MODULE IV: Computable Functions

Turing Machine - Definition, model, Design of TM, computable functions.

Recursive Enumerable Languages and Theorems - Recursively enumerable languages, Church's hypothesis, counter machine, types of Turing Machines (proofs not required)

Module – IV Subjective Question Bank with Notes

S No	Question	Marks	BT Level	CO
1	**Explain** Turing machine model	5		4
	OR			
2	**Summarize** different types of Turing Machines	5		4
3	Describe various Techniques for Turing Machine Construction	5		4
	OR			
4	Define Counter Machine and show equivalence of Counter machines and Turing machine in detail.	5		4
5	**Describe** briefly Recursively Enumerable Languages	3		4
	OR			
6	Design a Turing machine for palindrome strings over $\{a, b\}$.			4
7	Design a TM for accepting strings of the language defined as $\{ ww^r \mid w \in (0 + 1)^*\}$.			4
	OR			
8	i) Design a TM for finding 1's complement of a given binary number. ii) Design a TM to add two numbers a and b.	2 3		4

Turing Machines

- Turing machine is a yardstick for computations that can be carried out on a digital computer.
- Turing machines, first described by Alan Turing in (Turing 1937), are simple, abstract, computational devices intended to help investigate the extent and limitations of what can be computed.
- **what it means to be computable**? Intuitively, a task is computable if one can specify a
- sequence of instructions which when followed will result in the completion of the task.

Turing Assumptions

Turing proposed a set of assumptions that lead to a formal notion of computation that we will call *Turing computability.*

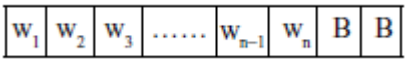
- No creativity implies that each step in the calculation must be fully spelled out.
- The list of instructions followed must be finite; i.e, programmes must be finite objects.
- Each individual step in the calculation must take a finite amount of time to complete.
- Intermediate results may need to be calculated, so a scratch-pad area is needed.
- There has to be a way to keep track of the current step of the calculation.
- There has to be a way to view the complete current state of the calculation.

Turing Machine as a Computational Machine

A turing machine M computes a function f if, when given input w in the domain of f, the machine halts in its accept state with f(w) written (leftmost) on the tape. To use TM as a computational machine, it is required to place the integer numbers as 0m. Suppose it is required to add two numbers; that is, f(m, n) = m + n, then the numbers m and n are to be placed on the tape as 0m10n where 1 is a separator for the numbers m and n. Once processing is completed and the TM halts, the tape would have the contents as 0(m+n), which is the required result of computation.

1. Explain Turing machine model

Alan modeled a TM as a machine with a finite number of control states and an infinite tape, bounded at the left and stretching off to the right. The tape is divided into cells, each of which can hold one symbol. The input of the machine is a string $w = w_1w_2 \ldots w_n$ initially written on the leftmost portion of the tape, followed by an infinite sequence of blanks B.



The machine is able to move a read/write head left and right over the tape as it performs its computation. It can read and write symbols on the tape as it pleases. These considerations led Turing to the following formal definition.

Definition 1: A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where
Q is a finite set of states.
Σ is the input alphabet, which never includes blanks.
Γ is the tape alphabet, which always includes blanks. Moreover, every input symbol is in Γ.
$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function, where L and R are directions, telling the machine head in which direction to go in a step.
$q_0 \in Q$ is the start state.
B is a special symbol indication of a blank cell.
F is a set of final states, where the TM halts on reaching these states.

The program that a TM executes is embodied in the transition function δ. Conceptually, the following happens when a transition $\delta(q_i, a) = (q_j, b, d)$ is made by a TM M:

- M writes b to the current tape cell, overwriting a.
- The current state changes from q_i to q_j.
- The tape head moves to the left or right by one cell, depending on whether d is L or R.

A sequence of linked transitions starting from the initial state is an execution of the given task.

2. Describe various Techniques for Turing Machine Construction

Various techniques for Turing Machine Construction process can be briefly described using the following high-level conceptual tools

a. Storage in Finite Control

A Turing Machine has a finite number of states in its CPU. However, the number of states is not always small. Like a Pentium chip, we can store values in it as long as there is only a finite number of states. For example, all real computers have registers, but there are only a fixed number of them, and each register can only hold one of a fixed (and finite) number of bits. Similarly, we define a state as a pair which stores the details of control and other stores the symbol. To account for this modification, we can define the Turing Machine as $M = (Q, \Sigma, \Gamma, \delta, [q0, B], B, F)$ where Q is of the form [q,a] where q is a state, and $a \in \Sigma$. The transitions are defined by $([Q\chi\Sigma], \Gamma) \rightarrow ([Q\chi\Sigma], \Gamma, \{R/L\})$. For example, the transition $\delta([q, a], b) = ([p, b], c, R)$indicates that the control is in state q, and a is stored in finite control. On the input symbol b, it moves to p state, changes the symbol in finite control to b, changes the cell content as c and moves one cell right.

b. Multi-track Tape

The tape is imagined as divided into cells where the input to be processed is placed. We can further imagine that the tape is divided into k tracks for some finite number k as shown in Figure.

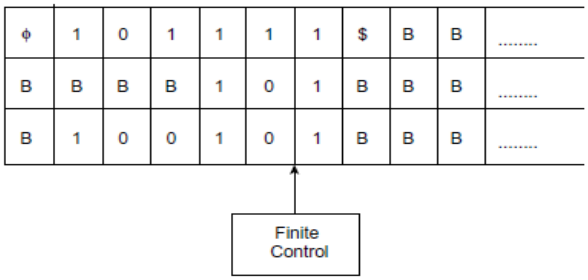


Fig. 6.5 *Multi-track Tape*

The reading head considers k symbols belonging to different tracks in same column and processes it. There are two special symbols Φ and $ used in the first track and they indicate the boundary of the input. The other tracks are used to place the intermediate results and the final result of the processing. The blank input is identified as 'all B's in all tracks', that is, as [B, B, B]. The input at the current position of the reading head is [1, 1, 1].

c. Checking off Symbols

This is one useful technique that can be used to visualize how TM would recognize the languages. This technique uses an extra track that indicates the symbol on the other track is processed. The languages which have repeated strings or some conditions relating to other part of string can be solved with this procedure. Such languages are listed below.

a. $\{ww \mid w \text{ in } \Sigma^*\}$
b. $\{ww^R \mid w \text{ in } \Sigma^*\}$
c. $\{a^ib^i \mid i \geq 1\}$
d. $\{a^ib^jc^k \mid i \neq j \text{ or } j \neq k\}$
e. $\{wcw \mid w \text{ in } \Sigma^*\}$

For the languages mentioned above, we can use the tape with two tracks where on one track, we place the given input, and on the other track, we place either B or √. If the upper track symbol is B, it indicates the symbol on lower track is not considered. If the symbol on upper track is √, it indicates that the symbol on the lower track is considered.

d. Subroutines

A Turing machine can simulate any type of subroutines found in programming languages, including recursive procedures, and any of the known parameter passing mechanisms. We can design a TM program to serve as a subroutine. It has a designated initial state and a return state. These states are used to indicate the call to a subroutine and return to caller subroutine. To design a call to subroutine, a new set of states, are defined which are used to enter the initial state of the subroutine and return from the return state of subroutine. As an example, a TM designed to accept strings with balanced parenthesis.

e. Shifting Over

A turing machine can make space on its tape by shifting non-blank symbols by a finite number of cells to the right. To perform this operation, we can use the state with a small amount of storage. This storage space is used to store the symbols and then replacing the current cell on tape by blank, and to move right. Read the right symbol and replace it with the symbol stored in the finite control. To perform this operation without loosing the date, it requires storage capacity to store at least two symbols. The following example is given to explain the procedure.

3. Summarize the different types of Turing Machines:

There are a number of other types of Turing machines in addition to the one we have seen, such as Turing machines with multiple tapes, one tape but with multiple heads, two-dimensional tapes, non-deterministic TMs, etc. It turns out that computationally all these TMs are equally powerful. That is, what one type can compute any other type can also compute. However, the efficiency of computation, that is, how fast they can compute, may vary.

1. 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 f both the state and the cell content. In all actions, it may move right or left.

For example, let us consider the language L = {ww | w ∈ {a, b}*}. Given a string x, a nondeterministic TM that accepts this language L would first guess the midpoint of x, which is the place where the second half of x starts. Then it would compare the first half of x with the second half by comparing the ith symbol of the first half with the ith symbol of the second half for i = 1, 2, ... A deterministic Turing Machine, on the other hand, cannot guess the midpoint of the string x. It must find the midpoint by, pairing off symbols from the two ends of x. Formally, a non-deterministic TM is a TM whose transition function takes values that are subsets of {(Q ∪ {h}) × (Γ ∪ {Δ}) × (R, L, H)}. As in the case of a NFA, it is understood that a non-deterministic TM, at any configuration, selects one combination of next state, tape symbol and head movement out of the set of triples without following any specific predetermined rule. It can be shown that a non-deterministic TM is also as powerful as a deterministic Turing machine.

2. 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; i.e, the former can be simulated by the latter. To simulate a two-dimensional tape with a one-dimensional tape, first we map the squares of the 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	.<>	...	...	...	...	...
...	...	...	...	...	...	...	...	...

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

The head of a two-dimensional tape moves one square up, down, left or right. Let us simulate this head move with a one-dimensional tape. Let i be the head position of the two-dimensional tape. If the head moves down from i, then move the head of the one dimensional tape to right until it hits h or v counting the number of squares it has visited after i. Let k be the number of squares visited by the head of the one-dimensional tape. If h was hit first, then from h move the head of the one-dimensional tape further right to the k^{th} square from h. That is the square corresponding to the square below i in the two dimensional tape. If v was hit first, then (k + 1)th square to the right from **v** is the new head position. For example, suppose that the head is positioned at position numbered 8 in the table representing the two-dimensional tape; that is, i = 8. In the

two-dimensional table, if the head moves down it reaches the square numbered 13. For equivalence on the one-dimensional tape, the head has to make the following moves: From position 8 move to the right, then it meets h first, and that is the third square from 8. In this circumstance, from h, move 3 positions to the right. By such a move the head position of the one dimensional tape will be at the cell corresponding to 13 on the two-dimensional tape. If i = 5 and the head moves down on the other hand, then on the one-dimensional tape, the head moves to the right and it hits v first, which is the second square from i = 5. Thus this time, the third square is the head position of the one-dimensional tape corresponding to 9 on the two-dimensional tape. Thus some TMs 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 a 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.

3. 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 (Q, Σ, Γ, δ, q0, B, F). Its transition function is a partial function:

$$\delta: QX(\Gamma \cup \{B\})^n \rightarrow (Q \cup \{h\})X(\Gamma \cup \{B\})^nX\{R,L,S\}^n$$

A configuration for this kind of Turing machine 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 Turing Machine can be accepted by a one-tape TM and that any function computed by a 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.

4. Turing Machines with Multiple Heads

This kind of Turing Machines 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 5-tuple (Q, Σ, Γ, δ, q0, B, F). The transition function is a partial function:

$$\delta: QX\{H_1 , H_2 , ...H_n \}X(\Gamma \cup \{B\}) \rightarrow (Q \cup \{h\})X(\{B\})X\{R,L,S\}$$

where H1, H2 ..., Hn denote the tape heads. It can be easily seen that this type of Turing machines are as powerful as one-tape TMs.

5. 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 Turing machines are also as powerful as onetape Turing Machines whose tape has a left end.

Two-way infinite tapes

In all our formulations, we specified that the tape had a left end and stretched infinitely far to the right. Relaxing this stipulation to allow the tape to stretch infinitely far to right and left results in a new formulation of Turing Machines equivalent to the original. That is, for any Turing Machine using a two-way tape, there is a Turing Machine with a one-way infinite tape with the same input–output behaviour, and vice versa.

One can simulate a Turing Machine with two-way infinite tape on a Turing Machine with one-way infinite tape. Let the two-way infinite tape have the contents as shown below:

Let the reading head be left of A0. This can be simulated by a Turing machine with one way infinite tape with two tracks. The contents on tape is placed in such a way that, content on right of A0 is placed on upper track and the content on left of A0 is placed on lower track in reverse order. The left end cell contains on upper track, and on the lower track, the left end cell contains a special symbol Φ:

The moves are simulated such that when the reading head is to the right of A0, the moves are implemented as if they are reading symbols from upper track. If the reading head is to the left of A0, then symbols are read from lower track head, but the direction of the reading head would be in opposite direction in which it moves on the two-way infinite tape. That is, if it moves left then on one way infinite tape, it moves right on the two-way infinite tape, and vice versa.

4. Define Counter Machine and show equivalence of Counter machines and Turing machine in detail

A counter machine is an abstract machine used in formal logic and theoretical computer science to model computation. A counter machine comprises a set of one or more unbounded **registers**, each of which can hold a single non-negative integer, and a list of arithmetic and control instructions for the machine to follow.

***Definition*:** A counter machine consists of

1. A finite set of registers r_0 ... r_n, where each register is labelled and can hold any single non-negative integer.
2. A special register to identify the current instruction to be executed is maintained and is called the **state register**.
3. A finite set of instructions I0 ... Im, where each instruction is in the same physical space as the registers.

Equivalence of Counter machines and Turing machine

Every turing machine can be simulated by a two counter machine. The turing machine consists of a Finite State machine and infinite tape that can be divided and used as a stack.

The proof is explained in the following three steps.

1. Simulate turing machine by a finite-state machine (FSM) equipped with two stacks.
2. Simulate two stacks by four counters.
3. Simulate four counters by two counters.

Step 1: Simulating turing machine by two stacks

The tape is initially filled with zeroes, which can be modified with ones and zeroes. At any time, the read/write head of the machine points to one cell on the tape. This tape is conceptually cut into half and each half is treated as a stack. The top is the cell nearest the read/write head, and the bottom is some distance away from the head, with all zeroes on the tape beyond the bottom. Accordingly, a turing machine can be simulated by a FSM plus two stacks. Moving the head left or right is equivalent to popping a bit from one stack and pushing it onto the other. Writing is equivalent to changing the bit before pushing it.

Step 2: Simulate two stacks by four counters

A stack containing zeroes and ones can be simulated by two counters. The bits on the stack are assumed to be represented in binary, with the top being the least significant bit.

0 is pushed onto the stack to indicate the doubling of the number 1, to indicate doubling and adding 1. Popping the top is equivalent to dividing by 2, where the remainder is the bit that was popped. Two counters can simulate this stack: One counter holds a number whose binary representation represents the bits on the stack, and the other counter is used as a scratch pad. To double the number in the first counter, the FSM can initialize the second counter to zero, then repeatedly decrement the first counter once and increment the second counter twice. This continues until the first counter reaches zero. At that point, the second counter will hold the doubled number. To divide by 2, repeat decrementing in Counter 1 by twice and increment the other by one until the first counter reaches zero. The remainder can be determined by whether it reached zero after an even or an odd number of tries.

Step 3: Simulate four counters by two counters

In Step 2, one counter is used as scratch pad, the other is a real counter that holds an integer whose prime factorization is $2^a3^b5^c7^d$. The exponents a, b, c, and d can be thought of as four virtual counters that are being simulated.

- To set all virtual counters to zero, the real counter is set to zero and then incremented once.
- To double or make it half, increment or decrement a.
- To multiply or divide by 3, increment or decrement b.

As a result, a FSM with two counters can simulate four counters, which are in turn simulating two stacks, which are simulating a TM. Therefore, a FSM plus two counters is at least as powerful as a TM. A TM can easily simulate a FSM with two counters. Therefore, the two machines have equivalent power.

5. Describe briefly Recursively Enumerable Languages

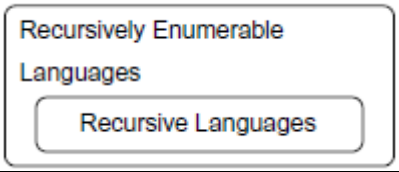
There are *three* possible outcomes of executing a TM over a given input. The TM may

- halt and accept the input;
- halt and reject the input; or
- never halt.

A language is said to be recursive if there exists a TM that accepts every string of the language and rejects every string (over the same alphabet) that is not in the language.
Note: If a language L is recursive, then its complement L must also be recursive.

A language is said to be recursively enumerable if there exists a TM that accepts every string of the language, and does not accept strings that are not in the language. For 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:



6. Define and describe Church's Thesis in brief

The notion that a computable function can be identified with the class of partial recursive functions is known as Church's hypothesis or Church–Turing thesis. The TM is equivalent to the digital computer, in computing power.

The Church–Turing thesis (formerly commonly known simply as Church's thesis) says that any real-world computation can be translated into an equivalent computation involving a TM. In Church's original formulation (Church 1935, 1936), the thesis says that real-world calculation can be done using the lambda calculus, which is equivalent to using general recursive functions.

The Church–Turing thesis can be applied to any kind of computations, those involving cellular automata, combinators, register machines and substitution systems. It is also applicable to other kinds of computations found in theoretical computer science, such as quantum computing and probabilistic computing.

There are conflicting points of view about the Church–Turing thesis. One says that it can be proven, and the other says that it serves as a definition for computation. Church– Turing thesis has not been proved, but the support for its validity comes from the fact that every realistic model of computation, so far discovered, has been shown to be equivalent. As long as our method of computation places no bound on the number of steps or on the amount of storage, it seems that the partial recursive functions are intuitively computable. Random Access Machine (RAM) is an abstract model that makes use of partial recursive functions.

RAM consists of an infinite number of memory words, numbered 0, 1, …., where each word can hold any integer; and a finite number of arithmetic registers capable of holding any integer. These integers can be decoded into the form of a computer instruction. To simulate RAM by a TM, we use multi-tapes. One of the tapes holds the words of the RAM, where each word is separated by #, as shown below.

$$\#0*V_0\ \#1*V_1\#2*V_2\#3*V_3\ldots\#i*V_i\ldots$$

V_i is the contents, in binary, of the i^{th} word. At any given time, there will be a finite number of words of the RAM stored on the tape; so, it is required to keep record of the largest numbered word that had been used so far.

RAM has finite number of arithmetic registers. One tape is used to hold each register's contents, and one tape is used to hold the location counter that indicates the number of the next word to be taken. One tape is used as a memory address register on which the number of a memory word may be placed.

Suppose that each word has the first 10 bits indicating the operation to be performed such as LOAD, STORE, ADD and so on, and let the remaining bits denote the address of the operand. Suppose the location counter holds number i in binary. The processing is done as below.

1. The reading head searches the first tape for the pattern #i*.
2. If not found, there is no instruction in the word i in the RAM. So, it halts.
3. If the pattern #i* is found, then the bits followed by * till next # are examined. Suppose the first 10 bits correspond ADD to Register 2, and suppose the remaining bits are some number j in binary.
4. M adds 1 to i on location counter tape.
5. Copies j into memory address tape.
6. M searches for #j* on the first tape, again starting from the left.
7. If #j* is not found, we assume j holds 0 and go to next instruction.
8. If #j* is found, V_j is added to the contents of Register 2, which is stored on its own tape.
9. Repeat Steps 1–8 for every instruction.

7. Design a Turing machine for palindrome strings over {*a*, *b*}.

In this case, size of strings could be even or odd.

For example, abba. To accept such strings, first find the symbol encountered and travel extreme right; and if the same symbol is found, then replace it by blank. Repeat until no more symbol is found.

1. In initial state, if b is encountered, change to q_1 state:.

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

2. In q1 state, travel extreme right until B is seen. On B, change to q2 state and move left:

$$\delta(q_1, b) = (q_1, b, R)$$
$$\delta(q_1, a) = (q_1, a, R)$$
$$\delta(q_1, B) = (q_2, B, L)$$

3. In q_2 state, if the input is 1, replace it by B and change to q5 state:

$$\delta(q_2, b) = (q_5, B, L)$$
$$\delta(q_2, B) = (q_A, B, R)$$

4. In q_5, move left until to extreem left until it finds 'B':

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

5. In initial state, if input is 0, go q_3 state:

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

6. In q3 state, travel extreme right until B is seen. On B, change to q4 state and move left:

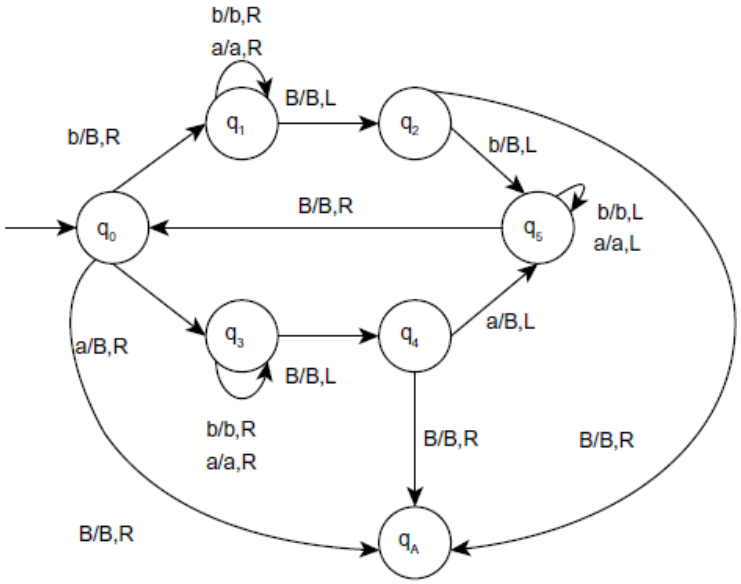
$\delta(q_3, b) = (q_3, b, R)$
$\delta(q_3, a) = (q_3, a, R)$
$\delta(q_3, B) = (q_4, B, L)$

7. In q_4 state, if the input is 0, replace it by B and change to q5 state:

$\delta(q_4, 0) = (q_5, B, L)$
$\delta(q_4, B) = (q_A, B, R)$

8. Repeat Steps 1–3, or Steps 5–7 and Step 4 till it encounters B:

$\delta(q_5, B) = (q_0, B, R)$
$\delta(q_0, B) = (q_A, B, R)$



	a	b	B
$\rightarrow q_0$	(q_1, B, R)	(q_3, B, R)	(q_A, B, R)
q_1	(q_1, a, R)	(q_1, b, R)	(q_2, B, L)
q_2	---	(q_5, B, L)	(q_A, B, R)
q_3	(q_3, a, R)	(q_3, b, R)	(q_4, B, L)
q_4	(q_5, B, L)	---	(q_A, B, R)
q_5	(q_5, a, L)	(q_5, b, L)	(q_0, B, R)
q_A	---	---	---

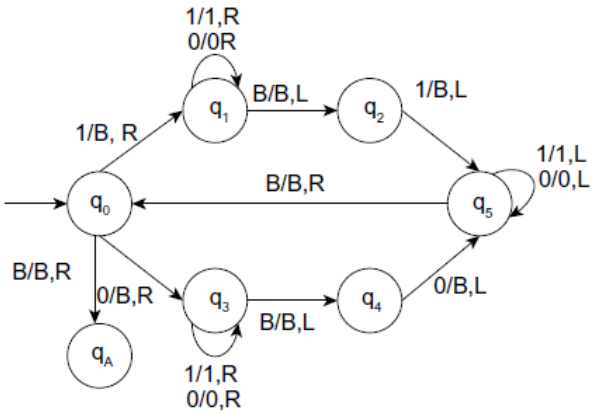
8. Design a TM for accepting strings of the language defined as { ww^r | $w \in (0 + 1)^*$}.

The given language contains strings which are even palindromes.
For example, abba. To accept such strings, first find the symbol encountered and travel extreme right; and if the same symbol is found, then replace it by blank. Repeat until no more symbol is found.

1. In initial state, if 1 is encountered, change to q_1 state:.

$\delta(q_0, 1) = (q_1, B, R)$

2. In q_1 state, travel extreme right until B is seen. On B, change to q_2 state and move left:
   $\delta(q_1, 1) = (q_1, 1, R)$
   $\delta(q_1, 0) = (q_1, 0, R)$
   $\delta(q_1, B) = (q_2, B, L)$
3. In q_2 state, if the input is 1, replace it by B and change to q_5 state:
   $\delta(q_2, 1) = (q_5, B, L)$
4. In q5, move left until to extreem left until it finds 'B':
   $\delta(q_5, 1) = (q_5, 1, L)$
   $\delta(q_5, 0) = (q_5, 0, L)$
5. In initial state, if input is 0, go q_3 state:
   $\delta(q_0, 0) = (q_3, B, R)$
6. In q3 state, travel extreme right until B is seen. On B, change to q_4 state and move left:
   $\delta(q_3, 1) = (q_3, 1, R)$
   $\delta(q_3, 0) = (q_3, 0, R)$
   $\delta(q_3, B) = (q_4, B, L)$
7. In q4 state, if the input is 0, replace it by B and change to q_5 state:
   $\delta(q_4, 0) = (q_5, B, L)$
8. Repeat Steps 1–3, or Steps 5–7 and Step 4 till it encounters B:
   $\delta(q_5, B) = (q_0, B, R)$
   $\delta(q_0, B) = (q_A, B, R)$



	0	1	B
$\rightarrow q_0$	(q_1, B, R)	(q_3, B, R)	(q_A, B, R)
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_2, B, L)
q_2	---	(q_5, B, L)	---
q_3	$(q_3, 0, R)$	$(q_3, 1, R)$	(q_4, B, L)
q_4	(q_5, B, L)	---	---
q_5	$(q_5, 0, L)$	$(q_5, 1, L)$	(q_0, B, R)
q_A	---	---	---

9. Design a TM for finding 1's complement of a given binary number.

If the given number is 100110, then its 1's complement is 011001.
If the given number is 11110, then its 1's complement is 00001.
From the above examples, it is clear that the TM has to be designed such that while moving to right, if the input is 1, it should be changed to 0 and if the input is 0 it should be changed to 1 and halt on B. The transition diagram is shown in the Figure.

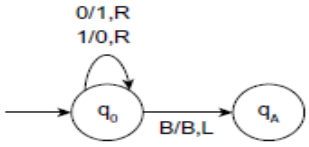


Fig. 6.2 *TM for 1's Complement*

	a = 0	a = 1	a = B
$\rightarrow q_0$	$(q_0, 1, R)$	$(q_0, 0, R)$	(q_A, B, L)
q_A	----	----	----

10. Design a TM to add two numbers a and b.

Let the numbers be 2 and 3. The addition of these numbers using simple logic is explained. The numbers are placed as $B0^210^3B$. After processing, the tape content would be $B0^5B$. The simple logic that can be used is: replace the occurrence of 0 by B and move to right and replace 1 to 0, so that it is in required form as B05B. Sequence of steps is given for understanding and Figure shows the details.

1. In initial state, 0's is replaced by B and the state is changed to a new state:
   $\delta(q_0, 0) = (q_1, B, R)$

2. In q1 state, move to right on '0'. When the input is '1' replace it by '0'.
   $\delta(q_1, 0) = (q_1, 0, R)$
   $\delta(q_1, 1) = (q_1, 0, R)$
   $\delta(q_1, B) = (q_A, B, R)$

$B\bullet001000B \vdash B\bullet01000B \vdash B0\bullet1000B \vdash B00\bullet000B \vdash B000\bullet00B \vdash B0000\bullet0B \vdash B00000\bullet B \vdash B00000B$

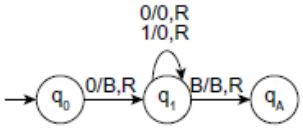


Fig. 6.3 *TM to Add Two Numbers*

	0	1	B
$\rightarrow q_0$	(q_1, B, R)	---	---
q_1	$(q_1, 0, R)$	$(q_1, 0, R)$	(q_A, B, R)
q_A	---	---	---

11. Compare NFA and PDA

NFA	PDA
The language accepted by NFA is the regular language	The language accepted by PDA is a context free language
NFA has no memory	PDA is essentially a NFA with a stack (memory)
It can store only a limited amount of information	Amount of information it can store is unlimited
A language/string is accepted only by reaching the final state	It accepts a language either by an empty stack or by reaching a final state

12. Compare NPDA and DPDA.

DPDA	NPDA
The PDA that has at most one choice of move in any state is called a deterministic PDA	NPDA provides non-determinism to PDA

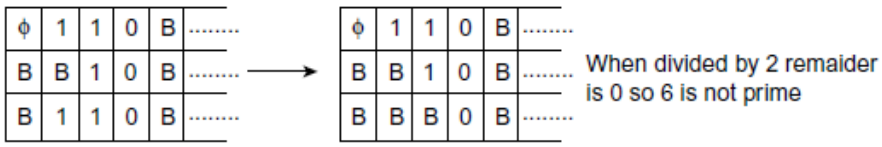
Deterministic PDA's (DPDAs) are very useful in programming languages. For example, parsers are deterministic.	DCFLs are recognized by NPDA. Syntax of most of the programming languages is described by DCFLs.
Strings such as $\{wcw^R \mid w \text{ is in } (a + b)^*\}$ can be recognized by DPDA	Strings such as $\{ww^R \mid w \text{ is in } (a + b)^*\}$ cannot be recognized by DPDAs.

13. Design a TM to find whether the given number is prime or not.

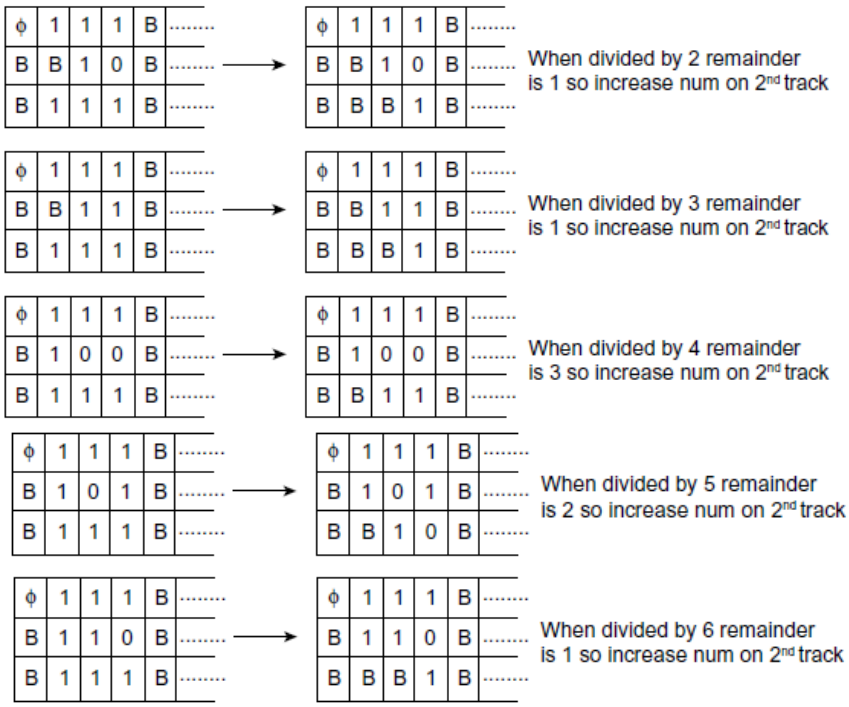
Solution: To design a TM to identify a given number is prime or not, we have to find whether the number has a factor other than 1 and itself.

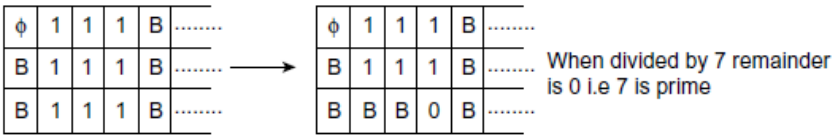
1. Let us place the given number on the first track in binary form bounded by Φ and $.
   For example, 47 is represented as Φ101111$.
2. On the second track, write 2 in binary form, 10.v
3. Copy the number on the first track to the third track.
4. Perform repeated subtraction of the number on third track with the number on the second track until the number on the third track is either 0 or less than the number on the second track.
5. If the number on the third track is zero and number on the second is not equal to the number on the first track, then the number on the first track is not prime, otherwise prime.
6. If the number on the third track in non-zero then increase the number on the second track by one.
7. Repeat the Steps 4–6 until the number on the second track is equal to number on the first.

Find whether 6 is a prime or not



Find whether 7 is a prime or not



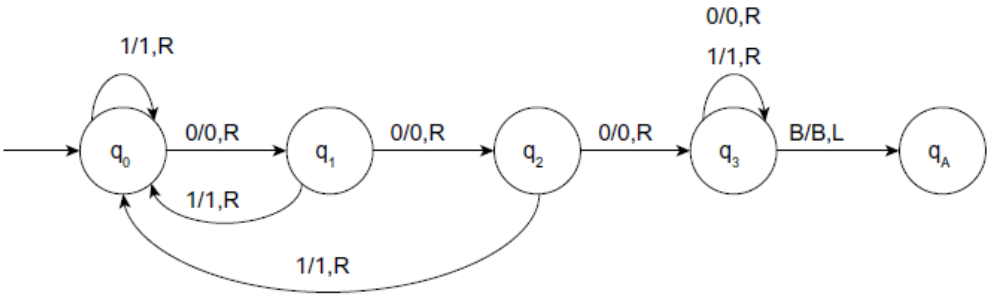


Example Using Multitrack Tape

14. Design a TM to accept strings formed with 0 and 1 that have the substring 000.

Solution: To accept strings with substring 000, the TM would be similar to the finite automaton constructed for the same language.

1. In state q_0, defi ne transition to ignore the 1's encountered. If 0 is seen, then change to new state which indicates one 0 is present:
   $\delta(q_0, 1) = (q_0, 1, R)$
   $\delta(q_0, 0) = (q_1, 0, R)$
2. In q_1 state, if input is 0, change to new state to indicate the substring 00; if input is 1, go back to initial state:
   $\delta(q_1, 1) = (q_0, 1, R)$
   $\delta(q_1, 0) = (q_2, 0, R)$
3. In q^2 state, if input is 0, change to new state to indicate substring 000; if input is 1, go back to initial state:
   $\delta(q_2, 1) = (q_0, 1, R)$
   $\delta(q_2, 0) = (q_3, 0, R)$
4. In q_3 state, if input is 0 or 1, be in same state, and on B move to final state:
   $\delta(q_3, 1) = (q_3, 1, R)$
   $\delta(q_3, 0) = (q_3, 0, R)$
   $\delta(q_3, B) = (q_A, B, R)$



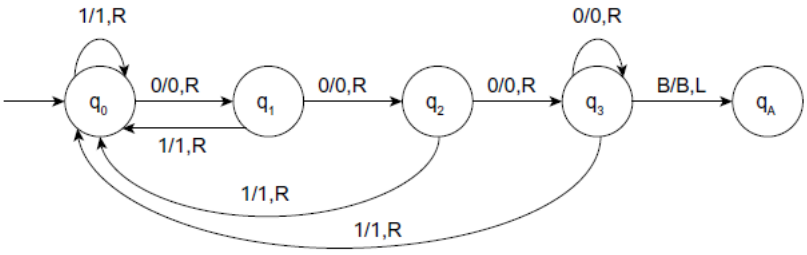
	0	1	B
$\rightarrow q_0$	$(q_1, 0, R)$	$(q_0, 1, R)$	---
q_1	$(q_2, 0, R)$	$(q_0, 1, R)$	---
q_2	$(q_3, 0, R)$	$(q_0, 1, R)$	---
q_3	$(q_3, 0, R)$	$(q_3, 1, R)$	(q_A, B, R)
q_A	---	---	---

15. Design a TM to accept strings formed on {0, 1} that end with 000.

Solution: To accept strings ending with 000, the TM would be similar to the one in the example above, but the only difference is, in q3 state, define moves such that on 0, it is in same state, and on 1, it changes to q0 state.

1. In state q_0, define transition to ignore the 1's encountered. If 0 is seen, then change to new state, which indicates one 0 is present:
   $\delta(q_0, 1) = (q_0, 1, R)$
   $\delta(q_0, 0) = (q_1, 0, R)$

2. In q_1 state, if input is 0, change to new state to indicate substring 00; if input is 1, go back to initial state:
   $\delta(q_1, 1) = (q_0, 1, R)$
   $\delta(q_1, 0) = (q_2, 0, R)$
3. In q_2 state, if input is 0, change to new state to indicate substring 000; if input is 1, go back to initial state:
   $\delta(q_2, 1) = (q_0, 1, R)$
   $\delta(q_2, 0) = (q_3, 0, R)$
4. In q_3 state, if input is 0 or 1 be in same state, and on B move to final state:
   $\delta(q_3, 1) = (q_0, 1, R)$
   $\delta(q_3, 0) = (q_3, 0, R)$
   $\delta(q_3, B) = (q_A, B, R)$



	0	1	B
$\rightarrow q_0$	$(q_1, 0, R)$	$(q_0, 1, R)$	---
q_1	$(q_2, 0, R)$	$(q_0, 1, R)$	---
q_2	$(q_3, 0, R)$	$(q_0, 1, R)$	---
q_3	$(q_3, 0, R)$	$(q_0, 1, R)$	(q_A, B, R)
q_A	---	---	---

16. Design a Turing Machine to accept strings of the language defined by $\{a^n b^n \mid n \geq 1\}$.

Solution: Here if the strings are formed with a's followed by b's and if the number of both a's and b's are equal, then they are accepted. The processing can be illustrated with the example as below.

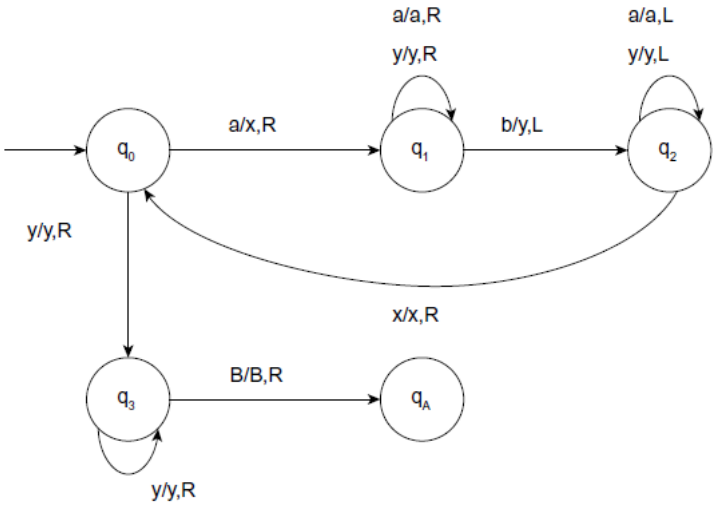
1. In initial state, if a is encountered, change to q1 state:
   $\delta(q_0, a) = (q_1, x, R)$
2. In q_1 state, travel extreme right until b is seen. On B, change to q_2 state, change the symbol to y, and move left:
   $\delta(q_1, a) = (q_1, a, R)$
   $\delta(q_1, b) = (q_2, y, L)$
3. In q_2, move left until it encounters x:
   $\delta(q_2, a) = (q_2, a, L)$
   $\delta(q_2, x) = (q_0, x, R)$
4. Repeat until in q_0 state it encounters y and then change to state q_3:
   $\delta(q_0, y) = (q_3, y, R)$
5. In q3, move right until it encounters B:
   $\delta(q_3, y) = (q_3, y, R)$

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

6. To support the repetition, some transitions have to be add at q1 and q2:
   $\delta(q_1, y) = (q_1, y, R)$
   $\delta(q_2, y) = (q_2, y, L)$

B•aabbB ⊢ Bx•abbB ⊢ Bxa•bbB ⊢ Bx•aybB ⊢ B•xaybB ⊢ Bx•aybB ⊢ Bxx•ybB ⊢ Bxxy•bB ⊢ Bxx•yyB ⊢ Bx•xyyB ⊢ Bxxy•yB ⊢ Bxxyy•B

If the TM halts replacing all a's by x and all b's by y, then the string is accepted.
To design this, we need five states as depicted in the following figure:



	a	b	X	Y	B
$\rightarrow q_0$	(q_1, x, R)	---	---	(q_3, y, R)	---
q_1	(q_1, a, R)	(q_2, y, L)	---	(q_1, y, R)	---
q_2	(q_2, a, L)	---	(q_0, x, R)	(q_2, y, L)	---
q_3	---	---	---	(q_3, y, R)	(q_A, B, R)
q_A	---	---	---	---	---

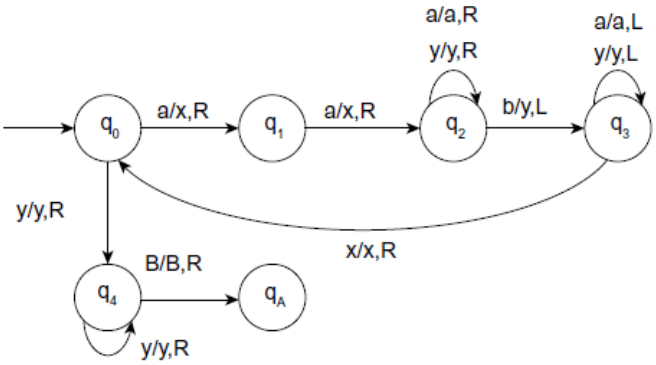
17. Design a TM to accept strings of the language defined as $\{a^{2n}b^n \mid n \geq 0\}$.

Solution: Here if the strings are formed with a's followed by b's and if the number of a's are twice the number of b's, then they are accepted. The processing can be illustrated with an example below.

1. In initial state, if a is encountered, change to q_1 state:
   $\delta(q_0, a) = (q_1, x, R)$
2. In q_1 state, if a is encountered, change to q2 state:
   $\delta(q_1, a) = (q_2, x, R)$
3. In q_2 state, travel extreme right until b is seen. On B, change to q_3 state, change the symbol to y, and move left:
   $\delta(q_2, a) = (q_2, a, R)$
   $\delta(q_2, b) = (q_3, y, L)$
4. In q_3, move extreme left until it encounters x:
   $\delta(q_3, a) = (q_3, a, L)$
   $\delta(q_3, x) = (q_0, x, R)$
5. In q_0 state, repeat until it encounters y and change to state q_4:

$\delta(q_0, y) = (q_4, y, R)$

6. In q4 move extreme right until it encounters B:
   $\delta(q_4, y) = (q_4, y, R)$
   $\delta(q_4, B) = (q_A, B, R)$
7. To support the repetition, some transitions have to be add to q_2 and q_3 states:
   $\delta(q_2, y) = (q_2, y, R)$
   $\delta(q_3, y) = (q_3, y, L)$



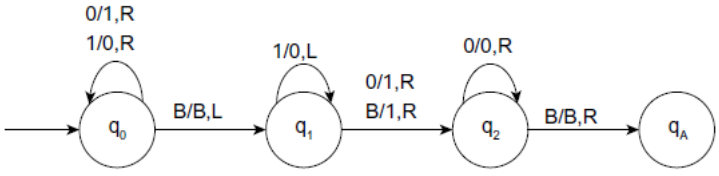
	a	b	X	Y	B
$\rightarrow q_0$	(q_1, x, R)	---	---	(q_4, y, R)	---
q_1	(q_2, x, R)	---	---	---	---
q_2	(q_2, a, R)	(q_3, y, L)	---	(q_2, y, R)	---
q_3	(q_3, a, L)	---	(q_0, x, R)	(q_3, y, L)	---
q_4	---	---	---	(q_4, y, R)	(q_A, B, R)
q_A	---	---	---	---	---

18. Draw a TM that gives two's complement for the given binary representation.

Solution: Two's complement is computed by first computing one's complement and then adding 1 to it. For example,

Binary number	One's complement	Two's complement
1001	0110	0111
0011	1100	1101
1000	0111	1000

By seeing these few examples, it is clear that the machine has to be designed such that it first complements every bit moving left to right, travels back complementing each bit until it sees first occurrence of 0, and then goes to the right end and halts.



	a = 0	a = 1	a = B
q_0	$(q_0, 1, R)$	$(q_0, 0, R)$	(q_1, B, L)
q_1	$(q_2, 1, R)$	$(q_1, 0, L)$	$(q_2, 1, R)$
q_2	$(q_A, 0, R)$	---	(q_A, B, R)
q_A	----	----	----