9.1 Turing machine Model

The Turing machine model is shown in figure 9.1. It is a finite automaton connected to read-write head with the following components:

- Tape
- Read-write head
- Control unit

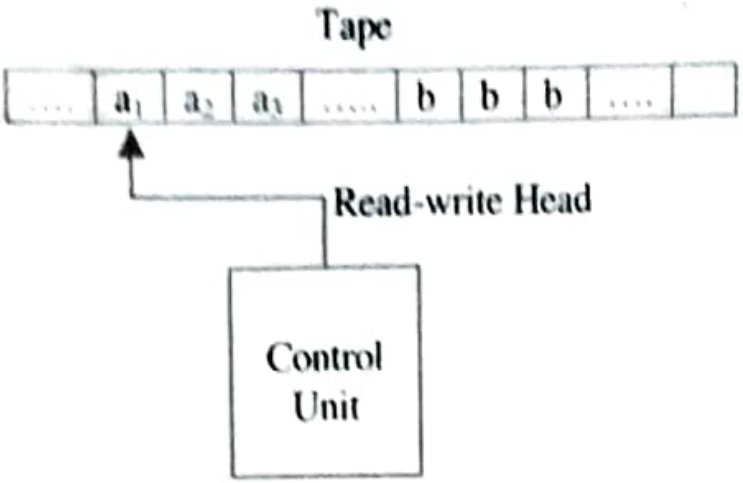


Fig 9.1 Turing machine model

Tape is used to store the information and is divided into cells. Each cell can store the information of only one symbol. The string to be scanned will be stored from the leftmost position on the tape. The string to be scanned should end with blanks. The tape is assumed to be infinite both on left side and right side of the string.

Read-write head The read-write head can read a symbol from where it is pointing to and it can write into the tape to where it points to.

Control Unit The reading from the tape or writing into the tape is determined by the control unit. The different moves performed by the machine depends on the current *scanned symbol* and the *current state*. The control unit consults *action table* i.e., *transition table* and carry out the tasks.

The read-write head can move either towards left or right i.e., movement can be on both the directions. The various actions performed by the machine are:

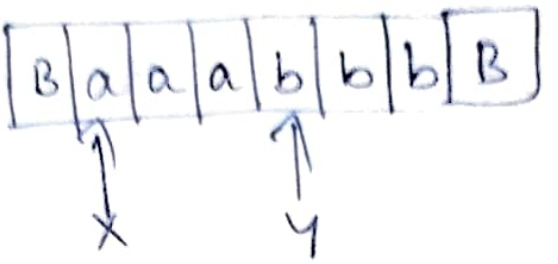
1. Change of state from one state to another state
2. The symbol pointing to by the read-write head can be replaced by another symbol
3. The read-write head may move either towards left or towards right.

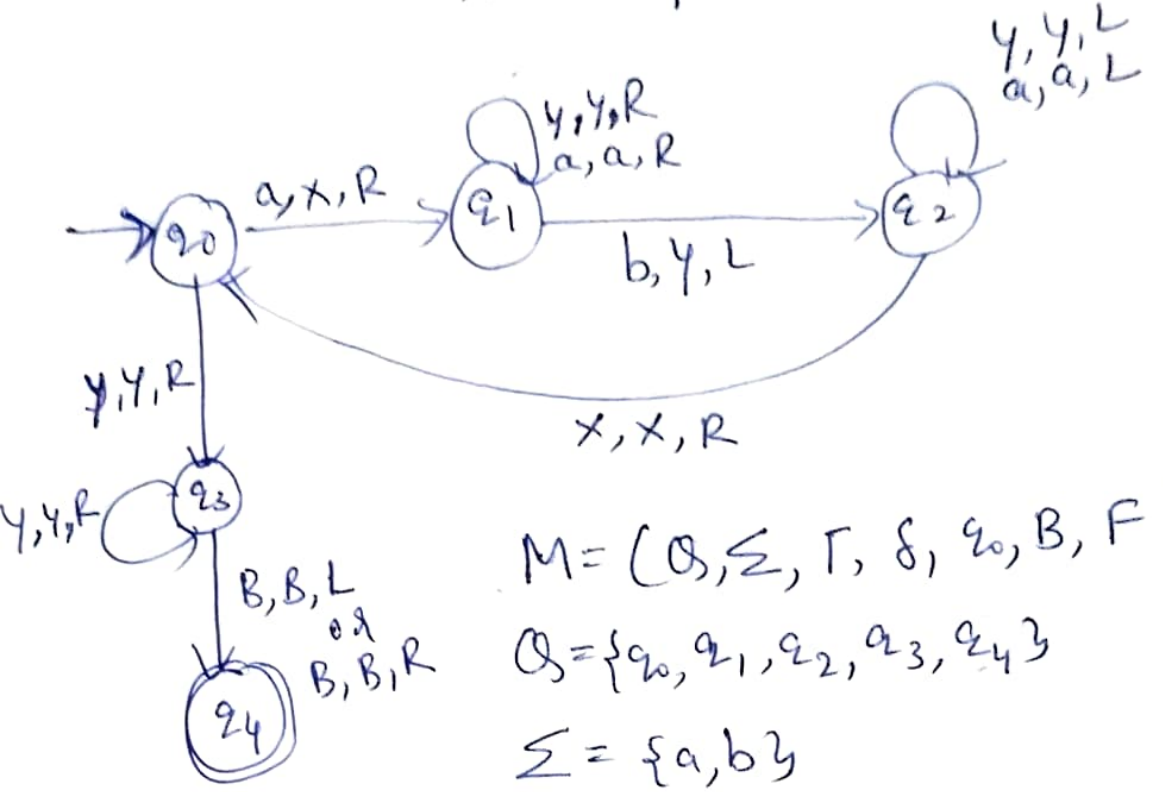
If there is no entry in the table for the current combination of symbol and state, then the machine will halt. The Turing machines can be represented using various notations such as

- Transition tables
- Instantaneous descriptions
- Transition diagram

Turing Machine

1] Design a Turing Machine for $\{a^n b^n \mid n \geq 1\}$





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

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

$\Sigma = \{a, b\}$

$\Gamma = \{a, b, X, Y, B\}$

δ : $\delta(q_0, a) = (q_1, X, R)$

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

$\delta(q_1, Y) = (q_1, Y, R)$

$\delta(q_1, b) = (q_2, Y, L)$

$\delta(q_2, a) = (q_2, a, L)$

$\delta(q_2, Y) = (q_2, Y, L)$

$\delta(q_2, X) = (q_0, X, R)$

$\delta(q_0, Y) = (q_3, Y, R)$

$\delta(q_3, Y) = (q_3, Y, R)$

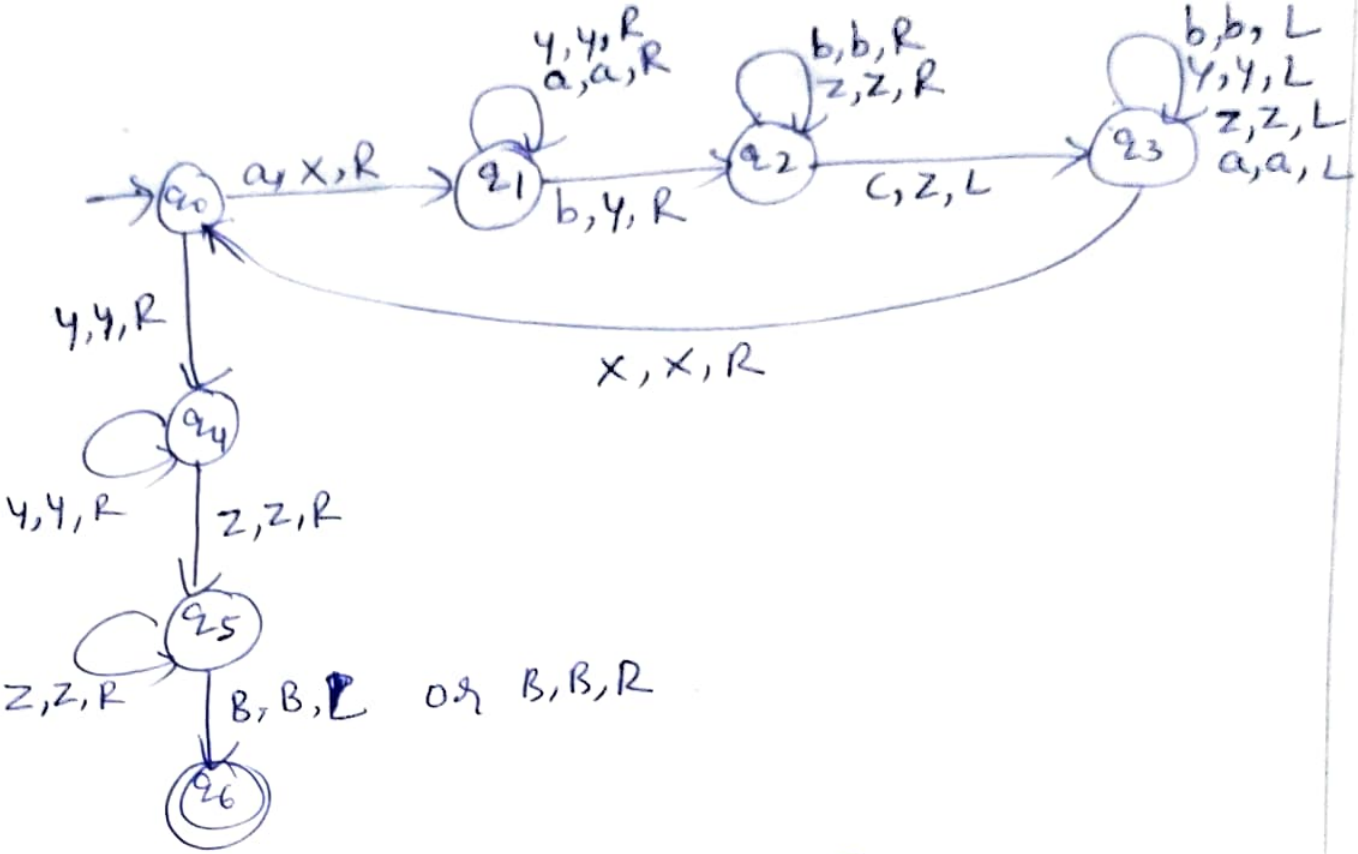
$\delta(q_3, B) = (q_4, B, L)$.

$=$

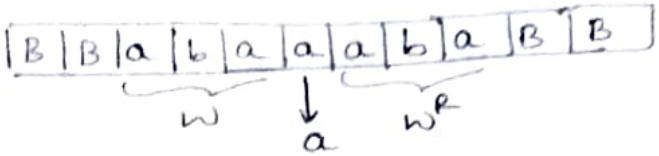
final state .

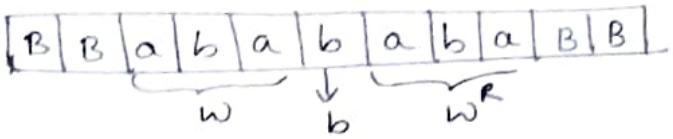
2) $L = \{a^n b^n c^n \mid n \geq 1\}$

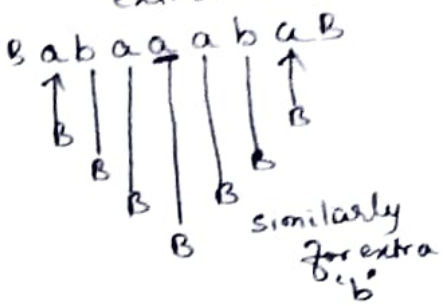
B	a	aa	b	b	b	C	C	C	B

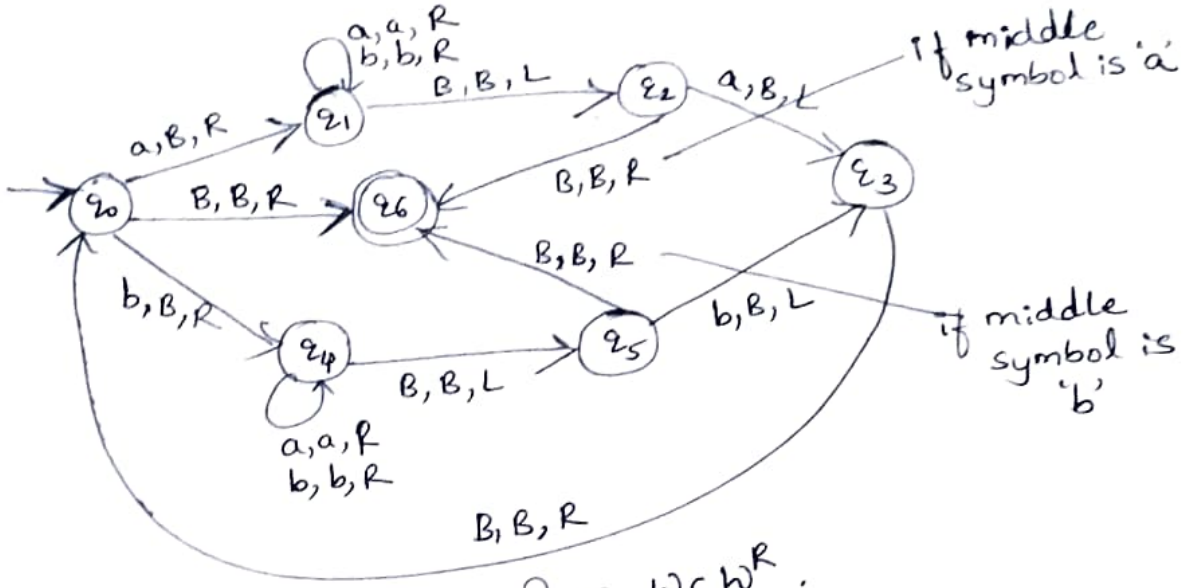


2) Design a turing machine for odd palindrome (or $wa\,w^R$, $w\,b\,w^R$)

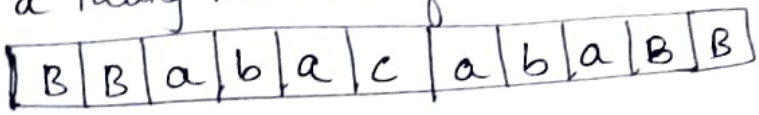


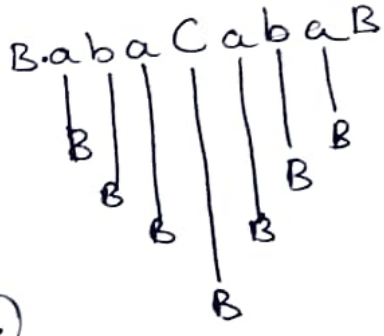


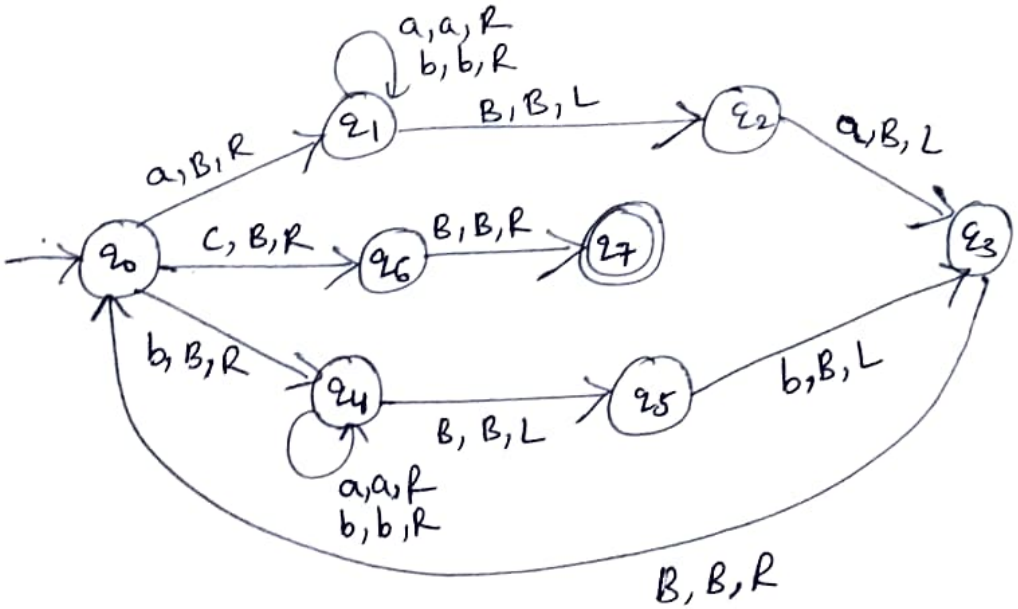




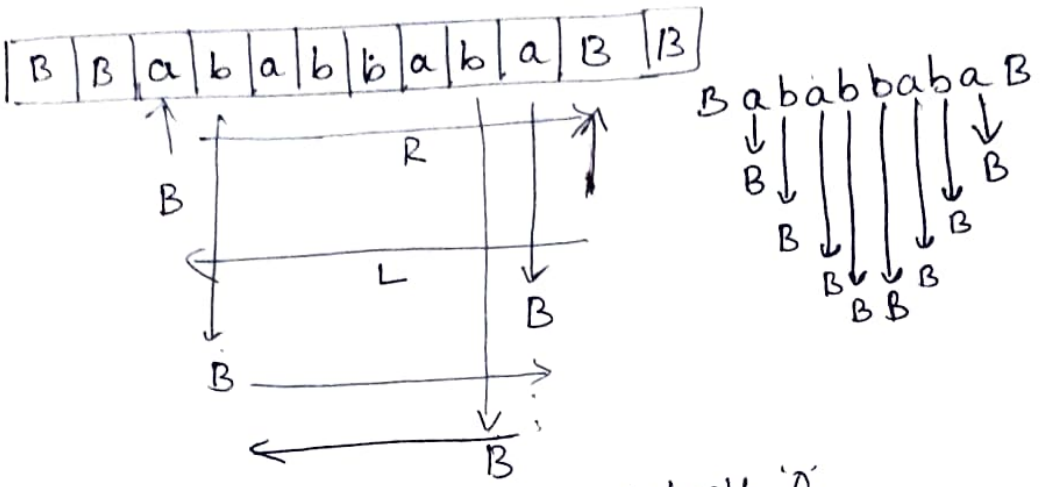
3) Design a turing machine for WCW^R.

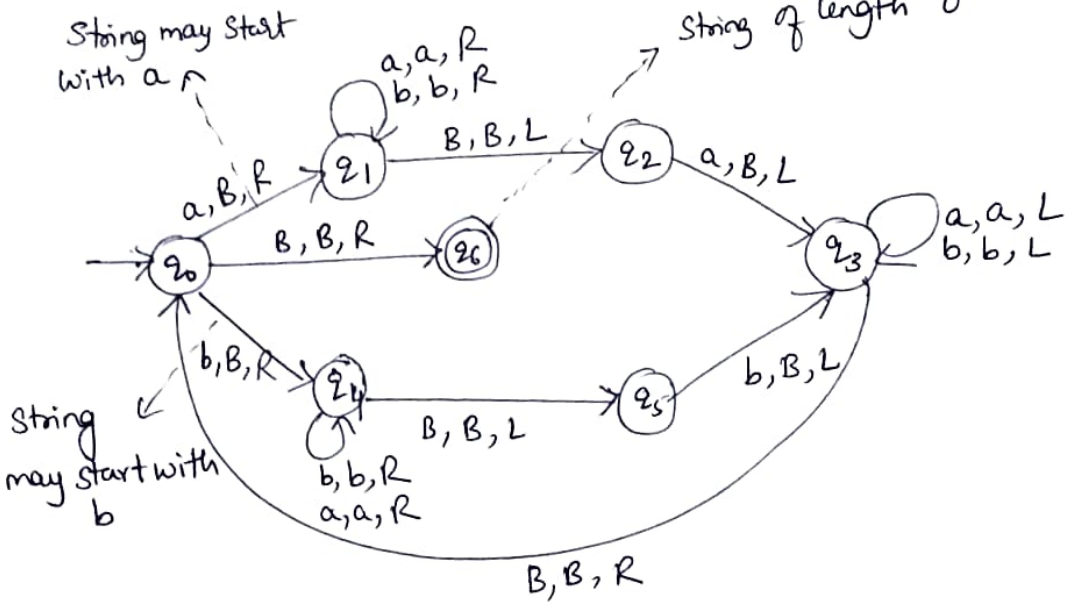






1] Design a turing machine for ww^R or (even palindrome)





~~Design a turing~~ $Q, \Sigma, \delta, \Gamma, B, q_0, F$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a, b, B\}$

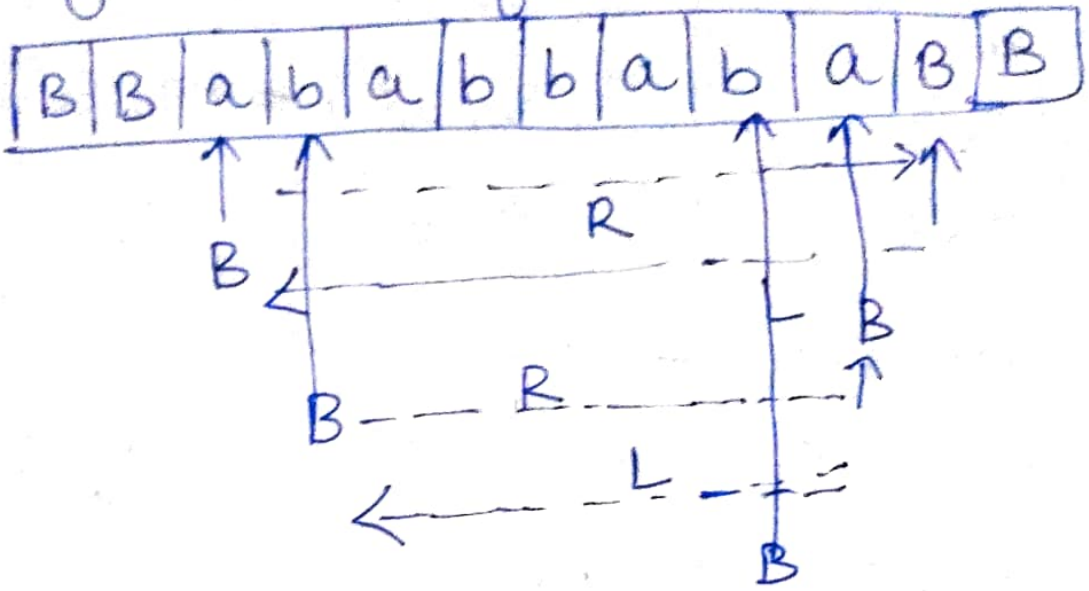
q_0 = Start state

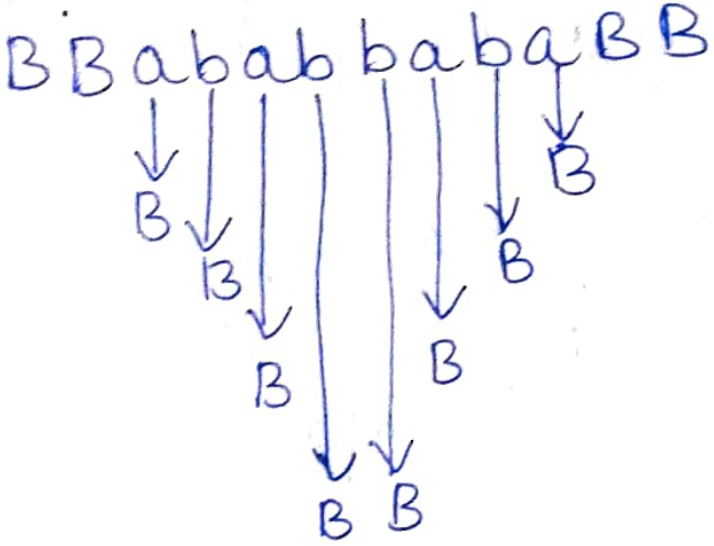
$F = q_6$

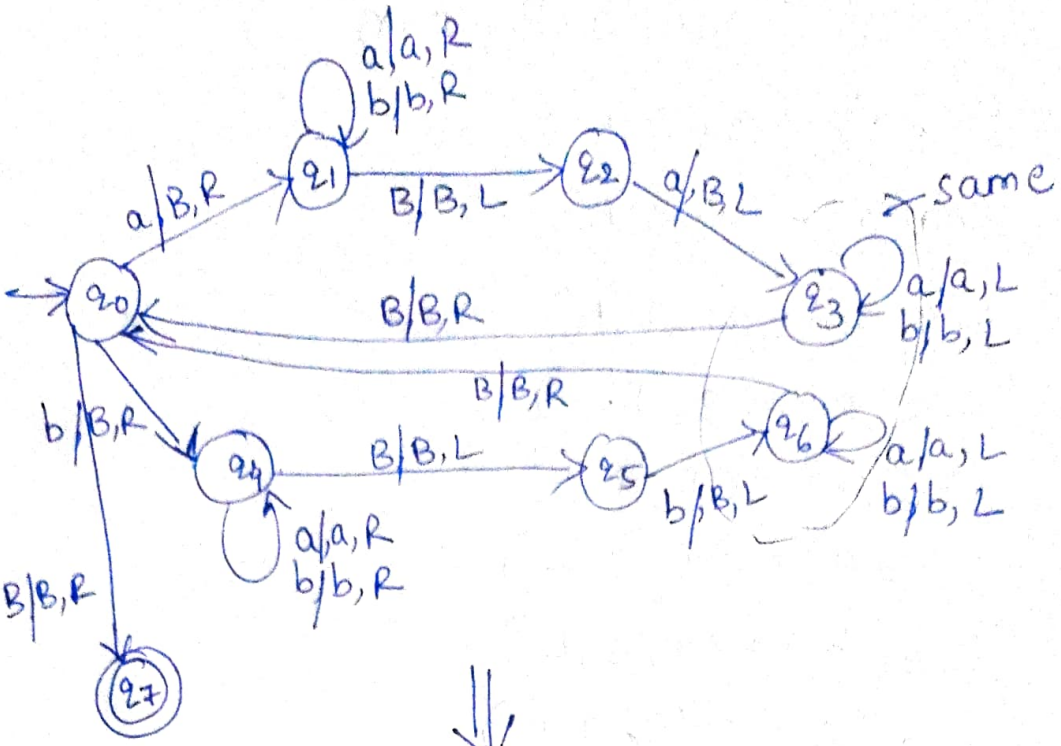
Transition table

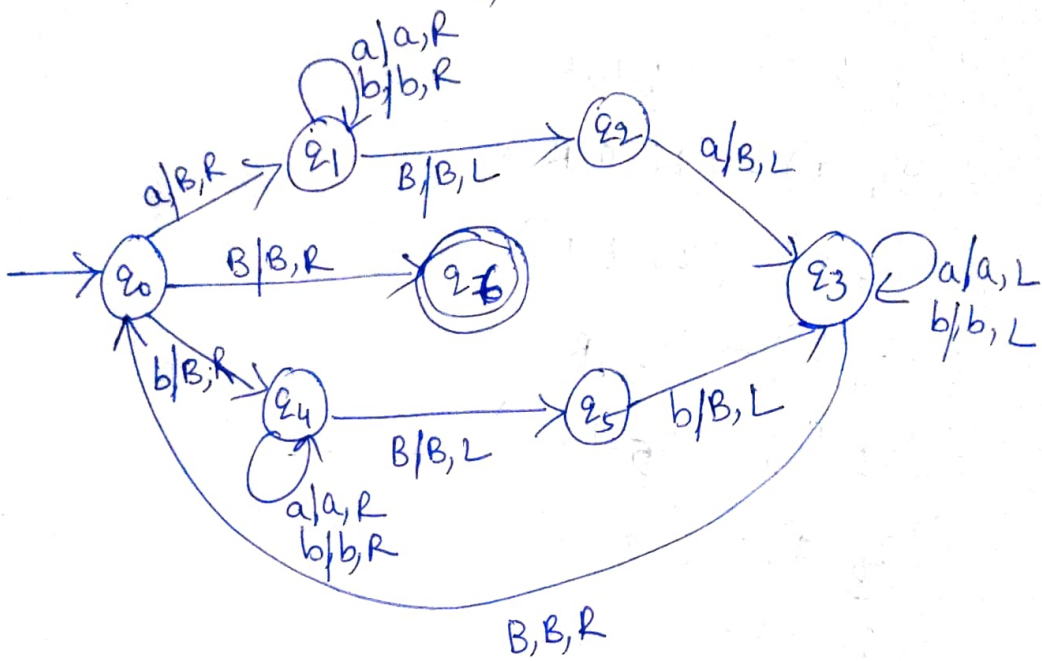
	a	b	B
$\rightarrow q_0$	q_1, B, R	q_4, B, R	q_6, B, R
q_1	q_1, a, R	q_1, b, R	q_2, B, L
q_2	q_3, B, L	-	-
q_3	q_3, a, L	q_3, b, L	q_0, B, R
q_4	q_4, a, R	q_4, b, R	q_5, B, L
q_5	-	q_3, B, L	-
$* q_6$	-	-	-

1) Turing Machine for $\{ww^R / w \in \{a,b\}^*\}$









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

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a, b, B\}$

$q_0 = \{q_0\}$

$B = \{B\}$

$F = \{q_6\}$

δ: Transition function

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

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

$\delta(q_1, b) = (q_1, b, R)$

$\delta(q_1, B) = (q_2, B, L)$

$\delta(q_2, a) = (q_3, B, L)$

$\delta(q_3, a) = (q_3, a, L)$

$\delta(q_3, b) = (q_3, b, L)$

$\delta(q_3, B) = (q_6, B, R)$

$\delta(q_0, b) = (q_4, B, R)$

$\delta(q_4, a) = (q_4, a, R)$

$\delta(q_4, b) = (q_4, b, R)$

$\delta(q_4, B) = (q_5, B, L)$

$\delta(q_5, b) = (q_5, B, L)$

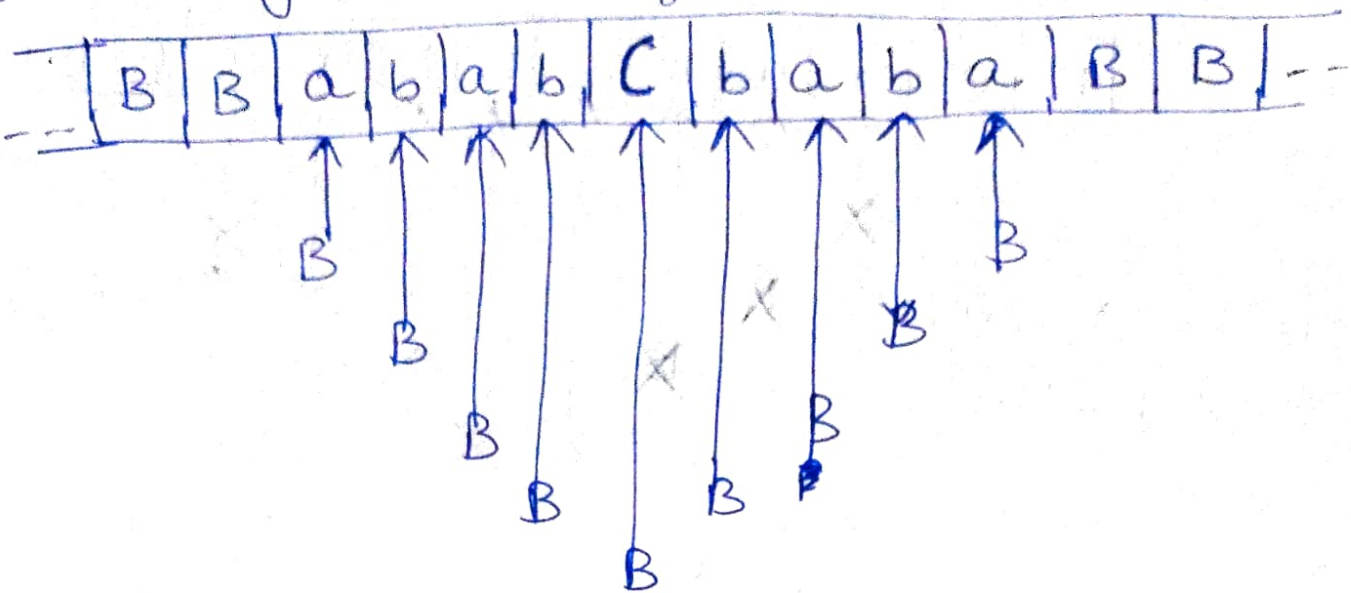
$\delta(q_5, a) = (q_3, a, L)$

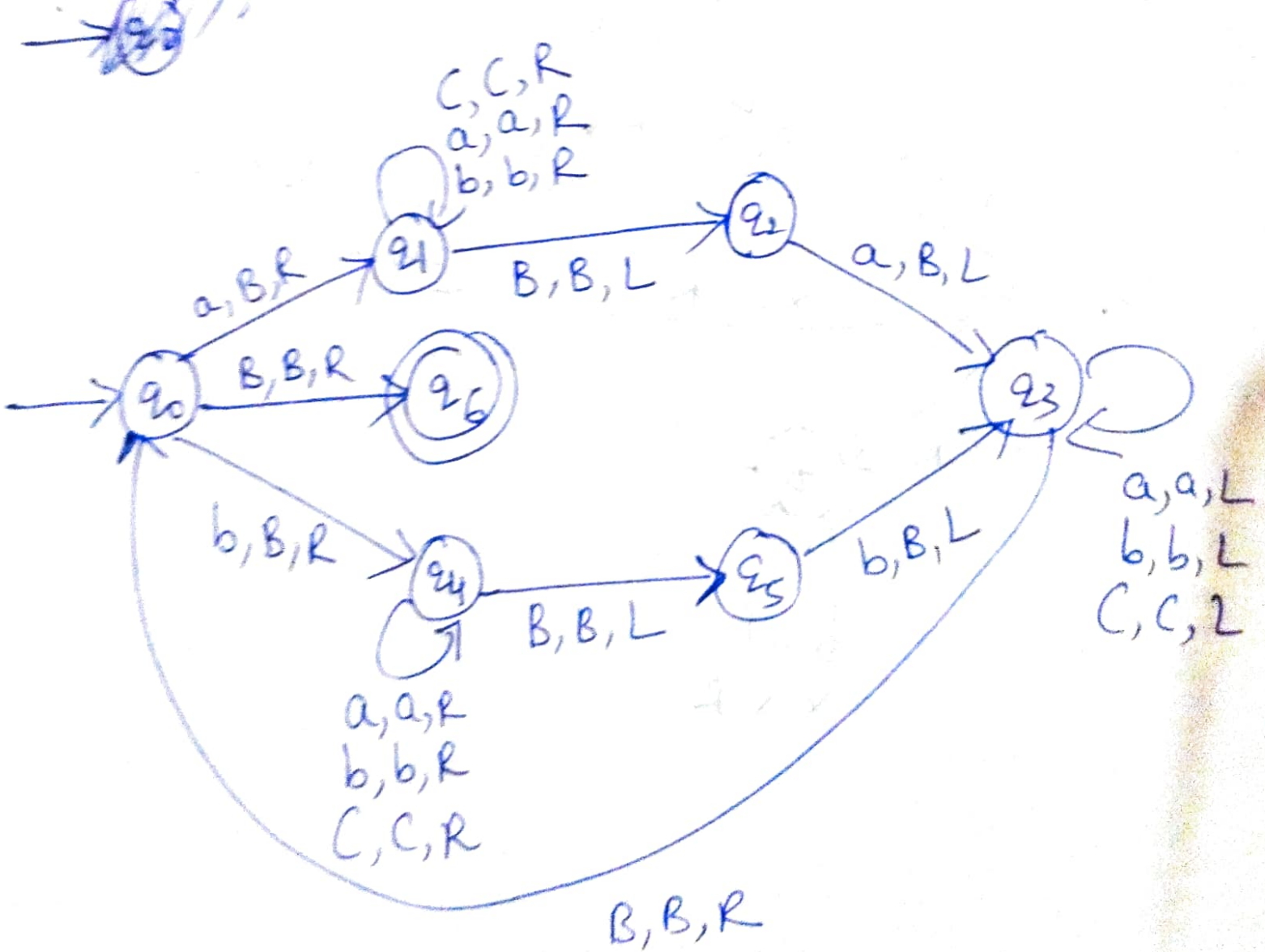
$\delta(q_3, b) = (q_3, b, L)$

$\delta(q_3, B) = (q_0, B, R)$

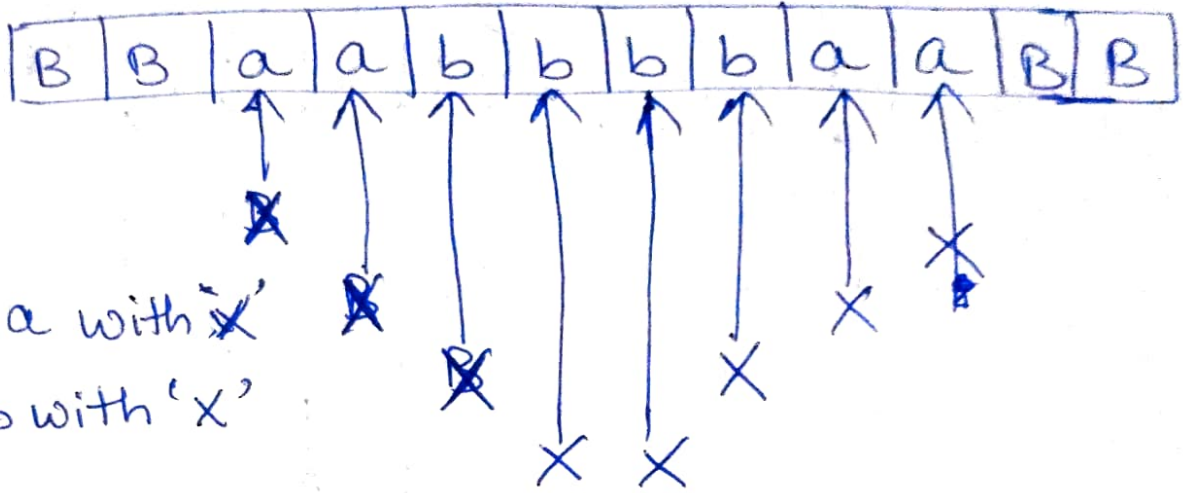
$\delta(q_0, B) = (q_6, B, R)$ // final state

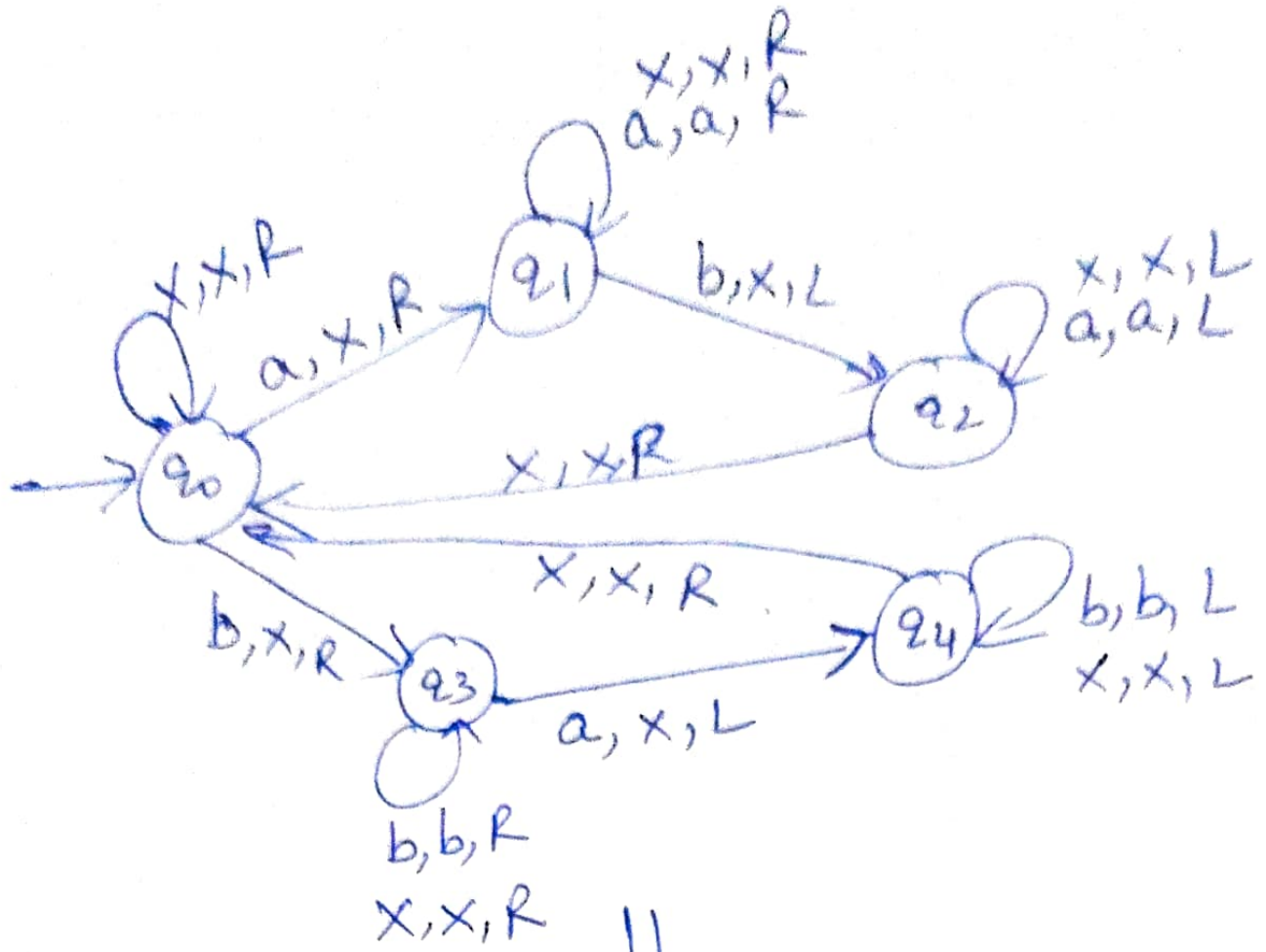
2) Turing Machine for $\{WCW^R / W \in \{a,b\}^*\}$

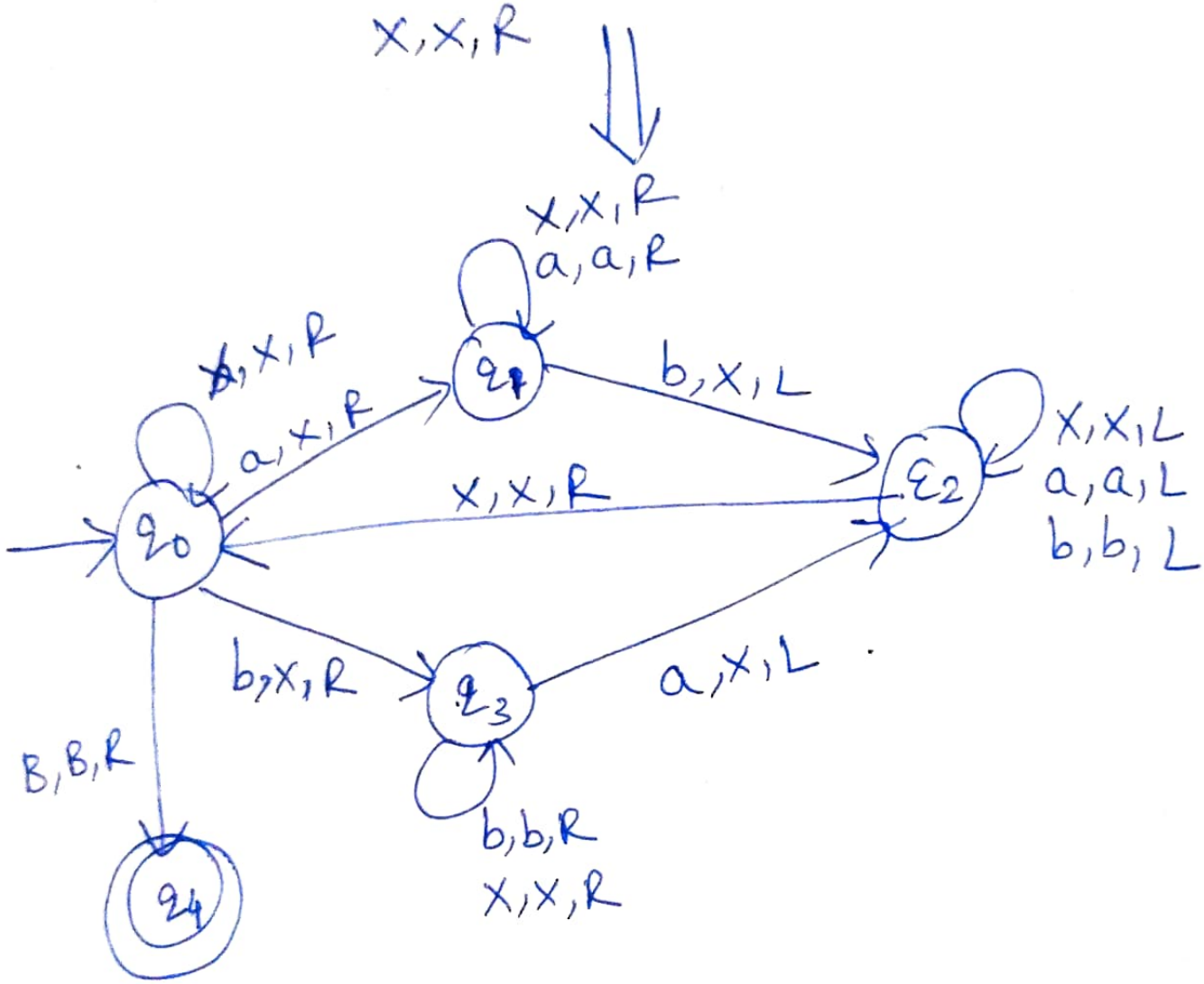



C, C, R
a, a, R
b, b, R
a, B, R
B, B, L
a, B, L
B, B, R
b, B, R
B, B, L
b, B, L
a, a, L
b, b, L
C, C, L
a, a, R
b, b, R
C, C, R
B, B, R

3) Turing Machine for $n_a(w) = n_b(w)$

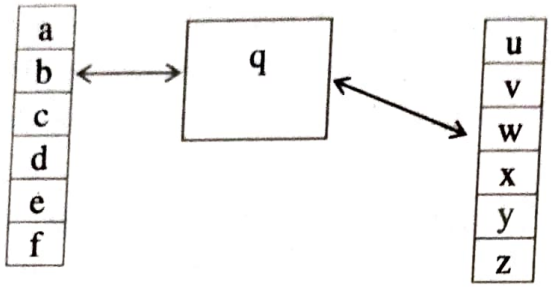



X, X, R
a, a, R
X, X, R
a, X, R
q1
b, X, L
X, X, L
a, a, L
q2
q0
X, X, R
X, X, R
b, X, R
q3
q4
b, b, L
X, X, L
a, X, L
b, b, R
X, X, R


X, X, R
a, a, R
X, X, R
a, X, R
q1
b, X, L
X, X, L
a, a, L
b, b, L
q2
q0
X, X, R
b, X, R
q3
a, X, L
B, B, R
q4
b, b, R
X, X, R

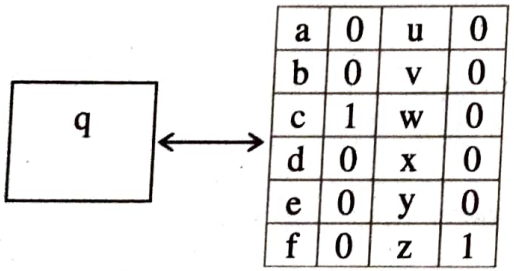
❖ **Theorem:** Every language accepted by a multi-tape TM Is recursively enumerable.

Proof: This can be shown by simulation. For example, consider a TM with two tapes as shown below:



Multi-tape

The above 2-tape TM can be simulated using single tape TM which has four tracks as shown in figure below:



Single tape Multiple tracks

The first and third tracks consist of symbols from first and second tape respectively. The second and fourth track consists of the positions of the read/write head with respect to first and second tape respectively. The value 1 indicates the position of the read/write head. It is clear from the above figure that, the machine in state q and when the first read/write head points to the symbol *c*, the second read/write head points to the symbol *z*, then the machine enters into state p, if and only if this transition is defined for TM with multi-tapes. So, whatever transitions have been applied for multi-tape TM, if we apply the same transitions for the new machine that we have constructed, then the two machines are equivalent.

Non-deterministic Turing Machine

→ It is similar to deterministic turing machine except that for any i/p symbol and current state it has a number of choices

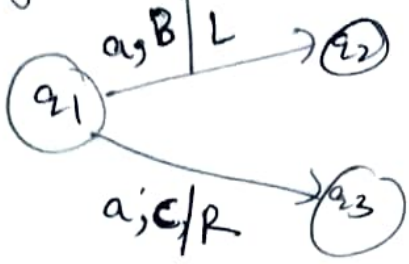
→ A string is accepted by a Non-deterministic turing machine if there is a sequence of moves that leads to a final state.

→ The transition function

$$\delta : Q \times X \longrightarrow 2^{Q \times X \times (L,R)}$$

(X: Tape symbol)

→ A NDTM is allowed to have more than one transition in a given tape symbol.



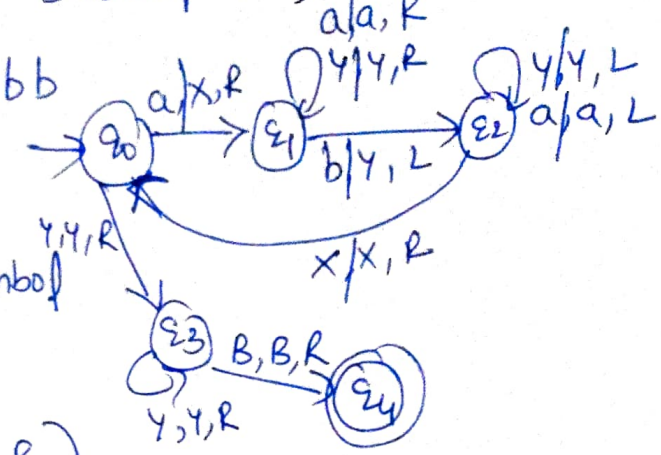
Def^n : The non-deterministic turing machine $M = (Q, \Sigma, T, \delta, q_0, B, F)$ where

Q → set of finite states
Σ → set of input alphabets
T – set of tape symbols
δ – transition function from $Q \times T$ to $2^{Q \times T \times (L,R)}$
q_0 – start state
B → blank character
$F \subseteq Q$ → set of final states

Turing machine $L = \{a^n b^n \mid n \geq 1\}$

Instantaneous Description

$n = 2$, $aabb$



$q_0 aabbB$

Start state | String | Blank symbol

$\vdash q_0 aabbB$

$(q_0, a) = (q_1, X, R)$

Replace a with X, move right with q_1 state

$\vdash X q_1 abbB$

$(q_1, a) = (q_1, a, R)$

don't Replace a with any symbol, keep as it is. move right with q_1.

$\vdash Xa q_1 bbB$

$(q_1, b) = (q_2, Y, L)$

replace b with y and move left with state q_2.

$\vdash X q_2 aYbB$

$(q_2 a) = (q_2, a, L)$

Keep a as it is, move left with q_2.

$\vdash q_2 XaYbB$

$(q_2, X) = (q_0, X, R)$

Keep X as it is, move right with q_0.

$\vdash X q_0 aYbB$

$(q_0, a) = (q_1, X, R)$

$\vdash \$ X X q_1 Y B B$ $\quad (q_1, Y) = (q_1, Y, R)$

$\vdash X X Y q_1 b B$ $\quad (q_1, b) = (q_2, Y, L)$

$\vdash X X q_2 Y Y B$ $\quad (q_2, Y) = (q_2, Y, L)$

$\vdash X q_2 X Y Y B$ $\quad (q_2, X) = (q_0, X, R)$

$\vdash X X q_0 Y Y B$ $\quad (q_0, Y) = (q_3, Y, R)$

$\vdash X X Y q_3 Y B$ $\quad (q_3, Y) = (q_3, Y, R)$

$\vdash X X Y Y q_3 B$ $\quad (q_3, B) = (q_4, B, R)$

$\vdash X X Y Y B q_4$

Programming Techniques in turing machines and Halting problem

Programming techniques for Turing Machine

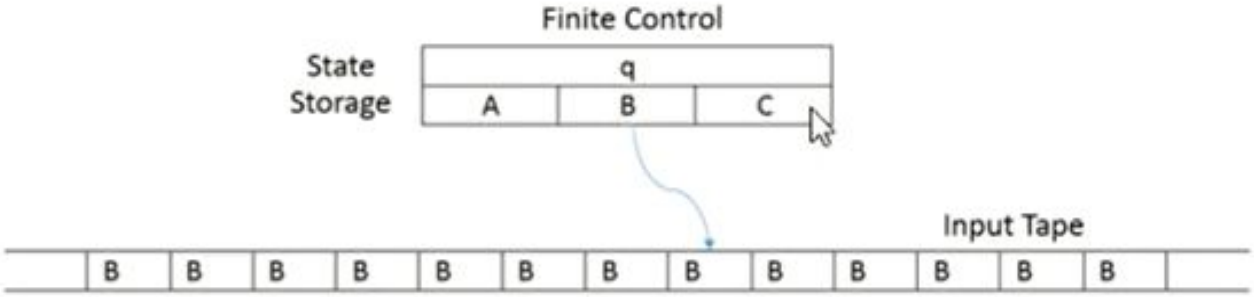
- The programming techniques that are used to construct an efficient Turing machine that functions as powerful as conventional computer.
- The different techniques that are used to design a Turing machine are as follows.
  - Storage in finite control
  - Multiple Tracks
  - Subroutines

Programming Techniques in Turing machine

- Storage in Finite Control
- Multiple Tracks in Turing Machine
- Subroutines

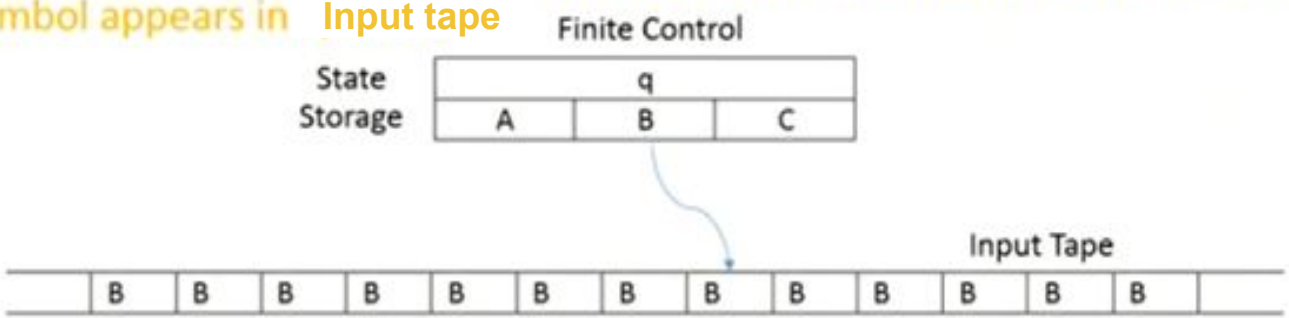
1) Storage in Finite control

- In Turing machine, finite control contains the finite automata with the state transitions. It represents the set of states. But in storage of finite control, we store the data along with the state. So here we use the finite control to hold finite amount of data and it shown below,

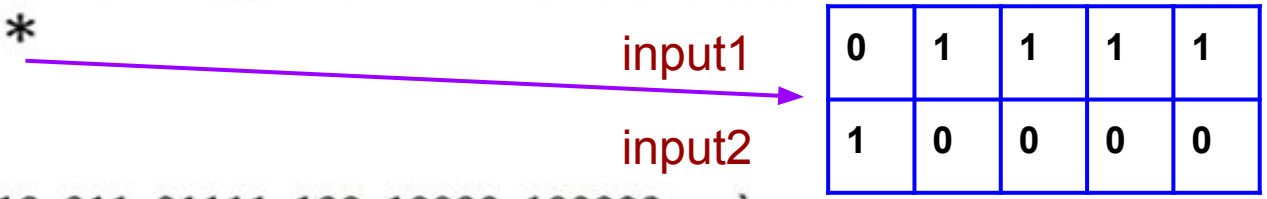


Storage in Finite control

- Turing machine makes the state to remember and to have a memory for the symbol scanned in the input. The state is q, and this state q contains A,B,C as the symbol in storage with q.
  - It can be designed to store in the state with any data from the input.
  - Each state contains the 'B' blank symbol as its storage initially.
  - It is used to store any symbol in the input and to check whether the stored symbol appears in Input tape

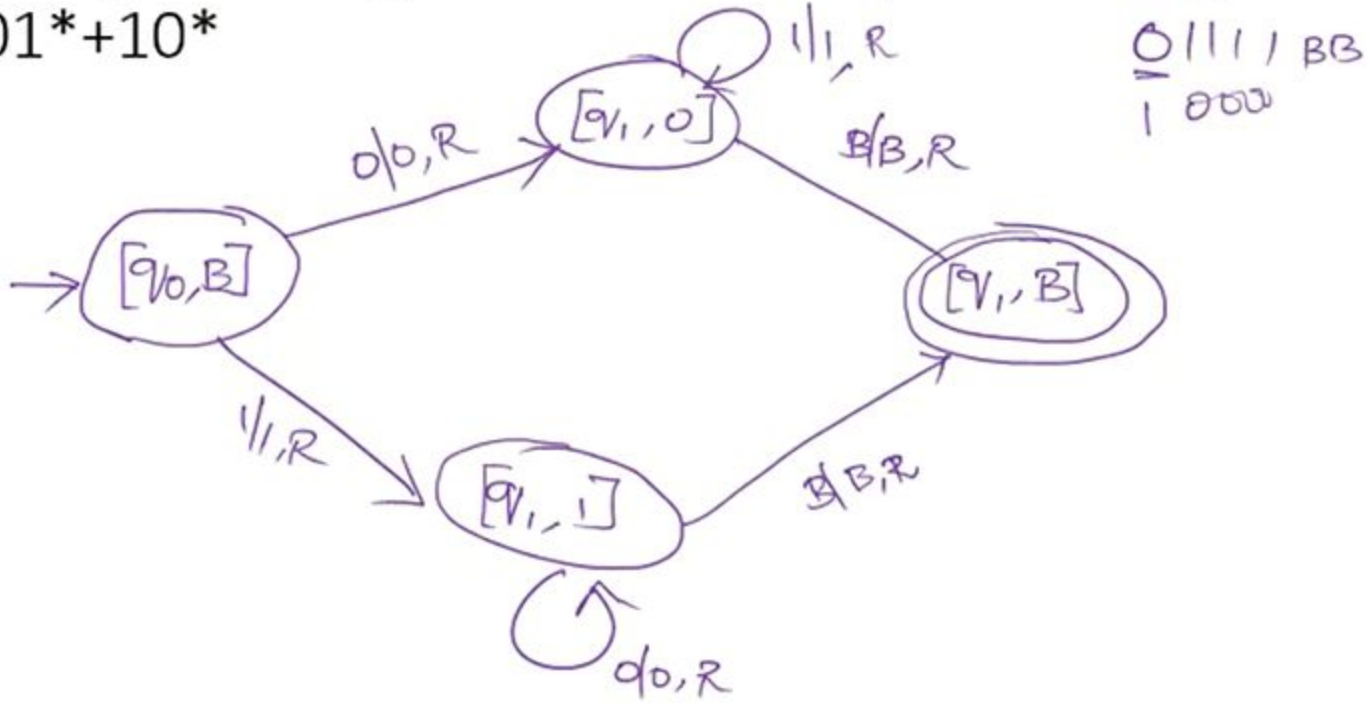


Design a Turing machine to accept the string 01*+10*



- Soln:
  - L= { 01, 10, 011, 01111, 100, 10000, 100000 .....}
- Steps:
  - At state initial state [q0,B], if it finds '0' or '1', it stores in its state without replacing and moves right and enters the state q1.
  - At state [q1,0], if it finds '1', then it skips all '1' and moves right and remains in this same state.
  - At state [q1,1], if it finds '0', then it skips all '0' and moves right and remains in this same state.
  - At state [q1,0], if it finds 'B', then it reaches the accepting state [q1,B].
  - At state [q1,1], if it finds 'B', then it reaches the accepting state [q1,B].

Design a Turing machine to accept the string 01*+10*



0111 / BB
1 000

Design a Turing machine to accept the string 01*+10*

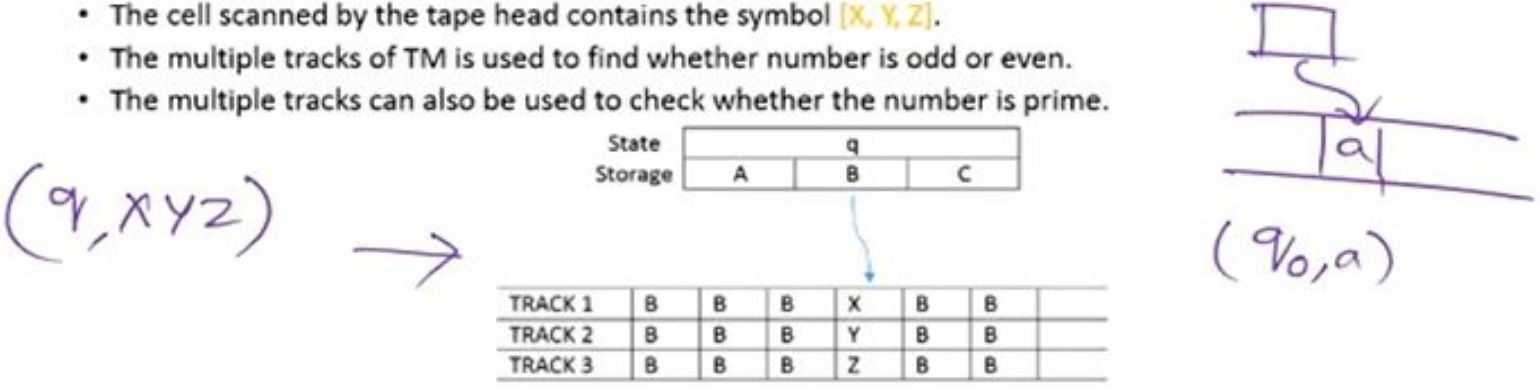
State	0	1	B
→[q0,B]	([q1,0], 0, R)	([q1,1], 1, R)	--
[q1,0]	--	([q1,0], 1, R)	([q1,B], B, R)
[q1,1]	([q1,1], 0, R)	--	([q1,B], B, R)
*[q1,B]	--	--	--

$M = \{ \{[q_0, B], [q_1, 0], [q_1, 1], [q_1, B]\}, \{0, 1\}, \{0, 1, B\}, \delta, [q_0, B], 'B', \{[q_1, B]\}\}$

$\delta([q_0, B], 0) = ([q_1, 0], 0, R)$

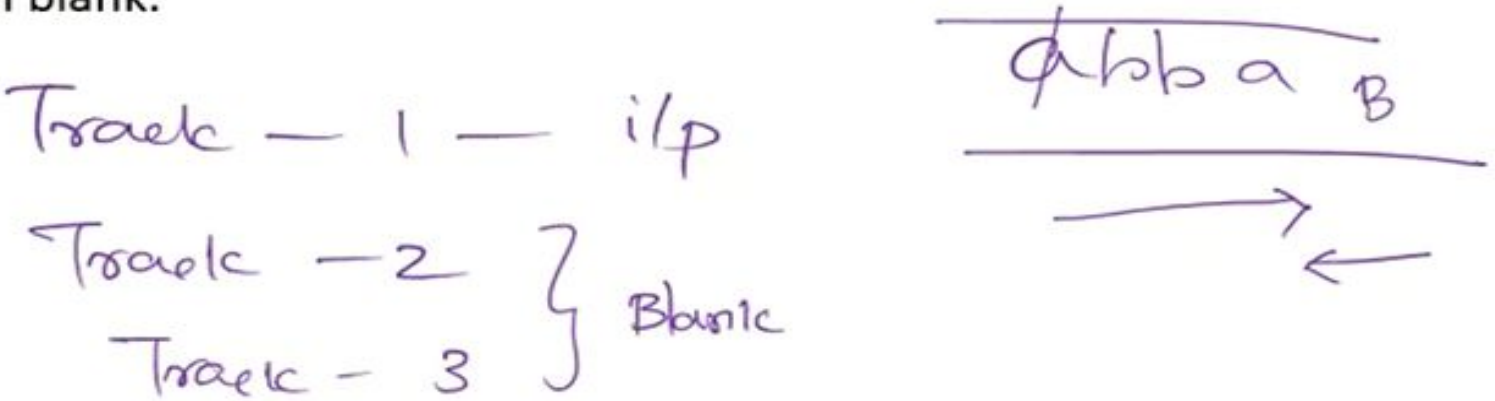
2) Multiple Tracks in Turing Machine

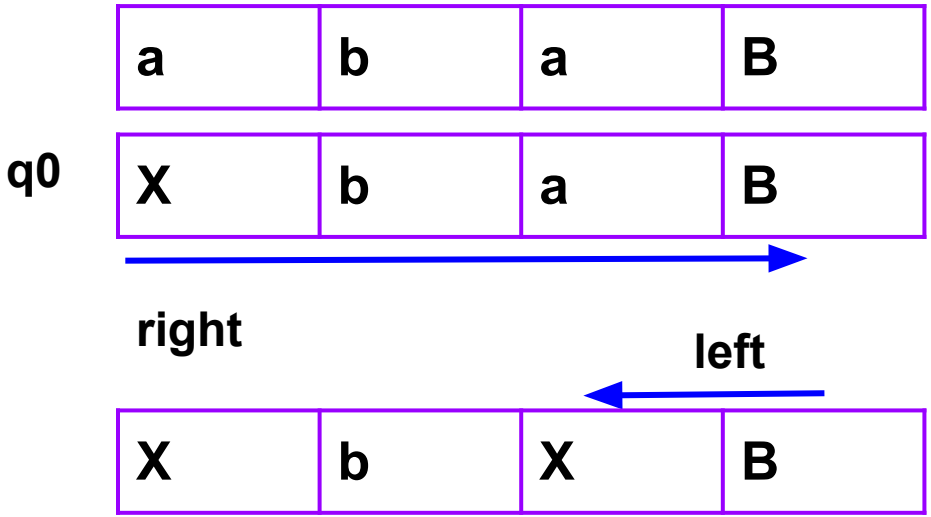
- In storage in Finite Control
  - 2 components – 1. State and 2. Memory – stores symbols.
- Now we extend Turing machine to include multiple tracks in the input tape
  - In this TM, where the finite control contains state and its storage and the input tape contains multiple tracks.
  - Each track in the input tape contains one symbol.
  - The tape alphabet of TM consists of tuples with one component in each track and the number of components in the tuples depends on the number of tracks of the input tape.
  - The cell scanned by the tape head contains the symbol [X, Y, Z].
  - The multiple tracks of TM is used to find whether number is odd or even.
  - The multiple tracks can also be used to check whether the number is prime.



Design a Turing machine using multiple tracks to check whether the given input number is palindrome over {a,b}.

- Solution:
- The idea to design this TM is that let us store the input symbol in the FIRST track of input tape. Let us keep SECOND and THIRD Tape filled with blank.

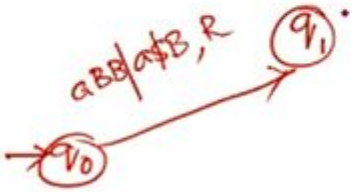



a
b
a
B
q0
X
b
a
B
right
left
X
b
X
B

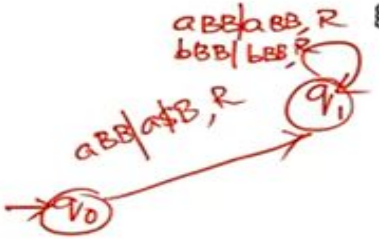
track1	a	b	b	a
track2	B	B	B	B
track3	B	B	B	B

input

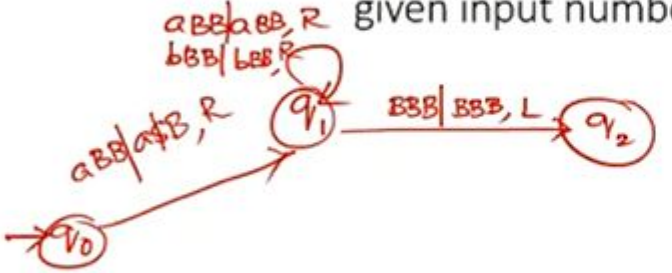
Design a Turing machine using multiple tracks to check whether the given input number is palindrome over {a,b}.



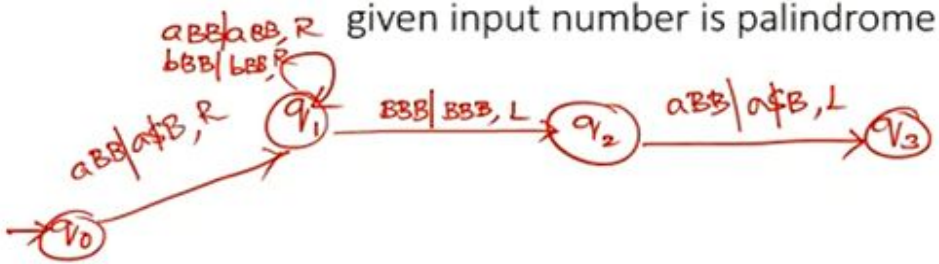
Design a Turing machine using multiple tracks to check whether the given input number is palindrome over {a,b}.

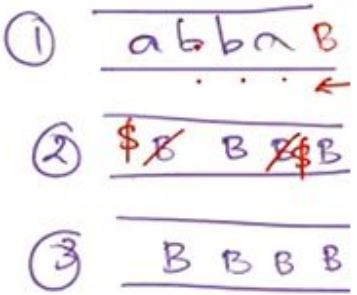


Design a Turing machine using multiple tracks to check whether the given input number is palindrome over {a,b}.

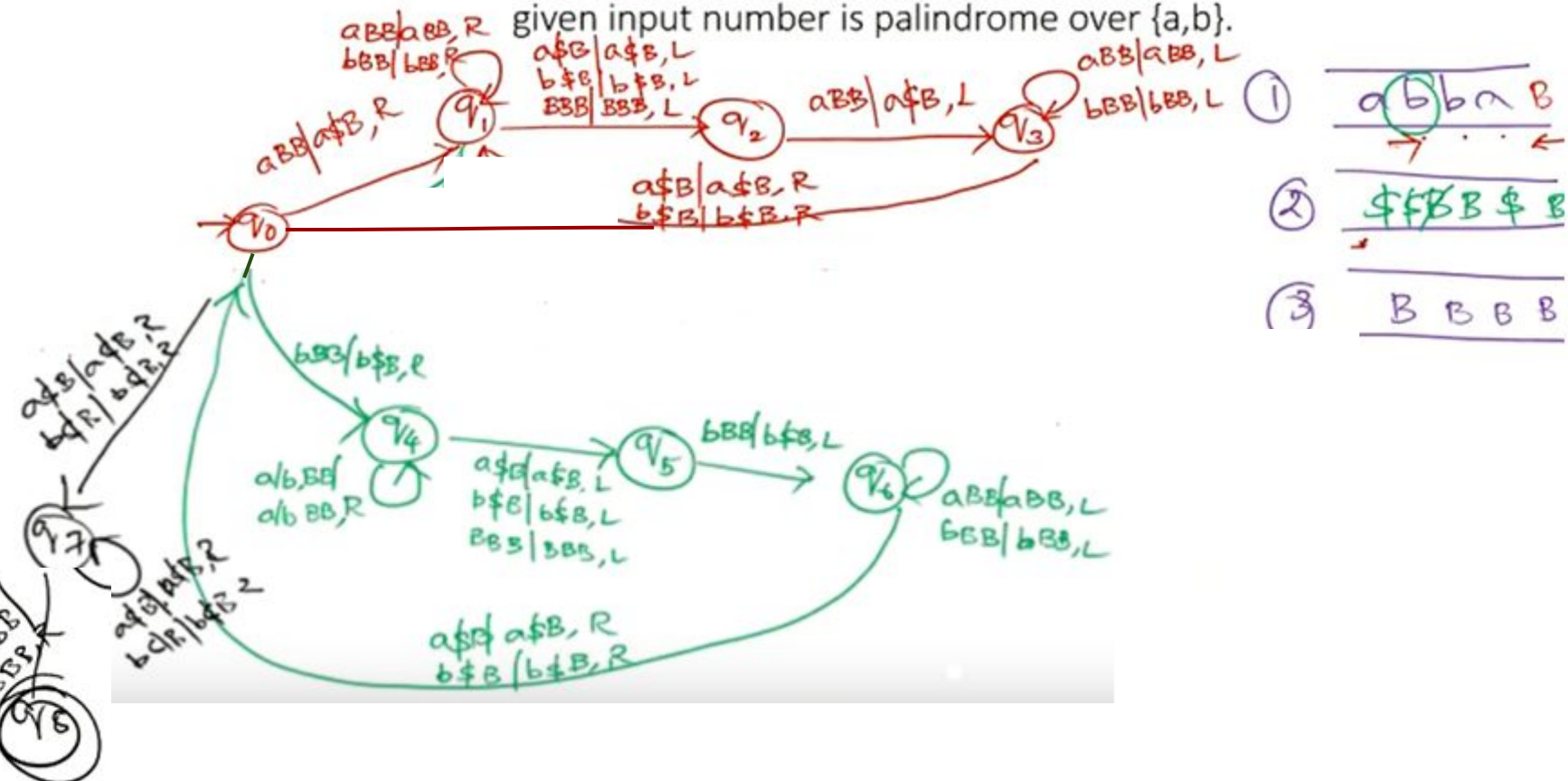


Design a Turing machine using multiple tracks to check whether the given input number is palindrome over {a,b}.





Design a Turing machine using multiple tracks to check whether the given input number is palindrome over {a,b}.



Design a Turing machine using multiple tracks to check whether the given input number is palindrome over {a,b}.

$M = \{ \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\} \times \{a,b\}, \{a,b\}, \{a,b,B\}, \delta, 'B', \{q_0\}, \{q_8\} \}$ //

State	aBB	a$B	bBB	b$B	BBB
→q0	(q1, a$B, R)	--	(q4, b$B, R)	--	--
q1	(q1, aBB, R)	(q2, a$B, L)	(q1, bBB, R)	(q2, b$B, L)	(q2,BBB, L)
q2	(q3, a$B, L)	(q7, a$$, R)	--	(q7, b$$, R)	--
q3	(q3, aBB, L)	(q0, a$B, R)	(q3, bBB, L)	(q0, b$B, R)	--
q4	(q4, aBB, R)	(q5, a$B, L)	(q4, bBB, R)	(q5, b$B, L)	(q5,BBB, L)
q5	--	(q7, a$$, R)	(q6, b$B, L)	(q7, b$$, R)	--
q6	(q6, aBB, L)	(q0, a$B, R)	(q6, bBB, L)	(q0, b$B, R)	--
*q7	--	--	--	--	--

3) Subroutines

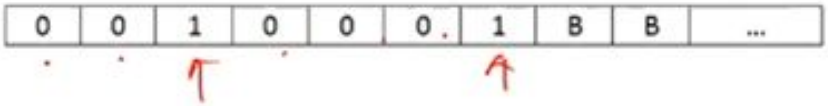
- In Some problems, some tasks needed to be performed repeatedly and it can be done by subroutines called as FUNCTION.
- Subroutine in TM is a set of states that specifically performs some tasks.
  - The set if states in the subroutine has one start state and another state namely the return state.
  - The return state of the subroutine does not have moves and it pass the control to other set of states of the TM that calls the subroutine.
  - The subroutine is called whenever there is transition to its initial state.
  - The calls are made to the start states of different copies of the subroutine and each copy returns to different state.
  - The subroutine of the TM performs some task simultaneously.

Design a Turing to perform the multiplication function F(m,n) = m*n using subroutine.

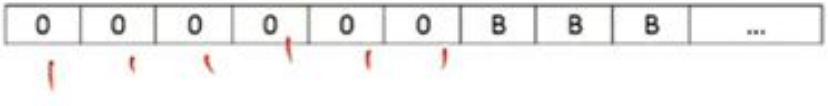
- The idea to design this TM is that we place the input as 0^m10^n1 on the Turing machine. The multiplication is done by performing successive addition and it is shown below.

M=2 and n=3

Input:



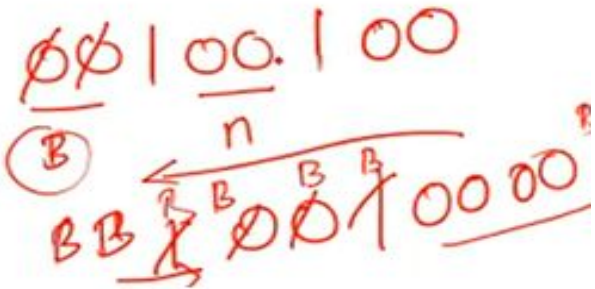
Output:



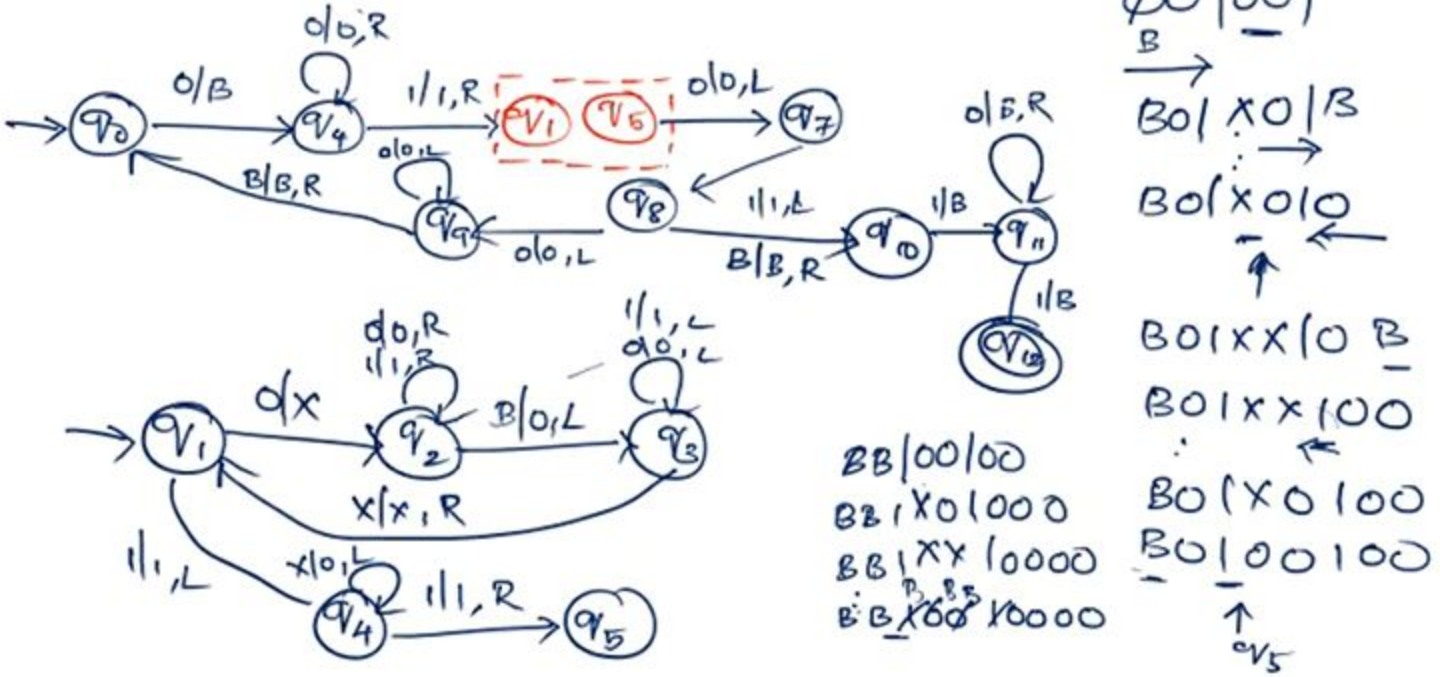
2 arguments:
0^m
0^n

Design a Turing to perform the multiplication function F(m,n) = m*n using subroutine.

- Steps:
- At initial state, when '0' is found in the input, replace it to B and move right for searching 1.
- After finding '1', call the subroutine to copy the 'n' numbers of '0's for 'm' number of times.
- After performing 'm' number of copy with 'n' number of '0's, we replace 0^m10^n1 to 'B' and tape contains 0^{mn}.



Design a Turing to perform the multiplication function F(m,n) = m*n using subroutine.



Design a Turing to perform the multiplication function F(m,n) = m*n using subroutine.

Main function.

State	0	1	B
q0	(q6,B,R)	-	-
q5	(q7,0,L)	-	-
q6	(q6,0,R)	(q1,1,R)	-
q7	-	(q8,1,L)	-
q8	(q9,0,L)	-	(q10,B,R)
q9	(q9,0,L)	-	(q10,B,R)
q10	-	(q11,B,R)	-
q11	(q11,B,R)	(q12,B,R)	-
q12	-	-	-

Subroutine

State	0	1	X	B
q1	(q2,X,R)	(q4,1,L)	-	-
q2	(q2,0,R)	(q2,1,R)	-	(q3,0,L)
q3	(q3,0,L)	(q3,1,L)	(q1,X,R)	-
q4	-	(q5,1,R)	(q4,0,L)	-

Halting Problem

- The problem which are insolvable by computers are called intractable or undecidable problem. Alan Turing proved some problems are undecidable which is known as halting problem.
- Given a Turing machine and input to the Turing machine, does the Turing machine finish computing in a finite number of times. In order to solve the problem, an answer either yes or no, must be given in a finite amount of time regardless of the machine or input in question.
- To see that no Turing machine can solve the halting problem, we begin by assuming that such a machine exists, and then show that its existence is self contradictory. We call the machine as halting machine.
- Halting machine is a machine that operates on another machine and its inputs to produce a yes or no answer in finite time either the machine in question finishes in finite time or it does not.

Halting Problem

Given a turing machine, will it halt when run on some particular given input string?

Given some program written in some language(Java/C/etc.) will it ever it get into an infinite loop or will it always terminate?

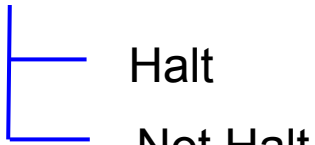
Answer:

- In general, we can’t always know.
- The best we can do is run the program and see whether it halts.
- For many programs we can see that it will always halt or sometimes loop

Can we design a machine which if given a program can find out or decide if that program will always halt or not on a particular input?

Let us assume that we can:

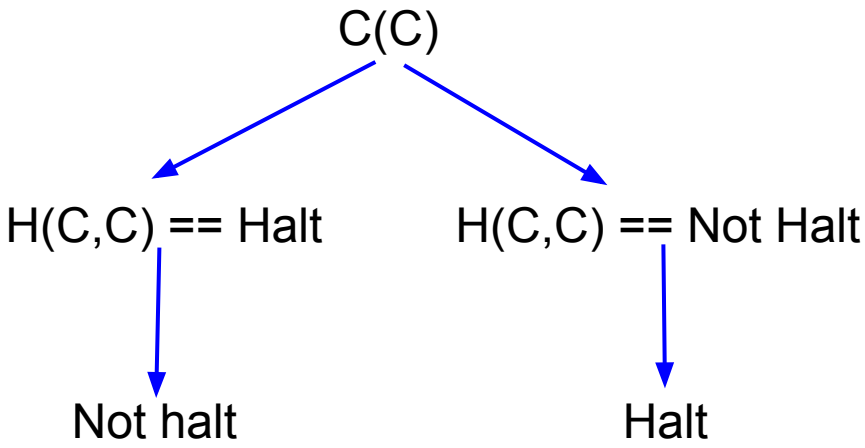
H(P,I)



This allows us to write another program:

```
C(X)
    if(H(X,X) == Halt)
       Loop Forever;
    else
       Return;
```

If we run ‘C’ on itself:



Extensions or variations or variants of turing machine

- Multitape turing machine—-- refer notes
- Multitrack turing machine
- Non-deterministic turing machine—-refer notes
- Two way infinite tape
- Turing machine with semi-infinite tape

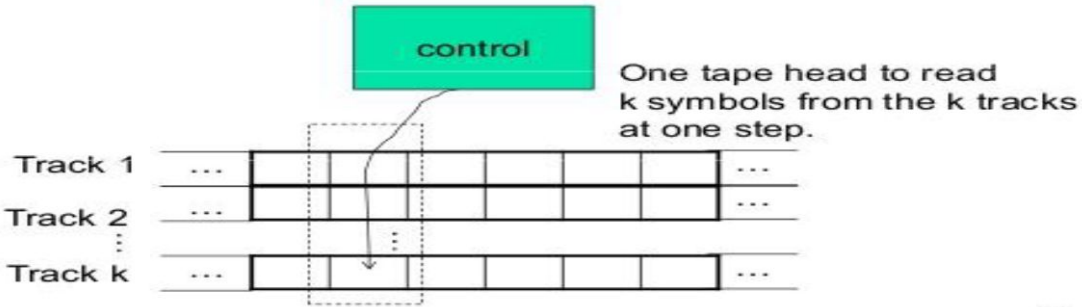
MULTI-TRACK TM

- 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.

- **δ** is a relation on states and symbols where
- $\delta(Q_i, [a_1, a_2, a_3, ...]) = (Q_j, [b_1, b_2, b_3, ...]$, Left_shift or Right_shift)

Multi-track Turing Machines

- TM with multiple tracks,
  but just one unified tape head



18

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

- **Q** is a finite set of states
- **X** is the tape alphabet
- **Σ** is the input alphabet
- **δ** is a relation on states and symbols where

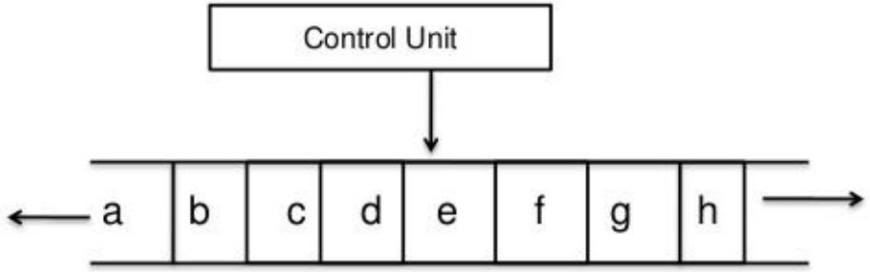
  $\delta(Q_i, [a_1, a_2, a_3, ...]) = (Q_j, [b_1, b_2, b_3, ...], \text{Left\_shift or Right\_shift})$

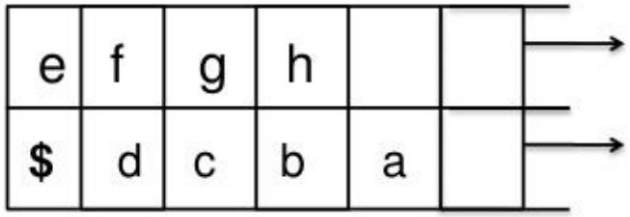
- $\mathbf{q_0}$ is the initial state
- **F** is the set of final states

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

Two way infinite tape:

- Two way infinite tape simulated by semi -infinite tape

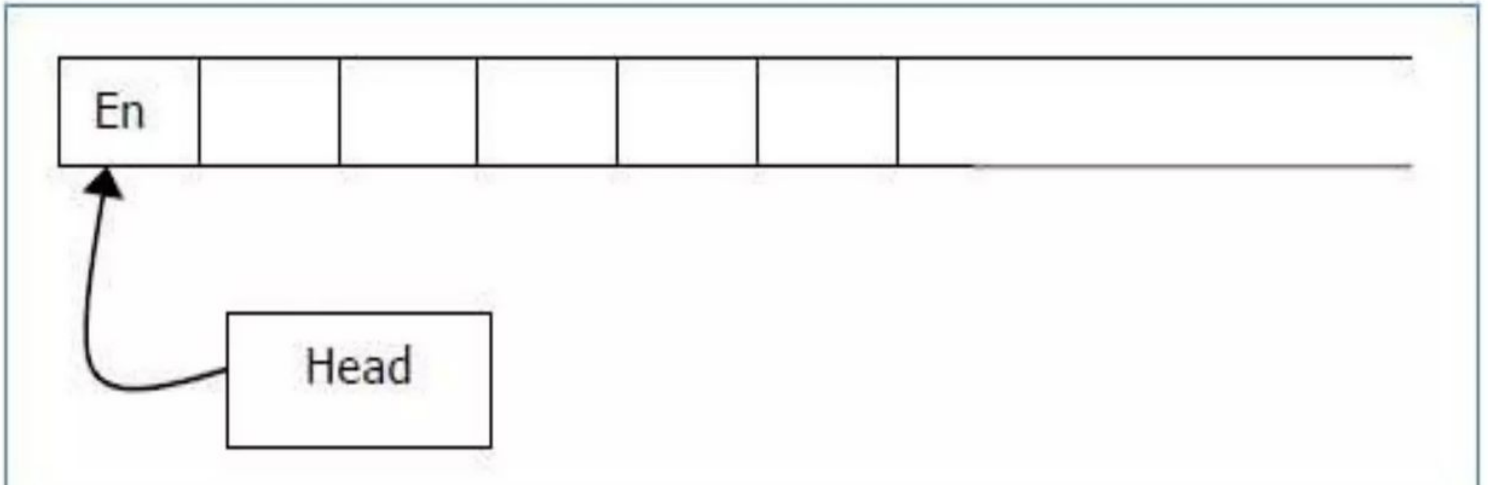




A two way infinite tape TM can move indefinitely in either direction. This can be shown to be equivalent to a one way infinite tape TM by the following argument: It is clear that two one-way infinite tape TMs can be used to simulate a two-way infinite tape. If these are lined up ``side by side'' and the two tapes are interleaved, then a single one-way infinite TM can be constructed which is equivalent.

TM WITH SEMI-INFINITE TAPE

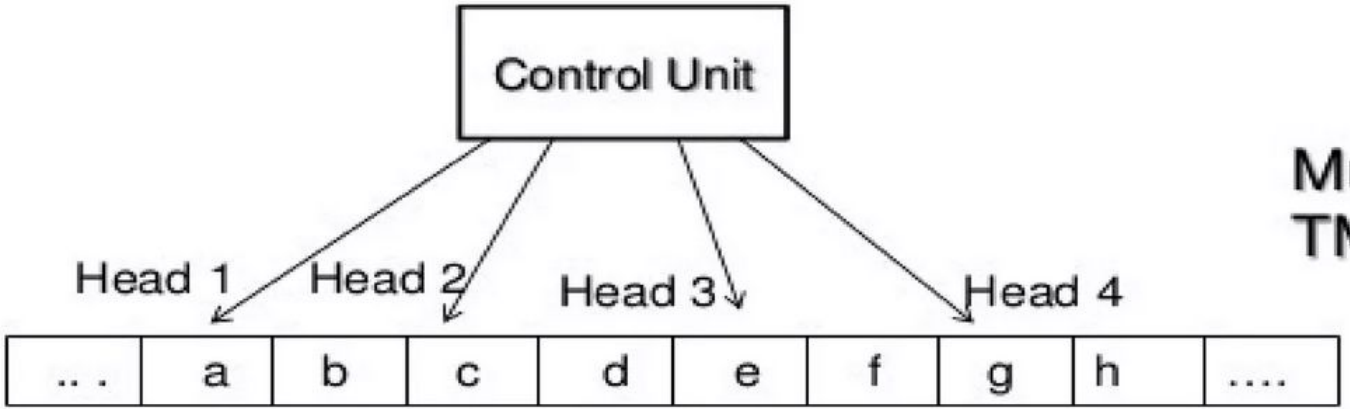
- A Turing Machine with a semi-infinite tape has a left end but no right end. The left end is limited with an end marker.



MULTI HEAD TURING MACHINE

- A multi head Turing machine is a single tape TM having n heads reading symbols on the same tape. In one step all the heads sense the scanned symbols and move or write independently.
- The heads are numbered 1 through n and move of TM depends upon the state and symbols scanned by each head.
- In one move the heads may move left , right or remain stationary.
- This type of TM is as powerful as one tape Turing machine.
- The multi head Turing Machine is as shown in the following

MULTI HEAD TURING MACHINE

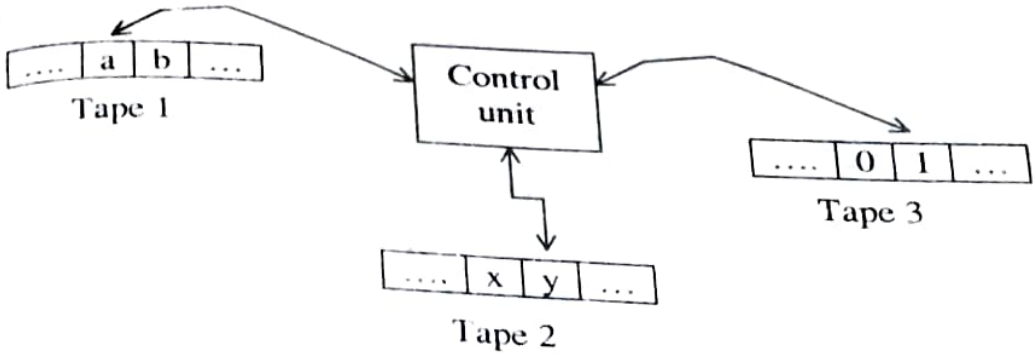


Multihead TM

(restricted TMs) are discussed in the chapter 12.

10.1 Multi-tape Turing Machines

A multi-tape Turing Machine is nothing but a standard Turing Machine with more number of tapes. Each tape is controlled independently with independent read-write head. The pictorial representation of multi-tape Turing machine is shown in figure below:



The various components of multi-tape Turing Machine are:

a. Finite control
b. Each tape having its own symbols and read/write head.

Each tape is divided into cells which can hold any symbol from the given alphabet. To start with the TM should be in start state q_0. If the read/write head pointing to tape 1 moves towards right, the read/write head pointing to tape 2 and tape 3 may move towards right or left depending on the transition. The formal definition of Multi-tape Turing machine can be defined as follows.

Definition: The Multi-tape Turing Machine is an n-tape machine

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

where

Q is set of finite states
Σ is set of input alphabets
Γ is set of tape symbols
δ is transition function from $Q \times \Gamma^n$ to $Q \times \Gamma^n \times \{L,R\}^n$
q_0 is the start state
B is a special symbol indicating blank character
$F \subseteq Q$ is set of final states

The move of the multi-tape TM depends on the current state and the scanned symbol by each of the tape heads. For example, if number of tapes in TM is 3 as shown in the figure and if there is a transition

$$\delta(q, a, b, c) = (p, x, y, z, L, R, S)$$

where q is the current state. The transition can be interpreted as follows. The TM in state q will be moved to state p only when the first read/write head points to a, the second read-write head points to b and third read/write head points to c and the read/write head will be moved to left in the first case and right in the second case. But, the read/write head with respect to third tape will not be altered. At the same time, the symbols a, b and c will be replaced by x, y and z. It can be easily shown that the n-tape TM in fact is equivalent to the single tape Standard Turing Machine as shown below.

np complete and np hard

Based on Time Complexity:

1) Polynomial Time Algorithms: (Time complexity is less)

Ex: Linear Search O(n), Bubble Sort (O(n^2)), Merge Sort(O(nlogn),etc

2) Non-Polynomial (or) Exponential Algorithms: (Time complexity is high)

Ex: Travelling Salesman Problem (O(n^2 2^n), Knapsack problem (O(2^n/2))

Based on Result:

- Decision problems
  - ➢Result is yes or no {0,1}
- Optimization problems
  - ➢Result is a number representing an objective value.

P Class – Problem:

- P is set of problems that can be solved (deterministic) in Polynomial [P] Time.

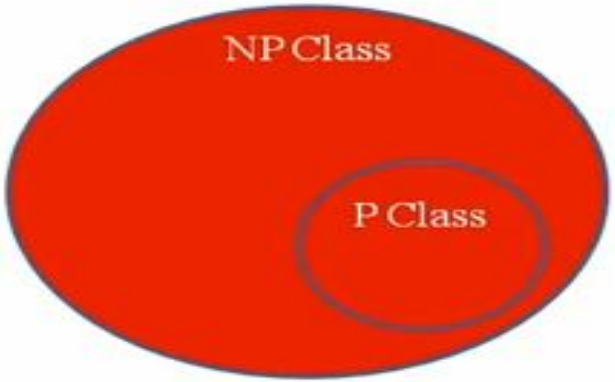
Example: Linear Search (O(n)), Binary Search(O(logn)), etc

- Formally, an algorithm is polynomial time algorithm, if there exists a polynomial p(n) such that the algorithm can solve any instance of size n in a time O(p(n)).

NP Class – Problem:

- NP is set of problems that can solved (non-deterministic) in exponential (Non-deterministic Polynomial [NP]) Time.
- But these kind of problem can be verified in Polynomial Time.

Example: Travelling Salesman Problem



NP – Class:

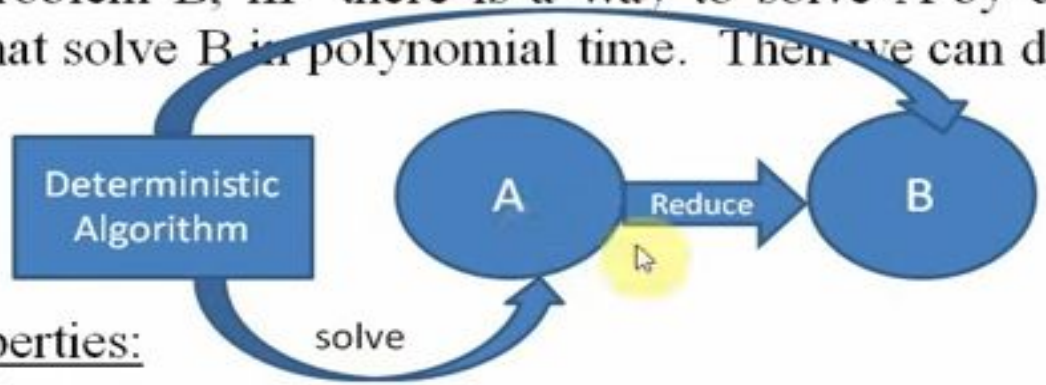
- Solved in Non-Polynomial Time
- Verified in Polynomial Time
- Intractable Problem

P – Class:

- Solved in Polynomial Time
- Verified in Polynomial Time
- Tractable Problem

Reduction:

- Let A and B are two problems in NP. If problem A is reduce to problem B, iff there is a way to solve A by deterministic algorithm that solve B in polynomial time. Then we can denote AαB.

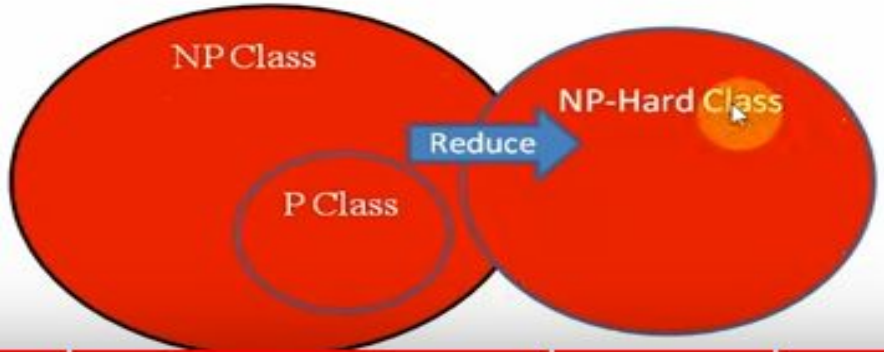


Properties:

- If A is reducible to B and B is in Polynomial time, then A also in Polynomial time.
- A is not in Polynomial time, it implies that B is not in Polynomial time.

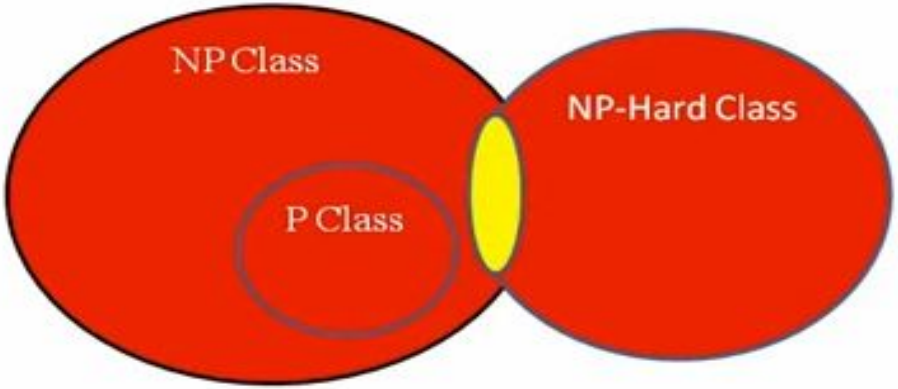
NP-Hard Problem:

- Every problem in NP class can be reduced into other set using polynomial time, then its called as NP-Hard problem.



NP-Complete Problem:

- The group of problems which are both in NP and NP-hard are known as NP-Complete problem.
- All NP-Complete problems are NP-Hard but not all NP-Hard problems are not NP-Complete problem.



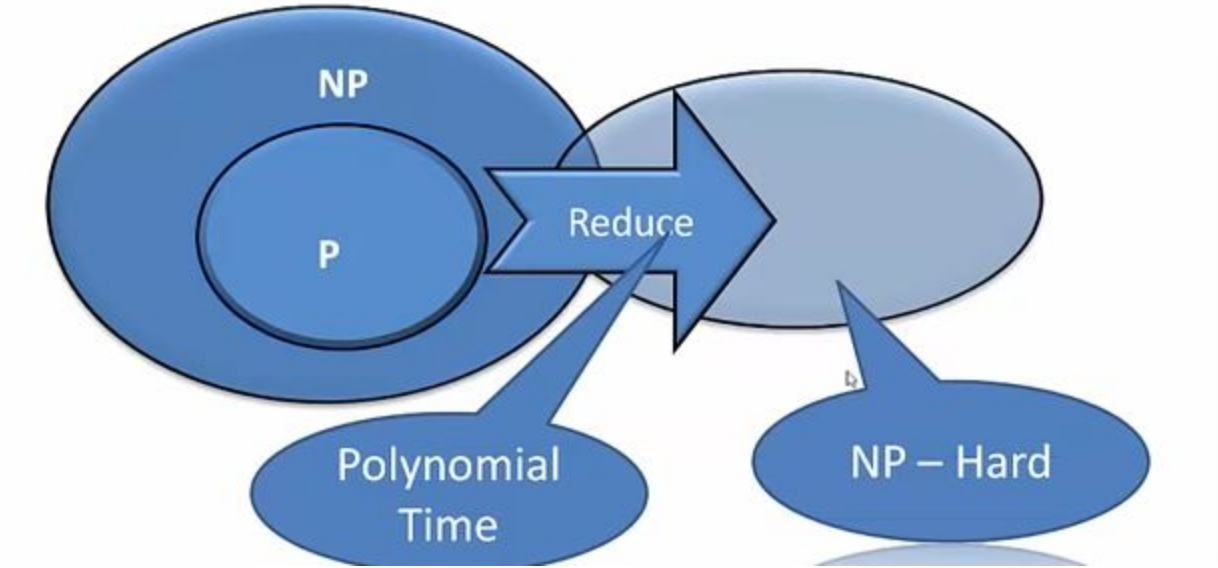
Reduction:

Properties:

1. if A is reducible to B and B in P then A in P.
2. A is not in P implies B is not in P

NP Hard Problem:

A Problem is NP-Hard if every problem in NP can be polynomial reduced to it.

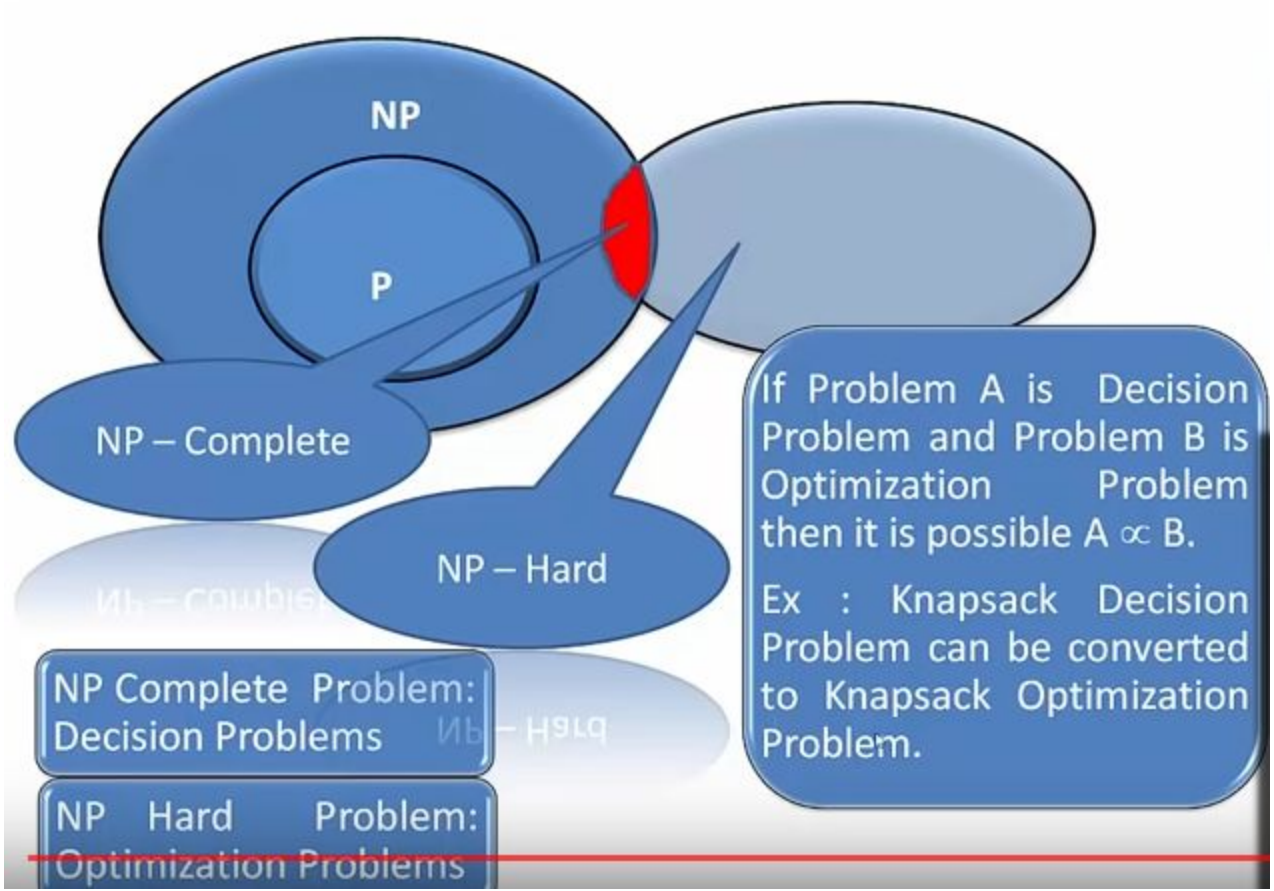


A decision problem is classified as NP-hard if every problem in the class NP (Non-deterministic Polynomial time) can be polynomial-time reduced to it. This means that if you can solve the NP-hard problem efficiently, you can solve any problem in NP efficiently through a polynomial-time reduction.

1. **NP-Hard:**
    - A problem is NP-hard if it is at least as hard as the hardest problems in NP.
    - Informally, it means that if you can solve this NP-hard problem efficiently, you can solve any problem in NP efficiently.
    - The reduction from an NP problem to an NP-hard problem is done in polynomial time.
2. **Polynomial-Time Reduction:**
    - If we can transform instances of problem A into instances of problem B in polynomial time, we say that A is polynomial-time reducible to B, often denoted as A≤pBA≤pB.
    - In the context of NP-hardness, the reduction from an NP problem to an NP-hard problem must be done in polynomial time.

Example:

- If problem X is NP-hard, and you have an NP problem Y, you can create a polynomial-time reduction from Y to X. If you can efficiently solve X, you can use this reduction to efficiently solve Y.


NP
P
NP – Complete
NP – Hard
If Problem A is Decision Problem and Problem B is Optimization Problem then it is possible A ∝ B.
Ex : Knapsack Decision Problem can be converted to Knapsack Optimization Problem.
NP Complete Problem: Decision Problems
NP Hard Problem: Optimization Problems

Think of NP-hard as a category that includes really hard problems (some of which may be in NP, some may not). NP-complete is a special subset of NP-hard problems that are both really hard and in NP.

NP-complete:

- NP-complete problems are a subset of NP problems, and they're special because they're the "hardest" problems in NP.
- If you figure out a fast way to solve any one of these NP-complete problems, you can solve all problems in NP quickly.

Example - Clique Problem:

- Imagine you have a group of people, and you know who is friends with whom. The Clique Problem is like asking, "Can you find a group of at least kk people where everyone is friends with everyone else?"

Explanation:

1. **In NP (easy to check):**
    - If someone says, "Hey, here's a group of people," you can quickly check if they are all friends with each other.
2. **NP-complete (hard to find):**
    - The challenge is finding such a group in the first place. If you can efficiently find this group for any group of people, you can do the same for any other NP problem.

- Think of NP-complete problems as the most difficult puzzles in a set of puzzles. If you can solve the hardest puzzle efficiently, you can solve all puzzles in that set efficiently.

Takeaway:

- NP-complete problems are like a special category of puzzles – solve one efficiently, and you've cracked the code for solving any problem that falls into the NP category. The Clique Problem is just one example; there are many others like it.