Matrix Chain Multiplication :-

```
MATRIX_MULTIPLY (A, B)
1) If A.column ≠ B.rows
2)      Error "Incompatible Dimensions"
3) else let C be a new A rows X B columns matrix
4)    for i=1 to A.rows
5)       for j=1 to B.columns
6)            Cij = 0
7)    for k=1 to A.columns
8)            Cij = Cij + Aik X Bkj

9)    return C.
```

→ optimization function — no. of matrices

$$P(n) \cdot \begin{cases} 1 & \text{, if } n=1 \\ \sum_{k=1}^{n-1} P(k) \cdot P(n-k) & \text{if } n \geq 2. \end{cases}$$

↓
Possible parenthesization

(1) Characterize structure of optimal solution.
(2) Recursively define value of solution.
(3) Find best solution
(4) Compute value of optimization
(5) Construct an optimal solution from computed information...

Make table → Find answer through table

$$n[i,j] = \begin{cases} 0 & i=j \\ \min \{ m[i,k] + m[k+1, j] \\ \quad + p_i + p_k\ p_j \} & \text{if } i<j \\ & \text{if } i>j \end{cases}.$$

m[i,k] - Possible Algo's for this calculated and minimum is chosen.

```
MARTIX_CHAIN ORDER(p).
1) n = p.length - 1
2) let m[1---n, 1--n] and
        S[1---n-1, 2--n] be new tables.
                                  m, S → matrix
3)    for i = 1 to n
4)        m[i,j] = 0
5)    for l = 2 to n      → choose level
        for i = 1 to n-l+1
            j = i+l-1
            m[i,j] = ∞
      for K = i to j-1
        q = m[i,K] + m[K+1, j] + pi-1 pK pj
      if q < m[i,j]
            m[i,j] = q      - otherwise remains ∞.
            S[i,j] = K
```

2 to 9 +1 → 2 to 5-2+1 → 2 to 5-1 → 2 to 4

l → matrix chain length

K = i to j-1
= 2 to 5-1
= 2 to 4

$i = 1$ to $n-l+1$, $j = i+l-1$, $m[i,j] = \infty$

Calculate 1 out of 2 $\quad k = i$ to $j-1$.

Ques) $A_1 = 30 \times 35 \quad p_0 = 30$

$A_2 = 35 \times 15 \quad p_1 = 35$

$A_3 = 15 \times 5 \quad p_2 = 15$

$A_4 = 5 \times 10 \quad p_3 = 5$

$A_5 = 10 \times 20 \quad p_4 = 10$

$A_6 = 20 \times 25. \quad p_5 = 20$

$p_6 = 25.$

(A) (B)

1) $n = p.length - 1$

$n = 5.$

2) let $m[1---5, 1---5]$ $m[5,5]$ and

$s[1---4, 2---5]$.

3) for $i = 1$ to $n.(5)$

4) $m[1,1] = 0$

$m[2,2] = 0$

$m[3,3] = 0$

$m[4,4] = 0$

$m[5,5] = 0$

5) for $l = 2$ to $n.(5)$

$l = 2$

for $i = 1$ to $5-2+1 = 4$

$i = 1$ to 4

$i=1$ { $j = 1+2-1 = 2$

$m[1,2] = \infty$

for $k = 1$ to 1

$q = m[i,k] + m[k+1, j] + p_{i-1} p_k p_j$

$q = m[1,1] + m[2,2] + p_0 p_1 p_2$

$= 0 + 0 + 30 \times 35 \times 15$

$q = 15750$

$j = i+l-1$

$i = 2$ { $j = 2+2-1 = 4-1 = 3$

$m[2,3] = \infty$

for $k = 2$ to 2.

$k=2$ { $q = m[2,2] + m[3,3] + p_1 p_2 p_3$

$= 0 + 0 + 35 \times 15 \times 5$

$= 2625$

$i = 3$ { $j = 3+2-1 = 5-1 = 4$

$m[3,4] = \infty$

for $k = 3$ to 3

$q = m[3,3] + m[4,4] + p_2 p_3 p_4$

$= p_2 p_3 + p_4$

$= 15 \times 5 \times 10 = 750.$

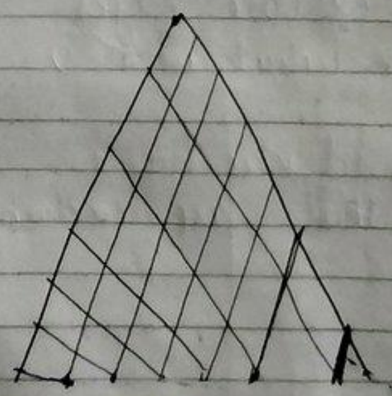
$i = 4$ { $j = 4+2-1 = 6-1 = 5$

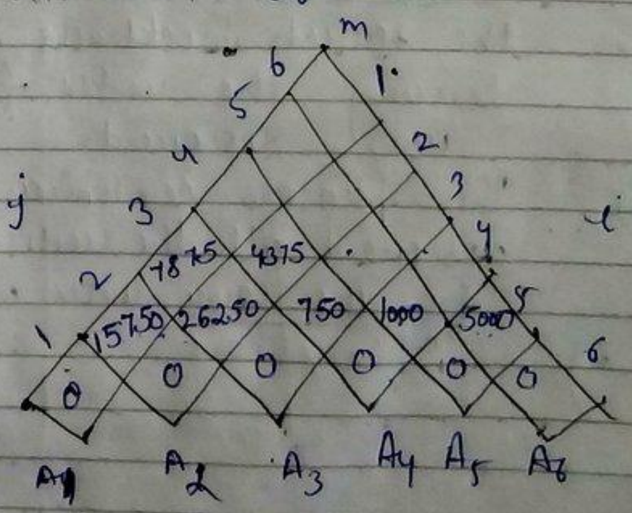
$m[4,5] = \infty$

for $k = 4$ to 4

$q = m[4,4] + m[5,5] + p_3 p_4 p_5$

$= 5 \times 10 \times 20 = 1000.$





$m[5,6]$

$= m(0,1)$

Analysis of BFS

After initialisation, BFS never whitens a vertex & the line B connects that comes next each line

Depth First search

1) Depth First search explores edges out of the most recently discov

---

1) Depth first search explore edges out of the most recently discovered vertex v that ~~has~~ still ~~undiscovered~~ unexplored edges leaving it.
2) ~~When~~ Once the v's edges have been explored. the search backtracks to explore edges leaving the vertex from which v was discovered.
3) This process continues until we have discovered all the vertices that are reachable from the source vertex.
4) If we have

All pair shortest path

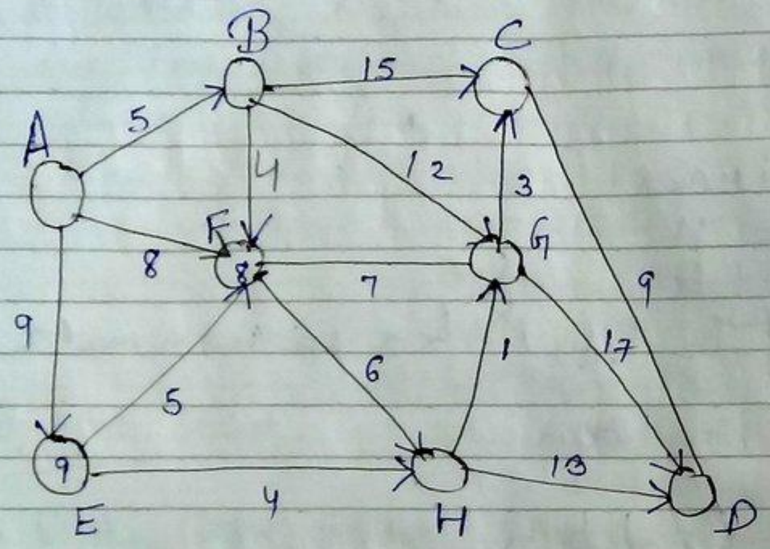
Warshall

1. n = no. of rows
2. $D^{(0)} = W$
3. for K = 1 to n
4. Let $D^{(k)} = (dij)^k$ be a new n x n matrix
5. for i = 1 to n
6. for j = 1 to n
7. $dij^{(k)} = \min dij^{(k-1)}, dik^{(k-1)} + dkj^{(k-1)}$

---

1. n = no. of rows
2. $D^{(0)} = W$
3. For K = 1 to n
4. Let $D^{(k)} = (dij^k)$ be a new n x n matrix
5. for i = 1 to n
6. for j = 1 to n
7. $dij^{(k)} = \min (dij^{(k-1)}, dik^{(k-1)} + dkj^{(k-1)})$
8. Return $D^{(n)}$

Bellman-Ford :-



initialization,

Pass 0	v.π	v.d	u.d	w(u,v)	
A	NIL	∞	0	5=5	v.d=5
B	NIL	∞	0	8=8	v.d=8
C	NIL	∞	0	9=9	v.d=9
D	NIL	∞	0		
E	"	∞	0		
F	"	∞	0		
G	"	∞	0		

Pass 1	v.π	v.d	u.d	w	
A-B	A=0	∞	0	5	B=5
A-F	A=0	∞	0	8	F=8
A-E	A=0	∞	0	9	E=9
B-C	B=5	∞	5	15	→20 C
B-G	B=5	∞	5	12	→17 G
B-F	B=5	8	5	4	
F-G	F=8	17	8	7	v.d→15
F-H	F=8	∞	8	6	v.d→14
E-F	E=9	8	9	5	
E-H	E=9	14	9	4	
H-D	H=13	∞	13	13	2.6

	v.π	v.d	u.d	w	
H-G	H=13	15	13	1	14
G-D	G=14	26	14	11	25
C-D	C=17	25	17	9	
G-C	G=14	20	14	3	

INITIALIZE-SINGLE-SOURCE (G,s)

(1) S = ∅
(2) Q = G.V
(3) while Q ≠ ∅
(4) u = EXTRACT-MIN(Q)
(5) S = S ∪ {u};
(6) Relax each v∈G.adj[u]
(7) & Relax them.

NP = ℚ

→ Dynamic programming is an optimization technique.
→ Dynamic programming computes its solution bottom up by synthesizing them from smaller subsolutions, and by trying many possibilities and choices before it arrives at the optimal set of choice.
→ Dynamic programming splits its input at every possible split points rather that at a prespecified points. After trying all split points, it determines which split point is optimal.

Steps of Dynamic programming

1) Develop a mathematical notation that can express any solution and subsolution for the problem at hand.
2) Prove that principle of optimality holds.
3) Develop a recurrence relation that relates a solution to its subsolutions using the math notation of step 1.
   Indicates what the initial values are for that recurrence relation, and which term signifies the final solution.
4) Write an algorithm to compute the recurrence relation.

Principle of optimality: A problem is said to be satisfy the principle of optimality, if the subsolutions of an optimal solution of the problem are themselves optimal solution of their subproblems.

Topological sort → A topological sort of dag $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v) then u appears before v in the ordering.

```
Topological_Sort(G)
1) Call DFS(G) to compute finishing times
   v.f for each vertex v.
2) As each vertex is finished, insert it onto the
   front of a linked list.
3) return the linked list of vertices.
```

```
MATRIX_MULTIPLY(A, B)
1) if A.columns ≠ B.rows
2)    error "incompatible dimensions"
3) else let C be a new A.rows x B.columns matrix
4)    for i = 1 to A.rows
5)       for j = 1 to B.columns
6)          C[i,j] = 0
7)          for k = 1 to A.column
8)             C[i,j] = C[i,j] + a[i,k] . b[k,j]
9) Return C.
```

```
MATRIX_CHAIN_ORDER(P)
1) n = P.length - 1
2) let m = [1..n, 1..n] and
   s = [1..n-1, 2..n] be new tables.
3) for i = 1 to n
4)    m[i, i] = 0
5) for l = 2 to n
6)    for i = 1 to n - l + 1
7)       j = i + l - 1
8)       m[i, j] = ∞
9)       for k = i to j -
```

b) $q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j$

ii) If $q < m[i,j]$
iii) $m[i,j] = q$
iv) $s[i,j] = k$
v) Return m and s.

BFS (Breadth-First Search)

1. for each vertex $u \in G.V$
2. $u.color = WHITE$
3. $u.d = \infty$
4. $u.\pi = NIL$
5. $s.color = GRAY$
6. $s.d = 0$
7. $s.\pi = NIL$
8. $Q = \phi$
9. ENQUEUE (Q, s)
10. while $Q \neq \phi$
11. $u =$ DEQUEUE (Q)
12. for each $v \in G.adj[u]$
13. If $v.color == WHITE$
14. $v.color = GRAY$
15. $v.d = u.d + 1$
16. $v.\pi = u$
17. ENQUEUE (Q, v)
18. $u.color = BLACK$.

DFS (G)

1. for each vertex $v \in G.V$
2. $u.color = WHITE$
3. $u.\pi = NIL$
4. time = 0
5. for each vertex $v \in G.V$
6. If $u.color == WHITE$
7. DFS_VISIT (G, u)

DFS_VISIT (G, u)

1. time = time + 1
2. $u.d$ = time
3. $u.color = GRAY$
4. for each Vertex $v \in G.adj(u)$
5. If $v.color == WHITE$
6. $v.\pi = u$
7. DFS_VISIT (G, v)
8. $u.color = BLACK$
9. time = time + 1
10. $u.f$ = time.

---

1) BFS is one of the simplest algorithm for searching a graph.
2) Prims minimum spanning tree and Dijkstra's single source shortest path algorithm uses idea similar to those in BFS.
3) BFS color each of the vertices white, gray or black. All the vertices are initially colored initialized to white when they are constructed. A white vertex is an undiscovered vertex. When a vertex is initially discovered, it is colored gray and when BFS has completely explored a vertex it is colored black.

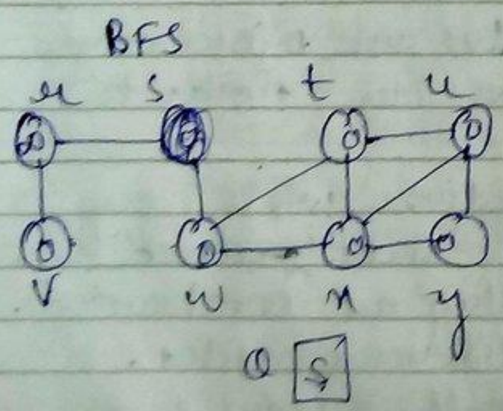
Analysis of BFS

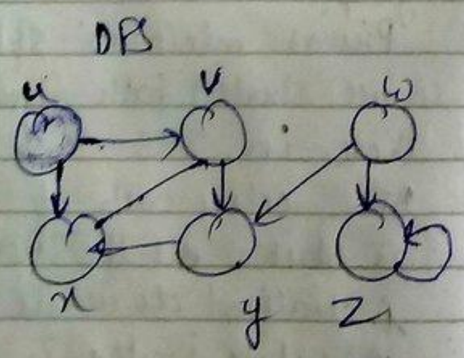
After Initialization, BFS never whitens a vertex, and thus the test in line 13 ensures that each is enqueued at most once and hence dequeued at most once. The operations of enqueuing

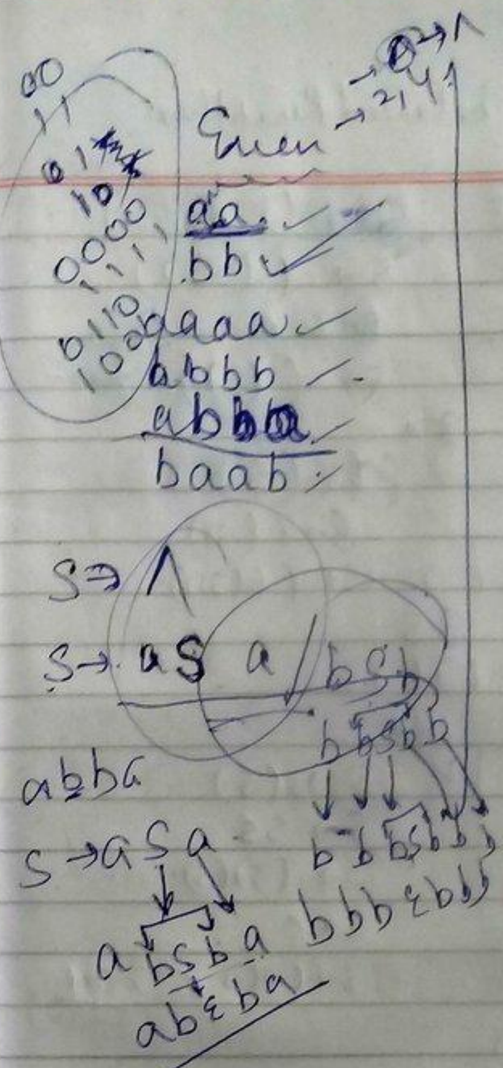
---

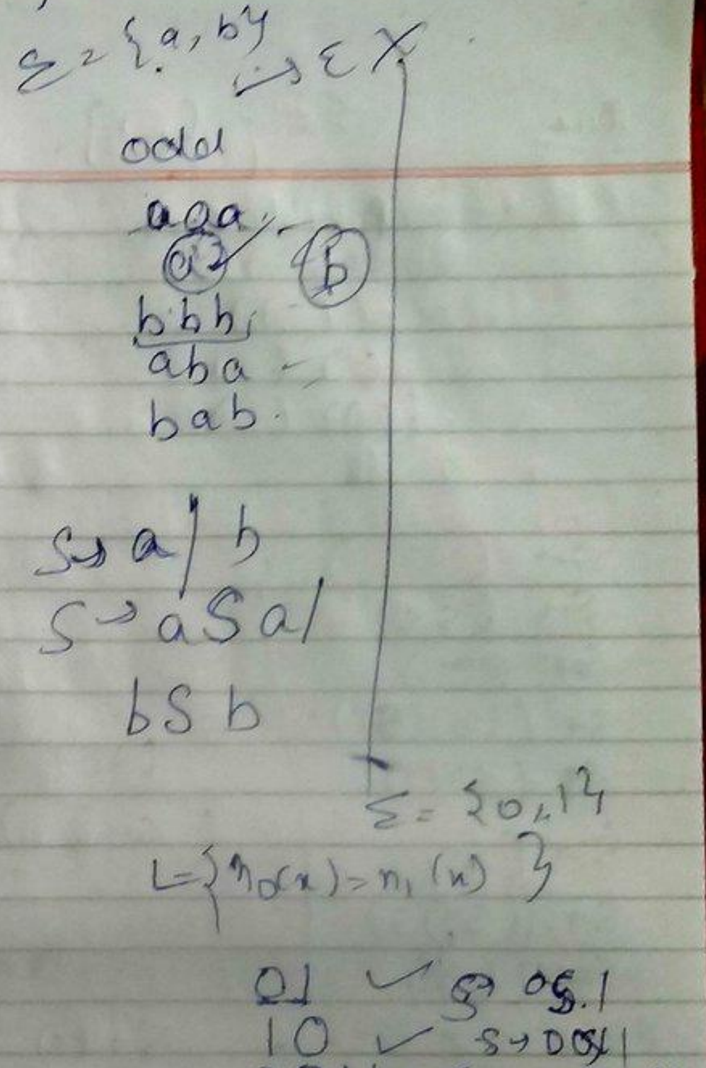
DFS explores edges out of the most recently discovered vertex that still has unexplored edges leaving it.

→ Once all the edges have been explored, the search "backtracks" to explore edges leaving the vertex from which v was discovered.





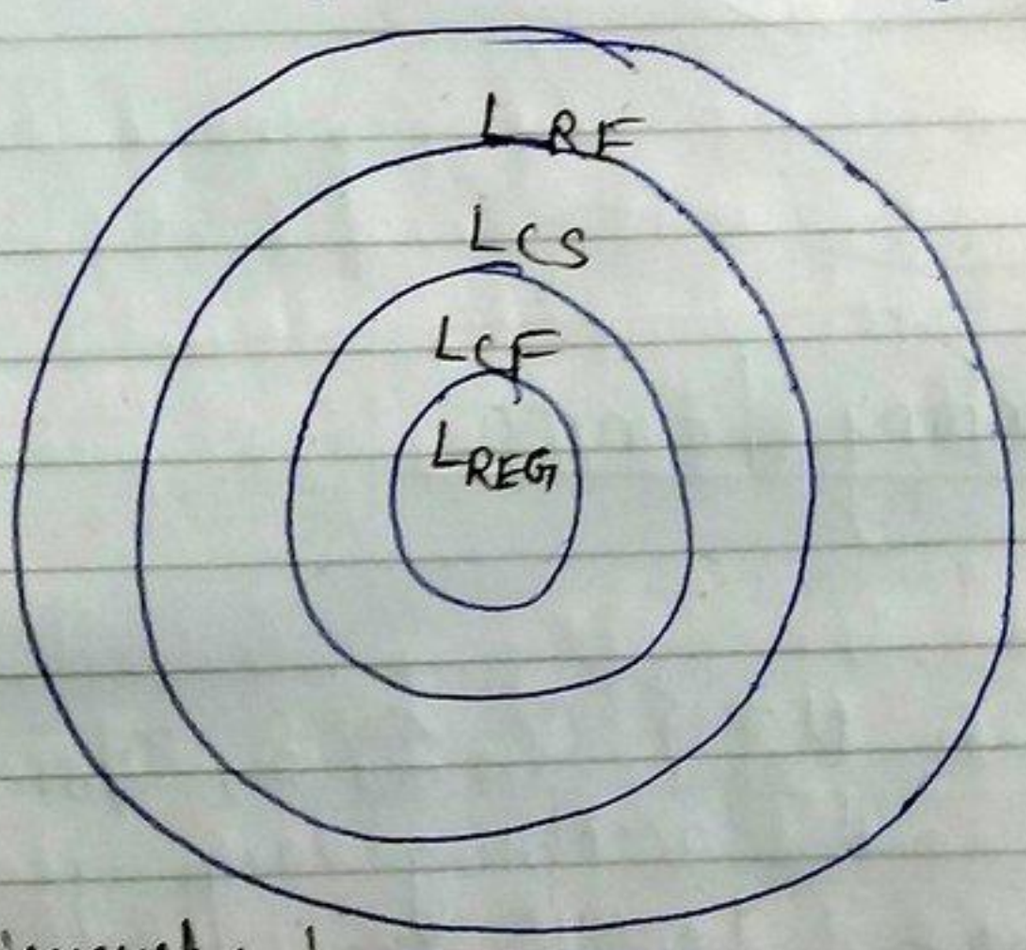




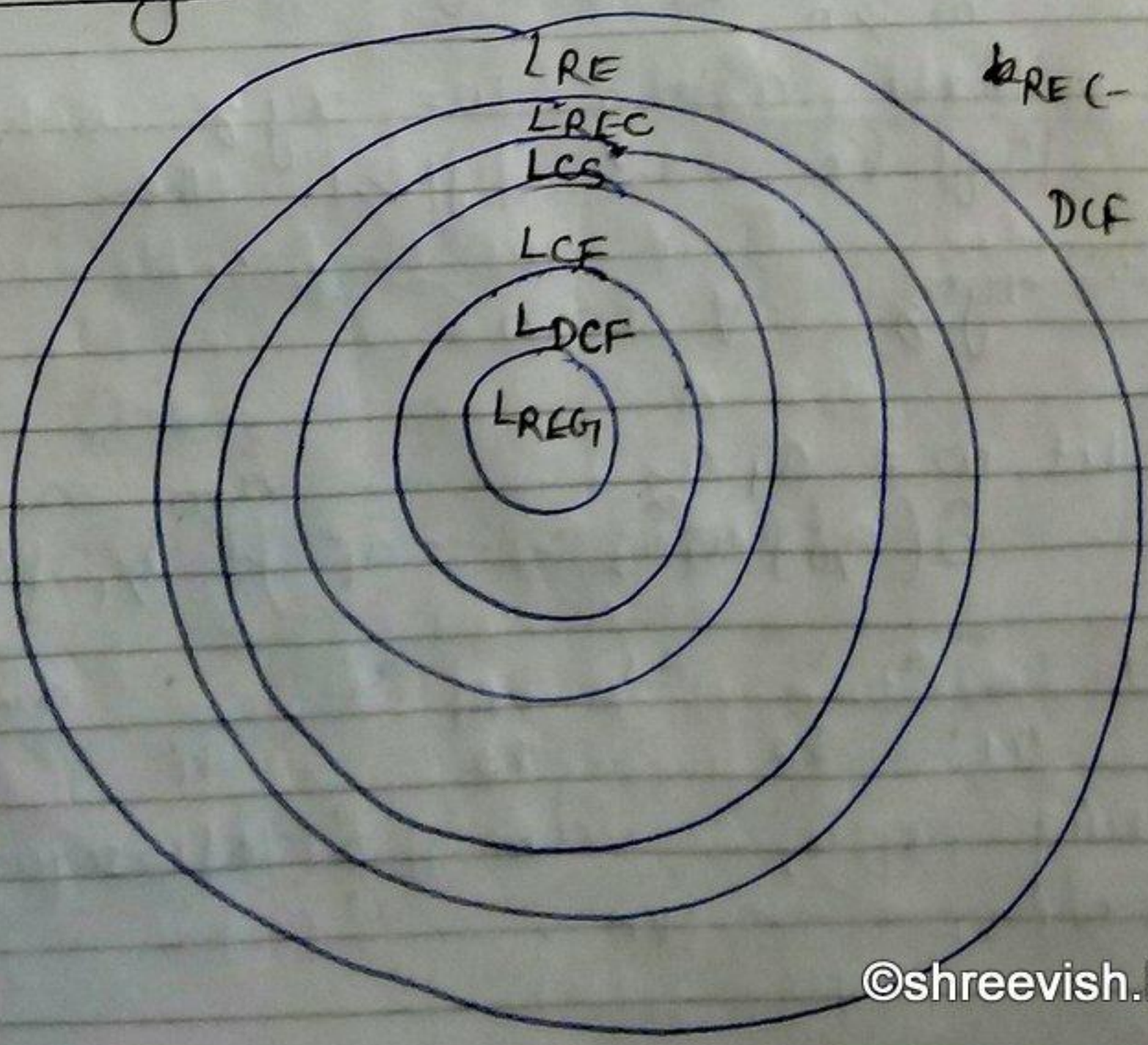
Chomsky Hierarchy

L_{RE} → Recursively Enumerated language
L_{CS} → Context ~~Free~~ Sensitive language.
L_{CF} → Context Free language
L_{REG} → Regular language

Type 0 → is generated by Unrestricted Grammar
Type 1 → is generated by Context Sensitive
Type 2 → is generated by Context Free
Type 3 → is generated by Regular language.



Modified Hierarchy :



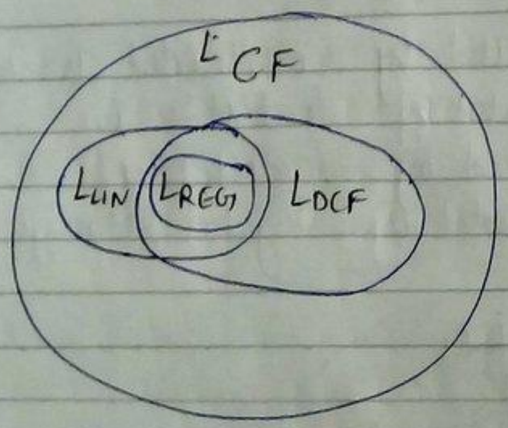
L_{REC} - Recursive language
DCF → Deterministic Context free

CLRD

Relation b/w
Context free
Linear
Regular
Deterministic Context free

Context free
Linear
Regular
DCF

Contro

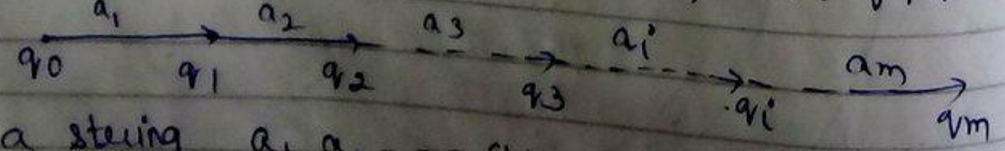


Pumping Lemma

(a) Consider a DFA $M = \{Q, \Sigma, \delta, q_0, F\}$ having n states and accepting language L.
Let $w \in L$ is a string of length m such that $m \geq n$.
Suppose we have a string $w = xyz$ such that $|xy| \leq n$ and $|y| \geq 1$ then

$xy^iz \quad \forall\ i \geq 0$.

Proof: Let $w = a_1 a_2 \dots a_m$ and
$\delta(q_0, a_1 a_2 \dots a_i) = q_i \quad \forall\ 1 \leq i \leq m$



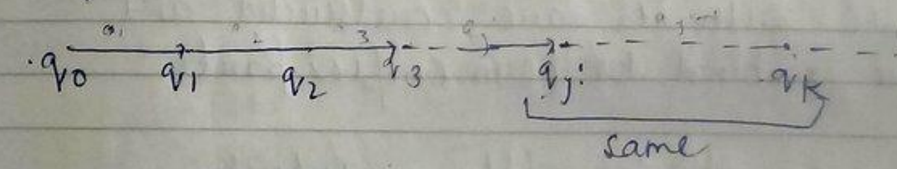
For a string $a_1 a_2 \dots a_m$, a state sequence is $q_0 q_1 q_2 \dots q_m$.

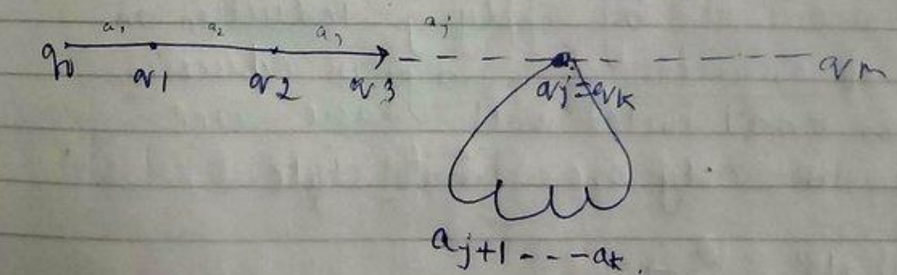
If $m = n$, the state sequence is
$q_0 q_1 q_2 \dots q_m$.
As the state sequence has $(n+1)$ state, but the given FA has only n states so the states are not distinct.

- let q_j and q_k are same
$q_j = q_k$ for i, $j \leq m$.

- let $w = \underbrace{a_1 a_2 \dots a_j}_{x} \underbrace{a_{j+1} \dots a_k}_{y} \underbrace{a_{k+1} \dots a_m}_{z}$

Since $j < k$, so $y = a_{j+1} \dots a_k$ has a length atleast 1.





Let $w = a_1 a_2 \dots a_m \in L$
Consider a string $a_1 a_2 \dots a_j\ a_{k+1} \dots a_m$.
i.e. which is not passing through loop.
$= \delta(q_0, a_1 a_2 \dots a_j\ a_{k+1} \dots a_m)$
$= \delta(\delta(q_j, a_1 a_2 \dots a_j)\ a_{k+1} \dots a_m)$
$= \delta(q_j, a_{k+1} \dots a_m)$
δ q_m

24 feb

Kleene Theorem (Part - I)

Statement :- If r is a regular expression then we can construct an NFA that accepts $L(r)$. So $L(r)$ is a regular language.

Proof :- A language is regular if it is accepted by some dfa. Due to the equivalence between nfa and dfa, a language is also regular if it is accepted by some nfa.

* We will show that if we have any regular expression r, we can construct an nfa that accepts $L(r)$.
* Initially simple automata are constructed and then they are combined to form complex parts.

1) We will prove it by applying induction on the number of operators of a regular expression.
2) The NFA so constructed will have one initial state and only one final state.

Basis of induction :→

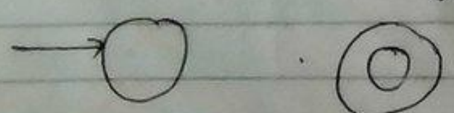
For zero operator :- There are 3 possible cases.

$r = \phi$ $\quad L(r) = \{\}$

$r = \wedge$ $\quad L(r) = \{\wedge\}$

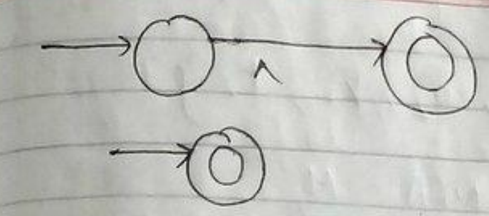
$r = a$ $\quad L(r) = \{a\}$

Consider the following NFA M



Clearly $L(M) = \{\}$
$L(r) = \{\}$
$\therefore L(M) = L(r)$
∴ There is an NFA.

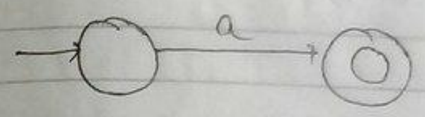
Consider the following NFA M



Clearly $L(M) = \{\wedge\}$
$L(r) = \{\wedge\}$
$\therefore L(r) = L(M)$

There is an NFA.

Consider the following NFA M $\forall a \in \Sigma$



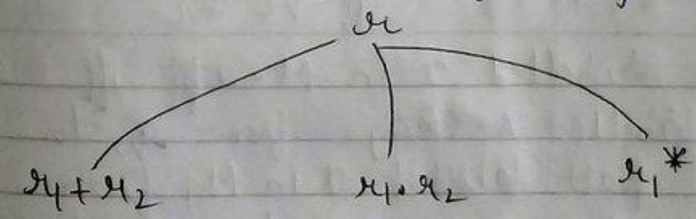
$L(M) = \{a\}$
$L(r) = \{a\}$
$\therefore L(M) = L(r)$

Inductive Hypothesis :-

Let it is true for any regular expression r, which contains less than i operators, we have an NFA M such that

$$L(M) = L(r).$$

Proof :→ We have to prove it for a regular expression r having i operators. There are 3 possible forms of writing the expression r as :



Case 1 :- $r = r_1 + r_2$

* Let '+' is contributing a value 1 to the total no. of operators. So both r_1 and r_2 will have operators less than i.
* Let M_1 and M_2 be two NFA's accepting language corresponding to r_1 and r_2.

i.e. $L(M_1) = L(r_1)$ $\quad L(M_2) = L(r_2)$

$L(r) = L(r_1) \cup L(r_2)$

$L(r) = L(M_1) \cup L(M_2)$ ⟶ (1) eq

Suppose M_1 is defined as $M_1 = \{Q_1, \Sigma, \delta_1, q_1, f_1\}$

and M_2 as $\{Q_2, \Sigma, \delta_2, q_2, f_2\}$

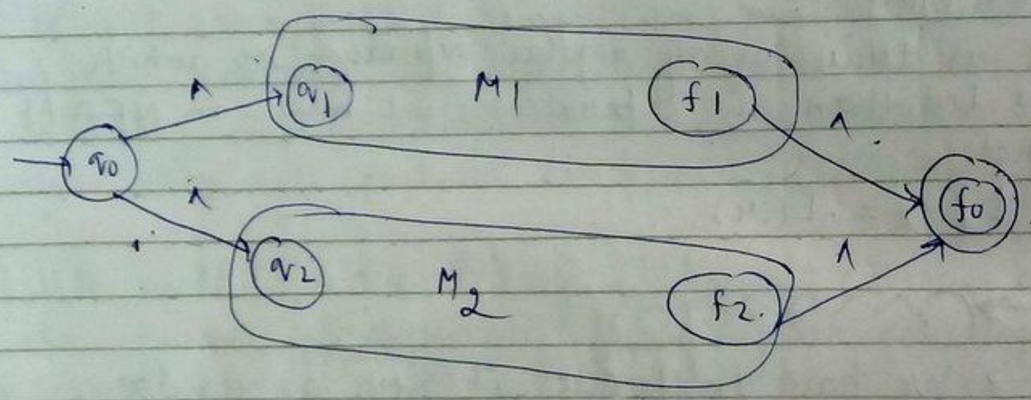
* Suppose we construct an NFA M

$M = \{Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma, \delta, q_0, f_0\}$

where $\delta(q_0, \wedge) = \{q_1, q_2\}$

$\delta(f_1, \wedge) = f_0$

$\delta(f_2, \wedge) = f_0$



Also, $\delta(q, a) \to \delta(q_1, a)$ iff $q \in Q_1$

$\delta(q, a) = \delta(q_1, a)$ iff $q \in Q_2$

i.e. there is a path labelled x in M iff there is a path labelled $\wedge$-string from q_0 to q_1 followed by a path labelled x in M_1 from q_1 to f_1 followed by a $\wedge$-string transition from f_1 to f_0. Or

there is a path labelled $\wedge$-string from q_0 to q_2 followed by a path labelled x in M_2 from q_2 to f_2 followed by a path labelled $\wedge$ from f_2 to f_0.

i.e. $x \in L(M)$ iff $(\wedge \cdot x \in L(M_1) \cdot \wedge)$ or $(\wedge \cdot x \in L(M_2) \cdot \wedge)$

$x \in L(M)$ iff $x \in L(M_1)$ or $L(M_2)$.

$L(M) = L(M_1) \cup L(M_2)$ — (2)

∴ From (1) & (2)

$L(r) = L(M)$.

Case:- $r = r_1 + r_2$

* Let 'i' is contributing a value 1 to the total number of operators. So, both r_1 and r_2 will have operators less than i.

* Let M_1 & M_2 be two NFAs accepting languages corresponding to r_1 & r_2

i.e. $L(M_1) = L(r_1)$

$L(M_2) = L(r_2)$

$L(r) = L(r_1) + L(r_2)$

$= L(M_1) + L(M_2)$. — (3)

Suppose we construct an NFA M defined as

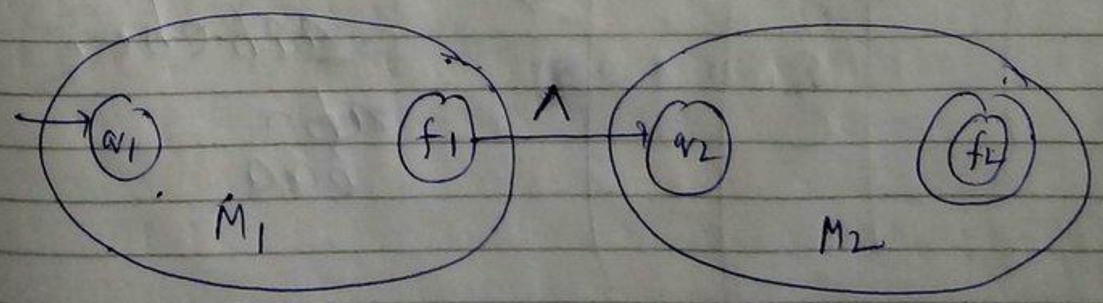
$M = \{Q_1 \cup Q_2, \Sigma, \delta, q_1, f_2\}$

where δ is defined as

$\delta(f_1, \wedge) = q_2$

$\delta(q, a) = \delta_1(q, a)$ iff $q_1 \in Q_1$

$\delta(q, a) = \delta_2(q, a)$ iff $q_2 \in Q_2$.



There is a path labelled x in M iff there is a path labelled x_1 from q_1 to f_1 in M_1 followed by λ-string transition from f_1 to q_2 followed by a path labelled x_2 in M_2 from q_2 to f_2.

$\Rightarrow$ $x \in L(M)$ iff $x \in L(M_1) \cdot \Lambda \cdot x \in L(M_2)$
iff $x \in L(M_1) \cdot x \in L(M_2)$

$L(M) = L(M_1) \cdot L(M_2)$ ——— (4)

from (3) & (4)
$L(R) = L(M)$.

WCW^R

abab a b

a
a
z_0

a
b
z_0

state	I/p	stack top	Move
q_0	Λ	z_0	(q_0, z_0)
q_0	a	z_0	$q_0, a z_0$
q_0	b	z_0	$q_0, b z_0$
q_0	a	a	$q_0, aa z_0$
	a	b	$aq_0, ab z_0$
	b	b	$q_0, bb z_0$
	b	a	$q_0, ba z_0$
		a	
		C	
		C	

aabcbaa

baa

$w \cdot C \cdot w^R$

wCw^R

Λ

Finite state Machine

A FSM is a Basic Machine

Basic Machine:- It is the most primitive machine which recognizes an input state & generate output.

It acts as a function called Machine function (MAF) which

FSM- is a basic machine which has its internal state control.

A FSM has

I, S, f_1.

Two functions MAF $\rightarrow$ $I \times S \rightarrow S$.
state STF $I \times S \rightarrow S$.

$I \times S \rightarrow O$

$I \times S \rightarrow S$

Block diagram
Revision
Applications of FSM

1. Design of lexical Analyzer
2. Text Editor
3. Shell checker
4. Design of compiler