UNIT-V
TURING MACHINE

Turing machine :- Turing machine (TM) is a modified version of the PDA and it is much more powerful than PDA. Instead of using stack as in PDA, the TM uses the tape to store the symbols.

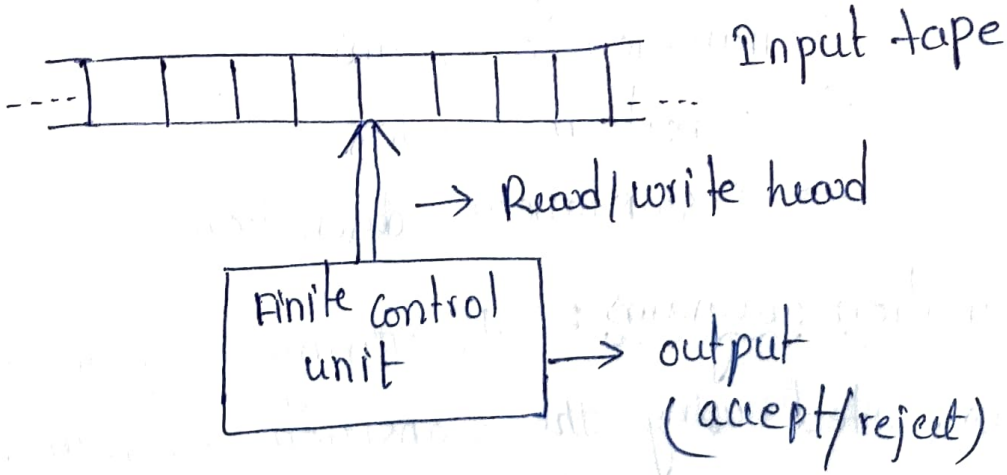
→ The Turing machine is a generalized machine which can recognize all types of languages i.e Regular languages, CFL'S, CSL'S and recursively enumerable lang.

→ Thus turing machine can accept any language.

Model of Turing machine :-

It is a finite automata connected to read write head with the following components:

→ Input tape
→ Read/write head
→ Control unit



Input tape :- It 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 from leftmost position on the tape. 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 and 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 read-write head can move either towards left/right i.e movement can be on both the directions.

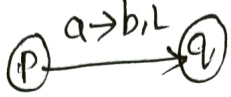
Representation of Turing machines :- The Turing machines can be represented using various notations such as,

→ Transition diagrams
→ Transition tables
→ Instantaneous descriptions.

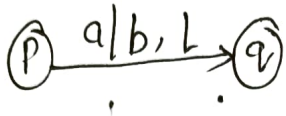
1) Transition diagrams :- The Turing machine can be represented using the transition diagram. In this the arc (edge between the two states)

going from one vertex to another vertex. and the edge label can be represented ~~can~~ in different forms, they are

Form 1 :- Read symbol(a) → write symbol(b) → move left(L)/ Right(R).



Form 2 :-



2) Five tuple specification :- the action performed by the TM, from one state to another state, can be specified by using the 5-tuple.

< state 1, Read symbol, write symbol, L/R/N, state-2 >

Ex:- <q1, b, K, R, q2>

3) Transition table :- the transition table describes the following for the given state and the symbol it currently reads :

→ write a symbol
→ move the head (L/R/No move)
→ assume the same (or) a new state.

The transition table can be represented in different forms.

Form 1 :-

current state	Read symbol	write symbol	move tape	final state	5tuple specifi

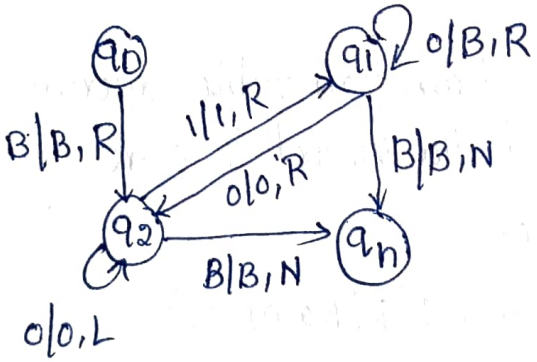
Form2 :-

current state	q_1			----	q_n		
Tape symbol	write symbol	move tape	next State		write symbol	move tape	next state

Form3 :-

Tape symbol / states	q_1	q_2	q_3	----	q_n
	<action>				

Ex:- Consider a TM, whose task is described in the transition diagram



The transition table is.

I/p symbol / states	0	1	B
q_0	-	-	$\langle q_2, B, R\rangle$
q_1	$\langle q_2, 0, R\rangle$	$\langle q_1, B, R\rangle$	$\langle h, B, N\rangle$
q_2	$\langle q_2, 0, L\rangle$	$\langle q_1, 1, R\rangle$	$\langle h, B, N\rangle$

→ **Elements of TM** :- A TM has the following seven characteristics: Formally a turing machine M is represented as a 7-tuple

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

where,

1) Q is the finite set of states.
2) Σ is the finite set of non blank symbols.
3) Γ is the set of tape characters.
4) $q_0 \in Q$ is the initial state.
5) B is the blank character.
6) $h \subseteq Q$ is the final state.
7) δ is the transition function of a TM and is defined as $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R, N\}$.

Transitions of a TM :- The transition of a turing machine is represented as,

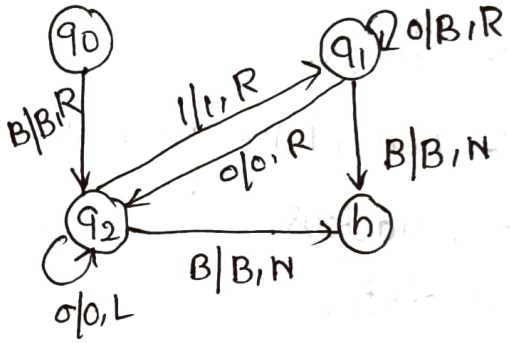
$$\delta(q_i, a_K) = (q_j, a_l, x)$$

for $q_i \in Q$, $(a_K, a_l) \in \Gamma$ and x is any one of the values 'L', 'R' and 'N'.

The meaning of $\delta(q_i, a_K) = (q_j, a_l, K)$ is that, if q_j is the current state of the TM and a_K is the cell currently under the head, then TM writes a_l in the cell currently under the head, enters the state q_j and the head moves to the adjacent cell to the right, if the

value of x is R. otherwise, the head moves to the adjacent cell to the left, if the value of x is L and continues scanning the same cell, if the value of x is N

Ex :- For the TM in figure, the transition functions are



→ $\delta(q_0, B) = (q_2, B, R)$

q_0 is the current state
B is the cell currently under the head

→ $\delta(q_1, 0) = (q_2, 0, R)$

→ $\delta(q_1, 1) = (q_1, B, R)$

→ $\delta(q_1, B) = (h, B, N)$

→ $\delta(q_2, 0) = (q_2, 0, L)$

→ $\delta(q_2, 1) = (q_1, 1, R)$

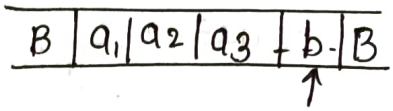
→ $\delta(q_2, B) = (h, B, N)$.

Instantaneous description of a TM :-

The complete state of a TM, at any point during a computation, may be described by.

→ name of the state that in which the machine is
→ the symbols on the tape
→ the cell currently being scanned.

A description of these three data is called instantaneous description (ID) (or) configuration of TM. The simple way to represent such a description



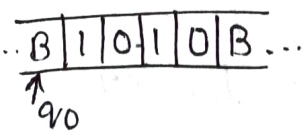
The initial ID is denoted by q_0xy, where q_0 is the start state and the head points to the first symbol from left x. The final ID is denoted by xyq_hB, where q_h is the final state and the head points to the blank character denoted by B.

Ex:- Consider a TM, whose action is described in the transition table

	0	1	B
q_1	$(q_2, 0, R)$	(q_1, B, R)	$(q_2, B, R)^h$ $(q_h, B, R)^N$
q_2	$(q_2, 0, L)$	$(q_1, 1, R)$	(h, B, N)
q_0	-	-	(q_2, B, R)

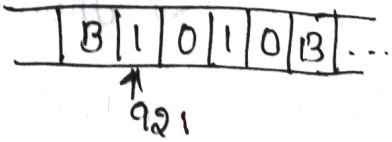
The action of the TM with the string $w = 1010$ is

Time 0:



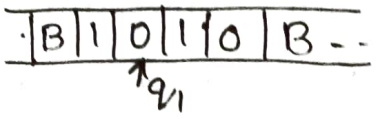
ID: $q_0B1010B$

Time 1:



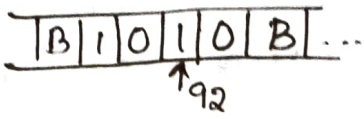
ID: Bq_21010B

Time 2 :



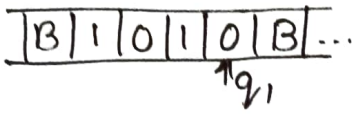
ID : B1$q_1$010B

Time 3 :



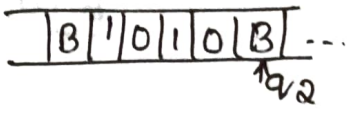
ID : B10$q_2$10B.

Time 4 :



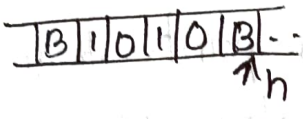
ID : B101$q_1$0B

Time 5 :



ID : B1010q_2B

Time 6 :



ID : B1010Bh

Moves of a TM :- there are three possible types of moves,

→ move to the left
→ " " right
→ No move

we give the formal definition to the moves of TM

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, h)$ be a TM, let the ID of M be

$$(q, a_1, a_2 \ldots a_{i-1}, a_i, a_{i+1}, \ldots a_n)$$

Consider the following transitions :

→ $\delta(q, a_i) = \delta(p, b, L)$, for moving to the left.

Case1 :- if $i > 1$, then the move of the TM is going from ID $(q, a_1 a_2 \ldots a_{i-1}, a_i, a_{i+1} \ldots a_n)$ to $(p, a_1, a_2 \ldots a_{i-2}, a_{i-1}, a_i, a_{i+1} \ldots a_n)$ is denoted by,

$$(q, a_1, a_2 \ldots a_{i-1}, \underline{a_i}, a_{i+1} \ldots a_n) \vdash (p, a_1, a_2 \ldots a_{i-2}, \underline{a_{i-1}}, \underline{b}, a_{i+1} \ldots a_n)$$

Case II:- if i=1, the TM crashes, as it is already scanning the left most symbol at cell i and attempts to move to the left, which is not possible. hence move is not defined.

case III:- if i=n and B is the blank symbol, then

$(q, a_1 a_2 \ldots a_{n-1}, a_n, e) \vdash (q, a_1 a_2 \ldots a_{n-2}, a_{n-1}, B, e)$

→ $\delta(q, a_i) = \delta(P, b, R)$. moving to the right

case i :- if i<n, then move of TM is

$(q, a_1, \ldots a_{i-1}, a_i, a_{i+1}, \ldots a_n) \vdash (P, a_1, \ldots a_{i-1}, b, a_{i+1}, \ldots a_n)$

case ii :- if i=n then

$(q, a_1, \ldots a_{n-1}, a_n, e) \vdash (P, a_1 \ldots B, e)$

Example :- Consider a TM, whose action is duscribed in the transition table.

	0	1	B
q_0	-	-	(q_2, B, R)
q_1	$(q_2, 0, R)$	(q_1, B, R)	(h, B, N)
q_2	$(q_2, 1, L)$	$(q_1, 1, R)$	(q_2, B, R)

Sol:- the action of TM for the string w = 0101 as follows. Consider the following transitions:

→ $\delta(q_0 B) = (q_2, B, R)$ this is represented by

$q_0 B 0101 B \vdash B q_2 0101 B$

→ $\delta(q_2,0) = (q_2,1,L)$

$Bq_20101B \vdash q_2B1101B$

→ $\delta(q_2,B) = (q_2,B,R)$

$q_2B1101B \vdash Bq_21101B$

→ $\delta(q_2,1) = (q_1,1,R)$

$Bq_21101B \vdash Bq_1101B$

→ $\delta(q_1,1) = (q_1,B,R)$

$Bq_1101B \vdash BBq_101B$

→ $\delta(q_1,0) = (q_2,0,R)$

$BBq_101B \vdash BBOq_21B$

→ $\delta(q_2,1) = (q_1,1,BR)$

$BBOq_21B \vdash BBOlq_1B$

→ $\delta(q_1,B) = (h,B,N)$

$BBOlq_1B \vdash BBOlBh$

Thus the move is

$q_0B0101B \vdash Bq_20101B$
$\vdash q_2B1101B$
$\vdash Bq_21101B$
$\vdash Bq_1101B$
$\vdash BBq_101B$
$\vdash BBOq_21B$
$\vdash BBOlq_1B$
$\vdash BBOlBh$

the equivalent notation is

$q_0B0101B \overset{*}{\vdash} BBOlBh.$

Language of a TM :- the language accepted by a TM is the set of accepted strings $w \in \Sigma^*$.

Formally, let $M = (Q, \Sigma, \Gamma, \delta, q_0, h, B)$ be a TM. The language accepted by TM denoted by $L(M)$ is defined as

$$L(M) = \{w \mid w \in \Sigma^* \text{ and if } w = a_1 a_2 \dots a_n \text{ then } (q_0, e, a_1, a_2 \dots a_n) \vdash^* (h, b_1, b_2 \dots b_n) \text{ for some } b_1, b_2 \dots b_n \in \Gamma^*\}$$

(or)

$$L(M) = \{w : q_0 w \vdash x_1 h x_2\}$$

→ **Turing acceptable language** :- A language L over some alphabet is said to be Turing acceptable language, if there exists a TM, M such that $L = L(M)$.

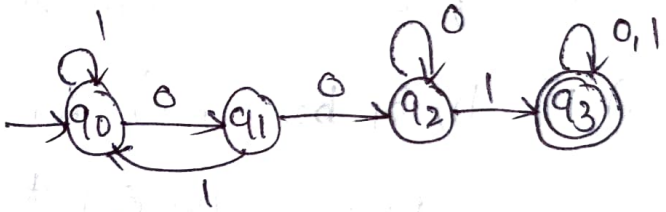
→ **Turing decidable language** :- A language L over Σ i.e $L \subseteq \Sigma^*$ is said to be Turing decidable, if both the languages L and it's complement $\Sigma^* - L$ are Turing acceptable.

→ **Recursively enumerable language** :- A language L is recursively enumerable, if it is accepted by a TM.

Design of TM's :-

Ex :- obtain a TM to accept the language $L=\{w \mid w \in (0+1)^*\}$ containing the substring 001.

Sol :- The DFA which accepts the language consisting of strings of 0's and 1's having a substring 001.



The transition table for the above DFA

	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
q_3	q_3	q_3

TM also processes the input string in only one direction. TM enters into the same states on same input symbols, replacing 0 by 0 and 1 by 1 & read-write head moves towards right. The transition table for TM is

	0	1	B
q_0	$(q_1, 0, R)$	$(q_0, 1, R)$	-
q_1	$(q_2, 0, R)$	$(q_0, 1, R)$	-
q_2	$(q_2, 0, R)$	$(q_3, 1, R)$	-
q_3	$(q_3, 0, R)$	$(q_3, 1, R)$	(q_4, B, R)
q_4	-	-	-

The TM is given by

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

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

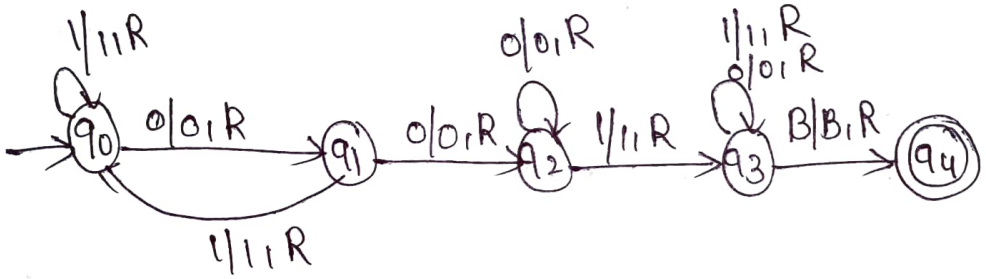
$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

$q_0 = \{q_0\}$ $F = \{q_4\}$

B = Blank character

The transition diagram for this is shown below.



Ex:2 obtain a Turing machine to accept the language

$$L = \{0^n 1^n \mid n \geq 1\}$$

Sol:-

procedure :-

→ Let q_0 be the start state. The general procedure to design TM for this case is

→ Replace the leftmost 0 by X and change the state to q_1 and then move head towards right. After 0 is replaced, we have to replace the corresponding '1'. So that no. of 0's matches with the no. of 1's.

→ Search for the leftmost '1' and replace it by 'y'

and move towards left. steps 2 and 3 can be repeated.

Consider the situation,

$$\begin{array}{l} XX00YY11 \\ \uparrow \\ q_0 \end{array}$$

1) $\delta(q_0, 0) = (q_1, X, R)$

The resulting configuration is

$$\begin{array}{l} XXX0YY11 \\ \uparrow q_1 \end{array}$$

The transitions for this can be of the form

2) $\delta(q_1, 0) = (q_1, 0, R)$

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

when these transitions are repeatedly applied, the following configuration is obtained.

$$\begin{array}{l} XXX0YY11 \\ \uparrow \\ q_1 \end{array}$$

3) The transition for this can be of the form

$\delta(q_1, 1) = (q_2, Y, L)$

The resulting configuration is shown below

$$\begin{array}{l} XXX0YYY1 \\ \uparrow \\ q_2 \end{array}$$

4) the transitions for this can be of the form

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

$\delta(q_2, 0) = (q_2, 0, L)$

the following configuration is obtained

$$XXX\underset{\uparrow}{0}YYY1$$
$$q_2$$

5) the transitions for this can be of the form

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

and following configuration is obtained.

$$XXX\underset{\uparrow}{0}YYY1$$
$$q_0$$

Now repeating steps 1 through 5, we get the configuration shown below

$$XXXX\underset{\uparrow}{Y}YYY$$
$$q_0$$

6) $\delta(q_0, Y) = (q_3, Y, R)$ and the configuration

$$XXXXY\underset{\uparrow}{Y}YY$$
$$q_3$$

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

Repeatedly applying this transition

$$XXXXYYYYB$$
$$\uparrow q_3$$

$$\delta(q_3, B) = (q_4, B, R)$$

where q_4 is the final state, the configuration

$$XXXXYYYYBB$$
$$\uparrow$$
$$q_4$$

so the turing machine to accept the language $L=\{a^n b^n \mid n \geq 1\}$ is given by

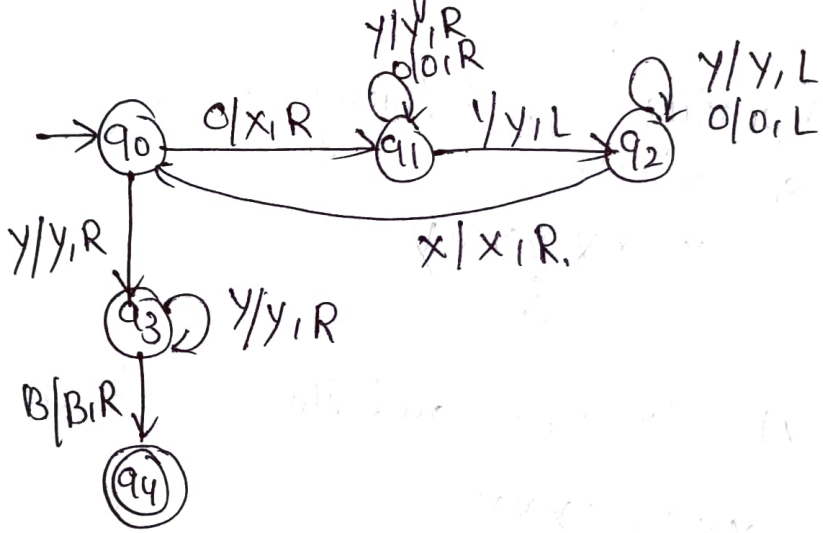
$M = (Q, \Sigma, \Gamma, \delta, q_0, F, B)$

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

$\Sigma = \{0, 1\}$ $\Gamma = \{0, 1, X, Y, B\}$

$q_0 = \{q_0\}$ $F = \{q_4\}$

The transition diagram for the TM is



To accept the string :- the input string $w = 0011$

Initial ID :- $q_0 0011 \vdash X q_1 011$

$\vdash X 0 q_1 11$

$\vdash X q_2 0 Y 1$

$\vdash q_2 X 0 Y 1$

$\vdash X q_0 0 Y 1$

$\vdash X X q_1 Y 1$

$\vdash X X Y q_1 1$

$\vdash X X q_2 Y Y$

$\vdash X q_2 X Y Y$

$\vdash X X q_0 Y Y$

$\vdash X X Y q_3 Y$

$\vdash X X Y Y q_3$

$\vdash X X Y Y B q_4$

Since the final state q_4 is reached, the string 0011 is rejected.

Techniques for Turing machine construction :- Three different Techniques are used for constructing Turing machine.

1) storage in the state
2) Multiple track TM
3) subroutine.

Storage in the state :- A state can be used for storing a symbol and for holding transitions. Generally a state contains a set of two things. i.e (q,a)

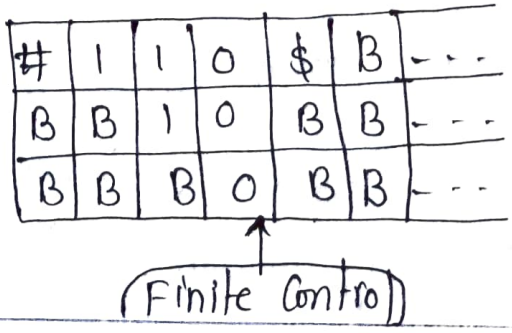
where q → state
a → tape symbol which is stored in (q,a)

Multiple track TM :- the input tape is to be divided into several tracks. each track consists of cells which holds that data i.e to be processed.

→ All these tracks uses a single head for performing both read and write operations.

→ The cells which do not contain any inputs hold the blank input which is represented by the letter 'B'

Ex :- Consider TM that performs the devision operation on 6 by 2 and returns the remainder.



In the above figure, the numbers are (6 and 2) converted into their binary form and are placed at the first and second tracks of the tape.

→ The last track of the tape shows the result of the devision that is remainder of 6 devided by 2 is equal to '0'.

3) Sabroutines :- Thee are Commonly used in programming languages and it can be implimented in turing machines.

→ It is dubined as a self contained block group of instructions that can performs one task.

→ It is a portion of the software that can be stored in memory once, but can be used as many times as required.

→ This leads to less usage of memory and easy software duelopment.

At the time of program execution, a subroutine is called multiple times to carry out its functions at different points.

→ After execution of the subroutine, control is returned back to the main program so that the programe may resume it's normal operations.

Types of Turing machine :-

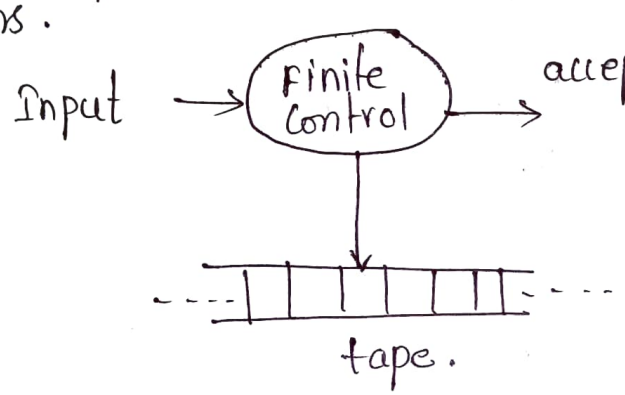
variouy types of turing machines are

1) Turing machine with two way infinite tape
2) Multiple turing machine

3) Multihead Turing machine
4) Multidimentional TM
5) Nondeterministic TM
6) offline TM.

1) Turing machine with two-way infinite tape :-

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



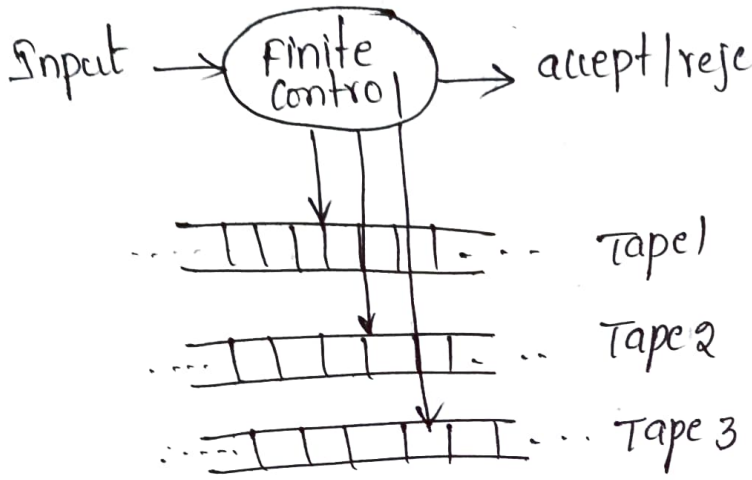
it turns out that this type of Turing machines are as powerful as one tape Turing machines whose tape has left end.

2) Multiple Turing machine :- (or) Multitape TM :-

A multiple turing machine consists of a finite control with k tape heads and k tapes, each tape is infinite in both directions.

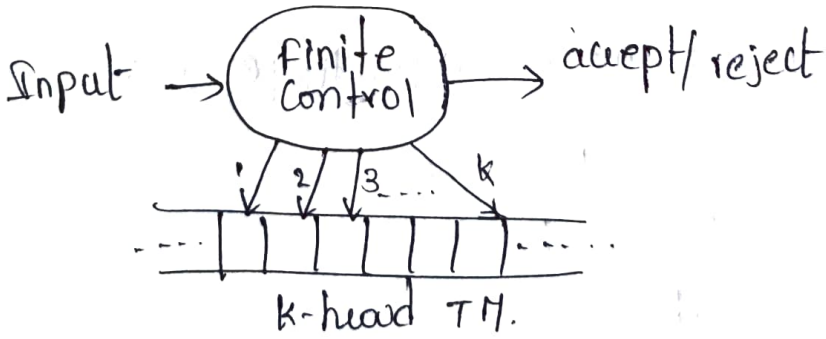
on a single move depending on the state of the finite control and the symbol scanned by each of the tape heads, the machine can

1) change state
2) print a new symbol on each of the cells scanned by it's tape heads.
3) move each of it's tape heads, independently.



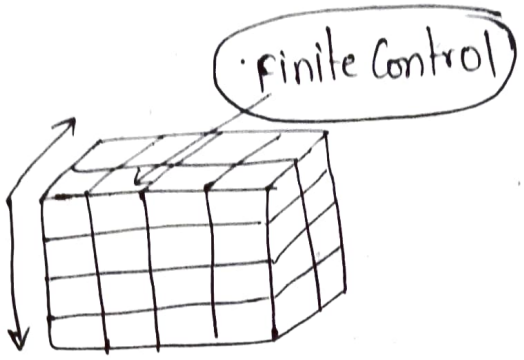
Initially, the input appears on the first tape and the other tapes are blank.

3) Multihead TM :- A k-head turing machine has some fixed number, k heads.



The heads are numbered 1 through k, and a move of the TM depends on the state and on the symbol scanned by each head. In one move, the heads may each move indipendently left, right or remain stationary.

4) Multidimensional TM :- The TM has the usual finite control, but the tape consists of a k-dimensional array of cells infinite in 2k directions, for some fixed k.



Depending on the state and symbol scanned, the device changes state, prints a new symbol and moves its tape head in one of 2k directions, either positively (or) negatively along one of the k-axes.

5) Non deterministic Turing machine :-

A NTM is a device with a finite control and a single, one way infinite tape

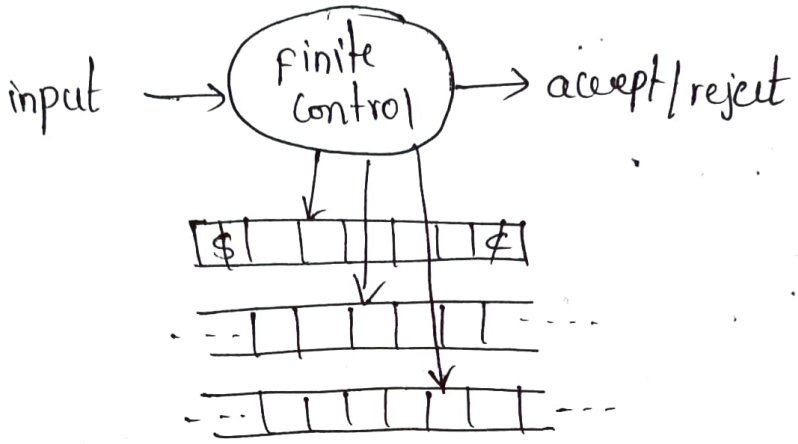
→ For a given state and a tape symbol scanned by the tape head, the machine has a finite number of choices for the next move.

→ The non deterministic TM accepts it's input if any sequences of choices of moves leads to an accepting state.

6) off line Turing machine :- An off-line Turing machine is a multiple TM whose input tape is read only. The Turing machine is not allowed to move the input tape head off the region

between $ and ¢

→ Offline TM is just a special case of the multiple TM, and is no more powerful than any of the models we have considered.



Church's Hypothesis :- According to church's hypothesis, all the functions which can be defined by human beings can be computed by turing machine. The turing machine is believed to be ultimate computing machine.

The church's original statement was slightly different because he gave his thesis before machines were actually developed. He said that any machine that can do certain list of operations will be able to perform all algorithms

church tied both recursive functions and computable functions together. Every partial recursive function is computable on TM such as RAM.

Important of church's hypothesis is as follows,

1) First we will prove certain problems which cannot be solved using TM.
2) If churches thesis is true this implies that problems cannot be solved by any computer (or) any programming languages we might ever develop.
3) Thus in studying the capabilities and limitations of Turing machines we are indeed studying the fundamental capabilities and limitations of any computational device we might ever construct.

Universal Turing machine :-

A Turing machine is said to be universal TM (U_M) if it is capable of simulating computation of turing machine (M) by taking it's discription and string w as input.

It is constructed in the following way

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

$$M = \{Q, \{0,1\}, \{0,1,B\}, \delta, q, B, q_1\}$$

where $Q = \{q_0, q_1, \ldots q_n\}$

$\Sigma = \{0,1\}$ $\{\Gamma = \{0,1,B\}$, $F = \{q_1\}$

The left and Right head directions are denoted by d_L and d_R respectively.

→ Um transforms the transitions into a particular binary format where in every symbol is isolated by '1'.

Consider the transition $\delta(q_i, d_j) = (q_k, q_l, d_m)$

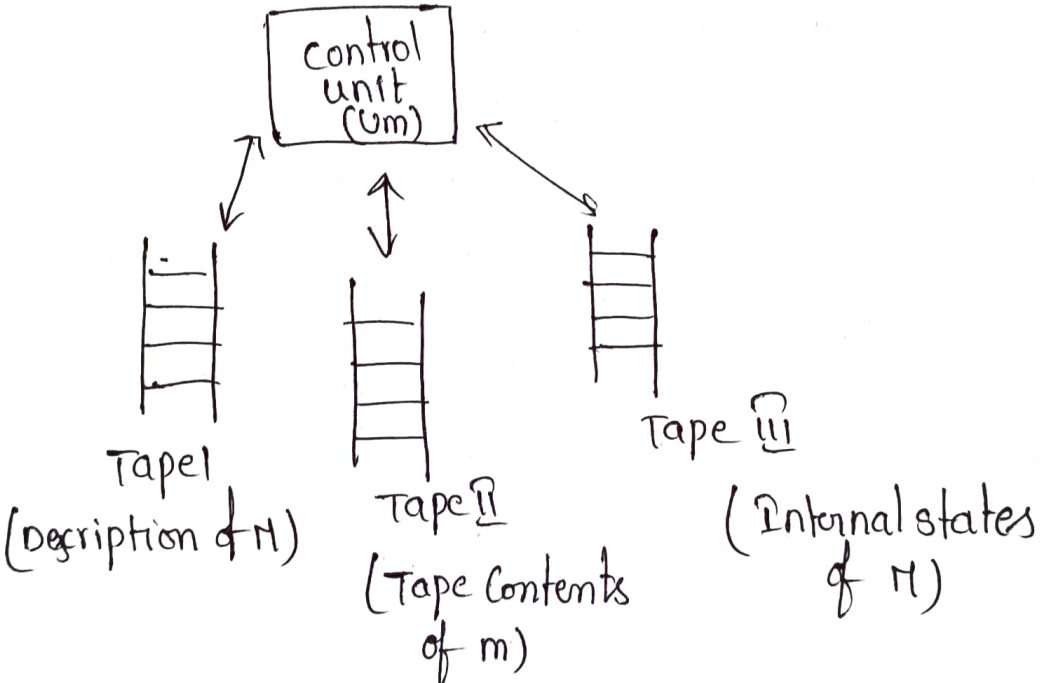
K. Kumar

→ this transitions are transformed to binary format by Um as $0^i10^j10^k10^l10^m$

→ Incase if the turing machine (M) contains the transitions $t_1, t_2 \ldots t_n$ then Um as follows as,

$$111\ t_1\ 11\ t_2\ 11 \ldots 11\ t_n 111$$

To perform unification of string Um supports the form $\langle M, w \rangle$ to describe the problem, where M represents the definition of TM and w represents the string.



The behaviour of turing machine M is evaluated in the following way. varify the tape I format,

1) a) if $i = j$, Multiple transitions must be with $0^i0^j_1$
   b) it must hold the Condition $1 \le x \le 3$ where $x = j, l, m$ for the transition $0^i10^j10^k10^l10^m$
2) Initialize the tape 2 and tape 3 to hold w and '0' respectively during their Initialization.

3) stopping the TM if the contents of tape III becomes 000 which is the final state.

4) scan tape I from the left end to the second 111 looking for substring beginning with 110^i10^j1.

a) if no such string is found then halt and reject.

b) if places the new state (0^k) on tape III, d_i on the tape cell in place of a_j and proceed the head in direction Dm.

The following conclusions are made,

→ U_m accepts $\langle M, w \rangle$ if w is accepted by M.

→ U_m stops and rejects w if M rejects w

→ U_m runs infinitely on $\langle M, w \rangle$ if M runs infinitely on w.

Restricted Turing machine :-

we can have so many variations of standard Turing machines. with minor modifications we can have the following Turing machines :

→ Turing machine with semi-infinite tape

→ Multi stack machines

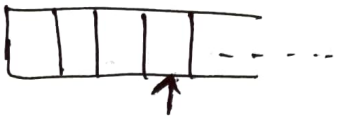
→ counter machines.

1) Turing machine with semi-infinite tape :-

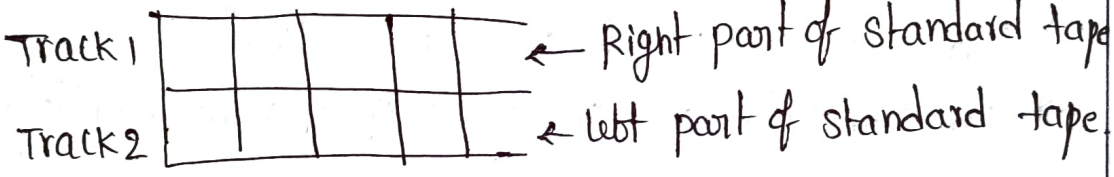
In the standard TM, there was no boundary specified for the left side as well as right side of the tape. The read/write head can be moved infinitely towards left

as well as towards right i.e the tape was unbounded in both the directions.

Now if we restrict the read/write head to move only in one direction say towards right, then the Turing machine is called TM with semi infinite tape.



Here no cells to the left hand side of the initial head position. The TM M can simulate the semi infinite tape Ms using two tracks for a tape.



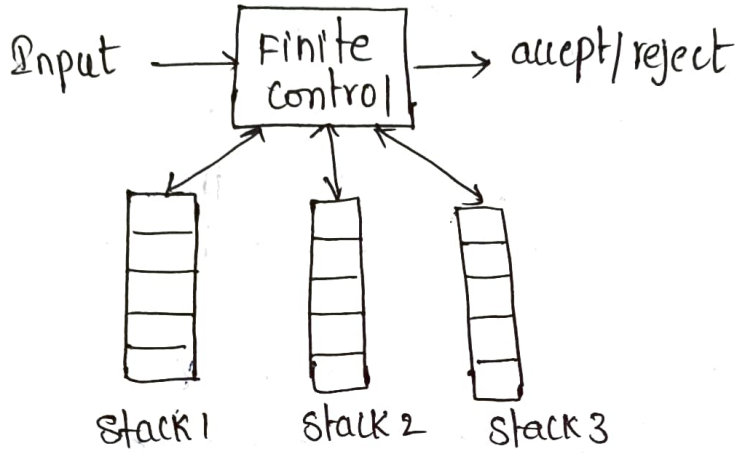
→ The upper part represents track 1 and stores the information which lies to the right of some reference point.

→ The lower part i.e track 2 contains the left part of the reference point in reverse order.

We can partition the states of M into Q_1 and Q_2 so that the states of Q_1 are used when working with respect to track 1 and states of Q_2 are used when working with respect to track 2.

Some special markers such as #s can be placed at the left hand side of track 1 and track 2 to enable the programmer to switch between the tracks.

2) Multi stack machines :- All the languages which are accepted by PDA are accepted by TM. At the same time, any language which is not accepted by PDA is also accepted by TM. TM is a generalized machine which can accept all languages. The multi stack machine is a generalized model of PDA.



⇒ the pictorial representation of 3-stack machine will have 3 stacks. In general, a k-stack machine is a deterministic PDA with k stacks.

The move of the multistack depends on:

→ the current state of the finite control
→ the input symbol choosen from Σ
→ the symbol on top of each stack.

once the transition is applied for the current state, input symbol and the current top of each stack, the multistack machine may:

→ change the state
→ it may replace the top of the stack.

$\delta(q, a, X_1, X_2 \ldots X_k) = (P, \alpha_1, \alpha_2 \ldots \alpha_k)$ where X_i is on top of the stack for $1 \le i \le k$. It is clear from this transition that each symbol on top of the stack can be replaced by different symbols.

3) counter machines :-

The counter machine is a restricted multi stack machine which can interpreted in several ways.

→ counter machine is similar to that of multistack machine with some modifications. Each stack in the machine is replaced by a counter. Each counter holds a non-negative number.

The move of the machine depends on the current state, current input symbol and if one of the value of the counter is zero. The machine can do the following activities in one move :

- → it can change the state
- → It can add or subtract 1 from the counters. But if the counter is zero, subtracting of 1 from that counter is not possible.

→ the languages accepted by counter machines are recursively enumerable i.e we can obtain an equivalent TM to accept those languages which are accepted by counter machines.

Computability

Decidable and undecidable problems :-

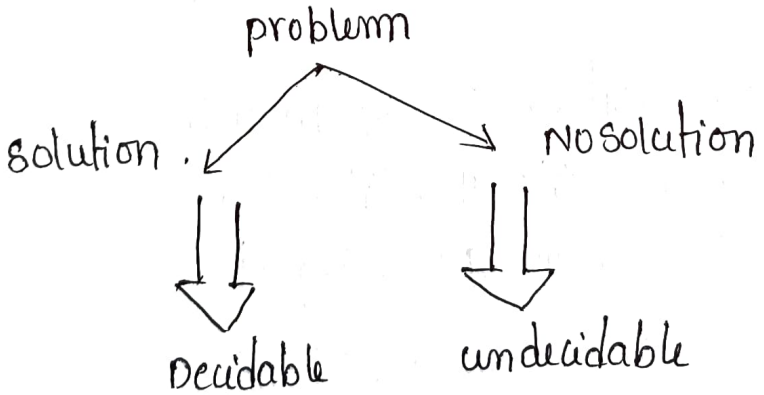
Decidable problems :- A problem is said to be decidable, if it support 'recursive languages'.

→ A decision problem can be easily solved with the help of algorithmic approach, then that problem is called as decidable problem.

(or)

If a problem can be solved or answered based on some algorithm, then it is decidable otherwise undecidable.

undecidable problems :- A decision problem which cannot be solved by any algorithmic approach is refered as undecidable. In other words, problems for which no algorithm exists are undecidable.



Examples of decidability & undecidability :-

Consider the following problems and their possible answers,

1) Does the sunrise in the east (yes)

2) Does the earth move around the sun? (Yes)

3) will tommorrow be a rainy day? (No answer)

we have solutions for all problems except the last. we cannot answer the last problem, because we have no way to tell about the weather of tomorrow, but to some extent we can only predict.

Halting problem of a Turing machine :-

Halting problem is nothing but a decision problem that is encountered while designing a turing machine. various scientists defines this problem in a different manner.

→ Given a discription of a problem and a finite input, halting problem decide whether the program finishes running (or) will run forever, in loops for given input.

(or)

→ Let M_1 an arbitary turing machine and 'Σ_1' arbitary input for the machine, then it check whether the given machine halts on the given input.

one of the important consequence of the undecidability of the halting problem is Rice's theorem which states that the truth of any nontrivial statement about the function that is defined by an algorithm is undecidable.

Post correspondence problem (PCP) :-

It was first introduced by Emil post in 1946. An instance of post correspondence problem (PCP) consists of two lists of strings over some alphabet Σ the two lists must be of equal length, generally refer them as A and B lists and write

$$A = w_1, w_2 \cdots w_k$$

and next $B = x_1, x_2 \cdots x_k$ for some integer k.

For each i, the pair (w_i, x_i) is said to be a corresponding pair.

this instance of pcp has a solution, if there is a sequence of one or more integers $i_1, i_2 \cdots i_m$ that, when interpreted as indices for strings in the A and B lists, yield the same string i.e

$$w_{i_1}, w_{i_2} \cdots w_{i_m} = x_{i_1}, x_{i_2} \cdots x_{i_m}$$

The sequence $i_1, i_2 \cdots i_m$ is a solution to this instance of Pcp. pcp is undecidable.

Example :- Let $\Sigma = \{0, 1\}$. Let A and B lists of 3 strings each

	list A	list B
i	w_i	x_i
1	11	111
2	100	011
3	111	11

in this case there pcp has a solution 1 2 3 $m=3$, $i_1=1, i_2=2, i_3=3$

The sequence is $w_1 w_2 w_3 = x_1 x_2 x_3 = 111001111$

Modified post correspondence problem :-

This is an intermediate version of the pcp called modified pcp or MPCP where an instance of the Mpcp is exactly the same as an instance of pcp with an additional requirement on a solution sequence.

In mpcp, the first element of the sequence plays an important role. we say that the pair (T, S) has a modified post correspondence solution if there exists a sequence of zero (or) more integers $i_1, i_2, \ldots i_m$ such that

$$w_1, w_{i_1}, w_{i_2} \cdots w_{i_n} = x_1, x_{i_1}, x_{i_2} \cdots x_{i_n}$$

The empty list could be a solution in mpcp $w_1 = x_1$, which is different from the pcp where the solution has to have at least one integer on the solution list.

classes of paied NP :-

There are many different classifications depending on the time taken by the problem. The following are the types of classication.

- → P problem
- → NP problem.

1) P problem :- These problems can be solved by a Turing machine (deterministic) in polynomial time (p stands for polynomial). P problems are a class of problems which can be solved efficiently.

P-problems are usually decision problems that include p-class. All problems in p-class is said to be tractable if the time and space complexity for machine is reasonble to solve that problem otherwise the problem of p-class is called as intractable.

Examples of p-class problems.

1) Transitive closure
2) Matrix multiplication
3) Kruskals algorithm for minimum spanning

NP problem :- These are the problems that can be solved by a non deterministic Turing machine in polynomial time.

→ A problem is in NP if you can quickly test whether a solution is correct
→ NP problems are a class of problems which cannot be solved efficiently. NP does not stand for non polynomial.
→ There are many complexity classes that are much harder than NP.
→ The relationship between the paired NP-problems is $P \subseteq NP$ i.e p problem is subset of (or) equal to set of NP-problems.

Examples of NP-problems are

1) Four color problem
2) Travelling salesman problem
3) The vertex Cover problem
4) the hamilton cycle problem.

NP-Complete and NP-hard problems :-

NP Complete problem :- A problem is said to be NP-Complete if and only if it satisfies following conditions,

1) problem should be present in class NP
2) Each problem should also be solved in polynomial time. (i.e NP-hard problem)

However, there is no one who can prove whether NP-Complet problems can be solved in polynomial time (or) not. Hence these types of problems are unsoluable problems.

Ex :- 1) graph Isomorphism
2) Knapsack problem
3) Travelling salesman problem
4) Graph coloring problem.

NP-hard problem :- A problem is said to be NP-hard if and only if it is solved in polynomial time (or) reduced to an NP-hard problem.

whenever, an NP-hard problem is solvable

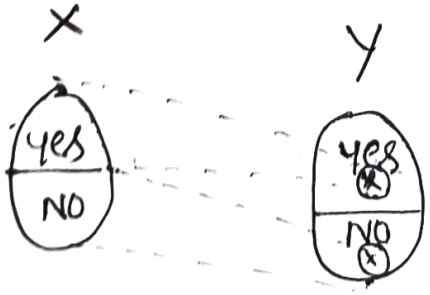
then each NP-problem can also be solved (in polynomial time).

Example of NP-hard problem is travelling salesman problem. It is a problem in which cost of travelling between two pair of cities and no. of cities are provided.

However each NP-complete problem is said to be NP-hard whereas each NP-hard problem cannot said to be NP-Complete problem.

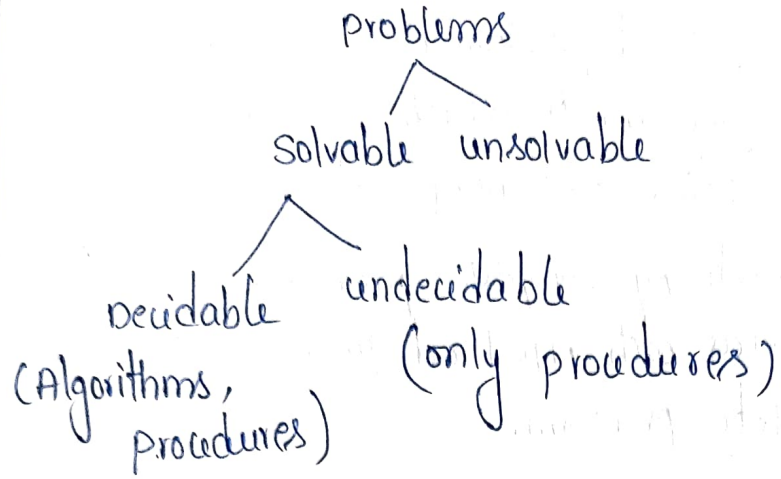
Turing reducibility :- Reduction is a Technique in which if a problem X is reduced to problem Y then any solution of Y solves X.

In general, if we have an algorithm to convert some instance of problem X to some instance of problem Y that have the same answer then it is called X reduces Y.



Definition: Let X and Y be the two sets such that $X, Y \subseteq N$ of natural numbers. then X is Turing reducible to Y and denoted as $X \leq_T Y$.

Decidable & undecidable problems :-



Procedure :- step by step instruction to solve a problem there is no time constraint.

Algorithm :-

Algorithm = procedure + Approximate time in which a problem can be solved.

→ If you are able to specify how much time is required to solve a problem. is algorithm

→ If you are not able to tell how much time is required then it is procedure.

→ The procedure can halt (or) need not halt but algorithm always halt and give you the output.

Ex :- bubble sort algorithm - Time Complexity - $O(n^2)$ worst case

Insertion sort algorithm - worst case time Complexity - $(O(n^2))$

Decidable :- solution is definite (ether yes or no)

Ex :- → Does sun rises in the east ?

Ans : yes

→ Does earth moves around the sun ?

Ans :- yes

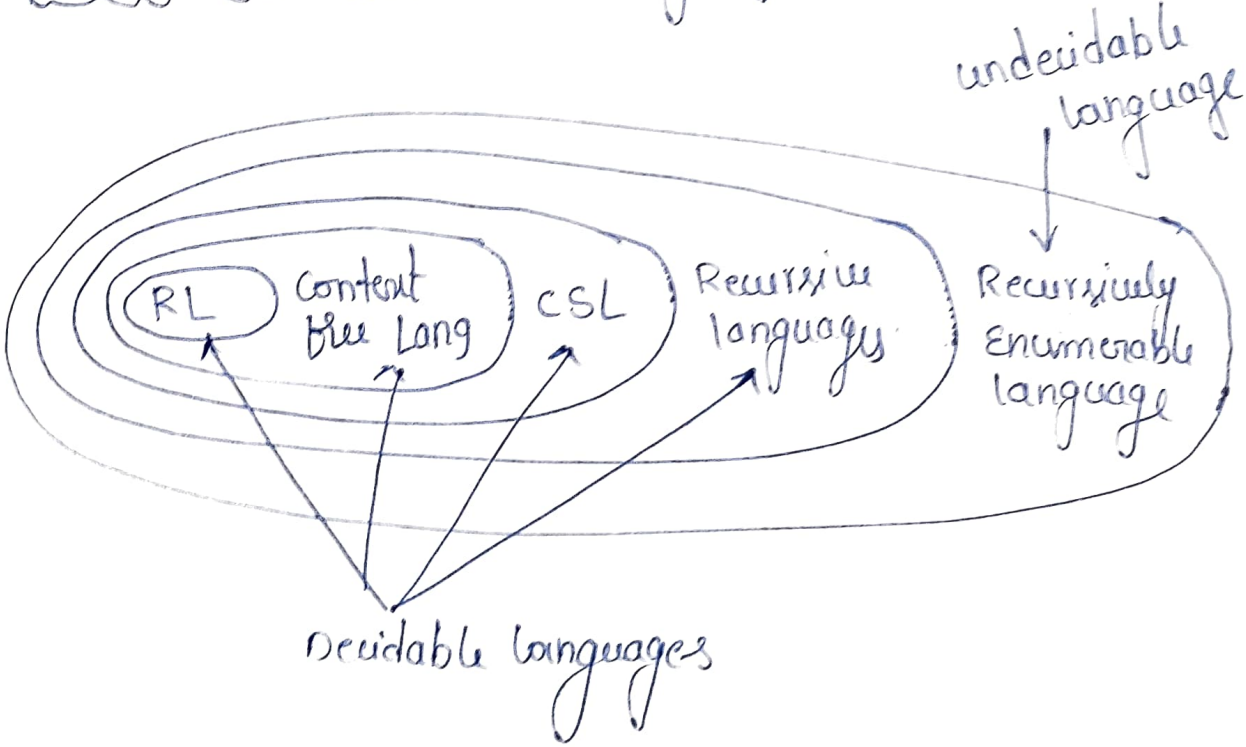
→ Does sun rises in the west ?

Ans :- NO

undecidable :- solution is indefinite (i.e sometimes 'yes' and sometimes 'no')

Ex :- will tomorrow would be a rainy day ?

Decidable and undecidable languages :-



Decision problem :- Any problem for which the answer is either yes (or) no is called decision problem. The algorithm for decision problem is called decision algorithm.

Ex :- → sum of subsets problem
→ max clique problem.

Deterministic algorithm :- The algorithm in which every operation is uniquely defined is called deterministic algorithm.

Ex :- PCA algorithm

Non deterministic algorithm :- the algorithm in which operations are not uniquely defined but are limited to specific set of possibilities for every operation is called non deterministic algorithm.

Ex :- Travelling sales person problem

P and NP classes :-

Definition of P :- P denotes set of all decision problems solvable by deterministic algorithm in polynomial time.

(or)

Problems that can be solved in polynomial time.

Ex :- 1) Basic Multiplication of two numbers
2) Sorting
3) Searching

Definition of NP :- NP denote set of all decision problems solvable by non deterministic algorithm in polynomial time.

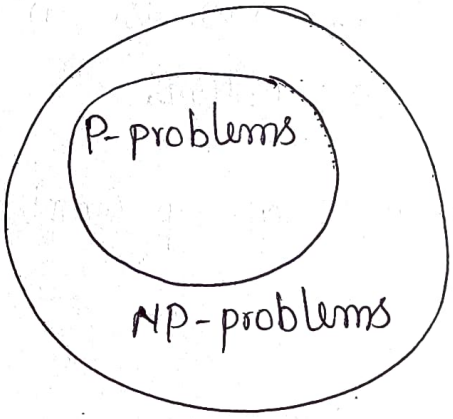
→ NP means non deterministic polynomial time.

Ex :- 1) Traveling sales person problem

2) Knapsack problem.

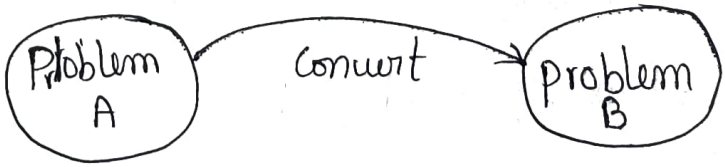
∴ since deterministic algorithms are special case of non-deterministic algorithms, $P \subseteq NP$

∴ All NP problems are Non deterministic in nature.



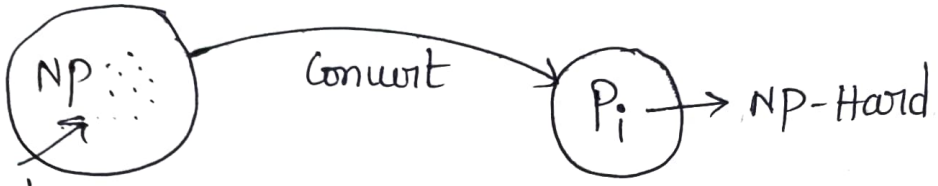
NP-Hard and NP-complete classes :-

Reduction :- A problem it is difficult to solve a problem in such case that problem is converted to another problem.



Conversion time is polynomial time i.e $O(n^k)$

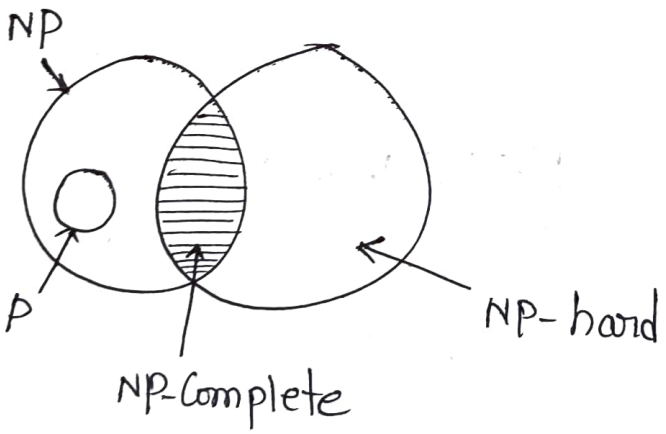
NP-Hard problem :- P_i is said to be NP-hard if all NP problems are reducible to P_i in polynomial-time.



all NP-class Problems — Conversion time is polynomial time i.e $O(n^k)$

NP-Complete problem :- After reducing this all NP-problems into P_i, the resulting problem may be NP (or) may not be NP. If that resulting problem is also in NP, then that problem is NP-Complete problem..

Relationship among P, NP, NP-Complete and NP-hard



→ Every NP-hard problem need not be NP-problem

→ NP problem as well as NP hard problem is NP-Complete problem.

Post correspondence problem (pcp):-

The post correspondence problem is an undecidable decision problem that was introduced by Emil post in 1946. Because it is simpler than the halting problem.

→ It is very helpful tool for proving problems in logic (or) in formal language theory to be undecidable.

→ i.e the undecidability of string is determined with the help of post correspondence problem (pcp).

Statement:- The pcp consists of two lists of strings that are of equal length over the input alphabet, two lists are $A = w_1, w_2, w_3 \ldots w_n$ and $B = x_1, x_2, x_3 \ldots x_n$ then there exists a non-empty set of integers $i_1, i_2 \ldots i_n$ such that $w_1 w_2 w_3 \ldots w_n = x_1 x_2 x_3 \ldots x_n$, to solve the pcp we try all the combinations of $i_1, i_2 \ldots i_n$ to find $w_i = x_i$ then we say that pcp has a solution.

→ pcp contains set of dominos / Tiles / elements / dice

→ Dominos are denoted as pair of elements.

Ex:- $\left[\frac{ab}{ac}\right], \left[\frac{c}{b}\right]$

(or)

$X = \{ab, c\}$, $Y = \{ac, b\}$ (or)

i	x_i	y_i
1	ab	ac
2	c	b

→ we can use each domino for finite number of times to find the solution

→ PCP is Non deterministic problem. i.e solution is not unique.

→ Find the finite sequence of dominos to form the strings such that top and bottom of the strings are same.

Ex:1 Prove that PCP have the solution with the two lists

$A = \{1, 10111, 10\}$ $B = \{111, 10, 0\}$

Sol:- The dominos are

$$\left[\frac{1}{111}\right], \left[\frac{10111}{10}\right], \left[\frac{10}{0}\right]$$

Start with the matching domino $\left[\frac{x\emptyset 11}{x\emptyset}\right]$

$$\Rightarrow \left[\frac{x\emptyset xxx}{1\emptyset}\right]\left[\frac{1}{xxx}\right]$$

$$\Rightarrow \left[\frac{10111}{10}\right]\left[\frac{1}{111}\right]\left[\frac{1}{111}\right]$$

$$\Rightarrow \left[\frac{10111}{10}\right]\left[\frac{1}{111}\right]\left[\frac{1}{111}\right]\left[\frac{10}{0}\right]$$

⇒ Now 101111110 → Top string

101111110 → Bottom string

Both the strings are same.

∴ PCP has solution.

Ex2 :- prove that pcp with two lists $A = \{01, 1, 1\}$ $B = \{111, 10, 11\}$ has no solution with justification

Sol:-

i	a_i	b_i
1	01	111
2	1	10
3	1	11

Case 1 :- $\left[\frac{01}{111}\right]_1 \left[\frac{1}{10}\right]_2 \left[\frac{1}{11}\right]_3$

First symbol not matched. It can not be Considered

Case 2 :- 2nd domino

$\Rightarrow \left[\frac{x}{1\emptyset}\right]^2 \left[\frac{\emptyset x}{x11}\right]^1$

$\Rightarrow \left[\frac{x}{1\emptyset}\right]^2 \left[\frac{\emptyset x}{x11}\right]^1 \left[\frac{x}{10}\right]^2$

$\Rightarrow \left[\frac{x}{1\emptyset}\right] \left[\frac{\emptyset x}{xxx}\right] \left[\frac{x}{10}\right] \left[\frac{x}{11}\right]^3$

$\Rightarrow \left[\frac{1}{10}\right] \left[\frac{01}{111}\right] \left[\frac{1}{10}\right] \left[\frac{1}{11}\right] \left[\frac{1}{11}\right]^3$

$\Rightarrow \left[\frac{1}{10}\right] \left[\frac{01}{111}\right] \left[\frac{1}{10}\right] \left[\frac{1}{11}\right] \left[\frac{1}{11}\right] \left[\frac{01}{111}\right]$

$\Rightarrow \left[\frac{1}{10}\right] \left[\frac{01}{111}\right] \left[\frac{1}{10}\right] \left[\frac{1}{11}\right] \left[\frac{1}{11}\right] \left[\frac{01}{111}\right] \left[\frac{1}{11}\right]^3$

$\Rightarrow$ " " $\left[\frac{1}{11}\right]^3$

$\Rightarrow$ " " " $\left[\frac{1}{11}\right]^3$

same steps are repeating. domino three is repeated number of times.

so the pcp doesn't have solution.

(3) Prove that pcp has solution for $x: \{a, b, ca, a, abc\}$ and $y: \{ab, ca, a, ab, c\}$.

Modified post correspondence problem

if the first substring used in apcp are w_1 and x_1 for all instances then the pcp is called mpcp.

(or)

Another version of pcp is modified post correspondence problem i.e it is the 1st element in both x and y string should begin with x_i and y_i respectively.

Ex:-

$\left[\frac{1}{111}\right]\left[\frac{10111}{10}\right]\left[\frac{10}{0}\right]$ 3 dominos

$x_1\left[\frac{1}{111}\right]$ $x_2\left[\frac{10111}{10}\right]$ $x_3\left[\frac{10}{0}\right]$
y_1 y_2 y_3

the output is

$$\begin{array}{l} \text{Top string} - \overset{x_2\ \ x_1\ \ x_1\ \ x_3}{10111\ \ 1\ \ 1\ \ 10} \\ \text{Bottom string} - \underset{y_2\ \ y_1\ \ y_1\ \ y_3}{10\ \ 111\ \ 111\ \ 0} \end{array} \Big\} \text{Both are same}$$

$\Rightarrow$ $\frac{x_2\ x_1\ x_1\ x_3}{y_2\ y_1\ y_1\ y_3}$

The first substring is should be same in mpcp i.e $x_2 = y_2$

Recursive language :- A language L and Turing machine TM, w is input string

→ for every $w \in L$, TM accepts $w \Rightarrow$ TM reaches final state and halts

→ For every $w \notin L$, TM reaches a non final state and halts
$\Rightarrow$ TM rejects w

$w \in$ TM ← final state + halt
Non final state + halt

→ If the language is recursive language, then we can decide on the membership property.

→ the membership function is decidable

→ Recursive languages are Turing decidable languages.

Recursively enumerable language :- 'L' is a language of Turing machine TM, w is the input string.

∀ $w \in L$, Tm accepts $\Rightarrow$ TM reaches final state & halts

∀ $w \notin L$ ← TM rejects w $\Rightarrow$ TM reaches non final state & halt
(or)
Infinite loop

i.e

$w \rightarrow$ TM ← final state + halt
non final state + halt
Infinite loop .

→ Membership function is undecidable for recursively enumerable language. & these are Turing undecidable lang's

NOTE:1 : Every recursive language is recursively Enumerable language

Note:2 : Every recursively enumerable language need not be recursive language.