② Memory "dependent on" the size of input.
write it down on tape

③ use multiple tracks if convenient

④ use multiple tapes if needed.

<u>Exercise:</u>

· Design a TM to compute

① sum of 2 numbers

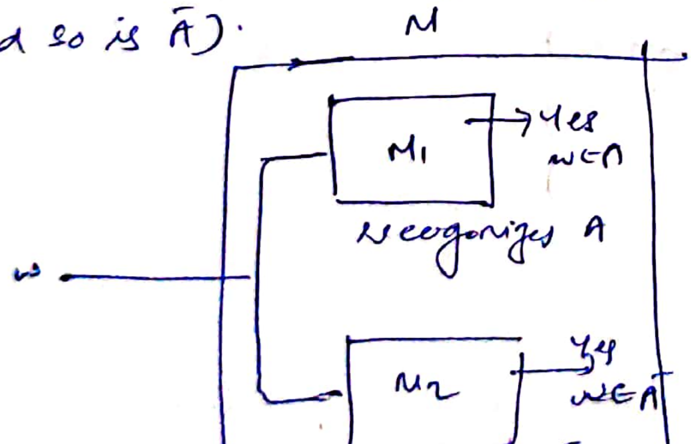
② square of a number

③ Find max of 2 numbers

<u>Theorem:</u> Any TM with an arbitrary set of tape alphabet Γ and set of input alphabet $\{0,1\}$, there is an equivalent turing machine that uses $\Gamma' = \{0,1,\sqcup\}$.

→ If A is turing decidable, then $\bar{A}$ is also turing decidable.
we ~~get~~ can get a TM M' that decides $\bar{A}$ by switching q_{accept} and q_{reject} states

→ If A and $\bar{A}$ are turing recognizable, then A is turing decidable (and so is $\bar{A}$).



M_1's yes = M's q_{accept}

M_2's yes = M's q_{reject}.

→ If A_1 and A_2 is turing decidable.

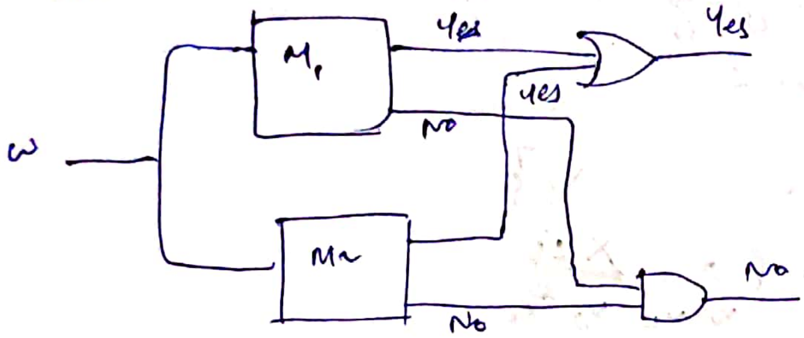
① Union $A_1 \cup A_2$

② Concatenation $A_1 \cdot A_2$ ($w_1 w_2$ are strings in $A_1 \cdot A_2$) $w_1 \in A_1, w_2 \in A_2$

③ A_1^* (any no of w, where $w \in A_1$) w^0(empty string), w, w^2(ww)......

④ $A_1 \cap A_2$ intersection .

<u>$A_1 \cup A_2$</u>



<u>concatenation ($A_1 \cdot A_2$)</u> (decidable)

$w = \boxed{10|1000} \in A_1 \cdot A_2$

→ If A_1 and A_2 are TM recogonizable

① $A_1 \cup A_2$

② $A_1 \cap A_2$

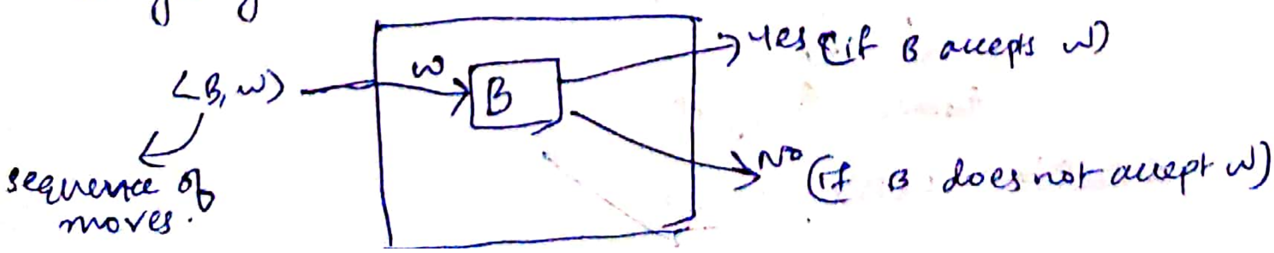
③ $A_1 \cdot A_2$

④ A_1^* (w^n, where $n \geq 0$)

are all TM recogonizable.

<u>Decidable languages.</u>

Language $\langle B, w \rangle$ where B is a DFA that accepts w.



Theorem: L_{DFA} is Turing decidable (Recursive)

Algorithm:

Run DFA B with w as input.

If B accepts w, say YES (goto accept), else say No (go to reject).

Q: $L_{TM} = \{ \langle M, w \rangle : M$ accepts string $w \}$. Is it decidable?

L_{TM} is undecidable.

We will construct a language in such a way that it is not Turing recogonizable.

Tape ~~Input~~ alphabet: $\{x_1, x_2, x_3 \dots x_n\}$

~~Tape alphabet:~~ eg: $\{0, 1, \cancel{X}, \cancel{+}\}$
$x_1, x_2, x_3 \quad x_4$

Order the states also: $\{q_1, q_2 \dots q_m\}$
(m states)

$q_1 \downarrow$ start state, $q_2 \downarrow$ accept state

Proof outline

1. We number every TM
2. We number every string w.
3. We define a language in such a way that there is no TM that recognizes that language (By 'diagonalization technique).

Tape alphabet: $x_1, x_2, \dots x_n$

States: $q_1, q_2 \dots q_m$

$q_1 \downarrow q_{start}$, $q_2 \downarrow q_{reject}$

direction: $L (D_1)$, $R (D_2)$

Scanned by CamScanner

$\delta(q_0, 1) = (q_1, 1, R)$ $\delta(q_1, 1) = (q_3, 1, R)$

$\delta(q_0, 0) = (q_0, x, R)$ $\delta(q_1, 0) = (q_1, x, R)$

$\delta(q_3, 0) = (q_1, 1, R)$ $\delta(q_4, 0) = (q_3, 1, R)$

$\delta(q_1, 1) = (q_3, x, L)$ $\delta(q_3, 1) = (q_4, x, L)$

$\delta(q_1, 0) = (q_{accept}, 0, R)$ $\delta(q_3, 0) = (q_2, 0, R)$

$q_0, q_{accept}, q_1, q_3$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$

$q_1 \quad q_2 \quad q_3 \quad q_4$

write down the machine code as a sequence of moves (transition function moves) of a given TM.

"Encode" the turing machine

$\underbrace{\underbrace{\langle move1\rangle}_{binary} \,11\, \langle move2\rangle \,11\, \ldots \langle move5\rangle}_{\text{'binary code' of TM}}$

'binary code' of TM ~~(need not be unique)~~

while numbering ('encoding') a TM, Note that a TM M does not have a unique number (unique encoding).

But a given number represents exactly one TM (or No turing machine)

$\delta(q_1, 1) = (q_3, 1, R)$

$\delta(q_1, x_2) = (q_3, x_2, D_2)$

$\langle 1, 2, 3, 2, 2\rangle$

move1: 0100100010010 0

If move is of the form $\delta(q_i, x_j) = (q_k, x_l, D_m)$

$0^i 1 0^j 1 0^k 1 0^l 1 0^m$

① For every TM, we can find a number

② Every number corresponds to exactly one or zero TM.

~~order~~

0	1
1	2
00	3
01	4
10	5
11	6
000	7
.	.
.	.
111	.

↖ ordering of strings.

$w_5 \Rightarrow$ string "10"

	M_1	M_2	...	M_i	M_n
w_1	0				
w_2	0	- - -	- -	1	
.	0				
w_i	0				
w_n	0				
	.				

→ M_i accepts w_2

$\boxed{L_d = \{w_i : \text{entry at } (i,i) \text{ is } 0\}}$

→ $L_{TM} = \{\langle M, w\rangle : M \text{ accepts } w\}$

M ↓ Turing machine; w ↳ string

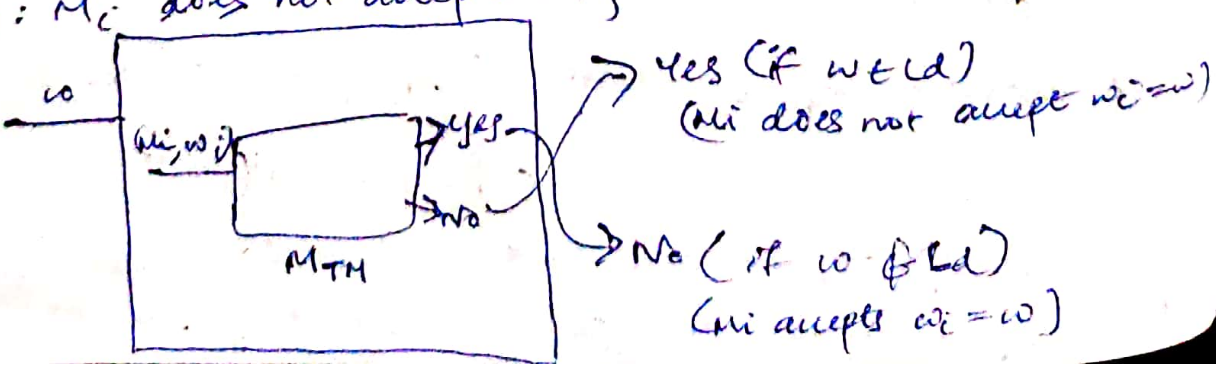
→ L_{TM} is turing recognizable (recursively enumerable)

<u>theorem</u>: L_{TM} is NOT Turing decidable (NOT recursive)

<u>Proof outline</u>: If L_{TM} is turing decidable, (If there is a TM M_{TM} that decides L_{TM}, then L_d is also decidable

$L_d = \{w_i : \text{entry}(i,i) \text{ is } 0\}$

* $L_d = \{w_i : M_i \text{ does not accept } w_i\}$



① Find i such that $w = w_i$

② give $\langle M_i, w_i \rangle$ as input to M_{TM}

③ - If M_{TM} says yes; answer No
NO; answer Yes.

$w_1 = 0$
$w_2 = 1$
$w_3 = 00$
$w_4 = 01$
$w_5 = 10$
$w_6 = 11$
$w_7 = 000$
$w_8 = 001$

q1 : given i, how to get w_i

q2 : Given w_i how to get i

w is given in terms of
① No of bits(k).
② value of string (n).

Eg: $i = 7$ ~~and k=3~~

$k = \lfloor \log_2(i+1) \rfloor = 3$

$\cancel{n \quad 0 \ldots 2^{K-1}}$

$n = i - (2^K - 1)$

$7 - (2^3 - 1) = 0$

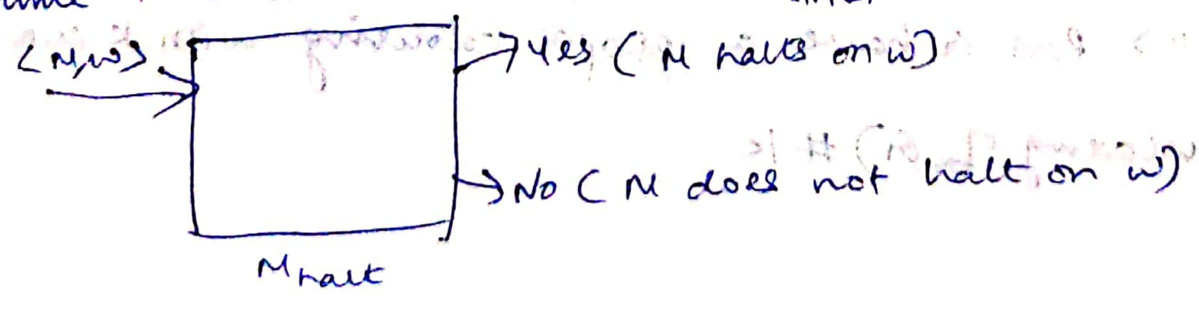
Eg: $k = 3$
$n = 5$
$i = 12$

$$\boxed{i = 2^K - 2 + (n+1)}$$

q: L_{HALT}: $\{ \langle M, w \rangle : M \text{ halts on } w \}$

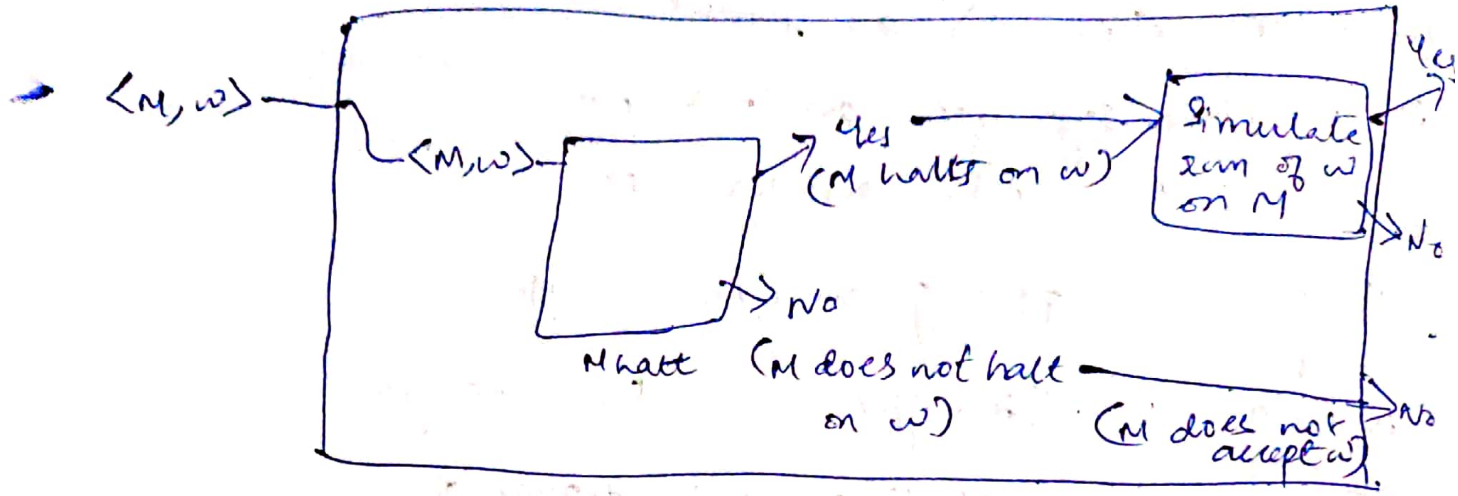
Theorem: HALTING PROBLEM is undecidable
(L_{HALT} is undecidable)

Proof:
Assume there exists a machine M_{HALT}



Scanned by CamScanner

Give ~~alego~~ algo for L_{TM}



① First run M halt with <M,w> as input

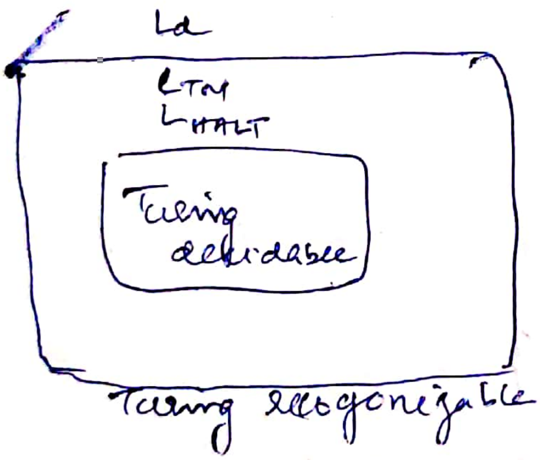
② if yes (iff M halt on w)

then run w on M; (it will surely say yes or No)

if yes → answer yes

if No → answer No

③ else No from M halt (means M does not halt on w)



→ A is turing decidable iff A and $\bar{A}$ are both turing decidable

Q: <G, k> Does G have a proper colouring with k colours?

(encoding of G) # k

Q: given encoding of graph G, check whether it is connected. ~~or not~~

$G = (\{1,2,3,4,5\}, \{(1,2),(3,3),(3,1),(1,4),(4,5)\})$

- $L_{NFA} = \{\langle A, w\rangle : A \text{ accepts } w\}$ is also decidable.

~~Convert this NFA to a DFA~~

we first convert A to an equivalent DFA A′, then we use the algorithm M_{DFA}.

(verify that final states are not connected to initial state in the transition graph)

- $L_{EMPTY\ DFA} = \{\langle A\rangle : L(A) = \phi\}$ DFA: A does not accept any string.
- $L_{EQ\text{-}DFA} : \{\langle A, B\rangle : L(A) = L(B)\}$ is decidable

construct $C = (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)})$

Run $L_{EMPTY.DFA}$ algo on C.

If C is empty

in $(L(C) = \phi)$

the ans is yes $(L(A) = L(B))$

else ans is no $(L(A) \neq L(B))$

Reducibility

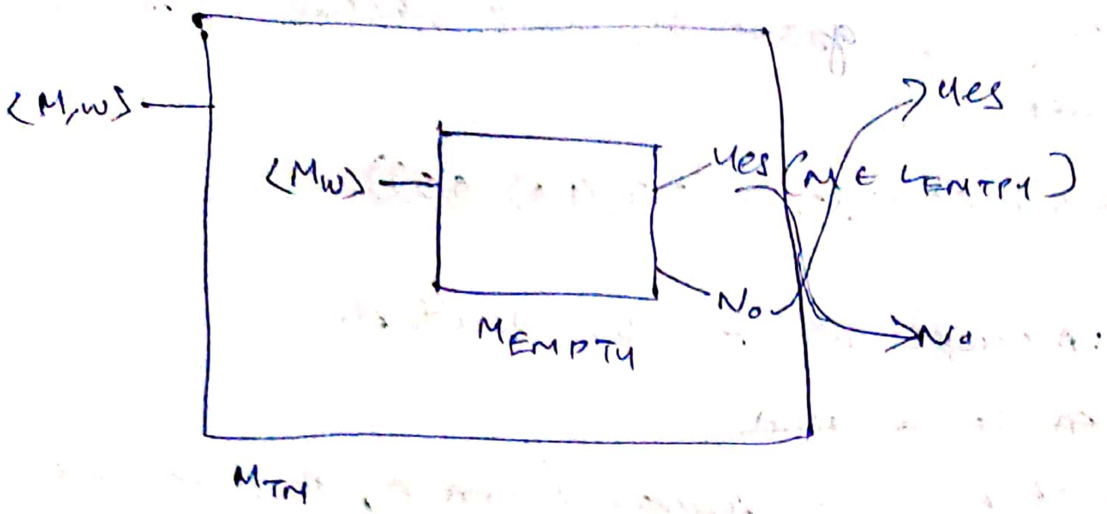
→ Reducing one problem to another

$L_{EMPTY} = \{\langle M\rangle : L(M) = \phi\}$

M does not accept any string.

Theorem: L_{EMPTY} is undecidable

Proof Idea: If L_{EMPTY} is decidable, (if we have a machine M_{EMPTY}, that checks if a given $M \in L_{EMPTY}$ or not), then we can design M_{TM} using this M_{EMPTY}



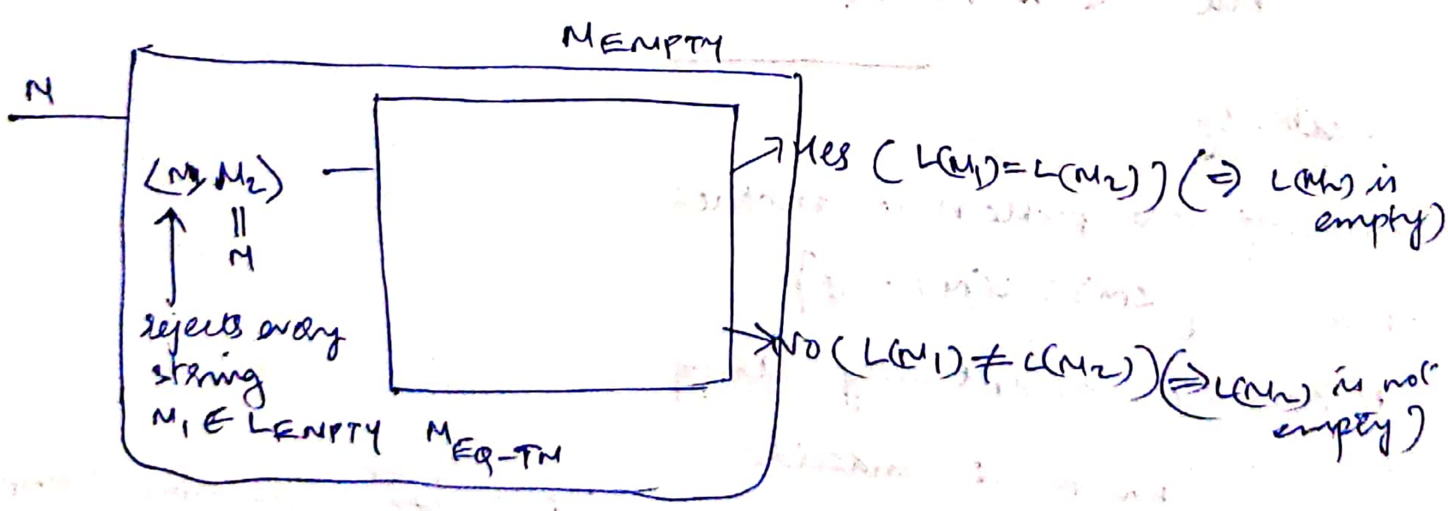
<u>Mw on ~~for~~ input x:</u>

if $x == w$ (if input is w)
    run M on i/p x
if $x \neq w$. (if input is not w)
    reject.

Language of Mw is nonempty $\equiv L(Mw) = \{w\}$
$\Rightarrow$ M accepts w.

Language of Mw is empty $\equiv L(Mw) = \phi$
$\Rightarrow$ M does not accept w.

$L_{EQ\text{-}TM} = \{\langle M_1, M_2\rangle : L(M_1) = L(M_2)\}$ is ~~also~~ undecidable



So if $M_{EQ\text{-}TM}$ exists M_{EMPTY} would exist. But ~~we~~ we have prooved M_{EMPTY} is not possible

Q: given a turing machine M, is the language recognized by M, L(M), regular?

$L_{REGULAR} = \{ \langle M \rangle : L(M) \text{ is regular} \}$

- M_1 accepts strings of form $0^n 1^n$
- M_2 " all strings with odd no of zeros.
- M_3 accepts the language $w\#w$.
- M_4 accepts all strings.

$M_2, M_4 \in L_{REGULAR}$

* "Pumping lemma" can be used to check if a language is regular.

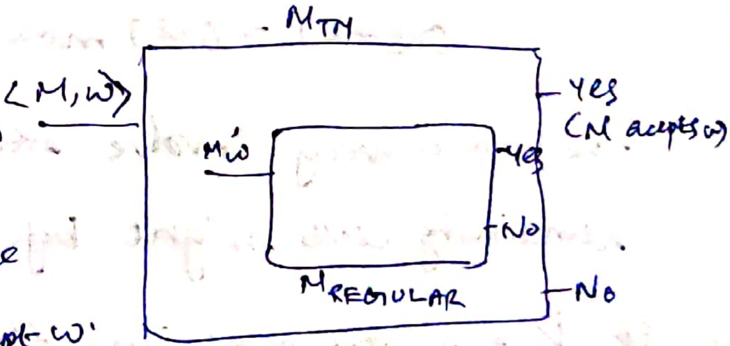
Show that $L_{REGULAR}$ is undecidable.

Proof idea: If $L_{REGULAR}$ is decidable, then L_{TM} is also decidable.

undecidable
L_{TM}
L_{HALT}
L_{EMPTY}
L_{EQ-TM}

Build M'_w such that

$L(M'_w)$ is regular iff M accepts w



M'_w: Accept a non regular language (say $0^n 1^n$) if M does not accept w;

Accept a regular language if M accepts w.

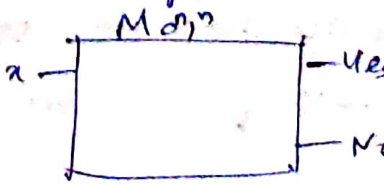
OR

if M does not accept w, then M'_w accepts empty string, 01, 0011, ~~0001~~ 000111 etc.

If M accepts w, M'_w accepts regular languages (eg: all strings)

Step 1:

- Accept all strings that are of form $0^n 1^n$



Step 2:

if $x \neq 0^n1^n$; then run w on M

Time complexity of a Turing machine

The no of ~~of~~ moves TM makes in terms of input size in the worst case before the TM halts (with an accept or reject)

→ TIME(f(n)): set of languages for which there is one deterministic single tape TM that has time complexity O(f(n)) and decides L

$0^n1^n \in TIME(n^2)$

$\in TIME(n \log n)$

One move on multitape TM

= 2 scans on the single tape + changing symbols if needed

$O(l+l) = O(l)$ moves.

The moves may involve extending a tape by printing all remaining cells right by one.

→ If k tape TM runs in f(n) moves, the worst case length of each of k tapes = f(n). So total length of k tapes = k·f(n).

one move = O f(n) moves.

f(n) moves = $O(f(n)^2)$ moves.

Non deterministic and deterministic TM

Assume that we take f(n) moves on NTM, number of moves on DTM = $O(b) + O(b^2) + \dots O(b^{f(n)})$ moves

max no of choices = b

$O(b) + O(b^2) + \cdots O(b^{f(n)}) = c(b^{f(n)})$

$= O(b^{f(n)})$ moves

$= O(2^{f(n)}) = 2^{O(f(n))}$ moves.

This does not mean that a language in TIME(f(n)) in NTM always takes $O(2^{f(n)})$ moves on a deterministic TM.

<u>Class P:</u>

$P = \bigcup_{i \geq 1} \text{TIME}(n^i)$

All problems or languages for which there is a TM that runs in polynomial time.

$0^n 1^n \in P$

Q: Given two numbers n_1 and n_2, check if they are relatively prime $\in P$ ~~O(log n)~~ $O(\log(\max(n_1, n_2)))$

Q: check if a number is a prime $\in P$

<u>Class NP:</u>

(I) NTIME(f(n)) : set of languages for which there is one nondeterministic TM that runs in O f(n) time.

$NP = \bigcup_{i \geq 1} \text{NTIME}(n^i)$

Eg: Travelling salesman problem $\in NP$

$P \subseteq NP$ (Non-deterministic polynomial)

Any deterministic TM can be seen as a non-deterministic TM with exactly one choice. So $P \subseteq NP$.

Q. Given a graph, check if it is connected.

<G> encoded (as a matrix, adjacency list)

NP class:

(I) A language L is in NP if w∈L can be verified with help of a certificate (also size ~~is~~ is polynomial in size of w), with a polynomial time algorithm.

~~e.g.~~: L = {G that are properly 3 colourable}.

There is no polynomial time algorithm to check if a given G ∈ L

If a colouring is given, check all the edges to see if their 2 end points have different colours. O(|E|) algorithm.

∴ L ∈ NP.

Q: Given a graph G and a number k, is there a clique of size k in G

certificate: The ~~clique~~ clique vertices.

Verify: All these vertices are connected to each other.

* L ∈ NP or L ∈ co-NP if a no answer can be verified in polynomial time.

$$\boxed{L \in P \Rightarrow L \in \text{co-NP and } L \in NP}$$

(I) ⇒ (II)

T.P ~~(II) ⇒ (I)~~

Use the path of Non deterministic TM as our certificate (a sequence of choice of each moves)

T.P (II) ⇒ (II) (I)

"guess" the certificate c (non deterministically), then run verifier algorithm, it runs in polynomial time

Q: Find the least weight hamiltonian cycle in the graph (cycle going through all ~~of~~ vertices)

Optimization problem.

Decision problem. [YES/NO problem]

Given such a graph G and a distance d, is there a ham. cycle with distance $\leq d$?

If A is not in P then B is also not in P

⇒ If B is in P then A is in P

assume B is in P (assume there is an algorithm)