q ≤ p, has q – 1 search key field values (and hence has q – 1 data pointers).




- Order of non- leaf node in B Tree:

  P* Bp + (P-1)[Search Key + Rp] ≤ Block Size

  Where, P is the maximum possible Block pointers of B tree node i.e. order P

  Bp is the block pointer

  Rp is the record pointer

- Order of Leaf node in B Tree:

$(P_{leaf}-1)$*(size of key field + size of record pointer) + P*(size of block pointer)≤ Block Size

## SUBJECT 7 : OPERATING SYSTEMS

- Next CPU burst predicted is given by:
  - $Tn = \alpha \times tn + (1 - \alpha)T(n - 1)$; $0 \leq \alpha \leq 1$ ; and tn is current CPU burst, $T(n - 1)$ is the past predicted value, and Tn is the new predicted value.
- For n fork() calls $2^n - 1$ child processes will be created.
- Turn Around time= Completion time- Arrival time
- Waiting time= Turnaround time – Burst time
- Throughput= number of process/ Maximum (completion time)- minimum(completion time)
- Response ratio= (waiting time + burst time) / Burst time
- Round Robin Algorithm:
  - If there are n processes in ready queue and time quantum is q, then each process gets $\frac{1}{n}$ of the CPU time in chunks of at most q time units.
  - Each process must wait no longer than (n-1)q time units until next time quantum.
  - Turnaround time also depends on the size of the time quantum.
- If waiting time or fraction of each process is P and n is the number of processes, then
  - CPU utilization = $1 - P^n$
  - And probability that N processes will all wait at the same time = $P^N$

**For Main Memory-**

- Physical Address Space = Size of main memory
- Size of main memory = Total number of frames x Page size
- Frame size = Page size
- If number of frames in main memory = $2^X$, then number of bits in frame number = X bits
- If Page size = $2^X$ Bytes, then number of bits in page offset = X bits
- If size of main memory = $2^X$ Bytes, then number of bits in physical address = X bits

**For Process-**

- Virtual Address Space = Size of process
- Number of pages the process is divided = Process size / Page size
- If process size = $2^X$ bytes, then number of bits in virtual address space = X bits

**For Page Table-**

- Size of page table = Number of entries in page table x Page table entry size
- Number of entries in pages table = Number of pages the process is divided
- Page table entry size = Number of bits in frame number + Number of bits used for optional fields if any

**NOTE-**

- In general, if the given address consists of 'n' bits, then using 'n' bits, $2^n$ locations are possible.
- Then, size of memory = $2^n$ x Size of one location.
- If the memory is byte-addressable, then size of one location = 1 byte.
- Thus, size of memory = $2^n$ bytes.
- If the memory is word-addressable where 1 word = m bytes, then size of one location = m bytes.
- Thus, size of memory = $2^n$ x m bytes.

**Effective Access Time =**

**Hit ratio of TLB × {Access time of TLB + Access time of main memory}**
**+**
**Miss ratio of TLB × {Access time of TLB + (L + 1) × Access time of main memory}**

**where L = Number of levels of page table**

- Unix Inode:
  - Suppose there are 12 direct blocks and 1 indirect block and 1 double indirect block and 1 triple indirect block then the maximum size of process supported by Inode is

$$= \left[12 + \frac{BS}{ES} + \left(\frac{BS}{ES}\right)^2 + \left(\frac{BS}{ES}\right)^3\right] \times BS$$ ; Where BS = block size; ES = entry size (or block pointer size).

## SUBJECT 8 : DIGITAL LOGIC

| Gate | Behaviour | Expression |
|---|---|---|
| NOT | Inverse the input | $\bar{A}$ |
| AND | O/p is 1 if both inputs are 1 else 0 | A.B |
| OR | O/p is 0 if both inputs are 0 else 1 | A+B |
| NAND | Inverse the output of AND gate | $(AB)'$ or $A \uparrow B$ |
| NOR | Inverse the output of OR gate | AB or $A \downarrow B$ |
| EX-OR | Outputs 1 If two Inputs are different else 0 | $A\bar{B} + \bar{B}A$ or $A \oplus B$ |
| EX-NOR | Outputs 0 if two inputs are different else 1 | $AB + \overline{AB}$ or $A \odot B$ |

- **Note:** NAND and NOR gates are called UNIVERSAL GATES because all other gates can be constructed from any of them.

Number of NAND or NOR gate required to implement other GATE:

| GATE to be implemented | NOR GATE required | NAND GATE required |
|---|---|---|
| NOT | 1 | 1 |
| AND | 3 | 2 |
| OR | 2 | 3 |
| EX-OR | 5 | 4 |
| EX-NOR | 4 | 5 |

**Combinational circuits:** In combinational circuits, output depends on present input only; it does not require any feedback and memory.

| Circuit Type | Expression | NAND/NOR gates required |
|---|---|---|
| Half Adder | Sum = $A \oplus B$<br>Carry = AB | 5 |
| Full Adder | Sum = $A \oplus B \oplus C$<br>Carry = AB + BC + CA | 9 |
| Half Subtractor | Diff = $A \oplus B$<br>Borrow = ÁB | 5 |
| Full Subtractor | Diff = $A \oplus B \oplus C$<br>Borrow = ÁB+ÁC+BC | 9 |

**Multiplexer:**

It selects the input from one of many input lines and sends it single output line. The input line chosen from output is based upon set of selection lines. For $2^n$ input lines, there will be n select lines and 1 output lines.

**Note:** No. of mux required to implement n x 1 mux using m x 1 mux is ceil(n-1 / m-1)A⊕B.
No. of mux required to implement 16 x 1 mux using 4 x 1 mux is ceil(15/3)=5.

**Demultiplexer:** It selects the input from one input line and sends it to one out of many output lines. The output line chosen is based upon set of selection lines. For 1 input lines, there will be n select lines and $2^n$ output lines.

**Note:** No. of mux required to implement 1 x n mux using 1 x m mux is ceil(n-1 / m-1)A⊕B.
No. of mux required to implement 1 x 16 mux using 1 x 4 mux is ceil(15/3)=5.

**Encoder:**

For $2^n$ input lines, it has n output. It can be used to convert octal or hexadecimal to binary data.

**Decoder:**

For n input lines, it has either $2^n$ outputs or less then that. It can be used to convert binary data to other codes like octal or hexadecimal.

**Code Converter:** Code converters are used to convert one type of code to others.

- **Binary to Gray Converter**

  **Function:**

  4 bit Input($B_3B_2B_1B_0$) with $B_3$ as MSB and 4 bit output($G_3G_2G_1G_0$) with $G_3$ as MSB, O/P is

  $G_3=B_3$

  $G_2=B_3 \oplus B_2$

  $G_1=B_2 \oplus B_1$

  $G_0=B_1 \oplus B_0$

- **Gray to Binary Converter**

  **Function:**

  4 bit Input($G_3G_2G_1G_0$) with $G_3$ as MSB and 4 bit output($B_3B_2B_1B_0$) with $B_3$ as MSB, O/P is

  $B_3=G_3$

  $B_2=B_3 \oplus G_2$

  $B_1=B_2 \oplus G_1$

  $B_0=B_1 \oplus G_0$

**Flip flops:**

Flip-Flop are sequential circuits where O/P depends on present input as well as previous output.

**S-R Flip-Flops:**

| Clock | S | R | $Q_{n+1}$ |
|---|---|---|---|
| 0 | × | × | $Q_n$ |
| 1 | 0 | 0 | $Q_n$ |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | Invalid |

Characteristics equation: $Q_n = S + R'Q_n$

**J-K Flip-Flops:**

| Clock | J | K | $Q_{n+1}$ |
|---|---|---|---|
| 0 | × | × | $Q_n$ |
| 1 | 0 | 0 | $Q_n$ |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | $\overline{Q_n}$ |

Characteristics equation: $Q_n = J\,Q'_n + R'Q_n$

**D Flip-Flops:**

| Clock | D | $Q_{n+1}$ |
|---|---|---|
| 0 | × | $Q_n$ |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Characteristics equation: $Q_{n+1} = D$

**T Flip-Flops:**

| Clock | D | $Q_{n+1}$ |
|---|---|---|
| 0 | × | $Q_n$ |
| 1 | 0 | Qn |
| 1 | 1 | $\overline{Q_n}$ |

Characteristics equation: $Q_{n+1} = T\,Q'_n + T'\,Q_n$.

**Counters:**

is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.

**Counters are basically divided into two types:**

- ✓ **Asynchronous counter:** In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following counters is driven by output of previous flip flops.
- ✓ **Synchronous counter:** Unlike the asynchronous counter, synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop.

**Important point:** Number of flip flops used in counter are always greater than equal to ($\log_2 n$) where n=number of states in counter.

**Number System**

- **R's complement**:
  - ➢ N: a +'ve number in base r with an integer part of n digits, then

R's complement of N = $r^n - N$ for $N \neq 0$ and 0 for N = 0;

- **R-1 complement:**
  - ➢ N: a +'ve number in base r with an integer part of n digits and a fraction part of m digits, then

  (R – 1)'s complement of N = $r^n - r^{-m} - N$;
- **Operator Precedence:**
  - ➢ () > NOT > AND > OR
- N bit Binary parallel adder:
  - ➢ Total delay = (2N + 1)tp ; where tp= propagation delay of each gate.
- **N bit Look ahead Adder:**
  - ➢ Total delay = 4tp ; where tp= propagation delay of each gate.
- If the n-bit decoded information has unused or don't care combinations, the decoded output will have less than $2^n$ outputs.
  - ➢ The decoders is also called n-to-m line decoder where $m \leq 2^n$
- **Sequential Circuits:**
  - ➢ Duty cycle = $\frac{1}{\text{Clock period}}$
- **With N-flip flops,**
  - ➢ Ring counter can have N modulus counter
  - ➢ Johnson counter can have 2N modulus counter

- **Integer representation:**
  - Signed Magnitude representation:
    $-(2^{k-1}-1)$ to $+(2^{k-1}-1)$
  - 1's complement representation:
    $-(2^{k-1}-1)$ to $+(2^{k-1}-1)$
  - 2's complement representation:
    $-(2^{k-1})$ to $+(2^{k-1}-1)$
- Duty cycle: A duty cycle is the percentage of one-period in which a signal is active.
  - So duty cycle, $D=\left(\frac{T}{P}\times 100\right)\%$
- Time period$=\frac{1}{\text{frequency}}$

# SUBJECT 9 : ENGINEERING MATHEMATICS

## Matrix Algebra

### Unit matrices

The unit matrix I of order n is a square matrix with all diagonal elements equal to one and all off-diagonal elements zero, i.e., $(I)_{ij} = \delta_{ij}$. If A is a square matrix of order n, then AI = IA = A. Also $I = I^{-1}$.

I is sometimes written as $1_n$ if the order needs to be stated explicitly.

### Products

If A is a (n × l) matrix and B is a (l × m) then the product AB is defined by

$$(AB)_{ij} = \sum_{k=1}^{l} A_{ik}B_{kj}$$

In general AB ≠ BA.

### Transpose matrices

If A is a matrix, then transpose matrix $A^T$ is such that $(A^T)_{ij} = (A)_{ji}$.

### Inverse matrices

If A is square matrix with non-zero determinant, then its inverse $A^{-1}$ is such that $AA^{-1} = A^{-1}A=I$.

$$(A^{-1})_{ij} = \frac{\text{transpose of cofactor of } A_{ij}}{|A|}$$

where the cofactor of $A_{ij}$ is $(-1)^{i+j}$ times the determinant of the matrix A with j-th row and i-th column delete.

### Determinants

If A is a square matrix then the determinant of A, |A| (≡ det A) is defined by

$$|A| = \sum_{i,j,k\ldots} \in_{ijk\ldots} A_{1i}A_{2j}A_{3k}\ldots$$

where the number of the suffixes is equal to the order of the matrix.

#### 2 × 2 matrices

If $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ then,

$$|A| = ad - bc \quad A^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \quad A^{-1} = \frac{1}{|A|}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

**Product rules**

$(AB... N)^T = T^N ... B^T A^T$

$(AB ...N)^{-1} = N^{-1} ...B^{-1}A^{-1}$ (if individual inverses exist)

$|AB...N| = |A||B|...|N|$ (if individual matrices are square)

**Orthogonal matrices**

An orthogonal matrix Q is a square matrix whose columns $q_i$ form a set of orthonormal vectors.

For any orthogonal matrix Q,

$Q^{-1} = Q^T$, $|Q| = \pm 1$, $Q^T$ is also orthogonal.

**Solving sets of linear simultaneous equations**

If A is square then Ax = b has a unique solution $x = A^{-1}b$ if $A^{-1}$ exists, i.e., if $|A| \neq 0$.

If A is square then Ax = 0 has a non-trivial solution if and only if |A| = 0.

An over-constrained set of equations Ax = b is one in which A has m rows and n columns, where m (the number of equations) is greater than n (the number of variables). The best solution x (in the sense that it minimizes the error |Ax – b|) is the solution of the n equations $A^TAx = A^Tb$. If the columns of A are orthonormal vectors then $x = A^Tb$.

**Hermitian matrices**

The Hermitian conjugate of A is $A^\dagger = (A^*)^T$, where A* is a matrix each of whose components is the complex conjugate of the corresponding components of A. If $A = A^\dagger$ then A is called a Hermitian matrix.

**Eigenvalues and eigenvectors**

The n eigenvalues $\lambda_i$ and eigenvectors $u_i$ of an n × n matrix A are the solutions of the equation Au = Au. The eigenvalues are the zeros of the polynomial of degree n, $P_n(\lambda) = |A - AI|$. If A is Hermitian then the eigenvalues $\lambda_i$ are real and the eigenvectors $u_i$ are mutually orthogonal. |A – λI} = 0 is called the characteristic equation of the matrix A.

$$\text{Tr } A = \sum_i \lambda_i, \quad \text{also } |A| = \prod_i \lambda_i$$

If S is a symmetric matrix, ∧ is the diagonal matrix whose diagonal elements are the eigenvalues of S, and U is the matrix whose columns are the normalized eigenvectors of A, then UTSU = A and S = UA UT. If x is an approximation to an eigenvector of A then

$U^TSU = \wedge$ and $S = U\wedge U^T$.

If x is an approximation to an eigenvector of A then $x^T Ax/(x^Tx)$ (Rayleigh's quotient) is an approximation to the corresponding eigenvalue.

**Limits**

$n^c x^n \to 0$ as $n \to \infty$ if $|x| < 1$ (any fixed c)

$x^n/n! \to 0$ as $n \to \infty$ (any fixed x)

$(1 + x/n)^n \to e^x$ as $n \to \infty$, $x \ln x \to 0$ as $x \to 0$

If f(a) = g(a) = 0 then $\lim_{x\to a}\frac{f(x)}{g(x)} = \frac{f'(a)}{g'(a)}$ (l'Hôpital's rule)

## Differentiation

$(uv)' = u'v + uv'$, $\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$

$(uv)^{(n)} = u^{(n)}v + nu^{(n-1)}v^{(1)} + \cdots + {}^nC_r u^{(n-r)}v^{(r)} + \cdots + uv^{(n)}$ Leibniz Theorem

where ${}^nC_r \equiv \binom{n}{r} = \frac{n!}{r!(n-r)!}$

$\frac{d}{dx}(\sin x) = \cos x$ $\quad$ $\frac{d}{dx}(\sinh x) = \cosh x$

$\frac{d}{dx}(\cos x) = -\sin x$ $\quad$ $\frac{d}{dx}(\cosh x) = \sinh x$

$\frac{d}{dx}(\tan x) = \sec^2 x$ $\quad$ $\frac{d}{dx}(\tanh x) = \operatorname{sech}^2 x$

$\frac{d}{dx}(\sec x) = \sec x \tan x$ $\quad$ $\frac{d}{dx}(\operatorname{sech} x) = -\operatorname{sech} x \tanh x$

$\frac{d}{dx}(\cot x) = -\operatorname{cosec}^2 x$ $\quad$ $\frac{d}{dx}(\coth x) = -\operatorname{cosech}^2 x$

$\frac{d}{dx}(\operatorname{cosec} x) = -\operatorname{cosec} x \cot x$ $\quad$ $\frac{d}{dx}(\operatorname{cosech} x) = -\operatorname{cosech} x \coth x$

## Integration

Standard forms

$\int x^n dx = \frac{x^{n+1}}{n+1} + c$ for $n \neq -1$

$\int \frac{1}{x} dx = \ln x + c$ $\quad$ $\int \ln x \, dx = x(\ln x - 1) + c$

$\int e^{ax} dx = \frac{1}{a} e^{ax} + c$ $\quad$ $\int x\, e^{ax} dx = e^{ax}\left(\frac{x}{a} - \frac{1}{a^2}\right) + c$

$\int x \ln x dx = \frac{x^2}{2}\left(\ln x - \frac{1}{2}\right) + c$

$\int \frac{1}{a^2 + x^2} dx = \frac{1}{a}\tan^{-1}\left(\frac{x}{a}\right) + c$

$\int \frac{1}{a^2 - x^2} dx = \frac{1}{a}\tanh^{-1}\left(\frac{x}{a}\right) + c = \frac{1}{2a}\ln\left(\frac{a+x}{a-x}\right) + c$ for $x^2 < a^2$

$\int \frac{1}{x^2 - a^2} dx = -\frac{1}{a}\coth^{-1}\left(\frac{x}{a}\right) + c = \frac{1}{2a}\ln\left(\frac{x-a}{x+a}\right) + c$ for $x^2 > a^2$

$$\int \frac{x}{\left(x^2 \pm a^2\right)^n} dx = \frac{-1}{2(n-1)} \frac{1}{\left(x^2 \pm a^2\right)^{n-1}} + c \qquad \text{for } n \neq 1$$

$$\int \frac{x}{x^2 \pm a^2} dx = \frac{1}{2} \ln\left(x^2 \pm a^2\right) + c$$

$$\int \frac{1}{\sqrt{a^2 - x^2}} dx = \sin^{-1}\left(\frac{x}{a}\right) + c$$

$$\int \frac{1}{\sqrt{x^2 \pm a^2}} dx = \ln\left(x + \sqrt{x^2 \pm a^2}\right) + c$$

**Sets and Random Variables:**

**The Algebra of Sets**

$S \cup T = T \cup S$,

$S \cup (T \cup U) = (S \cup T) \cup U$,

$(S_c)_c = S$

$S \cup \Omega = \Omega$,

$S \cup (T \cap U) = (S \cup T) \cup U$,

$S \cap S^C = \emptyset$,

$S \cap \Omega = S$.

**Probability Axioms**

- **(Non negativity)** $P(A) \geq 0$, for every event A.
- **(Additive)** If A and B are two disjoint events, then the probability of their union satisfies $P(A \cup B) = P(A) + P(B)$.
  Furthermore, if the sample space has an infinite number of elements and $A_1$, $A_2$, ... is a sequence of disjoint events, then the probability of their union satisfies $P(A_1 \cup A_2 \cup \ldots) = P(A_1) + P(A_2) + \ldots$
- **(Normalization)** The probability of the entire sample space $= \Omega$ is equal to 1, that is, $P(\Omega) = 1$.

**Properties of Probability Laws**

Consider a probability law, and let A, B, and C be events.

(a) If $A \subset B$, then $P(A) \leq P(B)$.

(b) $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

(c) $P(A \cup B) \leq P(A) + P(B)$.

(d) $P(A \cup B \cup C) = P(A) + P(A^c \cap B) + P(A^c \cap B^c \cap C)$.

**TOTAL PROBABILITY THEOREM**

Let $A_1, \ldots, A_n$ be disjoint events that form a partition of the sample space (each possible outcome is included in one and only one of the events $A_1, \ldots, A_n$) and assume that $P(A_i) > 0$, for all $i = 1, \ldots, n$. Then, for any event B, we have

$P(B) = P(A_1 \cap B) + \ldots + P(A_n \cap B)$

$= P(A_1)P(B \mid A_1) + \ldots + P(A_n)P(B \mid A_n)$.

**Bayes' Rule**

Let $A_1, A_2, \ldots, A_n$ be disjoint events that form a partition of the sample space, and assume that $P(A_i) > 0$, for all i. Then, for any event B such that $P(B) > 0$, we have

$$P(A_i \mid B) = \frac{P(A_i)P(B / A_i)}{P(B)}$$

$$= \frac{P(A_i)P(B / A_i)}{P(A_1)P(B \mid A_1) + \ldots + P(A_n)P(B \mid A_n)}$$

**Probability Mass Function of Discrete Random Variable**

- $0 \le p_x(x_k) \le 1 \quad k = 1, 2, \ldots$
- $p_x(x) = 0$ if $x \# x_k$ $(k = 1, 2, \ldots)$
- $\sum_k p_x(x_k) = 1$

**Properties of CDF of Continuous Random Variable:**

- $F_x(-\infty) = 0$
- $F_x(\infty) = 1$
- $P(a < x \le b) = F_x(b) - F_x(a)$

**Probability Density Function of Continuous Random Variable**

- $f_x(x) \ge 0$
- $\int_{-\infty}^{\infty} f_x(x)dx = 1$
- $P(X \le x) = F_x(x) = \int_{-\infty}^{x} f_x(\lambda)d\lambda$
- $P(a < x \le b) = \int_a^b f_x(X)dx$

**Mean or Expected Value:**

Let a random variable X characterized by its PDF $f_x(x)$. The mean or expected value of X is defined as

$$E(X) = \overline{X} = \int_{-\infty}^{\infty} x f_x(x)dx$$

**Variance:**

The variance $\sigma_x^2$ of a random variable X is the second moment taken about its mean. i.e.

Var [X] = $\sigma_x^2$ = E [(X – $\mu_x$)$^2$]

$= \int_{-\infty}^{\infty} (x - \mu_x)^2 f_x(x)dx$

**Standard Deviation:**

The standard deviation $\sigma_x$ of a random variable is the square root of its variance, i.e.,

$$\sigma_x = \sqrt{var[x]} = \sqrt{X^2 = \mu_x^2}$$

**Covariance:**

The covariance of the random variables X and Y is defined as:

cov[XY] = $\sigma_{xy}$ = E [(X - $\mu_x$) (Y – $\mu_y$ )] $= \overline{(X - \mu_x)(Y - \mu_y)}$

where $\mu_x$ and $\mu_y$ are the mean of random variables X and Y, respectively. We may expand the above result as

cov[XY] = $\sigma_{xy}$ = E[XY] – $\mu_x\mu_y = \overline{XY} - \mu_x\mu_y$

**Correlation Coefficient:**

The correlation coefficient of random variables X and Y can he defined as

$$\rho_{xy} = \frac{cov[xy]}{\sigma_x\sigma_y}$$

where cov [XY] is the covariance of X and Y, and $\sigma_x$, $\sigma_y$ are the standard deviations of random variables.

**Uniform Random Variable**

$E[X] = \int_{-\infty}^{\infty} xfx(x)dx$

We have

$E[X] = \int_{-\infty}^{\infty} xf(x)dx$

$= \int_a^b x.\frac{1}{b-a}dx$

$= \frac{1}{b-a}.\frac{1}{2}x^2\Big|_a^b$

$= \frac{1}{b-a}.\frac{b^2-a^2}{2}$

$= \frac{a+b}{2}$,

Variance is obtained as

Var(X) = E[X$^2$] – (E[X])$^2$

$= \frac{a^2+ab-b^2}{3} - \frac{(a+b)^2}{4}$

$= \frac{(a+b)^2}{12}$

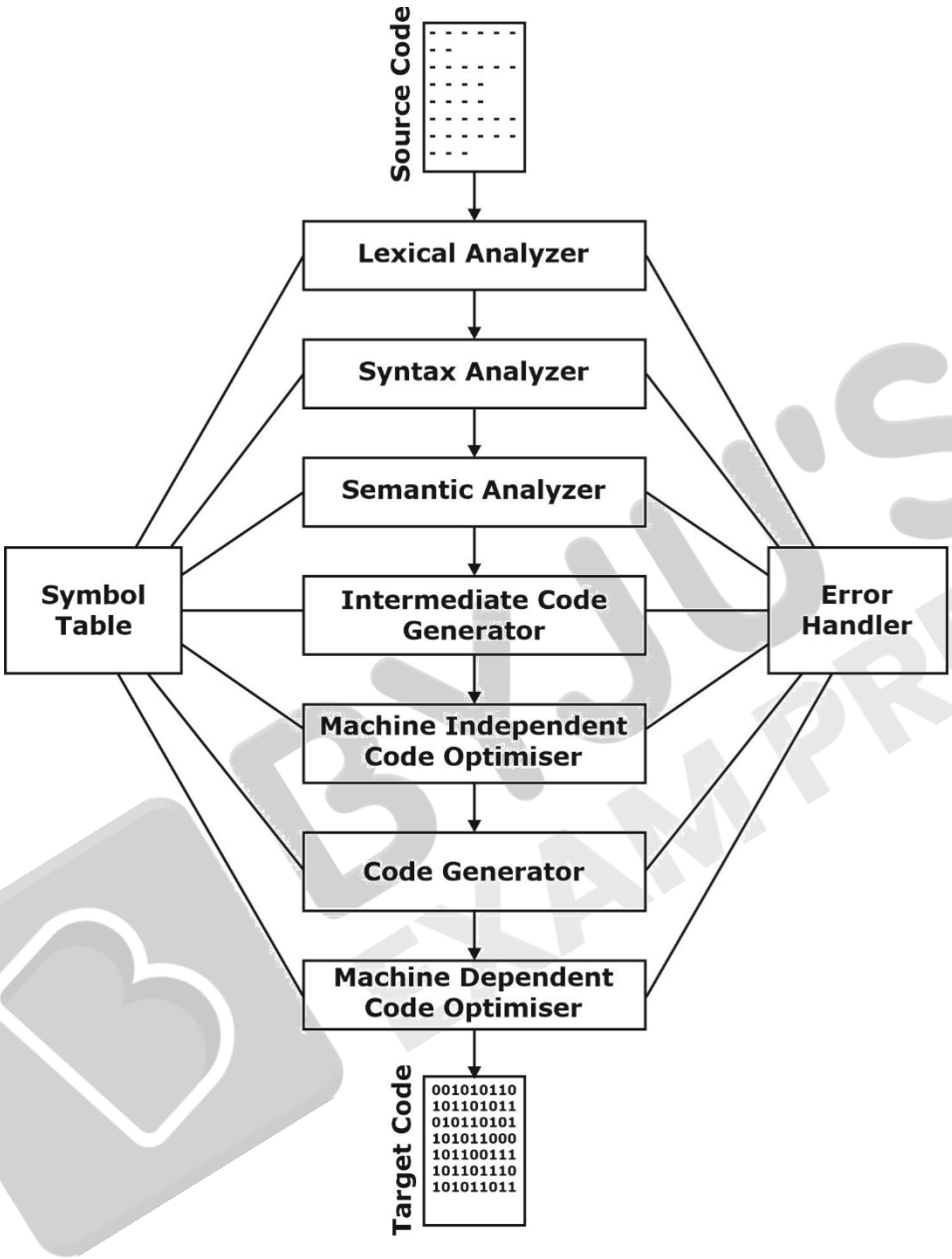
**Exponential Random Variable**

The mean and the variance can be calculated can be calculated to be

$E[X] = \frac{1}{\lambda}$, $var(X) = \frac{1}{\lambda^2}$

# SUBJECT 10 : COMPILER DESIGN

**Phases of compiler**




**Symbol Table**

It is a data structure being used and maintained by the compiler, consists all the identifier's name along with their types. It helps the compiler to function smoothly by finding the identifiers quickly.

1. **Lexical Analysis** : Lexical analyzer reads a source program character by character to produce tokens. Tokens can be identifiers, keywords, operators, separators etc.
2. **Syntax Analysis** : Syntax analyzer is also known as parser. It constructs the parse tree. It takes all the tokens one by one and uses Context Free Grammar to construct the parse tree.

3. **Semantic Analyzer** : It verifies the parse tree, whether it's meaningful or not. It furthermore produces a verified parse tree.
4. **Intermediate Code Generator** : It generates intermediate code, that is a form which can be readily executed by machine We have many popular intermediate codes.
5. **Code Optimizer** : It transforms the code so that it consumes fewer resources and produces more speed.
6. **Target Code Generator** : The main purpose of Target Code generator is to write a code that the machine can understand. The output is dependent on the type of assembler.

**Error Handling:**

The tasks of the Error Handling process are to detect each error, report it to the user, and then make some recover strategy and implement them to handle error. An Error is the blank entries in the symbol table. There are two types of error :

**Run-Time Error** : A run-time error is an error which takes place during the execution of a program, and usually happens because of adverse system parameters or invalid input data.

**Compile-Time Error**: Compile-time errors rises at compile time, before execution of the program.

1. **Lexical :**This includes misspellings of identifiers, keywords or operators.
2. **Syntactical :**missing semicolon or unbalanced parenthesis.
3. **Semantical :**incompatible value assignment or type mismatches between operator and operand.
4. **Logical :**code not reachable, infinite loop.

**Left Recursion** : The grammar : A -> Aa | a is left recursive. Top down parsing techniques cannot handle left recursive grammar so we convert left recursion into right recursion.

Left recursion elimination : A -> Aa | a ⇒ A -> aA'

A' -> aA' | a

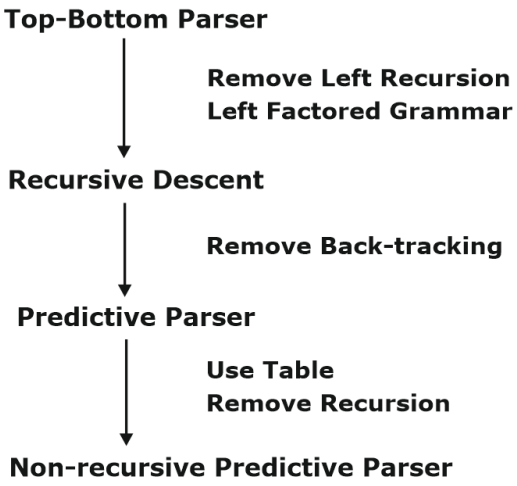
**Left Factoring** : If a grammar has common prefixes in r.h.s of nonterminal then sub grammar needs to be left factored by eliminating common prefixes as follows :

A -> ab1 | ac2 ⇒ A -> A -> aA'

A' -> A -> b1 | c2

**FIRST(A)** is a set of the terminal symbols which occur as first symbols in string derived from A

**FOLLOW(A)** is the set of terminals which occur immediately after the nonterminal A in the strings derived from the starting symbol.




**LR(0) Parser** : Closure() and goto() functions are used to create canonical collection of LR items.
Conflicts in LR(0) parser :

1. **Shift Reduce (SR) conflict :** when the same state in DFA contains both shift and reduce items. A -> B .xC (shifting) B -> a. (reduced)
2. **Reduced Reduced (RR) conflict :** two reductions in same state of DFA A -> a. (reduced) B -> b. (reduced)

**SLR Parser** : It is powerful than LR(0).

Ever LR(0) is SLR but every SLR need not be LR(0).

Conflicts in SLR

1. SR conflict : A -> B . xC (shifting) B -> a. (reduced) if FOLLOW(B) ∩ {x} ≠ φ
2. RR conflict : A -> a. (reduced) B -> b. (reduced) if FOLLOW(A) ∩ FOLLOW(B) ≠ φ

**CLR Parser** : It is same as SLR parser except that the reduced entries in CLR parsing table go only in the FOLLOW of the l.h.s nonterminal.

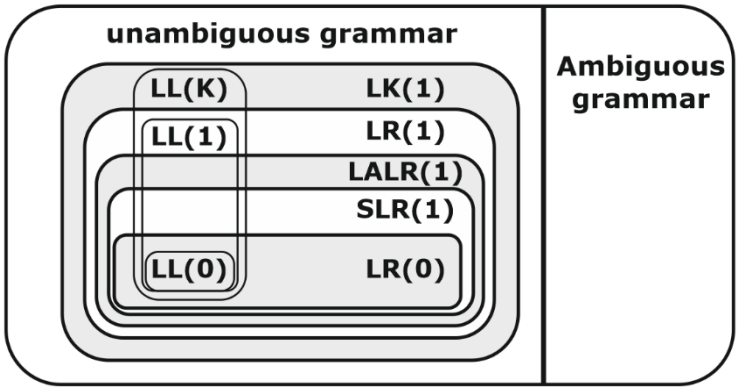
**LALR Parser** : It is constructed from CLR parser, if two states having same productions but may contain different lookaheads, those two states of CLR are combined into single state in LALR.
Every LALR grammar is CLR but every CLR grammar need not be LALR.

**Parsers Comparison :**

LR(0) ⊂ SLR ⊂ LALR ⊂ CLR

LL(1) ⊂ LALR ⊂ CLR

If number of states LR(0) = n1, number of states SLR = n2, number of states LALR = n3, number of states CLR = n4 then, n1 = n2 = n3 <= n4




**Syntax Directed Translation**: Syntax Directed Translation are augmented rules to the grammar that facilitate semantic analysis.

**Eg –** S -> AB {print (*)}
A -> a {print (1)}
B -> b {print (2)}

**Synthesized Attribute** : attribute whose value is evaluated in terms of attribute values of its children.

**Inherited Attribute** : attribute whose value is evaluated in terms of attribute values of siblings or parents.

**S-attributed SDT**: If an SDT uses only synthesized attributes, it is called as S-attributed SDT. S-attributed SDTs are evaluated in bottom-up parsing, as the values of the parent nodes depend upon the values of the child nodes.

**L-attributed SDT** : If an SDT uses either synthesized attributes or inherited attributes with a restriction that it can inherit values from left siblings only, it is called as L-attributed SDT. Attributes in L-attributed SDTs are evaluated by depth-first and left-to-right parsing manner.

**Activation Record** : Information needed by a single execution of a procedure is managed using a contiguous block of storage called activation record. An activation record is allocated when a procedure is entered and it is deallocated when that procedure is exited.

**Intermediate Code** : They are machine independent codes. Syntax trees, postfix notation, 3-address codes can be used to represent intermediate code.

**Three address code**:

1. Quadruples (4 fields : operator, operand1, operand2, result)
2. Triplets (3 fields : operator, operand1, operand2)
3. Indirect triples

**Code Optimization** :

Types of machine independent optimizations –

1. **Loop optimizations** :
   - Code motion : reduce the evaluation frequency of expression.
   - Loop unrolling : to execute less number of iterations

- Loop jamming : combine body of two loops whenever they are sharing same index.

2. **Constant folding** : replacing the value of constants during compilation
3. **Constant propagation** : replacing the value of an expression during compile time.
4. **Strength reduction** : replacing costly operators by simple operators.

****