UNIT - III

Context Free Grammars & Language

* Context Free Grammar (CFG) ≠ (Regular Grammar)

"A context free Grammar G is a quadruple (V, T, P, S)

Where, V is set of variables

T is set of terminals

P is set of productions

S is a special variable called start symbol $S \in V$.

A production is of the form

$V_i \rightarrow \alpha_i$ where $V_i \in V$ and α_i is string of terminal and variables.

Notations :-

1. terminal are denoted by lower case letters a, b, c or digits 0, 1, 2 --- 9.
2. Non-terminal (variables) are denoted by capital letters A, B, C --- Z.
3. A string of terminals or a word $w \in L$ is represented using u, v, w, x, y, z.
4. Sentential form is a string of terminals and variables and it is denoted by α, β, γ etc.

e.g. Anand writes, Anand reads, Sunny watch

A sentense for above word can be written as

<sentense> → <noun> <verb>

Noun → Anand | Sunny

Verb → writes | reads | watch.

P = { sentense → noun | verb ,
noun → Anand | sunny
verb → writes | reads | watch
}.

* Sentential form :

In Sentential form, derivation starts from the start symbol through finite application of productions.

A string α is derived so far consist of terminals and non-terminals.

$$S \overset{*}{\underset{G}{\Rightarrow}} \alpha \mid \alpha \in (V \cup T)^*$$

- A final string consist of terminals.
- In left sentential form, leftmost symbol is picked up for expansion.
- In right sentential form, rightmost symbol is picked up for expansion.
- A string can be derived in two ways.
  1) Leftmost Derivation
  2) Rightmost Derivation.

e.g.

$S \rightarrow A1B$

$A \rightarrow 0A \mid \epsilon$

$B \rightarrow 0B \mid 1B \mid \epsilon$

where G is given by (V, T, P, S)

with

$V = \{S, A, B\}$

$T = \{0, 1\}$

$P = \{$ production $S \rightarrow A1B$, $A \rightarrow 0A \mid \epsilon$, $B \rightarrow 0B \mid 1B \mid \epsilon \}$

S = start Symbol.

"A Language of grammar can be defined by creating the production rules for the given condition.

The language generated by context free grammar is called as Context Free Language (CFL)

Page No.
Date : / /

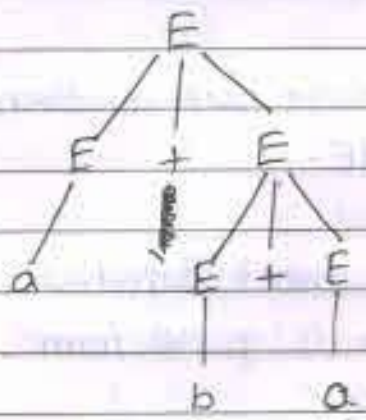
* Ambiguous Grammar :-

"A grammar is said to be ambiguous if the language generated by grammar contains some string that has two different parse tree".

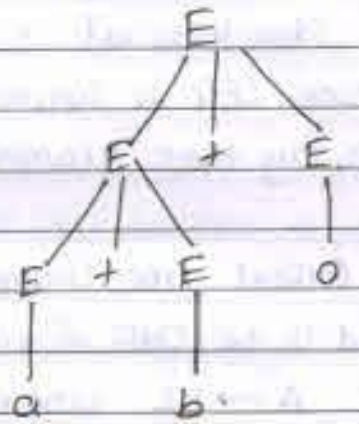
e.g.

$E \rightarrow E+E \mid a \mid b$;

A string a+b+a is generated by given grammar



Parse tree 1



Parse tree 2

As above, two parse tree are generated for string a+b+a so that's why it is ambiguous grammar.

* Elimination of Useless Symbol & Production

A grammar may contain symbols & productions which are not useful for derivation of string.

Two types of symbol are useless

1. Non-generating symbols
2. Non-reachable symbols

* Normal form :-

There are 2 imp normal form

1] Chomsky's Normal form

2] Greibach Normal form

1] Chomsky's Normal form (CNF) :-

→ The CNF can be defined as

Non-terminal → Non-terminal • Non terminal.

Non-terminal → Terminal

The given CFG is converted in the above format then we can say that grammar is in CNF.

"A Context Free Grammar (CFG) without ε-production is said to be CNF if every production is of the form.

1. $A \rightarrow BC$ where $A, B, C \in V$
2. $A \rightarrow a$ where $A \in V$ and $a \in T$.

The grammar should have no useless symbols.

Every CFG without ε Productions can be converted into an equivalent CNF form."

* Algorithm for CFG to CNF.

→ 1) Eliminate ε-production, unit Production & useless Symbol from grammar.

2) Every variable deriving a string of length 2 or more should consist only of variable.

$A \rightarrow \alpha$ with $|\alpha|$ should consist on variable.

e.g. $A \rightarrow V_2V_2aV_3bV_4$

or $A \rightarrow V_1V_2C_aV_3C_bV_4$ and adding two prod :-

$C_a \rightarrow a$, $C_b = C_b$

3) Every prodn deriving ≥ 3 or more variables

($A \rightarrow \alpha$ with $|\alpha| \geq 3$ can be split into cascade Production with each deriving a string of two variables.

e.g $A \rightarrow X_1X_2X_3 \ldots X_n$ where $n \geq 3$ with $|\alpha| \geq 3$

e.g Consider production $A \rightarrow X_1X_2 \ldots X_n$ where $n \geq 3$ and X_i are variable.

$A \rightarrow X_1C_1$

$C_1 \rightarrow X_2C_2$

$C_2 \rightarrow X_3C_3$

Page No.
Date : / /

* Greibach Normal form (GNF)

→ "A context free Grammar $G = (V, T, P, S)$ is said to be in GNF if every production is of the form

$$A \rightarrow a\alpha$$

where; $a \in T$ is a terminal &
α is string of zero or more variable

- The Language $L(G)$ should be without β
- RHS of each Prod should start with a terminal followed by a string.

- Removing Left Recursion :

→ Elimination of left Recursion is a important step in algo. used in conversion of a CFG into GNF form

- Left Recursive grammar :

A prod of the form $A \rightarrow A\alpha$ is called left recursive as the left hand side variable appears as the first symbol on the right-hand.

4. TURING MACHINE

* Introduction to Turing Machine

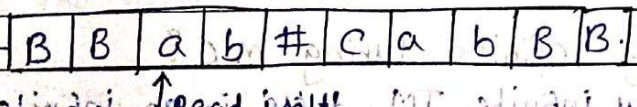
→ "Turing m/c is mathematical model which consist of an infinite length tape divided into cells on which input is given. It consist of head which reads the i/p tape."

- A state register stores the state turing m/c. After reading an i/p symbol, it is replaced with another symbol, it's internal state is changed and it moves from one cell to the right or left.
- If TM m/c reaches to final state, the i/p string is accepted otherwise rejected.

A TM m/c can be described as 7-tuple $(Q, X, \Sigma, \delta, q_0, B, F)$ where,

Q is finite set of state.
X is tape alphabet
Σ is i/p alphabet
δ is transition function
q_0 is initial state
B is Blank symbol
F is set of final state.

e.g.



Example of Turing Machine.

- Turing m/c is more powerful than PDA.
- Turing m/c is capable of performing computation on i/p and producing a new result.

* Applications of Turing Machine :

→ 1) To read or write infinite tape.
2) to solve problem in computer science & testing limit of computation.
3) it is used to simulate other turing machine.
4) Turing m/c is used to reverse string of any character.
5) it is used in algorithmic information theory
6) Turing m/c is used for high performance computing, m/c learning, SW engg and computer network.
7) Turing m/c is used to perform computation for compiler system.
8) Turing m/c is used in theory of computation.

* Different ways of Extension of TM :

→ In std TM, the tape is semi-infinite. It is bounded on the left and unbounded on the right side.

Some of the Extension of TM :

1) Tape is of infinite length in both the direction.
2) Multiple heads and single tapes.
3) Multiple tape with each tape having its own independent head.
4) K-dimensional tape.
5) Non-deterministic turing m/c.

1) Two-way infinite turing m/c :

— In 2-way infinite TM, there is an infinite sequence of blank on each side of i/p string. In instantaneous description, these blocks are never shown.

2) A turing m/c with multiple head -

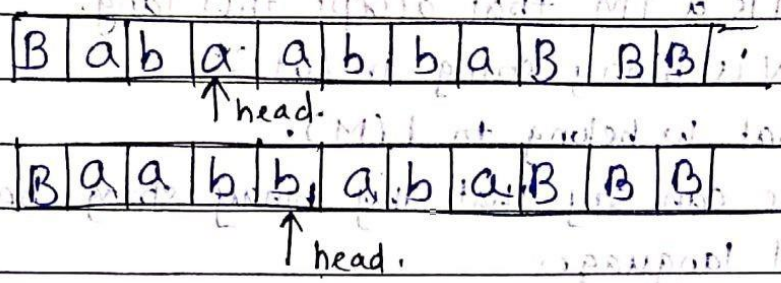
→ A TM with single tape can have multiple heads. Let's consider TM with two head H1 & H2. Each head is capable of performing read/write operation

B a a b a a b a BB
↑H2 ↑H2 ← TM with two head

3] Multi-tape Turing Machine :-

→ - Multi-tape turing m/c has multiple ~~tape~~ tuples with each tape having it's own independent head.

- Let's consider case of two tape turing m/c as shown in fig.



Two Tape Turing Machine

- The transition behaviour of a two-tape turing m/c can be defined as given below.

$$\delta(q_1, a_1, a_2) = (q_2, (S_1, M_1), (S_2, M_2))$$

where,

q_1 is current state.
q_2 is next state.
a_1 symbol head on tape 1
a_2 symbol under head on tape 2.
S_1 is symbol written in current cell on tape 1.
S_2 is symbol written in current cell on tape 2.
M_1 is the movement (L, R, N) of head on tape 1
M_2 is the movement (L, R, N) of head on tape 2.

* Limitation of Turing Machine :-

→ 1) Computational Complexity theory :-
limitation is that they do not model strength of particular arrangement

2) Concurrency :-
limitation is that they do not model concurrency well.

3) There are always halting concurrent system with no i/p.

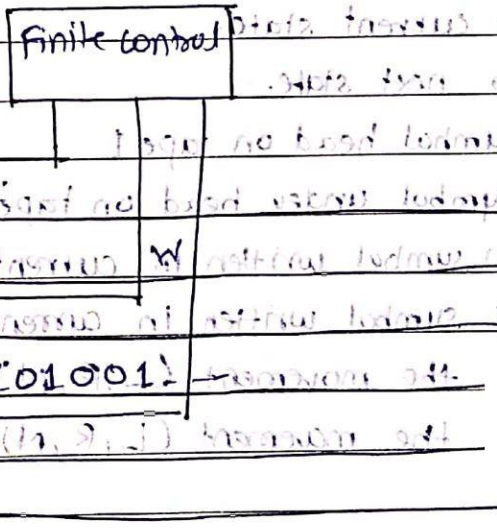
4) If two CFGs are given G_1 & G_2 then $L(G_1) \cap L(G_2) = \phi$ is undecidable

5) Recursively Enumerable lang. and the halting problem.

6) TM m/c is weak to describe the property the internet, evolution or robotics bcz it is closed model.

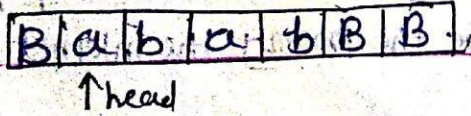
* Universal Turing Machine :-

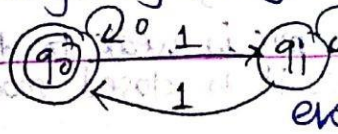
1) The universal lang. L_u is the set of binary string which can be modeled by turing m/c.
2) The universal lang can be represented by pair (M, w) where M is a TM that accept this lang.
   W is binary string in $(0+1)^*$
   such that w belong to L(M).
   Thus, we can say that any binary string belongs to universal language.
3) The universal lang. can be represented by $L_u = L(U)$ where U is universal Turing m/c.
4) In fact, U is binary string. This binary string represents various codes of many turing machine.
5) Thus, the universal turing m/c is a turing m/c which accepts many turing m/c.



* Language acceptability by turing machine :-

- TM accepts all lang even though there are recursively enumerable.
  - Recursive means repeating the same set of rule for any nos of times.
  & enumerable means list of elements.
- TM also accepts computable function, such as Addition, multiplication, substraction, division, power fn, square fn & logarithmic function.
- we can solve some e.g. for accepting lang using TM.

e.g. 



even nos of 1's.

* TM's Halting Problem :-

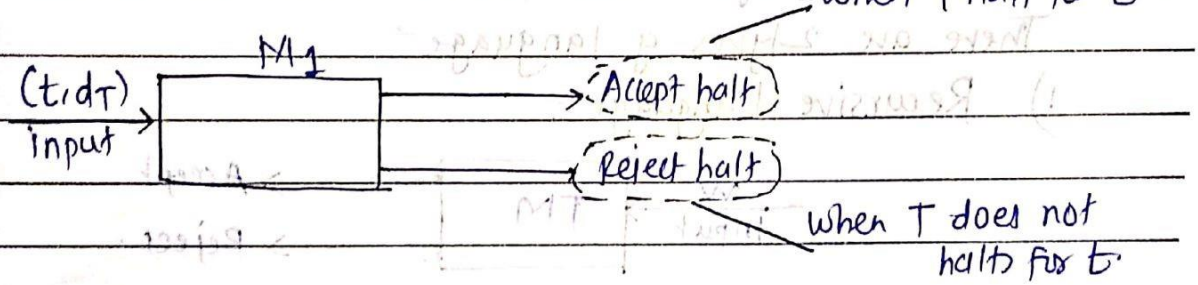
"Halting Problem is unsolvable"

proof :-

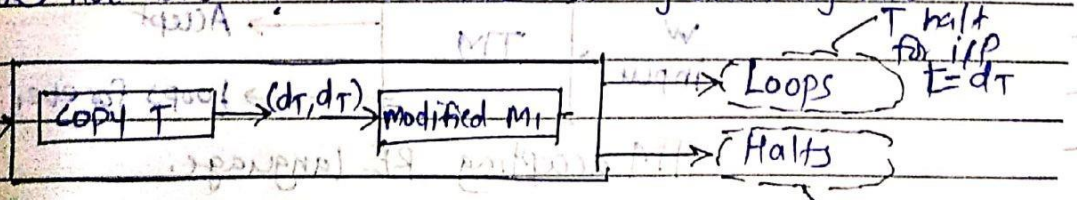
Let, there exist, a TM i.e. M_1 which decides whether or not any computation by a TM T will ever halt when a description of T and tape of T is given.

Then for every i/p (t, d_T) to M_1 if T halt for i/p t. M_1 also halts which is called accept halt.

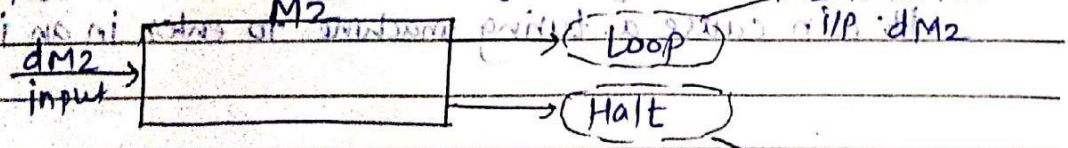
- similarly if T does not halt for i/p t then the M_1 will halt which is called reject state



When we will consider another TM M_2 which takes i/p d_T. It first copy d_T and duplicate d_T on it's tape and then this duplicate tape info. is given as i/p m/c M_1. But M_1 is modified m/c with the modification that whenever M_1 is supposed to reach as accept halt, M_2 loop forever. Hence behaviour of M_2 is given. It loops if T halts for i/p $t = d_T$ and halts if it does not halt for $T = d_T$. The T is any arbitrary m/c.



As M_2 itself is one TM. we will take $M_2 = T$, that means we will replace 'T' by 'M_2' from above given m/c.



This is contradiction, that means M_1 which can tell whether any other m/c TM will halt on particular i/p does not exits. Hence halting problem is solvable.

* TM and Type 0 Grammers:

→ TM can accept Type 0 grammar.

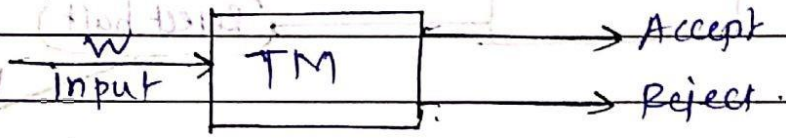
- The type 0 grammar is said to be unrestricted grammar e.g. restricted lang are almost all natural lang.
- The class of lang accepted by type 0 grammars are called recursively enumerable language.
- The prod can be in form $\alpha \rightarrow \beta$

where α is string of terminal and non-terminal with atleast one non-terminal and α cannot be null.

β is string of terminal & non-terminal.

There are 2 types of language:

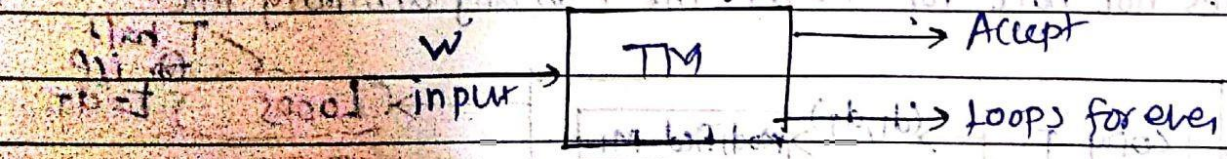
1) Recursive Language:



TM Accepting Recursive Language

A lang is said to be recursive if there exists a TM that accept every string of lang. and every string is rejected if it is not belonging to that lang.

2) Recursively Enumerable Language:



TM accepting RE language.

- A lang is said to be recursive if there exist TM that accepts every string belonging to that language. And if the string does not belong to that language then it can cause a tuning machine to enter in an infinite loop.

5. PUSHDOWN AUTOMATA:

* Introduction to Pushdown Automata (PDA)

→ Pushdown Automata (PDA) can be viewed as a finite automata with stacks.

An added stack provides memory and increase capability of machine.

A pushdown Automata can do the following.

1. Read i/p symbol
2. Perform stack operation
   - push oper
   - POP oper
   - check empty condition of a stack thro. initial stack symbol
   - Read top symbol of stack without a pop.
3. make state change.

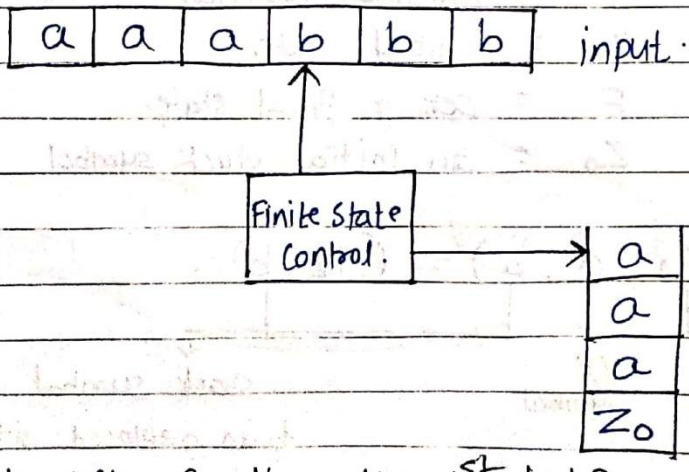
- PDA is more powerful than FA.
- A context free lang (CFL) can be recognized by PDA
- A Context free lang is a PDA lang.

Example :

A string of the form $a^n b^n$ can't be handled by FA. But same can be handled by PDA.

$n = 3$ $a^3 b^3$

a a a b b b



Stacks After Reading the 1st half of $a^3 b^3$

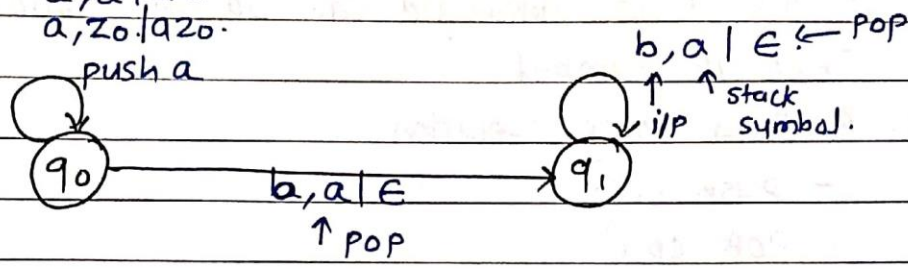
A PDA uses three stack operation :-

1) POP operation, - it removes the top symbol of stack.

2) PUSH operation - it insert symbol onto the top of stack.

3) NOP operation - it does nothing to stacks.

The Language $\{a^n b^n \mid n \geq 1\}$ can be accepted by PDA.



PDA for $a^n b^n$.

* Definition of PDA.

→ A PDA can be defined as 7 tuple.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where,

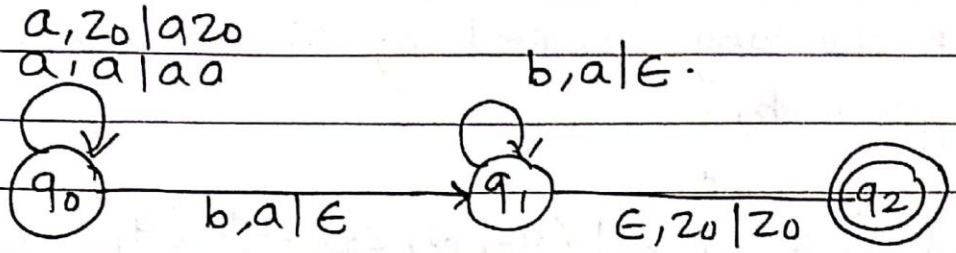
Q = set of state
Σ = i/p alphabets
Γ = stack symbol
δ = transition function
q_0 = Initial state.
F = set of final state.
Z_0 = an initial stack symbol.

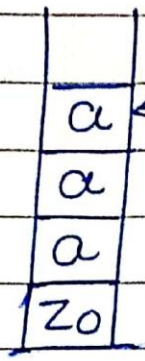
1) $\delta(q_1, a, b) = (q_2, b)$

current state — q_1
i/p symbol — a
stack symbol — b is replaced with b.
i.e. no stack operation.

* Example :-

$a^n b^n \mid n \geq 1$ aaabbb.





Transition Diagram

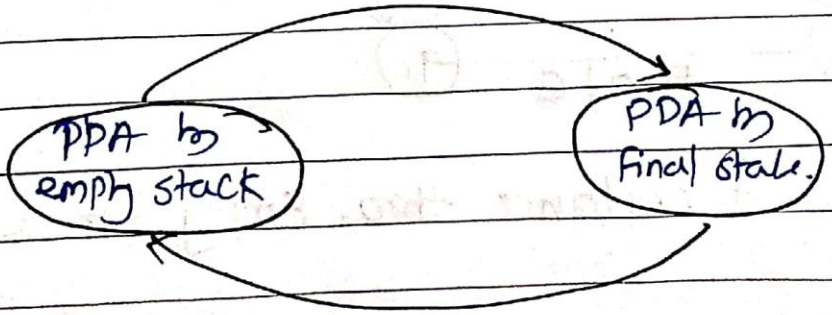
Using set of eq^n :- (Transition Rules)

1) $\delta(q_0, a, Z_0) = (q_0, aZ_0)$
2) $\delta(q_0, a, a) = (q_0, aa)$
3) $\delta(q_0, b, a) = (q_1, \epsilon)$
4) $\delta(q_1, b, a) = (q_1, \epsilon)$
5) $\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

* The Language of PDA :-

→ A Lang, L can be accepted by a PDA in two ways
1) Through final state
2) Through empty stacks.

It is possible to convert bet^n two clases
1) From final state to empty stack.
2) From empty stack to final state.

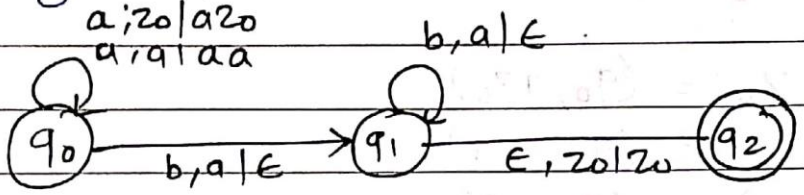


1) Acceptance by final state :-

Let the PDA, $M = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$.
then the lang accepted by M through a final state is given by:

$$L(M) = \left\{ w \mid (q_0, w, Z_0) \overset{*}{\underset{M}{\Rightarrow}} (q_1, \epsilon, \alpha) \right\}$$

where the state $q_1 \in F$.. α, the final contents of the stack are irrelevant. as a string accepted through a final state.



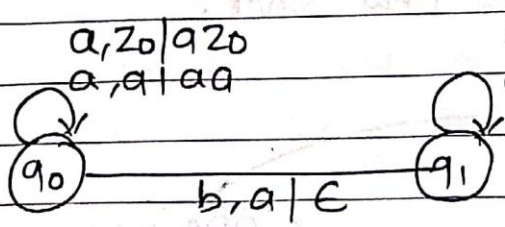
Acceptance thro. final state.

2) Acceptance by Empty stack :-

Let the PDA, $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \phi\}$.
then the lang accepted through an empty stack is given by:

$$L(M) = \left\{ w \mid (q_0, w, Z_0) \overset{*}{\underset{M}{\Rightarrow}} (q_1, \epsilon, \epsilon) \right\}$$

where, q_1 is any state belonging to Q. and the stack becomes empty on application of i/p string w.



Acceptance thro. Empty stack.

Note :-

1) Final state having more state	1) Empty stack having less state.
2) more ~~less~~ time required	2) less time required.

* Non-deterministic PDA :-

→ There are 2 types of PDA
1) Deterministic PDA (DPDA)
2) Non-deterministic PDA (NPDA)

- In a DPDA, there is only one moves in every situat whereas in NPDA, there are multiple moves under situatio
- DPDA is less powerful than NPDA
- Every context free lang. cannot be recognized by a DPDA but it can be recognised by NPDA.
- The class of lang. a DPDA can accept lies in between a regular lang. and CFL.
- A palindrom can be accepted by NPDA but it cannot be accepted by a DPDA.

* Application of PDA :-

→ 1) PDA is machine for CFL
2) A string belonging to CFL can be recognized by PDA.
3) PDA is extensively used for parsing.
4) PDA is an abstract machine, it can also be used for giving proofs of lemma on CFL.
5) PDA can write symbol on stack.
6) PDA can Push or Pop symbol on stacks

UNIT-VI

Undecidability & Intractable Problem.

* Recursively Enumerable and Recursive Language :

→ 1) A language is called recursively enumerable if and only if the language is accepted by some turing Machine M.

2) In other word, L is said to be L = L(M) for some TM M. There are certain language is not recursively enumerable.

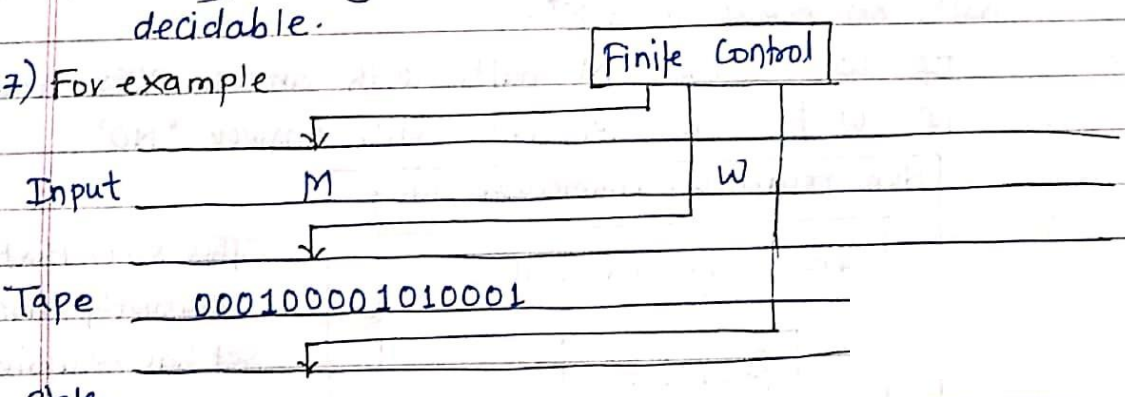
3) Consider language consisting a pair (M, w) where, 1) M, a Turing Machine with i/p set (0, 1)
2) w, a binary string consisting of 0's and 1's.
3) M accept w.

4) Following statements are equivalent
- The lang. L is turing acceptable
- The lang L is recursively enumerable.
- The lang L is recursive.
- The lang L is Turing Decidable.

5) Every turing decidable lang is Turing acceptable.

6) Every turing acceptable lang need not be turing decidable.

7) For example



8) - The universal lang can be represented by (M, w) where M is TM accepted lang, & w is binary string $(0+1)^*$.

9) -The universal Turing m/c U accepts the I/P M.
-The transition g M are stored initially on first tape along with string w.

10) on the 2nd tape, the simulated tape g M is placed. Here the tape symbol X_i g M will be represented by 0^i and tape symbol are seperated by single 1's.

* Turing acceptable language :

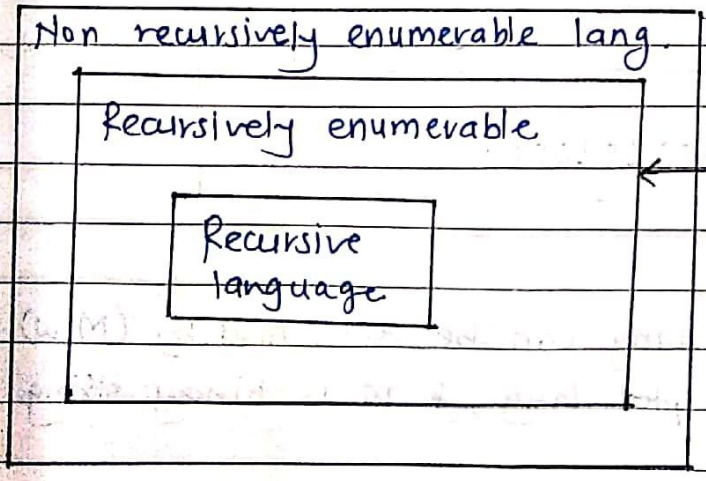
→ NOTE : you can write same answer as above question just add definition g turing acceptable.

" A language $L \subseteq \Sigma^*$ is said to be turing being acceptable lang. if there is turing Machine M. which halts on every $w \in L$ with an answer 'YES' However, if $w \notin L$, then M may not halt.

* Turing decidable / Solvable :

→ NOTE : Same answer as above. Just add definition g it.

" The Language $L \subseteq \Sigma^*$ is said to be turing being decidable if there is turing machine M which always halts on every $w \in \Sigma^*$.
If $w \in L$ then M halts with answer 'YES' and if $w \notin L$ then M halts with answer 'NO'



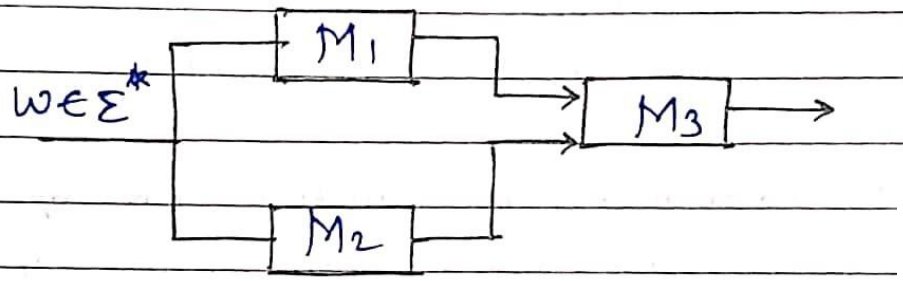
This shaw that every recursively enumerable set has recursive Subset.

* Show that for two recursive languages L_1 and L_2 each of the following is recursive.
i) $L_1 \cup L_2$ ii) $L_1 \cap L_2$ iii) L_1'

→

1) $L_1 \cup L_2$ is recursive ?

→ Let the TM M_1 decides L_1 and M_2 decides L_2.
- If a word $w \in L_1$ then M_1 returns 'Yes' else. it returns 'No'
Similarly if word $w \in L_2$ then M_2 return 'Yes' else 'No'
- Let us construct TM M_3 as shown in figure



A Turing Machine for $L_1 \cup L_2$

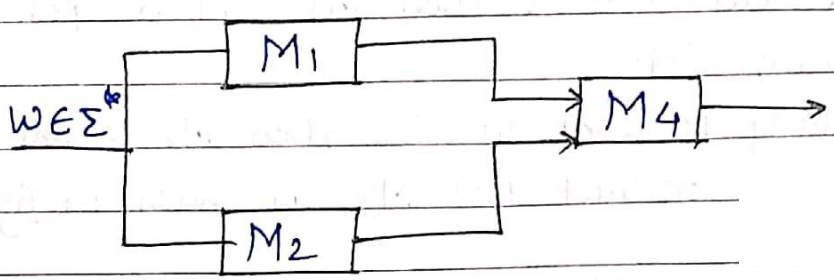
- output of machine M_1 is written on the tape of M_3
- output of m/c M_2 is written on the tape of M_3
- The m/c M_3 returns 'Yes' as o/p, if atleast One of the o/p of M_1 or M_2 is 'Yes'.

- It should be clear that M_3 decides $L_1 \cup L_2$. As both L_1 & L_2 are turing decidable. after finite time both M_1 & M_2 will halt with answer 'Yes' or 'No'
- The machine M_3 gets activated after M_1 or M_2 is halted. (stopped)
- The machine M_3 halts with answer 'Yes' if $w \in L_1$ or $w \in L_2$ else M_3 halts with o/p 'No'

Thus, $L_1 \cup L_2$ is turing decidable or Recursive.

ii) $L_1 \cap L_2$ is recursive ?

→ - Let the turing machine M_1 decides L_1 and M_2 decides L_2. If word $w \in L$ then M_1 return 'Yes' else it returns 'No'
Similarly if $w \in L_2$ then M_2 returns 'Yes' else 'No'
- Let's consider TM M_4 as shown in fig.

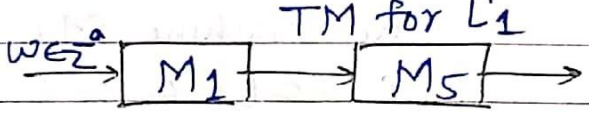


A Turing Machine for $L_1 \cap L_2$

- o/p of machine M_1 is written on the tape of M_4.
- o/p of m/c M_2 is written on the tape of M_4.
- The m/c M_4 returns 'Yes' as o/p, If both M_1 and M_2 are "Yes" otherwise M_4 returns 'No'
- It should be clear that M_4 decides $L_1 \cap L_2$. As both L_1 & L_2 are turing decidable, after finite time both M_1 & M_2 ~~are~~ will halt with answer 'Yes'/'No'.
- The m/c M_4 is activated after M_1 & M_2 are halted. The m/c M_4 halts with answer 'yes' if $w \in L_1$ & $w \in L_2$ else M_4 halts with answer 'No'.

iii) L_1' is recursive. ?

→ Let the TM ~~decides~~ M_1 decides L_1
- Let's construct TM M_5



- o/p of machine M_1 is written on the tapes of M_5.
- The m/c M_5 returns "Yes" as o/p if the o/p of M_1 is "No" otherwise. M_5 returns "No".
- It should be clear that M_5 decides L_1'. As L_1 is turing decidable after finite time M_1 will halt with answer Yes/No.
- The m/c M_5 is activated after M_1 halts.

* Undecidability ÷ (Unsolvable)

→ " A Problem is said to be decidable if there exist a Turing machine that gives correct answer for every statement in the domain of problem otherwise, the class of problem is said to be un-decidable "

- These two statements are equivalent.
1) A class of problem is Un-decidable
2) A class of problem is Un-solvable

- A language can be proved to be un-decidable through a method of reduction.
- As we have already seen that halting Problem is Undecidable.
- Some standard Un-decidable problem.
1) Halting problem of Turing M/C.
2) Diagonalization lang.
3) The Post Correspondance Problem.
4) The Universal lang.

1) Halting Problem ÷

→ NOTE ÷ 1) Already written answer in notes of Turing Machine.
2) write above theory & explain any 1 standard.

* Un-decidability of Post Correspondence ÷

→ " Let A and B be two non-empty list of string over Σ.
A and B are given below.

$A = \{x_1, x_2, x_3 \dots x_n\}$

$B = \{y_1, y_2, y_3 \dots y_n\}$

We say, there is post correspondence between A & B. if there is a sequence of one/more integer i, j, k ... m such that
The string $x_i, x_j \dots x_m$ is equal to $y_i y_j \dots y_m$

Post Correspondence Problem :-

example :-

$A = \{a_1, aba^3_2, ab_3\}$ and $B = \{a^3_1, ab_2, b_3\}$

We will have to find a sequence using which when the elements of A and B are listed, it will produce identical string.

The required sequence is (2, 1, 1, 3)

$A_2\ A_1\ A_1\ A_3 = \{aba^3\ a\ a\ ab\} = aba^6b$

$B_2\ B_1\ B_1\ B_3 = \{ab\ a^3\ a^3\ b\} = aba^6b.$

We are accepting the Un-decidability of post Correspondence Problem.

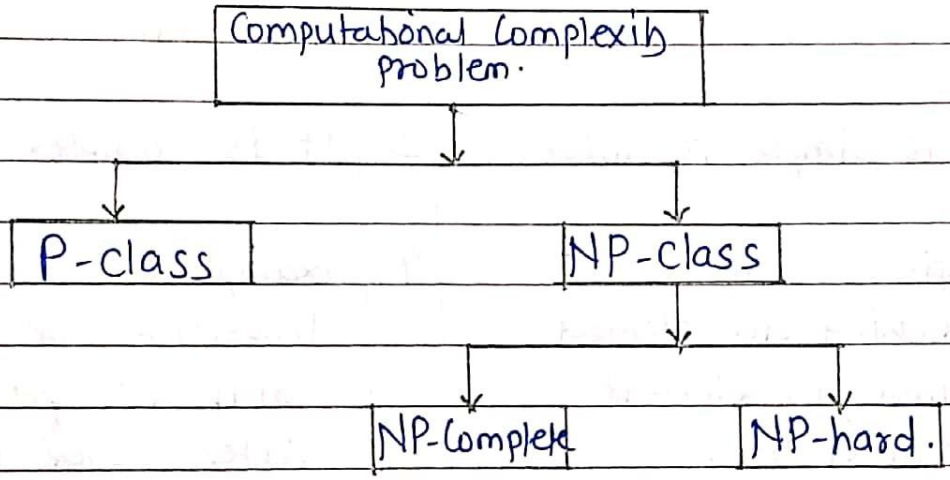
* The classes P and NP.

P → "The class of problem P is denoted by P are solvable by a deterministic Turing m/c in polynomial time."

NP → "The class of problem denoted by NP are solvable by Non-deterministic Turing m/c polynomial time."

- P Problem are feasible or theoretically not difficult to solve by Computational means.
- The distinguish features of problem is that for each instance of any of these problem, there exists deterministic turing m/c that solves the Problem having time complexity as polynomial function of the size the Problem.
- P stands for polynomial
- NP stand for Non-deterministic Polynomial.

- There are two group in which Problem can be classified.
the 1st Problem that can be solved in polynomial time.
e.g. Searching element
Sorting element.
- The 2nd problem that can be solved in non-deterministic polynomial time
e.g. Knapsack Problem o
Travelling salesman Problem.



* P Problem Example :-
→ 1) Searching element from list
2) Sorting element
3) Minimal Spanning tree.
4) Prim's Algorithm
5) Kruskal Algorithm

* NP-Complete Problem Example :-
→ 1) Traveling Salesman Problem (TSP)
2) Vertex Cover Problem (VCP)
3) Hamiltonia Circuit Problem (HCP)
4) Satisfiability Problem (SAT)
5) Exact Cover Problem

* Difference bet^{n} P class & NP class problem

Sr No	P Class	Sr No	NP-class problem
1.	P. stands for Polynomial time (deterministic)	1.	NP stands for Non-deterministic Polynomial.
2.	Problem that can be solved in polynomial time are called as P-class	2.	Problem that can be solved in non-deterministic Polynomial time are called as NP-class.
3.	It is simple to solve.	3.	It is complex to solve.
4.	Example: Searching an element Sorting an element. Spanning tree.	4.	Example: Travelling Salesman Problem. Knapsack problem Vertex Cover Problem.
5	P class is tractable	5	NP class is Intractable

* Polynomial time Reduction:

→ "A polynomial time reduction is a Polynomial-time algo. which construct the instance of a problem P_2 from the instance of some other problem P_1."

- A Problem P_1 equivalently represents a lang L_1.
- we says that Problem P_2 can be solved in polynomial time if we can reduce, another problem P_1, which is known not be in P to P_2.
- Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be language, A polynomial time computable fu^{n} $f: \Sigma_1^* \to \Sigma_2^*$ is called Polynomial-time reduction from L_1 to L_2 if and only if. for each $x \in L_1$, $f(x) \in L_2$.

* Polynomial time Reduction :-

- To Prove whether Particular Problem is NP complete or not, we use Polynomial time reducibility. that means if.

$A \xrightarrow{poly} B$ and $B \xrightarrow{poly} C$ then $A \xrightarrow{poly} C$

- The reduction is imp task in NP Completenen Proof.
- Various times of reduction :-

1) Local Replacement :-

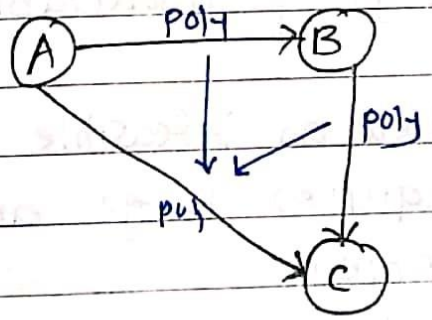
In this reduction, $A \rightarrow B$ is dividing i/p to A in the form of components and then these component can be. Converted to Components of B.

2) Component design :-

In this reduction. $A \rightarrow B$ by building special Component for input B that enforce Properties required by A.

The reduction can be denoted by $A \leq_T^P B$.

Example :-



If $f(A \rightarrow B)$ and $f(B \rightarrow C)$ then $(A \rightarrow C)$ this is Polynomial time Reduction.

* Tractable and Intractable :-

→ "Tractable Problems are the class of problem that can be solved within reasonable time & space"

for example :-

- Searching a key from list
- Sorting a list

- These algo takes $O(n \log n)$, $O(n)$, $O(n)^2$ time Complexity.
- We normally expect that tractable problem to be solved in Polynomial time.
- It is also called as P-Problem.

"Intractable Problem are the class of problem that can be solved within Polynomial time."

- This leads to two classes of solving Problem - P class and NP-class.
- The lower bound of these algo. take exponential time complexities.

→ For example,

Tower of Honoi is e.g. of Intractable Problem.

- Intractable is also called as infeasible Problem.
- Intractable Problem requires large amount of resources to solve Problem.

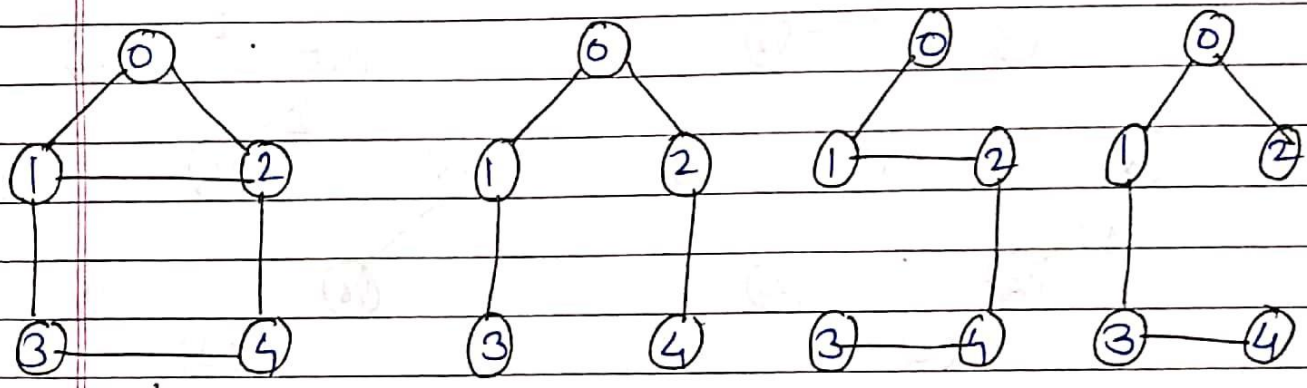
Hence, It is infeasible.

* Example of Polynomial time :-
* Kruskal Algorithm :-

→ A spanning tree of a graph $G=(V,E)$ is a connected subgraph of G having all vertices of G and no cycle in it.

If graph G is not connected then there is no spanning tree of G.

A graph may have multiple spanning tree.



Sample connected graph.

Spanning tree of graph.

- There are two types of spanning tree
  1) Prims algo.
  2) Kruskal's algo.

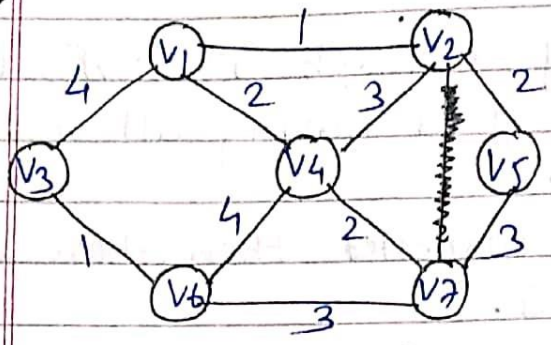
* Kruskal's algorithm :-

- It is one of the method for finding the minimum cost of spanning tree of the given graph.
- In kruskal algo., edges are added to spanning tree in increasing order of cost.
- If any selected edge forms cycle in spanning tree, then it is discarded.

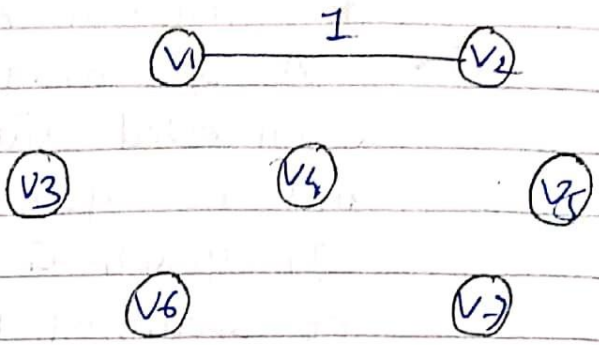
Algo :- 1) Arrang the edges of Graph G in ascending order of weight
2) Let $G=(V,E)$ has n vertices. construct min spanning tree
$G_T = (V_T, E_T)$ $\therefore V_T = V$ and $E_T = \{\ \}$
3) for every edges e_i in $(e_1, e_2 \ldots e_k)$
if e_i does not form a cycle in G_T then $E_T = E_T \cup \{e_i\}$

Example of Kruskal algo :-

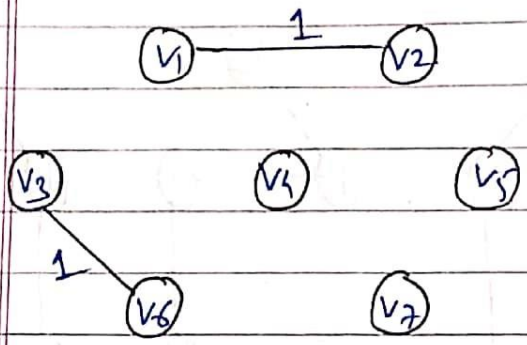
→



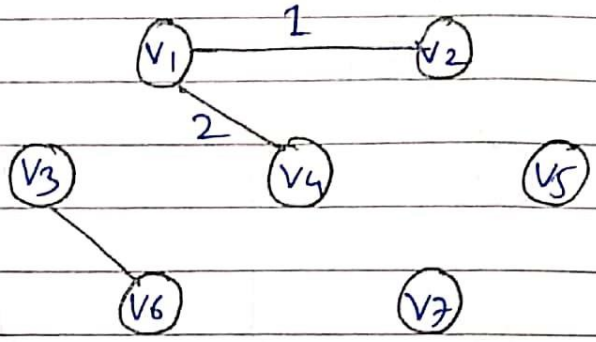
(a) Given Graph G



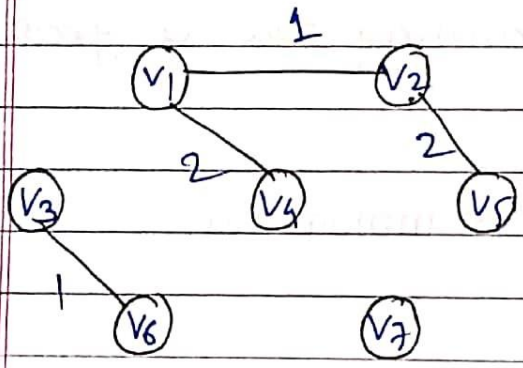
(b)



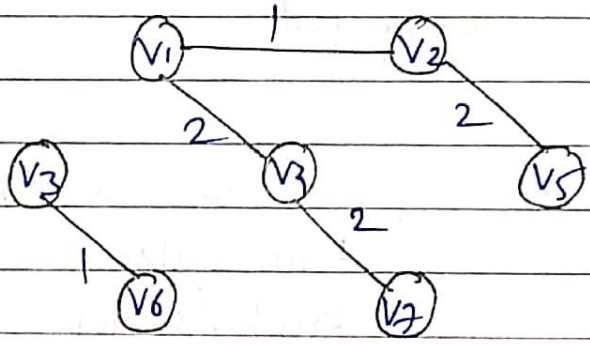
(c)



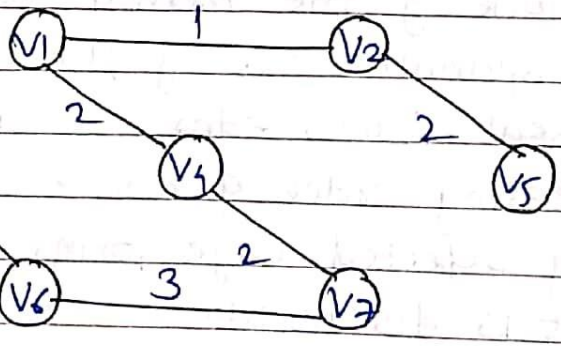
(d)



(e)



(f)



(g)

(g)

* Kruskal algo. using Turing Machine (TM) :-

→ Kruskal algo is implemented using multitape TM.
- Edes g minimum weight is selected to connect two component.
- Initially, every node is in it's a component by itself.

1) One tape g TM can be used to store every node with its current component

2) A tape can be used for finding least edge weight. among the edges in which have not been used in the spanning tree.

3) when an edges is selected, its two vertices are. copied on the tape. then we look for the component g the two vertices.

4) If two component (i and j) found in the Previous. step are not the same component. then they can be merged into single component with help g another tape.

Using abave algo, we can find minimum spanning tree in n round.

Thus, multiple tape TM will require $O(n)^2$ and given problem is in P-Problem.

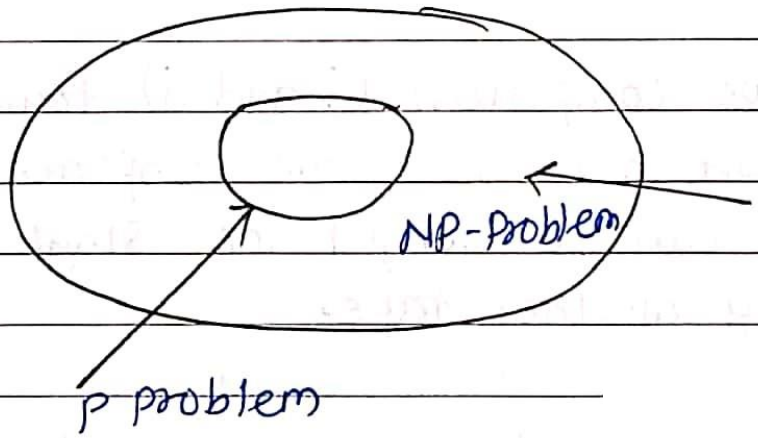
* NP Complete Problem :-

→ "A Problem P is said to be NP complete if the following two condition are satisfied.

1. The Problem L2 is in the class NP.
2. for any problem L2 in NP, there is Polynomial time reduction of L1 to L2.

- A Problem is NP-complete if it is in NP and for which no Polynomial time. Deterministic TM solution is known so far.

- NP denotes Non-deterministics Polynomial language Problem.
  Hence $P \subseteq NP$.
- NP Problem is also called as Intractable Problem.



- Example of NP Complete Problem :-

1) Travelling salesman Problem (TSP)
2) Vertex Cover Problem (VCP)
3) Hamiltonian Circuit Problem (HCP)
4) Satisfiability Problem (in short SAT)
5) Exact Cover Problem.

Example of NP-Complete :-

1) Satisfiability Problem (SAT)

→ "The satisfiability Problem is :

"Given a Boolean expression, is it satisfiable?

"A Boolean expression is said to be satisfiable if at least one truth assignment makes the boolean expression true."

The Boolean expr are created using.

1) The variable whose value can be 0 or 1.
2) The operator that can be used in expression can be $\vee$, $\wedge$. The $\vee$ mean OR and $\wedge$ means AND operator.
3) Unary operator $\neg$ stands for negation.
4) The Parenthesis are used to group the operand & operators in the expression

The highest Precedence is to $\neg$, then $\wedge$ then $\vee$.

for example :-

The boolean expr is $(x \wedge y) \vee (y \wedge z)$.

If $(y \wedge z)$ and $x \wedge y$ both are true then boolean expr is true if $(x \wedge y)$ and $(y \wedge z)$ both are false then boolean expr. is false.

The boolean expression $((x_1 \wedge x_2) \vee \neg x_3)$ is true.

for $x_1 = 1$, $x_2 = 0$ and $x_3 = 0$

Therefore $((x_1 \wedge x_2) \vee \neg x_3)$ is satisfiable.

- $(x \wedge y \wedge z)$ is satisfiable when $x = 1$, $y = 1$ & $z = 1$

But

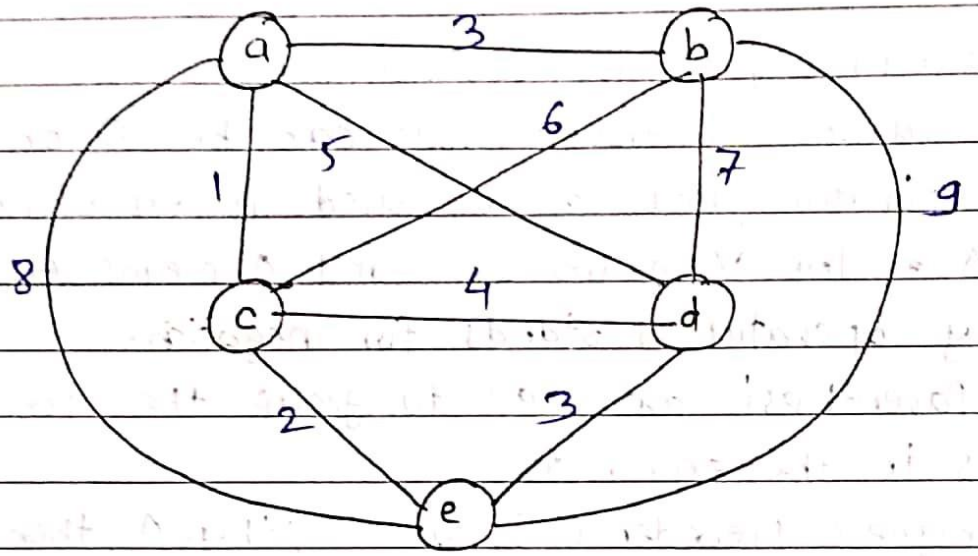
$x \wedge (\neg y)$ is not satisfiable.

2) Travelling Salesman Problem (TSP) :-

→ This Problem can be stated as

"Given set of cities and cost to travel between each pair of cities, determine whether there is path that visit every city once and return to the first city, such that cost travelled is less."

for example,



- The tour will be a-b-d-e-c-a and total cost of tour will be 16.
- This Problem is NP Problem as there may exist some path with least (shortest) distance between the cities.
- If you get the solution by applying certain algorithms then travelling salesman Problem is NP-Complete Problem.
- If we get no solution at all by applying an algorithm then the travelling salesman Problem belong to NP-Hard class Problem.

* Node-Cover Problem :

→ " A node cover Problem is to find node cover of minimum size in a given graph.

- The word node cover means each node covers its incident edges.
- Thus by node cover, we expect to choose the set of vertices which cover all the edges in a graph."

- A node cover undirected graph $G = (V, E)$ is a subset V' of the vertices of graph which contain at least one of the two endpoint of each edge.

- The node cover problem is the optimization problem of finding a node cover of minimum size in a graph.
- This problem can be stated as Decision Problem.

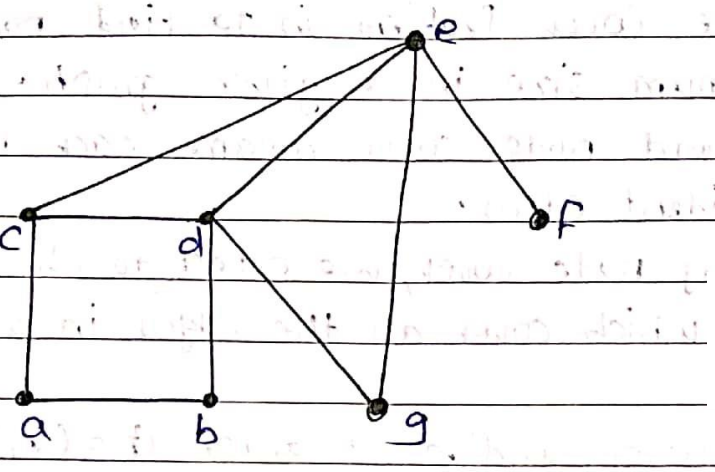
NODE-COVER = { $\langle G, K \rangle$ graph G has vertex cover of size K }.

- To show that, node cover Problem $\in$ NP, for a given graph $G = (V, E)$, we take $V_1 \subseteq V$ and verifies to see if it form node cover. verification can be done by ~~using~~ checking for each edge $(u, v) \in E$, whether $v \in V_1$ or $u \in V_1$.
- This verification can be done in Polynomial time.

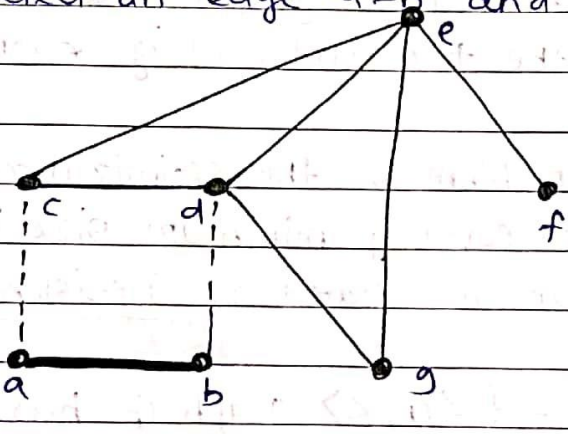
Example :

Consider a graph G as given below.
Now we will select some arbitory edge and delete all the incident edges.
repeat this proces until all the identical edges get deleted.

* Example Node - Cover Problem :-

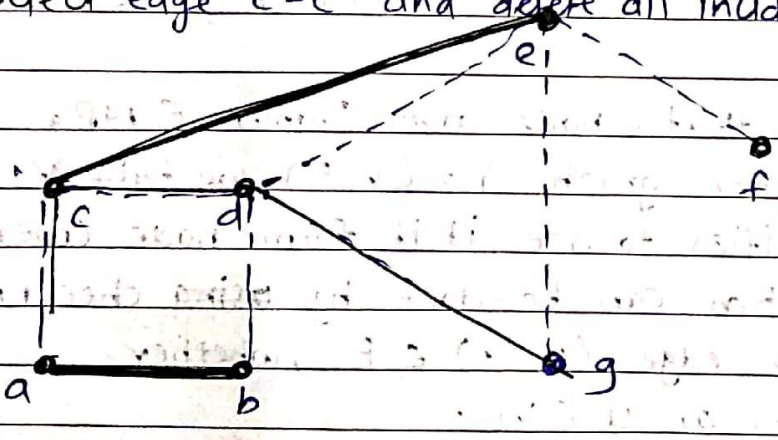


Step 1 :- select an edge a-b and delete incident edges



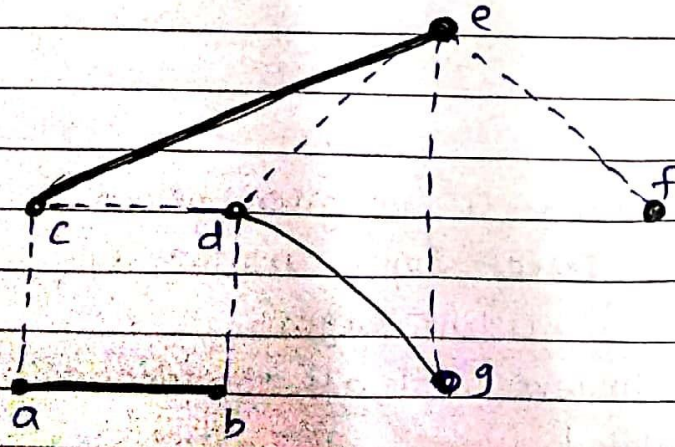
$W = (a, b)$

Step 2 :- select edge c-e and delete all incident edges



$W = (a, b, c, e)$

Step 3 :- select an edge d-g. All incident edges are already deleted



$W = \{a, b, c, e, d, g\}$

Thus, we obtain node cover as $\{a, b, c, d, e, g\}$.