08 Mar 2013.

IInd SESSIONAL

DELTA Page No. Date:

GREEDY ALGORITHM

Huffman Code :

(*) Used for making .rar file.

(*) Helps in network security through coding ~~password~~ words, during transmission

Eg :- Transmission of file.

	a	b	c	d	e	f
frequency :	45K	12K	13K	16K	9K	5K
code word :	000	001	010	011	100	101

During encoding :- use of fixed code length. encoding

Now, we use 3 bit code word,

Now, total file size = 45000×3 + 12000×3 + 13000×3 + 16000×3 + 9000×3 + 5000×3

= 3000000 bits.

large size ⇒ a problem.

Now, variable size scheme ⇒ for large frequency ⇒ small bit length
for small frequency ⇒ bigger bit length
→ This is done using huffman coding algorithm.

(*) let there by file C & |C| represent no. of bits.

Here, we use tree Method.

No. of leaves = |C| → leaves contains character
No. of Nodes = |C| - 1 → Contains frequency

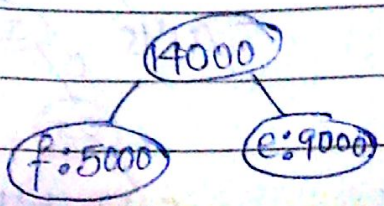
$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c)$$

No. of bits | gabbbcddd | frequency | depth.

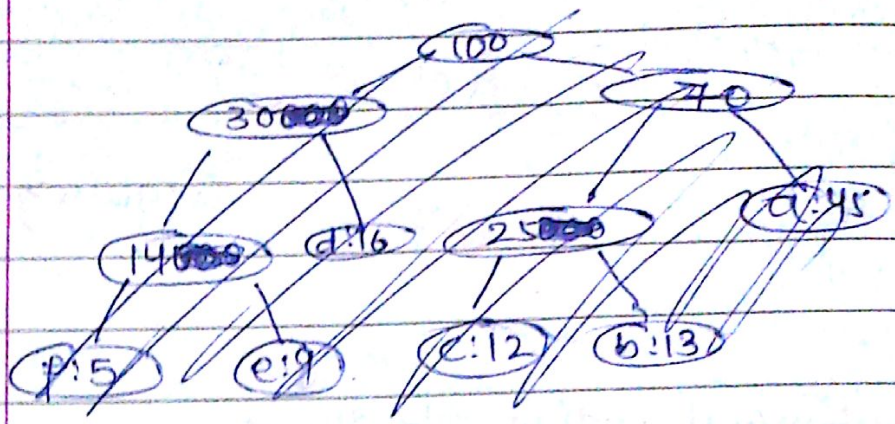
Huffmann Coding Algorithm:

HUFFMANN (C)
1. n ← |C|
2. Q ← C
3. for i ← 1 to n-1
4. do allocate a new node z.
5. left[z] ← x ← EXTRACT_MIN(Q)
6. Right[z] ← y ← EXTRACT_MIN(Q)
7. f[z] ← f[x] + f[y]
8. INSERT (Q, z)
9. return EXTRACT_MIN(Q)

Q is Minimum priority queue.
Insert z to Q.



Ques f:5. e:9 c:12 b:18 d:16 a:45



Coding.

Stepwise

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

2. c:12 b:13 14 d:16 a:45
   f:5 e:9

3. 14 d:16 25 a:45
   f:5 e:9 c:12 b:13

4. 25 30 a:45
   c:12 b:13 14 d:16
   f:5 e:9

5) a:45 55
   25 30
   c:12 b:13 14 d:16
   f:5 e:9.

6) 100
   0/ \1
   a:45 55
   0/ \1
   25 30
   0/ \1 0/ \1
   c:12 b:13 14 d:16
   0/ \1
   f:5 e:9

coding : 0 & 1 0 → left 1 → Right

codeword : a = 0

b = 101

c = 100

d = 111

e = 1101

f = 1100

$B(T) = 45 \times 1 + (13+12) \times 3 + (5+9) \times 4 + (16) \times 3$

$= 45 + 75 + 56 + 48$

$= 224$

~~CORRECTNESS~~.

ANALYSIS OF HUFFMAN ALGO.

$\log_2(n)$: Complexity of Heap (Maximum)

Complexity of huffman : $O(n \log n)$.

∵ (n−1) iteration for transforming of Max Heap

∴ complexity $O(n \log n)$

[∵ $(n-1)(\log_2 n)$]

CORRECTNESS OF HUFFMANN CODING ALGORITHM

1) Greedy choice Property:

Theorem:

Let 'C' be an alphabet in which each character 'c' has a frequency $f[c]$. let x and y be two characters in 'C' having the lowest frequencies than there exists a optimal prefix code for 'C' in which the code words for x and y have the same length and differ only in the last bit.

2)

Activity Selection Problem

A_{ij} ⇒ set of Activites.

$A_{ij} = \{a_i, a_2, a_3 \ldots\ldots a_n\}$

a_i ⇒ starting time = s_i

finishing time = f_i

two activites a_i and a_j are compatible only if

$s_i > f_j$ or $s_j > f_i$.

Eg:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

$\therefore \{a_1, a_4, a_8, a_{11}\}$

Steps (Dynamic Programming)

1) Optimal substructure:

$A_{i,j} = \{a_1, a_2 \dots a_j\}$

a_k is selected.

$0 \le s_i \le f_i \le s_k \le f_k \le s_j < \infty$

$A[i \dots k \dots j]$

$A[i \dots k]$ $\quad$ $A[k \dots j]$

Sol = Tot. Activity of $i \dots k + (k) \dots j + k$.

i.e.

$A[i \cdot k] \cup A[k, j] \cup \{a, k\}$

2) Recursive solution

No. of Activities

$$c[i, j] = \begin{cases} 0 & S_{ij} = \phi. \\ \max(\quad) & S_{ij} \neq \phi \end{cases}$$

$c[i,k] + c[k,j] + 1.$

3) Algo design

(Greedy choice)

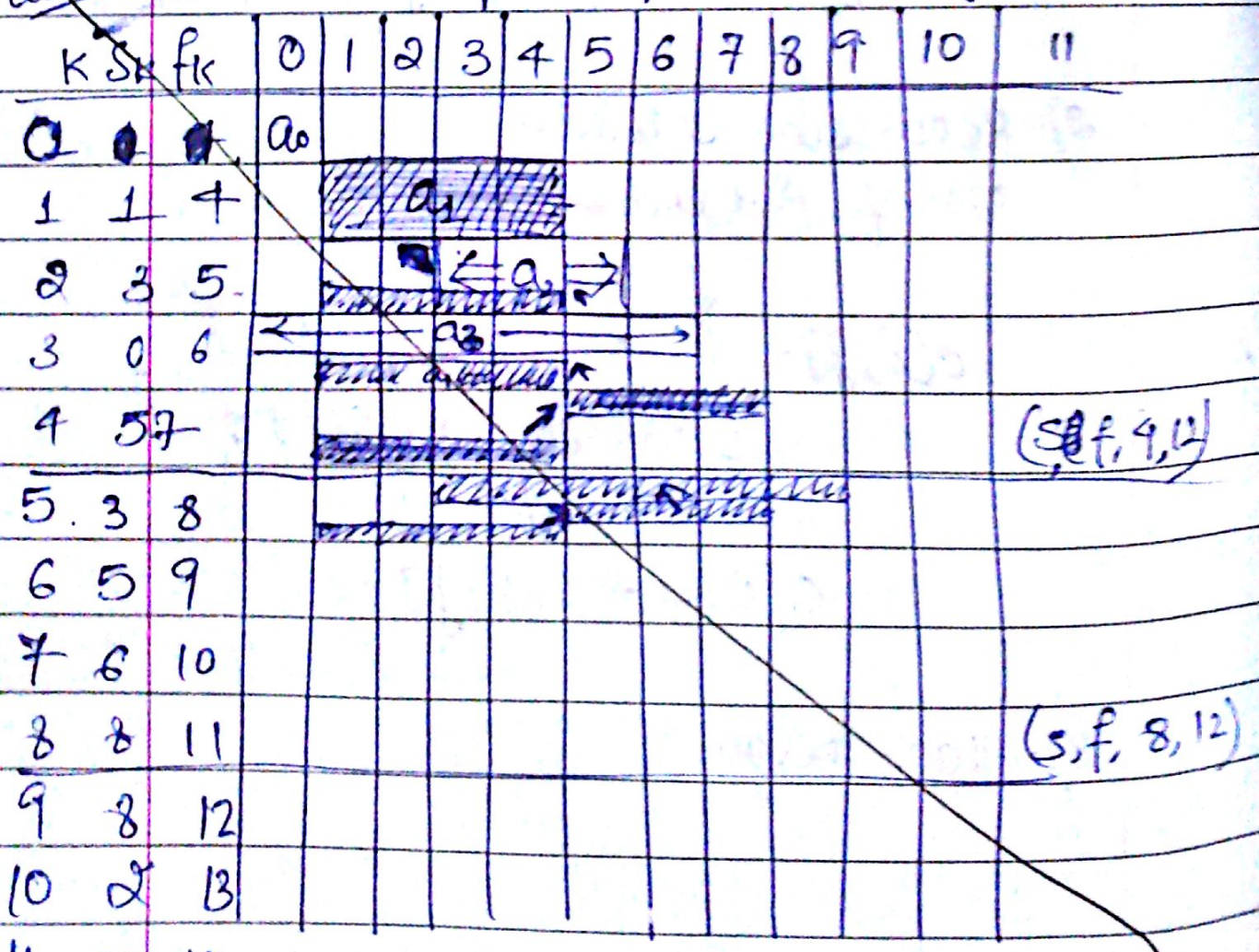
Algo:

RECURSIVE_ACTIVITY_SELECTOR. (s, f, i, j)

1. $m \leftarrow i+1$
2. while $m < j$ and $s_m < f_i$
3. do $m \leftarrow m+1$
4. if $m < j$.
5. then return {am} ∪ Recursive_activity_selector (s, f, m, j)
6. else return ϕ.

———X———X———

Ans: Same example as previous page



k	s_k	f_k
0	•	•
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14

$S_{i,j} = S_{0,m+1}$

a_{m+1}.

f_0 $S_{m+1} = \infty$

Next step in activity selection. Problem.

4)

Convert recursive algorithm to Iterative Algo.

GREEDY-ACTIVITY-SELECTOR(s, f).

1. $n \leftarrow$ length [s]
2. $A \leftarrow \{a_1\}$
3. $i \leftarrow 1$.
4. for $m \leftarrow 2$ to n
5. do if $s_m \geq f_i$
6. then $A \leftarrow A \cup \{a_m\}$.
7. $i \leftarrow m$
8. return A

ELEMENTS OF GREEDY STRATEGY.

A Greedy algorithm obtain optimal solution to a problem by making a sequence of choices. For each decision point in the algorithm, the best choice at the moment is selected.

steps to follow Greedy Algorithm

~~1) Greedy Algo~~

1) Determine optimal substructure of prob.
2) Develop a recursive solution.
3) Prove that at any stage of the recursion, one of the optimal choices is the greedy choice, thus it is always safe to make a greedy choice
4) Show that all but one of the sub-problems induced by having made the greedy choice are empty.
5) Develop a recursive algorithm that implements the greedy strategy.
6) Convert the recursive algo to an iterative problem

ELEMENTS OF G. ALGO.

Steps

1) Cast the optimization problem as 1, In which we make a choice and left with one subproblem to solve
2) Prove that there is always an optimal solu- to the original problem that makes the greedy choice

3) Demonstrate the having made the Greedy choice, we remains with a subproblem, with a property that, If we combine an optimal solution to a subproblem with a greedy choice, we have made, we arrive at an optimal solu to the original problem

Prop:

1) Greedy choice 2) Optimal substructure

Greedy choice Property:

1)

2) How it is differ from D.P.

KNAPSACK PROBLEM

- 0-1 Knapsack
- Fractional Knapsack

13/02/2013.

Ques: Information to be transmitted over the internet contains following character with their associated frequency as shown:- character: a e L n o s t

frequency: 45 65 13 45 18 22 53

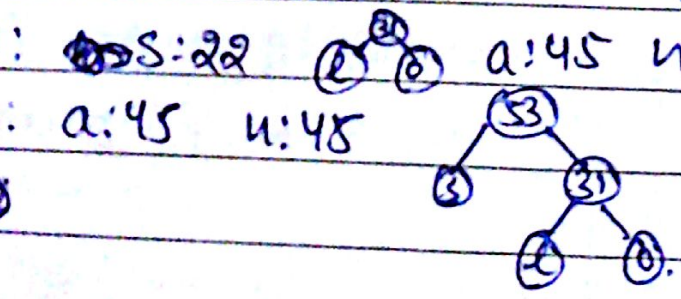
1) Use Huffmann technique
2) Built the Huffmann code tree
3) Find code word for each character.
4) Data consist of only these character. Total bit transmitted = ?
5) What is % saving if data is sent with 8 bit ASCII value without compression.
6) Verify that ur computed Huffmann code are correct

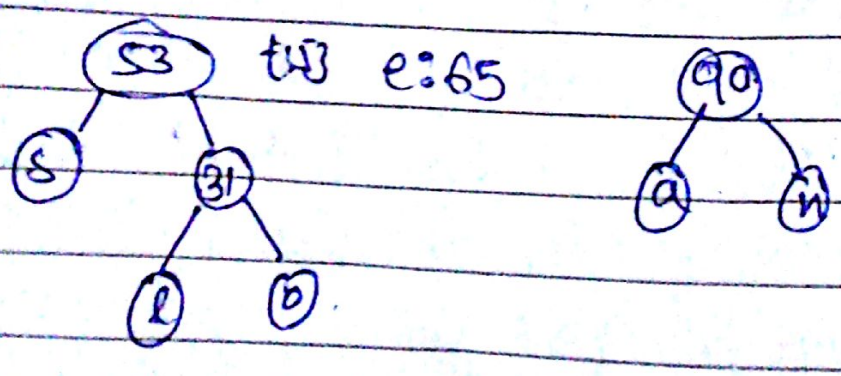
1) step 1: l:13 o:18 s:22 a:45 n:45 t:53 e:65

2: s:22 (l,o):31 a:45 n:45 t:53 e:65

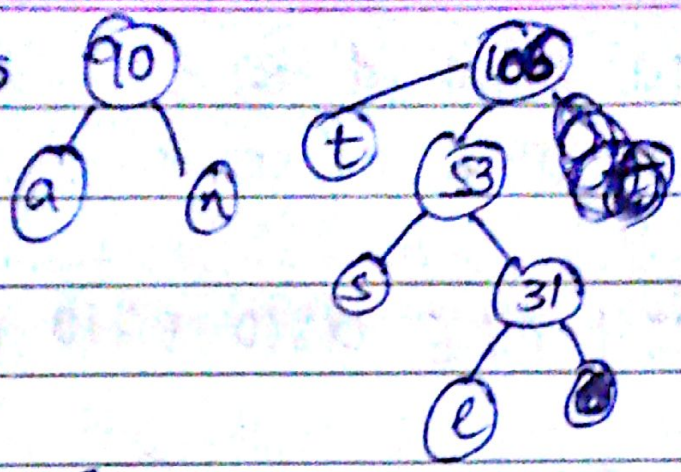
3: a:45 n:45 (s,(l,o)):53 t:53 e:65

4: (s,(l,o)):53 t:53 e:65 ((a,n)):90

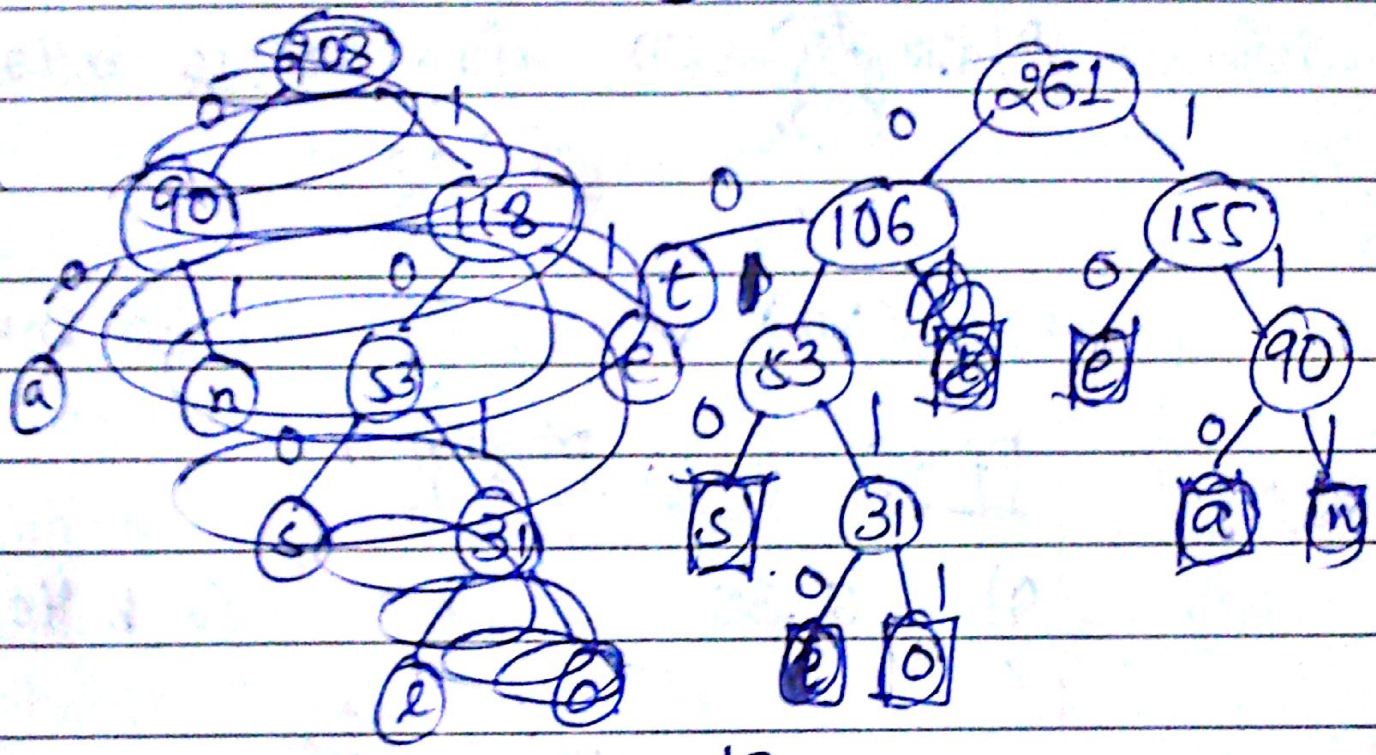




5. e : 65



6



S = 000 e = 10

t = 00 a = 110

l = 0010 n = 111

a = 0011

Total bit = 66 + 106 + 52 + 72 + 130 + 135 + 135

= 696 bits

for 8 bit ASCII total bits = 261 × 8 = 2088 bits

% Savings = $\frac{2088 - 696}{2088} \times 100 = 66.6\%$

Ques Word: a b c d e f

f: 30 40 6 10 10 4.

1) f:4 c:6 d:10 e:10 a:30 b:40

2) d:10 (10) e:10 (10) a:30 b:40

(f:4) (c:6)

3) (10) (20) a:30 b:40

(f:4) (c:6) (d:10) (e:10)

4) a:30 (30) b:40.

(10) (20)

(f:4) (c:6) (d:10) (e:10)

5) (100)

0 1

b:40 (60)

0 1

e:30 (30)

0 1

(10) (20)

0 1 0 1

(f:4) (c:6) (d:10) (e:10)

a: 10 b: 0 c: 1101 d: 1110

e: 1111 f: 1100.

Total bit = 60 + 40 + 24 + 40 + 40 + 16

= 219 bits

BREADTH FIRST SEARCH

1) BFS is used for searching a graph.
2) Given a graph $G=(V,E)$ and a distinguished source vertex, s.
3) BFS explores the edges of G to discover every vertex that is reachable from s. It computes the distance from s to each reachable vertex & forms a BFS tree from s to v.
4) BFS colours each vertex. white, grey & black. All vertices start out white and may later become grey and then black.
5) When a vertex is discovered it becomes non white.
6) In the BFS tree, the root of tree is source vertex, s.
7) d[u] = distance from s to u.
   $\pi[u]$ = Stores parent of u.
8) complexity = $O(E+V)$

ALGO:

BFS(G,S)

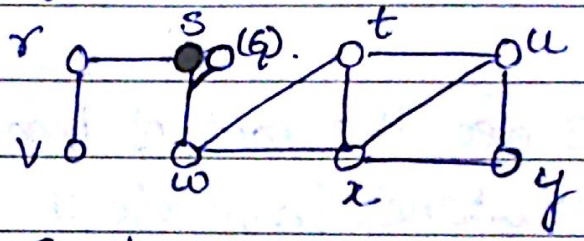
1. for each vertex $u \in V[G] - \{s\}$
2. do color[u] ← white
3. d[u] ← ∞
4. $\pi[u]$ ← Nil.
5. Color[s] ← Gray
6. d[s] ← 0.
7. $\pi[s]$ ← Nil
8. Q ← ϕ.
9. ENQUEUE (Q, s).

```
10. While Q ≠ ϕ.
11. do u ← DEQUEUE (Q).
12. for each v ∈ Adj[u].
13. do if color[v] = WHITE
14.       then COLOR[V] = GRAY.
15.            d[v] ← d[u] + 1.
16.            π[v] ← u.
17.            ENQUEUE (Q, v)
18. Colour[u] ← Black.
```

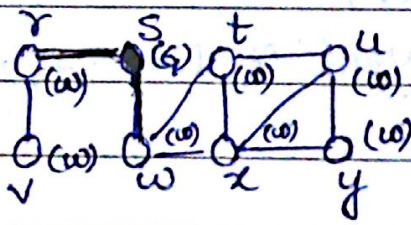
Example:

Graph:

Source: s.

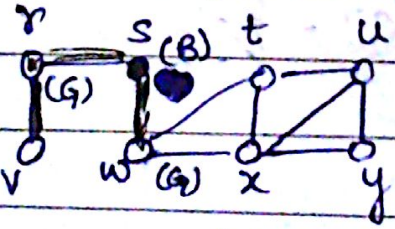


Step 1:



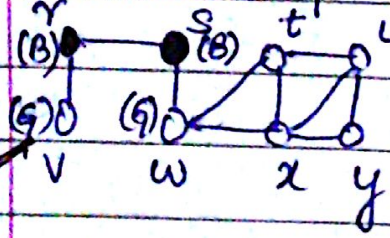
Q ⇒ [s] ⇒ dequeue ⇒ u = s.

Enqueue. Adj[s] = r, w.



Q = [r | w]
1 1

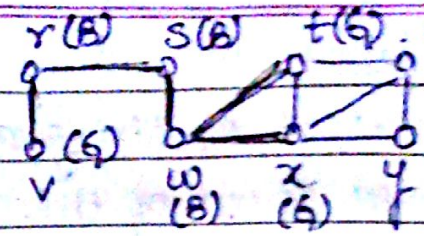
Step 2: u = r. (dequeue)



Enqueue Adj[r] = v.

Q = [w | v]
1 2

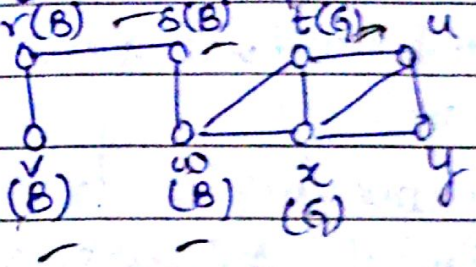
Step 3: dequeue. w ⇒ u = w.



Enqueue Adj[w] = t, x.

Q = | v | t | x |
2 2 2

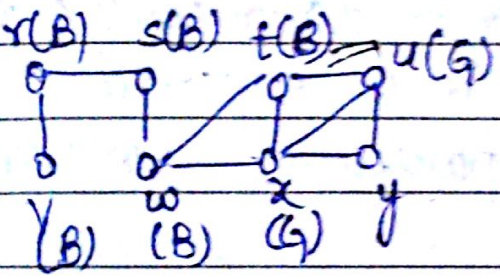
step4: dequeue. v ⇒ u = v



Enqueue
Adj[v] = Nil

Q = | t | x |
2 2

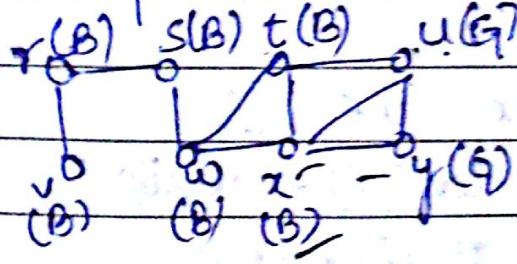
step5: dequeue. t ⇒ u = t.



Enqueue.
Adj[t] = u.

Q = | x | u |
2 3

Step6: dequeue x ⇒ u = x



Enqueue
Adj[x] = y.

Q = | u | y |
3 3

step7: dequeue u ⇒ u = u.

Enqueue Adj[u] = Nil.

Q = | y |
3

step8: dequeue y ⇒ u = y.

Enqueue Adj[y] = Nil

Q = ϕ

r w v t x u y
1 1 2 2 2 3 3

Complexity of Adjacency list = O(E)

18/03/2018

DELTA Page No. Date:

DEPTH FIRST SEARCH

1) DFS searches in depth of the graph, edges
2) Edges are explored out of the most recently discovered vertice v that still has unexplored vertices leaving it.
3) It also maintain a time stamp for every vertex i.e.
   d(v) => Discovering time of v
   f(v) = finishing time of v
4) Every vertex is initially white, it becomes gray when it is discovered & than black when it is explored
5) $\pi(v)$ store parent of v or predecessor of v

ALGO :

DFS(G)

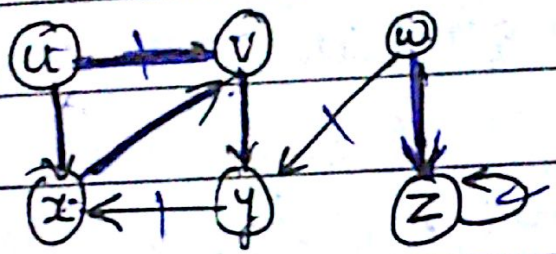
1. for each vertex $u \in V[G]$
2. do color[u] ← white
3. $\pi[u]$ ← Nil
4. time ← 0.
5. for each vertex $v \in V[G]$.
6. do if color[u] = white
7. then DFS_VISIT(u)

DFS_VISIT(u)

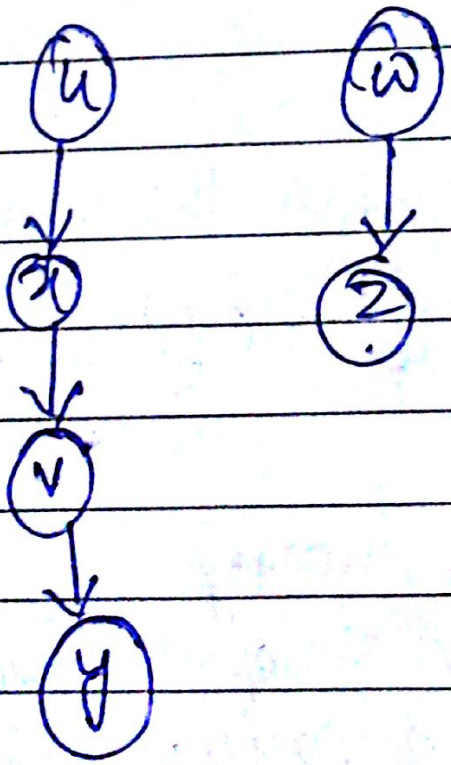
1. Color[u] ← Gray
2. time ← time + 1.

3. d[u] ← time.
4. for each v ∈ Adj[u].
5. do if color[v] = White
6. then π[v] ← u.
7. DFS_VISIT(v)
8. color[u] ← BLACK.
9. f[u] ← time ← time + 1

Eg :-



Vertex	π[u]	D	F
u	Nil	1	8
x	u	2	7
v	x	3	6
y	v	4	5
w	Nil	9	12
z	w	10	11



30/03/2016

for n-vertices
total spanning tree $\Rightarrow n^{n-1}$

Page No.
Date:

Running Time of DFS

$\Rightarrow \Theta(E+V)$

MINIMUM SPANNING TREE.

(*) Generic algorithm for minimum spanning tree.

```
GENERIC_MST(G, W)
1. A ← ϕ
2. While A does not form a spanning tree
3.     do find an edge (u,v) that is
           safe for A.
4.     A ← A ∪ {(u,v)}
5. return A.
```

A ← set having all the vertices of S.T.

safe edge ← do it form a cycle or not
(cycle → not safe, not cycle → safe)

(*) Cut :-
A cut $(S, V-S)$ of an undirected graph $G=(V,E)$ is a partition of V.

(*) Edge crossing a cut :
An edge (u,v) cross the cut $(S, V-S)$

(*) Light Edge :
An edge is light edge crossing a cut if its weight is minimum of any edge crossing the cut.

(*) Cut Respect :
A cut respect to set A, if no edge in A cross the cut

(*) Algorithm for Kruskal's for minimum Spanning Tree A

MST-KRUSKAL(G,W)

1. A ← ϕ
2. for each vertex v ∈ V[G]
3. do MAKE_SET(v)
4. Sort the edges of E into non-decreasing order by weight w.
5. for each edge (u,v) ∈ E, taken ← do →
6. do if FIND-SET(u) ≠ FIND-SET(v).
7. then A ← A ∪ {u,v}
8. UNION(u,v)
9. Return A.

Complexity = O(E+V) log E

No of edges = No. of vertices - 1

$|E| \geq |V| - 1$.

02/04/2013

Kruskal → forest
Prim → tree

DELTA Page No. Date:

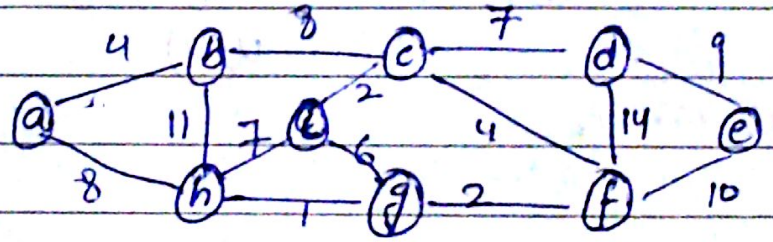
PRIM's ALGO.

MST_PRIM(G, w, r)

1. for each u ∈ V[G]
2. do key[u] ← ∞
3. π[u] ← NIL
4. key[r] ← 0
5. Q ← V[G]
6. while Q ≠ ϕ.
7. do u ← EXTRACT_MIN(Q)
8. for each v ∈ Adj[u]
9. do if v ∈ Q and w[u,v] < key[v]
10. then π[v] ← u
11. key[v] ← w(u,v)

O|V| ← [1–3]
O|V| ← [5]
O|V| ← [6]
O(V log V) ← [7]
O|E| ← [8]
[9–10]
O(E log V) ← [11]

Ques



step1:	a	b	c	d	e	f	g	h	i
key:	0	∞	∞	∞	∞	∞	∞	∞	∞

u ← a. Now v ∈ {b, h}

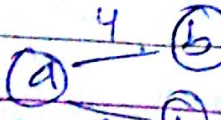
Now h = w(a,b) < ∞

& w(a,h) < ∞

∴ b is deleted from Q & key of b with & π[b] = a

Similarly for h. → key = 8 & π[h] = a

Graph:-



	a	b	c	d	e	f	g	h	i
key	0	4	∞	∞	∞	∞	∞	8	∞

step 2 Now, $u \leftarrow b$ & $v \in \{h, c\}$

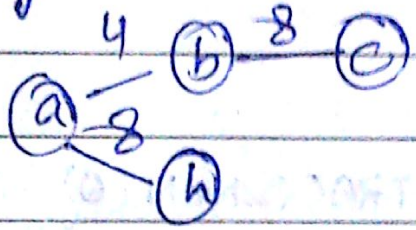
Now. $key(h) = 8 < w(b,h) = 11$

∴ no change

& $key(c) = \infty > w(b,c) = 8$

∴ updated.

	a	b	c	d	e	f	g	h	i
key	0	4	8	∞	∞	∞	∞	8	∞



step 3

Now $u \leftarrow c$. & $v \in \{i, f, d\}$.

Now $key(i) = \infty < w(c,i) = 2$

∴ update

$key(f) = \infty < w(c,f) = 4$

∴ update

$key(d) = \infty < w(c,d) = 7$

	a	b	c	d	e	f	g	h	i
key =	0	4	8	7	∞	4	∞	8	2

step 4

E7/V.

4/4/13

Complexity of Prim's Algorithm

$O|V| + O|V| + O|V| + O|V \log V| + O|E| + O|V| + O|E \log V|$

$\Rightarrow O[E \log V]$

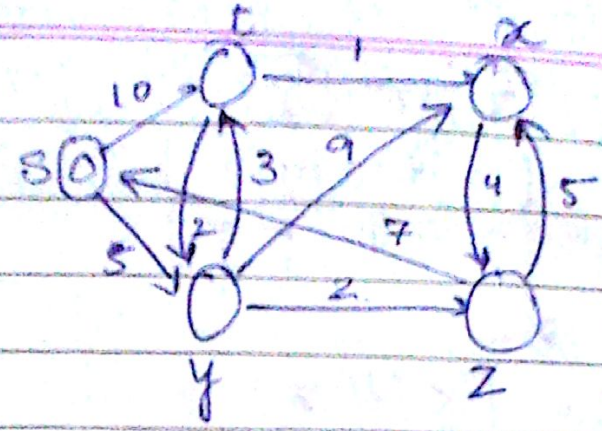
⑧ DIJKSTRA'S ALGORITHM.

```
DIJKSTRA (G, S, ω).
1. INITIALIZE_SINGLE_SOURCE (G, S)
2. S ← ϕ
3. Q ← V[G].
4. while. Q ≠ ϕ.
5.        do u ← EXTRACT_MIN (Q)
6.           S ← S U {u}.
7.           for each vertex. v ∈ Adj[u]
8.  do RELAX (u, v, ω)
```

```
RELAX (u, v, ω)
1. if d[v] > d[u] + ω[u, v]
2.        then d[v] ← d[u] + ω(u, v)
3.             π[v] ← u.
```

```
INITALIZE_SINGLE-SOURCE (G, S)
1. for each vertex v ∈ V[G]
2.        do d[v] ← ∞
3.           π[v] ← NIL
4. d[s] ← 0
```

Eg:-



Step 1:-

v	s	t	x	y	z
d	0	∞	∞	∞	∞

Q = [s | t | x | y | z]

u ← s s is deleted from Q.

S = [s]

v ∈ {t, y}

for t ⇒ d[t] = 0 + 10 = 10

for y ⇒ d[y] = 0 + 5 = 5.

π	s	t	x	y	z
	s		s		

Step 2:-

Now Q = [t | x | y | z]

v	s	t	x	y	z
d	0	10	∞	5	∞

u ← y.

S = [s | y]

v ∈ {t, z, x}

for t ⇒ d[t] = 5 + 3 = 8

for z ⇒ d[z] = 5 + 2 = 7

x ⇒ d[x] = 5 + 9 = 14

π	s	t	x	y	z
		y	y	s	y

Step 3:- Q = [t | x | z]

v	s	t	x	y	z
d	0	8	14	5	7

u ← x

S = [s | y | z]

v ∈ {s, z}

for z ⇒ d[z] = 7 < 14 ∴ No change

π	s	t	x	y	z
	y	y	s	y	

Step 4:- for s ⇒ 0 < 7 + 7. No change

for x ⇒ 14 > 7 + 5

∴ d[x] = 12.

π	s	t	x	y	z
	y	z	s	y	

Step 4:- Q = [t | x]

v	s	t	x	y	z
d	0	8	12	5	7

u ← t

S = [s | y | z | t]

v ∈ {x, y}

for x ⇒ 12 > 8 + 1 ⇒ d[x] = 9

y ⇒ 5 < 8 + 2 No change

π	s	t	x	y	z
	y	t	s	y	

Step 5:- Q = [x]

v	s	t	x	y	z
d	0	8	9	5	7

v ∈ {z}

for z ⇒ 7 < 9+4 No change.

BELLMAN-FORD ALGO

```
BELLMAN-FORD(G, S, W)
1. INITIALIZE-SINGLE-SOURCE(G, S)
2. for i ← 1 to V[G] - 1
3.     do for each edge(u,v) ∈ E[G]
4.         do RELAX(u,v,w)
5. for each edge(u,v) ∈ E[G]
6.     do if d[v] > d[u] + w(u,v)
7.         Return FALSE
8. Return true
```

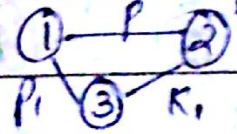
5/4/13

FLOYD. WARSHALL. ALGO. → dynamic Programming.

1) Form shortest path b/w vertices
(structure of shortest path)

→ Intermediate vertice

→ If there exists a intermediate vertix k to a path P from i to j then we break P into 2 shortest path P_1 & P_2



~~P~~ P_1 reside b/w i — k & P_2 b/w j - k.

If k is not the intermediate vertice of path P then all the intermediate vertices of path P will reside b/w 1, 2 .... k-1.

2) Form the recursive solution for the shortest path.

$d_{ij}(k)$ → weight of path i to j via k.

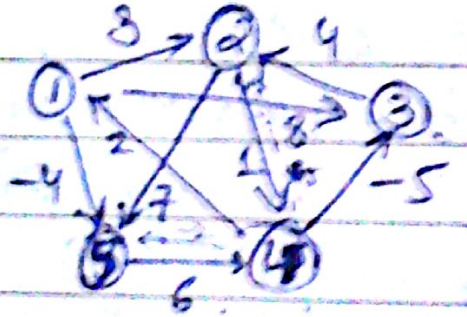
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1 \end{cases}$$

* Algorithm. (Complexity = $O(V^3)$)

FLOYD-WARSHALL(W)

1. $n \leftarrow$ rows[W]
2. $D^{(0)} \leftarrow W$.
3. for $k \leftarrow 1$ to n
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. do $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7. return $D^{(n)}$

Eg:-



$$D^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(0)} = \begin{bmatrix} \phi & 1 & 1 & \phi & 1 \\ \phi & \phi & \phi & 2 & 2 \\ \phi & 3 & \phi & \phi & \phi \\ 4 & \phi & 4 & \phi & \phi \\ \phi & \phi & \phi & 5 & \phi \end{bmatrix}$$

$$D^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(1)} = \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ 4 & 1 & 4 & \phi & 1 \\ & & & & \end{bmatrix}$$

Page No.
Date:

(A) Parent Matrix

$$\pi_{ij}^{(0)} = \begin{cases} Nil & \text{if } i=j \text{ or } w_{ij}=\infty \\ i & \text{if } i\neq j \text{ and } w_{ij}<\infty \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{jk}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Recursive Solution :

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i=j \\ \infty & \text{if } i\neq j. \end{cases}$$

$$l_{ij}^{(m)} = \min\left(l_{ij}^{(m-1)}, \min_{1\leq k\leq n}\{l_{ik}^{(m-1)} + w_{kj}\}\right)$$

Algo

EXTENDED-SHORTEST-PATH(L, w)

1. $n \leftarrow$ rows$[L]$
2. let $L' = (l_{ij}')$ be an $n \times n$ matrix
3. for $i \leftarrow 1$ to n
4. for $j \leftarrow 1$ to n
5. do $l_{ij}' \leftarrow \infty$
6. for $k \leftarrow 1$ to n
7. do $l_{ij}' \leftarrow \min(l_{ij}', l_{ik} + w_{kj})$
8. return L'

TOPOLOGICAL SORT

TOPOLOGICAL_SORT(G)

1. Call DFS(G) to compute finishing time $f[v]$ for each vertex V
2. As each vertex is finished Insert it into front of a link list //Insert vertex in decreasing order of finish time
3. Return the link list of vertex

Definations

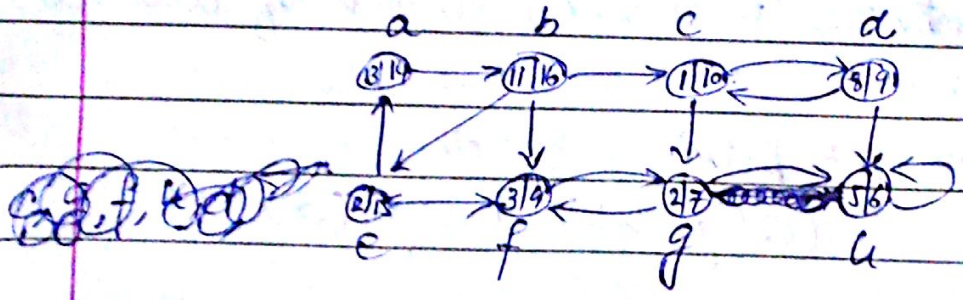
A topological sort of a directed acyclic graph (If graph is not acyclic then topological sort of graph will not exist), $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v. In the ordering of vertices on the horizontal line. All the directed edges of the graph will go from left to right on the horizontal line.

Strongly Connected Components of Graph

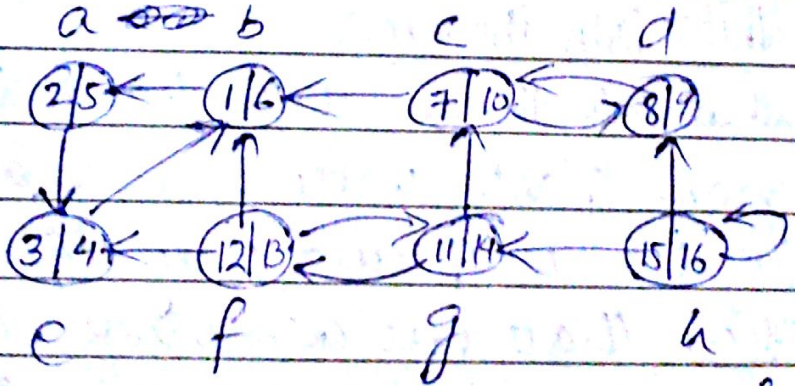
Strongly connected components of directed graph G equals to V, E is a maximal set of vertices. C which is subset of vertex $C \subseteq V$ set V such that for every pair of vertex (u, V) in C, we have both $u \rightsquigarrow v$ & $v \rightsquigarrow u$, i.e. vertex u & v are reachable from each other

STRONGLY-CONNECTED-COMPONENT(G).

1. Call DFS(G) to compute finishing times $f[v]$ for every vertex v.
2. Compute Graph transpoition, G^T.
3. Call DFS(G^T), but in main loop of consider the vertices in order of decreasing $f[v]$ (as computed in line 1)
4. Output the vertex of each tree in DFS ~~tree~~ forest formed in line 3 as a seprate strongly connected component.



h, abe, cd, fg



h, fg, cd, abe

↓

a b e cd f g h.

Strongly connected

Parantheis theorm:

In DFS of a directed or undirected graph, G(V,E) for any two vertices (u,v) one of the following three conditions holds.

i) The intervals [d[u], f[u]] and [d[v], f[v]] are entirely disjoint & neither u nor v is a descendd of each other

ii) The interval [d[u], f[u]] is contained entirely within the interval [d[v], f[v]] and u is a descendded of v in the DFS tree.

iii) The interval [d(v), f[v]] is contained entirely within the interval [d[u], f[u]] and v is a descended of u in DFS tree

for eg :- Previous graph
(c(g(uu)(d d)g)(dd)c)

White Path theorm.:

White Path theorm states that in DFS of a graph G(V,E) vertex v, is descend of vertex u if and only if at the time d[u] that the discovers u, the vertex v can be reached from u along a path consisting entirely of white vertices

String Matching Algorithm

4 Algorithm, order is considered.

2 strings ⇒ main string : T, substring P is checked in main string

(file of character)

(#) Text is a ~~file~~ the array is represented by T[1....n] of length n and the pattern in the array is represented by P[1...m] of length m where m is less than equals to n ($m \leq n$)

(#) P occurs with a shift s in the text T and the s is a valid shift if it occurs b/w 0 to n-m [$s>0$ & $s<n-m$] ∴ string matching problem is a problem of finding all valid shifts with which a pattern P occurs in a given text T

~~Name~~

Naive_stringmatching_algo:

```
NAIVE-STRING-MATCHER (T, P)
1. n ← length[T]
2. m ← length[P]
3. for s ← 0 to n-m
4.     do if P[1...m] = T[s+1...s+m]
           then print Pattern occurs with shift s.
```

Eg: T = [a|c|a|a|b|c] P = [a|a|b]

The complexity of this algo is O(n-m+1)

⑧ Rabin_Karp_Algorithm

We use mod of any prime number as

$h \equiv p \bmod q$ ← (prime number)

We use hashing function (w) to find matched string

RABIN-KARP-~~MATCHER~~ MATCHER (T, p, d, q)

1. $n \leftarrow$ length $[T]$
2. $m \leftarrow$ length $[P]$
3. $h \leftarrow d^{m-1} \bmod q$
4. $p \leftarrow 0$
5. $t_0 \leftarrow 0$
6. for $i \leftarrow 1$ to n
7. do $p \leftarrow (dp + P[i]) \bmod q$
8. $t_s \leftarrow (dt_0 + T[i]) \bmod q$
9. for $s \leftarrow 0$ to $n-m$
10. do if $p = t_s$.
11. then if $P[1 \ldots m] = T[s+1 \ldots s+m]$
12. then print "Pattern occurs with shift s"
13. If $s < n-m$.
14. then $t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$

Complexity $\Rightarrow O(n+m)$

(4) String Matching with finite Automata

A finite Automata. M is a 5-tuple $(Q, q_0, A, \Sigma, \delta)$ where.

Q is a finite set of states.

q_0 which belong to Q is a start state

A is subset of Q is a distinguished set of accepting states.

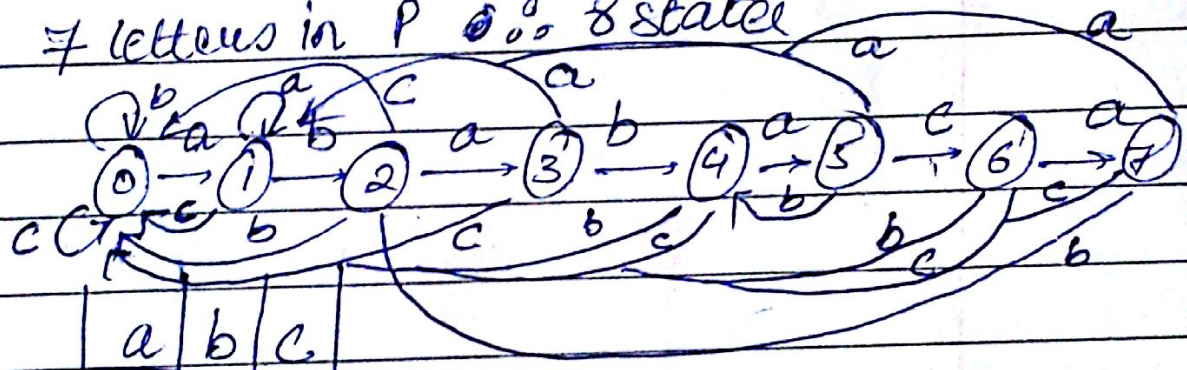
Σ is a finite input alphabet.

δ is transition function

Eg: T := abababacaba

P := ababaca

7 letters in P ∴ 8 states



	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	4	0
4	5	0	0
5	1	4	6
6	7	0	0
7	1	2	0