APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

THIRD SEMESTER B.TECH DEGREE EXAMINATION, DECEMBER 2017

CS 301: THEORY OF COMPUTATION

(scheme of evaluation)

Max. Marks:100 Duration: 3 Hours

PART A

Answer all questions.

1. Define Deterministic Finite Automaton with suitable example. (3)

Definition

A finite automation, M is a 5 tuple structure,

$$M\ (Q,\ \sum,\ q_0,\ \delta,\ F\ )$$

where M is the finite automation,

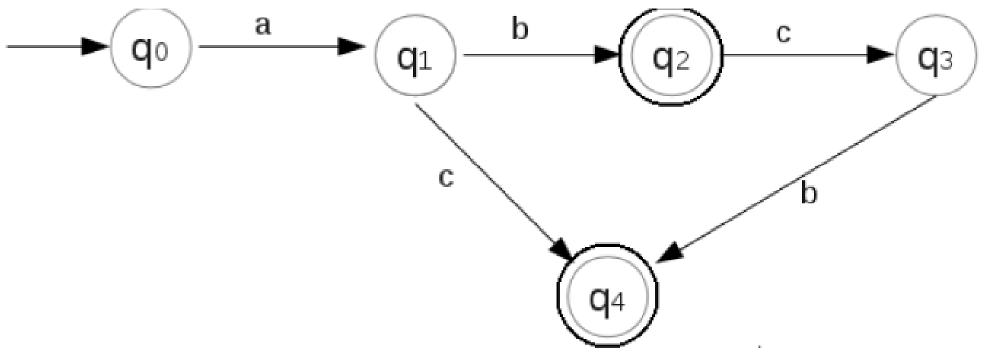
Q is a finite set of states,

$\sum$ is a set of input symbols,

q_0 is the start state,

δ is the set of transition functions,

F is the set of final states.



In the above finite automation, M,

1. $Q = \{q_0, q_1, q_2, q_3, q_4\}$

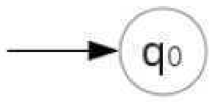
Q is the set of states in the finite automata. In the above finite automata, q_0, q_1, q_2, q_3, q_4 are the states. The states are shown in circles.

2. $\sum = a, b, c$

$\sum$ is the set of input symbols in the finite automata. In the above, a, b, c are the input symbols. The arrowsare labelled with input symbols.

3. $q_0 = \{q_0\}$

q_0 is the start state. In the above, q_0 is the start state. For our study, a finite automata contains only one start state. A state with arrow from free space (not coming from any other state) is designated as start state.



4. δ describes the operation of finite automation. δ is the transition function.

4. δ describes the operation of finite automation. δ is the transition function.

For example,

$$\delta(q_0, a) = q_1$$

This means, given the current state q_0 and the input symbol, a, the finite automation moves (transits) to state q_1.

Similarly,

$$\delta(q_1, b) = q_2$$
$$\delta(q_2, c) = q_3$$
$$\delta(q_1, c) = q_4$$
$$\delta(q_3, b) = q_4$$

5. $F = \{q_2, q_4\}$

F is the set of final states or accepting states. In the above finite automation, $q_2$and q_4 are the final states. They are shown enclosed in double circles.

2. Design a DFA which accept set of all strings which are divisible by 5 for binary alphabets. (3)

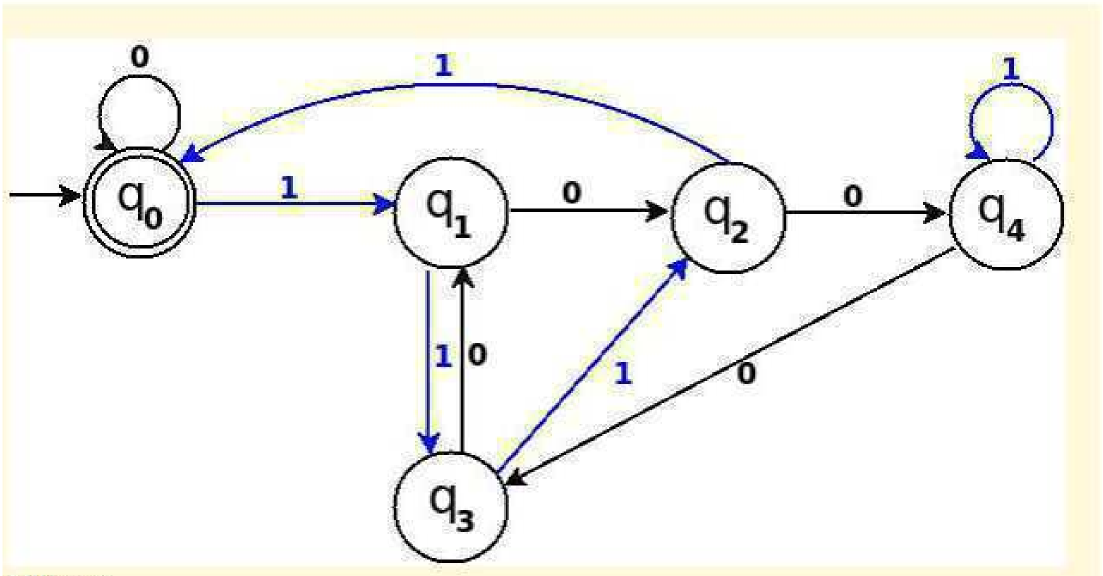


Figure-7

3. Define regular expression and rules of regular expression (3)

A **Regular Expression** can be recursively defined as follows −

- **ε** is a Regular Expression indicates the language containing an empty string. **(L (ε) = {ε})**
- **φ** is a Regular Expression denoting an empty language. **(L (φ) = { })**
- **x** is a Regular Expression where **L = {x}**
- If **X** is a Regular Expression denoting the language **L(X)** and **Y** is a Regular Expression denoting the language **L(Y)**, then
    - **X + Y** is a Regular Expression corresponding to the language **L(X) ∪ L(Y)** where **L(X+Y) = L(X) ∪ L(Y)**.
    - **X . Y** is a Regular Expression corresponding to the language **L(X) . L(Y)** where **L(X.Y) = L(X) . L(Y)**
    - **R*** is a Regular Expression corresponding to the language **L(R*)**where **L(R*) = (L(R))***
- If we apply any of the rules several times from 1 to 5, they are Regular Expressions.

4. b) Construct a Mealy machine that gives 2's complement of any binary input. (3)

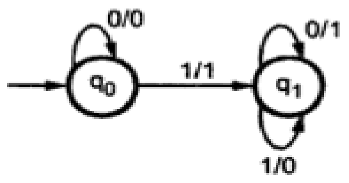
soln:

Read from LSB

Keep the first from LSB as it is and toggle the remaining bits we will get

0101

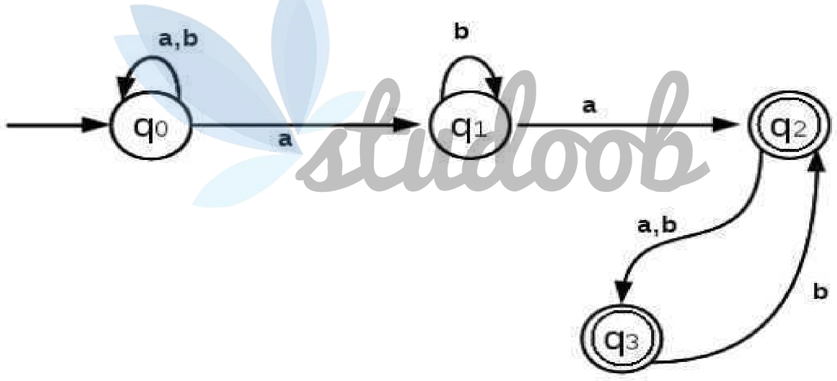
Thus 2's complement of 1011 is 0101. The required Mealy machine will be-



PART B

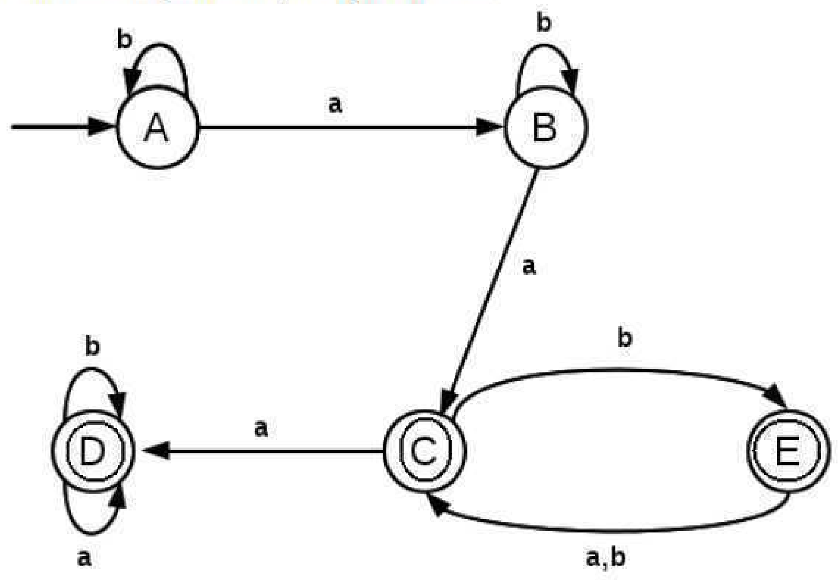
Answer any two questions.

5.a) Convert the following NFA to DFA (4.5)



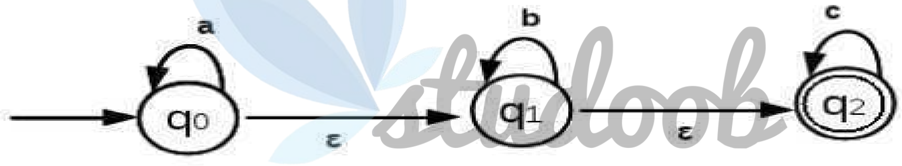
Soln:

The transition diagram corresponding to the DFA is,

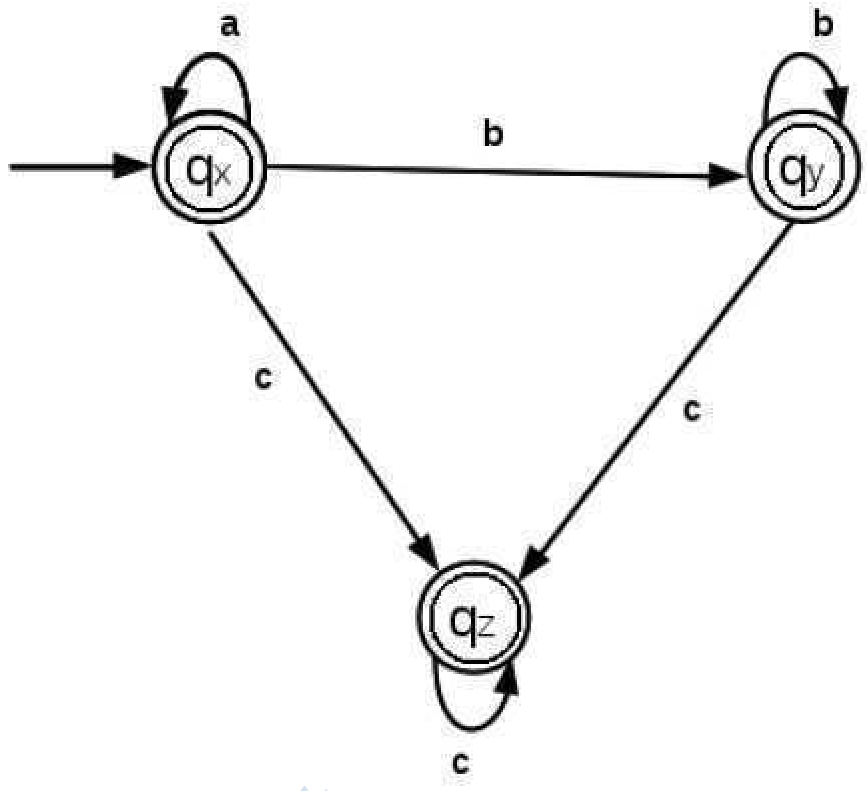


From the above diagram, it is a DFA since, there is only one transition corresponding to an input symbol from a state.

b) Convert the following ϵ- NFA to NFA without ϵ transition (4.5)



Soln:



Above is the NFA without epsilon transitions.

Note that above NFA is also a DFA.

6. Explain the steps to minimize DFA by My hill Nerode theorem and use this theorem to minimize the following DFA.. (9)

Current State	Input 0	symbol 1
$\longrightarrow q_0$	q_1	q_3
q_1	q_0	q_3
q_2	q_1	q_4
$*q_3$	q_5	q_5
q_4	q_3	q_3
$*q_5$	q_5	q_5

Soln:

Steps:

Input − DFA

Output − Minimized DFA

Step 1 − Draw a table for all pairs of states (Q_i, Q_j) not necessarily connected directly [All are unmarked initially]

Step 2 − Consider every state pair (Q_i, Q_j) in the DFA where $Q_i \in F$ and $Q_j \notin F$ or vice versa and mark them. [Here F is the set of final states]

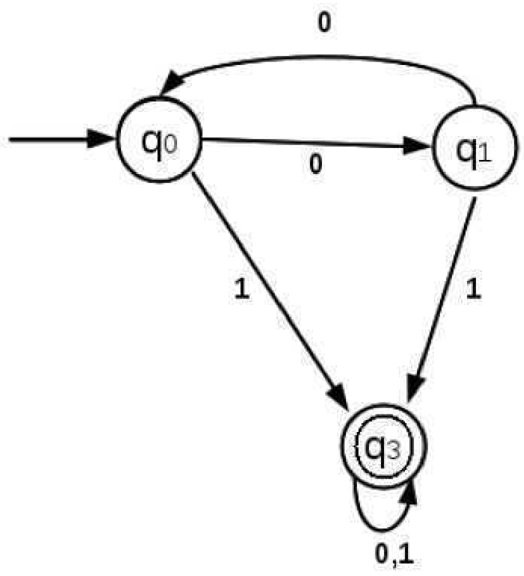
Step 3 − Repeat this step until we cannot mark anymore states −

If there is an unmarked pair (Q_i, Q_j), mark it if the pair {δ (Q_i, A), δ (Q_i, A)} is marked for some input alphabet.

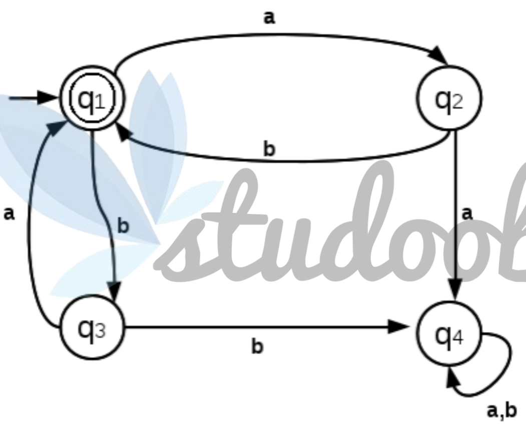
Step 4 − Combine all the unmarked pair (Q_i, Q_j) and make them a single state in the reduced DFA.

Current State	Input symbol	
	a	b
$\longrightarrow q_0$	q_1	q_3
q_1	q_0	q_3
$*q_3$	q_3	q_3

Now this is a minimised DFA.



7. a) Find regular expression corresponding to the following DFA (4)



Soln:

Here we write equations for every state.

We write,

$q_1 = q_2 b + q_3 a + \varepsilon$

The term $q_2 b$ because there is an arrow from q_2 to q_1 on input symbol b.

The term $q_3 a$ because there is an arrow from q_3 to q_1 on input symbol a.

The term ε because q_1 is the start state.

$q_2 = q_1 a$

The term $q_1 a$ because there is an arrow from q_1 to $q_2$on input symbol a.

$q_3 = q_1 b$

The term $q_1 b$ because there is an arrow from q_1 to $q_3$on input symbol b.

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$

The term $q_2 a$ because there is an arrow from q_2 to $q_4$on input symbol a.

The term $q_3 b$ because there is an arrow from q_3 to $q_4$on input symbol b.

The term $q_4 b$ because there is an arrow from q_4 to $q_4$on input symbol b.

The final state is q_1.

Putting $q_2$and q_3 in the first equation (corresponding to the final state), we get,

$$q_1 = q_1 ab + q_1 ba + \varepsilon$$

$$q_1 = q_1(ab + ba) + \varepsilon$$

$$q_1 = \varepsilon + q_1(ab + ba)$$

From Arden's theorem,

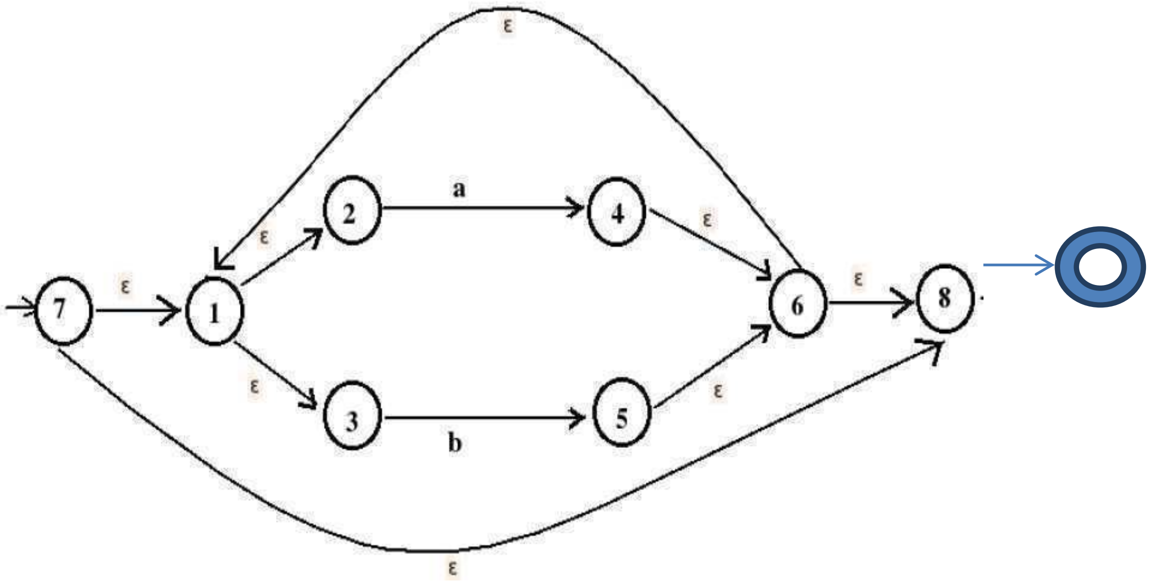
$$q_1 = \varepsilon(ab + ba)^*$$

$$q_1 = (ab + ba)^*$$

So the regular expression is, $((ab)/(ba))^*$

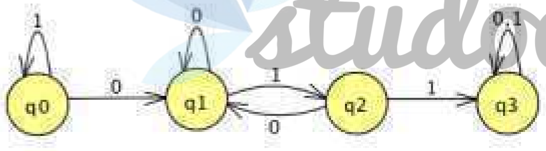
b) Draw NFA corresponding to the regular Expression (a+b)*c (3)

The NFA corresponding to $(a|b)^*a$ is



c) Design a DFA which accept set of all strings containing 011 as a substring (2)

soln:



PART C

Answer all questions.

9. Explain about the closure properties of CFL (3)

Ans:

Context-free languages are **closed** under –

- Union
- Concatenation
- Kleene Star operation

Union

Let L_1 and L_2 be two context free languages. Then $L_1 \cup L_2$ is also context free.

Example

Let $L_1 = \{ a^n b^n , n > 0\}$. Corresponding grammar G_1 will have P: $S1 \rightarrow aAb|ab$

Let $L_2 = \{ c^m d^m , m \geq 0\}$. Corresponding grammar G_2 will have P: $S2 \rightarrow cBb| \varepsilon$

Union of L_1 and L_2, $L = L_1 \cup L_2 = \{ a^n b^n \} \cup \{ c^m d^m \}$

The corresponding grammar G will have the additional production $S \rightarrow S1 \mid S2$

Concatenation

If L_1 and L_2 are context free languages, then L_1L_2 is also context free.

Example

Union of the languages L_1 and L_2, $L = L_1L_2 = \{ a^n b^n c^m d^m \}$

The corresponding grammar G will have the additional production $S \rightarrow S1\ S2$

Kleene Star

If L is a context free language, then L* is also context free.

Example

Let $L = \{ a^n b^n , n \geq 0\}$. Corresponding grammar G will have P: $S \rightarrow aAb| \varepsilon$

Kleene Star $L_1 = \{ a^n b^n \}^*$

The corresponding grammar G_1 will have additional productions $S1 \rightarrow SS_1 \mid \varepsilon$

Context-free languages are **not closed** under −

- **Intersection** − If L1 and L2 are context free languages, then $L1 \cap L2$ is not necessarily context free.
- **Intersection with Regular Language** − If L1 is a regular language and L2 is a context free language, then $L1 \cap L2$ is a context free language.
- **Complement** − If L1 is a context free language, then L1’ may not be context free.

10. Define Push Down Automata (3)

A pushdown automaton is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

Basically a pushdown automaton is −

"Finite state machine" + "a stack"

A pushdown automaton has three components −

- an input tape,
- a control unit, and
- a stack with infinite size.

The stack head scans the top symbol of the stack.

A stack does two operations −

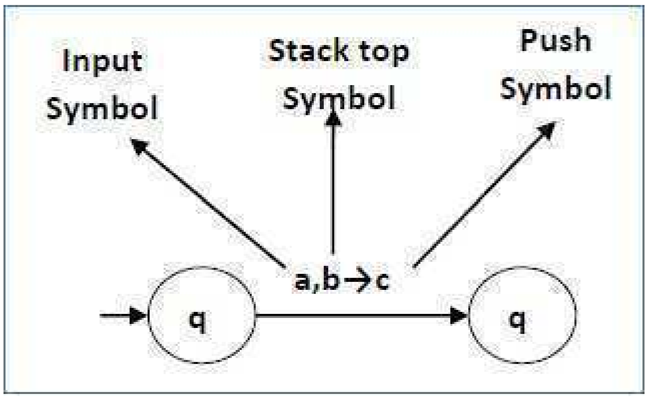
- **Push** − a new symbol is added at the top.
- **Pop** − the top symbol is read and removed.

A PDA may or may not read an input symbol, but it has to read the top of the stack in every transition.

A PDA can be formally described as a 7-tuple (Q, ∑, S, δ, q_0, I, F) −

- **Q** is the finite number of states
- ∑ is input alphabet
- **S** is stack symbols
- **δ** is the transition function: Q × (∑ ∪ {ε}) × S × Q × S*
- **q_0** is the initial state (q_0 ∈ Q)
- **I** is the initial stack top symbol (I ∈ S)
- **F** is a set of accepting states (F ∈ Q)

The following diagram shows a transition in a PDA from a state q_1 to state q_2, labeled as a,b → c −



This means at state $\mathbf{q_1}$, if we encounter an input string **`a'** and top symbol of the stack is **`b'**, then we pop **`b'**, push **`c'** on top of the stack and move to state $\mathbf{q_2}$.

11. Prove that L = { $a^n b^n$ / n >=1 } is not regular by applying Pumping Lemma for Regular Languages (3)

Soln:

The set of strings in this language are,

{ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, ...}

Consider one string in this language, aaabbb.

Let w=aaabbb.

Break w into three parts, x, y and z.

Let w=aaabbb=xyz

That is

w =	a	aa	bbb
	x	y	z

Then,

xy^iz is

Let i=1,

we have, xy^1z

w =	a	aa	bbb
	x	y^1	z

xy^1z=aaabbb =a^3b^3 is in the above language.

Let i=2,

we have, xy^2z

w =

a	aaaa	bbb
x	y^2	z

xy^2z=aaaaabbb =a^5b^3 is not in the above language.

Since some of the strings of the form xy^iz , are not in the above language, the language a^nb^n is not regular.

12 .Eliminate epsilon production from the CFG given below (3)

$$S \longrightarrow ABAC$$
$$A \longrightarrow aA|\varepsilon$$
$$B \longrightarrow bB|\varepsilon$$
$$C \longrightarrow c$$

Soln:

Eliminate epsilon productions from this grammar.

This CFG contains the epsilon productions, $A \longrightarrow \varepsilon$, $B \longrightarrow \varepsilon$.

To eliminate $A \longrightarrow \varepsilon$, replace A with epsion in the RHS of the productions, $S \longrightarrow ABAC$, $A \longrightarrow aA$,

For the production, $S \longrightarrow ABAC$, replace A with epsilon one by one as,

we get,

$S \longrightarrow BAC$

$S \longrightarrow ABC$

$S \longrightarrow BC$

For the production, $A \longrightarrow aA$, we get,

$A \longrightarrow a$,

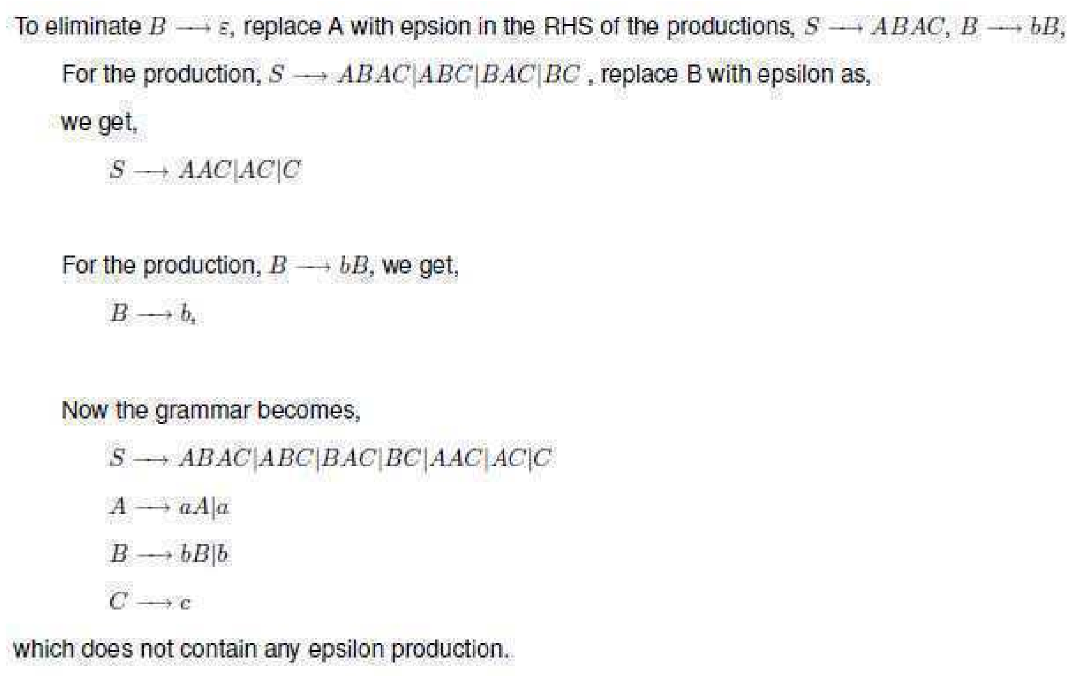
Now the grammar becomes,

$S \longrightarrow ABAC|ABC|BAC|BC$

$A \longrightarrow aA|a$

$B \longrightarrow bB|\varepsilon$

$C \longrightarrow c$



To eliminate $B \longrightarrow \varepsilon$, replace A with epsion in the RHS of the productions, $S \longrightarrow ABAC$, $B \longrightarrow bB$,

For the production, $S \longrightarrow ABAC|ABC|BAC|BC$, replace B with epsilon as,

we get,

$S \longrightarrow AAC|AC|C$

For the production, $B \longrightarrow bB$, we get,

$B \longrightarrow b$,

Now the grammar becomes,

$S \longrightarrow ABAC|ABC|BAC|BC|AAC|AC|C$

$A \longrightarrow aA|a$

$B \longrightarrow bB|b$

$C \longrightarrow c$

which does not contain any epsilon production.

PART D

Answer any two questions.

12.a) Write an algorithm to convert given CFG into GNF (3)

Ans:

Step 1 − If the start symbol **S** occurs on some right side, create a new start symbol **S'** and a new production **S' → S**.

Step 2 − Remove Null productions. (Using the Null production removal algorithm discussed earlier)

Step 3 − Remove unit productions. (Using the Unit production removal algorithm discussed earlier)

Step 4 − Remove all direct and indirect left-recursion.

Step 5 − Do proper substitutions of productions to convert it into the proper form of GNF.

b) Convert the given CFG into GNF (6)

$$S \longrightarrow AA|a$$

$$A \longrightarrow SS|b$$

Soln:

Step 1:

Above CFG is in CNF.

Rename the variables, that is,

Rename S, A by A_1, A_2.

Then the grammar becomes,

$A_1 \longrightarrow A_2A_2|a$

$A_2 \longrightarrow A_1A_1|b$

Step 2: In this, productions must be of the form that

the RHS of a production must start with a terminal followed by nonterminals, or

the RHS of a production starts with a higher numbered variable.

The productions,

$A_1 \longrightarrow A_2A_2|a$

$A_2 \longrightarrow b$

satisfy this criterion.

Consider the production,

$A_2 \longrightarrow A_1A_1$

Replace the first A_1 on the RHS by its production.

Then,

$A_2 \longrightarrow A_2A_2A_1|aA_1$

$A_1 \longrightarrow A_2A_2|a$

$A_2 \longrightarrow A_2A_2A_1|aA_1|b$

The production, $A_2 \longrightarrow A_2A_2A_1|aA_1|b$ contains left recursion.

To eliminate left recursion, this is written as,

$A_2 \longrightarrow aA_1|b|aA_1Z|bZ$

$Z \longrightarrow A_2A_1|A_2A_1Z$

Now the CFG is,

$A_1 \longrightarrow A_2A_2|a$

$A_2 \longrightarrow aA_1|b|aA_1Z|bZ$

$Z \longrightarrow A_2A_1|A_2A_1Z$

Consider the production, $A_1 \longrightarrow A_2A_2$

Replace first A_2 on the RHS with its production, we get,

$A_1 \longrightarrow aA_1A_2|bA_2|aA_1ZA_2|bZA_2$

Now the CFG is

$A_1 \longrightarrow aA_1A_2|bA_2|aA_1ZA_2|bZA_2$

$A_2 \longrightarrow aA_1|b|aA_1Z|bZ$

$Z \longrightarrow A_2A_1|A_2A_1Z$

Consider the production, $Z \longrightarrow A_2A_1|A_2A_1Z$

Replace first A_2 on the RHS with its production,

$Z \longrightarrow aA_1A_1|aA_1A_1Z|bA_1|bA_1Z|aA_1ZA_1|aA_1ZA_1Z|bZA_1|bZA_1Z$

Now the CFG is,

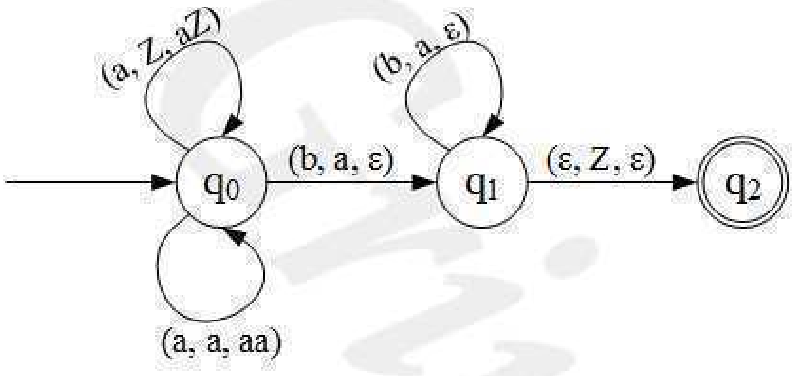
$A_1 \longrightarrow aA_1A_2|bA_2|aA_1ZA_2|bZA_2$

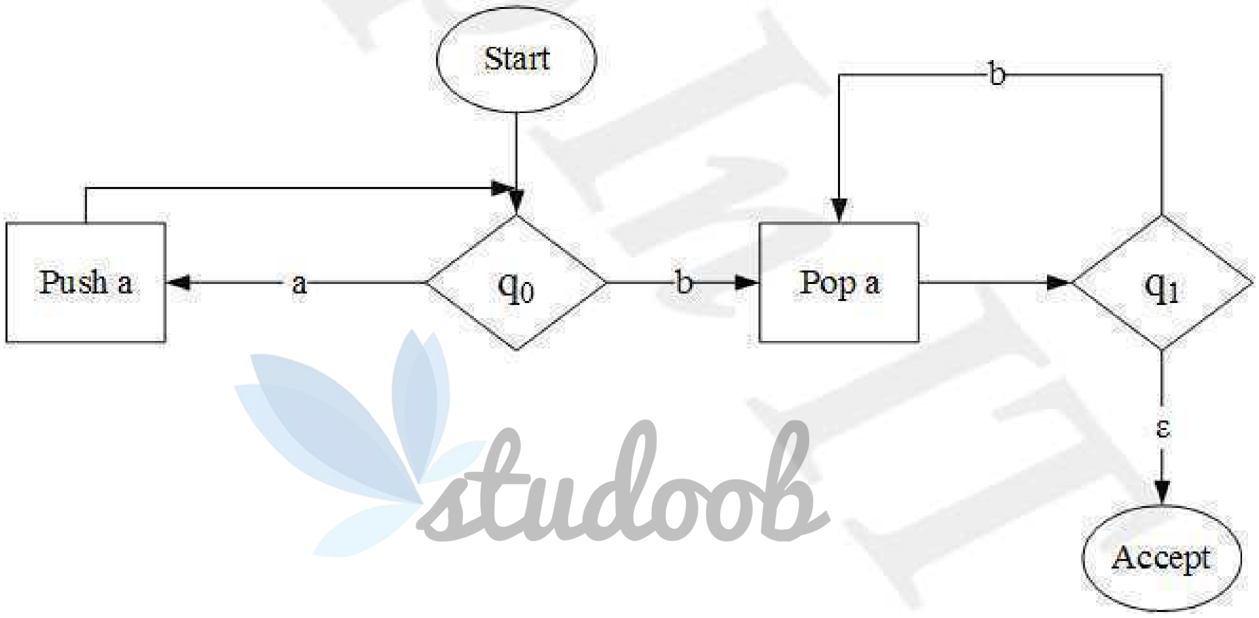
$A_2 \longrightarrow aA_1|b|aA_1Z|bZ$

$Z \longrightarrow aA_1A_1|aA_1A_1Z|bA_1|bA_1Z|aA_1ZA_1|aA_1ZA_1Z|bZA_1|bZA_1Z$

Above CFG is in GNF.

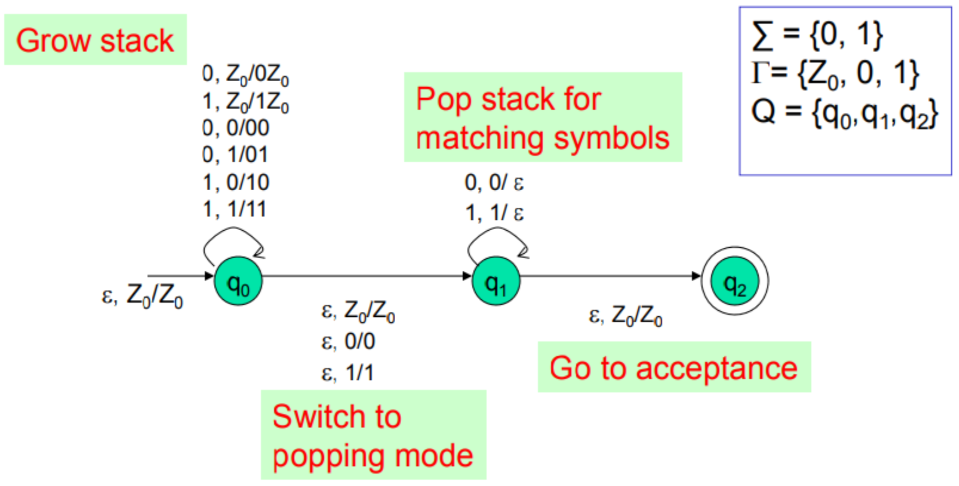
13.a) Construct PDA which accepts L= $\{a^n b^n / n>=1\}$ (4.5)





b) Design a PDA which accepts L = { ww^R / w in (0+1)* } (4.5)

soln:



14.a) State pumping lemma for regular languages (3)

Soln:

The pumping lemma states that for a regular language L, there exists a constant n such that every string w in L (such that length of w (|w|) >= n) we can break w into three substrings, w = xyz, such that

- y is not null
- |xy| <= n
- For all i >= 0, the string xy^iz is also in L

Note that |x| and |z| can be 0 but |y| has to be greater than 1.
The lemma states that for a regular language every string can be broken into three parts x,y and z such that if we repeat y i times between x and z then the resulting string is also present in L.

Applications of the Pumping Lemma
The pumping lemma is extremely useful in proving that certain sets are non-regular. The general methodology followed during its applications is :

1. Select a string z in the language L.
2. Break the string z into x, y and z in accordance with the above conditions imposed by the pumping lemma.
3. Now check if there is any contradiction to the pumping lemma for any value of i.

It is suggested that you try out the questions before looking at the solutions.

b) Explain about acceptance of PDA (3)

ans:

There are two different ways to define PDA acceptability.

Final State Acceptability

In final state acceptability, a PDA accepts a string when, after reading the entire string, the PDA is in a final state. From the starting state, we can make moves that end up in a final state with any stack values. The stack values are irrelevant as long as we end up in a final state.For a PDA (Q, ∑, S, δ, q_0, I, F), the language accepted by the set of final states F is −

$L(PDA) = \{w \mid (q_0, w, I) \vdash^* (q, \varepsilon, x), q \in F\}$

for any input stack string **x**.

Empty Stack Acceptability

Here a PDA accepts a string when, after reading the entire string, the PDA has emptied its stack.

For a PDA (Q, ∑, S, δ, q_0, I, F), the language accepted by the empty stack is −

$L(PDA) = \{w \mid (q_0, w, I) \vdash^* (q, \varepsilon, \varepsilon), q \in Q\}$

c) Write an algorithm to convert given CFG to CNF (3)

ans:

A CFG is in Chomsky Normal Form if the Productions are in the following forms −

- A → a
- A → BC
- S → ε

where A, B, and C are non-terminals and **a** is terminal.

Algorithm to Convert into Chomsky Normal Form −

Step 1 − If the start symbol **S** occurs on some right side, create a new start symbol **S'** and a new production **S'→ S**.

Step 2 − Remove Null productions. (Using the Null production removal algorithm discussed earlier)

Step 3 − Remove unit productions. (Using the Unit production removal algorithm discussed earlier)

Step 4 − Replace each production **A → $B_1 \ldots B_n$** where **n > 2** with **A → B_1C**where **C → $B_2 \ldots B_n$**. Repeat this step for all productions having two or more symbols in the right side.

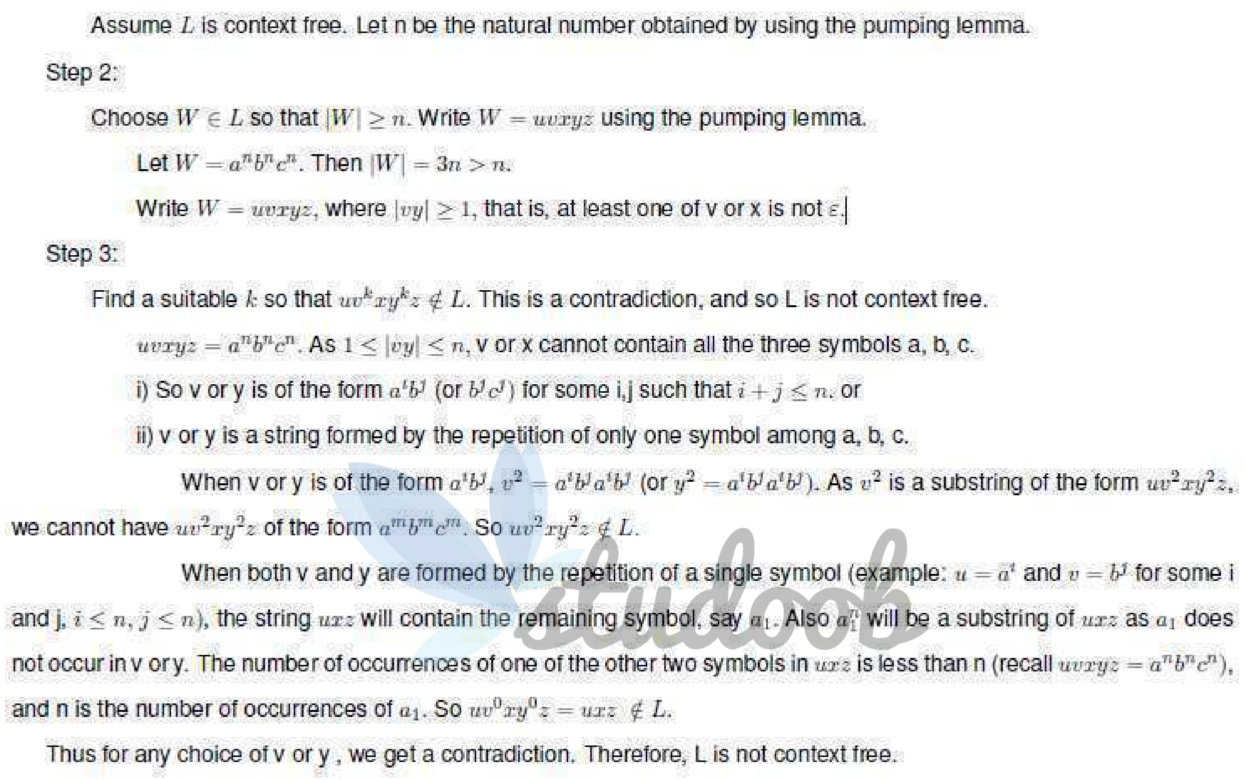
Step 5 − If the right side of any production is in the form **A → aB** where a is a terminal and **A, B** are non-terminal, then the production is replaced by **A → XB** and **X → a**. Repeat this step for every production which is in the form **A → aB**.

PART E

Answer any four questions.

15. a) Show that the language L= $\{a^n b^n c^n$ /n>=1} is not Context Free Language (5)

Ans:



Assume L is context free. Let n be the natural number obtained by using the pumping lemma.

Step 2:

Choose $W \in L$ so that $|W| \geq n$. Write $W = uvxyz$ using the pumping lemma.

Let $W = a^n b^n c^n$. Then $|W| = 3n > n$.

Write $W = uvxyz$, where $|vy| \geq 1$, that is, at least one of v or x is not ε.

Step 3:

Find a suitable k so that $uv^k xy^k z \notin L$. This is a contradiction, and so L is not context free.

$uvxyz = a^n b^n c^n$. As $1 \leq |vy| \leq n$, v or x cannot contain all the three symbols a, b, c.

i) So v or y is of the form $a^i b^j$ (or $b^j c^j$) for some i,j such that $i + j \leq n$. or

ii) v or y is a string formed by the repetition of only one symbol among a, b, c.

When v or y is of the form $a^i b^j$, $v^2 = a^i b^j a^i b^j$ (or $y^2 = a^i b^j a^i b^j$). As v^2 is a substring of the form $uv^2 xy^2 z$, we cannot have $uv^2 xy^2 z$ of the form $a^m b^m c^m$. So $uv^2 xy^2 z \notin L$.

When both v and y are formed by the repetition of a single symbol (example: $u = a^i$ and $v = b^j$ for some i and j, $i \leq n, j \leq n$), the string uxz will contain the remaining symbol, say a_1. Also a_1^n will be a substring of uxz as a_1 does not occur in v or y. The number of occurrences of one of the other two symbols in uxz is less than n (recall $uvxyz = a^n b^n c^n$), and n is the number of occurrences of a_1. So $uv^0 xy^0 z = uxz \notin L$.

Thus for any choice of v or y , we get a contradiction. Therefore, L is not context free.

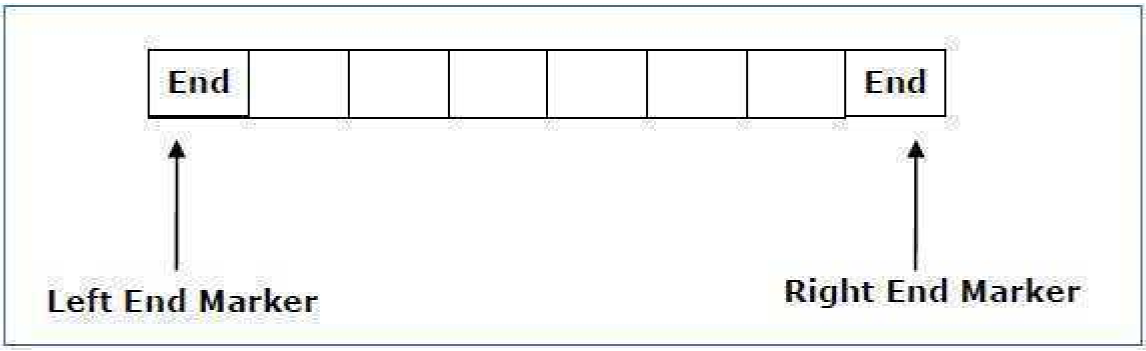
b) Define Linear Bounded Automata (5)

ans:

A linear bounded automaton can be defined as an 8-tuple (Q, X, ∑, q_0, ML, MR, δ, F) where −

- **Q** is a finite set of states
- **X** is the tape alphabet
- ∑ is the input alphabet
- **q_0** is the initial state

- **M_L** is the left end marker
- **M_R** is the right end marker where $M_R \neq M_L$
- **δ** is a transition function which maps each pair (state, tape symbol) to (state, tape symbol, Constant `c') where c can be 0 or +1 or -1
- **F** is the set of final states



A deterministic linear bounded automaton is always **context-sensitive** and the linear bounded automaton with empty language is **undecidable.**.

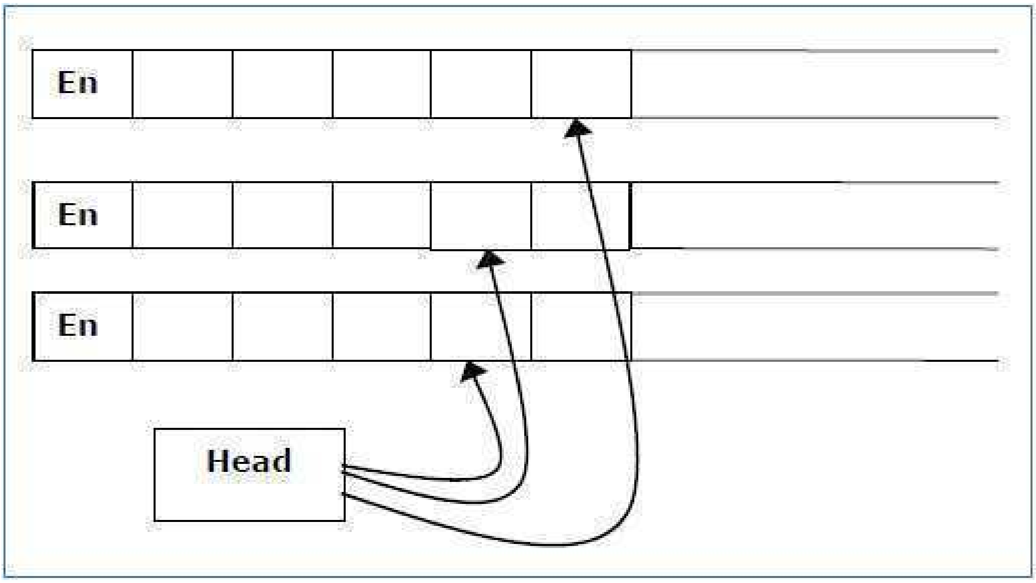
16 Explain about different variants of Turing machine (10)

Ans:

1. Multitape Turing machine

- A **multi-tape Turing machine** is like an ordinary Turing machine with several tapes. Each tape has its own head for reading and writing. Initially the input appears on tape 1, and the others start out blank
- This model intuitively seems much more powerful than the single-tape model, but any multi-tape machine, no matter how many tapes, can be simulated by a single-tape machine using only quadratically more computation time.[2] Thus, multi-tape machines cannot calculate any more functions than single-tape machines,[3] and none of the robust complexity classes (such as polynomial time) are affected by a change between single-tape and multi-tape machines.

Multi-tape Turing Machines have multiple tapes where each tape is accessed with a separate head. Each head can move independently of the other heads. Initially the input is on tape 1 and others are blank. At first, the first tape is occupied by the input and the other tapes are kept blank. Next, the machine reads consecutive symbols under its heads and the TM prints a symbol on each tape and moves its heads.



A Multi-tape Turing machine can be formally described as a 6-tuple (Q, X, B, δ, q_0, F) where −

- **Q** is a finite set of states
- **X** is the tape alphabet
- **B** is the blank symbol
- **δ** is a relation on states and symbols where

  δ: $Q \times X^k \rightarrow Q \times (X \times \{Left\_shift, Right\_shift, No\_shift\})^k$

  where there is **k** number of tapes
- **q_0** is the initial state
- **F** is the set of final states

Note − Every Multi-tape Turing machine has an equivalent single-tape Turing machine.

- 2.Multi track turing machine:

Multi-track Turing machines, a specific type of Multi-tape Turing machine, contain multiple tracks but just one tape head reads and writes on all tracks. Here, a single tape head reads n symbols from **n** tracks at one step. It accepts recursively enumerable languages like a normal single-track single-tape Turing Machine accepts.

A Multi-track Turing machine can be formally described as a 6-tuple (Q, X, ∑, δ, q_0, F) where −

- **Q** is a finite set of states
- **X** is the tape alphabet

- ∑ is the input alphabet
- **δ** is a relation on states and symbols where

  $\delta(Q_i, [a_1, a_2, a_3, ...]) = (Q_j, [b_1, b_2, b_3, ...], \text{Left\_shift or Right\_shift})$
- **q_0** is the initial state
- **F** is the set of final states

Note − For every single-track Turing Machine **S**, there is an equivalent multi-track Turing Machine **M** such that **L(S) = L(M)**

3.Non-Deterministic Turing Machine:

In a Non-Deterministic Turing Machine, for every state and symbol, there are a group of actions the TM can have. So, here the transitions are not deterministic. The computation of a non-deterministic Turing Machine is a tree of configurations that can be reached from the start configuration.

An input is accepted if there is at least one node of the tree which is an accept configuration, otherwise it is not accepted. If all branches of the computational tree halt on all inputs, the non-deterministic Turing Machine is called a **Decider** and if for some input, all branches are rejected, the input is also rejected.

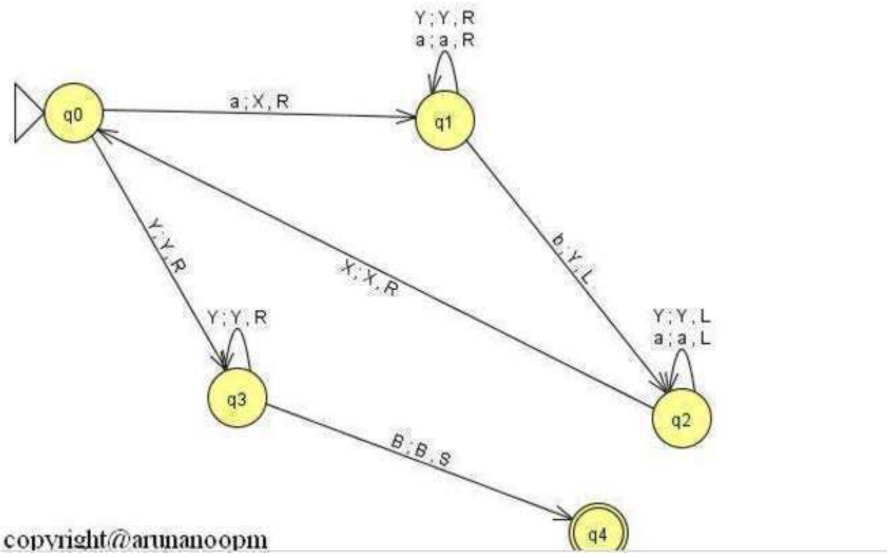
A non-deterministic Turing machine can be formally defined as a 6-tuple (Q, X, ∑, δ, q_0, B, F) where −

- **Q** is a finite set of states
- **X** is the tape alphabet
- ∑ is the input alphabet
- **δ** is a transition function;

  $\delta : Q \times X \rightarrow P(Q \times X \times \{\text{Left\_shift}, \text{Right\_shift}\})$.
- **q_0** is the initial state
- **B** is the blank symbol
- **F** is the set of final states

17. a) Design a Turing machine to accept the language L= $\{a^n b^n / n>=1\}$ (5)

Soln:



Concept :

aaabbb
Case1: XaaYbb
Case2:XXaYYb
Case3:XXXYYY

Case 1:
Read first 'a' and replace it by 'X'
Then read first 'b' and replace it as 'Y'

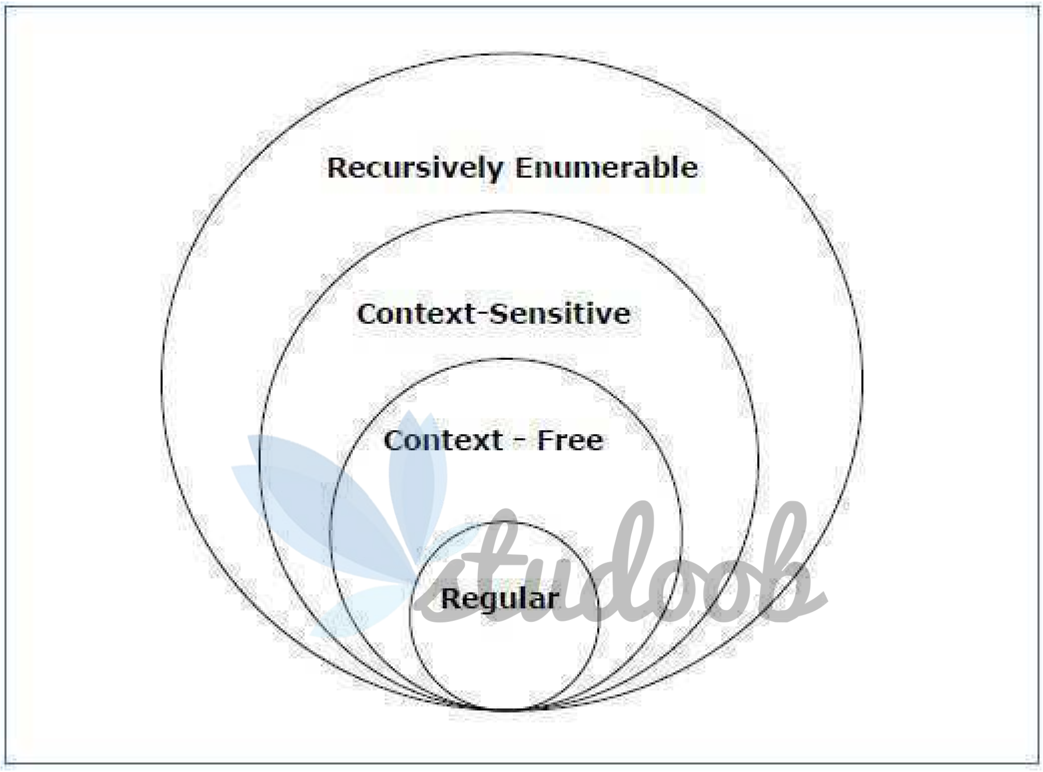
b) List the Chomsky classification of languages and grammars. (5)

According to Noam Chomosky, there are four types of grammars − Type 0, Type 1, Type 2, and Type 3. The following table shows how they differ from each other –

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing Machine

Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton

Take a look at the following illustration. It shows the scope of each type of grammar −



Type - 3 Grammar

Type-3 grammars generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.

The productions must be in the form **X → a or X → aY**

where **X, Y ∈ N** (Non terminal)

and **a ∈ T** (Terminal)

The rule **S → ε** is allowed if **S** does not appear on the right side of any rule.

Example

```
X → ε
X → a | aY
Y → b
```

Type - 2 Grammar

Type-2 grammars generate context-free languages.

The productions must be in the form **A → γ**

where **A ∈ N** (Non terminal)

and **γ ∈ (T ∪ N)*** (String of terminals and non-terminals).

These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

Example

```
S → X a
X → a
X → aX
X → abc
X → ε
```

Type - 1 Grammar

Type-1 grammars generate context-sensitive languages. The productions must be in the form

α A β → α γ β

where **A ∈ N** (Non-terminal)

and **α, β, γ ∈ (T ∪ N)*** (Strings of terminals and non-terminals)

The strings **α** and **β** may be empty, but **γ** must be non-empty.

The rule **S → ε** is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.

Example

```
AB → AbBc
A → bcA
B → b
```

Type - 0 Grammar

Type-0 grammars generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.

They generate the languages that are recognized by a Turing machine.

The productions can be in the form of **α → β** where **α** is a string of terminals and nonterminals with at least one non-terminal and **α** cannot be null. **β** is a string of terminals and non-terminals.

Example

```
S → ACaB
Bc → acB
CB → DB
aD → Db
```

18. a) Explain about Recursive and Recursively enumerable languages (5)

Ans:

Recursive Enumerable (RE) or Type -0 Language
RE languages or type-0 languages are generated by type-0 grammars. An RE language can be accepted or recognized by Turing machine which means it will enter into final state for the strings of language and may or may not enter into rejecting state for the strings which are not part of the language. It means TM can loop forever for the strings which are not a part of the language. RE languages are also called as Turing recognizable languages.

Recursive Language (REC)
A recursive language (subset of RE) can be decided by Turing machine which means it will enter into final state for the strings of language and rejecting state for the strings which are not part of the language. e.g.; L= {$a^nb^nc^n$|n>=1} is recursive because we can construct a turing machine which will move to final state if the string is of the form anbncn else move to non-final state. So the TM will always halt in this case. REC languages are also called as Turing decidable languages. The relationship between RE and REC languages can be shown in Figure 1.

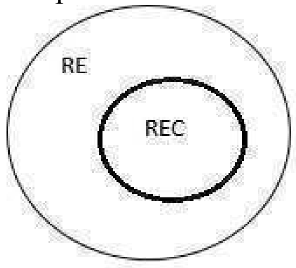


Figure 1

b) What are the properties of Recursively enumerable languages (5)

ans:

Closure Properties of Recursive Languages

- **Union**: If L1 and If L2 are two recursive languages, their union L1∪L2 will also be recursive because if TM halts for L1 and halts for L2, it will also halt for L1∪L2.
- **Concatenation:** If L1 and If L2 are two recursive languages, their concatenation L1.L2 will also be recursive. For Example:

- L1= $\{a^nb^nc^n|n>=0\}$
- L2= $\{d^me^mf^m|m>=0\}$
- L3= L1.L2
- = $\{a^nb^nc^nd^me^mf^m|m>=0$ and $n>=0\}$ is also recursive.

L1 says n no. of a' s followed by n no. of b' s followed by n no. of c' s. L2 says m no. of d' s followed by m no. of e' s followed by m no. of f' s. Their concatenation first matches no. of a' s, b' s and c' s and then matches no. of d' s, e' s and f' s. So it can be decided by TM.

- **Kleene Closure:** If L1is recursive, its kleene closure L1* will also be recursive. For Example:

L1= $\{a^nb^nc^n|n>=0\}$
L1*= $\{ a^nb^nc^n||n>=0\}^*$ is also recursive.

- **Intersection and complement**: If L1 and If L2 are two recursive languages, their intersection L1 ∩ L2 will also be recursive. For Example:

- L1= $\{a^nb^nc^ndm|n>=0$ and $m>=0\}$
- L2= $\{a^nb^nc^nd^n|n>=0$ and $m>=0\}$
- L3=L1 ∩ L2
- = $\{ a^nb^nc^nd^n |n>=0\}$ will be recursive.

L1 says n no. of a' s followed by n no. of b' s followed by n no. of c' s and then any no. of d' s. L2 says any no. of a' s followed by n no. of b' s followed by n no. of c' s followed by n no. of d' s. Their intersection says n no. of a' s followed by n no. of b' s followed by n no. of c' s followed by n no. of d' s. So it can be decided by turing machine, hence recursive. Similarly, complementof recursive language L1 which is ∑*-L1, will also be recursive.

19. a) Explain about Universal Turing Machine (5)

Ans:

- A Turing machine is said to be universal Turing machine if it can accept:
  - The input data, and
  - An algorithm (description) for computing.
- This is precisely what a general purpose digital computer does. A digital computer accepts a program written in high level language. Thus, a general purpose Turing

machine will be called a universal Turing machine if it is powerful enough to simulate the behavior of any digital computer, including any Turing machine itself.

- More precisely, a universal Turing machine can simulate the behavior of an arbitrary Turing machine over any set of input symbols. Thus, it is possible to create a single machine that can be used to compute any computable sequence.

  If this machine is supposed to be supplied with the tape on the beginning of which is written the input string of quintuple separated with some special symbol of some computing machine M, then the universal Turing machine U will compute the same strings as those by M.
- The model of a Universal Turing machine is considered to be a theoretical breakthrough that led to the concept of stored programmer computing device.
- Designing a general purpose Turing machine is a more complex task. Once the transition of Turing machine is defined, the machine is restricted to carrying out one particular type of computation.
- Digital computers, on the other hands, are general purpose machines that cannot be considered equivalent to general purpose digital computers until they are designed to be reprogrammed.
- By modifying our basic model of a Turing machine we can design a universal Turing machine. The modified Turing machine must have a large number of states for stimulating even a simple behavior. We modify our basic model by:
  - Increase the number of read/write heads
  - Increase the number of dimensions of input tape
  - Adding a special purpose memory
- All the above modification in the basic model of a Turing machine will almost speed up the operations of the machine can do.
- A number of ways can be used to explain to show that Turing machines are useful models of real computers. Anything that can be computed by a real computer can also be computed by a Turing machine. A Turing machine, for example can simulate any type of functions used in programming language. Recursion and parameter passing are some typical examples. A Turing machine can also be used to simplify the statements of an algorithm.
- A Turing machine is not very capable of handling it in a given finite amount of time. Also, Turing machines are not designed to receive unbounded input as many real programmers like word processors, operating system, and other system software.

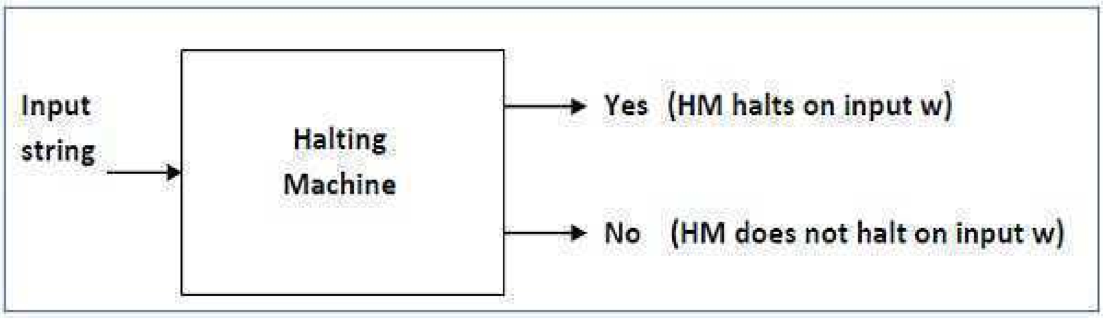
b) Explain halting problem prove that it is un decidable (5)

ans:

Input − A Turing machine and an input string **w**.

Problem − Does the Turing machine finish computing of the string **w** in a finite number of steps? The answer must be either yes or no.

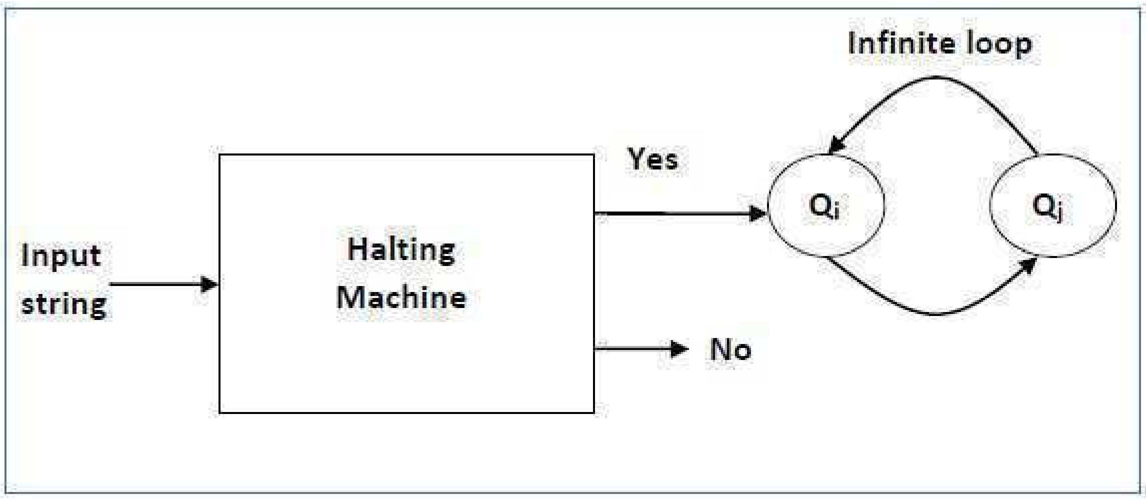
Proof − At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a **Halting machine** that produces a `yes' or `no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as `yes', otherwise as `no'. The following is the block diagram of a Halting machine −



Now we will design an **inverted halting machine (HM)'** as −

- If **H** returns YES, then loop forever.
- If **H** returns NO, then halt.

The following is the block diagram of an 'Inverted halting machine' −



Further, a machine **(HM)₂** which input itself is constructed as follows −If (HM)₂ halts on input, loop forever.Else, halt.

Here, we have got a contradiction. Hence, the halting problem is **undecidable**.

20. Design a Turing Machine which accepts L = { ww^R / w in (a+b)* } (10)

Soln:

String recognition process of the Turing machine is as follows:

1. Initially, TM is in state q_0 and head points to first symbol in the string.

2. The symbol may be a or b. Aftyer reading this symbol, this symbol is replaced with a # and keeps moving towards right till the whole of the string is crossed and the first # after the string is reached.

3. From this blank, the head turns one cell left and reaches the last symbol of the input string. If this symbol matches with the first symbol of the input string, it converts it into a #.

4. Steps 1-3 match the first and last symbols of the input string. If this matching is successful, the head traverses back and crosses the string to reach the first #. From this #, it turns towards right and matches the leftmost and rightmost symbols of the remaining string. If the matching is successful, then the symbols are converted to #s.

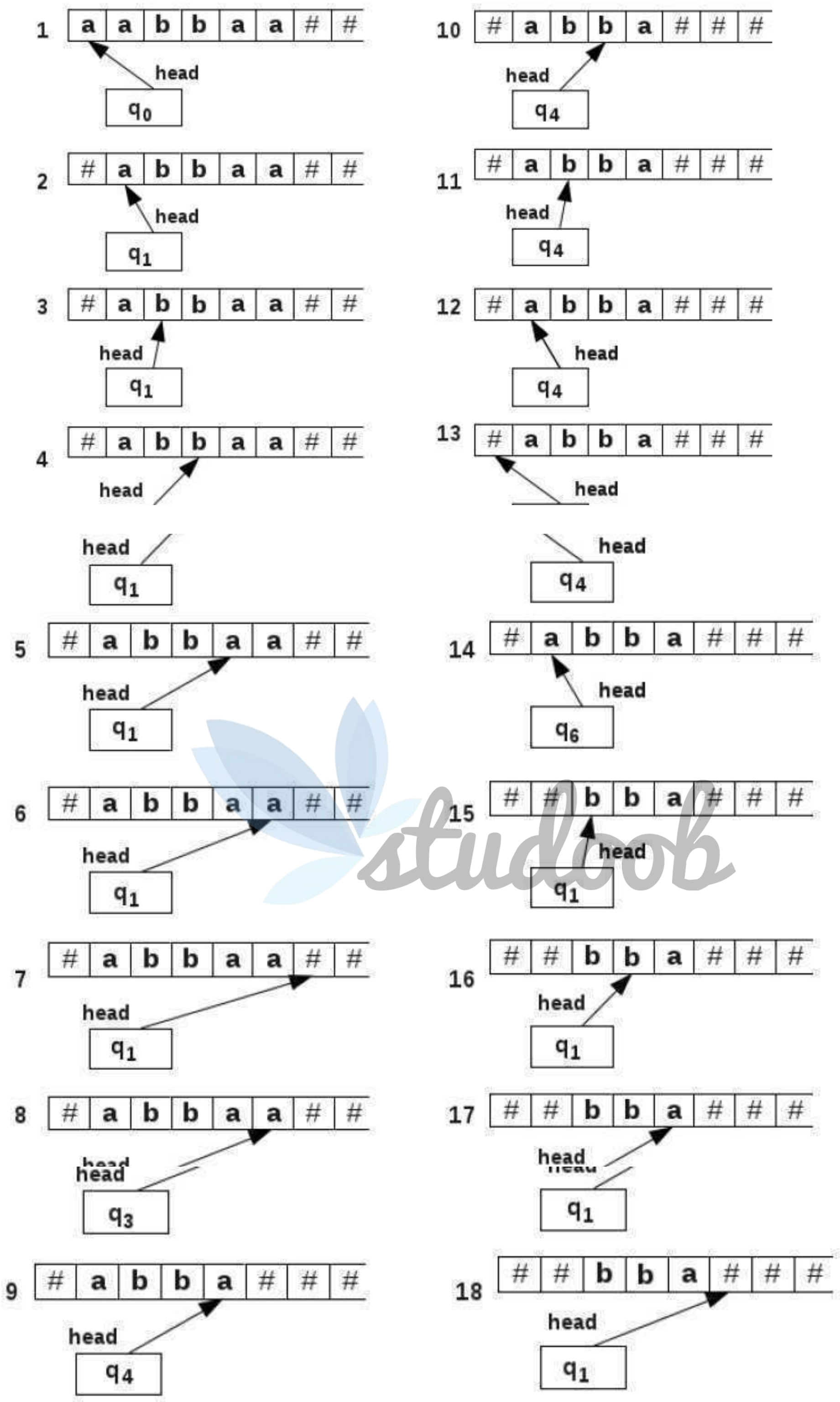
5. The process repeats till the symbols are matched.

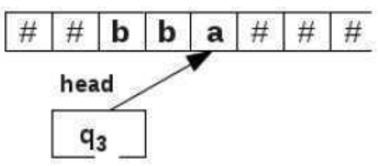
6. To keep track of the symbol to be matched, if symbol 'a' is received in state q_0, then head enters state q_1, and if symbol 'b' is received, then head enters state q_2.

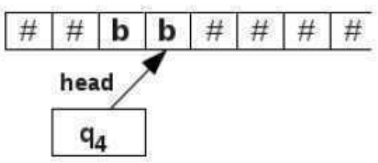
Transition table corresponding to the Turing machien is given below:

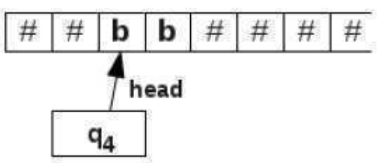
	Input	Symbol	
Current State	a	b	#
$\longrightarrow q_0$	$(q_1, \#, R)$	$(q_2, \#, R)$	—
q_1	(q_1, a, R)	(q_1, b, R)	$(q_3, \#, L)$
q_2	(q_2, a, R)	(q_2, b, R)	$(q_5, \#, L)$
q_3	$(q_4, \#, L)$	—	—
q_4	(q_4, a, L)	(q_4, b, L)	$(q_6, \#, R)$
q_5	—	$(q_4, \#, L)$	—
q_6	$(q_1, \#, R)$	$(q_2, \#, R)$	$(q_f, \#, R)$
$*q_f$	—	—	—

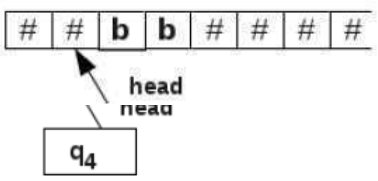
Let the input string is abaaba.

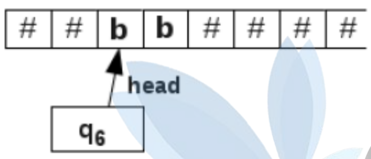

1 a a b b a a # # head q0
2 # a b b a a # # head q1
3 # a b b a a # # head q1
4 # a b b a a # # head head q1
5 # a b b a a # # head q1
6 # a b b a a # # head q1
7 # a b b a a # # head q1
8 # a b b a a # # head q3
9 # a b b a # # # head q4
10 # a b b a # # # head q4
11 # a b b a # # # head q4
12 # a b b a # # # head q4
13 # a b b a # # # head head q4
14 # a b b a # # # head q6
15 # # b b a # # # head q1
16 # # b b a # # # head q1
17 # # b b a # # # head q1
18 # # b b a # # # head q1

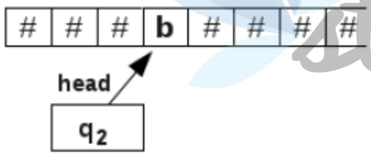
19


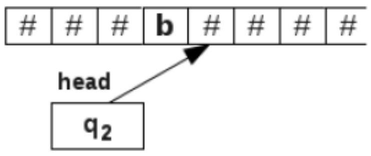
20


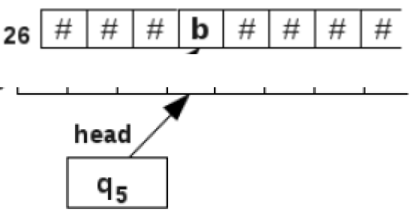
21


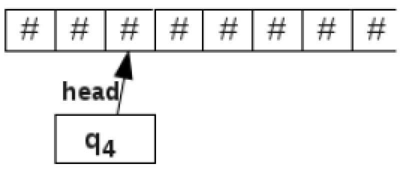
22


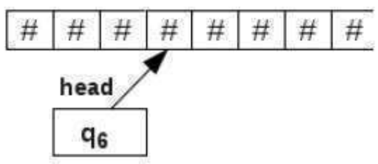
23


24


25


26


27


28


29