UNIT-V TURING MACHINE

→ The machine is able to move a read or write head, left and right over the tape as it performs its competition.

→ It can read and write symbols on the tape as it places.

→ This consideration lead turing to the formal definition.

- DEFINITION:-

A turing machine is a seven tuple

$M = \{Q, \Sigma, \tau, \delta, q_0, B, F\}$

Q - finite set of states

Σ - input alphabet

τ - tape alphabet which always include blanks

δ - $\delta : Q \times \tau \rightarrow Q \times \tau \times \{\overset{\text{Left}}{L}, \overset{\text{Right}}{R}\}$

state using tape symbol it goes to same or different state moves left or right

q_0 - initial state

B - special symbol (indication of a blank cell

F - set of final states.

→ The machine simply moves right along the tape until it hits a blank and then it halts.

→ At each step it just writes back the

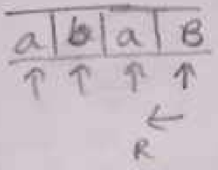
current symbol, remains in q_0, and moves right by one cell.

the transitions can be defined as

→ $\delta(q_0, 0) = (q_0, 0, R)$

$\delta(q_0, 1) = (q_0, 1, R)$

Once the machine hits a blank it moves 1 cell to the left and stops.

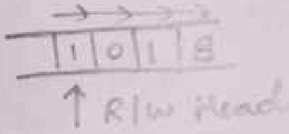


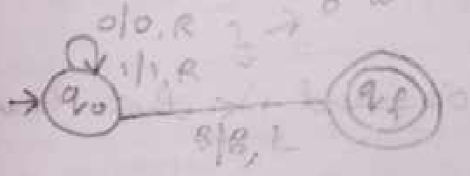
Once blank hit, it turns left and stops.

$\delta(q_0, B) = (q_f, B, L)$

Q.1. Design a turing machine to accept strings belonging to the language $(0+1)^*$

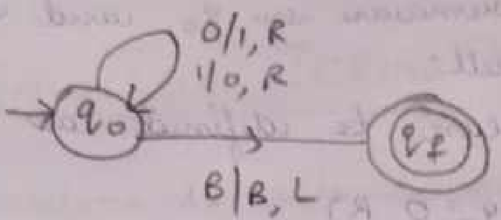
$\{\epsilon, 0, 1, 00, 11, 01, 000, \ldots\}$





B|B - after looking at blank, we did not modify, but moved to left

Q.2. Design a turing machine to give the one's complement of a binary number.

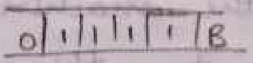


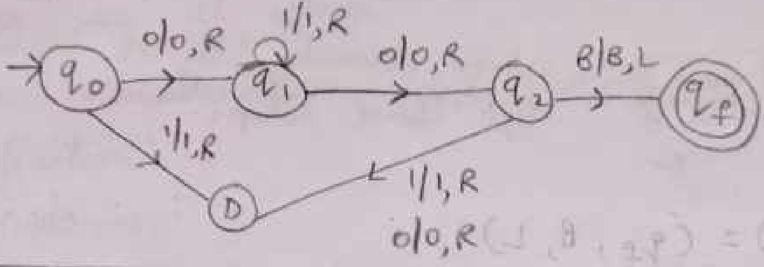
0|1, R

At 0, we are modifying it to 1, moving write.

Q-3 01* 0

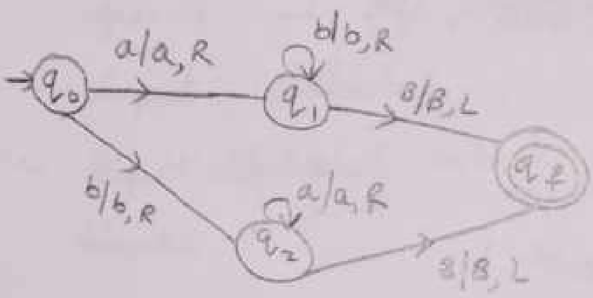
{00, 010, 0110, 01110, ... }





Q-4. ab* or ba*

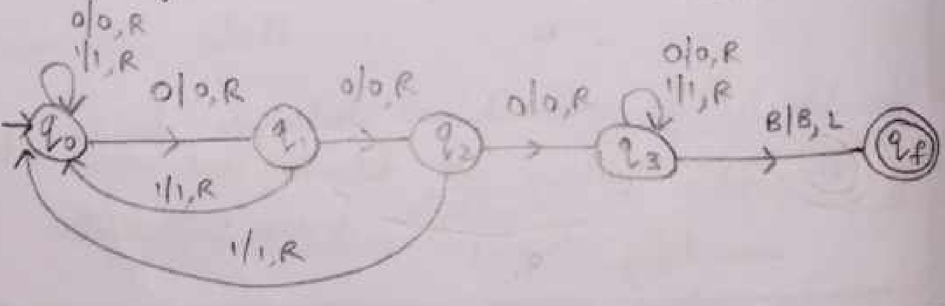
{ a, ab, abb, ..., b, ba, ba, ... }



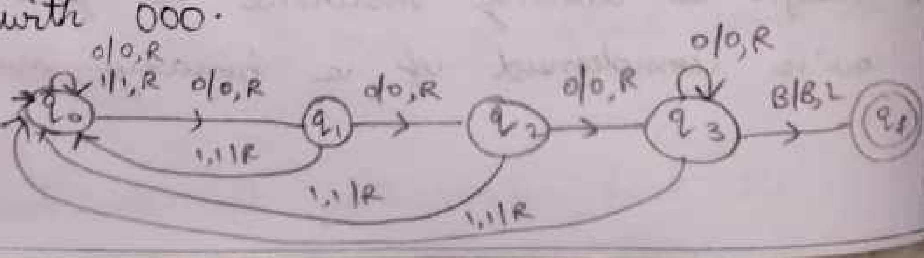
Q-5. Strings formed with 0's and 1's that have a substring 000.

01100011

10101 0001



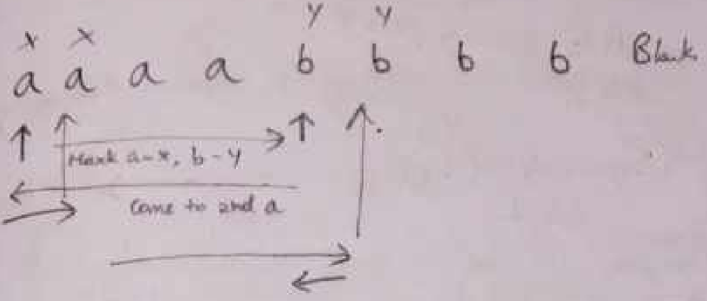
Q-6. Ends with 000.



27/9/24

$a^n b^n$

{aaaabbbb}

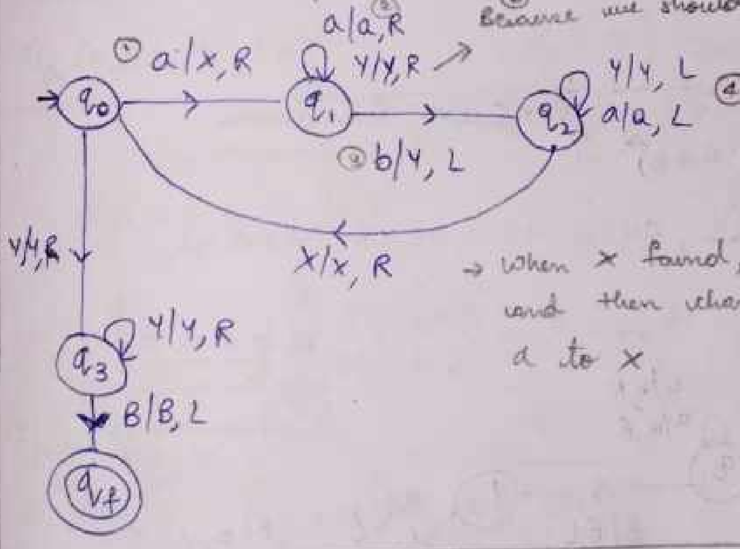


① a is read, make a as x and move to right.

② Any no. of a's skip, move right until b.

③ when b found make it Y, and turn left.

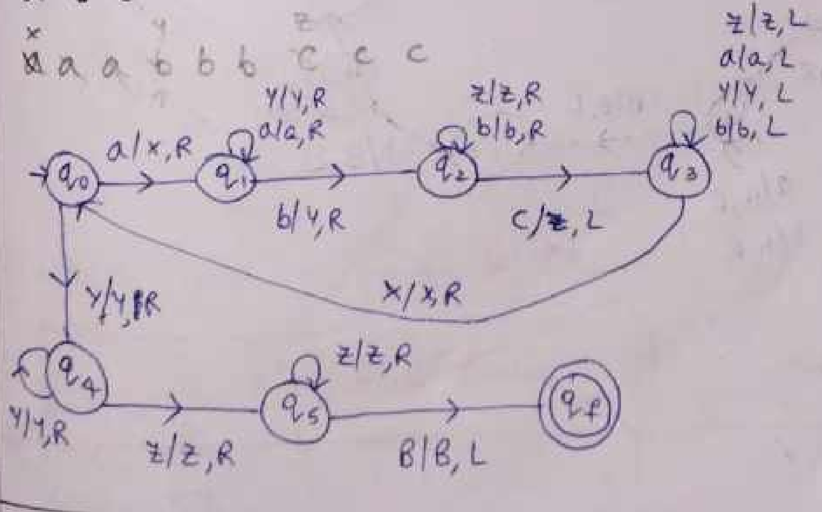
④ while moving left Any no of a's, y's skip, move left until x found, so move right and make 'a' as x



⑥ Because we should cross Y also along with a

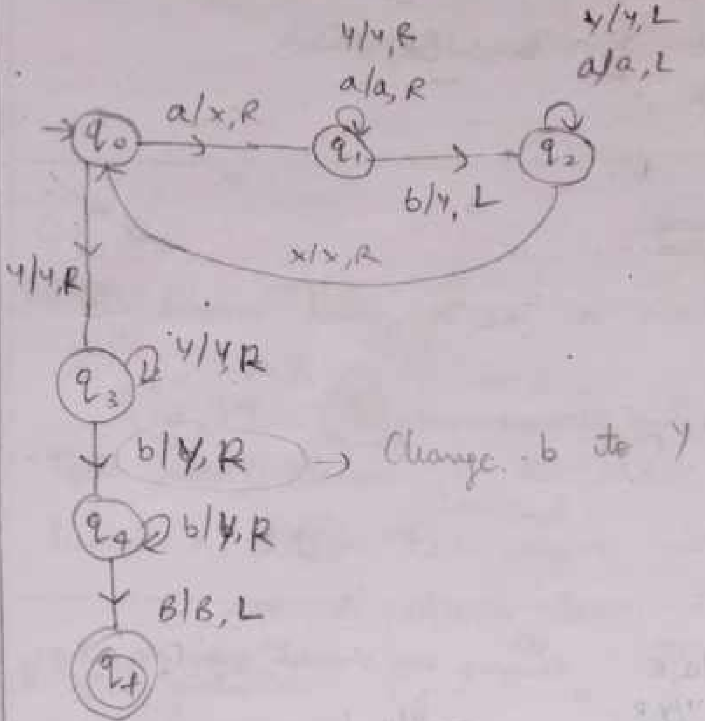
→ when x found, turn right and then change the next a to x

Q.8 $a^n b^n c^n$ a-x, b-y, c-z



Q.9 $a^n b^m$, $n<m$, $n\geq 1$, $m\geq 2$

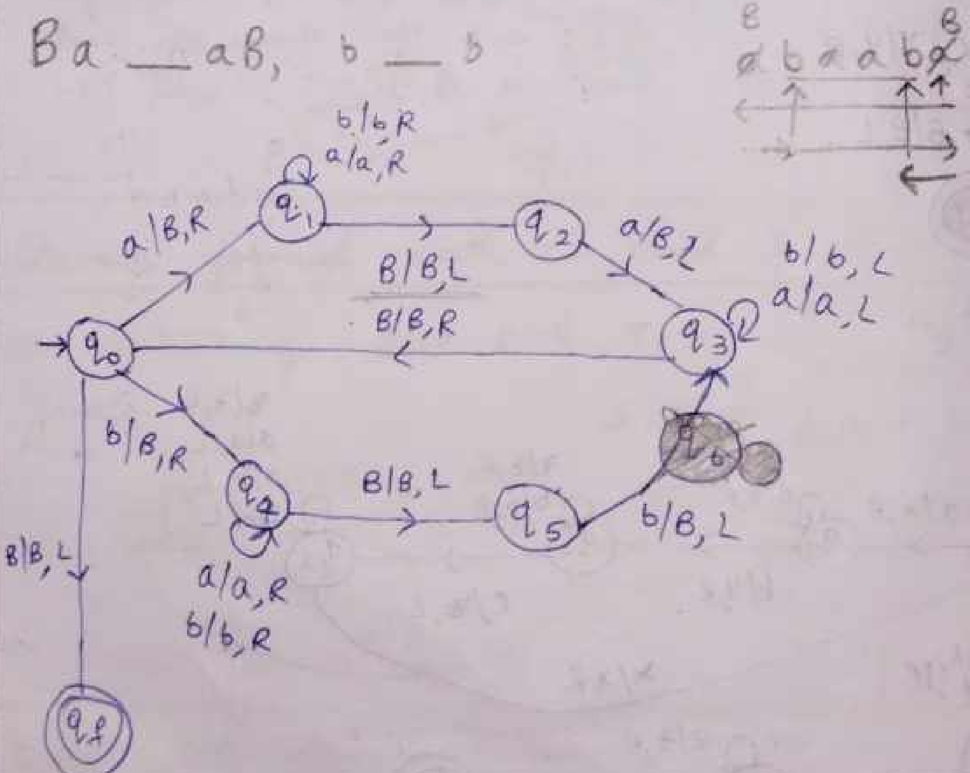
$\overset{x}{a}a\overset{y}{b}bb$



★ Q.10 $w|w^R$ $w\in(a+b)^*$ (OR) Even Palindrome (String_Reverse) String)

aabb|bbaa, aaaa, bbbb, baab, abaaba

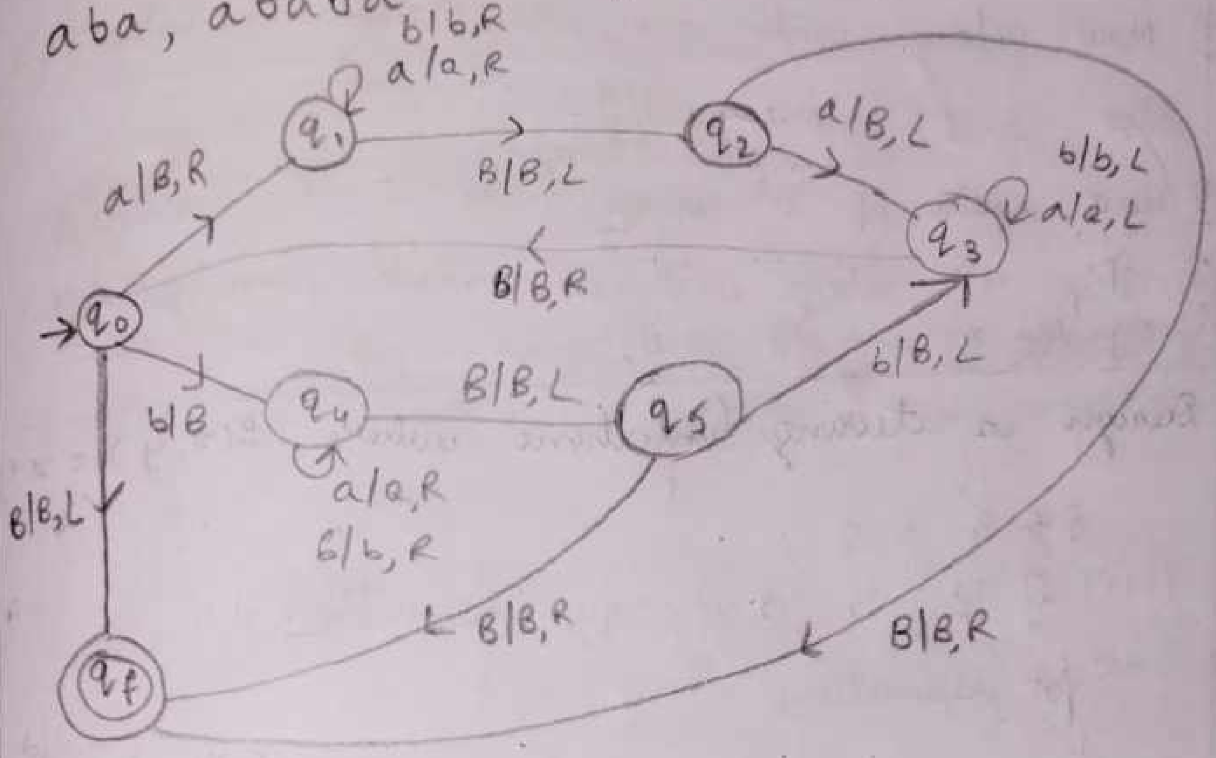
Ba __ aB, b __ b



Q.11 Palindrome

abb(a)bba, a, b, ε
↑

aba, ababa



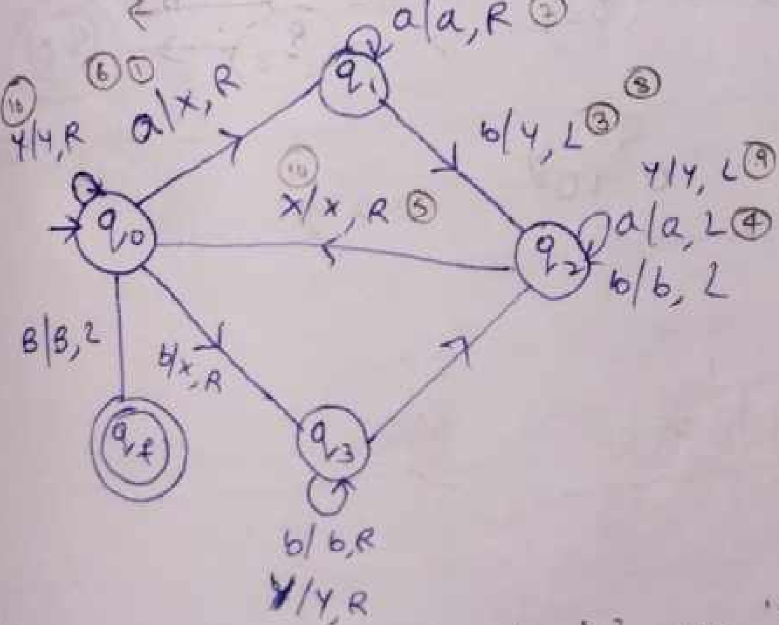
Q.12 $n_a(a) = n_b(b)$

$\overset{x}{a}bab$, $\overset{x}{a}a\overset{y}{b}abb$

bbaa, abba, abab, baba, aabb

x xy y yy
aaabbbaabb (B)
↑ ↑ ↑

~~a~~aabbb {$a^n b^n$
x → y



Starts with a

Starts with b

to 'a' as 'x' and then move

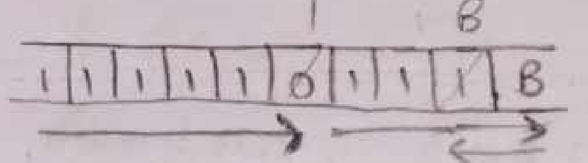
Q.13. Design a turing machine where $f(x, y) = x+y$

5 + 3 = 8

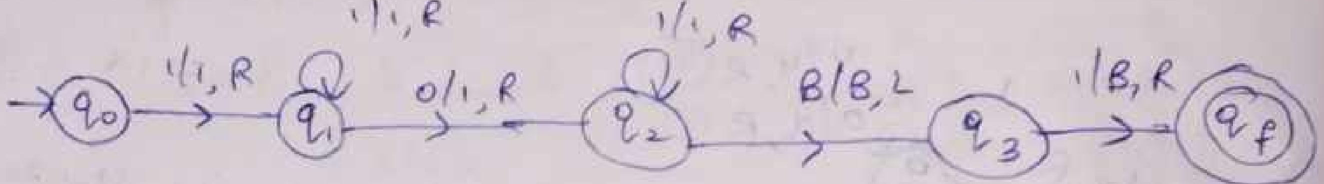
11111 0 111 → unary operation

↑

for seperation



If 0 encountered, make it 1 and move right, whenever B occurs, turn left and make it B, and turn right.

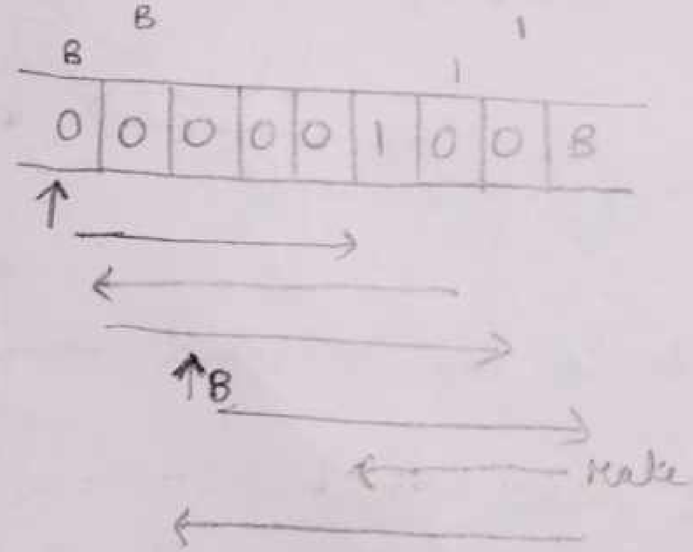


Q-14 $f(x, y) = x - y$ $0^x 1 0^y$ $x > y$

00000100

5 - 2 = 1

①



Make all 1's blank

0

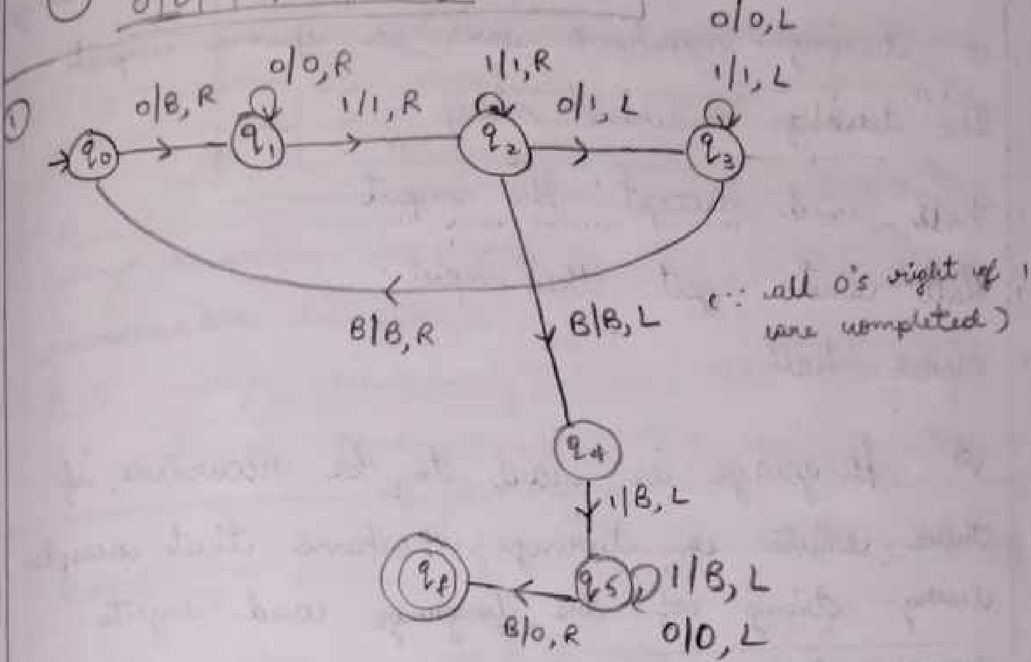
(∵ This blank has no 0 after 1, make it 0 again)

(A)

② B B B   0 B B

0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | B

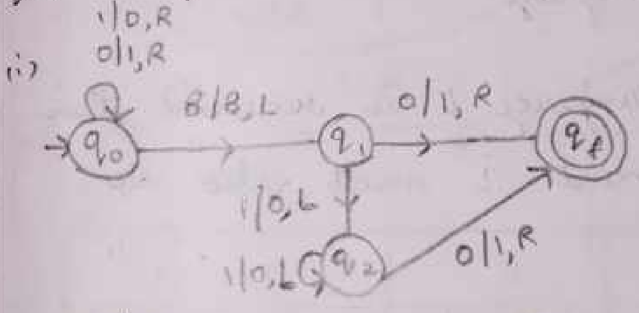
①



(∵ all 0's right of 1 are completed)

Q.15 2's complement

(i)



(i) ① 1011011

0100100

1

0100101

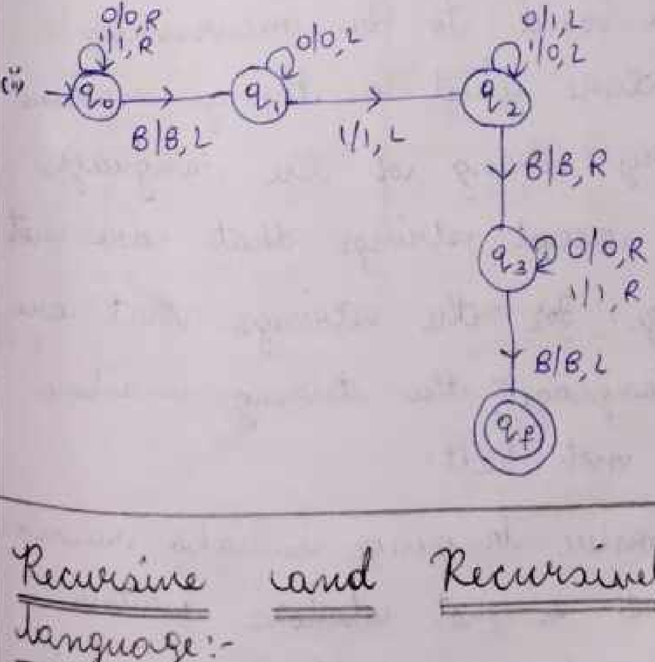
② 100100

011011

1

011100

(ii)



(ii) 100100

011100

1 encountered. All digits before that are complemented

* Recursive and Recursively Enumerable language:-

- There are three possible outcomes of executing a turing machine over a given input. The turing machine may:-

(1) Halt and accept the input.

(2) Halt and reject the input.

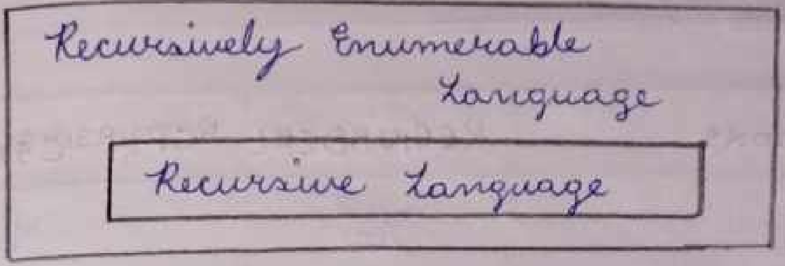
(3) Never halt.

- A language is said to be recursive, if there exists a turing machine that accepts every string of the language and rejects every string (over the same alphabet) i.e. not in the language.

NOTE:- If the language L is recursive, then its complement $\bar{L}$ must also be recursive.

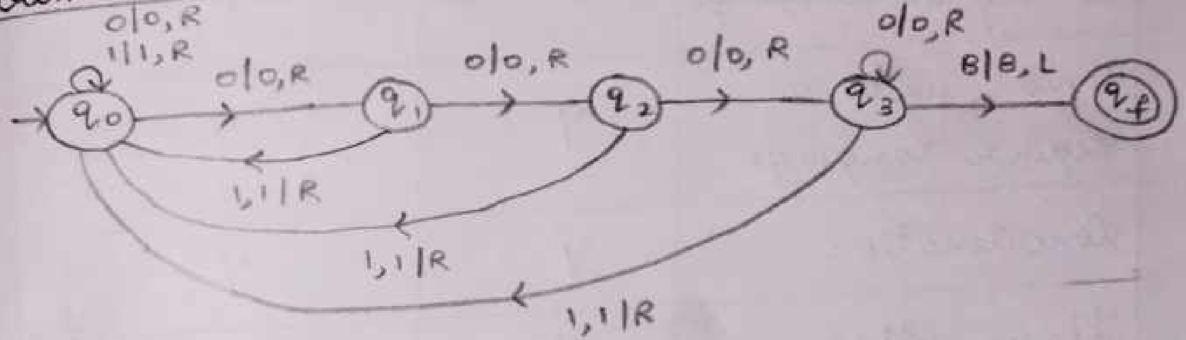
- A language is said to be recursively enumerable, if there exist a turing machine that accepts every string of the language and does not accept strings that are not in the language. For the strings that are not in the language, the turing machine may or may not halt.

NOTE:- Every recursive language is also recursively enumerable. It is not obvious whether every recursively enumerable language is also recursive.



Every recursive language is a recursively enumerable language.
But every recursively enumerable language is not a recursive language.

* Transition table for string ends with 000.



	0	1	B
→ q0	(q1, 0, R)	(q0, 1, R)	–
q1	(q2, 0, R)	(q0, 1, R)	–
q2	(q3, 0, R)	(q0, 1, R)	–
q3	(q3, 0, R)	(q0, 1, R)	(qf, B, L)
* qf	–	–	–

7/10/24

* Closure Properties :-

OPERATIONS	RECURSIVE	RECURSIVE ENUMERABLE
Union	Yes	Yes
Intersection	Y	Y
Set Difference	Y	No
Complement	Y	No
Intersection with a Regular Language	Y	Y
Union with a Regular Language	Y	Y
Concatenation	Y	Y
Kleene Star	Y	Y
Kleene Plus	Y	Y
Reversal	Y	Y
Epsilon - free Homomorphism	Y	Y
Homomorphism	N	Y
Inverse Homomorphism	Y	Y
Epsilon - free Substitution	Y	Y
Substitution	N	Y
Subset	N	N

OPERATIONS	RECURSIVE	RECURSIVE ENUMERABLE
Left Difference with a Regular Language (L-Regular)	Y	Y
Right Difference with a Regular Language (Regular-R)	Y	N
Left Quotient with a Regular Language	Y	Y
Right Quotient with a Regular Language	Y	Y

* Computable Functions:-

In Theory of Computation (ToC), a computable function is a mathematical function that can be computed using a Turing machine or any other equivalent model of computation.

Definition:

A function $f: \Sigma^* \rightarrow \Sigma^*$ (where Σ is a finite alphabet) is computable if there exists a Turing machine M such that:

(1) For every input $x \in \Sigma^*$, M halts (i.e., stops computing) on x.

(2) If M halts on x, the output of M on x is $f(x)$.

- **Properties of Computable Functions :-**

1. **Partiality** : Computable functions can be partial, meaning they may not be defined for all inputs.

2. **Determinism** : Computable functions are deterministic, meaning their output depends only on the input.

3. **Effectiveness** : Computable functions can be computed effectively, meaning there exists an algorithm (Turing machine) to compute them.

Examples of Computable Functions :

1. Arithmetic operations (+, -, x, /)
2. Logical operations (AND, OR, NOT)
3. String manipulation (concatenation, substring extraction)
4. Sorting algorithms (eg. linear search, binary search)

Non Computable Functions:

1. The Halting Problem (determining whether a Turing machine halts on a given input)
2. The Decision Problem for first-order logic (determining whether a given sentence is true in all models)
3. The Word Problem for finitely presented groups (determining whether 2 words represent the same group element).

*Types of Turing Machines:-

Non-deterministic Turing Machines

A non-deterministic turing Machines is a machine for which like non-deterministic finite automata, at any current state and for the tape symbol it is reading, there may be different possible actions to be performed. Here an action means a combination of writing a symbol on the tape, moving the tape head and going to a next state. One action could be just changing the state without modifying the cell content. One action could be not changing the state and changing the cell content. One action could be changing both the state and cell content. In all actions, it may move left or right.

For example, $L = \{WW / W \in \{a,b\}^*\}$..

It must findout the midpoint by, pairing off symbols from the two ends of x....

* Any language accepted by a non-deterministic Turing Machine is also accepted by deterministic Turing Machine

*Transition function (δ):- $Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{L,R\}}$

Turing Machines with two-dimensional tapes

Turing machines with two-dimensional tape is a kind of TM that has one finite control, one read-write head and one two-dimensional tape. The cells in the tape is two-dimensional i.e., the tape has the top end and the left end, but extends indefinitely to the right and down. It is divided into rows of small squares. For any TM of this type, there is an equivalent TM with a one-dimensional tape that is equally powerful.

* To simulate a two-dimensional tape with a one-dimensional tape, first we map the squares of two-dimensional tape to those of a one-dimensional tape diagonally as shown in the following tables.

Two-dimensional tape:-

v	v	v	v	v	v	v	...	...
h	1	2	6	7	15	16	...	...
h	3	5	8	14	17	26	...	...
h	4	9	13	18	25	...	...	...
h	10	12	19	24	...	...	...	...
h	11	20	23	...	...	...	...	...
h	21	22	<.7	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Here, the numbers indicate the correspondence between the squares of two tapes: square numbered i in two-dimensional tape is mapped to square numbered i in one-dimensional tape. Symbols h and v are not in tape alphabet and they are used to mark the left and the top end ~~end~~ of the tape respectively.

Equivalent one-dimensional tape:-

v	1	v	2	3	h	4	5	6	v	7	8	9	10	h	11	...	...

Head of two-dimensional tape moves one square up, down, left or right.

* Some TM's with a one-dimensional tape can simulate every move of TM with a two-dimensional tape. Hence they are at least as powerful as TMs with two-dimensional tape.

* Since TMs with a two-dimensional tape obviously can simulate TMs with a one-dimensional tape, it can be said that they are equally powerful

* Transition function (δ):- $Q \times \Gamma \to Q \times \Gamma \times \{L, R, T, B\}$

Turing Machines with Multiple Tapes:-

* This kind of TM has one finite control and more than one tape each with its own read/write head.

* It is denoted by a 7-tuple.

  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F\}$

* It's transition function is a partial function

  $$\delta : Q \times \Gamma^n \longrightarrow (Q \times \{h\}) \times \Gamma^n \times \{L, R\}^n$$

A configuration for this kind of TM must show the current state the machine is in and the state of each tape. It can be proved that any language accepted by a n-tape TM can be accepted by a one-tape TM and that any function computed by n-tape TM can be computed by a one-tape TM. Since the converses are obviously true, one can say that one-tape TMs are as powerful as n-tape TMs.

Turing Machines with Multiple heads:-

* This kind of TMs has one finite control and one tape, but more than one read/write heads. In each state, only one of the heads is allowed to read and write.

* It is denoted by a 7-tuple

  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

* The transition function is a partial function:

  $$\delta : Q \times \{H_1, H_2, \ldots, H_n\} \times \Gamma \to (Q \cup \{h\}) \times \Gamma \times \{L, R\}.$$

  Where $H_1, H_2, \ldots, H_n$ denote the tape heads.

* It can be easily seen that this type of TMs are as powerful as one-tape TMs.

Turing Machines with Infinite tape

* This is a kind of TM that have one finite control and one tape which extends infinitely in both directions.

* It turns out that this type of TMs are also as powerful as one-tape TMs whose tape has left end.

* Two-way infinite tapes:-

A_{-4}	A_{-3}	A_{-2}	A_{-1}	A_0	A_1	A_2	A_3	A_4	...